Project work

NTNU
Norwegian University of
Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

Aksel Rogne Solbu,
Victor Gusland Ihlebæk

Density and Viscosity of Hydrogen and Natural Gas Mixtures: Comparing Models with Experimental Data

Trondheim, December 2024

NTNU – Trondheim
Norwegian University of
Science and Technology

Abstract

The global energy transition requires innovative and sustainable solutions to reduce carbon emissions, while maintaining reliable energy supplies. Norway has a unique opportunity to integrate low-carbon hydrogen into its natural gas value chain to reduce emissions. For this solution to be effective, models used for calculating properties such as density and viscosity for mixtures of natural gas and hydrogen must have high accuracy. However, hydrogen has proven difficult to model due to its distinct properties.

This study researches well established thermodynamic models, such as Soave-Redlich-Kwong (SRK), Peng Robinson (PR) and GERG2008 for density, as well as Lohrenz-Bray-Clark (LBC), Pedersen-Fredenslund-Christensen-Thomassen (PFCT) and Friction Theory (FT) for viscosity. The models are systematically compared with experimental data for temperatures and pressures relevant to the natural gas value chain. Special attention is given to investigating NeqSim and HYSYS' implementations of the SRK and PR equation of states, with the goal of further enhancing the NeqSim software.

The results for the density models were consistent across experimental data collected from various sources, demonstrating that GERG2008 was able to model the density within 0.1% of the experimental data. Overall, the GERG2008 equation of state undoubtedly outperforms both the SRK and the PR equation of state. The PFCT viscosity model proved most accurate for higher mole fractions of hydrogen, and the LBC viscosity model was most accurate for pure methane. This study provides a foundation to further improve the understanding of thermodynamic models related to hydrogen-rich gas mixtures, and offers suggestions on how to improve the NeqSim software.

Sammendrag

Den globale energiovergangen krever innovative og bærekraftige løsninger for å redusere karbonutslipp, samtidig som pålitelige energiforsyninger opprettholdes. Norge har en unik mulighet til å integrere lavkarbonhydrogen i sin naturgassverdikjede for å redusere utslipp. For at denne løsningen skal være effektiv, må modeller som brukes til å beregne egenskaper som tetthet og viskositet for blandinger av naturgass og hydrogen ha høy nøyaktighet. Hydrogen har imidlertid vist seg vanskelig å modellere på grunn av sine unike egenskaper.

Denne studien undersøker etablerte termodynamiske modeller, som Soave-Redlich-Kwong (SRK), Peng Robinson (PR) og GERG2008 for tetthet, samt Lohrenz-Bray-Clark (LBC), Pedersen-Fredenslund-Christensen-Thomassen (PFCT) og Friction Theory (FT) for viskositet. Modellene sammenlignes systematisk med eksperimentelle data for temperaturer og trykk som er relevante for naturgassverdikjeden. Spesiell oppmerksomhet rettes mot å undersøke implementeringene av SRK- og PR-tilstandsligningene i NeqSim og HYSYS, med mål om å videreutvikle NeqSim-programvaren.

Resultatene for tetthetsmodellene var konsistente på tvers av eksperimentelle data samlet fra ulike kilder, og viste at GERG2008 var i stand til å modellere tettheten med et avvik på $0,1\%$ fra de eksperimentelle dataene. Totalt sett presterer GERG2008 utvilsomt bedre enn både SRK- og PR-tilstandsligningene. PFCT-viskositetsmodellen viste seg å være mest nøyaktig for høyere molfraksjoner av hydrogen, mens LBC-viskositetsmodellen var mest nøyaktig for ren metan. Denne studien gir et grunnlag for å forbedre forståelsen av termodynamiske modeller relatert til hydrogenrike gassblandinger, og gir forslag til hvordan NeqSim-programvaren kan forbedres.

Contents

List of Figures vi

List of Tables viii

Acronyms x

Nomenclature xi

1 Introduction 1

1.1 Background and Motivation . . . 1

1.2 Objectives, Approach and Scope . . . 1

2 Theory 3

2.1 Thermodynamic Models . . . 3

2.2 Cubic Equations of State . . . 3

2.2.1 Soave-Redlich-Kwong EOS . . . 4

2.2.2 Peng Robinson EOS . . . 6

2.3 Fundamental EOSs . . . 7

2.3.1 Leachman EOS . . . 7

2.3.2 GERG2008 EOS . . . 8

2.4 Viscosity Models . . . 10

2.4.1 LBC . . . 10

2.4.2 PFCT . . . 11

2.4.3 Friction Theory . . . 13

2.5 Statistics . . . 16

2.5.1 Uncertainty Analysis . . . 16

2.5.2 Relative Deviation . . . 17

3 Method 18

3.1 Density . . . 18

3.1.1 Experimental Data . . . 18

3.1.2 Simulation Data . . . 21

3.1.3 NeqSim . . . 21

3.1.4 HYSYS . . . 21

3.2 Viscosity . . . 21

3.2.1 Experimental Data . . . 21

3.2.2 Simulation Data . . . 22

4 Results **23**

4.1 Density . . . 23

4.1.1 Pure Hydrogen . . . 23

4.1.2 Model Comparison with Experimental Data from Hernández-Gómez et al. [16] 24

4.1.3 Model Comparison with Experimental Data from Richter et al. . . . 26

4.1.4 Model Comparison with Experimental Data from Ben Souissi et al. . . . 28

4.1.5 Model Comparison with Experimental Data from Hernández-Gómez et al.[15] 29

4.1.6 Binary Interaction Coefficients . . . 30

4.2 Viscosity . . . 31

4.2.1 Pure Methane . . . 31

4.2.2 Pure Hydrogen . . . 31

4.2.3 Mixture: $0.9CH_4 + 0.1H_2$ . . . 32

4.2.4 Mixture: $0.8CH_4 + 0.2H_2$ . . . 33

4.2.5 Mixture: $0.5CH_4 + 0.5H_2$ . . . 34

4.2.6 Mixture: $0.1CH_4 + 0.9H_2$ . . . 35

5 Discussion **37**

5.1 Density . . . 37

5.1.1 Pure Hydrogen . . . 37

5.1.2 Comparison with Hernández-Gómez et al.[16] . . . 37

5.1.3 Comparison with Richter et al. . . . 37

5.1.4 Comparison with Ben Souissi et al. . . . 37

5.1.5 Comparison with Hernández-Gómez et al.[15] . . . 38

5.1.6 Binary coefficients NeqSim vs HYSYS . . . 38

5.2 Viscosity . . . 38

6 Conclusion **39**

6.1 Further Work . . . 40

Bibliography **41**

Appendix A - Original Project Description **45**

Appendix B - Python Code for Density **46**

B.1 Code for Pure Hydrogen . . . 46

B.2 Code for the study by Hernández-Gómez et al.[16] . . . 57

B.3 Code for the study by Richter et al. 63
B.4 Code for the study by Ben Souissi et al. 70
B.5 Code for the study by Hernández-Gómez et al.[15] 76

Appendix C - Binary Interaction Coefficients **82**

C.1 Binary Interaction Coefficients: NeqSim, SRK 82
C.2 Binary Interaction Coefficients: HYSYS, SRK 82
C.3 Binary Interaction Coefficients: NeqSim, PR 83
C.4 Binary Interaction Coefficients: HYSYS, PR 83

Appendix D - Python Code for Viscosity **84**

D.1 Experimental Data 84
D.2 Viscosity Calculation 88
D.3 Plotting 90

List of Figures

1 Plots of isotherms, showing the relative difference comparing the different density models to the Leachman model for normal hydrogen. 23

2 Plots of isotherms, showing the relative difference between the models and experimental data, for the mixture of (0.95 CH_4 + 0.05 H_2), from Hernández-Gómez et al.[16]'s study. 24

3 Plots of isotherms, showing the relative difference between the models and experimental data, for the mixture of (0.9 CH_4 + 0.1 H_2), from Hernández-Gómez et al.[16]'s study. 25

4 Plots of isotherms, showing the relative difference between the models and experimental data, for the mixture of (0.5 CH_4 + 0.5 H_2), from Hernández-Gómez et al.[16]'s study. 25

5 Plots of isotherms, showing the relative difference between the models and experimental data from Richter et al. for NG1, from table 8, with a hydrogen mole fraction of approximately 5% 27

6 Plots of isotherms, showing the relative difference between the models and experimental data from Richter et al. for NG2, from table 8, with a hydrogen mole fraction of approximately 10% 27

7 A plot for T = 283.15 K, showing the relative difference between the models and experimental data from Richter et al. for NG3, from table 8, with a hydrogen mole fraction of approximately 30% 27

8 Plots of isotherms, showing the relative difference between the models and experimental data from Ben Souissi et al., for the mixture of (0.94638 CO_2 + 0.05362 H_2) 29

9 Plots of isotherms, showing the relative difference between the models and experimental data from Hernández-Gómez et al.[15], for NG mixture in Table 9,with a hydrogen mole fraction of approximately 3%. 30

10 Percentage viscosity deviations of experimental viscosity data for pure methane of Owuna et al. and Chuang et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model. 31

11 Percentage viscosity deviations of experimental viscosity data for pure hydrogen of Owuna et al., Betken et al. and Chuang et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model. 32

12 Percentage viscosity deviations of experimental viscosity data for composition with mole fraction 0.9 CH_4 and 0.1 H_2 of Owuna et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model. 33

13 Percentage viscosity deviations of experimental viscosity data for composition with mole fraction 0.8 CH_4 and 0.2 H_2 of Owuna et al. and Chuang et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model. 34

14 Percentage viscosity deviations of experimental viscosity data for composition with mole fraction 0.5 CH_4 and 0.5 H_2 of Owuna et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model. 35

15 Percentage viscosity deviations of experimental viscosity data for composition with mole fraction 0.1 CH_4 and 0.9 H_2 of Betken et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model. 36

List of Tables

1 Components and Properties from GERG-2008 EOS[19]. 8

2 Uncertainty of the GERG2008 model for different temperature and pressure ranges.[19] 9

3 Constants for the PFCT Eqs. (51)-(54). 13

4 Parameters for the friction theory Eqs. (73) for SRK and PR EOSs for ψ in cm^3/mol. 16

5 Value of the coverage factor, k, and corresponding confidence interval, assuming a normal distribution. 17

6 Summary of studies for density of hydrogen mixtures. Showing the mixture, molar fraction of hydrogen, temperature and pressure range in K an MPa, respectively, and number of data points. NG compositions for different studies are presented in tables 8 and 9. 18

7 The relative expanded combined density uncertainty in the measurements from different studies. The superscripts [1], [2], and [3] correspond to different mole fractions of the $CH_4 + H_2$ mixture, specifically $(0.95 + 0.05)$, $(0.9 + 0.1)$, and $(0.5 + 0.5)$, respectively. 19

8 Composition of Natural Gas Samples from (Richter et al., 2014). Where the hydrogen molar fraction for NG1, NG2 and NG3 is 5.37%, 10.41% and 30.47%, respectively. 20

9 Composition of Gas Sample from (Hernández-Gómez et al., 2018)[15]. With a hydrogen molar fraction of approximately 3.01%. 20

10 Summary of studies for experimental viscosity data of methane-hydrogen mixtures collected for this study. Showing the mixture, molar fraction of hydrogen, temperature and pressure range in K an MPa, respectively, and number of data points. . 22

11 The relative expanded combined data uncertainty in the studies where experimental data has been collected. 22

12 Mean and maximum relative deviation of the density models from the Leachman model for the temperature and pressure range of $240 - 400$ K and $0 - 35$ MPa. . . 23

13 Mean and maximum ARD from the experimental density from Hernández-Gómez et al.[16] for the temperature and pressure range of $240 - 350$ K and $0 - 20$ MPa, for the different compositions. 26

14 Average and maximum absolute relative deviation from the experimental density from Richter et al., for the temperature and pressure range of $273.15 - 293.15$ K and $0 - 8$ MPa, for the different compositions of NG, presented in Table 8 28

15 Mean and maximum relative deviation from the experimental density from Ben Souissi et al. for the temperature and pressure range of $273.15 - 323.15$ K and $0 - 5$ MPa, for a composition of (0.94638 CO_2 + 0.05362 H_2). 29

16 Mean and maximum relative deviation from the experimental density from Hernández-Gómez et al. for the temperature and pressure range of $260 - 350$ K and $0 - 20$ MPa, for the NG-compostion presented in table 9 ($H_2 = 3\%$). 30

17 Mean and maximum percentage viscosity deviations of the experimental viscosity data for pure methane from the same modelled viscosity values used in Figure 10. 31

18 Mean and maximum percentage viscosity deviations of the experimental viscosity data for pure hydrogen from the same modeled viscosity values used in Figure 11. 32

19 Mean and maximum percentage viscosity deviations of the experimental viscosity data for composition with mole fraction 0.9 methane and 0.1 hydrogen from the same modelled viscosity values used in Figure 12. 33

20 Mean and maximum percentage viscosity deviations of the experimental viscosity data for composition with mole fraction 0.8 CH_4 and 0.2 H_2 from the same modeled viscosity values used in Figure 13. 34

21 Mean and maximum percentage viscosity deviations of the experimental viscosity data for composition with mole fraction 0.5 CH_4 and 0.5 H_2 from the same modelled viscosity values used in Figure 14. 35

22 Mean and maximum percentage viscosity deviations of the experimental viscosity data for composition with mole fraction 0.1 CH_4 and 0.9 H_2 from the same modelled viscosity values used in Figure 15. 36

Acronyms

AARD	Average Absolute Relative Difference
ARD	Absolute Relative Difference
BIC	Binary Interaction coefficient
CEOS	Cubic Equation of State
EOS	Equation of State
FT	Friction theory
GUM	The Guide to the Expression of Uncertainty
LBC	Lohrenz-Bay-Clark
NG	Natural gas
PR	Peng Robinson
RK	Redlich-Kwong
SRK	Soave-Redlich-Kwong

Nomenclature

Latin letters

a	Helmholtz free energy	J/mol
F_{ij}	Adjustable binary interaction factor	-
M	Molar mass	g/mol
p	Pressure	Pa
R	Universal gas constant	kJ/molK
T	Temperature	K
T_c	Critical temperature	K
T_r	Reduced temperature	-
U	Expanded uncertainty	-
v	Molar volume	m^3/mol
x	Molar composition	-
Z	Compressibility factor	-

Greek letters

α	Reduced helmholtz free energy	-
β	Adjustable binary interaction factor	-
γ	Adjustable binary interaction factor	-

Superscripts

0	Ideal-gas property	-
r	Residual	-

Subscripts

c	Critical-point property	-
i	Index	-
ij	Index	-
r	Reduced property	-

1 Introduction

1.1 Background and Motivation

Norway has established itself as a cornerstone in the European energy system, supplying approximately 30% of Europe's total gas consumption to provide energy for electricity, heating and industry [10]. However, as the world needs to adapt to support the green transition, fossil fuels' negative impact on the climate cannot be overlooked. To meet the Paris Agreement's goal of being carbon neutral by 2050, low-carbon technologies must be integrated into the gas value chain. Norway has the potential to significantly reduce carbon emissions in Europe by substituting a portion of its exported natural gas with hydrogen gas. The hydrogen itself can be produced either from green electrolysis or steam reforming with carbon capture and storage (CCS).

Hydrogen has long been viewed as the energy carrier of the future. It is regarded as a sustainable fuel due to its high energy content and its unique quality to be both produced and utilized without emitting CO_2. It was therefore believed that hydrogen would play a pivotal role in steering the energy sector towards greener and more environmentally friendly solutions [1]. Unfortunately, hydrogen has proven to be problematic for many reasons, one of which is the difficulty in accurately calculating the properties of hydrogen-containing streams using the thermodynamic models currently available. Since hydrogen is the smallest molecule in the universe, its quantum properties often need to be taken into consideration for accurate modelling, something conventional models often neglect [20].

Density and viscosity are crucial properties in natural gas value chain calculations, where accurate estimations are essential for understanding fluid flow behaviour, storage requirements, operational efficiency, economic viability and safety considerations. This paper examines the accuracy of conventional density and viscosity models for gas mixtures relevant to the natural gas value chain, with varying molar fractions of hydrogen. The thermodynamic models analysed include SRK, PR and GERG2008 for density, and LBC, PFCT and friction theory for viscosity.

One of the key motivations for this work is to further develop the NeqSim library. NeqSim is a free-to-use software designed for calculating fluid behaviour, phase equilibrium, and process simulations. Developed for the oil and gas industry, it prioritizes accurate modelling of thermo- and fluid dynamics, utilizing advanced equations of state to model properties for various compositions. By investigating the performance of thermodynamic models in NeqSim, this study aims to provide new insights that can further develop and expand NeqSim's capabilities for hydrogen-rich mixtures.

1.2 Objectives, Approach and Scope

The objective of this work is to evaluate the performance of common thermodynamic models for density and viscosity for gas mixtures with varying molar fractions of hydrogen. The mixtures assessed are gas mixtures at temperatures and pressures relevant for natural gas export. Model performance is evaluated using publicly available experimental data, with considerable effort devoted to ensuring the reliability of the data. The requirements for the experimental data are:

- Temperature and pressure range that were relavant for the natural gas value chain
- Components that are relevant for the natural gas industry
  - Binary mixtures of methane and hydrogen
  - Binary mixtures of CO_2 and hydrogen
  - More conventional mixtures, containing mainly methane and hydrogen, but also ethane, propane, etc.
- Low uncertainty, assuring the data is reliable.

To evaluate NeqSim's performance, it is useful to compare it with another well-established software tool designed for thermodynamic calculations. Comparing NeqSim to HYSYS, a widely acknowledged software in the industry with long-standing use, provides valuable insight into how NeqSim performs relative to more established tools. This comparison gives meaningful insight and can help develop NeqSim further, as coefficients for SRK and PR differ between NeqSim and HYSYS.

2 Theory

2.1 Thermodynamic Models

Thermodynamic models are the cornerstone for simulation and calculation of process and chemical engineering. These models are used to calculate thermodynamic properties, energy balances, mass balances, and thermodynamic equilibrium. In petroleum and chemical engineering, the most widely used category of thermodynamic models is the equation of state (EOS). This is due to its broad applicability across a large pressure range, extending from low to very high pressures[45].

An equation of state is a thermodynamic equation describing the "state of matter under a given set of physical conditions" (Sun and Smith, 2010). It is in other words an equation that describes the relationship between two or more state variables of a given matter, such as reservoir fluids. A state variable is a quantity that describes the state of a matter under a given set of physical conditions, such as temperature, pressure, and density [44].

There is to this date no EOS that gives accurate property results for all substances and all conditions. Therefore, there have been developed many different EOSs for both liquids and gases over the years. While some EOSs seem to predict properties better than others by and large, they often have different areas of use. For example in process engineering, cubic EOSs like the Peng Robinson EOS and the Soave-Redlich-Kwong EOS are widely used, whereas fundamental EOSs are often used for developing environmentally friendly refrigerants [21].

In thermodynamic modelling, *mixing rules* are relations concerning the mixture composition for a multicomponent fluid mixture, such as petroleum export gas. Most EOSs were initially developed for pure components, with parameters usually based on pure component properties. While accurate pure component data is essential for thermodynamic modelling, they alone do not ensure correct mixture properties [9]. In practice, most engineering purposes involve multicomponent mixtures. The selection of a fitting mixing rule is therefore an important part of thermodynamic modelling, in addition to using an appropriate EOS.

The density of a fluid is an important quantity in thermodynamic modelling. Different EOSs can predict different densities for the same fluid at the same conditions. Since all EOSs have their limitations, it is crucial to have an understanding of how various EOSs are constructed to be able to develop improved models for fluid- and thermodynamic property prediction. This chapter therefore introduces different EOSs relevant for natural gas export to facilitate comparison of modelled densities and experimental data. When using cubic equations of state for density calculations, the following relation between density, ρ, and molar volume, v, applies:

$$\rho = \frac{M}{v} \tag{1}$$

where M is the molar mass.

2.2 Cubic Equations of State

Cubic equations of state are popular for engineering applications because of their high accuracy compared to their simple structure and low computational expense[11]. The origin of the cubic equations of state (CEOS) dates back to van der Waals EOS proposed in 1873 [47], which connected gaseous and liquid states by predicting critical phenomena and vapour-liquid equilibrium. Van der Waals proposed the following relation:

$$p = \frac{RT}{v - b} - \frac{a}{v^2} \tag{2}$$

Where

p = Pressure

T = Temperature
v = Molar volume
R = Universal gas constant
a = "Attraction" parameter
b = "Repulsion" parameter

The van der Waals EOS is technically a modification of the ideal gas equation:

$$p = \frac{RT}{v} \tag{3}$$

However, the ideal gas equation makes several assumptions:

- The gas molecules have insignificant volume compared to the total volume of gas.
- The gas molecules are equally sized, and have no intermolecular forces between them.
- The gas molecules collide elastically, i.e. with no loss of kinetic energy.
- The movement of the gas molecules agree with Newton's laws of motion.

To account for the two first assumptions listed above, van der Waals added two new empirical constants for individual gases, a and b. The "repulsion" parameter, b, is considered to represent the volume of molecules and is subtracted from the actual volume of gas. The "attraction" parameter, a, is found in the corrective term, a/v^2, to account for attractive forces between the molecules [2].

Considering this, the van der Waals EOS and all other CEOSs can be generalized as pressure terms:

$$p = p_{repulsion} - p_{attraction} \tag{4}$$

For the van der Waals EOS, $p_{repulsion}$ is represented by $RT/(v-b)$, and $p_{attraction}$ is represented by a/v^2. The van der Waals EOS has its limitations, which was even expressed by Johannes Diderik van der Waals himself in his Nobel-prize speech:

> *"In fact, bluntly speaking, the result would be an equation of state compatible with experimental data is totally impossible. No such equation is possible, unless something is added, namely that the molecules associate to form larger complexes"*
> *- Van der Waals, 1910*

Since then, hundreds of cubic EOSs have been proposed in attempts to produce more accurate results than the van der Waals EOS. However only the Soave-Redlich-Kwong EOS (1972) and the Peng Robinson EOS (1976) have shown to be good enough for practical use in engineering today [18].

2.2.1 Soave-Redlich-Kwong EOS

The Soave-Redlich-Kwong EOS is one of the most used EOSs for engineering applications. The EOS is a modification of the Redlich-Kwong cubic EOS (Redlich and Kwong, 1949):

$$p = \frac{RT}{v-b} - \frac{a}{v(v+b)\sqrt{T}} \tag{5}$$

Soave (1972) introduced a new dimensionless variable, $\alpha(T)$, to replace the explicit temperature term $1/\sqrt{T}$ from the Redlich-Kwong (RK) EOS. Hence, the Soave-Redlich-Kwong (SRK) EOS is expressed as:

$$p = \frac{RT}{v-b} - \frac{\alpha a}{v(v+b)} \tag{6}$$

The variables a_i and b_i, where the i denotes the i-th component in a mixture, are given by:

$$a_i = 0.42747 \frac{R^2 T_{c_i}^2}{p_{c_i}} \tag{7}$$

$$b_i = 0.08664 \frac{R T_{c_i}}{p_{c_i}} \tag{8}$$

The dimensionless factor, $\alpha_i(T)$, is equal to 1 at critical temperature, and can be found by the relation:

$$\alpha_i^{0.5}(T) = 1 + m_i(1 - T_{r_i}^{0.5}) \tag{9}$$

where the reduced temperature, T_r, is defined as:

$$T_r = \frac{T}{T_c} \tag{10}$$

The temperature correction parameter, m_i, is defined as:

$$m_i = 0.480 + 1.574\omega_i - 0.176\omega_i^2 \tag{11}$$

where ω_i is the acentric factor of the substance. The acentric factor is a parameter related to molecular size for non-polar compounds[41], and is a function of the reduced vapour pressure when the reduced temperature is 0.7. It can be calculated from the relation:

$$p_{r_i}^{sat}(T_r = 0.7) = 10^{-1-w_i} \tag{12}$$

Values for the acentric factor can also be found in tables for many substances.

Soave (1972) also expressed Eq. (6) in terms of the compressibility factor Z by replacing the molar volume v with $\frac{ZRT}{p}$ to obtain

$$Z^3 - Z^2 + Z(A - B - B^2) - AB = 0 \tag{13}$$

where

$$Z = \frac{pv}{RT} \tag{14}$$

$$A = \frac{ap}{R^2T^2} \tag{15}$$

and

$$B = \frac{bp}{RT} \tag{16}$$

For mixtures, a mixing rule is required to determine the a and b parameters for the SRK EOS. Soave, as well as Redlich and Kwong (1949), used the quadratic mixing rule in his paper. For a mixture with N number of components and a mole fraction x_i, the quadratic mixture is calculated as follows:

$$a = \sum_{i=1}^{N} \sum_{j=1}^{N} x_i x_j a_{ij} \tag{17}$$

$$b = \sum_{i=1}^{N} x_i b_i \tag{18}$$

where

$$a_{ij} = (1 - k_{ij})(a_i a_j)^{\frac{1}{2}} \tag{19}$$

k_{ij} is an empirical correction factor typically referred to as the *binary interaction coefficient* (BIC). The BIC, k_{ij}, is determined from experimental data, and is considered independent of pressure, temperature and composition [42]. For most components relevant for engineering practices, values for k_{ij} are tabulated.

2.2.2 Peng Robinson EOS

The Peng Robinson EOS is another common EOS for engineering applications, especially in petroleum and chemical engineering. Ding-Yu Peng and Donald B. Robinson (1976) proposed the Peng Robinson (PR) EOS of the form:

$$p = \frac{RT}{v - b} - \frac{\alpha a}{v(v + b) + b(v - b)} \tag{20}$$

where the constants a and b are defined as:

$$a = 0.45724 \frac{R^2 T_c^2}{p_c} \tag{21}$$

$$b = 0.07780 \frac{RT_c}{p_c} \tag{22}$$

At critical temperature, $\alpha(T_r, \omega) = 1$. For temperatures not at the critical point, Peng and Robinson defined the dimensionless function as:

$$\alpha^{1/2} = 1 + \kappa(1 - T_r^{1/2}) \tag{23}$$

where κ, a function of the acentric factor, is defined as:

$$\kappa = 0.37464 + 1.54226\omega - 0.26992\omega^2 \tag{24}$$

The PR EOS can also be rewritten in terms of the compressibility factor:

$$Z^3 - (1 - B)Z^2 + (A - 3B^2 - 2B)Z - (AB - B^2 - B^3) = 0 \tag{25}$$

where

$$A = \frac{ap}{R^2 T^2} \tag{26}$$

and

$$B = \frac{bp}{RT} \tag{27}$$

For mixtures, the same quadratic mixing rule as for the SRK EOS applies, i.e. Eqs. (17)-(19).

2.3 Fundamental EOSs

Fundamental EOSs, as their name suggests, are based on fundamental thermodynamic potentials. The four main fundamental thermodynamic potentials are internal energy, enthalpy, Gibbs free energy and Helmholtz free energy. Together with experimental data, fundamental EOSs implicitly calculate thermal properties of a gas using long multi-parameter formulas with parameters based on experimental data. Fundamental EOSs are more complicated than CEOSs and often provide highly accurate results, but demand higher computational costs in return [5].

The fundamental EOSs discussed in this work, Leachman and GERG2008, are based on Helmholtz free energy [43]. The Helmholtz free energy is a function of temperature and density, and is defined as:

$$\frac{a(T,\rho,\bar{x})}{RT} = \alpha(\tau,\delta,\bar{x}) = \alpha^0(\tau,\delta,\bar{x}) + \alpha^r(\tau,\delta,\bar{x}) \tag{28}$$

Here, a is the Helmholtz free energy and α is the dimensionless Helmholtz free energy. Other properties such as pressure, entropy, and enthalpy can be calculated from Helmholtz free energy derivatives [20].

In Eq. (28), α^0 and α^r represent the ideal gas and the residual contributions respectively, which are both functions of the inverse reduced temperature, τ, the reduced density, δ, and the molar composition of the mixture, $\bar{x}$. The variables τ and δ are defined as:

$$\tau = \frac{T_c(\bar{x})}{T} \tag{29a}$$

$$\delta = \frac{\rho}{\rho_c(\bar{x})} \tag{29b}$$

Where the critical properties, T_c and ρ_c, are determined by the composition.

Using the derivative of the residual part of the Helmholtz free energy (α^r) with respect to the reduced density at constant reduced temperature, the pressure can be calculated:

$$p = \rho RT\left[1 + \delta\left(\frac{d\alpha^r}{d\delta}\right)_\tau\right] \tag{30}$$

From Eq. (30), a relationship between pressure, density and temperature is obtained through α^r. The modelling of α^r itself is determined by empiric data for mixtures and is unique for each EOS.

2.3.1 Leachman EOS

The Leachman EOS is created solely for pure hydrogen, and accounts for quantum states of the hydrogen atoms to achieve high accuracy [20]. Based on the spin state of the atoms, there are two types of hydrogen: *Ortho-* and *parahydrogen*. At room temperature and above, hydrogen gas consists of 75% orthohydrogen and 25% parahydrogen. For lower temperatures, orthohydrogen converts to parahydrogen. The conversion process from ortho- to parahydrogen is exothermic, and can potentially slow down cooling or condensing rate, something that complicates energy calculations.

The Leachman EOS utilizes the Helmholtz free energy to calculate properties of hydrogen with high accuracy. The EOS is designed for temperatures from approximately 14K up to 1000K, and pressure up to 2000 MPa [20]. Because Leachman's model is specifically designed for hydrogen and is not compatible with other components, it is highly accurate for density calculations. The

model has an uncertainty of only 0.04% within a temperature range of 250 to $450K$ for pressures up to $300MPa$ [20].

The pressure can be calculated using equation (30), together with the residual reduced Helmholtz free energy for the Leachman model, α^r, and equations (29a) and (29b). For the Leachman EOS, α^r is defined as:

$$\alpha^r(\tau,\delta) = \sum_{i=1}^{l} N_i \delta^{d_i} \tau^{t_i} + \sum_{i=l+1}^{m} N_i \delta^{d_i} \tau^{t_i} \exp(-\delta^{p_i}) + \sum_{i=m+1}^{n} N_i \delta^{d_i} \tau^{t_i} \exp\left[-\varphi_i(\delta - D_i)^2 - \beta_i(\tau - \gamma_i)^2\right]. \quad (31)$$

The constants in equation (31) are fit to experimental data. More detailed derivations of Leachman's model can be found in (Leachman et al., 2009).

2.3.2 GERG2008 EOS

The GERG2008 model is an EOS made specifically for natural gas mixtures. The model is fit to 21 components (see Table 1) typically found in natural gas (NG) mixtures. The EOS is based on Helmholtz free energy, and can be used for large temperature and pressure ranges with low uncertainty. The validity ranges of the model are found in Table 2.

Component	**Formula**	ρ_c **(mol/dm^3)**	T_c **(K)**	**M (g/mol^1)**
Methane	CH_4	10.1393	190.564	16.04246
Nitrogen	N_2	11.1839	126.192	28.0134
Carbon Dioxide	CO_2	10.6248	304.128	44.0095
Ethane	C_2H_6	6.8709	305.322	30.06904
Propane	C_3H_8	5.0000	369.825	44.09562
n-Butane	n-C_4H_{10}	3.9200	425.125	58.1222
Isobutane	i-C_4H_{10}	3.8601	407.817	58.1222
n-Pentane	n-C_5H_{12}	2.3558	469.700	72.14878
Isopentane	i-C_5H_{12}	2.3229	460.350	72.14878
n-Hexane	n-C_6H_{14}	2.0191	507.600	86.17536
n-Heptane	n-C_7H_{16}	1.7635	540.150	100.20194
n-Octane	n-C_8H_{18}	1.5664	568.700	114.22852
n-Nonane	n-C_9H_{20}	1.4056	594.600	128.25510
n-Decane	n-$C_{10}H_{22}$	1.264	617.700	142.28168
Oxygen	O_2	13.6300	154.595	31.99880
Carbon Monoxide	CO	10.8500	132.860	28.01010
Water	H_2O	17.8743	647.096	18.01528
Hydrogen Sulfide	H_2S	13.2720	373.200	34.08088
Helium	He	17.3990	5.1953	4.002602
Argon	Ar	13.4074	150.687	39.94800

Table 1: Components and Properties from GERG-2008 EOS[19].

	Normal Range of Validity	Extended Range of Validity
Temperature [K]	90 – 450	60 – 700
Pressure [MPa]	≤ 35	≤ 70
Uncertainty [%]	0.1	0.2 – 0.5

Table 2: Uncertainty of the GERG2008 model for different temperature and pressure ranges.[19]

Because of its high accuracy, GERG2008 is widely used in the natural gas industry and is today an ISO Standard for natural gases. GERG2008 uses well established mixing rules for binary gas component pairs to predict the state of more complex gas mixtures. Similarly to the Leachman model, GERG2008 employs a large number of empirical parameters fitted to experimental data, enabling high accuracy for several states and mixtures. The EOS claims an uncertainty of only 0.1% within its normal range of validity [19]. However, GERG2008 is not fitted to mixtures with high molar fractions of CO_2, and as a result, the level of uncertainty cannot be guaranteed for such mixtures.

For GERG2008, α^r for a mixture is given by:

$$\alpha^r(\delta,\tau,\vec{x}) = \sum_{i=1}^{N} x_i \alpha^r_{\mathrm{oi}}(\delta,\tau) + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} x_i x_j F_{ij} \alpha^r_{ij}(\delta,\tau) \tag{32}$$

where N is the total number of components in the mixture, and the subscript i denotes the i-th component in the mixture. The first term in Eq. (32) is the reduced residual Helmholtz free energy, α^r_{oi}, weighted by its mole fraction x_i. The second term in the equation accounts for interactions between components i and j. The parameter α^r_{ij} is the reduced residual Helmholtz free energy of the binary mixture consisting of components i and j. F_{ij} represents the degree of interaction between the two components and how much the interaction contributes to the overall Helmholtz free energy. F_{ij} is tuned to fit experimental data. α^r_{oi} and α^r_{ij} are defined as:

$$\alpha^r_{oi}(\delta,\tau) = \sum_{k=1}^{K_{\mathrm{Pol},i}} n_{oi,k}\delta^{d_{oi,k}}\tau^{t_{oi,k}} + \sum_{k=K_{\mathrm{Pol},i}+1}^{K_{\mathrm{Pol},i}+K_{\mathrm{Exp},i}} n_{oi,k}\delta^{d_{oi,k}}\tau^{t_{oi,k}} e^{-\delta^{c_{oi,k}}} \tag{33}$$

$$\begin{aligned}\alpha^r_{ij}(\delta,\tau) = &\sum_{k=1}^{K_{\mathrm{Pol},ij}} n_{ij,k}\delta^{d_{ij,k}}\tau^{t_{ij,k}} + \sum_{k=K_{\mathrm{Pol},ij}+1}^{K_{\mathrm{Pol},ij}+K_{\mathrm{Exp},ij}} n_{ij,k}\delta^{d_{ij,k}}\tau^{t_{ij,k}} \\ &\exp\left[-\eta_{ij,k}(\delta-\epsilon_{ij,k})^2 - \beta_{ij,k}(\tau-\gamma_{ij,k})\right].\end{aligned} \tag{34}$$

The critical temperature, T_c, and the inverse critical density, $1/\rho_c$, for the mixture are given by:

$$T_c(\bar{x}) = \sum_{i=1}^{N} x_i^2 T_{c,i} + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} 2x_i x_j \beta_{T,ij}\gamma_{T,ij}\frac{x_i+x_j}{\beta^2_{T,ij}x_i + x_j}\left(T_{c,i}T_{c,j}\right)^{0.5} \tag{35}$$

$$\frac{1}{\rho_c(\bar{x})} = \sum_{i=1}^{N} x_i^2 \frac{1}{\rho_{c,i}} + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} 2x_i x_j \beta_{\nu,ij}\gamma_{\nu,ij}\frac{x_i+x_j}{\beta^2_{\nu,ij}x_i + x_j}\frac{1}{8}\left(\frac{1}{\rho_{c,i}^{1/3}} + \frac{1}{\rho_{c,j}^{1/3}}\right)^3 \tag{36}$$

A quadratic mixing rule is employed in Eqs. (35) and (36) to assure accurate results. The constants $\beta_{T,ij}$, $\beta_{v,ij}$, $\gamma_{T,ij}$ and $\gamma_{v,ij}$ are tuned to experimental data for binary mixtures.

The values of the constants in Equations (32)-(36) and a more in-depth derivation of the GERG-2008 model can be found in (Kunz and Wagner, 2012).

2.4 Viscosity Models

Viscosity is a measure of the "resistance of a fluid (liquid or gas) to a change in shape, or movement of neighbouring portions relative to one another", according to Encyclopedia Britannica [13]. In other words, the viscosity of a fluid is a measure of the fluid's resistance to flow. Viscosity is an important physical fluid property for transport calculations, and piping and nozzle design. It is therefore important to be able to predict the viscosity of a fluid as accurately as possible, over a wide range of temperatures, pressures and compositions.

Many viscosity models feature a *dilute gas* term, η_0. The properties of a dilute gas are completely and solely determined from the particle kinematics and collisions between the molecules. In other words, a gas can typically only be considered dilute at low pressure and temperature. Therefore, if the viscosity of a gas is dependent on density, the gas is not dilute [6]. For gases of relevance to petroleum and chemical engineering, it is not sufficient to consider the gas as dilute when calculating the viscosity. Therefore, many different viscosity models have been proposed to predict the viscosity of both pure components as well as mixtures at a variety of different pressures and temperatures [24]. The viscosity models which will be considered in this study are the LBC, PFCT, and friction theory viscosity models.

2.4.1 LBC

The Lohrenz-Bray-Clark (LBC) viscosity model was published in 1964 [22], as a viscosity model for reservoir fluids. The equation consists of a power series with empirical coefficients:

$$\left[(\eta - \eta_0)\xi_T + 10^{-4}\right]^{1/4} = a_0 + a_1\rho_r + a_2\rho_r^2 + a_3\rho_r^3 + a_4\rho_r^4 \tag{37}$$

where the coefficients a_{0-4} are equal to [0.10230, 0.023364, 0.058533, -0.040758, 0.0093324] respectively. The reduced density, ρ_r is defined as:

$$\rho_r = \frac{\rho}{\sum_{i=1}^{n} x_i \rho_{c,i}} = \frac{\rho \sum_{i=1}^{n} x_i v_{c,i}}{M} \tag{38}$$

The mixture viscosity parameter is given by:

$$\xi_T = \frac{\left[\sum_{i=1}^{n} x_i T_{c,i}\right]^{1/6}}{\left[\sum_{i=1}^{n} x_i M_i\right]^{1/2} \left[\sum_{i=1}^{n} x_i p_{c,i}\right]^{2/3}} \tag{39}$$

The dilute gas viscosity, η_0 is here defined as:

$$\eta_0 = \frac{\sum_{i=1}^{n} x_i \eta_{0,i}^* \sqrt{M_i}}{\sum_{i=1}^{n} x_i \sqrt{M_i}} \tag{40}$$

where

$$\eta_{0,i}^* = \begin{cases} \frac{1}{\xi_T}(34 \times 10^{-5}) T_{r,i}^{0.94} & \text{if } T_{r,i} \leq 1.5 \\ \frac{1}{\xi_T}(17.78 \times 10^{-5})(4.58 T_{r,i} - 1.67)^{5/8} & \text{if } T_{r,i} > 1.5 \end{cases} \tag{41}$$

2.4.2 PFCT

The Pedersen, Fredenslund, Christensen, and Thomassen (PFCT) viscosity model [26] [25] is a modification by Ely and Hanley (1981) to fit the model specifically for petroleum fluids. The PFCT model calculates viscosity η as a function of pressure p and temperature T. The mixture viscosity is defined as:

$$\eta(p,T) = \eta_{ref}\left(\frac{T_{c,mix}}{T_{c,0}}\right)^{-1/6}\left(\frac{p_{c,mix}}{p_{c,0}}\right)^{2/3}\left(\frac{M_{mix}}{M_0}\right)^{1/2}\frac{\alpha_{mix}}{\alpha_0} \tag{42}$$

where $T_{c,0}$, $p_{c,0}$, and M_0 are the critical temperature, critical pressure and molar mass of the mixture respectively. The other variables are calculated as follows:

$$T_{c,mix} = \frac{\sum_i \sum_j x_i x_j \left[\left(\frac{T_{c,i}}{p_{c,i}}\right)^{1/3} + \left(\frac{T_{c,j}}{p_{c,j}}\right)^{1/3}\right]^3 [T_{c,i}T_{c,j}]^{1/2}}{\sum_i \sum_j x_i x_j \left[\left(\frac{T_{c,i}}{p_{c,i}}\right)^{1/3} + \left(\frac{T_{c,j}}{p_{c,j}}\right)^{1/3}\right]^3} \tag{43}$$

$$p_{c,mix} = \frac{8\sum_i \sum_j x_i x_j \left[\left(\frac{T_{c,i}}{p_{c,i}}\right)^{1/3} + \left(\frac{T_{c,j}}{p_{c,j}}\right)^{1/3}\right]^3 [T_{c,i}T_{c,j}]^{1/2}}{\left(\sum_i \sum_j x_i x_j \left[\left(\frac{T_{c,i}}{p_{c,i}}\right)^{1/3} + \left(\frac{T_{c,j}}{p_{c,j}}\right)^{1/3}\right]^3\right)^2} \tag{44}$$

$$M_{mix} = \overline{M}_n + 1.304 \times 10^{-4}(\overline{M}_w^{2.303} - \overline{M}_n^{2.303}) \tag{45}$$

$$\alpha_{mix} = 1 + 7.378 \times 10^{-3}\rho_r^{1.847} M_{mix}^{0.5173} \tag{46}$$

$$\alpha_0 = 1 + 8.374 \times 10^{-4}\rho_r^{4.265} \tag{47}$$

where $\overline{M}_n$ and $\overline{M}_w$ are the number average and the weight average molecular weights respectively:

$$\overline{M}_n = \sum_i x_i M_i \tag{48a}$$

$$\overline{M}_w = \frac{\sum_i x_i M_i^2}{\sum_i x_i M_i} \tag{48b}$$

The reduced density, ρ_r, is defined as:

$$\rho_r = \frac{\rho_0}{\rho_{c,0}} \tag{49}$$

where ρ_{c_0} is the critical density of the reference fluid, methane.

Pedersen and Fredenslund (1987) modified Hanley et al.'s model (1975) for the reference viscosity, using methane as the reference fluid:

$$\eta_{ref} = \eta_k(T) + \eta_1(T)\rho + F_1\Delta\eta'(\rho,T) + F_2\Delta\eta''(\rho,T) \tag{50}$$

where

$$\eta_k(T) = \sum_{i=1}^{9} GV_i T^{\frac{i-4}{3}} \tag{51}$$

$$\eta_1(T) = A + B\left(C - \ln\left(\frac{T}{F}\right)\right)^2 \tag{52}$$

$$\Delta\eta' = exp\left(j_1 + \frac{j_4}{T}\right)\left[exp\left[\rho^{0.1}\left(j_2 + \frac{j_3}{T^{3/2}}\right) + \theta\rho^{0.5}\left(j_5 + \frac{j_6}{T} + \frac{j_7}{T^2}\right)\right] - 1\right] \tag{53}$$

$$\Delta\eta'' = exp\left(k_1 + \frac{k_4}{T}\right)\left[exp\left[\rho^{0.1}\left(k_2 + \frac{k_3}{T^{3/2}}\right) + \theta\rho^{0.5}\left(k_5 + \frac{k_6}{T} + \frac{k_7}{T^2}\right)\right] - 1\right] \tag{54}$$

The constants F_1, F_2, and θ are given by:

$$F_1 = \frac{HTAN + 1}{2} \tag{55}$$

$$F_2 = \frac{1 - HTAN}{2} \tag{56}$$

$$\theta = \frac{\rho - \rho_c}{\rho_c} \tag{57}$$

with

$$HTAN = \frac{exp(\Delta T) - exp(-\Delta T)}{exp(\Delta T) + exp(-\Delta T)} \tag{58}$$

$$\Delta T = T - T_f \tag{59}$$

where T_f is the freezing point of methane.

The rest of the constants from Eqs. (51)-(54) are given in Table 3.

Constant	
GV_1	$-2.090\,975\times10^{5}$
GV_2	$2.647\,269\times10^{5}$
GV_3	$-1.472\,818\times10^{5}$
GV_4	$4.716\,740\times10^{4}$
GV_5	$-9.491\,872\times10^{3}$
GV_6	$1.219\,979\times10^{3}$
GV_7	$-9.627\,993\times10^{1}$
GV_8	$4.274\,152$
GV_9	$-8.141\,531\times10^{-2}$
A	$1.696\,985\,927$
B	$-0.133\,372\,346$
C	1.4
F	168.0
j_1	$-1.035\,060\,586\times10^{1}$
j_2	$1.757\,159\,967\,1\times10^{1}$
j_3	$-3.019\,391\,865\,6\times10^{3}$
j_4	$1.887\,301\,159\,4\times10^{2}$
j_5	$4.290\,360\,948\,8\times10^{-2}$
j_6	$1.452\,902\,344\,4\times10^{2}$
j_7	$6.127\,681\,870\,6\times10^{3}$
k_1	$-9.746\,02$
k_2	$1.808\,34\times10^{1}$
k_3	$-4.126\,66\times10^{3}$
k_4	$4.460\,55\times10^{1}$
k_5	$9.765\,44\times10^{-1}$
k_6	$8.181\,34\times10^{1}$
k_7	$1.564\,99\times10^{4}$

Table 3: Constants for the PFCT Eqs. (51)-(54).

2.4.3 Friction Theory

The friction theory viscosity model was published by Quiñones-Cisneros et al. in (2000) as a versatile viscosity model made for all industrial process simulations based on any of the popular CEOSs. The main difference between the friction theory viscosity model proposed by Quiñones-Cisneros et al. (2000) and the other viscosity models observed in this study, is that friction theory (FT) treats viscosity as a part mechanical property, in contrast to a pure transport property.

FT separates the total viscosity into two viscosity contributions: A dilute gas term, and a friction term. Hence, the total viscosity has the form:

$$\eta = \eta_0 + \eta_f \tag{60}$$

where η_0 denotes the dilute gas viscosity and η_f denotes the friction contribution to the viscosity.

The semi-empirical predictive model by Chung et al. [8] based on the modified Chapman-Enskog is used to calculate the dilute gas term, η_0 in units of micropascal second (μPa):

$$\eta_0 = 4.0785 \frac{\sqrt{M * T}}{v_c^{2/3} \Omega^*} F_c \tag{61}$$

where Ω^* is an empirical relation defined as:

$$\begin{aligned} \Omega^* = & \frac{1.16145}{T^{*0.14874}} + \frac{0.52487}{exp(0.77320T^*)} + \frac{2.16178}{exp(2.43787T^*)} \\ & - 6.435 * 10^{-4} T^{*^{0.14874}} sin(18.0323T^{*-0.76830} - 7.27371) \end{aligned} \tag{62}$$

where

$$T^* = \frac{1.2593T}{T_c} \tag{63}$$

The F_c factor is defined as:

$$F_c = 1 - 0.2756\omega + 0.059035\mu_r^4 + \kappa \tag{64}$$

where ω is the acentric factor and κ is a correction factor accounting for the hydrogen bonding effects in associating substances. The factor μ_r represents a dimensionless dipole moment. However, for non-polar gases, such as those considered in this study, the last two terms in Eq. (64) become zero, resulting in:

$$F_c = 1 - 0.2756\omega \tag{65}$$

For mixtures, the mixture viscosity is given by:

$$\eta_{mx} = \eta_{0,mx} + \eta_{f,mx} \tag{66}$$

where the dilute gas viscosity for the mixture is calculated by:

$$\eta_{0,mx} = \exp\left[\sum_{i=1}^{n} x_i \ln\left(\eta_{0,i}\right)\right] \tag{67}$$

The subscript i refers to the i-th component of n total components in the mixture. Quiñones-Cisneros et al. (2000) calculates the friction contribution term in Eq. (66) as a quadratic friction model:

$$\eta_{f,mx} = \kappa_{r,mx} p_{r,mx} + \kappa_{a,mx} p_{a,mx} + \kappa_{rr,mx} p_{r,mx}^2 \tag{68}$$

where $p_{a,mx}$ and $p_{r,mx}$ represent the attractive and repulsive pressure contributions of the mixture, and $\kappa_{a,mx}$, $\kappa_{r,mx}$ and $\kappa_{rr,mx}$ represent the corresponding viscous friction parameters for the mixture. The viscous friction coefficients are given by the following mixing rules:

$$\kappa_{r,mx} = \sum_{i=1}^{n} z_i \kappa_{r,i} \tag{69a}$$

$$\kappa_{a,mx} = \sum_{i=1}^{n} z_i \kappa_{a,i} \tag{69b}$$

$$\kappa_{rr,mx} = \sum_{i=1}^{n} z_i \kappa_{rr,i} \tag{69c}$$

where the weighted fraction factor, z_i, to account for differences in molecular mass, is given on the form:

$$z_i = \frac{x_i}{M_i^{\varepsilon} * MM} \tag{70}$$

where

$$MM = \sum_{i=1}^{n} \frac{x_i}{M_i^{\varepsilon}} \tag{71}$$

Quiñones-Cisneros et al. (2000) suggests that the value for ε should be slightly larger than 0 to enhance the accuracy of the results. Quiñones-Cisneros et al. (2001) found that $\varepsilon = 0.3$ generally gave the best results.

The pure component friction coefficients are calculated as described by Sergio E. Quiñones-Cisneros and Stenby (2004):

$$\kappa_{r,i} = \frac{\eta_{c,i} \hat{\kappa}_{r,i}}{p_{c,i}} \tag{72a}$$

$$\kappa_{a,i} = \frac{\eta_{c,i} \hat{\kappa}_{a,i}}{p_{c,i}} \tag{72b}$$

$$\kappa_{rr,i} = \frac{\eta_{c,i} \hat{\kappa}_{rr,i}}{p_{c,i}} \tag{72c}$$

where $\hat{\kappa}_{r,i}$, $\hat{\kappa}_{a,i}$, and $\hat{\kappa}_{rr,i}$ are reduced friction coefficients that only depend on the reduced temperature. The reduced friction coefficients are calculated as described by Quiñones-Cisneros et al. (2001):

$$\begin{aligned}\hat{\kappa}_{r,i} = \hat{\kappa_r^c} &+ \kappa_{r,0,0}(\Gamma - 1) + (\kappa_{r,1,0} + \kappa_{r,1,1}\psi) * (exp(\Gamma - 1) - 1) \\ &+ (\kappa_{r,2,0} + \kappa_{r,2,1}\psi + \kappa_{r,2,1}\psi^2)(exp(2\Gamma - 2) - 1)\end{aligned} \tag{73a}$$

$$\begin{aligned}\hat{\kappa}_{a,i} = \hat{\kappa}_a^c &+ \kappa_{a,0,0}(\Gamma - 1) + (\kappa_{a,1,0} + \kappa_{a,1,1}\psi) * (exp(\Gamma - 1) - 1) \\ &+ (\kappa_{a,2,0} + \kappa_{a,2,1}\psi + \kappa_{a,2,1}\psi^2)(exp(2\Gamma - 2) - 1)\end{aligned} \tag{73b}$$

$$\hat{\kappa}_{rr,i} = \hat{\kappa}_{rr}^c + \kappa_{rr,2,1}\psi(exp(2\Gamma) - 1)(\Gamma - 1)^2 \tag{73c}$$

where

$$\Gamma = \frac{T_{c,i}}{T} \tag{74}$$

and

$$\psi = \frac{RT_{c,i}}{p_c} \tag{75}$$

The in total 16 κ-constants used in Eqs. (73) have been fitted to the SRK EOS and the PR EOS, among other EOSs. The values of the constants have been proposed by Quiñones-Cisneros et al. (2001) and are given in Table 4.

	SRK	**PR**
$\hat{\kappa}_a^c$	$-0.165\,302$	$-0.140\,464$
$\hat{\kappa}_r^c$	$6.995\,74\times10^{-3}$	$1.199\,02\times10^{-2}$
$\hat{\kappa}_{rr}^c$	$1.263\,58\times10^{-3}$	$8.551\,15\times10^{-4}$
$\kappa_{a,0,0}$	$-0.114\,804$	$-4.891\,97\times10^{-2}$
$\kappa_{a,1,0}$	$0.246\,622$	$0.270\,572$
$\kappa_{a,1,1}$	$-1.156\,48\times10^{-4}$	$-1.104\,73\times10^{-4}$
$\kappa_{a,2,0}$	$-3.946\,38\times10^{-2}$	$-4.481\,11\times10^{-2}$
$\kappa_{a,2,1}$	$4.188\,63\times10^{-5}$	$4.089\,72\times10^{-5}$
$\kappa_{a,2,2}$	$-5.919\,99\times10^{-9}$	$-5.797\,65\times10^{-9}$
$\kappa_{r,0,0}$	$-0.315\,903$	$-0.357\,875$
$\kappa_{r,1,0}$	$0.566\,713$	$0.637\,572$
$\kappa_{r,1,1}$	$-1.008\,60\times10^{-4}$	$-6.021\,28\times10^{-5}$
$\kappa_{r,2,0}$	$-7.299\,95\times10^{-2}$	-7.9024×10^{-2}
$\kappa_{r,2,1}$	$5.174\,59\times10^{-5}$	$3.724\,08\times10^{-5}$
$\kappa_{r,2,2}$	$-5.687\,08\times10^{-9}$	$-5.656\,10\times10^{-9}$
$\kappa_{rr,2,1}$	$1.359\,94\times10^{-8}$	$1.372\,90\times10^{-8}$

Table 4: Parameters for the friction theory Eqs. (73) for SRK and PR EOSs for ψ in $\mathrm{cm}^3/\mathrm{mol}$.

2.5 Statistics

When working with experimental data, statistical methods are essential to better interpret the findings. All measuring equipment, and consequently all measurement data, are associated with some degree of uncertainty. To present data effectively, statistical concepts such as averaging and confidence intervals must be applied. This section aims to facilitate better understanding of the experimental data discussed in the Method section.

2.5.1 Uncertainty Analysis

For experimental data to be considered credible, it is important that the measurements have low uncertainty. To ensure valid comparisons across studies, a consensus on the method for calculating uncertainty is essential.

One method of calculating the uncertainty is the *Guide to the Expression of Uncertainty in Measurement* (GUM) [17]. GUM was first published in 1993 after recognizing the lack of international consensus on the matter of uncertainty in measurement. Today, GUM is an ISO-standard for calculating uncertainty of measuring equipment and is applied in many experimental studies.

The expanded uncertainty, U, is a quantity that defines an interval about the result of a measurement within which the true value is expected to lie with a specified level of confidence. The purpose of U is to define an interval around the measurement result that is expected to contain a

significant portion of the distribution of values that could be assigned to the property intended to be measured [17]. In other words, if U is large in comparison to the size of the measurement data, there is a large interval around the measurement data where the true value of the measured data could lie - meaning that there is low precision in the measurement, so one cannot with confidence determine the true value of the result. The result of a measurement, Y, can therefore be expressed as $Y = y \pm U$, where y is the measured value.

From GUM, the expanded uncertainty, U, depends on a variable called the coverage factor, k. The choice of coverage factor k is based on the level of confidence for the interval of U. Generally, k will range from 2 to 3. The choice of k requires much experience and knowledge about the measurement equipment in question, and about how the measurement result should be put. Confidence levels are shown for three k-values in Table 5.

Coverage factor k	**Level of confidence** [%]
1	68.27
2	95.45
3	99.73

Table 5: Value of the coverage factor, k, and corresponding confidence interval, assuming a normal distribution.

2.5.2 Relative Deviation

Relative deviation as a good way to compare the accuracy of something relative to its numerical size. In this work, the absolute relative deviation (ARD) is used to present results from data comparison. The ARD in percent is defined as:

$$ARD = 100 \times \frac{|y_{model} - y_{exp}|}{|y_{exp}|} \tag{76}$$

where y_{model} represents the numerical value of the models employed in this study and y_{exp} represents the numerical value of the experimental data.

The average absolute relative deviation (AARD) is also a convenient and intuitive method for presenting the results of numerical data comparisons. The AARD is, as the name implies, the average of the ARD's. In this work, the AARD in percent is defined as:

$$AARD = \frac{1}{N} \sum_{i=1}^{N} ARD_i \tag{77}$$

where N is the number of data points.

3 Method

To determine the accuracy of the different EOSs and viscosity models for several components and states, they must be compared against experimental data. For a reliable comparison, the experimental data must be of high quality and have low uncertainty. Unfortunately such data sets are limited in availability [23].

The experimental data was collected within a temperature and pressure range relevant to the natural gas value chain. For Europipe II, the pipeline is designed to handle a maximum pressure of 189barg and temperatures as low as $-5°C$ [37]. Downstream of a typical export compressor, the gas can reach temperatures of 150°C [28]. Therefore, most of the experimental data was selected to be within these pressure and temperature ranges. As previously mentioned, publicly available experimental data for relevant compositions within the desired pressure and temperature range is limited. Since NG primarily consists of methane, binary mixtures of hydrogen and methane are included. Datasets for CO_2 were also collected, to test the applicability range of the EOSs.

3.1 Density

3.1.1 Experimental Data

The Leachman EOS for normal hydrogen is used as the baseline for hydrogen density data. Experimental data for pure hydrogen was not used, unlike for the other mixtures, because most of the publicly available data is outdated, and the uncertainty of these experiments is not well documented. Additionally, much of the available data falls outside of the temperature and pressure range relevant to this work. The Leachman EOS was chosen instead due to its reported low uncertainty of 0.04% [20], as well as its adoption by the National Institute of Standards and Technology(NIST) for presenting density data for pure hydrogen. For the temperature range of this study, only normal hydrogen is relevant, so pure ortho- and parahydrogen are excluded.

To acquire the pure hydrogen data, the Leachman model was programmed in Python. To achieve a suitable format of the density data, the densities were calculated for varying pressure at several isotherms. Since α^r is a function of τ and δ (see Eq. (31)) which means that α^r is implicitly a function of temperature and density, a bisection method was implemented in the Python code. Several versions of Newton's method were tried in the code, such as the *fsolve* function from the scipy.optimize Python library [38]. However, the Newton's method implemented in the *fsolve* function failed because of poor initial guesses. Therefore, the bisection method was used because of its robustness [40].

Four different studies were used to collect experimental density data for mixtures with hydrogen. The studies fit the criteria for experimental studies presented above. Relevant information about the studies are summarized in Table 6. The documented uncertainty of the studies are shown in Table 7.

Source	Year	Mixture	x_{H_2}	T [K]	p [MPa]	Data points
Hernandez-Gomez et al. [16]	2018	CH_4 + H_2	0.05 - 0.5	240 - 350	1 - 20	391
Ritcher et al. [36]	2014	NG + H_2	0.05 - 0.3	273.15 - 293.15	1 - 8	86
Ben Souissi et al. [3]	2017	CO_2 + H_2	0.05	273.15 - 323.15	0.5 - 6	41
Hernández-Gómez et al. [15]	2018	NG + H_2	0.03	260 - 350	1 - 20	99

Table 6: Summary of studies for density of hydrogen mixtures. Showing the mixture, molar fraction of hydrogen, temperature and pressure range in K an MPa, respectively, and number of data points. NG compositions for different studies are presented in tables 8 and 9.

Source	$U(\rho_{\mathbf{exp}})/\rho_{\mathbf{exp}}$ **(k = 2)** [%]
Hernandez-Gomez et al. [16][1]	0.059-0.55
Hernandez-Gomez et al. [16][2]	0.033-0.57
Hernandez-Gomez et al. [16][3]	0.039-0.56
Ritcher et al. [36]	0.02
Ben Souissi et al. [3]	0.074
Hernández-Gómez et al. [15]	0.029-0.502

Table 7: The relative expanded combined density uncertainty in the measurements from different studies. The superscripts [1], [2], and [3] correspond to different mole fractions of the $CH_4 + H_2$ mixture, specifically $(0.95 + 0.05)$, $(0.9 + 0.1)$, and $(0.5 + 0.5)$, respectively.

The study by Hernández-Gómez et al.[16] was the most detailed study used for the collection of experimental density data. The study covers large temperature and pressure ranges for three molar fractions of hydrogen and methane: ($0.95CH_4$ + $0.05H_2$), (0.9 CH_4 + 0.1 H_2) and (0.5 CH_4 + 0.5 H_2). The study employs the GUM to document uncertainty in the measurements, thereby validating their results.

The study by Richter et al. is a smaller study with a narrower temperature and pressure range, examining realistic NG mixtures presented in Table 8. The study does not use the GUM for uncertainty calculations, but takes into account the uncertainties of temperature, pressure, and density, and calculates the relative expanded uncertainty in a similar way as in the GUM. However, the study does not account for the uncertainty in composition for the different mixtures, NG1, NG2, and NG3 shown in Table 8. This can be a source of error. For NG3, Richter et al. states that there were inaccuracies in the measured composition, caused by the high molar fraction of hydrogen. Regardless, the data from the study is still used in this work, because it is the only study that was found that used actual NG-compositions with varying degrees of hydrogen - something that is highly interesting in the context of the natural gas value chain. However, the results from Richter et al. must be interpreted cautiously.

The study by Ben Souissi et al. is a simpler study examining only one mixture of CO_2 and hydrogen, (0.94638 CO_2 + 0.05362 H_2). The study employs the GUM to document uncertainty in the measurements, thereby validating their results. The pressure and temperature range of the study is narrow, being that CO_2's critical point is close to atmospheric conditions, with a critical temperature and pressure of 304K and 7.38MPa. Since some of the data points are fairly close to the liquid phase for CO_2, the pressure is quite low for certain isotherms.

The study by Hernández-Gómez et al. [15] examines a single NG composition, presented in Table 9, with a hydrogen molar fraction of only 3%. The study employs the GUM to document uncertainty in the measurements, thereby validating their results. Unlike the study by Richter et al., Hernández-Gómez et al. [15] does not provide an in-depth analysis of hydrogen's effect on the density of mixtures. Nonetheless, the study covers a sufficient temperature and pressure range.

Component	NG1	NG2	NG3
CH_4	0.85381	0.810099	0.628871
C_2H_6	0.054534	0.050884	0.039369
C_3H_8	0.010347	0.009147	0.007226
n-C_4H_{10}	0.001479	0.001273	0.001017
iso-C_4H_{10}	0.001355	0.001217	0.000955
n-C_5H_{12}	0.000256	0.000229	0.00018
iso-C_5H_{12}	0.000355	0.000323	0.000253
neo-C_5H_{12}	0.000015	0.000014	0.000011
n-C_6H_{14}	0.000214	0.000198	0.000152
n-C_7H_{16}	0.000087	0.000081	0.000067
n-C_8H_{18}	0.000008	0.000008	0.000006
n-C_9H_{20}	0.000001	0.000001	0.000001
n-$C_{10}H_{22}$	<0.000001	<0.000001	<0.000001
C_6H_6	0.000008	0.000007	0.000006
C_7H_8	0.000004	0.000004	0.000003
o-C_8H_{10}	0.000001	0.000001	<0.000001
CO_2	0.016487	0.015529	0.01193
N_2	0.007284	0.006803	0.005212
O_2	<0.000100	<0.000100	<0.000100
He	0.000074	0.000076	0.000036
H_2	0.053681	0.104106	0.304705

Table 8: Composition of Natural Gas Samples from (Richter et al., 2014). Where the hydrogen molar fraction for NG1, NG2 and NG3 is 5.37%, 10.41% and 30.47%, respectively.

Component	NG
Methane	78.821237
Ethane	0.757359
Propane	0.297078
Butane	0.200439
Isobutane	0.197953
Pentane	0.050134
Isopentane	0.049928
Neopentane	0.049615
Hexane	0.050708
Carbon dioxide	4.001075
Nitrogen	12.017829
Oxygen	–
Helium	0.496897
Hydrogen	3.009733

Table 9: Composition of Gas Sample from (Hernández-Gómez et al., 2018)[15]. With a hydrogen molar fraction of approximately 3.01%.

3.1.2 Simulation Data

Most of the traditional EOSs, such as the SRK and the PR EOS, are implemented in established process simulation software. Still, the same EOS can yield two different results across different software. This is because different software often have varying constants for the same components and states. Especially the value of the BICs described in section 2.2 can differ between software. In Appendix C, The BICs from NeqSim and HYSYS are compared to show differences between the two software. For some of the binary mixtures, the BICs are not defined in NeqSim's database. When k_{ij} is set to zero, the interaction between the two components is modelled as ideal.

3.1.3 NeqSim

Python scripts were developed to compare models in NeqSim to the experimental data from the studies presented in Table 6. All Python code created for this work is presented in the Appendix.

An attempt was made to program the GERG2008 EOS in Python to gain a better understanding of the EOS. The Python code employs the equations from Section 2.3.2, and uses a bisection method, similarly to the code for Leachman EOS. The code was able to calculate densities for pure components, including pure hydrogen. To obtain data for pure hydrogen, the programmed GERG2008 EOS was used, and the Python code can be found in Appendix B.1. However, extending the code to handle mixtures proved time consuming. After verifying that NeqSim's implementation of the GERG2008 EOS was defined according to Kunz and Wagner, the attempt to program the model in Python was ended. Consequently, NeqSim's GERG2008 is used for mixtures in this work.

The NG-compositions presented in Tables 8 and 9 include compounds that the GERG2008 EOS is not fitted for (see Table 1). These NG mixtures contain compounds such as benzene (C_6H_6), toluene (C_7H_8), o-xylene (o-C_8H_10) and neopentane (neo-C_5H_{12}), which are not among the 21 compounds GERG2008 is fitted for. Additionally, the compound neopentane is not defined in NeqSim. When a mixture contains compounds that are not defined in NeqSim, the GERG2008 implementation in NeqSim will instead model these compounds as one of the 21 compounds defined for the GERG2008 EOS with the closest molar mass. In practice, this means that benzene, toluene and o-xylene are modelled as n-hexane, n-heptane and n-octane, respectively. Similarly, since neopentane is not defined in NeqSim, it is modelled as iso-pentane, due to their closer boiling points compared to those of neopentane and n-pentane [30][29][31].

3.1.4 HYSYS

Density data from HYSYS was collected for the PR and SRK EOS using the property table feature. This data was subsequently plotted and compared with experimental data, alongside density data generated by NeqSim.

3.2 Viscosity

3.2.1 Experimental Data

The sources for experimental viscosity data used in this study are listed in Table 10. Although efforts were made to find additional sources of experimental data, no other sources fitting the scope of this study were found. In addition, some of the data from the studies used in this work also fall outside of the scope of this work. For example, the studies by Owuna et al. and Chuang et al. include measurements at temperatures close to the critical point of methane, $190.564K$ [19]. Near the critical point, the uncertainty of the measurements increases significantly [7]. Therefore, viscosity data for temperatures lower than $248K$ are not used in this work.

Source	Year	Mixture	x_{H_2}	T [K]	p [MPa]	Data points
Owuna et al. [24]	2024	CH_4 + H_2	0.0 - 0.5, 1.0	213 - 324	1 - 30	253
Betken et al. [4]	2024	CH_4 + H_2	0.1	298 - 350	0.1 - 18	36
Chuang et al. [7]	1976	CH_4 + H_2	0.0 - 1.0	173.15 - 273.15	0.4 - 50.6	219

Table 10: Summary of studies for experimental viscosity data of methane-hydrogen mixtures collected for this study. Showing the mixture, molar fraction of hydrogen, temperature and pressure range in K an MPa, respectively, and number of data points.

Table 11 shows the uncertainty of the collected experimental viscosity data. The lower part of the uncertainty range for each study corresponds to measurements at higher temperatures, and vice versa.

Owuna et al. and Betken et al. employ the GUM methodology in their studies to calculate the uncertainty in viscosity, thereby improving the reliability of their data. In contrast, Chuang et al. does not provide details of the uncertainty is calculated, but reports that the actual error is of the order of ±0.5%. Despite this, the results of the study show good agreement with other similar studies for pure methane and pure hydrogen, which improves the reliability of their findings. As a result, their data is used for comparison in this study.

Source	x_{H_2}	$U(\eta_{exp})/\eta_{exp}$ (k = 2) [%]
Owuna et al. [24]	0.0	2.5 - 5.2
Owuna et al. [24]	0.1	2.9 - 5.4
Owuna et al. [24]	0.2	2.9 - 6.4
Owuna et al. [24]	0.5	2.8 - 4.0
Owuna et al. [24]	1.0	2.8 - 6.6
Betken et al. [4]	0.9	0.9 - 3.2
Betken et al. [4]	1.0	0.7 - 2.7
Chuang et al. [7]	0.0	0.5
Chuang et al. [7]	0.2	0.5
Chuang et al. [7]	1.0	0.5

Table 11: The relative expanded combined data uncertainty in the studies where experimental data has been collected.

3.2.2 Simulation Data

The three viscosity models available in NeqSim - LBC, FT and PFCT - are used for simulation data for viscosity. The simulations were performed using a Python script, which calculated the viscosities for each pressure-temperature combination and composition from the experimental data. For the LBC and FT models, both the SRK and PR EOS were implemented in the models for comparison. However, the PFCT model in NeqSim is hard-coded to use the SRK EOS, making comparison with the PR EOS impossible. The modelled viscosities were then compared to the experimental data for the corresponding pressure-temperature combinations and compositions, and the results were plotted. The constants for the FT model in NeqSim match those listed in Table 4, with $\varepsilon = 0.3$.

4 Results

4.1 Density

4.1.1 Pure Hydrogen

Figure 1 shows the relative deviation between the density models and the Leachman model, which is used instead of experimental pure hydrogen data. The plots show that temperature and pressure has an effect on the accuracy of the models. Notably, the GERG2008 EOS has small to no deviation from the Leachman model. The PR models from HYSYS and NeqSim both show increasing deviations from Leachman as the pressure increases. SRK performs drastically better than PR for both HYSYS and NeqSim, having an ARD below 1% for all isotherms. A summary of the AARD (average absolute relative difference) and maximum ARD (absolute relative difference) is presented in Table 12.

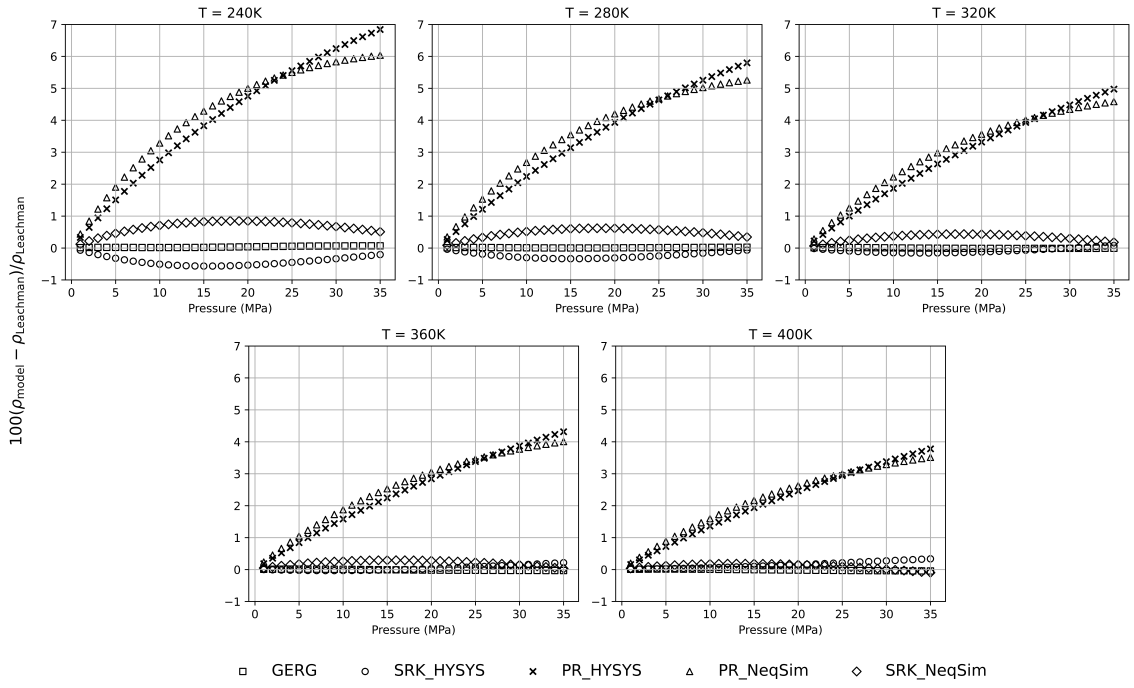

Figure 1: Plots of isotherms, showing the relative difference comparing the different density models to the Leachman model for normal hydrogen.

EOS	AARD (%)	Maximum ARD (%)
GERG2008	0.00391	0.0674
SRK-NeqSim	0.355	0.848
SRK-HYSYS	0.106	0.335
PR-NeqSim	3.14	6.04
PR-HYSYS	3.02	6.84

Table 12: Mean and maximum relative deviation of the density models from the Leachman model for the temperature and pressure range of $240 - 400$ K and $0 - 35$ MPa.

4.1.2 Model Comparison with Experimental Data from Hernández-Gómez et al. [16]

Figures 2, 3 and 4 show the relative deviation between the modelled densities and the experimental data from Hernández-Gómez et al.[16] for the mixtures (0.95 CH_4 + 0.05 H_2), (0.9 CH_4 + 0.1 H_2) and (0.5 CH_4 + 0.5 H_2), respectively. The GERG2008 EOS is highly accurate, showing little deviation from the experimental data. It displays an AARD below 0.1% and a maximum ARD below 0.3% for all three mixtures.

HYSYS' and NeqSim's implementations of PR tend to overestimate the density, while the HYSYS SRK underestimates. NeqSim's implementation of SRK is more accurate compared to the other SRK and PR implementations, and the accuracy increases with increasing hydrogen molar fraction. The differences between the HYSYS and NeqSim implementations grow bigger with increasing molar fractions of hydrogen. A summary of the AARDs and maximum ARDs for the different models is presented in Table 13.

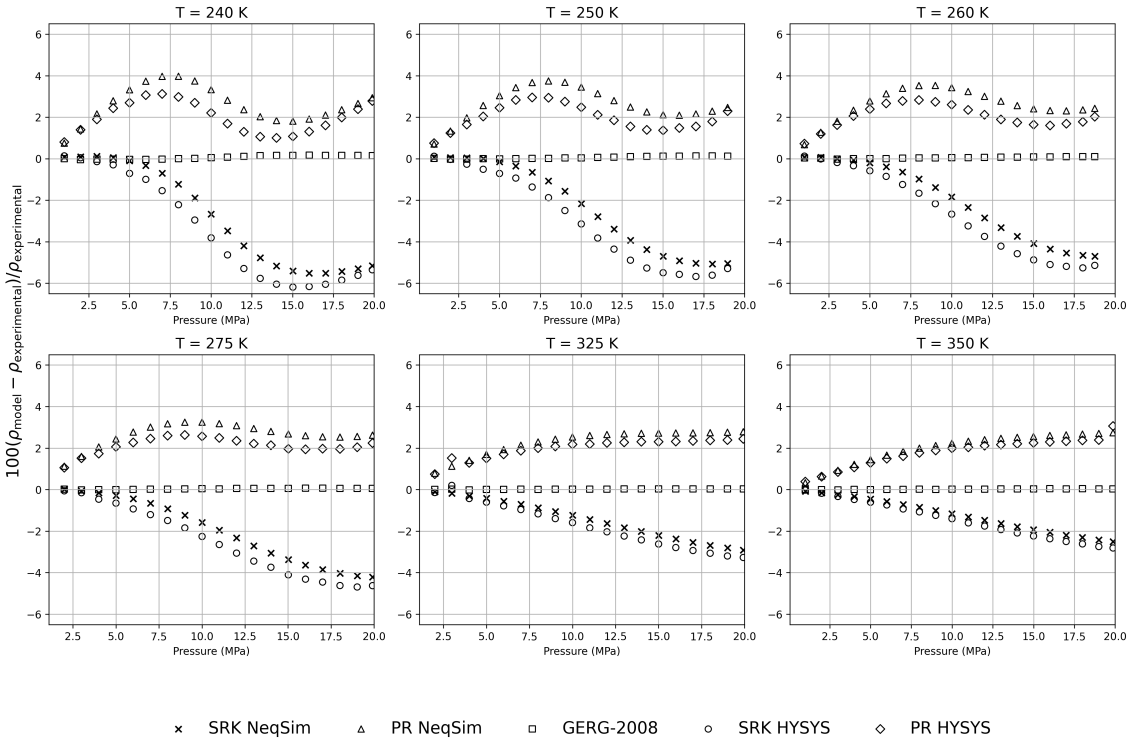

Figure 2: Plots of isotherms, showing the relative difference between the models and experimental data, for the mixture of (0.95 CH_4 + 0.05 H_2), from Hernández-Gómez et al.[16]'s study.

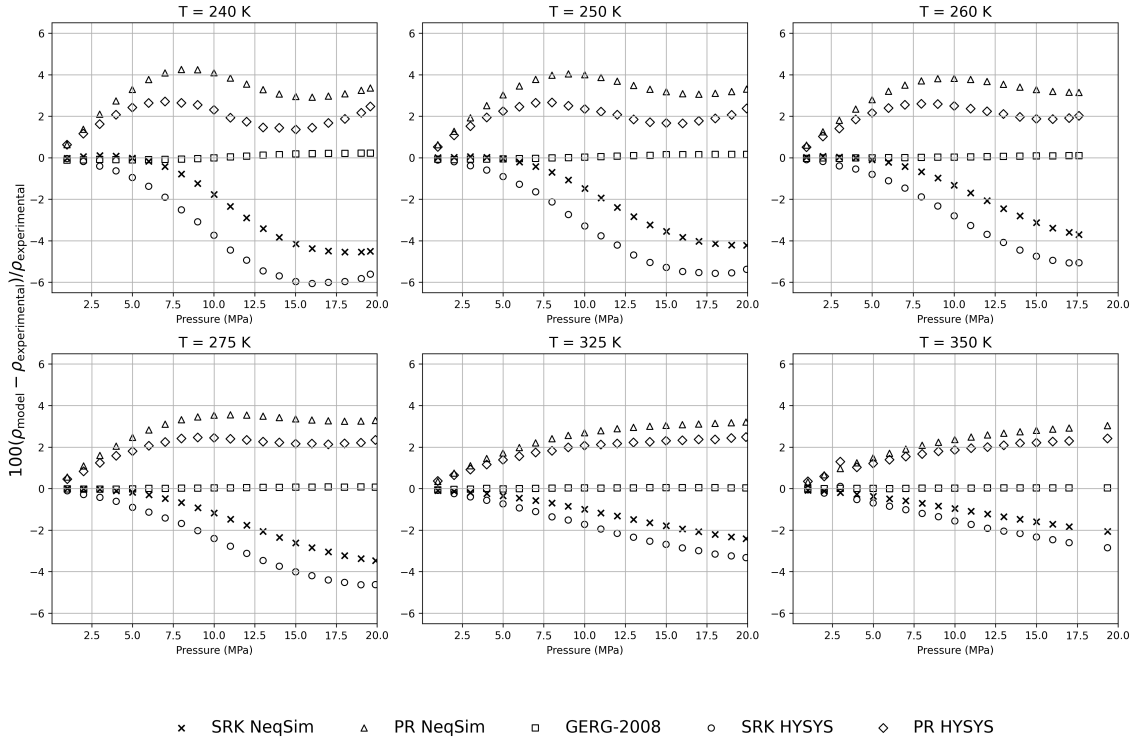

Figure 3: Plots of isotherms, showing the relative difference between the models and experimental data, for the mixture of (0.9 CH_4 + 0.1 H_2), from Hernández-Gómez et al.[16]'s study.

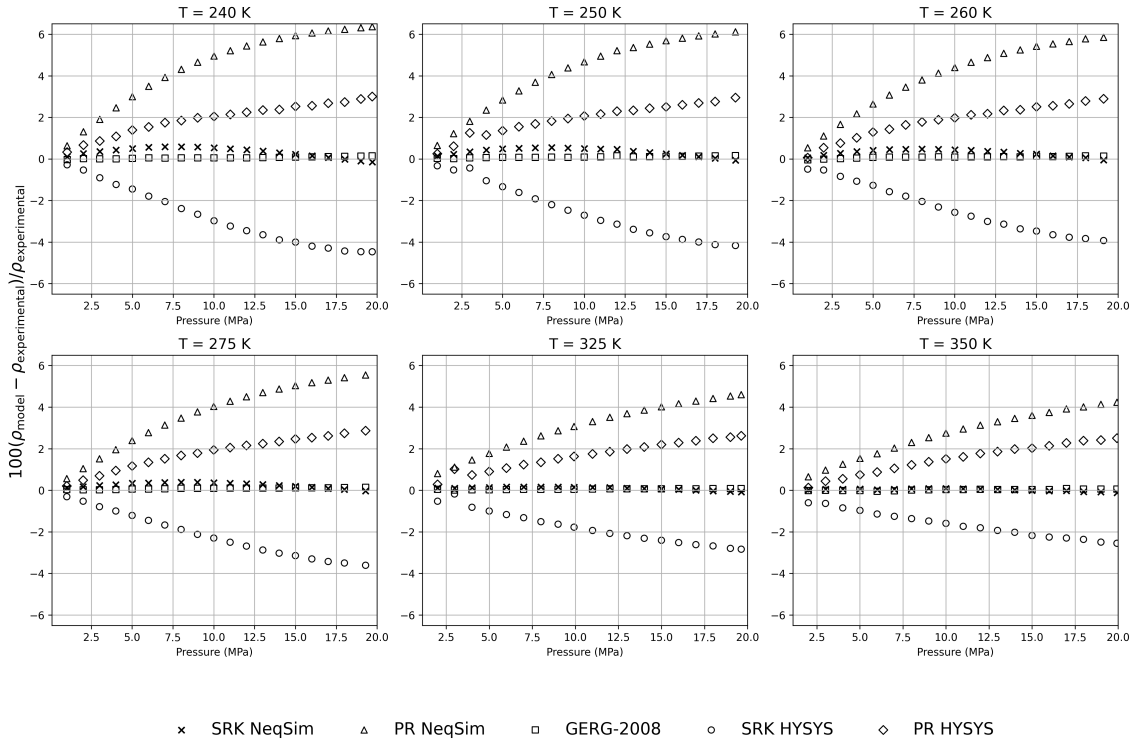

Figure 4: Plots of isotherms, showing the relative difference between the models and experimental data, for the mixture of (0.5 CH_4 + 0.5 H_2), from Hernández-Gómez et al.[16]'s study.

	AARD (%)	Maximum ARD (%)
EOS	**0.95 CH_4 + 0.05 H_2**	
GERG2008	0.0381	0.165
SRK-NeqSim	1.96	5.51
SRK-HYSYS	2.43	6.19
PR-NeqSim	2.45	3.97
PR-HYSYS	2.00	3.12
	0.90 CH_4 + 0.10 H_2	
GERG2008	0.0284	0.220
SRK-NeqSim	1.50	4.55
SRK-HYSYS	2.46	6.06
PR-NeqSim	2.75	4.25
PR-HYSYS	1.91	2.71
	0.5 CH4 + 0.5 H2	
GERG2008	0.0648	0.156
SRK-NeqSim	0.214	0.588
SRK-HYSYS	2.23	4.47
PR-NeqSim	3.69	6.36
PR-HYSYS	1.78	3.00

Table 13: Mean and maximum ARD from the experimental density from Hernández-Gómez et al.[16] for the temperature and pressure range of $240 - 350$ K and $0 - 20$ MPa, for the different compositions.

4.1.3 Model Comparison with Experimental Data from Richter et al.

Figures 5, 6 and 7 show the relative deviation between the modelled densities and the experimental data from Richter et al. for the mixtures NG1, NG2 and NG3 presented in Table 8. Again, GERG2008 has almost no deviation from the experimental density, with a max ARD below 0.1% for NG1 and NG2. For NG3, GERG2008 has a higher ARD, with both an average and max ARD just above 0.2%.

The trends observed for the SRK and PR EOS in the study by Hernández-Gómez et al. [16] are also evident in the plots generated for this study. A summary of the AARDs and maximum ARDs for the different models is presented in Table 14.

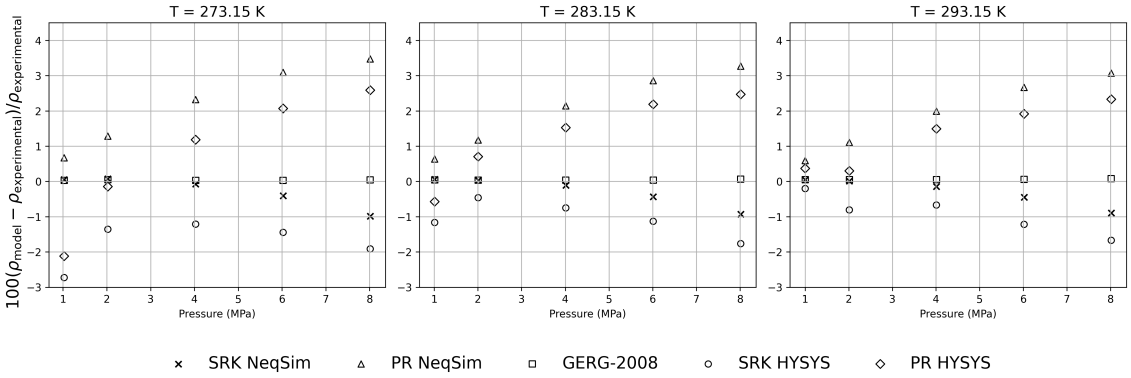

Figure 5: Plots of isotherms, showing the relative difference between the models and experimental data from Richter et al. for NG1, from table 8, with a hydrogen mole fraction of approximately 5%

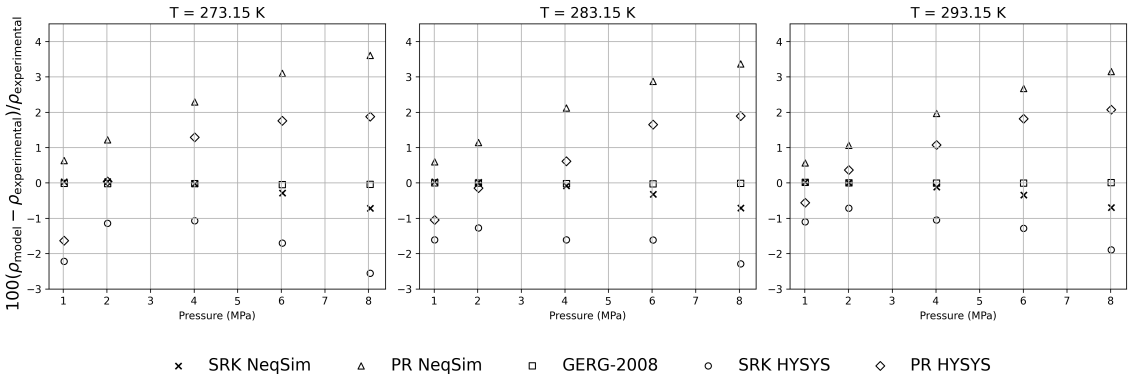

Figure 6: Plots of isotherms, showing the relative difference between the models and experimental data from Richter et al. for NG2, from table 8, with a hydrogen mole fraction of approximately 10%

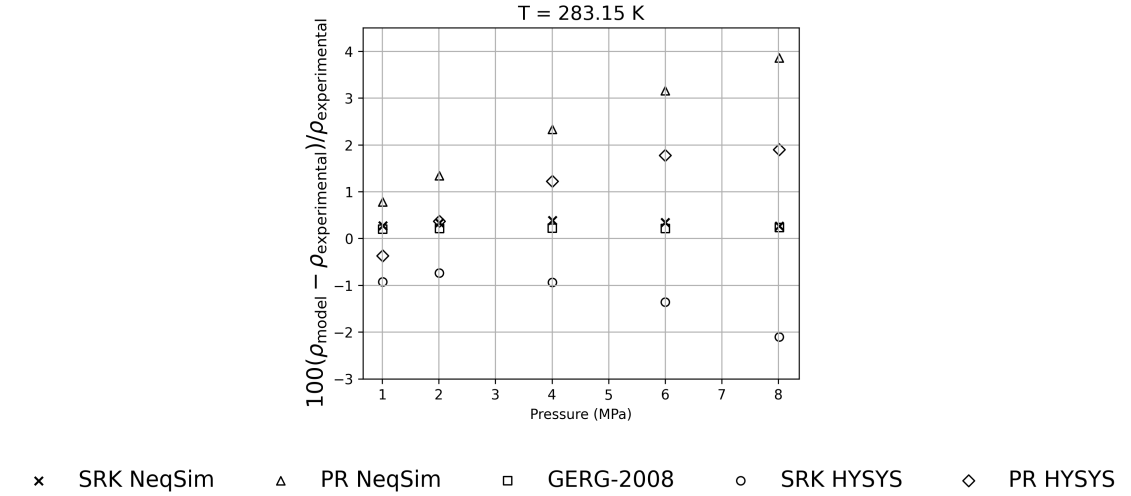

Figure 7: A plot for T = 283.15 K, showing the relative difference between the models and experimental data from Richter et al. for NG3, from table 8, with a hydrogen mole fraction of approximately 30%

	AARD (%)	Maximum ARD (%)
EOS	**NG1 (H_2 = 0.05)**	
GERG2008	0.0446	0.0782
SRK-NeqSim	0.277	0.986
SRK-HYSYS	1.24	2.73
PR-NeqSim	2.02	3.47
PR-HYSYS	1.08	2.59
	NG2 (H_2 = 0.1)	
GERG2008	0.0143	0.0498
SRK-NeqSim	0.211	0.722
SRK-HYSYS	1.55	2.56
PR-NeqSim	2.02	3.60
PR-HYSYS	0.734	2.07
	NG3 (H_2 = 0.3)	
GERG2008	0.215	0.236
SRK-NeqSim	0.320	0.385
SRK-HYSYS	1.22	2.11
PR-NeqSim	2.29	3.86
PR-HYSYS	0.975	1.90

Table 14: Average and maximum absolute relative deviation from the experimental density from Richter et al., for the temperature and pressure range of $273.15 - 293.15$ K and $0 - 8$ MPa, for the different compositions of NG, presented in Table 8

4.1.4 Model Comparison with Experimental Data from Ben Souissi et al.

Figure 8 shows the relative deviation between the modelled densities and the experimental data from Ben Souissi et al. for the mixture of CO_2 and H_2. The plots show a max ARD of 0.39% and a mean ARD of approximately 0.080% for the GERG2008 model.

The implementations of SRK and PR once again exhibit similar trends, apart from NeqSim's SRK which is consistently underestimating the density. A summary of the AARDs and maximum ARDs for the different models is presented in Table 15.

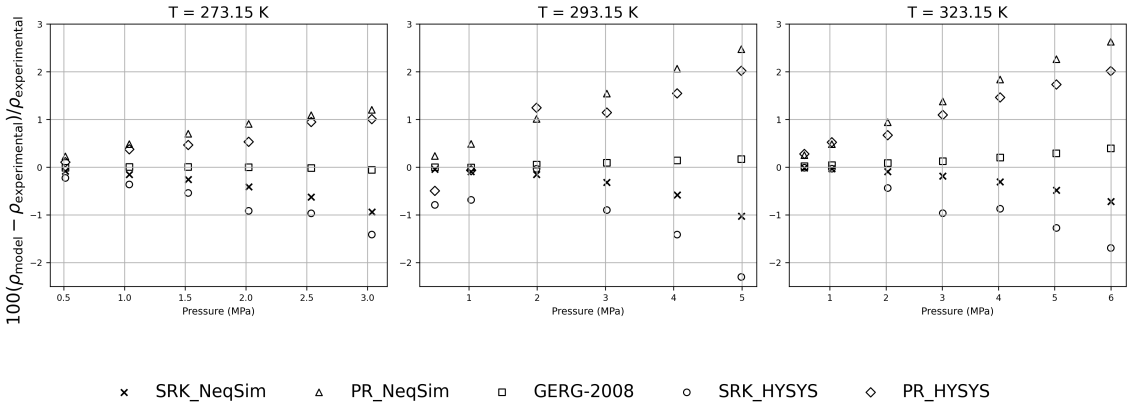

Figure 8: Plots of isotherms, showing the relative difference between the models and experimental data from Ben Souissi et al., for the mixture of (0.94638 CO_2 + 0.05362 H_2)

EOS	AARD (%)	Maximum ARD (%)
GERG2008	0.0791	0.394
SRK-NeqSim	0.345	1.03
SRK-HYSYS	0.835	2.30
PR-NeqSim	1.17	2.63
PR-HYSYS	0.872	2.02

Table 15: Mean and maximum relative deviation from the experimental density from Ben Souissi et al. for the temperature and pressure range of $273.15 - 323.15$ K and $0 - 5$ MPa, for a composition of (0.94638 CO_2 + 0.05362 H_2).

4.1.5 Model Comparison with Experimental Data from Hernández-Gómez et al.[15]

Figure 9 shows the relative deviation between the modelled densities and the experimental data from Hernández-Gómez et al.[15] for the NG composition presented in Table 9. The GERG2008 EOS is very accurate with a mean and maximum ARD of 0.10% and 0.18%, respectively.

The PR EOS follows the same trend observed in the other studies. For this study, NeqSim's SRK consistently underestimates the density, with the difference between HYSYS and NeqSim's implementations being smaller. A summary of the AARDs and maximum ARDs for the different models is presented in Table 16.

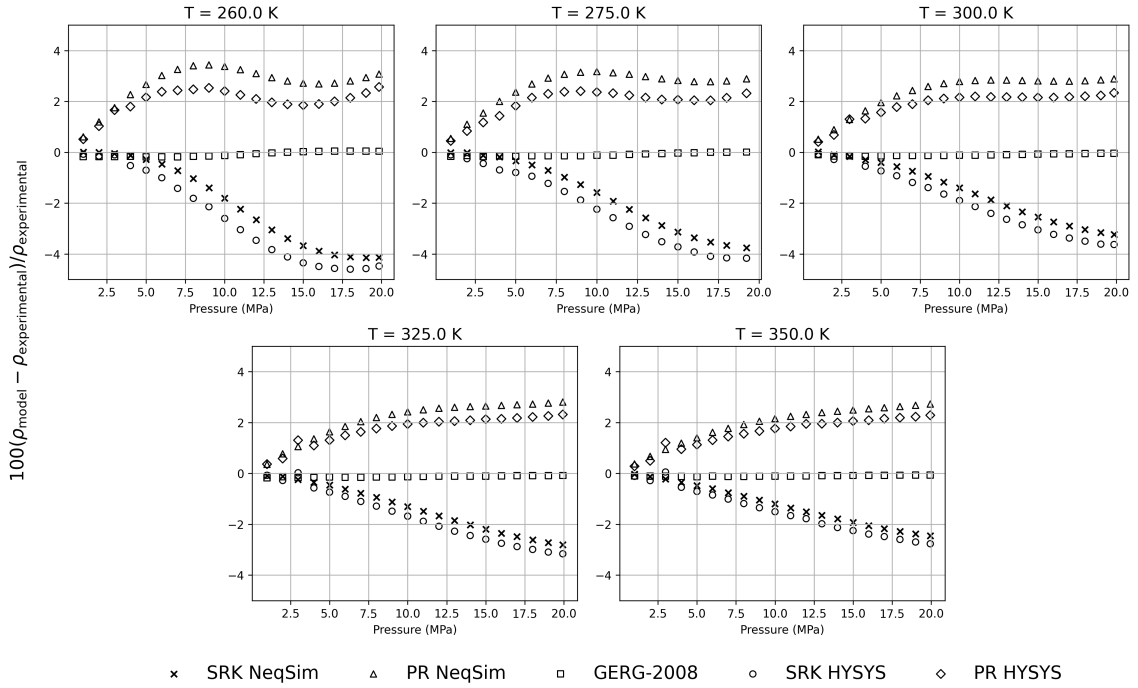

Figure 9: Plots of isotherms, showing the relative difference between the models and experimental data from Hernández-Gómez et al.[15], for NG mixture in Table 9,with a hydrogen mole fraction of approximately 3%.

EOS	AARD (%)	Maximum ARD (%)
GERG2008	0.101	0.183
SRK-NeqSim	1.60	4.14
SRK-HYSYS	2.00	4.60
PR-NeqSim	2.34	3.43
PR-HYSYS	1.83	2.57

Table 16: Mean and maximum relative deviation from the experimental density from Hernández-Gómez et al. for the temperature and pressure range of $260 - 350$ K and $0 - 20$ MPa, for the NG-compostion presented in table 9 ($H_2 = 3\%$).

4.1.6 Binary Interaction Coefficients

From the tables in Appendix C, it is clear that there are big differences between the BICs in HYSYS and NeqSim. For some binary mixtures, the BICs are missing in NeqSim's list of coefficients.

4.2 Viscosity

4.2.1 Pure Methane

Figure 10 shows the relative difference between the experimental viscosity data for pure methane and the three viscosity models: LBC, PFCT and FT. The figure shows that both temperature and pressure affect the accuracy of the viscosity models. The plot for 248K shows that the LBC$_{pr}$ model is very accurate when the pressure exceeds 20MPa. However, none of the models are able to accurately predict the viscosity of pure methane for the whole observed temperature-pressure range.

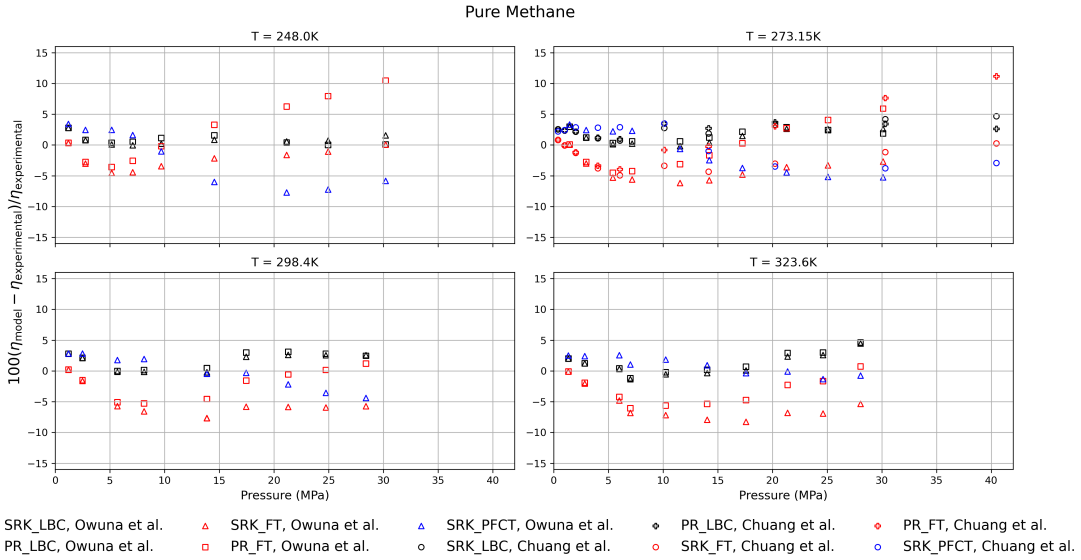

Figure 10: Percentage viscosity deviations of experimental viscosity data for pure methane of Owuna et al. and Chuang et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model.

Table 17 shows the mean and maximum percentage deviations of the experimental data and the viscosity models, for the same data points used in Figure 10. The table shows that the LBC$_{srk}$ model has the lowest AARD of 1.65%, with LBC$_{pr}$ right behind with an AARD of 1.67%. However, the LBC$_{pr}$ model has a lower maximum relative difference than the LBC$_{srk}$ model.

Vicsosity model	AARD (%)	Max ARD (%)
LBC_{srk}	1.65	5.87
LBC_{pr}	1.67	4.63
FT_{srk}	3.98	8.28
FT_{pr}	3.44	14.22
$PFCT_{srk}$	2.64	7.72

Table 17: Mean and maximum percentage viscosity deviations of the experimental viscosity data for pure methane from the same modelled viscosity values used in Figure 10.

4.2.2 Pure Hydrogen

Figure 11 shows the relative difference between the experimental viscosity data for pure hydrogen and the viscosity models. The PFCT model is not shown in any of the plots in the figure, because the NeqSim code for the PFCT viscosity model is not able to produce a numeric result, it returns 'nan', when the hydrogen mole fraction exceeds 75%. This was discovered through an iterative

process, increasing the hydrogen mole fraction with each iteration. Fortunately, the two other viscosity models in NeqSim, LBC and FT, are able to produce numeric results.

The figure shows that the FT viscosity model is unable to successfully predict the viscosity for pure hydrogen, with the error growing exponentially as pressure increases. The LBC model on the other hand, gives a more stable result, but also fails to predict the viscosity accurately as it underestimates by around 10% across the observed temperature-pressure range.

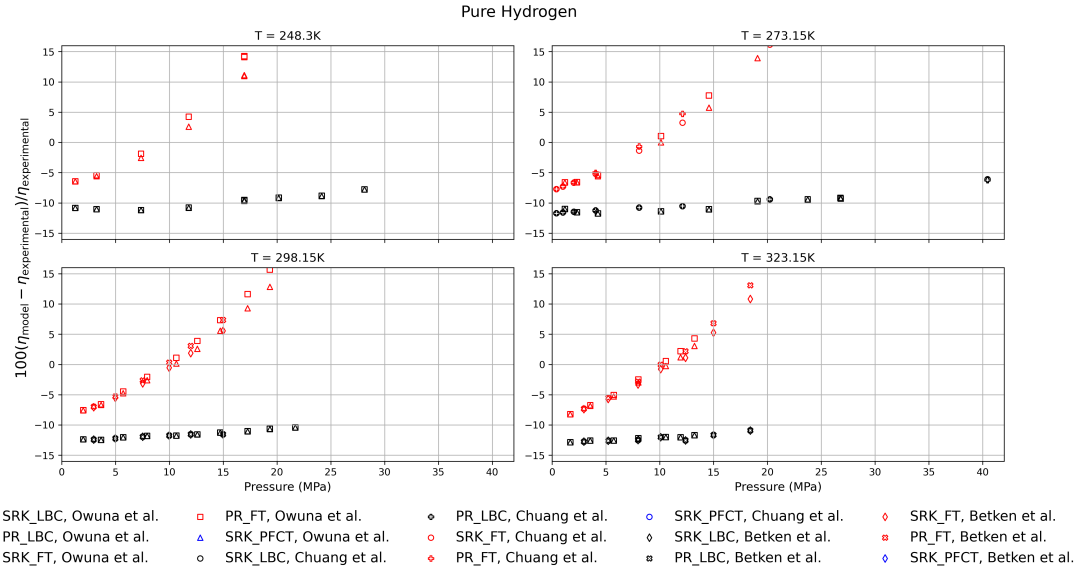

Figure 11: Percentage viscosity deviations of experimental viscosity data for pure hydrogen of Owuna et al., Betken et al. and Chuang et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model.

Table 18 shows the mean and maximum percentage viscosity deviations of the experimental data and the viscosity models, for the same data points used in Figure 11. It becomes clear from the table that none of the two viscosity models are able to accurately predict the viscosity of pure hydrogen.

Vicsosity model	AARD (%)	Max ARD (%)
LBC_{srk}	11.01	12.97
LBC_{pr}	11.00	12.87
FT_{srk}	10.35	93.39
FT_{pr}	12.06	108.89
$PFCT_{srk}$	-	-

Table 18: Mean and maximum percentage viscosity deviations of the experimental viscosity data for pure hydrogen from the same modeled viscosity values used in Figure 11.

4.2.3 Mixture: $0.9CH_4 + 0.1H_2$

Figure 12 shows the relative difference between the experimental viscosity data for a $0.9CH_4$+$0.1H_2$ mixture and the viscosity models. The plots in the figure show that all of the viscosity models struggle to consistently accurately predict the viscosity, however the PFCT model performs quite well in the plot for 323.7K. The low mole fraction of hydrogen does not seem to have a huge effect on the viscosity models' ability to predict viscosity, compared to the plots for pure methane in Figure 10.

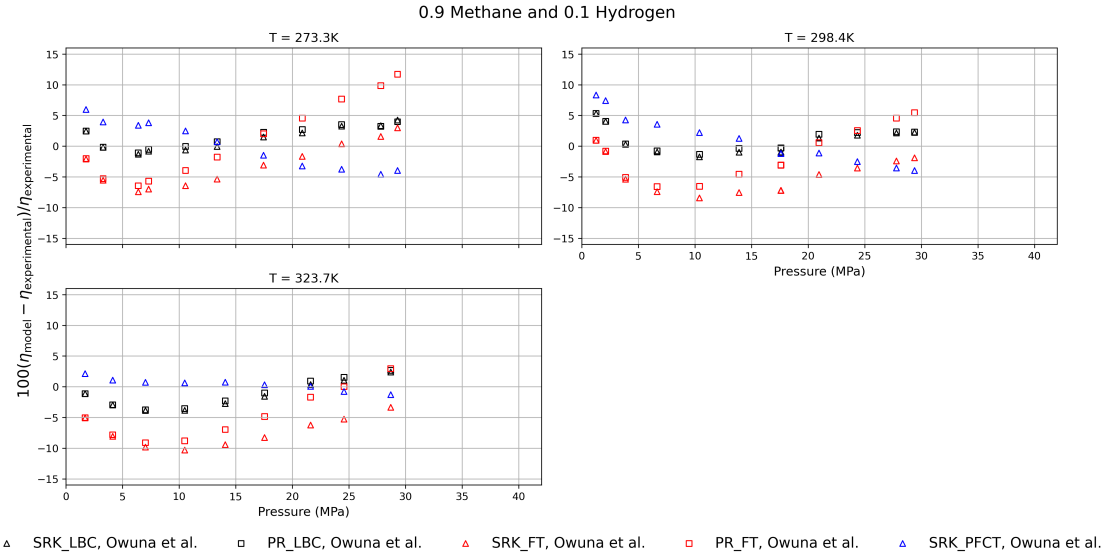

Figure 12: Percentage viscosity deviations of experimental viscosity data for composition with mole fraction 0.9 CH_4 and 0.1 H_2 of Owuna et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model.

Table 19 shows the mean and maximum percentage viscosity deviations of the experimental data and the viscosity models, for the same data points used in Figure 12. The AARD for the LBC model is still the lowest compared to the other models, but it shows a slight increase in AARD compared to its performance for pure methane, shown in Table 17. For the LBC_{srk} model, the max ARD decreased slightly compared to the max ARD for pure methane. The FT model shows a slight decrease in performance, having a higher ARD than it had for pure methane. The AARD for the $PFCT_{srk}$ model remains practically unchanged compared to the AARD for pure methane, but the model shows a slight increase in the max ARD.

Vicsosity model	**AARD (%)**	**Max ARD (%)**
LBC_{srk}	1.93	5.31
LBC_{pr}	1.89	5.31
FT_{srk}	5.28	10.32
FT_{pr}	4.70	11.70
$PFCT_{srk}$	2.63	8.32

Table 19: Mean and maximum percentage viscosity deviations of the experimental viscosity data for composition with mole fraction 0.9 methane and 0.1 hydrogen from the same modelled viscosity values used in Figure 12.

4.2.4 Mixture: $0.8CH_4 + 0.2H_2$

Figure 13 shows the relative deviation between the experimental viscosity data for a $0.8CH_4$+$0.2H_2$ mixture and the viscosity models. The plots in the figure show that the accuracy of the PFCT model increases with increasing pressure, whereas the error of the FT model grows when pressure increases. The LBC model demonstrates a larger amplitude in the shape of its curve compared to Figure 10 for pure methane and Figure 12 for the $0.9CH_4$+$0.1H_2$ mixture.

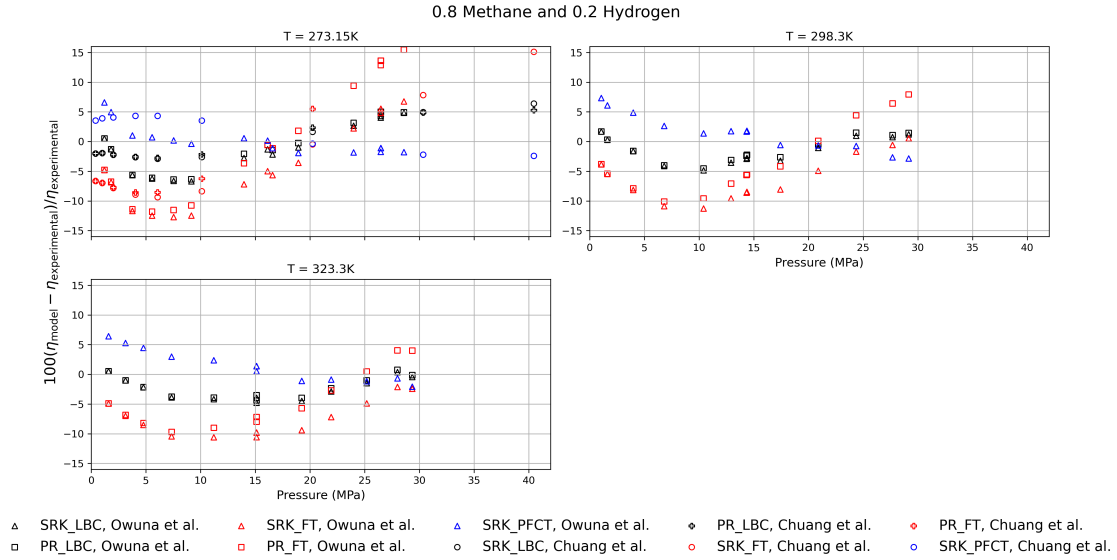

Figure 13: Percentage viscosity deviations of experimental viscosity data for composition with mole fraction 0.8 CH_4 and 0.2 H_2 of Owuna et al. and Chuang et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model.

Table 20 shows the mean and maximum percentage viscosity deviations of the experimental data and the viscosity models, for the same data points used in Figure 13. The PFCT model shows a decrease in both AARD and max ARD compared to its performance for pure methane and the $0.9CH_4$+$0.1H_2$ mixture. In contrast, the other viscosity models perform worse compared to their accuracy for the lower hydrogen mole fractions.

Vicsosity model	**AARD (%)**	**Max ARD (%)**
LBC_{srk}	3.01	6.70
LBC_{pr}	2.8	6.39
FT_{srk}	7.42	20.46
FT_{pr}	8.18	34.77
$PFCT_{srk}$	2.37	7.32

Table 20: Mean and maximum percentage viscosity deviations of the experimental viscosity data for composition with mole fraction 0.8 CH_4 and 0.2 H_2 from the same modeled viscosity values used in Figure 13.

4.2.5 Mixture: $0.5CH_4 + 0.5H_2$

Figure 14 shows the relative difference between the experimental viscosity data for a $0.5CH_4$+$0.5H_2$ mixture and the viscosity models. The plots in the figure show that the PFCT model is able to consistently predict the viscosity quite accurately. In contrast, the LBC- and FT models perform worse with the high mole fraction of hydrogen compared to their accuracy for pure methane and the lower hydrogen mole fractions.

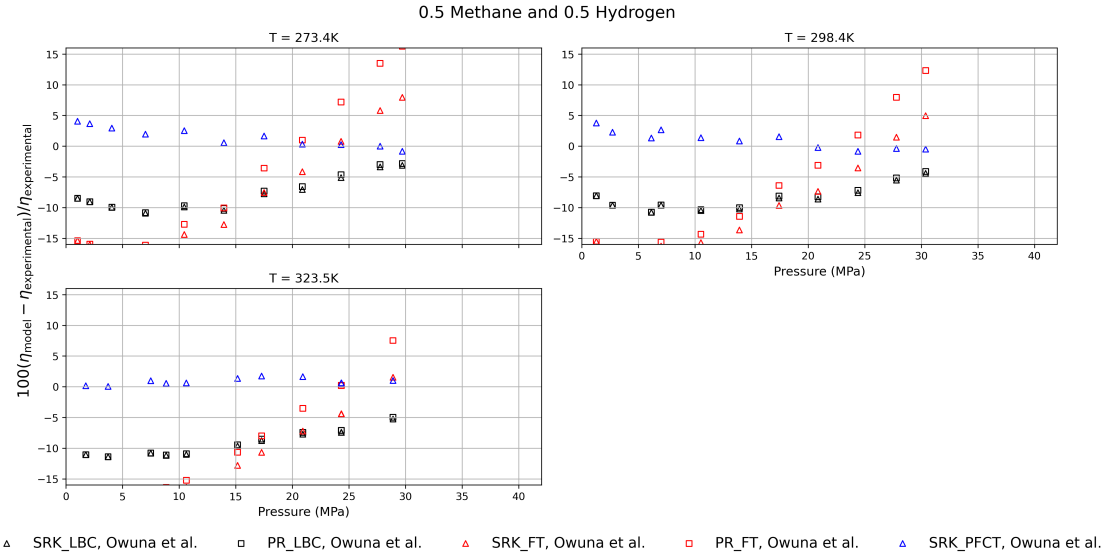

Figure 14: Percentage viscosity deviations of experimental viscosity data for composition with mole fraction 0.5 CH_4 and 0.5 H_2 of Owuna et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model.

Table 21 shows the mean and maximum percentage viscosity deviations of the experimental data and the viscosity models, for the same data points used in Figure 14. The PFCT model again shows a decrease in both AARD and max ARD compared its performance for pure methane and the mixtures with lower hydrogen mole fractions. In contrast, the other viscosity models continue to perform worse compared to their performance for the lower hydrogen mole fractions.

Vicsosity model	AARD (%)	Max ARD (%)
LBC_{srk}	8.47	11.42
LBC_{pr}	8.27	11.41
FT_{srk}	11.02	18.60
FT_{pr}	10.77	18.45
$PFCT_{srk}$	1.30	4.05

Table 21: Mean and maximum percentage viscosity deviations of the experimental viscosity data for composition with mole fraction 0.5 CH_4 and 0.5 H_2 from the same modelled viscosity values used in Figure 14.

4.2.6 Mixture: $0.1CH_4 + 0.9H_2$

Figure 15 shows the relative deviation between the experimental viscosity data for a $0.1CH_4$+$0.9H_2$ mixture and the viscosity models. The hydrogen mole fraction exceeds the 75% threshold for the PFCT model in NeqSim, preventing the generation of numerical results for this model. The plots in Figure 15 show similar characteristics to the pure hydrogen plots in Figure 11: The LBC model gives a stable result but significantly underestimates the viscosity, whereas the FT model becomes more unstable with increasing pressure.

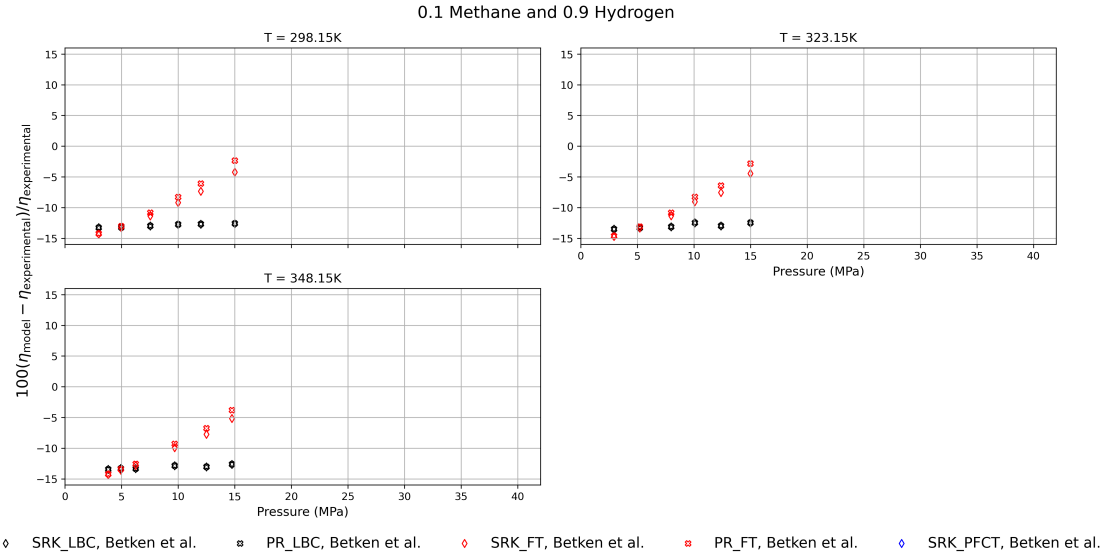

Figure 15: Percentage viscosity deviations of experimental viscosity data for composition with mole fraction 0.1 CH_4 and 0.9 H_2 of Betken et al. from modelled viscosity values calculated with the LBC-, FT-, and PFCT viscosity model.

Table 22 shows the mean and maximum percentage viscosity deviations of the experimental data and the viscosity models, for the same data points used in Figure 15. Both the AARD and the max ARD are high for both variants of the LBC and FT viscosity models.

EOS	**AARD (%)**	**Max ARD (%)**
LBC_{srk}	13.03	13.53
LBC_{pr}	13.01	13.52
FT_{srk}	10.22	14.72
FT_{pr}	9.51	14.64
$PFCT_{srk}$	-	-

Table 22: Mean and maximum percentage viscosity deviations of the experimental viscosity data for composition with mole fraction 0.1 CH_4 and 0.9 H_2 from the same modelled viscosity values used in Figure 15.

5 Discussion

5.1 Density

5.1.1 Pure Hydrogen

As expected, the GERG2008 EOS provides consistent and accurate results. This high level of accuracy comes from the fact that the EOS is fitted to experimental data, not only for typical NG-compositions, but also for those containing high fractions of hydrogen [19].

In contrast to the Leachman model, which is designed for normal hydrogen, none of the other EOSs explicitly specify which type of hydrogen that is used. It is therefore assumed that the BICs correspond to normal hydrogen, as this is the form of hydrogen under atmospheric conditions.

5.1.2 Comparison with Hernández-Gómez et al.[16]

The GERG2008 EOS was accurate for all of the three molar fractions of hydrogen. The maximum ARD of the GERG2008 EOS was above 0.1% for all three methane-hydrogen mixtures, which is higher than the accuracy promised by Kunz and Wagner. However, if the uncertainty of the measurements in the study (see Table 7) is accounted for, the results from GERG2008 can be within the 0.1% margin, thereby confirming Kunz and Wagner's promised accuracy of 0.1% for the GERG2008 EOS.

5.1.3 Comparison with Richter et al.

The GERG2008 EOS provided very accurate results for the compositions NG1 and NG2. The high accuracy can be attributed to GERG2008 being fitted to real NG compositions, not binary mixtures such as those in the study by Hernández-Gómez et al.[16].

GERG2008 has a significantly higher error for NG3 compared to NG1 and NG2. The error is higher than GERG2008's promised 0.1% margin. This result is expected, given the poorly documented uncertainty for NG3, as discussed in Section 3.1.1. Tables 12 and 13 also show that GERG2008 gives accurate results for mixtures with high molar fraction of hydrogen. Combined with the fact that GERG2008 is fitted for mixtures with high molar fractions of hydrogen, this suggests that the data for the NG3 mixture may not be sufficiently reliable to draw definitive conclusions about the performance of the GERG2008 EOS.

NG1, NG2 and NG3 (see Table 8) all contain benzene (C_6H_6), toluene (C_7H_8) and o-xylene (o-C_8H_10), compounds which the GERG2008 model is not fitted for. NeqSim handles this by substituting the compound with one of the 21 compounds supported by the GERG2008 EOS, enabling NeqSim's implementation of the GERG2008 EOS to be applied to components not included in the EOS. However, this can also be a source of error. For the three mixtures in Richter et al., the mole fractions of benzene, toluene and o-xylene are small, which minimizes the effect of the compound substitute. However, if these components had considerably larger mole fractions, GERG2008 would perhaps not be a suitable EOS.

5.1.4 Comparison with Ben Souissi et al.

GERG2008 showed a larger deviation than for the other studies, with a maximum ARD of approximately 0.4%. This exceeds GERG2008's promised 0.1% deviation margin, even when the uncertainty from the study is accounted for. However, this deviation is expected since GERG2008 is not designed for larger mole fractions of CO_2. Additionally, some of the data points from the study were close to the critical point of CO_2, which generally makes modelling less accurate. Kunz and Wagner recommends modifying GERG2008 to increase accuracy for larger mole fractions of

CO_2. Although the maximum ARD fell outside the 0.1% margin, the AARD is still within the margin.

5.1.5 Comparison with Hernández-Gómez et al.[15]

GERG2008 showed a lower accuracy for this study compared to the others, with both the AARD and the maximum ARD above the 0.1% margin. However, if the uncertainty of the measurements in the study (see Table 7) is accounted for, the results from GERG2008 can be within Kunz and Wagner's promised 0.1% accuracy.

5.1.6 Binary coefficients NeqSim vs HYSYS

As previously mentioned, there are big differences between the the BICs in NeqSim and HYSYS. The plots in the Results section show that the differences between HYSYS and NeqSim for the same EOSs are small at high temperatures and low pressures. This suggests that the contribution from the BICs is low at these conditions. However, at lower temperatures and higher pressures, the difference between NeqSim and HYSYS becomes more evident. It is impossible to conclude whether the BICs from NeqSim or HYSYS are better than an other, because the performance of the models varies across different states and mixtures.

NeqSim's SRK performed better than both implementations of the PR EOS for hydrogen rich mixtures. However, no clear trend in performance of any of the PR implementations was observed with changes in the hydrogen molar fraction. Results for pure hydrogen show that both NeqSim and HYSYS' SRK perform better than both PR implementations, which suggests that SRK should be preferred over PR when working with hydrogen-rich mixtures. With increasing molar fractions of hydrogen, the differences between the density from NeqSim's SRK and both PR implementations also increased, suggesting the BICs for hydrogen are defined differently in the different implementations of the EOSs.

Even though NeqSim's list of BICs appears flawed in comparison to HYSYS' list, the results demonstrate that this does not necessarily lead to poorer performance. As demonstrated in the mixing rule in Eq. (17), the BICs for components with higher molar fractions have bigger impact than the BICs for compounds with lower molar fractions. The majority of the mixtures in this study have methane as the main component, so it is important that the BIC for methane is properly defined for accurate results. The lists of BICs in Appendix C reveal that the components with undefined BICs in NeqSim are typically components with lower molar fractions in NG mixtures. This explains why NeqSim's SRK outperforms HYSYS's SRK, even though NeqSim's BIC list is less extensive.

5.2 Viscosity

In general, the viscosity models observed in this study performed worse in terms of AARD than the models for density, especially the GERG2008 EOS. This was expected, as viscosity models are generally not as accurate as EOSs for density prediction. Also, the available experimental data for viscosity has a drastically higher uncertainty than the experimental data for density [24, 4, 7].

The experimental data from Owuna et al. has a relative uncertainty of approximately $4 - 6.6\%$ in the lower temperature range (see Table 11), making it difficult to draw definitive conclusions about the models' performance at those conditions. At the higher end of the temperature range in the data from Owuna et al., the relative uncertainty of the experimental data is approximately $2.5 - 3\%$. This level of uncertainty makes it difficult to determine the performance of the viscosity models with high precision, but trends of how the models perform for different temperatures, pressures and compositions can still be observed.

The LBC viscosity model performs well for low mole fractions of hydrogen, and can be considered reliable for hydrogen mole fractions of 20% and less. However, for hydrogen mole fractions of 50%

and higher, the LBC model exhibits an AARD exceeding 8%. Attempts to compare the results with other sources with similar compositions and temperature-pressure ranges have been unsuccessful.

Based on the results, the FT viscosity model can be deemed unreliable for predicting viscosity when the hydrogen mole fraction surpasses 10%, as the AARD of the predicted viscosity exceeds 5%. Attempts to compare the results with other sources with similar compositions and temperature-pressure ranges have been unsuccessful.

It has been found that the PFCT viscosity model performs well in comparison to the LBC and the FT model in the temperature-pressure range for the available experimental data for methane-hydrogen mixtures collected for this study (273K-323K and 0-40MPa). Especially at high pressures. The PFCT model also performs better with increasing mole fraction of hydrogen, as opposed to the LBC and the FT model. The reason for this is unknown. The result of the PFCT model's performance has good agreement with the findings by Owuna et al., something that supports the results and shows that the viscosity model is correctly implemented in NeqSim. However, because the SRK EOS is hard-coded into the PFCT model in NeqSim, it is not possible to evaluate how a change in the EOS, and consequently a change in density, would impact the accuracy of the viscosity model. Another issue with the PFCT model is that it fails to give numeric results when the hydrogen mole fraction exceeds 75%. When the NeqSim code for PFCT is ran with higher mole fractions of hydrogen, the PFCT model returns "nan". The reason behind this has been investigated, but the time frame for this project has been too short to find a solution to the problem.

6 Conclusion

This study examines three equations of state - SRK, PR and GERG2008 - for predicting density, and three viscosity models - LBC, Friction Theory and PFCT - for predicting the viscosity of hydrogen-containing natural gas mixtures and simpler methane-hydrogen mixtures. The modelled densities and viscosities have been compared to experimental data from different sources for a variety of temperatures, pressures and compositions relevant for the natural gas value chain.

The results from the density models were consistent across data from different sources, and showed that the GERG2008 EOS outperforms the NeqSim's and HYSYS' implementations of SRK and PR for all compositions across all temperature and pressure ranges observed in this study. Kunz and Wagner's claim of under 0.1% deviation for the GERG2008 EOS holds up for the compositions GERG2008 is designed for. NeqSim uses a compound substitution method to handle compounds which are not implemented in the GERG2008 EOS, that could lead to error, but the effect of this have shown to be minimal in this work. SRK and PR overall have worse accuracy than GERG2008, and for some mixtures and states there are large deviations between NeqSim's and HYSYS' implementations of SRK and PR. NeqSim's implementation of SRK is the most accurate of the four implementations of cubic equation of states used in this work, showing increasing accuracy for increasing molar fraction of hydrogen.

For viscosity prediction, the PFCT model provided the most accurate results for compositions with higher mole fractions ($> 10\%$) of hydrogen, while the LBC model gave slightly better results for lower mole fractions of hydrogen and pure methane. However, the uncertainty in the experimental viscosity data was unsatisfactory, and the results must be interpreted with this in mind. The friction theory viscosity model had the lowest accuracy out of the three models and became unstable for higher mole fractions of hydrogen.

6.1 Further Work

To further enhance NeqSim's capabilities, several furter developments and improvements could be considered:

- **Implement the Leachman model:**
  Integrating the Leachman model into NeqSim would allow for extremely accurate calculations for hydrogen systems.
- **Expand and Update the BIC List:**
  Some of the BICs in NeqSim differ from the ones used in HYSYS, resulting in different results. Expanding the BIC list to include values for all the components, and updating the existing ones, could significantly improve the accuracy of NeqSim's models.
- **Incorporating the GERG2008 EOS for the Viscosity Models:**
  It has been shown that the choice of EOS used for modelling the density has a significant effect on the modelled viscosity. It would therefore be valuable to explore the effect of incorporating the GERG2008 EOS in the NeqSim code for the viscosity models, especially the PFCT model, to determine if it can improve the accuracy of the models. This will, however, demand more computational power.
- **Investigate PFCT Viscosity Model Issues:**
  It would also be beneficial to find out the reason why the PFCT viscosity model in NeqSim fails to give a numeric result when the mole fraction of hydrogen exceeds 75%. Whether this is a weakness in the viscosity model itself, or errors in the source code in NeqSim is yet to be found out.
- **Test Additional Viscosity Models:**
  Testing viscosity models beyond the three models available in NeqSim, for example the Simplified Residual Equation of State (SRES), which has shown promising results in the study by Owuna et al.
- **Improve User Feedback and Accessibility:**
  NeqSim could be made more user-friendly by addressing potential issues in the code. For example when the GERG2008 model is used for components that are not defined in the model, NeqSim solves this by using a substitute component, without notifying the user. Similarly, when undefined EOSs or viscosity models are called, NeqSim uses the last valid model that was called or uses a default model, instead of notifying the user that the entered model is not defined in NeqSim.

Bibliography

[1] J.O. Abe, A.P.I. Popoola, E. Ajenifuja, and O.M. Popoola. Hydrogen energy, economy and storage: Review and recommendation. *International Journal of Hydrogen Energy*, 44(29): 15072–15086, 2019. ISSN 0360-3199. doi: https://doi.org/10.1016/j.ijhydene.2019.04.068. URL https://www.sciencedirect.com/science/article/pii/S036031991931465X.

[2] Tarek Ahmed. Chapter 5 - equations of state and phase equilibria. In Tarek Ahmed, editor, *Equations of State and PVT Analysis (Second Edition)*, pages 467–597. Gulf Professional Publishing, Boston, second edition edition, 2016. ISBN 978-0-12-801570-4. doi: https://doi.org/10.1016/B978-0-12-801570-4.00005-2. URL https://www.sciencedirect.com/science/article/pii/B9780128015704000052.

[3] Mohamed A. Ben Souissi, Reiner Kleinrahm, Xiaoxian Yang, and Markus Richter. Vapor-phase (p, , t, x) behavior and virial coefficients for the binary mixture (0.05 hydrogen + 0.95 carbon dioxide) over the temperature range from (273.15 to 323.15) k with pressures up to 6mpa. *Journal of Chemical & Engineering Data*, 62(9):2973–2981, 2017. doi: 10.1021/acs.jced.7b00213. URL https://doi.org/10.1021/acs.jced.7b00213.

[4] Benjamin Betken, Anders Austegard, Francesco Finotti, Chiara Caccamo, Hans Georg Jacob Stang, Bahareh Khosravi, and Roland Span. Measurements of the viscosity of hydrogen and a (hydrogen+methane) mixture with a two-capillary viscometer. *International Journal of Thermophysics*, 45(4):60, March 2024. ISSN 1572-9567. doi: 10.1007/s10765-023-03328-6.

[5] N. Böttcher, J. Taron, O. Kolditz, R. Liedl, and C.-H. Park. Comparison of equations of state for carbon dioxide for numerical simulations. In M. P. Anderson, T. P. A. Ferré, and K. Loague, editors, *Models – Repositories of Knowledge*, volume 355 of *IAHS Publication*, pages 252–260, Wallingford, UK, 2012. IAHS Press. URL https://iahs.info/uploads/dms/15938.45-252-260-355-24-Boettcher.pdf. Proceedings of ModelCARE 2011, Leipzig, Germany, September 2011.

[6] G.E. Childs and H.J.M. Hanley. Applicability of dilute gas transport property tables to real gases. *Cryogenics*, 8(2):94–97, 1968. ISSN 0011-2275. doi: https://doi.org/10.1016/0011-2275(68)90049-0. URL https://www.sciencedirect.com/science/article/pii/0011227568900490.

[7] Sheng-Yi Chuang, Patsy S. Chappelear, and Riki Kobayashi. Viscosity of methane, hydrogen, and four mixtures of methane and hydrogen from -100.degree.c to 0.degree.c at high pressures. *Journal of Chemical & Engineering Data*, 21(4):403–411, 1976. doi: 10.1021/je60071a010. URL https://doi.org/10.1021/je60071a010.

[8] Ting Horng Chung, Mohammad Ajlan, Lloyd L. Lee, and Kenneth E. Starling. Generalized multiparameter correlation for nonpolar and polar fluid transport properties. *Industrial & Engineering Chemistry Research*, 27(4):671–679, 1988. doi: 10.1021/ie00076a024. URL https://doi.org/10.1021/ie00076a024.

[9] Ali Danesh. 4 - equations of state. In *PVT and Phase Behaviour of Petroleum Reservoir Fluids*, volume 47 of *Developments in Petroleum Science*, pages 129–166. Elsevier, 1998. doi: https://doi.org/10.1016/S0376-7361(98)80026-5. URL https://www.sciencedirect.com/science/article/pii/S0376736198800265.

[10] Norwegian Petroleum Directorate. URL https://www.norskpetroleum.no/en/production-and-exports/exports-of-oil-and-gas/?utm_source=chatgpt.com.

[11] Alexander W. Dowling, Qianwen Gao, and Lorenz T. Biegler. Equation-oriented optimization of cryogenic systems for coal oxycombustion power plants. In Mario R. Eden, John D. Siirola, and Gavin P. Towler, editors, *Proceedings of the 8th International Conference on Foundations of Computer-Aided Process Design*, volume 34 of *Computer Aided Chemical Engineering*, pages 501–506. Elsevier, 2014. doi: https://doi.org/10.1016/B978-0-444-63433-7.50068-7. URL https://www.sciencedirect.com/science/article/pii/B9780444634337500687.

[12] I. F. Ely and H.J.M. Hanley. Xi 323. *Znd. Engng Chem. Funa*, 1981.

[13] Encyclopedia Britannica. Viscosity. In *Encyclopedia Britannica*. Encyclopedia Britannica, 2024. URL https://www.britannica.com/science/viscosity.

[14] H.J.M. Hanley, R.D. McCarty, , and W.M. Haynes. The viscosity and thermal conductivity coefficients for dense gaseous and liquid methane. *Cryogenics 15. 413*, 1975.

[15] Roberto Hernández-Gómez, Dirk Tuma, Daniel Lozano-Martín, and César R. Chamorro. Accurate experimental (p, , t) data of natural gas mixtures for the assessment of reference equations of state when dealing with hydrogen-enriched natural gas. *International Journal of Hydrogen Energy*, 43(49):21983–21998, 2018. ISSN 0360-3199. doi: https://doi.org/10.1016/j.ijhydene.2018.10.027. URL https://www.sciencedirect.com/science/article/pii/S036031991833204X.

[16] Roberto Hernández-Gómez, Dirk Tuma, Eduardo Perez, and Cesar R. Chamorro. Accurate experimental (p, , and t) data for the introduction of hydrogen into the natural gas grid (ii): Thermodynamic characterization of the methane–hydrogen binary system from 240 to 350 k and pressures up to 20 mpa. *Journal of Chemical & Engineering Data*, 63(5):1613–1630, 2018. doi: 10.1021/acs.jced.7b01125. URL https://doi.org/10.1021/acs.jced.7b01125.

[17] JCGM. Evaluation of measurement data — guide to the expression of uncertainty in measurement. standard, JCGM, 2008.

[18] Georgios M. Kontogeorgis, Xiaodong Liang, Alay Arya, and Ioannis Tsivintzelis. Equations of state in three centuries. are we closer to arriving to a single model for all applications? *Chemical Engineering Science: X*, 7:100060, 2020. ISSN 2590-1400. doi: https://doi.org/10.1016/j.cesx.2020.100060. URL https://www.sciencedirect.com/science/article/pii/S259014002030006X.

[19] O. Kunz and W. Wagner. The gerg-2008 wide-range equation of state for natural gases and other mixtures: An expansion of gerg-2004. *Journal of Chemical & Engineering Data*, 57(11): 3032–3091, 2012. doi: 10.1021/je300655b. URL https://doi.org/10.1021/je300655b.

[20] J. W. Leachman, R. T Jacobsen, S. G. Penoncello, and E. W. Lemmon. Fundamental equations of state for parahydrogen, normal hydrogen, and orthohydrogen. *Journal of Physical and Chemical Reference Data*, 38(3):721–748, September 2009. ISSN 0047-2689. doi: 10.1063/1.3160306.

[21] Eric W. Lemmon and Richard T Jacobsen. Equations of State for Mixtures of R-32, R-125, R-134a, R-143a, and R-152a. *Journal of Physical and Chemical Reference Data*, 33(2):593–620, 06 2004. ISSN 0047-2689. doi: 10.1063/1.1649997. URL https://doi.org/10.1063/1.1649997.

[22] John Lohrenz, Bruce G. Bray, and Charles R. Clark. Calculating viscosities of reservoir fluids from their compositions. *Journal of Petroleum Technology*, 16:1171–1176, 1964. URL https://api.semanticscholar.org/CorpusID:108518213.

[23] Daniel Lozano-Martín, Alejandro Moreau, and César R. Chamorro. Thermophysical properties of hydrogen mixtures relevant for the development of the hydrogen economy: Review of available experimental data and thermodynamic models. *Renewable Energy*, 198:1398–1429, 2022. ISSN 0960-1481. doi: https://doi.org/10.1016/j.renene.2022.08.096. URL https://www.sciencedirect.com/science/article/pii/S096014812201271X.

[24] Friday Junior Owuna, Antonin Chapoy, Pezhman Ahmadi, and Rod Burgass. Viscosity of hydrogen and methane blends: Experimental and modelling investigations. *International Journal of Thermophysics*, 45(8):119, July 2024. ISSN 1572-9567. doi: 10.1007/s10765-024-03394-4.

[25] Karen Schou Pedersen and Aage Fredenslund. An improved corresponding states model for the prediction of oil and gas viscosities and thermal conductivities. *Chemical Engineering Science*, 42(1):182–186, 1987. ISSN 0009-2509. doi: https://doi.org/10.1016/0009-2509(87)80225-7. URL https://www.sciencedirect.com/science/article/pii/0009250987802257.

[26] Karen Schou Pedersen, Aage Fredenslund, Peter L. Christensen, and Per Thomassen. Viscosity of crude oils. *Chemical Engineering Science*, 39(6):1011–1016, 1984. ISSN 0009-2509. doi: https://doi.org/10.1016/0009-2509(84)87009-8. URL https://www.sciencedirect.com/science/article/pii/0009250984870098.

[27] Ding-Yu Peng and Donald B Robinson. A new two-constant equation of state. *Industrial & Engineering Chemistry Fundamentals*, 15(1):59–64, 1976.

[28] Petroleum Safety Authority Norway. Investigation of undesirable incident on troll c on 24 october 2021 – revised. Technical report, Petroleum Safety Authority Norway, 2022. URL https://www.havtil.no/contentassets/3ba6433593c14974ad8c4c1f13b08792/2021_1717-rapport-eng-gransking-equinor-troll-c-rev_final.pdf. Accessed: 2024-11-18.

[29] PubChem. Isopentane, . URL https://pubchem.ncbi.nlm.nih.gov/compound/6556.

[30] PubChem. Neopentane, . URL https://pubchem.ncbi.nlm.nih.gov/compound/10041.

[31] PubChem. Pentane, . URL https://pubchem.ncbi.nlm.nih.gov/compound/8003.

[32] Sergio E. Quiñones-Cisneros, Claus K. ZÃ©berg-Mikkelsen, and Erling H. Stenby. The friction theory (f-theory) for viscosity modeling. *Fluid Phase Equilibria*, 169(2):249–276, 2000. ISSN 0378-3812. doi: https://doi.org/10.1016/S0378-3812(00)00310-1. URL https://www.sciencedirect.com/science/article/pii/S0378381200003101.

[33] Sergio E Quiñones-Cisneros, Claus K ZÃ©berg-Mikkelsen, and Erling H Stenby. The friction theory for viscosity modeling: extension to crude oil systems. *Chemical Engineering Science*, 56(24):7007–7015, 2001. ISSN 0009-2509. doi: https://doi.org/10.1016/S0009-2509(01)00335-9. URL https://www.sciencedirect.com/science/article/pii/S0009250901003359. Festschrift in honour of Professor T.-M. Guo.

[34] Sergio E Quiñones-Cisneros, Claus K Zéberg-Mikkelsen, and Erling H Stenby. One parameter friction theory models for viscosity. *Fluid Phase Equilibria*, 178(1):1–16, 2001. ISSN 0378-3812. doi: https://doi.org/10.1016/S0378-3812(00)00474-X. URL https://www.sciencedirect.com/science/article/pii/S037838120000474X.

[35] Otto Redlich and Joseph NS Kwong. On the thermodynamics of solutions. v. an equation of state. fugacities of gaseous solutions. *Chemical reviews*, 44(1):233–244, 1949.

[36] Markus Richter, Mohamed A. Ben Souissi, Roland Span, and Peter Schley. Accurate (p, , t, x) measurements of hydrogen-enriched natural-gas mixtures at t = (273.15, 283.15, and 293.15) k with pressures up to 8mpa. *Journal of Chemical & Engineering Data*, 59(6):2021–2029, 2014. doi: 10.1021/je500181v. URL https://doi.org/10.1021/je500181v.

[37] Dr. B. Schuchardt, Dipl.-Georg. G. Krause, and Dr. H.-G. Kulp. Environmental impact assessment europipe ii in germany: Offshore and onshore section. Technical report, BIOCONSULT SCHUCHARDT SCHOLLE, 1998. URL file:///C:/Users/victo/Downloads/Statoil-Environmental%20Impact%20Assessment%20Europipe%20II%20July%201998%20(3).pdf.

[38] SciPy. URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fsolve.html.

[39] Anders Dalberg Sergio E. Quiñones-Cisneros and Erling H. Stenby. Pvt characterization and viscosity modeling and prediction of crude oils. *Petroleum Science and Technology*, 22(9-10):1309–1325, 2004. doi: 10.1081/LFT-200034092. URL https://doi.org/10.1081/LFT-200034092.

[40] K Sikorski. Optimal solution of nonlinear equations. *Journal of Complexity*, 1(2):197–209, 1985. ISSN 0885-064X. doi: https://doi.org/10.1016/0885-064X(85)90011-1. URL https://www.sciencedirect.com/science/article/pii/0885064X85900111.

[41] Richard Smith, Hiroshi Inomata, and Cor Peters. Chapter 6 - equations of state and formulations for mixtures. In Richard Smith, Hiroshi Inomata, and Cor Peters, editors, *Introduction to Supercritical Fluids*, volume 4 of *Supercritical Fluid Science and Technology*, pages 333–480. Elsevier, 2013. doi: https://doi.org/10.1016/B978-0-444-52215-3.00006-4. URL https://www.sciencedirect.com/science/article/pii/B9780444522153000064.

[42] Giorgio Soave. Equilibrium constants from a modified redlich-kwong equation of state. *Chemical engineering science*, 27(6):1197–1203, 1972.

[43] Roland Span, Eric W. Lemmon, Richard T Jacobsen, Wolfgang Wagner, and Akimichi Yokozeki. A reference equation of state for the thermodynamic properties of nitrogen for temperatures from 63.151 to 1000 k and pressures to 2200 mpa. *Journal of Physical and Chemical Reference Data*, 29(6):1361–1433, November 2000. ISSN 0047-2689. doi: 10.1063/1.1349047.

[44] Xuguang Sun and Lorenzo M. Smith. External heating closed-volume thermally activated tube forming: A fundamentally new approach for hydroforming thick-walled tubes. *Journal of Manufacturing Processes*, 12(2):63–66, 2010. ISSN 1526-6125. doi: https://doi.org/10.1016/j.jmapro.2010.01.005. URL https://www.sciencedirect.com/science/article/pii/S1526612510000162.

[45] David A. Tillman, Dao N.B. Duong, and N. Stanley Harding. Chapter 7 - modeling and fuel blending. In David A. Tillman, Dao N.B. Duong, and N. Stanley Harding, editors, *Solid Fuel Blending*, pages 271–293. Butterworth-Heinemann, Boston, 2012. ISBN 978-0-12-380932-2. doi: https://doi.org/10.1016/B978-0-12-380932-2.00007-6. URL https://www.sciencedirect.com/science/article/pii/B9780123809322000076.

[46] Johannes D Van der Waals. The equation of state for gases and liquids. *Nobel lectures in Physics*, 1:254–265, 1910.

[47] Johannes Diderik Van Der Waals and John Shipley Rowlinson. *On the continuity of the gaseous and liquid states*. Courier Corporation, 2004.

Appendix A

Original Project Description

Department:
EPT

Title:
Viscosity of Hydrogen and Natural Gas Mixtures: Comparing Experimental Data and Model Calculations

Supervisor:
Solbraa, Even

Co supervisor:

Firm:
Equinor

Description:
As the energy sector moves towards more sustainable sources, the integration of hydrogen into existing natural gas infrastructures is a promising approach. However, understanding the physical properties of these mixtures is crucial. This thesis will contribute to the field by evaluating and improving models for calculation of gas viscosity of hydrogen and natural gas mixtures.

A key focus will be on implementing and testing models within the NeqSim software framework. This research is critical for advancing the use of hydrogen in natural gas systems, with implications for energy efficiency, safety, and environmental impact.

Experimental work at the Equinor Research Centre to gather data on the viscosity and density of hydrogen and natural gas mixtures under various conditions will be considered as part of the thesis work.

Requirements:
Knowledge and interest in Python and Java programming is an advantage.

Appendix B

Python Code for Density

B.1 Code for Pure Hydrogen

Code written in python for calculating the density data for pure hydrogen. The code are split up to different segments; Leachman, SRK and PR, GERG2008, plotting and statistics.

```
import numpy as np
import pandas as pd
from neqsim.thermo import TPflash, fluid
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import os



##-------------------Leachman -----------------------

class HydrogenEOS:
    def __init__(self, M, rho_c_mole, rho_c, T_c, N_i, t_i, d_i, p_i,
        ↪ phi_i, beta_i, gamma_i, D_i, l, m, n):
        # Assign constants
        self.M = M
        self.R = 8.314472 / M  # kPam^3/kgK
        self.rho_c_mole = rho_c_mole  # kmol/m^3
        self.rho_c = rho_c  # kg/m^3
        self.T_c = T_c

        # Assign coefficients for the equations
        self.N_i = N_i
        self.t_i = t_i
        self.d_i = d_i
        self.p_i = p_i
        self.phi_i = phi_i
        self.beta_i = beta_i
        self.gamma_i = gamma_i
        self.D_i = D_i

        # Set ranges for summation indices
        self.l = l
        self.m = m
        self.n = n

    # Define the function for alpha_r
    def alpha_r(self, delta, tau):
        alpha_r_value = 0
        for i in range(self.l):
            alpha_r_value += self.N_i[i] * (delta ** self.d_i[i]) *
                ↪ (tau ** self.t_i[i])
        for i in range(self.l, self.m):
            alpha_r_value += self.N_i[i] * (delta ** self.d_i[i]) *
                ↪ (tau ** self.t_i[i]) * np.exp(-delta ** self.p_i[i])
        for i in range(self.m, self.n):
            alpha_r_value += self.N_i[i] * (delta ** self.d_i[i]) *
                ↪ (tau ** self.t_i[i]) * \
                            np.exp(self.phi_i[i] * (delta -
                                ↪ self.D_i[i]) ** 2 + self.beta_i[i] *
                                ↪ (tau - self.gamma_i[i]) ** 2)
```

```
        return alpha_r_value

    # Define a function that returns the difference between calculated and target pressure
    def pressure_difference(self, rho, T, P_target):
        tau = self.T_c / T
        delta = rho / self.rho_c
        alpha_res = self.alpha_r(delta, tau)

        # Compute the first derivative of alpha_r with respect to delta
        daddelta = (self.alpha_r(delta + 1e-8, tau) - self.alpha_r(delta - 1e-8, tau)) / (2e-8)
        P_calculated = self.R * T * rho * (1 + delta * daddelta)

        return P_calculated - P_target

    # Bisection method to find density
    def calculate_density_bisection(self, T, P_target, a=0.001, b=100, tol=1e-6, max_iter=100):
        fa = self.pressure_difference(a, T, P_target)
        fb = self.pressure_difference(b, T, P_target)

        # Check if signs of fa and fb are opposite
        if fa * fb > 0:
            raise ValueError("Bisection method requires a sign change over the interval.")

        for iteration in range(max_iter):
            c = (a + b) / 2
            fc = self.pressure_difference(c, T, P_target)

            if abs(fc) < tol or (b - a) / 2 < tol:  # Convergence criteria
                return c  # Solution found

            if fa * fc < 0:
                b = c  # Root is in left half
            else:
                a = c  # Root is in right half
                fa = fc

        raise ValueError("Bisection method did not converge after max iterations")


# Define values for normal hydrogen
normal_hydrogen_params = {
    'M': 2.01588,  # kg/kmol
    'rho_c_mole': 15.508,  # kmol/m^3
    'rho_c': 31.262267039999998,
    'T_c': 33.145,  # K
    'N_i': np.array([-6.93643, 0.01, 2.1101, 4.52059, 0.732564, -1.34086, 0.130985, -0.777414, 0.351944, -0.0211716, 0.0226312, 0.032187, -0.0231752, 0.0557346]),
    't_i': np.array([0.6844, 1, 0.989, 0.489, 0.803, 1.1444, 1.409, 1.754, 1.311, 4.187, 5.646, 0.791, 7.249, 2.986]),
    'd_i': np.array([1, 4, 1, 1, 2, 2, 3, 1, 3, 2, 1, 3, 1, 1]),
    'p_i': np.array([0, 0, 0, 0, 0, 0, 0, 1, 1]),
    'phi_i': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, -1.685, -0.489, -0.103, -2.506, -1.607]),
    'beta_i': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, -0.171, -0.2245, -0.1304, -0.2785, -0.3967]),
```

```
    'gamma_i': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0.7164, 1.3444,
        ↪ 1.4517, 0.7204, 1.5445]),
    'D_i': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1.506, 0.156, 1.736,
        ↪ 0.67, 1.662]),
    'l': 7,
    'm': 9,
    'n': 14
}



# Initialize normal hydrogen
normal_hydrogen = HydrogenEOS(**normal_hydrogen_params)
# Define pressure and temperature values
pressure_values = np.arange(1, 351) * 100  # Pressure in kPa (1 bar to
    ↪ 100 bar)
T_values = np.arange(240, 401, 40)  # Temperature values from 240 K to
    ↪ 400 K in steps of 40 K

# Prepare a list to store the data in the desired format
data = []

# Calculate densities for each temperature and pressure using Leachman
    ↪ EOS
for T in T_values:
    for P in pressure_values:
        try:
            density = normal_hydrogen.calculate_density_bisection(T, P)
            data.append([T, P * 1e-3, density])  # Convert P from kPa
                ↪ to MPa
        except ValueError:
            data.append([T, P * 1e-3, None])  # Use None if calculation
                ↪ fails

# Convert data to DataFrame with the desired column order
df = pd.DataFrame(data, columns=["Temperature (K)", "Pressure (MPa)",
    ↪ "Density (kg/m^3)"])

# Define the file path
file_path = r"C:\Users\victo\OneDrive - NTNU\5.
    ↪  r \Fordypningsprosjekt\Python\Pure_H2\pure_H2_data.xlsx"

# Write DataFrame to Excel file in a new sheet named "Leachman"
with pd.ExcelWriter(file_path, engine='openpyxl', mode='a',
    ↪ if_sheet_exists="replace") as writer:
    df.to_excel(writer, sheet_name="Leachman", index=False)

print(f"Data successfully saved to {file_path} in the 'Leachman' sheet")



##-------------------- SRK and PR----------------------

def generate_data(fluid_type):
    data = []
    fluid1 = fluid(fluid_type)
    fluid1.addComponent("hydrogen", 1.0, "mol/sec")

    # Define pressure and temperature ranges as specified
    temperatures = range(240, 401, 40)  # 240 K to 400 K in 40 K
        ↪ intervals
    pressures = [p * 0.1 for p in range(1, 351)]  # 0.1 MPa to 35 MPa
        ↪ in 0.1 MPa intervals
```

```
148
149     for T in temperatures:
150         fluid1.setTemperature(T, "K")  # Set temperature in Kelvin
151         for p in pressures:
152             fluid1.setPressure(p, "MPa")  # Set pressure in MPa
153             fluid1.setMixingRule("classic")
154             TPflash(fluid1)  # Perform TP flash calculation
155             density = fluid1.getDensity()  # Get density result
156             data.append([T, p, density])
157
158     # Return DataFrame with temperature as the first column
159     return pd.DataFrame(data, columns=["Temperature (K)", "Pressure
        ↪ (MPa)", "Density (kg/m^3)"])
160
161 # Generate data for Peng-Robinson (PR) and Soave-Redlich-Kwong (SRK) EOS
162 df_pr = generate_data("pr")
163 df_srk = generate_data("srk")
164
165 # Write data to different sheets in the existing Excel file
166 file_path = r"C:\Users\victo\OneDrive - NTNU\5.
    ↪  r \Fordypningsprosjekt\Python\Pure_H2\pure_H2_data_classic.xlsx"
167
168 with pd.ExcelWriter(file_path, engine="openpyxl", mode="a",
    ↪ if_sheet_exists="replace") as writer:
169     df_pr.to_excel(writer, sheet_name="PR_NeqSim", index=False)
170     df_srk.to_excel(writer, sheet_name="SRK_NeqSim", index=False)
171
172 print("Data successfully exported to Pure_H2_data.xlsx")
173
174
175 ## ---------------------GERG2008-----------------
176 R = 8.314472
177
178 class Fluid:
179     def __init__(self, name, n_oi, d_oi, t_oi, c_oi, K_poli, K_expi,
        ↪ T_c, rho_c, M):
180         self.name = name
181         self.n_oi = n_oi
182         self.d_oi = d_oi
183         self.t_oi = t_oi
184         self.c_oi = c_oi
185         self.K_poli = K_poli
186         self.K_expi = K_expi
187         self.T_c = T_c
188         self.rho_c = rho_c
189         self.M = M  # Molar mass in kg/kmol
190     #alpha residual
191     def alpha_r(self, delta, tau):
192         alphar_oi = 0
193         # Polynomial terms
194         for k in range(self.K_poli):
195             alphar_oi += self.n_oi[k] * (delta**self.d_oi[k]) *
                ↪ (tau**self.t_oi[k])
196
197         # Exponential terms
198         for k in range(self.K_poli, self.K_poli + self.K_expi):
199             alphar_oi += self.n_oi[k] * (delta**self.d_oi[k]) *
                ↪ (tau**self.t_oi[k]) * np.exp(-delta**self.c_oi[k])
200
201         return alphar_oi
202
203     #Pressure difference function to aqquire Pressure difference
        ↪ between the target pressure and the calculated
```

```
204     def pressure_difference(self, rho, T, P_target):
205         R = 8.314472  # kJ/kmolK
206         tau = self.T_c / T
207         densities = []
208         pressures = []
209         delta_values = []
210         dadd = []
211 
212         delta = rho / self.rho_c
213             #delta_values.append(delta)
214         daddelta = (self.alpha_r(delta + 1e-8, tau) -
                ↪ self.alpha_r(delta - 1e-8, tau)) / (2e-8)
215             #dadd.append(daddelta)
216         P_calculated = rho * R * T * (1 + delta * daddelta)
217 
218         return P_calculated - P_target
219 
220 
221     #Uses the Bisection Method to find the density for a given
            ↪ temperature and target pressure
222     def calculate_density_bisection(self, T, P_target, a=0.01, b=100,
            ↪ tol=1e-6, max_iter=100):
223         fa = self.pressure_difference(a, T, P_target)
224         fb = self.pressure_difference(b, T, P_target)
225 
226         # Check if the signs of the function values at a and b are
                ↪ opposite
227         if fa * fb > 0:
228             raise ValueError("Bisection method requires a sign change
                    ↪ over the interval.")
229 
230         for iteration in range(max_iter):
231             c = (a + b) / 2  # Midpoint
232             fc = self.pressure_difference(c, T, P_target)
233 
234             if abs(fc) < tol or (b - a) / 2 < tol:  # Convergence
                    ↪ criteria
235                 return c * self.M  # Convert to mass density
236 
237             if fa * fc < 0:
238                 b = c  # Root is in the left half
239             else:
240                 a = c  # Root is in the right half
241                 fa = fc  # Update fa to be the function value at c
242 
243         raise ValueError("Bisection method did not converge after max
                ↪ iterations")
244 
245 
246 
247 # Methane data
248 methane = Fluid(
249     name="Methane",
250     n_oi=[
251         0.57335704239162, -1.676068752373, 0.23405291834916,
                ↪ -0.21947376343441,
252         0.016369201404128, 0.01500440638928, 0.098990489492918,
                ↪ 0.58382770929055,
253         -0.7478686756039, 0.30033302857974, 0.20985543806568,
                ↪ -0.018590151133061,
254         -0.15782558339049, 0.12716735220791, -0.032019743894346,
                ↪ -0.068049729364536,
```

```
        0.024291412853736, 0.0051440451639444, -0.019084949733532,
            ↪ 0.0055229677241291,
        -0.0044197392976085, 0.040061416708429, -0.033752085907575,
            ↪ -0.0025127658213357
    ],
    d_oi=[
        1, 1, 2, 2, 4, 4, 1, 1, 1, 2, 3, 6, 2, 3, 3, 4, 4, 2, 3, 4, 5,
            ↪ 6, 6, 7
    ],
    t_oi=[
        0.125, 1.125, 0.375, 1.125, 0.625, 1.5, 0.625, 2.625, 2.75,
            ↪ 2.125, 2.0,
        1.75, 4.5, 4.75, 5.0, 4.0, 4.5, 7.5, 14.0, 11.5, 26.0, 28.0,
            ↪ 30.0, 16.0
    ],
    c_oi=[
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 6,
            ↪ 6, 6, 6
    ],
    K_poli=6,
    K_expi=18,
    T_c=190.564,  # Critical temperature for methane [K]
    rho_c=10.139342719,  # Critical density for methane [kmol/m3]
    M=16.04246  # Molar mass for methane [kg/kmol]
)

# CO2 data
CO2 = Fluid(
    name="CO2",
    n_oi=[
        0.52646564804653,   # noik[3][1]
        -1.4995725042592,   # noik[3][2]
        0.27329786733782,   # noik[3][3]
        0.12949500022786,   # noik[3][4]
        0.15404088341841,   # noik[3][5]
        -0.58186950946814,  # noik[3][6]
        -0.18022494838296,  # noik[3][7]
        -0.095389904072812, # noik[3][8]
        -8.0486819317679E-03,  # noik[3][9]
        -0.03554775127309,  # noik[3][10]
        -0.28079014882405,  # noik[3][11]
        -0.082435890081677, # noik[3][12]
        0.010832427979006,  # noik[3][13]
        -6.7073993161097E-03,  # noik[3][14]
        -4.6827907600524E-03,  # noik[3][15]
        -0.028359911832177, # noik[3][16]
        0.019500174744098,  # noik[3][17]
        -0.21609137507166,  # noik[3][18]
        0.43772794926972,   # noik[3][19]
        -0.22130790113593,  # noik[3][20]
        0.015190189957331,  # noik[3][21]
        -0.0153809489533    # noik[3][22]
    ],
    d_oi=[
        1,  # doik[3][1]
        1,  # doik[3][2]
        2,  # doik[3][3]
        3,  # doik[3][4]
        3,  # doik[3][5]
        3,  # doik[3][6]
        4,  # doik[3][7]
        5,  # doik[3][8]
        6,  # doik[3][9]
```

```
312         6,  # doik[3][10]
313         1,  # doik[3][11]
314         4,  # doik[3][12]
315         1,  # doik[3][13]
316         1,  # doik[3][14]
317         3,  # doik[3][15]
318         3,  # doik[3][16]
319         4,  # doik[3][17]
320         5,  # doik[3][18]
321         5,  # doik[3][19]
322         5,  # doik[3][20]
323         5,  # doik[3][21]
324         5   # doik[3][22]
325     ],
326     t_oi=[
327         0,       # toik[3][1]
328         1.25,    # toik[3][2]
329         1.625,   # toik[3][3]
330         0.375,   # toik[3][4]
331         0.375,   # toik[3][5]
332         1.375,   # toik[3][6]
333         1.125,   # toik[3][7]
334         1.375,   # toik[3][8]
335         0.125,   # toik[3][9]
336         1.625,   # toik[3][10]
337         3.75,    # toik[3][11]
338         3.5,     # toik[3][12]
339         7.5,     # toik[3][13]
340         8,       # toik[3][14]
341         6,       # toik[3][15]
342         16,      # toik[3][16]
343         11,      # toik[3][17]
344         24,      # toik[3][18]
345         26,      # toik[3][19]
346         28,      # toik[3][20]
347         24,      # toik[3][21]
348         26       # toik[3][22]
349     ],
350     c_oi=[
351         0,  # coik[3][1]
352         0,  # coik[3][2]
353         0,  # coik[3][3]
354         0,  # coik[3][4]
355         1,  # coik[3][5]
356         1,  # coik[3][6]
357         1,  # coik[3][7]
358         1,  # coik[3][8]
359         1,  # coik[3][9]
360         1,  # coik[3][10]
361         2,  # coik[3][11]
362         2,  # coik[3][12]
363         3,  # coik[3][13]
364         3,  # coik[3][14]
365         3,  # coik[3][15]
366         3,  # coik[3][16]
367         3,  # coik[3][17]
368         5,  # coik[3][18]
369         5,  # coik[3][19]
370         5,  # coik[3][20]
371         6,  # coik[3][21]
372         6   # coik[3][22]
373     ],
374     K_poli=4,
```

```
    K_expi=18,
    T_c=304.1282,  # Critical temperature for CO2 [K]
    rho_c=10.624978698,  # Critical density for CO2 [kmol/m3]
    M=44.0095  # Molar mass for CO2 [kg/kmol]

)

# Hydrogen data
hydrogen = Fluid(
    name="Hydrogen",
    n_oi=[
        5.3579928451252,     # noik[15][1]
        -6.2050252530595,    # noik[15][2]
        0.13830241327086,    # noik[15][3]
        -0.071397954896129, # noik[15][4]
        0.015474053959733,   # noik[15][5]
        -0.14976806405771,   # noik[15][6]
        -0.026368723988451, # noik[15][7]
        0.056681303156066,   # noik[15][8]
        -0.060063958030436, # noik[15][9]
        -0.45043942027132,   # noik[15][10]
        0.424788402445,      # noik[15][11]
        -0.021997640827139, # noik[15][12]
        -0.01049952137453,   # noik[15][13]
        -2.8955902866816E-03 # noik[15][14]
    ],
    d_oi=[
        1,   # doik[15][1]
        1,   # doik[15][2]
        2,   # doik[15][3]
        2,   # doik[15][4]
        4,   # doik[15][5]
        1,   # doik[15][6]
        5,   # doik[15][7]
        5,   # doik[15][8]
        5,   # doik[15][9]
        1,   # doik[15][10]
        1,   # doik[15][11]
        2,   # doik[15][12]
        5,   # doik[15][13]
        1    # doik[15][14]
    ],
    t_oi=[
        0.5,     # toik[15][1]
        0.625,   # toik[15][2]
        0.375,   # toik[15][3]
        0.625,   # toik[15][4]
        1.125,   # toik[15][5]
        2.625,   # toik[15][6]
        0,       # toik[15][7]
        0.25,    # toik[15][8]
        1.375,   # toik[15][9]
        4,       # toik[15][10]
        4.25,    # toik[15][11]
        5,       # toik[15][12]
        8,       # toik[15][13]
        8        # toik[15][14]
    ],
    c_oi=[
        0,   # coik[15][1]
        0,   # coik[15][2]
        0,   # coik[15][3]
        0,   # coik[15][4]
```

```
        0,  # coik[15][5]
        1,  # coik[15][6]
        1,  # coik[15][7]
        1,  # coik[15][8]
        1,  # coik[15][9]
        2,  # coik[15][10]
        2,  # coik[15][11]
        3,  # coik[15][12]
        3,  # coik[15][13]
        5   # coik[15][14]
    ],
    K_poli=5,
    K_expi=9,
    T_c=33.19,  # Critical temperature for hydrogen [K]
    rho_c=14.94,  # Critical density for hydrogen [kmol/m3]
    M=2.01588  # Molar mass for hydrogen [kg/kmol]
)

# Define pressure and temperature values
pressure_values = np.arange(1, 351) * 100  # Pressure in kPa (1 bar to
    ↪ 100 bar)
T_values = np.arange(240, 401, 40)  # Temperature values from 240 K to
    ↪ 400 K in steps of 40 K

# Prepare a list to store the data in the desired format
data = []

# Loop to calculate densities and store in data list
for T in T_values:
    for P in pressure_values:
        density = hydrogen.calculate_density_bisection(T, P)
        data.append([T, P * 1e-3, density])  # Convert P from kPa to
            ↪ MPa for storage

# Convert data to DataFrame with the desired column order
df = pd.DataFrame(data, columns=["Temperature (K)", "Pressure (MPa)",
    ↪ "Density (kg/m^3)"])

# Define the file path
file_path = r"C:\Users\victo\OneDrive - NTNU\5.
    ↪  r \Fordypningsprosjekt\Python\Pure_H2\pure_H2_data.xlsx"

# Write DataFrame to Excel file in a new sheet named "GERG"
with pd.ExcelWriter(file_path, engine='openpyxl', mode='a',
    ↪ if_sheet_exists="replace") as writer:
    df.to_excel(writer, sheet_name="GERG", index=False)

print(f"Data successfully saved to {file_path} in the 'GERG' sheet")


#---------- Plotting and statistics for the Pure hydrogen data
    ↪ ---------------------------

def plot_relative_density_difference_grouped(file_path):
    # Load the Excel file
    excel_data = pd.ExcelFile(file_path)

    # Load the Leachman sheet as the base case
    leachman_df = excel_data.parse("Leachman")

    # Set Leachman data as the base case, indexed by temperature and
        ↪ pressure
```

```
    leachman_data = leachman_df.set_index(["Temperature (K)", "Pressure (MPa)"])["Density (kg/m^3)"]

    # Dictionary to store relative differences for each sheet
    relative_diff_data = {}

    # Define marker styles for each sheet
    markers = ['s', 'o', 'o', 'x', '^', 'D']  # square, dot, cross, triangle, plus

    # Iterate through each sheet (excluding Leachman) to calculate relative differences
    for sheet_index, sheet_name in enumerate(excel_data.sheet_names):
        if sheet_name == "Leachman":
            continue

        # Load the data for the current sheet
        df = excel_data.parse(sheet_name)

        # Filter to include only rows where Pressure (MPa) is approximately an integer
        df = df[df["Pressure (MPa)"].round() == df["Pressure (MPa)"]]

        # Merge with Leachman data on temperature and pressure
        merged_df = df.merge(
            leachman_df,
            on=["Temperature (K)", "Pressure (MPa)"],
            suffixes=('', '_Leachman')
        )

        # Calculate relative difference in density
        merged_df["Relative Difference (%)"] = (
            100 * (merged_df["Density (kg/m^3)"] - merged_df["Density (kg/m^3)_Leachman"])
            / merged_df["Density (kg/m^3)_Leachman"]
        )

        # Store the relative differences grouped by temperature
        for temperature, group in merged_df.groupby("Temperature (K)"):
            if temperature not in relative_diff_data:
                relative_diff_data[temperature] = []
            # Assign the marker based on the index
            marker = markers[sheet_index % len(markers)]
            relative_diff_data[temperature].append((group["Pressure (MPa)"], group["Relative Difference (%)"], sheet_name, marker))

    # Create the grid layout with subplot2grid
    fig = plt.figure(figsize=(20, 8))

    # Create specific subplot locations
    axes = [
        plt.subplot2grid((2, 9), (0, 1), colspan= 2),
        plt.subplot2grid((2, 9), (0, 3), colspan= 2),
        plt.subplot2grid((2, 9), (0, 5), colspan= 2),
        plt.subplot2grid((2, 9), (1, 2), colspan= 2),
        plt.subplot2grid((2, 9), (1, 4), colspan= 2)
    ]

    handles = []
    labels = []

    # Plot data on the respective axes
```

```
548     for ax, (temperature, datasets) in zip(axes,
            ↪ relative_diff_data.items()):
549         for pressure, rel_diff, sheet_name, marker in datasets:
550             # Scatter plot data for each sheet at the current isotherm
551             if marker in ['o', 's', '^', 'D']:
552                 scatter = ax.scatter(pressure, rel_diff, marker=marker,
                        ↪ label=sheet_name, color="black", s=30, facecolors
                        ↪ = 'none')
553             else:
554                 scatter = ax.scatter(pressure, rel_diff, marker=marker,
                        ↪ label=sheet_name, color="black", s=30)
555 
556             # Collect legend handles and labels
557             if sheet_name not in labels:
558                 labels.append(sheet_name)
559                 handles.append(scatter)
560 
561         # Annotate the temperature above each plot
562         ax.set_title(f"T = {temperature}K", fontsize = 12)
563         ax.grid()
564         ax.set_ylim(-1, 7)
565         ax.set_xlabel("Pressure (MPa)")
566         #ax.set_ylabel(r'Relative Difference (%)')
567 
568     # Add a global legend
569     # Add a single legend outside the plots
570     fig.legend(handles, labels, loc='upper center',
            ↪ bbox_to_anchor=(0.45, 0), ncol=5, frameon=False, fontsize =
            ↪ 14)
571     fig.text(0.09, 0.5, r'$100 \left(\rho_{\mathrm{model}} -
            ↪ \rho_{\mathrm{Leachman}}\right)/\rho_{\mathrm{Leachman}}$',
572              ha='center', va='center', rotation='vertical', fontsize =
                     ↪ 14)
573     # Save the grouped plot
574     save_path = os.path.join(
575         r"C:\Users\victo\OneDrive - NTNU\5.
                ↪  r \Fordypningsprosjekt\Python\Pure_H2",
576         "Grouped_Pure_H2_relative_to_Leachman.png"
577     )
578     fig
579     plt.tight_layout()
580     plt.savefig(save_path, dpi=1000, bbox_inches='tight')
581     plt.close()
582 
583 # Example usage
584 file_path = r"C:\Users\victo\OneDrive - NTNU\5.
        ↪  r \Fordypningsprosjekt\Python\Pure_H2\pure_H2_data.xlsx"
585 plot_relative_density_difference_grouped(file_path)
586 
587 
588 
589 
590 def getDeviationSummary(file_path):
591     # Load the Excel file
592     excel_data = pd.ExcelFile(file_path)
593 
594     # Load the Leachman sheet as the base case
595     leachman_df = excel_data.parse("Leachman")
596 
597     # Set Leachman data as the base case, indexed by temperature and
            ↪ pressure
598     leachman_data = leachman_df.set_index(["Temperature (K)", "Pressure
            ↪ (MPa)"])["Density (kg/m^3)"]
```

```
599
600     # Dictionary to store max and average relative differences for each
            ↪ sheet
601     summary_data = {}
602
603     # Iterate through each sheet (excluding Leachman) to calculate
            ↪ relative differences
604     for sheet_name in excel_data.sheet_names:
605         if sheet_name == "Leachman":
606             continue
607
608         # Load the data for the current sheet
609         df = excel_data.parse(sheet_name)
610
611         # Filter to include only rows where Pressure (MPa) is
                ↪ approximately an integer
612         df = df[df["Pressure (MPa)"].round() == df["Pressure (MPa)"]]
613
614         # Merge with Leachman data on temperature and pressure
615         merged_df = df.merge(
616             leachman_df,
617             on=["Temperature (K)", "Pressure (MPa)"],
618             suffixes=('', '_Leachman')
619         )
620
621         # Calculate relative difference in density
622         merged_df["Relative Difference (%)"] = (
623             100 * (merged_df["Density (kg/m^3)"] - merged_df["Density
                    ↪ (kg/m^3)_Leachman"])
624             / merged_df["Density (kg/m^3)_Leachman"]
625         )
626
627         # Calculate maximum and average relative differences
628         max_rel_diff = merged_df["Relative Difference (%)"].max()
629         avg_rel_diff = merged_df["Relative Difference (%)"].mean()
630
631         # Store the summary data
632         summary_data[sheet_name] = {
633             "Max Relative Difference (%)": max_rel_diff,
634             "Average Relative Difference (%)": avg_rel_diff
635         }
636
637     # Convert summary data to a DataFrame for easier display and
            ↪ analysis
638     summary_df = pd.DataFrame.from_dict(summary_data, orient="index")
639     summary_df.index.name = "Sheet Name"
640
641     # Return the summary DataFrame
642     return summary_df
643
644 file_path = r"C:\Users\victo\OneDrive - NTNU\5.
        ↪  r \Fordypningsprosjekt\Python\Pure_H2\pure_H2_data.xlsx"
645 summary_df = getDeviationSummary(file_path)
646 print(summary_df)
```

B.2 Code for the study by Hernández-Gómez et al.[16]

Code written in python for calculating the density data and comparing them to the experimental data. The code contains functions for gathering the states for that study, to generate the density data and functions for plotting and statistics.

```python
from neqsim.thermo import TPflash, fluid, printFrame
import pandas as pd
import matplotlib.pyplot as plt
import os
import numpy as np
import matplotlib



# Function to obtain the temp and pressure from this study
def get_temp_pressure(filepath, sheet_name, T_col, P_col, skip_rows):
    temperature_list = pd.read_excel(filepath, sheet_name=sheet_name, skiprows=skip_rows, usecols=T_col).iloc[:, 0]
    pressure_list = pd.read_excel(filepath, sheet_name=sheet_name, skiprows=skip_rows, usecols=P_col).iloc[:, 0]

    # Combine the two columns into a DataFrame to ensure we handle both columns together
    temp_pressure_df = pd.DataFrame({'Temperature': temperature_list, 'Pressure': pressure_list})

    # Drop rows where either Temperature or Pressure is NaN
    temp_pressure_df.dropna(inplace=True)

    # Convert the columns back to lists after dropping NaN values
    temperature_list = temp_pressure_df['Temperature'].tolist()
    pressure_list = temp_pressure_df['Pressure'].tolist()

    return temperature_list, pressure_list

# Generating data for SRK, PR and GERG2008 for the relevant states and mixture
def generate_data(fluid_type, components, mole_frac, file_path, sheet_name, T_col, P_col, skip_rows):
    data = []
    if fluid_type == "GERG2008":
        fluid1 = fluid("pr")
    else:
        fluid1 = fluid(fluid_type)
    for i in range(len(components)):
        fluid1.addComponent(components[i], mole_frac[i], "mol/sec")
    T_values, P_values = get_temp_pressure(file_path, sheet_name, T_col, P_col, skip_rows)

    for i in range(len(T_values)):
        fluid1.setTemperature(T_values[i], "K")
        fluid1.setPressure(P_values[i], "MPa")
        fluid1.setMixingRule("classic")
        TPflash(fluid1)
        fluid1.initThermoProperties()

        if fluid_type == "GERG2008":
            #Check for liquid
            if fluid1.getNumberOfPhases() != 1:
                phase_names = [fluid1.getPhase(phase).getPhaseTypeName() for phase in range(fluid1.getNumberOfPhases())]
                print(f"Error: More than one phase detected. Number of phases: {fluid1.getNumberOfPhases()}")
                print(f"Phases detected: {', '.join(phase_names)}")
                raise RuntimeError("Too many phases detected, stopping the code.")
            density = fluid1.getPhase("gas").getDensity_GERG2008()
```

```
53          else:
54              density = fluid1.getDensity()
55
56          data.append((T_values[i], P_values[i], density))
57      return data
58
59
60  def save_data_to_excel(data_dict, output_file):
61      with pd.ExcelWriter(output_file, engine='xlsxwriter') as writer:
62          for eos, data in data_dict.items():
63              df = pd.DataFrame(data, columns=["Temperature (K)",
                    ↪ "Pressure (MPa)", "Density"])
64              df.to_excel(writer, sheet_name=eos, index=False)
65      print(f"Density data successfully saved to {output_file}")
66
67  matplotlib.use('Agg')
68
69  file_path = r"C:\Users\victo\OneDrive - NTNU\5.
        ↪  r \Fordypningsprosjekt\Python\Eksperimentelle data.xlsx"
70  output_file = r"C:\Users\victo\OneDrive - NTNU\5.
        ↪  r \Fordypningsprosjekt\Python\density_results_Ark1_classic.xlsx"
71
72
73  EOSS = ["srk", "pr", "GERG2008"]
74  components = ["methane", "hydrogen"]
75  mole_frac = [0.5, 0.5]
76  sheet_name = "Ark 1"
77  T_col = "O"
78  P_col = "P"
79  skip_rows = 6
80
81  #data = generate_data(EoS, components, mole_frac, file_path,
        ↪ sheet_name, T_col, P_col, skip_rows)
82
83  # Save only density data to a .txt file
84  all_data = {}
85  for eos in EOSS:  # Iterate over the list of EoS
86      print(f"Processing for EoS: {eos}")
87      all_data[eos] = generate_data(eos, components, mole_frac,
            ↪ file_path, sheet_name, T_col, P_col, skip_rows)
88
89  # Save all data to an Excel file
90  #save_data_to_excel(all_data, output_file)
91
92
93  def ard_grouped_plots_by_mole_fraction(file_path):
94      # Load all sheets from the Excel file into a dictionary of
            ↪ DataFrames
95      all_sheets = pd.read_excel(file_path, sheet_name=None)
96
97      # Dictionary to store data grouped by mole fraction of H2
98      grouped_data = {}
99
100     # Iterate over each sheet to group data by mole fraction of H2
101     for sheet_name, data in all_sheets.items():
102         # Drop rows with any missing values and explicitly make a copy
103         data = data.dropna().copy()
104
105         # Rename columns for clarity
106         data.columns = ['Temperature (K)', 'Pressure (MPa)',
                ↪ 'Experimental Density (kg/m^3)',
107                         'SRK_NeqSim (kg/m^3)', 'PR_NeqSim (kg/m^3)',
                            ↪ 'GERG Density (kg/m^3)',
```

```
                        'Molefrac_H2', 'SRK_HYSYS (kg/m^3)', 'PR_HYSYS
                            ↪ (kg/m^3)']

        # Calculate relative deviations as percentages
        data['ARD SRK (%)'] = 100 * (data['SRK_NeqSim (kg/m^3)'] -
            ↪ data['Experimental Density (kg/m^3)']) /
            ↪ data['Experimental Density (kg/m^3)']
        data['ARD PR (%)'] = 100 * (data['PR_NeqSim (kg/m^3)'] -
            ↪ data['Experimental Density (kg/m^3)']) /
            ↪ data['Experimental Density (kg/m^3)']
        data['ARD GERG (%)'] = 100 * (data['GERG Density (kg/m^3)'] -
            ↪ data['Experimental Density (kg/m^3)']) /
            ↪ data['Experimental Density (kg/m^3)']
        data['ARD SRK HYSYS (%)'] = 100 * (data['SRK_HYSYS (kg/m^3)'] -
            ↪ data['Experimental Density (kg/m^3)']) /
            ↪ data['Experimental Density (kg/m^3)']
        data['ARD PR HYSYS (%)'] = 100 * (data['PR_HYSYS (kg/m^3)'] -
            ↪ data['Experimental Density (kg/m^3)']) /
            ↪ data['Experimental Density (kg/m^3)']
        #print(data['ARD PR (%)'].max)

        # Group data by mole fraction of H2
        mole_H2 = data['Molefrac_H2'].iloc[0]
        if mole_H2 not in grouped_data:
            grouped_data[mole_H2] = []
        grouped_data[mole_H2].append(data)

    # Create grouped plots for each mole fraction of H2
    for mole_H2, datasets in grouped_data.items():
        # Filter out datasets with Temperature = 300 K
        datasets = [data for data in datasets if
            ↪ int(round(data['Temperature (K)'].iloc[0])) != 300]

        # If no datasets remain after filtering, skip this mole fraction
        if not datasets:
            continue

        num_isotherms = len(datasets)
        num_cols = 3
        num_rows = (num_isotherms + num_cols - 1) // num_cols  #
            ↪ Compute rows based on columns
        fig = plt.figure(figsize=(15, 5 * num_rows))

        # Plot each isotherm in a 3-column layout
        for i, data in enumerate(datasets):
            temperature = int(round(data['Temperature (K)'].iloc[0]))

            # Compute grid position
            row = i // num_cols
            col = i % num_cols
            ax = plt.subplot2grid((num_rows, num_cols), (row, col))

            # Scatter plot data
            ax.scatter(data['Pressure (MPa)'], data['ARD SRK (%)'],
                ↪ label='SRK NeqSim', marker='x', color='black')
            ax.scatter(data['Pressure (MPa)'], data['ARD PR (%)'],
                ↪ label='PR NeqSim', marker='^', color='black',
                ↪ facecolors='none')
            ax.scatter(data['Pressure (MPa)'], data['ARD GERG (%)'],
                ↪ label='GERG-2008', marker='s', color='black',
                ↪ facecolors='none')
            ax.scatter(data['Pressure (MPa)'], data['ARD SRK HYSYS
                ↪ (%)'], label='SRK HYSYS', marker='o', color='black',
```

```
                ↪ facecolors='none')
152             ax.scatter(data['Pressure (MPa)'], data['ARD PR HYSYS
                    ↪ (%)'], label='PR HYSYS', marker='D', color='black',
                    ↪ facecolors='none')
153
154             ax.set_title(f"T = {temperature} K", fontsize = 14)
155             ax.grid(True)
156             ax.set_ylim(-6.5, 6.5)
157             ax.set_xlim(None, 20)
158             ax.set_xlabel('Pressure (MPa)')
159
160         # Add shared y-axis label
161         fig.text(0.0001, 0.5, r'$100 \left(\rho_{\mathrm{model}} -
                ↪ \rho_{\mathrm{experimental}}\right)/\rho_{\mathrm{experimental}}$',
                ↪ ha='center', va='center', rotation='vertical', fontsize =
                ↪ 16)
162
163         # Add a single shared legend at the bottom
164         handles, labels = ax.get_legend_handles_labels()
165         fig.legend(handles, labels, loc='upper center',
                ↪ bbox_to_anchor=(0.5, 0), ncol=5, frameon=False, fontsize
                ↪ = 16)
166
167         # Save the grouped plot
168         save_path = os.path.join("C:\\Users\\victo\\OneDrive - NTNU\\5.
                ↪  r \\Fordypningsprosjekt\\Python\\Ark_1\\plots",
169                                  f"Grouped_{1-mole_H2:.2f}CH4_{mole_H2:.2f}H2.png")
170         plt.tight_layout(rect=[0, 0.05, 1, 0.95])  # Adjust layout to
                ↪ fit the legend
171         plt.savefig(save_path, dpi=300, bbox_inches='tight')
172         plt.close()
173
174 plot_data = r"C:\Users\victo\OneDrive - NTNU\5.
    ↪  r \Fordypningsprosjekt\Python\Ark_1\Plot_data_Ark_1.xlsx"
175
176
177
178 ard_grouped_plots_by_mole_fraction(plot_data)
179
180
181
182 def compute_rd_statistics_by_mole_fraction(file_path):
183     """
184     Computes the mean, maximum, and minimum Relative Deviations (RD)
            ↪ grouped by
185     mole fraction of H2 across all sheets, and prints the results with
            ↪ high decimal precision.
186
187     Parameters:
188     - file_path: str : Path to the Excel file.
189     """
190
191     # Load all sheets from the Excel file
192     try:
193         all_sheets = pd.read_excel(file_path, sheet_name=None)
194     except Exception as e:
195         raise ValueError(f"Error reading Excel file: {e}")
196
197     # List to aggregate all data
198     all_data = []
199
200     # Iterate over each sheet
201     for sheet_name, data in all_sheets.items():
```

```
        # Drop rows with any missing values and explicitly make a copy
        data = data.dropna().copy()

        # Rename columns for clarity (ensure these column names are correct for your data)
        data.columns = ['Temperature (K)', 'Pressure (MPa)', 'Experimental Density (kg/m^3)',
                        'SRK_NeqSim (kg/m^3)', 'PR_NeqSim (kg/m^3)', 'GERG Density (kg/m^3)',
                        'Molefrac_H2', 'SRK_HYSYS (kg/m^3)', 'PR_HYSYS (kg/m^3)']

        # Calculate relative deviations (signed) as percentages
        data['RD SRK_NeqSim (%)'] = 100 * ((data['SRK_NeqSim (kg/m^3)'] - data['Experimental Density (kg/m^3)']) / data['Experimental Density (kg/m^3)'])
        data['RD PR_NeqSim (%)'] = 100 * ((data['PR_NeqSim (kg/m^3)'] - data['Experimental Density (kg/m^3)']) / data['Experimental Density (kg/m^3)'])
        data['RD GERG (%)'] = 100 * ((data['GERG Density (kg/m^3)'] - data['Experimental Density (kg/m^3)']) / data['Experimental Density (kg/m^3)'])
        data['RD SRK_HYSYS (%)'] = 100 * ((data['SRK_HYSYS (kg/m^3)'] - data['Experimental Density (kg/m^3)']) / data['Experimental Density (kg/m^3)'])
        data['RD PR_HYSYS (%)'] = 100 * ((data['PR_HYSYS (kg/m^3)'] - data['Experimental Density (kg/m^3)']) / data['Experimental Density (kg/m^3)'])

        # Append to the aggregated list
        all_data.append(data)

    # Combine all sheets into one DataFrame
    combined_data = pd.concat(all_data, ignore_index=True)

    # Group by mole fraction of H2
    grouped = combined_data.groupby('Molefrac_H2')

    # Compute and print statistics for each mole fraction
    print("\n--- Combined RD Statistics by Mole Fraction of H2 ---")
    for mole_frac, group in grouped:
        print(f"\nMole Fraction of H2: {mole_frac:.6f}")
        for column in ['RD SRK_NeqSim (%)', 'RD PR_NeqSim (%)', 'RD GERG (%)', 'RD SRK_HYSYS (%)', 'RD PR_HYSYS (%)']:
            model_name = column.split(' ')[1]
            avg_rd = group[column].mean()
            max_rd = group[column].max()
            min_rd = group[column].min()
            print(f"  Model: {model_name}")
            print(f"    Mean RD (%): {avg_rd:.8f}")
            print(f"    Max RD (%): {max_rd:.8f}")
            print(f"    Min RD (%): {min_rd:.8f}")

    print("\n--- End of Results ---")


compute_rd_statistics_by_mole_fraction(plot_data)
```

B.3 Code for the study by Richter et al.

Code written in python for calculating the density data and comparing them to the experimental data. The code contains functions to generate the density data and functions for plotting and statistics. The states and composition for that study is implemented in the code manually.

```
from neqsim.thermo import TPflash, fluid
import pandas as pd
import matplotlib.pyplot as plt
from openpyxl.utils import column_index_from_string
import numpy as np
import os
import matplotlib
matplotlib.use('Agg')


# Function to generate density data for a given fluid type, components, mole fractions, and pressures
def generate_data(fluid_type, components, mole_frac, pressures):
    data = []
    if fluid_type == "GERG2008":
        fluid1 = fluid("pr")
    else:
        fluid1 = fluid(fluid_type)
    for i in range(len(components)):
        fluid1.addComponent(components[i], mole_frac[i], "mol/sec")
    fluid1.setMixingRule("classic")

    densities = []
    for i in range(len(T_values)):  # Iterate over temperatures
        temp_densities = []
        fluid1.setTemperature(T_values[i], "K")
        for j in range(len(pressures[i])):  # Iterate over pressures for the current temperature
            fluid1.setPressure(pressures[i][j], "MPa")
            TPflash(fluid1)
            fluid1.initThermoProperties()

            if fluid_type == "GERG2008":
                if fluid1.getNumberOfPhases() != 1:
                    phase_names = [fluid1.getPhase(phase).getPhaseTypeName() for phase in range(fluid1.getNumberOfPhases())]
                    print(f"Error: More than one phase detected. Number of phases: {fluid1.getNumberOfPhases()}")
                    print(f"Phases detected: {', '.join(phase_names)}")
                    raise RuntimeError("Too many phases detected, stopping the code.")
                density = fluid1.getPhase("gas").getDensity_GERG2008()
            else:
                density = fluid1.getDensity()

            temp_densities.append(density)
        densities.append(temp_densities)  # Append the density data for the current temperature
    return densities
```

```
# Function to save density data and pressures into an Excel file with multiple sheets and organized tables
def save_data_to_excel(pressure_data, density_data, output_file, mixtures, T_values, EOSS):
    with pd.ExcelWriter(output_file, engine='xlsxwriter') as writer:
        for mix_idx, mixture in enumerate(mixtures):
            # Initialize start_row to ensure each table is written below the previous one
            start_row = 0
            sheet_name = f"Mixture_{mix_idx + 1}"

            # Explicitly add a new worksheet to avoid KeyError
            writer.book.add_worksheet(sheet_name)
            worksheet = writer.sheets[sheet_name]

            for temp_idx, temperature in enumerate(T_values):
                # If there's no data for this temperature for the current mixture, skip it
                if not pressure_data[mix_idx] or not pressure_data[mix_idx][temp_idx]:
                    continue

                pressure_column = pressure_data[mix_idx][temp_idx]
                temp_df = pd.DataFrame({'Pressure': pressure_column})

                # Add density columns for each EOS
                for EOS in EOSS:
                    densities_for_eos = density_data.get(EOS, [])
                    density_values = (
                        densities_for_eos[mix_idx][temp_idx] if mix_idx < len(densities_for_eos) and temp_idx < len(densities_for_eos[mix_idx])
                        else []
                    )
                    # Fill with None if no density data is available for alignment with pressure column
                    temp_df[EOS] = density_values if density_values else [None] * len(pressure_column)

                # Add a row for the title
                worksheet.write(start_row, 0, f"Data at {temperature} K")  # Title for each table

                # Write DataFrame to the sheet, starting just below the title
                temp_df.to_excel(writer, sheet_name=sheet_name, startrow=start_row + 1, index=False)

                # Update `start_row` for the next table
                start_row += len(temp_df) + 5  # Move down for next table, including space between tables


file_path = r"C:\Users\victo\OneDrive - NTNU\5. r \Fordypningsprosjekt\Python\Eksperimentelle data.xlsx"
output_file = r"C:\Users\victo\OneDrive - NTNU\5. r \Fordypningsprosjekt\Python\density_results.xlsx"

EoS = ["srk", "pr", "GERG2008"]
# Components from the study
components = [
    "methane",  #  C H
    "ethane",  #    C  H
```

```
    "propane",  #   C  H
    "n-butane",  # n-   C   H
    "i-butane",  # iso-   C   H
    "n-pentane",  # n-   C   H
    "i-pentane",  # iso-   C   H
    "i-pentane",  # neo-   C   H
    "n-hexane",  # n-   C   H
    "n-heptane",  # n-   C   H
    "n-octane",  # n-   C   H
    "n-nonane",  # n-   C   H
    "nC10",  # n-    C    H
    "benzene",  #   C  H
    "toluene",  #   C  H
    "o-Xylene",  # o-   C   H
    "CO2",  #  C O
    "nitrogen",  #  N
    "oxygen",  #  O
    "helium",  # He
    "hydrogen"  #  H
]


NG1 = [
    0.85381, 0.054534, 0.010347, 0.001479, 0.001355, 0.000256,
    0.000355, 0.000015, 0.000214, 0.000087, 0.000008, 0.000001,
    1e-7, 0.000008, 0.000004, 0.000001,
    0.016487, 0.007284, 1e-4,
    0.000074, 0.053681
]

NG2 = [
    0.810099, 0.050884, 0.009147, 0.001273, 0.001217, 0.000229,
    0.000323, 0.000014, 0.000198, 0.000081, 0.000008, 0.000001,
    1e-7, 0.000007, 0.000004, 0.000001,
    0.015529, 0.006803, 1e-4,
    0.000076, 0.104106
]

NG3 = [
    0.628871, 0.039369, 0.007226, 0.001017, 0.000955, 0.00018,
    0.000253, 0.000011, 0.000152, 0.000067, 0.000006, 0.000001,
    1e-7, 0.000006, 0.000003, 1e-7,
    0.01193, 0.005212, 1e-4,
    0.000036, 0.304705
]


sheet_name = "Ark 1"
T_col = "O"
P_col = "P"
skip_rows = 6

# Temperature and pressure from the study
T_values = [273.15, 283.15, 293.15]
pressure_NG1 = [
    [8.01519, 8.01552, 8.0156, 6.02646, 6.02664, 6.02684, 4.02937,
        ↪ 4.0294, 2.02441, 2.02444, 2.02453, 1.02691, 1.02703,
        ↪ 1.02711],  # for 273.15K
    [8.01622, 8.01655, 6.01266, 6.01299, 6.01322, 4.0125, 4.01264,
        ↪ 2.00668, 2.00677, 1.01136, 1.01141],  # for 283.15K
    [8.01727, 8.01779, 8.01773, 6.02016, 6.02014, 4.00946, 4.00963,
        ↪ 4.00971, 2.01347, 2.01345, 1.00176, 1.0018]  # for 293.15K
]
```

```
156
157 pressure_NG2 = [
158     [8.04333, 8.04361, 8.04362, 6.02718, 6.02739, 6.02756, 4.01711,
        ↪ 4.01718, 4.01727, 2.01767, 2.01771, 2.01769, 1.0212, 1.02122,
        ↪ 1.02126],  # for 273.15K
159     [8.03641, 8.0365, 6.02663, 6.02681, 4.03782, 4.03791, 2.02035,
        ↪ 2.02043, 1.01513, 1.01514],  # for 283.15K
160     [8.01726, 8.01726, 6.01183, 6.01183, 4.01735, 4.01731, 4.01741,
        ↪ 2.00945, 2.00947, 1.01009, 1.01014]  # for 293.15K
161 ]
162
163 pressure_NG3 = [
164     [],
165     [8.01953, 8.01965, 8.0197, 6.0041, 6.00426, 4.00844, 4.00855,
        ↪ 2.01023, 2.01034, 2.01035, 1.00913, 1.00918, 1.00922],
166     []
167 ]
168
169
170 mixtures = [NG1, NG2, NG3]
171 pressures = [pressure_NG1, pressure_NG2, pressure_NG3]
172
173 EOSS = ["srk", "pr", "GERG2008"]
174
175 # To store pressure and density data
176 density_data = {EOS: [] for EOS in EOSS}
177
178 for EOS in EOSS:
179     for l in range(len(mixtures)):
180         try:
181             # Capture generated data for each mixture under each EOS
182             density_data[EOS].append(generate_data(EOS, components,
                ↪ mixtures[l], pressures[l]))
183         except Exception as e:
184             print(f"Error generating data for EOS: {EOS}, Mixture
                ↪ index: {l}, Error: {e}")
185             density_data[EOS].append([])  # Append an empty list in
                ↪ case of failure to keep the structure consistent
186
187 # Save the data to an Excel file
188 save_data_to_excel(pressures, density_data, output_file, mixtures,
    ↪ T_values, EOSS)
189
190 comparison_data = r"C:\Users\victo\OneDrive - NTNU\5.
    ↪  r \Fordypningsprosjekt\Python\Comparison of data.xlsx"
191
192 plot_data = r"C:\Users\victo\OneDrive - NTNU\5.
    ↪  r \Fordypningsprosjekt\Python\Ark_2\Plot_data_Ark_2.xlsx"
193 hysys_data = r"C:\Users\victo\OneDrive - NTNU\5.
    ↪  r \Fordypningsprosjekt\Python\Ark_2\Plot_data_HYSYS.xlsx"
194
195 # Function to create grouped ARD (Absolute Relative Deviation) plots
    ↪ based on Excel data for different models
196 def ard_grouped_plot(file_path):
197     """
198     Creates grouped plots with subplots for each molar fraction of
        ↪ hydrogen (H2).
199     Each subplot shows ARD values for different temperature data sets
        ↪ for the same molar fraction, arranged in a single row.
200
201     Parameters:
202     - file_path: str : Path to the Excel file.
203     """
```

```
204
205     # Load all sheets from the Excel file into a dictionary of
            ↪ DataFrames
206     all_sheets = pd.read_excel(file_path, sheet_name=None)
207
208     # Dictionary to group data by molar fraction of H2
209     grouped_data = {}
210
211     # Iterate over each sheet
212     for sheet_name, data in all_sheets.items():
213         # Drop rows with any missing values and make a copy to avoid
                ↪ SettingWithCopyWarning
214         data = data.dropna().copy()
215
216         # Rename columns for clarity, assuming the columns follow the
                ↪ same format as provided
217         data.columns = ['Temperature (K)', 'Pressure (MPa)',
                ↪ 'Experimental Density (kg/m^3)',
218                         'SRK_NeqSim Density (kg/m^3)', 'PR_NeqSim
                            ↪ Density (kg/m^3)', 'GERG Density
                            ↪ (kg/m^3)',
219                         'Molefrac_H2', 'SRK_HYSYS', 'PR_HYSYS']
220
221         # Get the molar fraction of H2
222         mole_H2 = data['Molefrac_H2'].iloc[0]
223
224         # Add the dataset to the grouped data dictionary
225         if mole_H2 not in grouped_data:
226             grouped_data[mole_H2] = []
227         grouped_data[mole_H2].append(data)
228
229     # Create grouped plots for each molar fraction
230     for mole_H2, datasets in grouped_data.items():
231         num_temps = len(datasets)
232
233         # Set number of columns equal to the number of temperatures for
                ↪ a single-row layout
234         num_cols = num_temps
235         num_rows = 1  # Only one row
236
237         # Initialize the figure
238         fig, axes = plt.subplots(num_rows, num_cols, figsize=(5 *
                ↪ num_cols, 5))
239         if num_temps == 1:
240             axes = [axes]  # Make axes iterable if there is only one
                    ↪ subplot
241
242         for i, data in enumerate(datasets):
243             ax = axes[i]
244
245             # Extract the temperature for labeling
246             temperature = data['Temperature (K)'].iloc[0]
247
248             # Calculate relative deviations as percentages for each
                    ↪ density model
249             data['ARD SRK_NeqSim (%)'] = 100 * (data['SRK_NeqSim
                    ↪ Density (kg/m^3)'] - data['Experimental Density
                    ↪ (kg/m^3)']) / data['Experimental Density (kg/m^3)']
250             data['ARD PR_NeqSim (%)'] = 100 * (data['PR_NeqSim Density
                    ↪ (kg/m^3)'] - data['Experimental Density (kg/m^3)']) /
                    ↪ data['Experimental Density (kg/m^3)']
251             data['ARD GERG (%)'] = 100 * (data['GERG Density (kg/m^3)']
                    ↪ - data['Experimental Density (kg/m^3)']) /
```

```
                ↪ data['Experimental Density (kg/m^3)']
252             data['ARD SRK_HYSYS (%)'] = 100 * (data['SRK_HYSYS'] -
                ↪ data['Experimental Density (kg/m^3)']) /
                ↪ data['Experimental Density (kg/m^3)']
253             data['ARD PR_HYSYS (%)'] = 100 * (data['PR_HYSYS'] -
                ↪ data['Experimental Density (kg/m^3)']) /
                ↪ data['Experimental Density (kg/m^3)']
254
255             # Scatter plot the ARD values
256             ax.scatter(data['Pressure (MPa)'], data['ARD SRK_NeqSim
                ↪ (%)'], label='SRK NeqSim', marker='x', color='black')
257             ax.scatter(data['Pressure (MPa)'], data['ARD PR_NeqSim
                ↪ (%)'], label='PR NeqSim', marker='^', color='black',
                ↪ facecolors='none')
258             ax.scatter(data['Pressure (MPa)'], data['ARD GERG (%)'],
                ↪ label='GERG-2008', marker='s', color='black',
                ↪ facecolors='none')
259             ax.scatter(data['Pressure (MPa)'], data['ARD SRK_HYSYS
                ↪ (%)'], label='SRK HYSYS', marker='o', color='black',
                ↪ facecolors='none')
260             ax.scatter(data['Pressure (MPa)'], data['ARD PR_HYSYS
                ↪ (%)'], label='PR HYSYS', marker='D', color='black',
                ↪ facecolors='none')
261
262             # Set subplot title and labels
263             ax.set_title(f"T = {temperature} K", fontsize = 14)
264             ax.set_xlabel('Pressure (MPa)')
265             ax.set_ylim(-3, 4.5)
266             #ax.set_ylabel(r'$100 \left(\rho_{\text{model}} -
                ↪ \rho_{\text{experimental}}\right)/\rho_{\text{experimental}}$')
267             ax.grid(True)
268
269         fig.text(0.0001, 0.5, r'$100 \left(\rho_{\mathrm{model}} -
            ↪ \rho_{\mathrm{experimental}}\right)/\rho_{\mathrm{experimental}}$',
            ↪ ha='center', va='center', rotation='vertical', fontsize =
            ↪ 16)
270
271         # Add a single legend below the entire figure
272         handles, labels = ax.get_legend_handles_labels()
273         fig.legend(handles, labels, loc='lower center',
            ↪ bbox_to_anchor=(0.5, -0.1), ncol=5, frameon=False,
            ↪ fontsize = 16)
274         #plt.ylim(-3, 4.5)
275
276
277         # Save the grouped plot
278         save_path = os.path.join("C:\\Users\\victo\\OneDrive - NTNU\\5.
            ↪  r \\Fordypningsprosjekt\\Python\\Ark_2\\plots",
279                                  f"Grouped_{1-mole_H2:.2f}CH4_{mole_H2:.2f}H2.png")
280         plt.tight_layout(rect=[0, 0.05, 1, 0.95])  # Adjust layout to
            ↪ fit the legend
281         plt.savefig(save_path, dpi=300, bbox_inches='tight')
282         plt.close()
283
284 #ard_grouped_plot(plot_data)
285
286
287
288 # Function to calculate and print statistical metrics (mean and max
      ↪ ARD) for different models based on Excel data
289 def ard_grouped_statistics(file_path):
290     """
```

```
    Calculates and prints the mean and maximum relative deviations for different models compared to
    experimental data for each molar fraction of hydrogen (H2).

    Parameters:
    - file_path: str : Path to the Excel file.
    """

    # Load all sheets from the Excel file into a dictionary of DataFrames
    all_sheets = pd.read_excel(file_path, sheet_name=None)

    # Dictionary to group data by molar fraction of H2
    grouped_data = {}

    # Iterate over each sheet
    for sheet_name, data in all_sheets.items():
        # Drop rows with any missing values and make a copy to avoid SettingWithCopyWarning
        data = data.dropna().copy()

        # Rename columns for clarity, assuming the columns follow the same format as provided
        data.columns = ['Temperature (K)', 'Pressure (MPa)', 'Experimental Density (kg/m^3)',
                        'SRK_NeqSim Density (kg/m^3)', 'PR_NeqSim Density (kg/m^3)', 'GERG Density (kg/m^3)',
                        'Molefrac_H2', 'SRK_HYSYS', 'PR_HYSYS']

        # Get the molar fraction of H2
        mole_H2 = data['Molefrac_H2'].iloc[0]

        # Add the dataset to the grouped data dictionary
        if mole_H2 not in grouped_data:
            grouped_data[mole_H2] = []
        grouped_data[mole_H2].append(data)

    # Iterate over grouped data to calculate statistics
    for mole_H2, datasets in grouped_data.items():
        # Combine all datasets for this molar fraction of H2
        combined_data = pd.concat(datasets, ignore_index=True)

        # Calculate relative deviations as percentages for each density model
        combined_data['ARD SRK_NeqSim (%)'] = 100 * (combined_data['SRK_NeqSim Density (kg/m^3)'] - combined_data['Experimental Density (kg/m^3)']) / combined_data['Experimental Density (kg/m^3)']
        combined_data['ARD PR_NeqSim (%)'] = 100 * (combined_data['PR_NeqSim Density (kg/m^3)'] - combined_data['Experimental Density (kg/m^3)']) / combined_data['Experimental Density (kg/m^3)']
        combined_data['ARD GERG (%)'] = 100 * (combined_data['GERG Density (kg/m^3)'] - combined_data['Experimental Density (kg/m^3)']) / combined_data['Experimental Density (kg/m^3)']
        combined_data['ARD SRK_HYSYS (%)'] = 100 * (combined_data['SRK_HYSYS'] - combined_data['Experimental Density (kg/m^3)']) / combined_data['Experimental Density (kg/m^3)']
        combined_data['ARD PR_HYSYS (%)'] = 100 * (combined_data['PR_HYSYS'] - combined_data['Experimental
```

```
            ↪ Density (kg/m^3)']) / combined_data['Experimental Density
            ↪ (kg/m^3)']

        # Calculate mean and maximum absolute relative deviations for
            ↪ each model
        statistics = {
            'SRK_NeqSim': {
                'mean': combined_data['ARD SRK_NeqSim (%)'].mean(),
                'max': combined_data['ARD SRK_NeqSim (%)'].abs().max()
            },
            'PR_NeqSim': {
                'mean': combined_data['ARD PR_NeqSim (%)'].mean(),
                'max': combined_data['ARD PR_NeqSim (%)'].abs().max()
            },
            'GERG': {
                'mean': combined_data['ARD GERG (%)'].mean(),
                'max': combined_data['ARD GERG (%)'].abs().max()
            },
            'SRK_HYSYS': {
                'mean': combined_data['ARD SRK_HYSYS (%)'].mean(),
                'max': combined_data['ARD SRK_HYSYS (%)'].abs().max()
            },
            'PR_HYSYS': {
                'mean': combined_data['ARD PR_HYSYS (%)'].mean(),
                'max': combined_data['ARD PR_HYSYS (%)'].abs().max()
            }
        }

        # Print results for this molar fraction
        print(f"Results for Molefrac_H2 = {mole_H2:.2f}")
        for model, stats in statistics.items():
            print(f"  {model} - Mean ARD (%): {stats['mean']:.8f}, Max
                ↪ ARD (%): {stats['max']:.8f}")
        print("-" * 50)


#ard_grouped_statistics(plot_data)
```

B.4 Code for the study by Ben Souissi et al.

Code written in python for calculating the density data and comparing them to the experimental data. The code contains functions to generate the density data and functions for plotting and statistics. The states and composition for that study is implemented in the code manually.

```
from neqsim.thermo import TPflash, fluid
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
import matplotlib
matplotlib.use('Agg')


# Function to generate density data for a given fluid type, components,
    ↪ mole fractions, pressures, and temperatures

def generate_data(fluid_type, components, mole_frac, pressures,
    ↪ T_values):
    data = []
    if fluid_type == "GERG2008":
        fluid1 = fluid("pr")
```

```
    else:
        fluid1 = fluid(fluid_type)
    for i in range(len(components)):
        fluid1.addComponent(components[i], mole_frac[i], "mol/sec")
    fluid1.setMixingRule("classic")

    densities = []
    for i in range(len(T_values)):  # Iterate over temperatures
        temp_densities = []
        fluid1.setTemperature(T_values[i], "K")
        for j in range(len(pressures[i])):  # Iterate over pressures
            ↪ for the current temperature
            fluid1.setPressure(pressures[i][j], "MPa")
            TPflash(fluid1)
            fluid1.initThermoProperties()

            if fluid_type == "GERG2008":
                if fluid1.getNumberOfPhases() != 1:
                    phase_names =
                        ↪ [fluid1.getPhase(phase).getPhaseTypeName()
                        ↪ for phase in
                        ↪ range(fluid1.getNumberOfPhases())]
                    print(f"Error: More than one phase detected. Number
                        ↪ of phases: {fluid1.getNumberOfPhases()}")
                    print(f"Phases detected: {', '.join(phase_names)}")
                    raise RuntimeError("Too many phases detected,
                        ↪ stopping the code.")
                density = fluid1.getPhase("gas").getDensity_GERG2008()
            else:
                density = fluid1.getDensity()

            temp_densities.append(density)
        densities.append(temp_densities)  # Append the density data for
            ↪ the current temperature
    return densities


# Function to save pressure and density data into an Excel file with
    ↪ multiple sheets and organized tables
def save_data_to_excel(pressure_data, density_data, output_file,
    ↪ T_values, EOSS):
    with pd.ExcelWriter(output_file, engine='xlsxwriter') as writer:
        # Initialize start_row to ensure each table is written below
            ↪ the previous one
        start_row = 0
        sheet_name = "Mixture_1"

        # Explicitly add a new worksheet to avoid KeyError
        writer.book.add_worksheet(sheet_name)
        worksheet = writer.sheets[sheet_name]

        for temp_idx, temperature in enumerate(T_values):
            # Get the pressure data for the current temperature
            if not pressure_data[temp_idx]:  # Skip if there's no
                ↪ pressure data for the current temperature
                continue

            pressure_column = pressure_data[temp_idx]
            temp_df = pd.DataFrame({'Pressure': pressure_column})

            # Add density columns for each EOS
            for EOS in EOSS:
                densities_for_eos = density_data.get(EOS, [])
```

```
68
69                  # Retrieve density values for the current temperature
                        ↪ (temp_idx)
70                  density_values = densities_for_eos[0][temp_idx] if
                        ↪ densities_for_eos and temp_idx <
                        ↪ len(densities_for_eos[0]) else []
71
72                  # Adjust length of density_values to match
                        ↪ pressure_column length
73                  if len(density_values) < len(pressure_column):
74                      density_values += [None] * (len(pressure_column) -
                            ↪ len(density_values))
75                  elif len(density_values) > len(pressure_column):
76                      density_values =
                            ↪ density_values[:len(pressure_column)]
77
78                  # Assign adjusted density values to DataFrame
79                  temp_df[EOS] = density_values
80
81              # Add a row for the title with temperature information
82              worksheet.write(start_row, 0, f"Data at {temperature} K")
                    ↪ # Title for each table
83
84              # Write DataFrame to the sheet, starting just below the
                    ↪ title
85              temp_df.to_excel(writer, sheet_name=sheet_name,
                    ↪ startrow=start_row + 1, index=False)
86
87              # Update `start_row` for the next table
88              start_row += len(temp_df) + 5
89
90
91
92
93 plot_data = r"C:\Users\victo\OneDrive - NTNU\5.
       ↪  r \Fordypningsprosjekt\Python\Ark_3\Plot_data_Ark_3.xlsx"
94
95
96 # Function to create grouped ARD plots with a single-row layout for
       ↪ multiple temperatures
97
98 def ard_grouped_plot_with_single_row(file_path):
99     # Load all sheets from the Excel file into a dictionary of
           ↪ DataFrames
100    all_sheets = pd.read_excel(file_path, sheet_name=None)
101
102    # Dictionary to group data by molar fraction of H2
103    grouped_data = {}
104
105    # Iterate over each sheet
106    for sheet_name, data in all_sheets.items():
107        # Make a copy of the data to avoid SettingWithCopyWarning
108        data = data.copy()
109
110        # Drop rows with any missing values
111        data = data.dropna()
112
113        # Rename columns for clarity, assuming the columns follow the
               ↪ same format as provided
114        data.columns = ['Temperature (K)', 'Pressure (MPa)',
               ↪ 'Experimental Density (kg/m^3)',
115                        'SRK_NeqSim Density (kg/m^3)', 'PR_NeqSim
                               ↪ Density (kg/m^3)', 'GERG Density
```

```
                                ↪ (kg/m^3)',
116                         'Molefrac_H2', 'SRK_HYSYS', 'PR_HYSYS']
117
118         # Get the molar fraction of H2
119         mole_H2 = data['Molefrac_H2'].iloc[0]
120
121         # Add the dataset to the grouped data dictionary
122         if mole_H2 not in grouped_data:
123             grouped_data[mole_H2] = []
124         grouped_data[mole_H2].append((sheet_name, data))
125
126     # Create grouped plots for each molar fraction
127     for mole_H2, datasets in grouped_data.items():
128         num_temps = len(datasets)
129         num_cols = num_temps  # Single row layout: columns equal to the
                ↪ number of datasets
130         num_rows = 1  # Always one row
131
132         # Initialize the figure
133         fig, axes = plt.subplots(num_rows, num_cols, figsize=(5 *
                ↪ num_cols, 5))
134         if num_temps == 1:
135             axes = [axes]  # Make axes iterable if there is only one
                    ↪ subplot
136
137         for i, (sheet_name, data) in enumerate(datasets):
138             ax = axes[i]
139
140             # Extract the temperature for labeling
141             temperature = data['Temperature (K)'].iloc[0]
142
143             # Calculate relative deviations as percentages
144             data['ARD SRK_NeqSim (%)'] = 100 * (data['SRK_NeqSim
                    ↪ Density (kg/m^3)'] - data['Experimental Density
                    ↪ (kg/m^3)']) / data['Experimental Density (kg/m^3)']
145             data['ARD PR_NeqSim (%)'] = 100 * (data['PR_NeqSim Density
                    ↪ (kg/m^3)'] - data['Experimental Density (kg/m^3)']) /
                    ↪ data['Experimental Density (kg/m^3)']
146             data['ARD GERG (%)'] = 100 * (data['GERG Density (kg/m^3)']
                    ↪ - data['Experimental Density (kg/m^3)']) /
                    ↪ data['Experimental Density (kg/m^3)']
147             data['ARD SRK_HYSYS (%)'] = 100 * (data['SRK_HYSYS'] -
                    ↪ data['Experimental Density (kg/m^3)']) /
                    ↪ data['Experimental Density (kg/m^3)']
148             data['ARD PR_HYSYS (%)'] = 100 * (data['PR_HYSYS'] -
                    ↪ data['Experimental Density (kg/m^3)']) /
                    ↪ data['Experimental Density (kg/m^3)']
149
150             # Scatter plot the ARD values
151             ax.scatter(data['Pressure (MPa)'], data['ARD SRK_NeqSim
                    ↪ (%)'], label='SRK_NeqSim', marker='x', color='black')
152             ax.scatter(data['Pressure (MPa)'], data['ARD PR_NeqSim
                    ↪ (%)'], label='PR_NeqSim', marker='^', color='black',
                    ↪ facecolors='none')
153             ax.scatter(data['Pressure (MPa)'], data['ARD GERG (%)'],
                    ↪ label='GERG-2008', marker='s', color='black',
                    ↪ facecolors='none')
154             ax.scatter(data['Pressure (MPa)'], data['ARD SRK_HYSYS
                    ↪ (%)'], label='SRK_HYSYS', marker='o', color='black',
                    ↪ facecolors='none')
155             ax.scatter(data['Pressure (MPa)'], data['ARD PR_HYSYS
                    ↪ (%)'], label='PR_HYSYS', marker='D', color='black',
                    ↪ facecolors='none')
```

```
156
157             # Set subplot title and labels
158             ax.set_title(f"T = {temperature} K", fontsize=14)
159             ax.set_xlabel('Pressure (MPa)', fontsize=9)
160             ax.tick_params(axis='both', labelsize=8)
161             ax.set_ylim(-2.5,3)
162             ax.grid(True)
163
164         # Add a shared y-axis label
165         fig.text(0.04, 0.5, r'$100 \left(\rho_{\text{model}} -
                ↪ \rho_{\text{experimental}}\right)/\rho_{\text{experimental}}$',
166                  va='center', rotation='vertical', fontsize=16)
167
168         # Add a single legend below the entire figure
169         handles, labels = ax.get_legend_handles_labels()
170         fig.legend(handles, labels, loc='upper center',
                ↪ bbox_to_anchor=(0.5, 0), ncol=5, frameon=False, fontsize
                ↪ = 16)
171
172         # Save the grouped plot
173         save_path = os.path.join("C:\\Users\\victo\\OneDrive - NTNU\\5.
                ↪  r \\Fordypningsprosjekt\\Python\\Ark_3\\plots",
174                                  f"Grouped_{1-mole_H2:.2f}CO2_{mole_H2:.2f}H2.png")
175         plt.tight_layout(rect=[0.05, 0.1, 1, 0.95])  # Adjust layout to
                ↪ fit the legend and shared ylabel
176         plt.savefig(save_path, dpi=300, bbox_inches='tight')
177         plt.close()
178
179 #ard_grouped_plot_with_single_row(plot_data)
180
181 #file_path = r"C:\Users\victo\OneDrive - NTNU\5.
    ↪  r \Fordypningsprosjekt\Python\Eksperimentelle data.xlsx"
182 output_file = r"C:\Users\victo\OneDrive - NTNU\5.
    ↪  r \Fordypningsprosjekt\Python\density_results.xlsx"
183
184 # Function to calculate and print statistical metrics (mean and max
    ↪ ARD) for a single molar composition of hydrogen (H2)
185
186 def ard_statistics_single_composition(file_path):
187     """
188     Calculates and prints the mean and maximum relative deviations for
            ↪ different models compared to
189     experimental data for a single molar composition of hydrogen (H2).
190
191     Parameters:
192     - file_path: str : Path to the Excel file.
193     """
194     import pandas as pd
195
196     # Load all sheets from the Excel file into a dictionary of
            ↪ DataFrames
197     all_sheets = pd.read_excel(file_path, sheet_name=None)
198
199     # Dictionary to group data by molar fraction of H2
200     grouped_data = {}
201
202     # Iterate over each sheet
203     for sheet_name, data in all_sheets.items():
204         # Make a copy of the data to avoid SettingWithCopyWarning
205         data = data.copy()
206
207         # Drop rows with any missing values
208         data = data.dropna()
```

```
209
210         # Rename columns for clarity, assuming the columns follow the
                ↪ same format as provided
211         data.columns = ['Temperature (K)', 'Pressure (MPa)',
                ↪ 'Experimental Density (kg/m^3)',
212                         'SRK_NeqSim Density (kg/m^3)', 'PR_NeqSim
                               ↪ Density (kg/m^3)', 'GERG Density
                               ↪ (kg/m^3)',
213                         'Molefrac_H2', 'SRK_HYSYS', 'PR_HYSYS']
214
215         # Get the molar fraction of H2
216         mole_H2 = data['Molefrac_H2'].iloc[0]
217
218         # Add the dataset to the grouped data dictionary
219         if mole_H2 not in grouped_data:
220             grouped_data[mole_H2] = []
221         grouped_data[mole_H2].append((sheet_name, data))
222
223     # Process grouped data and calculate mean and max ARD
224     for mole_H2, datasets in grouped_data.items():
225         # Combine all datasets for this molar fraction of H2
226         combined_data = pd.concat([data for _, data in datasets],
                ↪ ignore_index=True)
227
228         # Calculate relative deviations as percentages
229         combined_data['ARD SRK_NeqSim (%)'] = 100 *
                ↪ (combined_data['SRK_NeqSim Density (kg/m^3)'] -
                ↪ combined_data['Experimental Density (kg/m^3)']) /
                ↪ combined_data['Experimental Density (kg/m^3)']
230         combined_data['ARD PR_NeqSim (%)'] = 100 *
                ↪ (combined_data['PR_NeqSim Density (kg/m^3)'] -
                ↪ combined_data['Experimental Density (kg/m^3)']) /
                ↪ combined_data['Experimental Density (kg/m^3)']
231         combined_data['ARD GERG (%)'] = 100 * (combined_data['GERG
                ↪ Density (kg/m^3)'] - combined_data['Experimental Density
                ↪ (kg/m^3)']) / combined_data['Experimental Density
                ↪ (kg/m^3)']
232         combined_data['ARD SRK_HYSYS (%)'] = 100 *
                ↪ (combined_data['SRK_HYSYS'] - combined_data['Experimental
                ↪ Density (kg/m^3)']) / combined_data['Experimental Density
                ↪ (kg/m^3)']
233         combined_data['ARD PR_HYSYS (%)'] = 100 *
                ↪ (combined_data['PR_HYSYS'] - combined_data['Experimental
                ↪ Density (kg/m^3)']) / combined_data['Experimental Density
                ↪ (kg/m^3)']
234
235         # Calculate mean and maximum absolute relative deviations for
                ↪ each model
236         statistics = {
237             'SRK_NeqSim': {
238                 'mean': combined_data['ARD SRK_NeqSim (%)'].mean(),
239                 'max': combined_data['ARD SRK_NeqSim (%)'].abs().max()
240             },
241             'PR_NeqSim': {
242                 'mean': combined_data['ARD PR_NeqSim (%)'].mean(),
243                 'max': combined_data['ARD PR_NeqSim (%)'].abs().max()
244             },
245             'GERG': {
246                 'mean': combined_data['ARD GERG (%)'].mean(),
247                 'max': combined_data['ARD GERG (%)'].abs().max()
248             },
249             'SRK_HYSYS': {
250                 'mean': combined_data['ARD SRK_HYSYS (%)'].mean(),
```

```
                'max': combined_data['ARD SRK_HYSYS (%)'].abs().max()
            },
            'PR_HYSYS': {
                'mean': combined_data['ARD PR_HYSYS (%)'].mean(),
                'max': combined_data['ARD PR_HYSYS (%)'].abs().max()
            }
        }

        # Print results for this molar fraction
        print(f"Results for Molefrac_H2 = {mole_H2:.2f}")
        for model, stats in statistics.items():
            print(f"  {model} - Mean ARD (%): {stats['mean']:.6f}, Max
                ↪ ARD (%): {stats['max']:.6f}")
        print("-" * 50)


ard_statistics_single_composition(plot_data)




EoS = ["srk", "pr", "GERG2008"]
components = ["hydrogen", "CO2"]
mole_frac = [0.05362, 0.94638]

# Temperaure andpressure for the study
T_values = [273.15, 293.15, 323.15, 293.15]
p_1 =  [3.03596, 2.53690, 2.02237, 1.52462, 1.03957, 0.51352]
p_2 = [4.98489, 4.05114, 3.01670, 1.99199, 1.01460, 0.50316]
p_3 = [5.99737, 5.03044, 4.02943, 3.00975, 2.03106, 1.03812, 0.54921]
p_4 = [
    5.02447, 5.02513, 5.02570, 5.02620, 5.02693, 5.02715,
    4.99077, 4.99075, 4.05062, 4.05114,
    3.02162, 3.02246, 3.02304,
    2.03865, 2.03947, 2.05638, 2.05656,
    1.99173, 1.99199, 1.03389, 1.03446, 1.03503,
    0.52907, 0.52952, 0.53697, 0.53728, 0.50299, 0.50316
]

pressures = [p_1, p_2, p_3, p_4]

EOSS = ["srk", "pr", "GERG2008"]

# To store pressure and density data
density_data = {EOS: [] for EOS in EOSS}

for EOS in EOSS:
    density_data[EOS].append(generate_data(EOS, components, mole_frac,
        ↪ pressures, T_values))

# Save the data to an Excel file
#save_data_to_excel(pressures, density_data, output_file, T_values,
    ↪ EOSS)
```

B.5 Code for the study by Hernández-Gómez et al.[15]

Code written in python for calculating the density data and comparing them to the experimental data. The code contains functions for gathering the states for that study, to generate the density data and functions for plotting and statistics.

```
from neqsim.thermo import TPflash, fluid
```

```
import pandas as pd
import matplotlib.pyplot as plt
import os
import matplotlib
matplotlib.use('Agg')

# Function to extract temperature and pressure data from an Excel file

def get_temp_pressure(filepath, sheet_name, T_col, P_col, skip_rows):
    temperature_list = pd.read_excel(filepath, sheet_name=sheet_name, skiprows=skip_rows, usecols=T_col).iloc[:, 0]
    pressure_list = pd.read_excel(filepath, sheet_name=sheet_name, skiprows=skip_rows, usecols=P_col).iloc[:, 0]

    # Combine into a DataFrame and drop rows with NaN
    temp_pressure_df = pd.DataFrame({'Temperature': temperature_list, 'Pressure': pressure_list}).dropna()

    # Convert columns back to lists
    temperature_list = temp_pressure_df['Temperature'].tolist()
    pressure_list = temp_pressure_df['Pressure'].tolist()

    return temperature_list, pressure_list

# Function to generate density data for a given fluid type, components, and mole fractions using temperature and pressure from an Excel file

def generate_data(fluid_type, components, mole_frac, filepath, sheet_name, T_col, P_col, skip_rows):
    data = []
    if fluid_type == "GERG2008":
        fluid1 = fluid("pr")
    else:
        fluid1 = fluid(fluid_type)

    for i in range(len(components)):
        fluid1.addComponent(components[i], mole_frac[i], "mol/sec")

    T_values, P_values = get_temp_pressure(filepath, sheet_name, T_col, P_col, skip_rows)

    for i in range(len(T_values)):
        fluid1.setTemperature(T_values[i], "K")
        fluid1.setPressure(P_values[i], "MPa")
        fluid1.setMixingRule(2)
        TPflash(fluid1)
        fluid1.initThermoProperties()

        # Get density depending on the EOS
        if fluid_type == "GERG2008":
            if fluid1.getNumberOfPhases() != 1:
                raise RuntimeError("Too many phases detected, stopping the code.")
            density = fluid1.getPhase("gas").getDensity_GERG2008()
        else:
            density = fluid1.getDensity()

        data.append({"Temperature": T_values[i], "Pressure": P_values[i], f"Density_{fluid_type}": density})

    return pd.DataFrame(data)

```

```
# Function to save combined density data for different EOS into an Excel file

def save_data_to_excel(density_data, output_file):
    # Rename columns in each DataFrame to avoid conflicts
    for key, df in density_data.items():
        df.columns = [f"{col}_{key}" if col not in ['Temperature', 'Pressure'] else col for col in df.columns]

    # Concatenate all data along columns
    result_df = pd.concat(density_data.values(), axis=1)

    # Drop duplicate 'Temperature' and 'Pressure' columns, keeping the first occurrences
    result_df = result_df.loc[:, ~result_df.columns.duplicated()]

    # Save to Excel
    with pd.ExcelWriter(output_file, engine='xlsxwriter') as writer:
        result_df.to_excel(writer, sheet_name="Density_Data", index=False)


# Paths and parameters
file_path = r"C:\Users\victo\OneDrive - NTNU\5. r \Fordypningsprosjekt\Python\Eksperimentelle data.xlsx"
output_file = r"C:\Users\victo\OneDrive - NTNU\5. r \Fordypningsprosjekt\Python\density_results_2.xlsx"
EOSS = ["srk", "pr", "GERG2008"]
components = ["methane", "ethane", "propane", "n-butane", "i-butane", "n-pentane", "i-pentane", "i-pentane",
              "n-hexane", "CO2", "nitrogen", "helium", "hydrogen"]
mole_frac = [78.821237, 0.757359, 0.297078, 0.200439, 0.197953, 0.050134, 0.049928, 0.049615,
             0.050708, 4.001075, 12.017829, 0.496897, 3.009733]

skip_rows = 5
T_col = "E"
P_col = "F"
sheet_name = "Ark 4"

# Generate density data for each EOS and store in a dictionary
density_data = {}
for EOS in EOSS:
    density_data[EOS] = generate_data(EOS, components, mole_frac, file_path, sheet_name, T_col, P_col, skip_rows)

# Save the combined data to Excel

#save_data_to_excel(density_data, output_file)


# Function to create grouped ARD (Absolute Relative Deviation) plots for different EOS models

def grouped_ard_plot(file_path):

    # Load the single sheet from the Excel file
    data = pd.read_excel(file_path)

    # Drop rows with any missing values and make a copy to avoid SettingWithCopyWarning
    data = data.dropna().copy()

```

```
    # Rename columns for clarity
    data.columns = ['Temperature (K)', 'Pressure (MPa)', 'Experimental Density (kg/m^3)',
                    'SRK_NeqSim Density (kg/m^3)', 'PR_NeqSim Density (kg/m^3)', 'GERG Density (kg/m^3)',
                    'SRK_HYSYS', 'PR_HYSYS']

    # Convert columns to numeric in case they are read as strings
    for col in ['Pressure (MPa)', 'Experimental Density (kg/m^3)', 'SRK_NeqSim Density (kg/m^3)',
                'PR_NeqSim Density (kg/m^3)', 'GERG Density (kg/m^3)', 'SRK_HYSYS', 'PR_HYSYS']:
        data[col] = pd.to_numeric(data[col], errors='coerce')

    # Round temperatures to the nearest whole number
    data['Temperature (K)'] = data['Temperature (K)'].round()

    # Calculate relative deviations as percentages for each density model
    data['ARD SRK_NeqSim (%)'] = 100 * (data['SRK_NeqSim Density (kg/m^3)'] - data['Experimental Density (kg/m^3)']) / data['Experimental Density (kg/m^3)']
    data['ARD PR_NeqSim (%)'] = 100 * (data['PR_NeqSim Density (kg/m^3)'] - data['Experimental Density (kg/m^3)']) / data['Experimental Density (kg/m^3)']
    data['ARD GERG (%)'] = 100 * (data['GERG Density (kg/m^3)'] - data['Experimental Density (kg/m^3)']) / data['Experimental Density (kg/m^3)']
    data['ARD SRK_HYSYS (%)'] = 100 * (data['SRK_HYSYS'] - data['Experimental Density (kg/m^3)']) / data['Experimental Density (kg/m^3)']
    data['ARD PR_HYSYS (%)'] = 100 * (data['PR_HYSYS'] - data['Experimental Density (kg/m^3)']) / data['Experimental Density (kg/m^3)']

    # Group data by each unique temperature (isotherm)
    unique_temperatures = sorted(data['Temperature (K)'].unique())

    # Define the figure and subplot structure
    fig = plt.figure(figsize=(20, 8))
    axes = [
        plt.subplot2grid((2, 9), (0, 1), colspan=2),
        plt.subplot2grid((2, 9), (0, 3), colspan=2),
        plt.subplot2grid((2, 9), (0, 5), colspan=2),
        plt.subplot2grid((2, 9), (1, 2), colspan=2),
        plt.subplot2grid((2, 9), (1, 4), colspan=2)
    ]

    handles = []
    labels = []

    # Loop through each unique temperature and its corresponding axis
    for ax, temperature in zip(axes, unique_temperatures[:len(axes)]):  # Limit to the number of defined axes
        # Filter the data for the current temperature
        temp_data = data[data['Temperature (K)'] == temperature]

        # Plot ARD values for the current temperature
        for model, marker, label in zip(
            ['ARD SRK_NeqSim (%)', 'ARD PR_NeqSim (%)', 'ARD GERG (%)', 'ARD SRK_HYSYS (%)', 'ARD PR_HYSYS (%)'],
            ['x', '^', 's', 'o', 'D'],
```

```
153             ['SRK NeqSim', 'PR NeqSim', 'GERG-2008', 'SRK HYSYS', 'PR
                    ↪ HYSYS']
154         ):
155             scatter = ax.scatter(
156                 temp_data['Pressure (MPa)'], temp_data[model],
157                 label=label, marker=marker, color="black", s=30,
158                 facecolors='none' if marker in ['^', 's', 'o', 'D']
                        ↪ else None
159             )
160             # Collect unique legend entries
161             if label not in labels:
162                 labels.append(label)
163                 handles.append(scatter)
164 
165         # Annotate the temperature above each plot
166         ax.set_title(f"T = {temperature} K", fontsize=14)
167         ax.grid(True)
168         ax.set_ylim(-5, 5)
169         ax.set_xlabel("Pressure (MPa)")
170 
171     # Add a shared legend outside the plots
172     fig.legend(handles, labels, loc='upper center',
            ↪ bbox_to_anchor=(0.45, 0), ncol=5, frameon=False, fontsize=16)
173     fig.text(0.09, 0.5, r'$100 \left(\rho_{\mathrm{model}} -
            ↪ \rho_{\mathrm{experimental}}\right)/\rho_{\mathrm{experimental}}$',
174              ha='center', va='center', rotation='vertical', fontsize=16)
175 
176     # Save the grouped plot
177     save_path = os.path.join("C:\\Users\\victo\\OneDrive - NTNU\\5.
            ↪  r \\Fordypningsprosjekt\\Python\\Ark_4\\plots",
178                              "Grouped_Relative_Deviation.png")
179     plt.tight_layout()
180     plt.savefig(save_path, dpi=300, bbox_inches='tight')
181     plt.close()
182 
183 file_path = r"C:\Users\victo\OneDrive - NTNU\5.
        ↪  r \Fordypningsprosjekt\Python\Ark_4\Plot_data_Ark_4.xlsx"
184 #grouped_ard_plot(file_path)
185 
186 
187 # Function to calculate and print overall mean and maximum ARD for
        ↪ different EOS models
188 
189 def grouped_ard_statistics(file_path):
190 
191 
192     # Load the single sheet from the Excel file
193     data = pd.read_excel(file_path)
194 
195     # Drop rows with any missing values and make a copy to avoid
            ↪ SettingWithCopyWarning
196     data = data.dropna().copy()
197 
198     # Rename columns for clarity
199     data.columns = ['Temperature (K)', 'Pressure (MPa)', 'Experimental
            ↪ Density (kg/m^3)',
200                     'SRK_NeqSim Density (kg/m^3)', 'PR_NeqSim Density
                            ↪ (kg/m^3)', 'GERG Density (kg/m^3)',
201                     'SRK_HYSYS', 'PR_HYSYS']
202 
203     # Convert columns to numeric in case they are read as strings
204     for col in ['Pressure (MPa)', 'Experimental Density (kg/m^3)',
            ↪ 'SRK_NeqSim Density (kg/m^3)',
```

```
                'PR_NeqSim Density (kg/m^3)', 'GERG Density (kg/m^3)',
                    ↪ 'SRK_HYSYS', 'PR_HYSYS']:
        data[col] = pd.to_numeric(data[col], errors='coerce')

    # Calculate relative deviations as percentages for each density
        ↪ model
    data['ARD SRK_NeqSim (%)'] = 100 * (data['SRK_NeqSim Density
        ↪ (kg/m^3)'] - data['Experimental Density (kg/m^3)']) /
        ↪ data['Experimental Density (kg/m^3)']
    data['ARD PR_NeqSim (%)'] = 100 * (data['PR_NeqSim Density
        ↪ (kg/m^3)'] - data['Experimental Density (kg/m^3)']) /
        ↪ data['Experimental Density (kg/m^3)']
    data['ARD GERG (%)'] = 100 * (data['GERG Density (kg/m^3)'] -
        ↪ data['Experimental Density (kg/m^3)']) / data['Experimental
        ↪ Density (kg/m^3)']
    data['ARD SRK_HYSYS (%)'] = 100 * (data['SRK_HYSYS'] -
        ↪ data['Experimental Density (kg/m^3)']) / data['Experimental
        ↪ Density (kg/m^3)']
    data['ARD PR_HYSYS (%)'] = 100 * (data['PR_HYSYS'] -
        ↪ data['Experimental Density (kg/m^3)']) / data['Experimental
        ↪ Density (kg/m^3)']

    # Calculate overall mean and maximum absolute relative deviations
        ↪ for each model
    statistics = {
        'SRK_NeqSim': {
            'mean': data['ARD SRK_NeqSim (%)'].mean(),
            'max': data['ARD SRK_NeqSim (%)'].abs().max()
        },
        'PR_NeqSim': {
            'mean': data['ARD PR_NeqSim (%)'].mean(),
            'max': data['ARD PR_NeqSim (%)'].abs().max()
        },
        'GERG': {
            'mean': data['ARD GERG (%)'].mean(),
            'max': data['ARD GERG (%)'].abs().max()
        },
        'SRK_HYSYS': {
            'mean': data['ARD SRK_HYSYS (%)'].mean(),
            'max': data['ARD SRK_HYSYS (%)'].abs().max()
        },
        'PR_HYSYS': {
            'mean': data['ARD PR_HYSYS (%)'].mean(),
            'max': data['ARD PR_HYSYS (%)'].abs().max()
        }
    }

    # Print the results
    print("Overall Results:")
    for model, stats in statistics.items():
        print(f"  {model} - Mean ARD (%): {stats['mean']:.5f}, Max ARD
            ↪ (%): {stats['max']:.5f}")


grouped_ard_statistics(file_path)
```

Appendix C

Binary Interaction Coefficients

The Tables in this sections hows the the Binary Interaction Coefficients (BIC) for the implementations of SRK and PR in NeqSim and HYSYS.

C.1 Binary Interaction Coefficients: NeqSim, SRK

	methane	ethane	propane	i-butane	n-butane	i-pentane	n-pentane	n-hexane	n-heptane	n-octane	n-nonane	nC10	benzene	toluene	o-Xylene	CO2	nitrogen	oxygen	helium	hydrogen
methane		0,00295295	0,00747722	0,01369935	0,01289789	0,01817552	0,01847102	0,023474067	0,0307	0,044799998	0,044799998	0,041099999	0,0209	0		0,0973	0,0319	0	0	0,01
ethane	0,00295295		0,00185286	0,00511365	0,00464288	0,00791956	0,00811359	0	0				0,0289	0		0,1346	0,0388	0	0	0,0367
propane	0,00747722	0,00185286		0,00153851	0,00132268	0,00305075	0,00316586	0	0				0,02	0		0,1018	0,0886	0	0	0,0799
i-butane	0,01369935	0,00511365	0,00153851		0,00047251		0,00081462		0							0,13680001	0,13150001	0		
n-butane	0,01289789	0,00464288	0,00132268	0,00047251	0	0,00092632	0,00097665	0	0				0	0		0,1474	0,1007	0	0	0,1446
i-pentane	0,01817552	0,00791956	0,00305075		0,00092632		0,00039999		0							0,12970001	0,093	0		
n-pentane	0,01847102	0,00811359	0,00316586	0,00081462	0,00097665	0,00039999	0	0	0				0	0		0,1278	0,093598	0	0	0,2166
n-hexane	0,023474067	0	0		0		0		0							0,12	0,08	0		
n-heptane	0,0307	0	0	0	0	0	0	0					0	0		0,109999991	0,142199993	0	0	0,2166
n-octane	0,044799998															0,100000001	0,079999998			
n-nonane	0,044799998															0,100000001	0,079999998			
nC10	0,041099999															0,130400002	0,1293			
benzene	0,0209	0,0289	0,02		0		0		0							0,076700001	0,2131	0		
toluene	0	0	0		0		0		0							0	0	0		
o-Xylene																				
CO2	0,0973	0,1346	0,1018	0,13680001	0,14120001	0,12970001	0,1278	0,12	0,12	0,100000001	0,100000001	0,130400002	0,076700001	0			-0,0171	0	0	0,0362
nitrogen	0,0319	0,0388	0,0886	0,13150001	0,0597	0,093	0,093598	0,08	0,08	0,079999998	0,079999998	0,1293	0,2131	0		-0,0171		-0,0078	0	0,0071
oxygen	0	0	0	0	0	0	0	0	0				0	0		0	-0,0078	0	0	0,0079
helium	0	0	0		0		0		0							0	0	0		
hydrogen	0,01	0,0367	0,0799		0,1446		0,2166		0,2166							0,0362	0,0071	0,0079		

C.2 Binary Interaction Coefficients: HYSYS, SRK

	Methane	Ethane	Propane	i-Butane	n-Butane	i-Pentane	n-Pentane	n-Hexane	n-Heptane	n-Octane	n-Nonane	n-Decane	Benzene	Toluene	o-Xylene	CO2	Nitrogen	Oxygen	Helium	Hydrogen
Methane	---	0,002241	0,006829	0,013113	0,012305	0,017627	0,017925	0,023474	0,028864	0,034159	0,038926	0,043609	0,012809	0,054698	0,023581	0,095600	0,031199	-	0,750000	1,00E-04
Ethane	0,002241	---	0,001258	0,004573	0,004096	0,007413	0,007609	0,011414	0,015324	0,019319	0,023017	0,026730	0,028000	0,029998	0,011489	0,140100	0,031899	-	1,406900	1,00E-04
Propane	0,006829	0,001258	---	0,001041	0,000819	0,002583	0,002701	0,005142	0,007887	0,010850	0,013697	0,016630	0,019997	0,014990	0,005193	0,136800	0,088600	-	1,249300	1,00E-04
i-Butane	0,013113	0,004573	0,001041	---	0,000013	0,000346	0,000390	0,001565	0,003221	0,005214	0,007255	0,009448	0,000002	0,009999	0,001594	0,136800	0,131500	-	-	2,00E-04
n-Butane	0,012305	0,004096	0,000819	0,000013	---	0,000495	0,000547	0,001866	0,003646	0,005750	0,007883	0,010161	0,000005	0,009999	0,001897	0,141200	0,059700	-	-	2,00E-04
i-Pentane	0,017627	0,007413	0,002583	0,000346	0,000495	---	0,000001	0,000440	0,001459	0,002883	0,004449	0,006205	0,000399	0,004999	0,000455	0,129700	0,093000	-	-	2,00E-04
n-Pentane	0,017925	0,007609	0,002701	0,000390	0,000547	0,000001	---	0,000393	0,001373	0,002762	0,004299	0,006028	0,020997	0,004999	0,000408	0,134700	0,093598	-	-	2,00E-04
n-Hexane	0,023474	0,011414	0,005142	0,001565	0,001866	0,000440	0,000393	---	0,000297	0,001073	0,002098	0,003354	0,010998	0,000322	0,000000	0,142000	0,165000	-	-	2,00E-04
n-Heptane	0,028864	0,015324	0,007887	0,003221	0,003646	0,001459	0,001373	0,000297	---	0,000241	0,000818	0,001659	0,002000	0,008000	0,000285	0,109200	0,079989	-	-	0
n-Octane	0,034159	0,019319	0,010850	0,005214	0,005750	0,002883	0,002762	0,001073	0,000241	---	0,000171	0,000636	0,004000	0,002568	0,001050	0,135000	0,079989	-	-	0
n-Nonane	0,038926	0,023017	0,013697	0,007255	0,007883	0,004449	0,004299	0,002098	0,000818	0,000171	---	0,000148	0,007486	0,004999	0,002066	0,135000	0,079989	-	-	0
n-Decane	0,043609	0,026730	0,016630	0,009448	0,010161	0,006205	0,006028	0,003354	0,001659	0,000636	0,000148	---	0,009711	0,009999	0,003313	0,133900	0,127900	-	-	0
Benzene	0,012809	0,028000	0,019997	0,000002	0,000005	0,000399	0,020997	0,010998	0,002000	0,004000	0,007486	0,009711	---	0,000528	0,001704	0,080499	0,100000	-	-	0
Toluene	0,054698	0,029998	0,014990	0,009999	0,009999	0,004999	0,004999	0,000322	0,008000	0,002568	0,004999	0,009999	0,000528	---	0,000335	0,098200	0,180000	-	-	0
o-Xylene	0,023581	0,011489	0,005193	0,001594	0,001897	0,000455	0,000408	0,000000	0,000285	0,001050	0,002066	0,003313	0,001704	0,000335	---	0,108900	0,100000	-	-	0
CO2	0,095600	0,140100	0,136800	0,136800	0,141200	0,129700	0,134700	0,142000	0,109200	0,135000	0,135000	0,133900	0,080499	0,098200	0,108900	---	-0,017100	0,097500	0,916100	0,1164
Nitrogen	0,031199	0,031899	0,088600	0,131500	0,059700	0,093000	0,093598	0,165000	0,079989	0,079989	0,079989	0,127900	0,100000	0,180000	0,100000	-0,017100	---	-0,014000	0,094400	-1,00E-03
Oxygen	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0,097500	-0,014000	---	-	0
Helium	0,750000	1,406900	1,249300	-	-	-	-	-	-	-	-	-	-	-	-	0,916100	0,094400	-	---	0,3993
Hydrogen	1,00E-04	1,00E-04	1,00E-04	2,00E-04	2,00E-04	2,00E-04	2,00E-04	2,00E-04	0	0	0	0	0	0	0	0,1164	-1,00E-03	0	0,3993	---

C.3 Binary Interaction Coefficients: NeqSim, PR

	methane	ethane	propane	i-butane	n-butane	i-pentane	n-pentane	n-hexane	n-heptane	n-octane	n-nonane	nC10	benzene	toluene	o-Xylene	CO2	nitrogen	oxygen	helium	hydrogen
methane		0,00295295	0,00747722	0,01369935	0,01289789	0,01817552	0,01847102	0,023474067	0	0,044799998	0,044799998	0,041099999	0	0		0,0973	0,0319	0	0	0,01
ethane	0,00295295		0,00185286	0,00511365	0,00464288	0,00791956	0,00811359	0	0				0	0		0,12980001	0,0388	0	0	0,0367
propane	0,00747722	0,00185286		0,00153851	0,00132268	0,00305075	0,00316586	0	0				0,02	0		0,13500001	0,079998	0	0	0,0799
i-butane	0,01369935	0,00511365	0,00153851		0,00047251		0,00081462		0							0,12980001	0,094999	0		
n-butane	0,01289789	0,00464288	0,00132268	0,00047251	0	0,00092632	0,00097665	0	0				0	0		0,12980001	0,1007	0	0	0,1446
i-pentane	0,01817552	0,00791956	0,00305075		0,00092632		0,00039999		0							0,125	0,094999	0		
n-pentane	0,01847102	0,00811359	0,00316586	0,00081462	0,00097665	0,00039999	0	0	0				0	0		0,125	0,1	0	0	0,2166
n-hexane	0,023474067	0	0		0		0		0							0,12	0,08	0		
n-heptane	0	0	0	0	0	0	0	0					0	0		0,1	0,142199993	0	0	0,2166
n-octane	0,044799998															0,100000001	0,079999998			
n-nonane	0,044799998															0,100000001	0,079999998			
nC10	0,041099999															0,130400002	0,1293			
benzene	0	0	0,02		0		0		0							0,081	0,2131	0		
toluene	0	0	0		0		0		0							0	0	0		
o-Xylene																				
CO2	0,0973	0,12980001	0,13500001	0,12980001	0,12980001	0,125	0,125	0,12	0,12	0,100000001	0,100000001	0,130400002	0,081	0			-0,019997	0	0	0,0362
nitrogen	0,0319	0,0388	0,079998	0,094999	0,09	0,094999	0,1	0,08	0,08	0,079999998	0,079999998	0,1293	0,2131	0		-0,019997		-0,0078	0	0,0071
oxygen	0	0	0	0	0	0	0	0	0				0	0		0	-0,0078	0	0	0,0079
helium	0	0	0		0		0		0							0	0	0		
hydrogen	0,01	0,0367	0,0799		0,1446		0,2166		0,2166							0,0362	0,0071	0,0079		

C.4 Binary Interaction Coefficients: HYSYS, PR

	Methane	Ethane	Propane	i-Butane	n-Butane	i-Pentane	n-Pentane	n-Hexane	n-Heptane	n-Octane	n-Nonane	n-Decane	Benzene	Toluene	o-Xylene	CO2	Nitrogen	Oxygen	Helium	Hydrogen
Methane	---	0,0022414	0,0068288	0,0131134	0,0123046	0,0176275	0,0179254	0,0234740	0,0288643	0,0341591	0,0389260	0,0436087	0,0399990	0,0649000	0,0500000	0,1000000	0,0359990	-	0,7649000	0,202
Ethane	0,0022414	---	0,0012580	0,0045735	0,0040964	0,0074133	0,0076094	0,0114138	0,0153243	0,0193187	0,0230174	0,0267295	0,0199970	0,0344000	0,0329990	0,1298000	0,0500000	-	1,1232001	0,2231
Propane	0,0068288	0,0012580	---	0,0010405	0,0008190	0,0025834	0,0027005	0,0051420	0,0078874	0,0108503	0,0136966	0,0166296	0,0199970	0,0309980	0,0299980	0,1350000	0,0799980	-	1,0642000	0,2142
i-Butane	0,0131134	0,0045735	0,0010405	---	0,0000134	0,0003462	0,0003901	0,0015653	0,0032213	0,0052142	0,0072549	0,0094483	0,0000018	0,0004679	0,0015938	0,1298000	0,0949990	-	-	0,2037
n-Butane	0,0123046	0,0040964	0,0008190	0,0000134	---	0,0004951	0,0005472	0,0018663	0,0036464	0,0057502	0,0078831	0,0101611	0,0000052	0,0006388	0,0018973	0,1298000	0,0900000	-	-	0,1941
i-Pentane	0,0176275	0,0074133	0,0025834	0,0003462	0,0004951	---	0,0000013	0,0004399	0,0014591	0,0028828	0,0044489	0,0062051	0,0003986	0,0000091	0,0004551	0,1250000	0,0949990	-	-	0,2921
n-Pentane	0,0179254	0,0076094	0,0027005	0,0003901	0,0005472	0,0000013	---	0,0003935	0,0013733	0,0027618	0,0042986	0,0060278	0,0160000	0,0000036	0,0004078	0,1250000	0,1000000	-	-	0,2921
n-Hexane	0,0234740	0,0114138	0,0051420	0,0015653	0,0018663	0,0004399	0,0003935	---	0,0002972	0,0010733	0,0020981	0,0033543	0,0070000	0,0003223	0,0000002	0,1250000	0,1490000	-	-	0,2921
n-Heptane	0,0288643	0,0153243	0,0078874	0,0032213	0,0036464	0,0014591	0,0013733	0,0002972	---	0,0002413	0,0008175	0,0016586	-0,0020000	0,0060000	0,0002850	0,1199000	0,1439000	-	-	-0,1167
n-Octane	0,0341591	0,0193187	0,0108503	0,0052142	0,0057502	0,0028828	0,0027618	0,0010733	0,0002413	---	0,0001707	0,0006357	0,0030000	0,0099999	0,0010501	0,1150000	0,1000000	-	-	0,2921
n-Nonane	0,0389260	0,0230174	0,0136966	0,0072549	0,0078831	0,0044489	0,0042986	0,0020981	0,0008175	0,0001707	---	0,0001477	0,0074857	0,0040565	0,0020655	0,1010000	0,1000000	-	-	0,2921
n-Decane	0,0436087	0,0267295	0,0166296	0,0094483	0,0101611	0,0062051	0,0060278	0,0033543	0,0016586	0,0006357	0,0001477	---	0,0099999	0,0099999	0,0033131	0,1181000	0,1316000	-	-	0,2921
Benzene	0,0399990	0,0199970	0,0199970	0,0000018	0,0000052	0,0003986	0,0160000	0,0070000	-0,0020000	0,0030000	0,0074857	0,0099999	---	0,0005285	0,0017036	0,0806000	0,1597000	-	-	0,2851
Toluene	0,0649000	0,0344000	0,0309980	0,0004679	0,0006388	0,0000091	0,0000036	0,0003223	0,0060000	0,0099999	0,0040565	0,0099999	0,0005285	---	0,0003353	0,0935980	0,1932000	-	-	0,2851
o-Xylene	0,0500000	0,0329990	0,0299980	0,0015938	0,0018973	0,0004551	0,0004078	0,0000002	0,0002850	0,0010501	0,0020655	0,0033131	0,0017036	0,0003353	---	0,0900000	0,2140000	-	-	0,2921
CO2	0,1000000	0,1298000	0,1350000	0,1298000	0,1298000	0,1250000	0,1250000	0,1250000	0,1199000	0,1150000	0,1010000	0,1181000	0,0806000	0,0935980	0,0900000	---	-0,0199970	0,0974970	0,7967000	0,1202
Nitrogen	0,0359990	0,0500000	0,0799980	0,0949990	0,0900000	0,0949990	0,1000000	0,1490000	0,1439000	0,1000000	0,1000000	0,1316000	0,1597000	0,1932000	0,2140000	-0,0199970	---	-0,0120000	0,0684980	-3,60E-02
Oxygen	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0,0974970	-0,0120000	---	-	0
Helium	0,7649000	1,1232001	1,0642000	-	-	-	-	-	-	-	-	-	-	-	-	0,7967000	0,0684980	-	---	0,4058
Hydrogen	0,202000007	0,22310001	0,2142	0,20370001	0,19410001	0,29210001	0,29210001	0,29210001	-0,1167	0,29210001	0,29210001	0,29210001	0,28510001	0,28510001	0,29210001	0,1202	-3,60E-02	0	0,40580001	---

Appendix D

Python Code for Viscosity

D.1 Experimental Data

Python code with the collected experimental data for viscosity.

```
import pandas as pd

'''
components = [
    "methane",  #  C H
    "ethane",  #   C  H
    "propane",  #   C  H
    "n-butane",  # n-   C   H
    "i-butane",  # iso-   C   H
    "n-pentane",  # n-   C   H
    "i-pentane",  # iso-   C   H
    "n-pentane",  # neo-   C   H
    "n-hexane",  # n-   C   H
    "n-heptane",  # n-   C   H
    "n-octane",  # n-   C   H
    "n-nonane",  # n-   C   H
    "nC10",  # n-    C    H
    "benzene",  #   C  H
    "toluene",  #   C  H
    #"C8",  # o-   C   H
    "CO2",  #  C O
    "nitrogen",  #  N
    "oxygen",  #  O
    "helium",  # He
    "hydrogen"  #  H
]
'''

'''
Experimental data setup:
exp_data = [[mole fractions (composition)],
            [Temperatures (K)],
            Pressures (MPa)],
            [viscosity ( Pa *s)],
            ["Authors of study", plotting symbol, plotting symbol]]
'''
viscosity_file = pd.ExcelFile(r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity Data (exp).xlsx")

def expData(dataset):
    data_dict = {
        ### Ark 1 ###
        # Data from
            ↪ https://link.springer.com/article/10.1007/s10765-024-03394-4,
            ↪ "Viscosity of Hydrogen and Methane Blends: Experimental
            ↪ and Modelling Investigations"
        #Pure Hydrogen:
        "Ark1_1" : [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
                    pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                        ↪ 1", usecols = "A", skiprows =
                        ↪ 6).squeeze().dropna()).tolist(),
```

```
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "B", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "D", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        ["Owuna et al.", "^", "s"]],

    #Pure Methane:
    "Ark1_2" : [[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "J", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "K", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "M", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        ["Owuna et al.", "^", "s"]],

    #0.9CH4 + 0.1H2:
    "Ark1_3" : [[0.9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.1],
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "S", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "T", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "V", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        ["Owuna et al.", "^", "s"]],

    #0.8CH4 + 0.2H2:
    "Ark1_4" : [[0.8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.2],
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "AI", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "AJ", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "AL", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        ["Owuna et al.", "^", "s"]],

    #0.5CH4 + 0.5H2:
    "Ark1_5" : [[0.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.5],
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "AZ", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "BA", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 1", usecols = "BC", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        ["Owuna et al.", "^", "s"]],


    ### Ark 2 ###
```

```
80          # Data from https://doi.org/10.1007/s10765-023-03328-6,
                ↪ "Measurements of the Viscosity of Hydrogen and a
                ↪ (Hydrogen+Methane) Mixture with a  T w o Capillary
                ↪ Viscometer"
81          #Pure hydrogen:
82          "Ark2_1" : [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
83                      pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 2", usecols = "A", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
84                      pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 2", usecols = "B", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
85                      pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 2", usecols = "D", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
86                      ["Betken et al.", "d", "X"]],
87
88          #0.1CH4 + 0.9H2:
89          "Ark2_2" : [[0.1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.9],
90                      pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 2", usecols = "H", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
91                      pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 2", usecols = "I", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
92                      pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 2", usecols = "J", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
93                      ["Betken et al.", "d", "X"]],
94
95
96
97          ### Ark 3 ###
98          # Data from https://pubs.acs.org/doi/abs/10.1021/je60071a010,
                ↪ "Viscosity of methane, hydrogen, and four mixtures of
                ↪ methane and hydrogen from -100.degree.C to 0.degree.C at
                ↪ high pressures"
99          #Pure hydrogen:
100         "Ark3_1" : [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
101                     pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "U", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
102                     pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "V", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
103                     pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "W", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
104                     ["Chuang et al.","o", "P"]],
105
106         #Pure methane:
107         "Ark3_2" : [[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
108                     pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "A", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
109                     pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "B", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
110                     pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "C", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
111                     ["Chuang et al.","o", "P"]],
112
```

```
        #0.8058CH4 + 0.1942H2:
        "Ark3_3" : [[0.8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.2],
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "E", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "F", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "G", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        ["Chuang et al.","o", "P"]],

        #0.6625CH4 + 0.3375H2:
        "Ark3_4" : [[0.6625,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.3375],
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "I", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "J", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "K", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        ["Chuang et al.","o", "P"]],

        #0.4663CH4 + 0.5337H2:
        "Ark3_5" : [[0.4663,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.5337],
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "M", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "N", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "O", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        ["Chuang et al.","o", "P"]],

        #0.213CH4 + 0.787H2:
        "Ark3_6" : [[0.213,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.787],
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "Q", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "R", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        pd.to_numeric(viscosity_file.parse(sheet_name="Ark
                            ↪ 3", usecols = "S", skiprows =
                            ↪ 6).squeeze().dropna()).tolist(),
                        ["Chuang et al.","o", "P"]]
        }

    # Return the requested data if it exists
    if dataset in data_dict:
        return data_dict[dataset]
    else:
        raise ValueError(f"Dataset '{dataset}' not found. Please check
            ↪ the dataset name.")
```

D.2 Viscosity Calculation

Python code for calculating viscosity of mixtures at several pressures and temperatures.

```
from neqsim.thermo import TPflash, fluid


def getDensity(EOS, componentNames, componentFrac, T, p):
    if EOS == "GERG2008":
        fluid1 = fluid("pr")
    else:
        fluid1 = fluid(EOS)

    for i in range(len(componentNames)):
        fluid1.addComponent(componentNames[i], componentFrac[i],
            ↪ "mol/sec")     #make a fluid object

    fluid1.setMixingRule("classic")     #Define a mixing rule
    fluid1.setTemperature(T, "K")
    fluid1.setPressure(p, "MPa")
    TPflash(fluid1)
    fluid1.initProperties()             #Calculate thermodynamic- and
        ↪ fluid properties

    if EOS == "GERG2008":
        if fluid1.getNumberOfPhases() != 1:
            phase_names = [fluid1.getPhase(phase).getPhaseTypeName()
                ↪ for phase in range(fluid1.getNumberOfPhases())]
            print(f"Error: More than one phase detected. Number of
                ↪ phases: {fluid1.getNumberOfPhases()}")
            print(f"Phases detected: {', '.join(phase_names)}")
            raise RuntimeError("Too many phases detected, stopping the
                ↪ code.")
        density = fluid1.getPhase("gas").getDensity_GERG2008()
    else:
        density = fluid1.getDensity()
    return density


#Viscosity models that work in Neqsim:
#'friction theory'
#'LBC'
#'PFCT' (not for pure Hydrogen)


def getVisc(EOS, componentNames, componentFrac, T, p, method):
    # Ensure T is a single value
    if isinstance(T, list):
        raise ValueError("T should be a single value, not a list.")

    # Standardize p to be a list of lists
    if not any(isinstance(sublist, list) for sublist in p):
        p = [p]  # Wrap the flat list in another list

    if EOS == "GERG2008":
        fluid1 = fluid("pr")
    else:
        fluid1 = fluid(EOS)

    for i in range(len(componentNames)):
        fluid1.addComponent(componentNames[i], componentFrac[i],
            ↪ "mol/sec")  # Create fluid object

```

```
    fluid1.setMixingRule("classic")  # Define mixing rule

    viscosityList = []  # Initialize list for viscosity data

    for pressure_list in p:
        temp_viscosities = []  # Store viscosities for each pressure list
        for pressure in pressure_list:
            fluid1.setTemperature(T, "K")  # Constant temperature
            fluid1.setPressure(pressure, "MPa")
            TPflash(fluid1)
            fluid1.initProperties()

            if fluid1.getNumberOfPhases() != 1:
                phase_names = [fluid1.getPhase(phase).getPhaseTypeName() for phase in range(fluid1.getNumberOfPhases())]
                print(f"Error: More than one phase detected. Number of phases: {fluid1.getNumberOfPhases()}")
                print(f"Phases detected: {', '.join(phase_names)}")
                raise RuntimeError("Too many phases detected, stopping the code.")

            if EOS == "GERG2008":
                fluid1.getPhase(0).getProperties_GERG2008().setViscosityModel(method)
                fluid1.initProperties()  # Calculate properties
                viscosity = fluid1.getViscosity("Pas") * 10**6  #  Pa *s
                temp_viscosities.append(viscosity)
            else:
                fluid1.getPhase(0).getPhysicalProperties().setViscosityModel(method)
                fluid1.initProperties()  # Calculate properties
                viscosity = fluid1.getViscosity("Pas") * 10**6  #  Pa *s
                temp_viscosities.append(viscosity)
        viscosityList.append(temp_viscosities)

    return viscosityList    #viscosityList is a list of lists. If only one pressure provided [p1] ----> [[ 1 ]] (what is returned)



components = [
    "methane",  #  C H
    "ethane",  #   C  H
    "propane",  #   C  H
    "n-butane",  # n-   C   H
    "i-butane",  # iso-   C   H
    "n-pentane",  # n-   C   H
    "i-pentane",  # iso-   C   H
    "n-pentane",  # neo-   C   H
    "n-hexane",  # n-   C   H
    "n-heptane",  # n-   C   H
    "n-octane",  # n-   C   H
    "n-nonane",  # n-   C   H
    "nC10",  # n-    C    H
    "benzene",  #   C  H
    "toluene",  #   C  H
    #"C8",  # o-   C   H
    "CO2",  #  C O
    "nitrogen",  #  N
    "oxygen",  #  O
    "helium",  # He
    "hydrogen"  #  H
]
```

D.3 Plotting

Python code for plotting ARD scatter plots of viscosity data.

```
from Viscosity import getVisc
from Viscosity import components
from experimental_data import expData
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use("Agg")


def isotherm(target_temp, temperature_lists, pressure_lists, viscosity_lists):
    tolerance = 1
    for i in range(len(temperature_lists)):

        if len(temperature_lists[i]) != len(pressure_lists[i]) or len(pressure_lists[i]) != len(viscosity_lists[i]):
            raise RuntimeError("Lists for temperature, pressure and viscosity not same length, stopping the code.")
        else:
            for j in range(len(temperature_lists[i]) - 1, -1, -1):  # Iterate in reverse
                if abs(temperature_lists[i][j] - target_temp) > tolerance:
                    viscosity_lists[i].pop(j)
                    pressure_lists[i].pop(j)
    return pressure_lists, viscosity_lists



def ARDViscosityIsotherm(datasets, target_temp, EOS, viscosity_model):
    composition = expData(datasets[0])[0]
    all_temperatures = []
    all_pressures = []
    all_viscosities = []
    all_compositions = []
    author_list = []

    for dataset in datasets:
        all_compositions.append(expData(dataset)[0]) #This is used for checking later in the code that the compositions are the same in the different datasets
        #creating matrix with all temperatures, pressures, and viscosities for the datasets observed
        all_temperatures.append(expData(dataset)[1])
        all_pressures.append(expData(dataset)[2])
        all_viscosities.append(expData(dataset)[3])
        author_list.append(expData(dataset)[4])

    #check if composition in the datasets are the same:
    if len(all_compositions)>1:
        if all(composition==all_compositions[0] for composition in all_compositions):
            pass
        else:
            raise RuntimeError("Different compositions being compared, stopping the code.")

    #creating lists with pressures and viscosities corresponding to the temperature to be observed
```

```
    isotherm_exp_pressures, isotherm_exp_viscosities =
        ↪ isotherm(target_temp, all_temperatures, all_pressures,
        ↪ all_viscosities) #List of lists

    isotherm_model_viscosities = getVisc(EOS, components, composition,
        ↪ target_temp, isotherm_exp_pressures, viscosity_model)
    ARD_list = []        #initiating empty ARD value list
    for i in range(len(isotherm_exp_viscosities)):
        ard_list = []
        for j in range(len(isotherm_exp_viscosities[i])):
            last_iteration = len(isotherm_exp_viscosities[i])
            ard = 100* (isotherm_model_viscosities[i][j] -
                ↪ isotherm_exp_viscosities[i][j]) /
                ↪ isotherm_exp_viscosities[i][j]
            ard_list.append(ard)
            if j == last_iteration-1:
                ARD_list.append(ard_list)
    return  isotherm_exp_pressures, ARD_list, author_list, composition
        ↪   #All returned lists are list of lists, apart from
        ↪ composition which is a single list


def ARDPlot(datasets, target_temp, save_path):
    plt.figure(figsize=(10,6))
    #LBC
    SRK_LBC = ARDViscosityIsotherm(datasets, target_temp, "srk", "LBC")
    PR_LBC = ARDViscosityIsotherm(datasets, target_temp, "pr", "LBC")
    plt.scatter(SRK_LBC[0], SRK_LBC[1], label="SRK_LBC", marker='o',
        ↪ color = 'black', facecolors = "none")
    plt.scatter(PR_LBC[0], PR_LBC[1], label="PR_LBC", marker='x', color
        ↪ = 'black')

    #Friction theory
    friction_theory_SRK = ARDViscosityIsotherm(datasets, target_temp,
        ↪ "srk", "friction theory")
    PR_friction_theory = ARDViscosityIsotherm(datasets, target_temp,
        ↪ "pr", "friction theory")
    plt.scatter(friction_theory_SRK[0], friction_theory_SRK[1],
        ↪ label="SRK_friction_theory", marker='o', color = 'red',
        ↪ facecolors = "none")
    plt.scatter(PR_friction_theory[0], PR_friction_theory[1],
        ↪ label="PR_friction_theory", marker='x', color = 'red')

    #PFCT
    SRK_PFCT = ARDViscosityIsotherm(datasets, target_temp, "srk",
        ↪ "PFCT")
    PR_PFCT = ARDViscosityIsotherm(datasets, target_temp, "pr", "PFCT")
    plt.scatter(SRK_PFCT[0], SRK_PFCT[1], label="SRK_PFCT", marker='o',
        ↪ color = 'blue', facecolors = "none")
    plt.scatter(PR_PFCT[0], PR_PFCT[1], label="PR_PFCT", marker='x',
        ↪ color = 'blue')


    plt.xlabel("Pressure (MPa)")
    plt.ylabel(r'$100 \left(\eta_{\mathrm{model}} -
        ↪ \eta_{\mathrm{experimental}}\right)/\eta_{\mathrm{experimental}}$')
    plt.title(f"T = {target_temp}K")
    #ENDRE DETTE
    plt.xlim(0,42)
    plt.ylim(-16, 16)
    ###
    plt.legend(loc='upper center', bbox_to_anchor=(0.5,
        ↪ -0.1),fancybox=True, shadow=True, ncol=5, frameon = False)
```

```
    plt.grid()
    plt.savefig(save_path, dpi=300, bbox_inches='tight')




def groupedPlots(datasets, target_temps, save_path):
    # Create a 2x2 grid of subplots
    fig, axs = plt.subplots(2, 2, figsize=(14, 8), sharex=True,
        ↪ sharey=True)
    axs = axs.flat  # Flatten for easier iteration

    for idx, ax in enumerate(axs):
        if idx >= len(target_temps):
            ax.axis('off')  # Hide extra axes if any
            continue

        temp = target_temps[idx]
        dataset = datasets[idx % len(datasets)]  # Cycle through
            ↪ datasets if fewer than temps

        # Plot LBC
        SRK_LBC = ARDViscosityIsotherm(dataset, temp, "srk", "LBC")
        PR_LBC = ARDViscosityIsotherm(dataset, temp, "pr", "LBC")
        ax.scatter(SRK_LBC[0], SRK_LBC[1], label="SRK_LBC", marker='o',
            ↪ color='black', facecolors="none", s=30)
        ax.scatter(PR_LBC[0], PR_LBC[1], label="PR_LBC", marker='x',
            ↪ color='black', s=30)

        # Plot Friction Theory
        friction_theory_SRK = ARDViscosityIsotherm(dataset, temp,
            ↪ "srk", "friction theory")
        PR_friction_theory = ARDViscosityIsotherm(dataset, temp, "pr",
            ↪ "friction theory")
        ax.scatter(friction_theory_SRK[0], friction_theory_SRK[1],
            ↪ label="SRK_friction_theory", marker='o', color='red',
            ↪ facecolors="none", s=30)
        ax.scatter(PR_friction_theory[0], PR_friction_theory[1],
            ↪ label="PR_friction_theory", marker='x', color='red', s=30)

        # Plot PFCT
        SRK_PFCT = ARDViscosityIsotherm(dataset, temp, "srk", "PFCT")
        PR_PFCT = ARDViscosityIsotherm(dataset, temp, "pr", "PFCT")
        ax.scatter(SRK_PFCT[0], SRK_PFCT[1], label="SRK_PFCT",
            ↪ marker='o', color='blue', facecolors="none", s=30)
        ax.scatter(PR_PFCT[0], PR_PFCT[1], label="PR_PFCT", marker='x',
            ↪ color='blue', s=30)

        # Title and limits
        ax.set_title(f"T = {temp}K", fontsize = 12)
        ax.set_xlim(0, 42)
        ax.set_ylim(-16, 16)
        ax.grid()

    for ax in axs[-2:]:  # Select the last two plots (bottom row in a
        ↪ 2x2 grid)
        ax.set_xlabel('Pressure (MPa)', fontsize=12)

    if len(target_temps) == 3:
        axs[1].set_xlabel('Pressure (MPa)', fontsize=12)
        axs[1].tick_params(axis='x', which='both', labelbottom=True)  #
            ↪ Enable x-tick labels


```

```
    # Re-enable y-tick labels for the right column
    for ax in axs[1::2]:  # Every second plot (right column in 2x2 grid)
        ax.tick_params(labelleft=True)

    # Shared labels
    fig.text(0, 0.5, r'$100 \left(\eta_{\mathrm{model}} - \eta_{\mathrm{experimental}}\right)/\eta_{\mathrm{experimental}}$',
             ha='center', va='center', rotation='vertical', fontsize = 14)


    # Add a single legend outside the plots
    handles, labels = axs[0].get_legend_handles_labels()
    fig.legend(handles, labels, loc='upper center', bbox_to_anchor=(0.5, 0), ncol=5, frameon=False, fontsize = 14)

    # Adjust layout and save
    fig.tight_layout(rect=[0, 0.05, 1, 0.95])  # Leave space for legend
    plt.savefig(save_path, dpi=300, bbox_inches='tight')




def groupedAuthorPlots(datasets, target_temps, save_path):
    # Create a 2x2 grid of subplots
    fig, axs = plt.subplots(2, 2, figsize=(14, 8), sharex=True, sharey=True)
    axs = axs.flat  # Flatten for easier iteration

    # Initialize lists to store legend handles and labels
    legend_handles = []
    legend_labels = []

    def add_to_legend(handle, label):
        if label not in legend_labels:
            legend_handles.append(handle)
            legend_labels.append(label)

    for idx, ax in enumerate(axs):
        if idx >= len(target_temps):
            ax.axis('off')  # Hide extra axes if any
            continue

        temp = target_temps[idx]
        dataset = datasets[idx % len(datasets)]  # Cycle through datasets if fewer than temps

        # LBC
        SRK_LBC_p, SRK_LBC_visc, authors, composition = ARDViscosityIsotherm(dataset, temp, "srk", "LBC")
        PR_LBC_p, PR_LBC_visc, _, x = ARDViscosityIsotherm(dataset, temp, "pr", "LBC")
        #print(dataset, temp)
        #print("ARD_LBC_srk:", SRK_LBC_visc, "Pressures:", SRK_LBC_p)
        #print("ARD_LBC_pr:", PR_LBC_visc, "Pressures:", PR_LBC_p)

        # Friction Theory
        SRK_friction_theory_p, SRK_friction_theory_visc, _, x = ARDViscosityIsotherm(dataset, temp, "srk", "friction theory")
        PR_friction_theory_p, PR_friction_theory_visc, _, x = ARDViscosityIsotherm(dataset, temp, "pr", "friction theory")
```

```
        #print("ARD_FT_srk:", SRK_friction_theory_visc, "Pressures:",
            ↪ SRK_friction_theory_p)
        #print("ARD_FT_pr:", PR_friction_theory_visc, "Pressures:",
            ↪ PR_friction_theory_p)

        # PFCT
        SRK_PFCT_p, SRK_PFCT_visc, _, x = ARDViscosityIsotherm(dataset,
            ↪ temp, "srk", "PFCT")
        #SRK is hard coded into NeqSim, so changing EOS to PR will not
            ↪ change the result
        #print("ARD_PFCT_srk:", SRK_PFCT_visc, "Pressures:", SRK_PFCT_p)

        for i in range(len(SRK_LBC_p)):
            # Plot LBC
            handle = ax.scatter(SRK_LBC_p[i], SRK_LBC_visc[i],
                ↪ label=f"SRK_LBC, {authors[i][0]}",
                                marker=authors[i][1], color='black',
                                    ↪ facecolors="none", s=30)
            add_to_legend(handle, f"SRK_LBC, {authors[i][0]}")

            handle = ax.scatter(PR_LBC_p[i], PR_LBC_visc[i],
                ↪ label=f"PR_LBC, {authors[i][0]}",
                                marker=authors[i][2], color='black',
                                    ↪ facecolors="none", s=30)
            add_to_legend(handle, f"PR_LBC, {authors[i][0]}")

            # Plot Friction Theory
            handle = ax.scatter(SRK_friction_theory_p[i],
                ↪ SRK_friction_theory_visc[i], label=f"SRK_FT,
                ↪ {authors[i][0]}",
                                marker=authors[i][1], color='red',
                                    ↪ facecolors="none", s=30)
            add_to_legend(handle, f"SRK_FT, {authors[i][0]}")

            handle = ax.scatter(PR_friction_theory_p[i],
                ↪ PR_friction_theory_visc[i], label=f"PR_FT,
                ↪ {authors[i][0]}",
                                marker=authors[i][2], color='red',
                                    ↪ facecolors="none", s=30)
            add_to_legend(handle, f"PR_FT, {authors[i][0]}")

            # PFCT
            handle = ax.scatter(SRK_PFCT_p[i], SRK_PFCT_visc[i],
                ↪ label=f"SRK_PFCT, {authors[i][0]}",
                                marker=authors[i][1], color='blue',
                                    ↪ facecolors="none", s=30)
            add_to_legend(handle, f"SRK_PFCT, {authors[i][0]}")

        ax.set_title(f"T = {temp}K", fontsize=12)
        ax.set_xlim(0, 42)
        ax.set_ylim(-16, 16)
        ax.grid()

    for ax in axs[-2:]:  # Bottom row in a 2x2 grid
        ax.set_xlabel('Pressure (MPa)', fontsize=12)

    if len(target_temps) == 3:
        axs[1].set_xlabel('Pressure (MPa)', fontsize=12)
        axs[1].tick_params(axis='x', which='both', labelbottom=True)  #
            ↪ Enable x-tick labels

    for ax in axs[1::2]:  # Right column in a 2x2 grid
        ax.tick_params(labelleft=True)
```

```
240
241     fig.text(0, 0.5, r'$100 \left(\eta_{\mathrm{model}} -
            ↪ \eta_{\mathrm{experimental}}\right)/\eta_{\mathrm{experimental}}$',
242              ha='center', va='center', rotation='vertical', fontsize=15)
243
244     # Create the legend from collected handles and labels
245     fig.legend(legend_handles, legend_labels, loc='upper center',
            ↪ bbox_to_anchor=(0.5, 0.06),
246                ncol=5, frameon=False, fontsize=14)
247
248     #create title for entire plotgroup
249     if composition[0] == 1:
250         title = "Pure Methane"
251     elif composition[-1] == 1:
252         title = "Pure Hydrogen"
253     else:
254         title = f"{composition[0]} Methane and {composition[-1]}
                ↪ Hydrogen"
255
256     fig.suptitle(title, fontsize=16, y=0.93)
257
258     fig.tight_layout(rect=[0, 0.05, 1, 0.95])
259     plt.savefig(save_path, dpi=300, bbox_inches='tight')
260
261
262
263
264 ### PLOTS WITH AUTHOR LABEL ###
265 #Pure methane
266 groupedAuthorPlots([["Ark1_2"], ["Ark1_2", "Ark3_2"], ["Ark1_2"],
        ↪ ["Ark1_2"]], [248.0, 273.15, 298.4, 323.6],
        ↪ r"C:\Users\akser\OneDrive - NTNU\Prosjektoppgave\Viscosity
        ↪ PLOTS\Plots Author names 1.2\Pure Methane.png")
267
268 #Pure Hydrogen
269 groupedAuthorPlots([["Ark1_1"], ["Ark1_1", "Ark3_1"], ["Ark1_1",
        ↪ "Ark2_1"], ["Ark1_1", "Ark2_1"]], [248.3, 273.15, 298.15,
        ↪ 323.15], r"C:\Users\akser\OneDrive -
        ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\Plots Author names 1.2\Pure
        ↪ Hydrogen.png")
270
271 #Mixtures
272 #0.9CH4 + 0.1H2
273 groupedAuthorPlots([["Ark1_3"], ["Ark1_3"], ["Ark1_3"]], [273.3, 298.4,
        ↪ 323.7], r"C:\Users\akser\OneDrive -
        ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\Plots Author names
        ↪ 1.2\0.9CH4_0.1H2.png")
274
275 #0.8CH4 + 0.2H2
276 groupedAuthorPlots([["Ark1_4", "Ark3_3"], ["Ark1_4"], ["Ark1_4"]],
        ↪ [273.15, 298.3, 323.3], r"C:\Users\akser\OneDrive -
        ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\Plots Author names
        ↪ 1.2\0.8CH4_0.2H2.png")
277
278 #0.5CH4 + 0.5H2
279 groupedAuthorPlots([["Ark1_5"], ["Ark1_5"], ["Ark1_5"]], [273.4, 298.4,
        ↪ 323.5], r"C:\Users\akser\OneDrive -
        ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\Plots Author names
        ↪ 1.2\0.5CH4_0.5H2.png")
280
281 #0.1CH4 + 0.9H2
282 groupedAuthorPlots([["Ark2_2"], ["Ark2_2"], ["Ark2_2"]], [298.15,
        ↪ 323.15, 348.15], r"C:\Users\akser\OneDrive -
```

```
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\Plots Author names
    ↪ 1.2\0.1CH4_0.9H2.png")
283
284
285
286
287 ###GROUPED PLOTS:
288 #Pure Methane
289 #groupedPlots([["Ark1_2"], ["Ark1_2", "Ark3_2"], ["Ark1_2"],
    ↪ ["Ark1_2"]], [248.0, 273.15, 298.4, 323.6],
    ↪ r"C:\Users\akser\OneDrive - NTNU\Prosjektoppgave\Viscosity
    ↪ PLOTS\Plots GROUP\Pure Methane")
290
291 #Pure Hydrogen
292 #groupedPlots([["Ark1_1"], ["Ark1_1", "Ark3_1"], ["Ark1_1", "Ark2_1"],
    ↪ ["Ark1_1", "Ark2_1"]], [248.3, 273.15, 298.15, 323.15],
    ↪ r"C:\Users\akser\OneDrive - NTNU\Prosjektoppgave\Viscosity
    ↪ PLOTS\Plots GROUP\Pure Hydrogen")
293
294
295 #MIXTURES
296 #0.9CH4 + 0.1H2
297 #groupedPlots([["Ark1_3"], ["Ark1_3"], ["Ark1_3"], ["Ark1_3"]], [223.0,
    ↪ 273.3, 298.4, 323.7], r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\Plots
    ↪ GROUP\0.9CH4_0.1H2.png")
298
299 #0.8CH4 + 0.2H2
300 #groupedPlots([["Ark3_3"], ["Ark1_4", "Ark3_3"], ["Ark1_4"],
    ↪ ["Ark1_4"]], [223.15, 273.15, 298.3, 323.3],
    ↪ r"C:\Users\akser\OneDrive - NTNU\Prosjektoppgave\Viscosity
    ↪ PLOTS\Plots GROUP\0.8CH4_0.2H2.png")
301
302 #0.5CH4 + 0.5H2
303 #groupedPlots([["Ark1_5"], ["Ark1_5"], ["Ark1_5"], ["Ark1_5"]], [223.0,
    ↪ 273.4, 298.4, 323.5], r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\Plots
    ↪ GROUP\0.5CH4_0.5H2.png")
304
305 #0.1CH4 + 0.9H2
306 #groupedPlots([["Ark2_2"], ["Ark2_2"], ["Ark2_2"]], [298.15, 323.15,
    ↪ 348.15], r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\Plots
    ↪ GROUP\0.1CH4_0.9H2.png")
307
308
309
310
311 ### SINGLE PLOTS: ###
312 #Pure hydrogen
313 #ARDPlot(["Ark1_1", "Ark2_1"], 323.15, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\Pure Hydrogen
    ↪ 323.15K.png")
314 #ARDPlot(["Ark1_1", "Ark2_1"], 298.15, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\Pure Hydrogen
    ↪ 298.15K.png")
315 #ARDPlot(["Ark1_1", "Ark3_1"], 273.15, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\Pure Hydrogen
    ↪ 273.15K.png")
316
317 #Pure methane
318 #ARDPlot(["Ark1_2"], 323.6, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\Pure Methane
```

```
    ↪ 323.63K.png")
#ARDPlot(["Ark1_2"], 298.4, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\Pure Methane
    ↪ 298.37K.png")
#ARDPlot(["Ark1_2", "Ark3_2"], 273.15, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\Pure Methane
    ↪ 273.15K.png")
#ARDPlot(["Ark1_2"], 248.0, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\Pure Methane
    ↪ 248.06K.png")


#Mixtures
#0.9CH4 + 0.1H2
#ARDPlot(["Ark1_3"], 323.7, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.9CH4 & 0.1H2
    ↪ 323.73K.png")
#ARDPlot(["Ark1_3"], 298.4, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.9CH4 & 0.1H2
    ↪ 298.37.png")
#ARDPlot(["Ark1_3"], 273.32, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.9CH4 & 0.1H2
    ↪ 273.32.png")
#ARDPlot(["Ark1_3"], 223.0, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.9CH4 & 0.1H2
    ↪ 223.00.png")

#0.8CH4 + 0.2H2
#ARDPlot(["Ark1_4"], 323.3, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.8CH4 & 0.2H2
    ↪ 323.33K.png")
#ARDPlot(["Ark1_4"], 298.3, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.8CH4 & 0.2H2
    ↪ 298.27K.png")
#ARDPlot(["Ark1_4", "Ark3_3"], 273.15, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.8CH4 & 0.2H2
    ↪ 273.15K.png")

#0.5CH4 + 0.5H2
#ARDPlot(["Ark1_5"], 323.5, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.5CH4 & 0.5H2
    ↪ 323.53K.png")
#ARDPlot(["Ark1_5"], 298.4, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.5CH4 & 0.5H2
    ↪ 298.37K.png")
#ARDPlot(["Ark1_5"], 273.4, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.5CH4 & 0.5H2
    ↪ 273.42K.png")

#0.1CH4 + 0.9H2
#ARDPlot(["Ark2_2"], 298.15, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.1CH4 & 0.9H2
    ↪ 298.15K.png")
#ARDPlot(["Ark2_2"], 323.15, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.1CH4 & 0.9H2
    ↪ 323.16K.png")
#ARDPlot(["Ark2_2"], 348.15, r"C:\Users\akser\OneDrive -
    ↪ NTNU\Prosjektoppgave\Viscosity PLOTS\22.11.2024\0.1CH4 & 0.9H2
    ↪ 348.16K.png")


### END CREATE PLOTS ###
```

NTNU – Trondheim
Norwegian University of
Science and Technology