Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering

Optimization of PC−SAFT Equation of State Parameters for Modeling Thermophysical Properties of Fluids

Optimalizace parametrů stavové rovnice PC−SAFT pro modelování termofyzikáních vlastností tekutin

Bachelor's Degree Project

Author:	**Gabriela Spilková**
Supervisor:	**Ing. Václav Vinš, Ph.D**
Consultant:	**Ing. David Celný**
Language advisor:	**Mgr. Hana Čápová**

Academic year: 2021/2022

Katedra: matematiky Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Gabriela Spilková
Studijní program:	Aplikace přírodních věd
Studijní obor:	Aplikovaná informatika
Název práce (česky):	Optimalizace parametrů stavové rovnice PC-SAFT v modelování termofyzikálních vlastností tekutin
Název práce (anglicky):	Parameter optimization of PC-SAFT equation of state for the modeling of thermophysical properties of fluids

Pokyny pro vypracování:

1) Stručná rešerše problematiky stavových rovnic a jejich využití pro modelování termofyzikálních vlastností tekutin potřebných pro návrh inženýrských aplikací, zejména moderních rovnic typu SAFT (statistical associating fluid theory).
2) Vypracování rešerše možných optimalizačních algoritmů pro hledání látkových parametrů stavových rovnic, např. nelineárních gradientních metod, simplex algoritmu.
3) Sestavení vybraného optimalizačního algoritmu např. Levenberg-Marquardt v programovacím jazyce Python určeného pro nalezení látkových parametrů stavové rovnice PC-SAFT. Vlastní implementace napomůže k pochopení problematiky jako uvažování různých vah a druhů vstupních dat (např. tlak sytých par, hustota, rychlost zvuku).
Pozn.: kódy vlastní stavové rovnice PC-SAFT včetně databáze vstupních dat budou dodány školícím pracovištěm ÚT AV ČR.
4) Porovnání optimalizačních algoritmů dostupných v běžných programovacích jazycích pro technické výpočty (např. Matlab a Python) na dané aplikaci.
5) Optimalizace parametrů stavové rovnice pro vybrané látky od jednodušších se třemi látkovými parametry (např. etan) až po složitější systémy (např. moderní chladiva a teplonosné kapaliny) vykazující polaritu či asociaci.
6) Kritické zhodnocení výsledků optimalizace a ověření na dříve popsaných látkách.
7) Napočítání termofyzikálních dat pro vybrané doposud nedostatečně popsané a technicky zajímavé látky.

Doporučená literatura:

1) J. Gross, G. Sadowski, Perturbed-chain SAFT – An equation of state based on a perturbation theory for chained molecules. Industrial & Engineering Chemistry Research 40, 2001, 1244-1260.

2) V. Vinš, A. Aminian, D. Celný, M. Součková, J. Klomfar, M. Čenský, O. Prokopová, Surface tension and density of dielectric heat transfer fluids of HFE type – experimental data at 0.1 MPa and modeling with PC-SAFT equation of state and density gradient theory. International Journal of Refrigeration, 2021, doi: 10.1016/j.ijrefrig.2021.06.029.

3) T. Strutz, Data fitting and uncertainty – A practical introduction to weighted least squares and beyond. Vieweg & Teubner, Wiesbaden 2011.

4) K. Madsen, H.B. Nielsen, O. Tingleff, Methods for non-linear least squares problems. 2nd ed., Technical University of Denmark, 2004.

5) C.T. Kelley, Iterative Methods for Optimization. SIAM Frontiers in Applied Mathematics, 1999.

6) M.K. Transtrum, B.B. Machta, J.P. Setha, Geometry of nonlinear least squares with applications to sloppy models and optimization. Physical Review E 83, 2011, 036701.

Jméno a pracoviště vedoucího bakalářské práce:

Ing. Václav Vinš, Ph.D.
Ústav termomechaniky AV ČR, v. v. i., Dolejškova 1402, 182 00 Praha 8,
https://www.it.cas.cz/d2/l021/

Jméno a pracoviště konzultanta:

Ing. David Celný
Ústav termomechaniky AV ČR, v. v. i., Dolejškova 1402, 182 00 Praha 8

Datum zadání bakalářské práce: 31.10.2021

Datum odevzdání bakalářské práce: 7.7.2022

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 21. října 2021

[illegible]

....
garant oboru

[illegible]

....
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
-1-

[illegible]

....
děkan

Acknowledgment:

In this part, I would like to thank my supervisor Ing. Václav Vinš, Ph.D and my consultant Ing. David Celný. Thanks to their willingness and helpfulness, I was able to better penetrate not only the mathematical - physical and programming dimensions of this issue, but I also gained many other additional skills, such as proper work with professional resources, orientation in professional programs and work with shared programming environment Git and it's Bitbucket storage. I firmly believe that I will continue to make extensive use of this knowledge, which was willingly passed on to me during this year.

I would also like to express my gratitude to my language advisor Mgr. Hana Čápová for her professional assistance, helpful approach and detailed text control.

Just as importantly, I wish to express my appreciation to my father, who accompanied me throughout my entire university studies and to the rest of the family, who supported me in my studies in every possible way.

Author's declaration:

I declare that this Bachelor's Degree Project is entirely my own work and I have listed all the used sources in the bibliography.

Prague, July 7, 2022 Gabriela Spilková

Název práce:

Optimalizace parametrů stavové rovnice PC−SAFT pro modelování termofyzikáních vlastností tekutin

Autor: Gabriela Spilková

Obor: Aplikovaná informatika

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Václav Vinš, Ph.D, Ústav termomechaniky, Akademie věd české republiky

Konzultant: Ing. David Celný, Ústav termomechaniky, Akademie věd české republiky

Abstrakt: Problematika stavových rovnic je v moderní fyzice důležitým tématem. Existuje matematicko - fyzikální aparát, pomocí kterého lze predikovat stavové vlastnosti látky v různých podmínkách. Tento aparát se nazývá statistická asociační teorie tekutin. Na základě této teorie funguje stavová rovnice PC-SAFT, která po správné optimalizaci svých vstupních parametrů může správně predikovat chování zkoumané látky v různých fyzikálních podmínkách. Cílem této práce bylo provést tuto optimalizaci v programovacím jazyce Python. Existují různé optimalizační algoritmy, které lze použít. Zde byl použit a vlastnoručně naimplementován Levenberg-Marqardtův algoritmus, jež byl dále porovnán s metodou důvěryhodného regionu a metodou dogbox, oběmi z knihovny SciPy. Toto porovnání proběhlo na methanu, pentanu a pentafluorethanu, které se hojně využívají v průmyslu. Ukázalo se, že algoritmy fungují s podobnou přesností pro látky jednodušší, tj. např. s nulovým dipólem, jako je methan a pentan, pro složitější pentafluorethane ale vykazuje nejlepší výsledky právě vlastnoručně naimplementovaný Levenberg-Marquardtův algoritmus.

Klíčová slova: Gauss-Newtonova metoda, Lennard-Jonesův potenciál, Levenberg-Marquardtův algoritmus, metan, metoda dogbox, numerická derivace, optimalizace, pentafluorethan, pentan, Stavová rovnice PC-SAFT, programovací jazyk Python, hustota nasycené kapaliny, tlak nasycených par, knihovna SciPy, statistická asociační teorie tekutin, diagram teplota-objem

Title:

Optimization of PC−SAFT Equation of State Parameters for Modeling Thermophysical Properties of Fluids

Author: Gabriela Spilková

Abstract: The issue of equations of state is an important topic in modern physics. There is a mathematical - physical apparatus, which can be used to predict the state properties of a substance in different conditions. This apparatus is called statistical association theory of fluids. The PC-SAFT equation of state is based on this theory, which - after the proper optimization of its input parameters - can correctly predict the investigated substances in different physical conditions. The aim of this work was to perform this optimization in the Python programming language. There are various optimization algorithms that can be used. Here, the Levenberg-Marqardt algorithm was used and hand-implemented, which was further compared with the trusted region method and the dogbox method, both from the SciPy library. This comparison was made for methane, pentane and pentafluoroethane, which are widely used in the industry. It was observed that the algorithms work with similar accuracy for simpler substances, i.e. with a zero dipole such as methane and pentane, however, for more complex substances, such as pentafluoroethane, the best results were obtained from the self-implemented Levenberg-Marquardt algorithm.

Key words: dogbox method, Gauss-Newton method, Lennard-Jones potential, Levenberg-Marquardt algorithm, methane, numerical differentiation, optimization, PC-SAFT equation of state, pentafluorethane, pentane, Python programming language, SAFT parameters, saturated liquid density, saturated vapor pressure, SciPy library, statistical associating fluid theory, temperature-volume diagram

Contents

1 Essential concepts of thermodynamics

1.1 Motivation

Before proceeding in this work to the part describing the use of the Levenberg-Marquardt algorithm for optimization of the PC-SAFT (the perturbed-chain statistical associating fluid theory) equation of state parameters, it is necessary to introduce the physical background of this topic.

First, very basic concepts of thermomechanics, necessary for understanding the others, will be introduced. In short, these include the term pure substance and its phases and the terms saturation temperature and saturation pressure.

Then, the statistical associating fluid theory (SAFT) will be described. The important SAFT concepts and parameters will be introduced, namely London dispersion forces and Pauli repulsion with the approaches used for its description.

Finally a list of properties that can be calculated and determined by using SAFT will be described. For this purpose, some theory is needed of which the most important concept is the Helmholtz free energy.

1.2 Pure substances

The following theoretical part applies to pure substances. The subsection 1.2 is based on resource [1]. By the term pure substance is meant a substance that is chemically homogeneous and has a fixed chemical composition. As a pure substance might be considered also a combination of different phases of one substance and in some cases also mixtures, for example dry air. By the term *phase* is usually meant commonly known the solid, liquid and gaseous state of substance. However, some substances might have more species of one state. For example helium is known in different phases of its liquid state, which have a different molecular structure. For the solid phase of a substance, orientation of atoms is fixed by very high intermolecular forces. For the liquid phase, these molecular forces are weaker and they break. For the gaseous phase, the structure is broken up to the individual molecules. This is illustrated in Figure 1

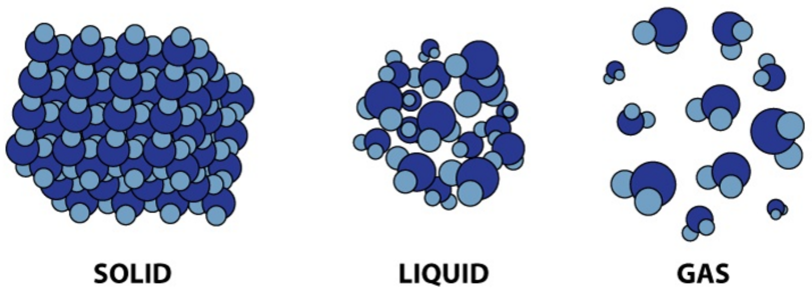

Figure 1: Molecular structure for solid, liquid and gaseous phase of a substance (https://quizzis.com)

Figure 2 describes that as energy is added to a solid, its temperature increases until the solid reaches its melting point temperature T_{MP}. At the first part of the diagram, from initial temperature to T_{MP}, the substance is fully solid. With the temperature increased to T_{MP} any additional energy added will start the melting phase transition, and the substance will be in a mixed solid/liquid phase. When enough energy is added to the substance, the substance will be fully melted and when the temperature is increased to T_{BP}, the liquid's temperature reaches its boiling point. As more energy is added, the substance is found in a mixed liquid gas phase, until enough energy is added to fully vaporize the liquid, resulting in a pure gas.

The important thing to notice is that during a phase change, temperature remains constant even while energy is being added and the substance is in a mixed phase. This could be tested by boiling a pot of water while measuring temperature with a thermometer.The water temperature remains at exactly 100 °C throughout the boiling process. Even as the flame continues to pour heat into the pot-water system, the water's temperature will not change until all the water is boiled away.

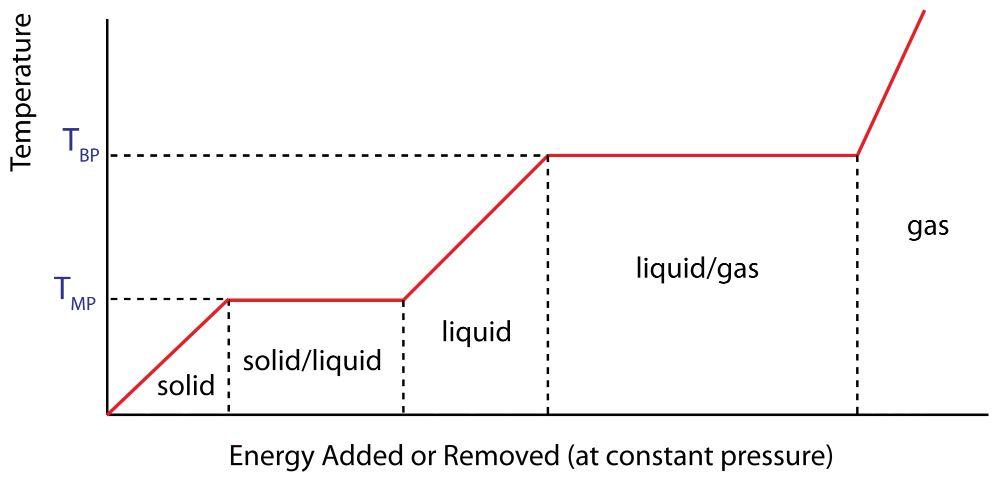

Figure 2: Temperature vs. energy added (https://phys.libretexts.org/)

1.3 Saturation temperature and saturation pressure

This subsection draws from the source [2].

Let us assume water as an example substance and atmospheric pressure 1 atm. At such atmospheric pressure, the water boils at 100 °C. With increasing pressure, the boiling point of water also increases. On the other hand, with decreasing pressure, it decreases. For example with pressure equal to 0.3 atm, the boiling point of water is 30 °C. By condensation point is meant the point, at which vapor condenses into a liquid without a change in temperature of a substance.

Saturation temperature is a temperature for a corresponding saturation pressure at which a liquid boils into its vapor phase. As follows from this definition, every saturation temperature has appropriate saturation pressure. Figure 3 shows how the saturation curve divides the area into two parts, which correspond to the two phases of the liquid - in this case water. Above the saturation curve, the substance is in the liquid phase and below the curve it is in the superheated vapor phase.

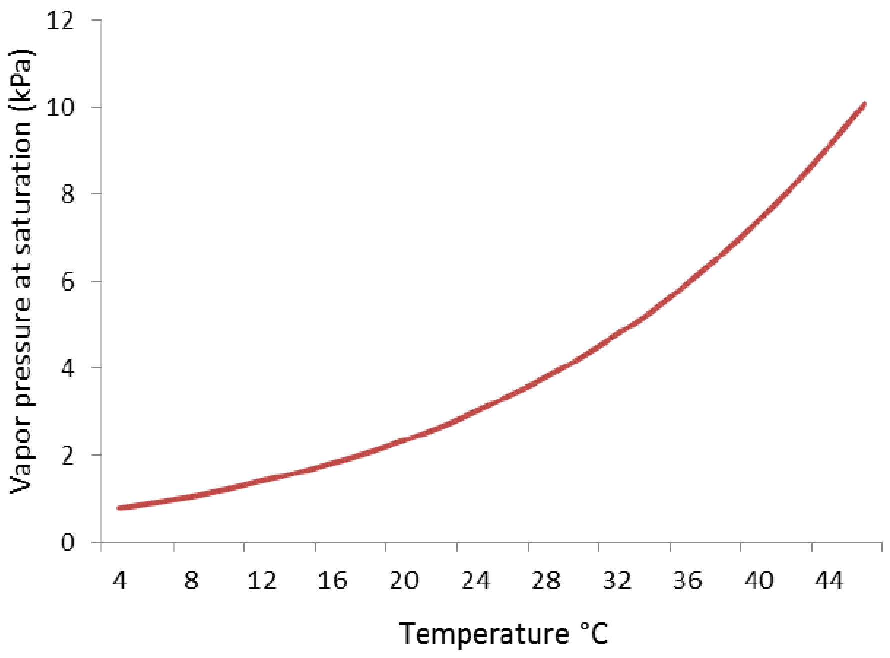

Figure 3: Vapor pressure at saturation dependence on temperature of water (https://www.researchgate.net/)

1.3.1 Saturated vapor and liquid density

Saturated vapor is the one that is in thermodynamic equilibrium with a liquid of the same temperature and pressure. This is a dynamic equilibrium in which the vaporized substance is precisely replaced by the condensed substance (the number of molecules leaving the surface is equal to the number of molecules returning to the liquid after liquefaction). If the liquid is in a closed container, saturated vapor will form after a certain time. As the temperature of the liquid increases, not only the saturated vapor pressure, but also the saturated vapor density increases. In contrast, the density of a saturated liquid, i.e., the density at which a liquid and vapor are in equilibrium, decreases with increasing temperature. It is especially these three inseparable values: the pressure and density of saturated vapor, and the density of saturated liquid, that will further be obtained using a suitable equation of state for next use.

1.3.2 Pressure-temperature diagram

Diagram, which is the most common way to show the relationship between pressure, temperature and phases of a substance is a pressure-temperature diagram. It can be constructed for any pure substance. Figure 4 shows the pressure-temperature diagram for water, which is divided into three parts by three curves. The curve separating the gas phase from the solid phase is called the sublimation curve. The curve separating the solid phase from the liquid phase is called the defusion curve, and finally, the curve separating the liquid phase from the vapor phase is called the boiling curve. It is possible to notice two important points in the diagram. The first one is the C critical point, in which there is almost no distinction between the vapor and the liquid phase. The second one is called the triple point. It occurs where the solid, liquid and gas transition curves meet. This point is the only condition in which all three phases can coexist and it is unique for every material. As can be seen in this diagram, water reaches its triple point at just above freezing (0.01 °C) and at the pressure of 0.6 kPa.

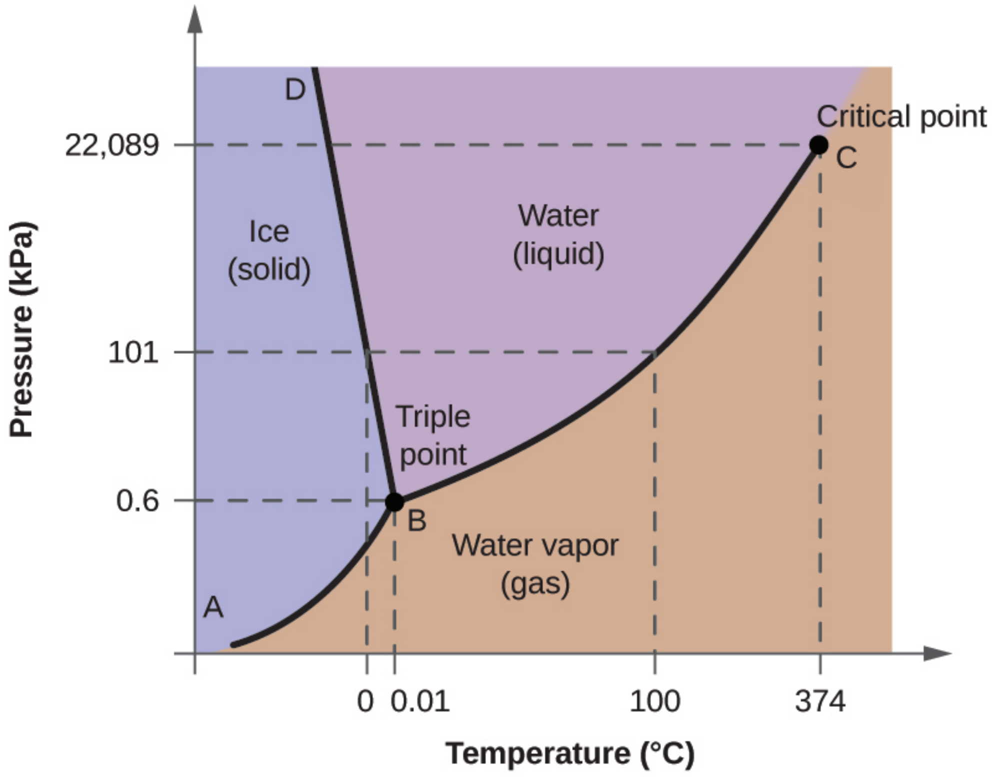

Figure 4: Pressure-temperature diagram for water (https://www.inspiritvr.com/discover)

1.3.3 Temperature-volume diagram and pressure-volume diagram

Another two important diagrams are the temperature-volume diagram and the pressure-volume diagram.

The temperature-volume diagram (Fig. 5) shows that as the pressure of saturation is increased, the amount of heat needed to change the phase from liquid to vapor is smaller (compare the intersections of P_1 and P_2 with the saturated line). It is clear that the higher the pressure is, the closer the intersections with the saturated line are one to each other, until they meet at a critical point. At this point, there is almost no difference between the compresed liquid and superheated vapor phase of a pure substance.

The pressure-volume diagram (Fig. 6) is very similar to the previous temperature-volume diagram. It can be seen that, in accordance with the diagram in Figure 5, with increasing temperature the amount of heat needed to change the phase from a liquid to a vapor is smaller. Again, if the temperature is increased sufficiently, it is possible to get to the critical point, where there is no difference between the compresed liquid and superheated vapor phase of a pure substance.

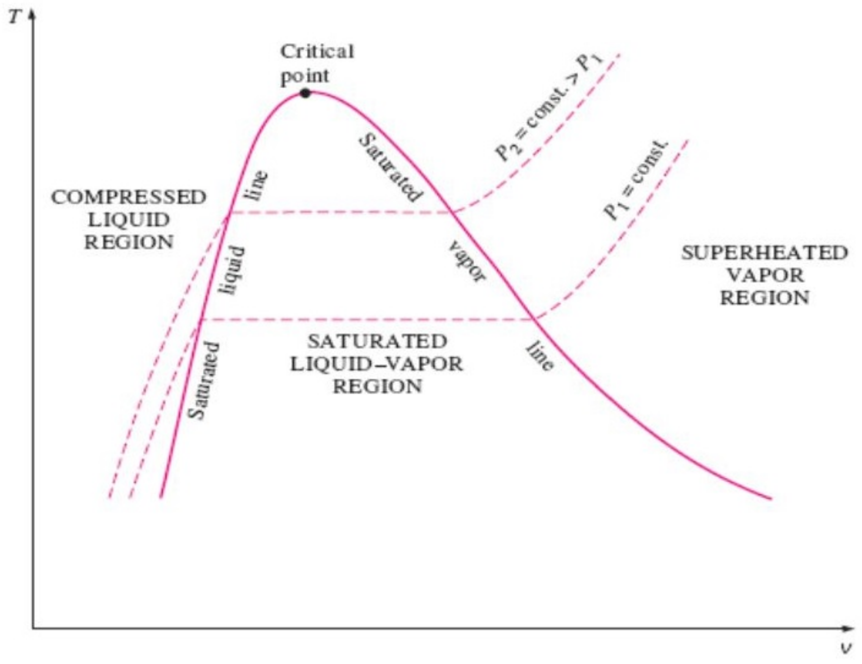

Figure 5: Temperature-volume diagram for a pure substance (https://www.unipamplona. edu.com/)

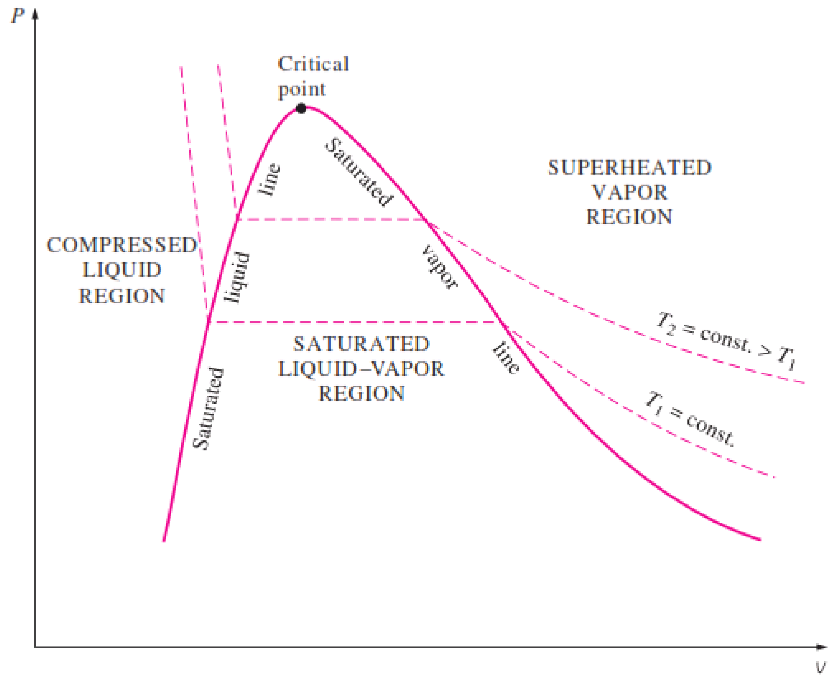

Figure 6: Pressure-volume diagram for a pure substance (https://www.unipamplona.edu. com/)

2 PC-SAFT equation of state

The equation of state combines basic thermodynamic quantities describing the behavior of a gas. Let us assume the equation of state for an ideal gas:

$$pV = nRT, \tag{1}$$

where p is the pressure, V is the volume, n the number of moles of the gas, R is the universal gas constant and T is the temperature. This equation of state is well suited for describing the temporary physical state of a gas at normal temperature.

At very low temperatures, such as in the coolant, a complex mathematical apparatus must be used. The PC-SAFT equation of state will serve this well. It is is a relatively modern model, developed by Joachim Gross and Gabriele Sadowski, firstly mentioned in their article from 2001 [3].

PC-SAFT EoS belongs to the family of that kind of EoS, which represents a laying between classic (for example, Cubic) EoS and approaches based on molecular simulations. If the thermodynamic properties and phase equilibrium of various substances (see section 1.3) are studied, the PC-SAFT EoS can be used to obtain much more accurate results than cubic EoS. It is also suitable for description of more complicated systems, such as the system of polymers, polar substances or associating fluids, e.g. mixtures with alkanols, or hydrogen sulfides [4].

Although the essence of this work is focused more on programming than physics, it is appropriate to describe the SAFT theory for understanding the physical background of this thesis, as mentioned in 1.1. The motivation is that the PC-SAFT equation of state is a good mathematical-physical apparatus used to describe physical states in substance without the need to perform experiments. After successful optimization of its parameters, its curve fits well the data of already measured substances. Due to this possibility, PC-SAFT EoS can be used to predict the behavior of substances not yet experimentally investigated.

The next part of this chapter is based on publicly available lectures by Dr Daniel Belton (University of Huddersfield), at the following link: https://www.youtube.com/c/ChemEngTutor. The other sources are reffered directly in the text.

2.1 A brief history of SAFT

The history of SAFT is relatively young. It began to develop in the 1970s. Pertrubed Hard-Chain Theory (PHCT) was the first to be introduced by S. Berthaim. The idea of this theory was that non-spherical molecules are treated as chains of freely jointed spherical segments. It can be used to calculate the properties of fluids and fluid mixtures containing both small and large molecules, including polymers, and at all fluid densities [5].

In the 1980s, M.S.Wertheim introduced the concept of short range highly directional interactions across a number of associating sites, see Figure 7.

Later, W.G.Chapman and co-workers extended Wertheim's theory an applied it to mixtures (see Figure 8). This theory is already known as SAFT.

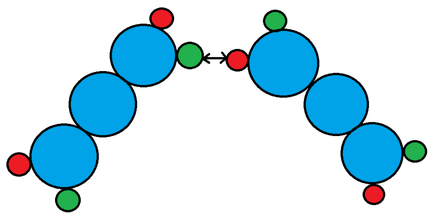

Figure 7: Wertheim's concept

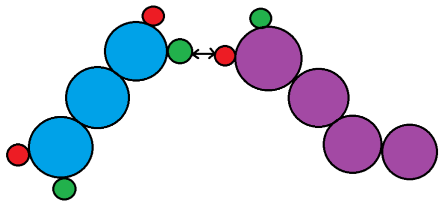

Figure 8: SAFT concept of associating mixtures

Then various modifications of SAFT were suggested over time, namely:

- CK-SAFT - which uses refinement of the dispersion term based on universal constants for argon
- LI-SAFT - which uses chain segments that are treated as Lennard-Jones spheres; a dipole-dipole interactions and dipole moment are also added
- SAFT-VR - which introduces dispersion potentials of variable range
- PC-SAFT - which considers dispersion attraction between entire chains
- SAFT-γ Mie - which uses the Mie potenntial for dispersion attraction

2.2 SAFT parameters

SAFT contains a number of parameters listed below:

- m - the number of segments in chain
- σ - the diameter of segments
- ϵ - the dispersion energy of interactions between the segments
- ϵ^{AB} - the association energy of interaction between A site and B site (see figures 7 and 8)
- K^{AB} - the volume interaction between A site and B site

These parameters can be either obtained from relevant literature, regressed from pure component data, or calculated using group-contribution methods.

In the following part, some important concepts for understanding SAFT are introduced.

2.3 London dispersion forces

The first concept concerns London dispersion forces. These are the forces, that occur between atoms and between non-polar molecules as a result of the motion of electrons. Let us see, for example, helium. The electron cloud of a helium atom contains two electrons, which are normally expected to be equally distributed spatially around the nucleus. At any moment, the electron distribution may be uneven resulting in an instaneous dipole. This weak and temporary dipole can influence neighboring atoms through electrostatic attractions and repulsion. This can induce a dipole in nearby athom. The instaneous and induced dipoles are weakly attracted to one another, as can be seen in Figure 9.

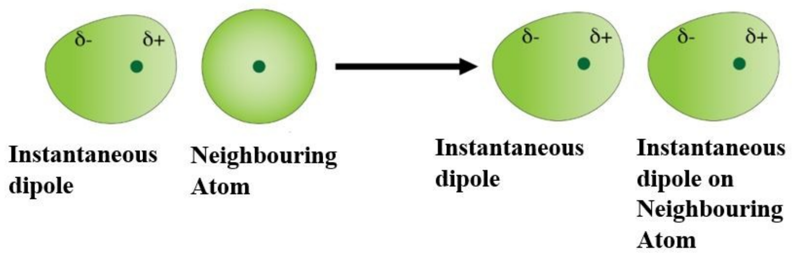

Figure 9: Illustration of London forces influencing neighbouring atom (https://stringfixer .com/)

Due to the affecting atom being slightly positive and the affected atom being slightly negative, the attraction between those two atoms is made. These atoms could be a segments in the SAFT mulecule. The London dispersion forces are important part of understanding SAFT, because they explain the attractive part between the individual segments of a system.

2.4 Pauli Repulsion

The Pauli repulsion has its origin in the quantum mechanical nature of electrons. Two electrons with the same spin cannot occupy the same orbital. If the atoms start to approach each other and the electrons of the same spin start to overlap in these orbitals, then they are pushed away. This can also be explained in terms of diminishing the electron density at the internuclear region. This results in a decreased screening of the nuclear-nuclear Coulomb repulsion.

If two atoms are being squezed together and the electron density is receding in the area between them (see Figure 10), the positive changes in both nuclei are not shielded too much. The reduced screening of those is obtained and so the positive charges start to repel each other. This is mechanistic understanding of the repulsion between non-polar molecules or atoms as they approach each other.

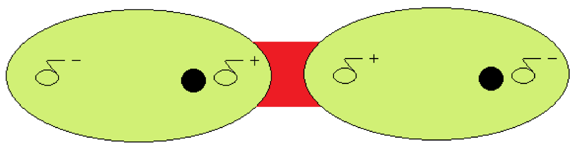

Figure 10: Ilustration of Pauli repulsion (https://www.sciencedirect.com/)

2.4.1 Lennard-Jones potential

In terms of having an equation to describe the Pauli Repulsion, one of the most popular approaches is to use the Lennard-Jones potential. It is gained by a very effective equation in statistical modeling of the chemical systems.

$$u(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right], \tag{2}$$

where:

- $u(r)$ is the intermolecular potential energy between a pair of neutral atoms and molecules
- ε is the depth of the potential well
- σ is the distance at which the inter-particle potential is zero
- r is the distance between the particles

In the figure 11 is shown a graph of the Pauli repulsion and dipole-dipole attraction, depending on distance and potential energy between two particles.

In the part of the graph where the repulsion occurs, the two particles repel each other. It is not the favorable state. On the other hand in the attraction part, the potential is lower than 0 and the attraction between the pair of particles is obtained. This is the favorable state.

2.4.2 Square-Well potential

Another approach to mathematical description of Pauli repulsion is called the Square-well potential. Its parameters are the same as in the previous equation. However, the λ parameter is added, which describes the reduced well-width. It applies that:

- $u(r) = \infty$ for $r < 0$
- $u(r) = -\varepsilon$ for $\sigma \leq r < \lambda\sigma$
- $u(r) = 0$ for $r \geq \lambda\sigma$

The graph of the Square-well potential tries to simplify the Lennard-Jones potential as can be seen in figure 12.

2.4.3 Modified Square-Well potential

If there is an interest in a better approximation of the Lennard-Jones potential, the modified version of the Square-Well potential must be used. It is also considered in PC-SAFT equation of state. The same parameters are used here as in the unmodified version, with the addition of s_1, which is a constant for which applies that

$$\frac{s_1}{\sigma} = 0.12 \tag{3}$$

It also applies:

- $u(r) = \infty$ for $r < (\sigma - s_1)$
- $u(r) = -3\varepsilon$ for $(\sigma - s_1) \leq r < \sigma$
- $u(r) = -\varepsilon$ for $\sigma \leq r < \lambda\sigma$
- $u(r) = 0$ for $r \geq \lambda\sigma$

To sum up the chapters 2.3 and 2.4, the London Dispersion forces were described, in terms of what is attracting the individual segments. The Pauli repulsion was discussed, in terms of what is repelling the individual segments and a number of potential-well models were listed, that can describe this behavior.

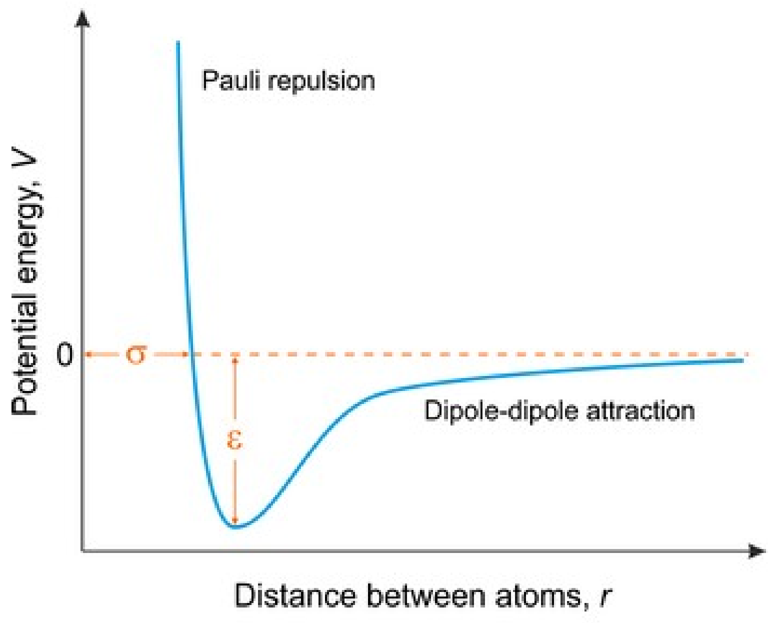

Figure 11: Pauli repulsion and dipole-dipole attraction between two particles illustrated by Lennard-Jones potential (https://glossary.periodni.com/)

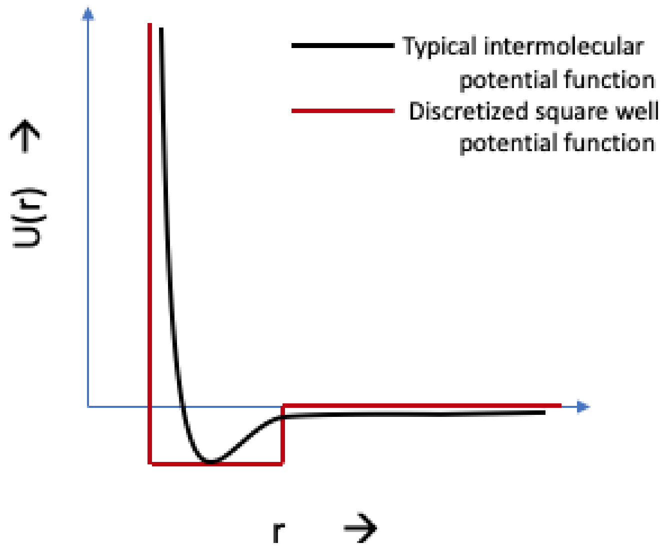

Figure 12: Comparsion of the Lennard-Jones and the Square-well potential (https://slidetodoc .com/)

2.5 Helmholtz free energy

Some of the properties that can be calculated and determinated by SAFT will now be discussed. The most important concept, related to the SAFT is the Helmholtz free energy.

The Helmholtz free energy is the capacity of a closed system at a constant temperature and volume and is defined as:

$$A = U - TS, \tag{4}$$

where:

- $A[J]$ is the Helmholtz free energy
- $U[J]$ is the internal energy of the system
- $T[K]$ is the absolute temperature
- $S[JK^{-1}]$ is the entropy of the system

Two properties which are useful to be defined are the reduced property and the reduced quantity. Reduced property is a property scaled by the corresponding critical temperature ($T_r = T/T_C$) , i.e. by the critical pressure ($P_r = P/P_C$). Reduced quantity is a dimensionless quantity expressed as the ratio of two dimensionally equal quantities. It is equal to Reduced Helmholtz free energy, which is the result of the relationship below:

$$\tilde{a} = \frac{A}{NkT}, \tag{5}$$

where N is the total numbers of molecules and $k[JK^{-1}]$ is the Boltzman constant.

The differences between a real liquid and an ideal solution property while maintaining the same pressure, temperature and composition is expressed by the excess property. Similary, the differences between a real liquid and an ideal solution property under the same conditions is expressed by the residual property. Therefore, the residual Helmholtz free energy can also be calculated according to this formula:

$$\tilde{a}^{res} = \frac{A^{res}}{NkT}, \tag{6}$$

where A^{res} [J] is the residual Helmholtz free energy.

SAFT is used to calculate the residual Helmholtz free energy. and subsequently this value is used to calculate the number of properties, which will be seen below.

2.5.1 Helmholtz free energy and the PC-SAFT EoS

The following equation applies:

$$\tilde{a}^{res} = \tilde{a}^{hc} + \tilde{a}^{disp} + \tilde{a}^{assoc}, \tag{7}$$

where:

- $\tilde{a}^{res}$ is the residual Helmholtz free energy
- $\tilde{a}^{hc}$ is the contribution of hard-chain system, which differs for each SAFT version
- $\tilde{a}^{disp}$ is the contribution due to dispersion attraction
- $\tilde{a}^{assoc}$ is the contribution due to the associating interactions

This is a starting point for the SAFT calculations. Additional equations are given, which are helpful in determining the individual members of the equation.

2.5.2 Parameters obtained by PC-SAFT EoS

This last part of the theoretical description of SAFT shows what can be calculated by SAFT equations of state based on reduced Helmholtz free energy.

First of all, let us mention the compressibility factor Z, which describes the deviation of a real gas from ideal gas at the same temperature and pressure. It is a useful thermodynamic property for modifying the ideal gas law to account for the real gas behaviour.[6]

It applies:

$$Z = 1 + \eta \left(\frac{\partial \tilde{a}^{res}}{\partial \eta} \right)_{T,x_i}, \tag{8}$$

where η is a packing factor and temperature and composition are fixed.

The most important output of PC-SAFT EoS is, in context of this work, the saturated vapor pressure and corresponding saturated vapor density, together with the saturated liquid density, whereas those outputs determines the equilibria between the phases of the substance (as discussed in chapter 1.3). Those will also be important outputs of the $pcsaft.py$ library, which is described in the programming section of this thesis.

For pressure P applies:

$$P = ZkT\rho, \tag{9}$$

where:

- Z is the compressibility factor
- $k[JK^{-1}]$ is the Boltzmann constant
- $T[K]$ is the absolute temperature
- $\rho[m^{-3}]$ is the number density of molecules

Another output from PC-SAFT EoS are fugacity coefficients, for which applies:

$$ln\phi_k = \frac{\mu_k^{res}(T, v)}{kT} - ln(z), \tag{10}$$

where ϕ is the fugacity coefficient and μ^{res} is residual chemical potential. The residual chemical potential can be obtained according to this relationship:

$$\frac{\mu_k^{res}(T, v)}{kT} = \tilde{a}^{res} + (z - 1) + \left(\frac{\partial \tilde{a}^{res}}{\partial x_n} \right)_{T,V,x_{i \neq k}} - \sum_{j=1}^{N} \left[x_j \left(\frac{\partial \tilde{a}^{res}}{\partial x_j} \right)_{T,V,x_{i \neq j}} \right], \tag{11}$$

where

$$\left(\frac{\partial \tilde{a}^{res}}{\partial x_n} \right)_{T,V,x_{i \neq k}}$$

is a partial derivative of the reduced residul Helmholt'z free energy with respect to mole fraction of component k at constannt T, V and constant composition.

The last three outputs are residual molar enthalpy, for which it applies:

$$\frac{\hat{h}^{res}}{RT} = -T\left(\frac{\partial \tilde{a}^{res}}{\partial T}\right)_{\rho,x_i} + (z-1), \tag{12}$$

residual molar entropy, for which it applies:

$$\frac{\hat{s}^{res}}{R} = -T\left[\left(\frac{\partial \tilde{a}^{res}}{\partial T}\right)_{\rho,x_i} + \frac{\tilde{a}^{res}}{T}\right] + ln(z), \tag{13}$$

and residual molar Gibbs free energy, for which it applies:

$$\frac{\hat{g}^{res}}{RT} = \tilde{a}^{res} + (z-1) - ln(z). \tag{14}$$

2.6 A brief summary of previous chapters

The previous chapters have clarified the relationships between the phases of the substance and the temperature and the pressure in which the substance is studied. It has been reported that there exist a saturation temperature and a saturation pressure at which the substance equilibrates between its condensed and vapor phases. There is also the so-called vapor temperature and vapor pressure, in which the substance is in balance between its vapor and gaseous phase. These values are very important from a physical point of view, and therefore there is a mathematical-physical apparatus by which they can be obtained. This is called SAFT. To fully understand it, it is necessary to get acquainted with a number of partial concepts that SAFT solves. These individual concepts have been elaborated in the previous chapters.

It is now already known, although certainly not in the greatest detail, on what SAFT is based and what results (values) it can provide to scientists. However, this description is sufficient as an introduction to this issue. There is no need to go into more detail, because the core of this work is more programming than physical. The *pcsaft* library, programmed in the Python programming language, was provided to us by the colleagues from the Institute of Thermomechanics of the Czech Academy of Sciences (IT CAS). This library can return different values, according to the theoretical description in the previous chapters. These values will be used in the programming part, where after their calculation, it will be possible to compare them with the measured data. The PC-SAFT EoS parameters can be optimized so that the resulting curve corresponds to the measured data. This optimization is described below.

3 Non-linear least squares

The least squares method is an approach in regression analysis to approximate solution of predetermined systems by minimizing the sum of squares of residues, that is the difference between the observed values and values provided by the model (in this case, values provided by PC-SAFT EoS). There are two categories of least squares problems. A linear problem is usually solved by iterative refinement; in each iteration, the non-linear system is approximated by the linear one. Therefore, the calculations are similar [7].

This chapter presents the theory of different approaches and methods for solving non-linear least squares problem. The steepest descent method and the Gauss-Newton method are presented here theoretically and have not been tested in the program. This is because the Levenberg-Marquardt algorithm, which was primarily implemented, is a combination of them. The following methods - trust region method and its modification dogbox method are also tested and their comparison with the Levenberg Marquardt algorithm is given in chapter 5. Furthermore, the simplex method is described in this chapter, which represents an interesting and perhaps a little less common approach for solving non-linear least squares problems. Therefore, it is definitely worth mentioning, because it makes it possible to avoid the various difficulties and inaccuracies that occur in classical methods [16].

3.1 Introduction to descent methods

The following text concerning the descent methods is inspired by the article [8].

Most methods for non-linear optimization are iterative and most of them have measures which enforce the descending condition:

$$F(x_{k+1}) < F(x_k)$$

,

where F is the cost function.

This condition makes less probable that the algorithm moves towards a saddle point. If the given function has more than one minimizer, then which of them will be found after using the algorithm, depends on the choice of initial estimate x_0 . Let x^* be a local minimizer for the given function. When x_0 is far from the solution, the descent method should produce iterates, which move steadily towards x^*. In the global stage of iteration, the errors are wanted to be just decreasing, except in the very first step, i.e.

$$||e_{k+1}||< ||e_k||,$$

for $k > K$, where e_k denotes the current error and $e_k = x_k - x^*$. By the end of the iteration, x_k is approaching x^* and faster convergence is required. There are three types of convergences:

- Linear convergence, where $||e_{k+1}||< a||e_k||$ for $||e_k||$ small and $0 < a < 1$,

- Quadratic convergence, where $||e_{k+1}||= \sigma(||e_k||^2)$ for $||e_k||$ small,
- Superlinear convergence, where $\frac{||e_{k+1}||}{||e_k||} \to 0$ for $k \to \inf$

The descent methods consist of following two parts: to find a descent direction $\boldsymbol{h_d}$ and to find a step length giving a good decrease in the F-value.

Below, the pseudocode of descent method is illustrated.

```
begin
k := 0 ; x := x_0 ; found := false {starting point}
while ( not found) and (k < k_max)
h_d := search_direction(x) {from x and downhill}
if (no such h exist)
found := true
else
alpha := step_length(x, h_d)
x := x + alpha h_d; k := k+1 {next iterate}
end
```

Let us consider the variation of the F-value along the half line starting at x and with the direction $\boldsymbol{h}$. From the Taylor expansion for F differentiable and smooth enough, it applies:

$$F(\boldsymbol{x} + \boldsymbol{h}) = F(x) + \boldsymbol{h}^T g + \frac{1}{2}\boldsymbol{h}^T \boldsymbol{H}\boldsymbol{h} + \sigma(||\boldsymbol{h}||^3), \tag{15}$$

We see that:

$$F(\boldsymbol{x} + \alpha\boldsymbol{h}) = F(\boldsymbol{x}) + \alpha\boldsymbol{h}^T \boldsymbol{F}'(\boldsymbol{x}) + \sigma(a^2) \approx F(\boldsymbol{x}) + \alpha\boldsymbol{h}^T \boldsymbol{F}'(\boldsymbol{x}) \tag{16}$$

for α sufficiently small.

3.1.1 Descent direction

We say that $\boldsymbol{h}$ is a descent direction if $F(\boldsymbol{x} + \alpha\boldsymbol{h})$ is decreasing function of α at $\alpha = 0$. We also say that $\boldsymbol{h}$ is a descent direction for F at $\boldsymbol{x}$ if $\boldsymbol{h}^T \boldsymbol{F}'(\boldsymbol{x}) < 0$. If no such $\boldsymbol{h}$ exist, then $\boldsymbol{x}$ is stationary. Otherwise, the α have to be chosen, i.e. how to move from $\boldsymbol{x}$ in the direction given by $\boldsymbol{h}_d$, to get a decrease in the value of the objective function. One way to achieve it is to find $\alpha_s = argmin_{\alpha>0}\{F(\boldsymbol{x} + \alpha\boldsymbol{h})\}$. This process is called a line search.

3.1.2 Steppest descent method

It is obvious that while performing a step $\alpha\boldsymbol{h}$ with $\alpha > 0$, the relative gain in function values satisfies:

$$\lim_{\alpha\to 0} \frac{F(\boldsymbol{x}) - F(\boldsymbol{x} + \alpha\boldsymbol{h})}{\alpha||h||} = \frac{1}{||h||}\boldsymbol{h}^T \boldsymbol{F}'(x) = -||\boldsymbol{F}'(\boldsymbol{x})||cos\phi, \tag{17}$$

where ϕ is the angle between the vectors $\boldsymbol{h}$ and $\boldsymbol{F}'(\boldsymbol{x})$. It follows acquirement of the greatest gain rate if $\phi = \pi$, i.e. if the steepest descent direction $\boldsymbol{h_{sd}}$ is used, which is given by

$$\boldsymbol{h_{sd}} = -\boldsymbol{F}(\boldsymbol{x}). \tag{18}$$

If the descent direction $\boldsymbol{h}_d$ is replaced in the pseudocode by $\boldsymbol{h}_{sd}$, the steppesst descend, also called as the Gradient descent method is obtained. The disadvantage of steepest descent method is, that it is linear and often very slow.

3.2 Gauss - Newton method

The text describing the Gauss-Newton method is based on the article [9].

Consider a set of m datapoints $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$ and a model function $y = f(x, \boldsymbol{\beta})$, that, in addition to the variable x, also depends on n parameters $\boldsymbol{\beta} = (\beta_1, \beta_2, ..., \beta_n)$, where $m \geq n.$. It is desired to find the vector $\boldsymbol{\beta}$ of parameters, such that the curve fits best the given data, in the least squares sence. That is, the sum of squares:

$$S = \sum_{i=1}^{m} r_i^2 \tag{19}$$

is minimized, where the residuals (in sample prediction errors) r_i are given by $r_i = y_i - f(x_i, \boldsymbol{\beta})$ for $i = 1, 2, ..., m$.

Initial values must be used for parameters. Then, the parameters are refined iteratively, that is, the values are obtained by succesive approximation

$$\beta_j \approx \beta_j^{k+1} = \beta_j^k + \Delta\beta_j, \tag{20}$$

were k is an iteration number and $\Delta\boldsymbol{\beta}$ is the vector of increments.

At each iteration, the model is linearized by approximation to a first-order Taylor polynomial expansion about $\boldsymbol{\beta}^k$. It applies:

$$f(x_i, \boldsymbol{\beta}) \approx f(x_i, \boldsymbol{\beta}^k) + \sum_j \frac{\partial f(x_i, \boldsymbol{\beta}^k)}{\partial \beta_j}(\beta_j - \beta_j^k) = f(x_i, \boldsymbol{\beta}^k) + \sum_j \mathbb{J}_{i,j} \Delta\beta_j \tag{21}$$

Jacobian $\mathbb{J}$ is a function of constants, the independent variable and the parameters, so it changes from one iteration to the next.

Residuals

$$\frac{\partial r_i}{\partial \beta_j} = -\mathbb{J}_{i,j} \tag{22}$$

are given by

$$\Delta y_i = y_i - f(x_i, \boldsymbol{\beta}^k), \tag{23}$$

$$r_i = y_i - f(x_i, \boldsymbol{\beta}) = (y_i - f(x_i, \boldsymbol{\beta}^k)) + (f(x_i, \boldsymbol{\beta}^k) - f(x_i, \boldsymbol{\beta})) \approx y_i - \sum_{s=1}^{n} \mathbb{J}_{i,s} \Delta \beta_s. \tag{24}$$

By substitution of these expressions into the gradient equations, we get:

$$-2 \sum_{i=1}^{m} \mathbb{J}_{i,j} (\Delta y_i - \sum_{s=1}^{n} \mathbb{J}_{i,s} \Delta \beta_s) = 0, \tag{25}$$

which is n simultaneous linear equations. Rewritten in matrix notation, it applies:

$$(\mathbb{J}^T \mathbb{J}) \boldsymbol{\Delta\beta} = \mathbb{J}^T \boldsymbol{\Delta y}. \tag{26}$$

The observations does not have to be equally reliable, so suitable weight might be added to each, in form of a weight diagonal matrix $\boldsymbol{W}$. Then, the normal equations will be - in matrix notation - rewritten as:

$$(\mathbb{J}^T \boldsymbol{W} \mathbb{J}) \boldsymbol{\Delta\beta} = \mathbb{J}^T \boldsymbol{W} \boldsymbol{\Delta y}. \tag{27}$$

This equation may be solved for $\boldsymbol{\Delta\beta}$ by Cholesky decomposition.

The parameters are updated iteratively:

$$\boldsymbol{\beta}^{k+1} = \boldsymbol{\beta} + \boldsymbol{\Delta\beta}, \tag{28}$$

where k is an iteration number. Gauss-Newton method is adequate for simple models, but it cannot work with divergence, so there is always neccesary to implement some protection against it.

3.3 Levenberg-Marquardt method

3.3.1 Numerical differentiation

It will later be seen that in the Levenberg-Marquardt algorithm, Jacobian $\mathbb{J}$ is used, which denotes the gradient of the fitted function f with respect to its parameter vector P. The analytical derivatives of f might be sometimes unavailable.[10] Numerical differentiation techniques are used to approximate the derivative using forward differentiation formula:

$$\mathbb{J}_{i,j} = \frac{f_i(\boldsymbol{\beta_k} + h_j \boldsymbol{e_j}) - f_i(\boldsymbol{\beta_k})}{h_j} \tag{29}$$

or a central differentiation one:

$$\mathbb{J}_{i,j} = \frac{f_i(\boldsymbol{\beta_k} + h_j\boldsymbol{e_j}) - f_i(\boldsymbol{\beta_k} - h_j\boldsymbol{e_j})}{2h_j}, \tag{30}$$

where e_j is the unit vector in the β_j direction and h_j is a small increment.

It was proven that the use of numerical approximation in the Levenberg-Marquradt algorithm does not jeopardize its convergence properties.[11] It is also known that numerical differentiation is an unstable procedure inclinable to truncation and subtractive cancellation errors. Decreasing the step size h will reduce the trunctation error. Unfortunately a smaller step has the opposite effect on the cancellation error. Selecting the optimal step size for a certain problem is computationally expensive and the benefits achieved are not justifiable as the effect of small errors in the values of the elements of the Jacobian matrix is minor.[12] For this reason, the sizing of the finite difference step is not attempted and a constant increment size is used in evaluating the gradient. Furthermore, to allow for the flexibility to adjust to the specific problem characteristics, the individual size of each parameter step can be adjusted by the user.[10]

3.3.2 Levenberg-Marquardt algorithm

This text is based on Levenberg's original publication [13].

Consider a set of m data points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ and a curve (model function) $y = f(x, \boldsymbol{\beta})$ that in addition to the variable x also depends on n parameters, $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_n)$ with $m \geq n$. It is desired to find the vector $\boldsymbol{\beta}$ of parameters such that the curve fits best the given data in the least squares sence, that is, the sum of squares:

$$S = \sum_{i=1}^{m} r_i^2 \tag{31}$$

is minimized, where the residuals (in sample prediction errors) r_i are given by

$$r_i = y_i - f(x_i, \boldsymbol{\beta}) \tag{32}$$

for $i = 1, 2, ..., m$.

Levenberg-Marquardt is an iterative procedure. To start a minimization, an initial guess of parametres has to be made, that is the parameter vector $\boldsymbol{\beta}$. In cases with only one minima, the initial guess could be for example $\boldsymbol{\beta}^T = (1, 1, \ldots, 1)$. In cases with multiple minimum, the algorithm converges to the global minimum only if the initial guess is very close to the solution.

In each iteration step, the vector parameter $\boldsymbol{\beta}$ is replaced by a new estimate $\boldsymbol{\beta} + \boldsymbol{\delta}$. To determine $\boldsymbol{\delta}$, the function $f(x_i, \boldsymbol{\beta} + \boldsymbol{\delta})$is approximated by its linearization:

$$f(x_i, \boldsymbol{\beta} + \boldsymbol{\delta}) \approx f(x_i, \boldsymbol{\beta}) + \mathbb{J}_i\boldsymbol{\delta}, \tag{33}$$

where

$$\mathbb{J}_i = \frac{\partial f(x_i, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \tag{34}$$

is a gradient of f with respect to $\boldsymbol{\beta}$.

We can approximate $f(x_i, \boldsymbol{\beta} + \boldsymbol{\delta})$ and get:

$$S(\boldsymbol{\beta} + \boldsymbol{\delta}) \approx \sum_{x=1}^{m} [y_i - f(x_i, \boldsymbol{\beta}) - \mathbb{J}_i \boldsymbol{\delta}]^2, \tag{35}$$

or in a vector notation:

$$\begin{aligned} S(\boldsymbol{\beta} + \boldsymbol{\delta}) &\approx \|\boldsymbol{y} - f(\boldsymbol{\beta}) - \mathbb{J}\boldsymbol{\delta}\|^2 \\ &= [\boldsymbol{y} - f(\boldsymbol{\beta}) - \mathbb{J}\boldsymbol{\delta}]^T [\boldsymbol{y} - f(\boldsymbol{\beta}) - \mathbb{J}\boldsymbol{\delta}] \\ &= [\boldsymbol{y} - f(\boldsymbol{\beta})]^T [\boldsymbol{y} - f(\boldsymbol{\beta})] - [\boldsymbol{y} - f(\boldsymbol{\beta})]^T \mathbb{J}\boldsymbol{\delta} - (\mathbb{J}\boldsymbol{\delta})^T [\boldsymbol{y} - f(\boldsymbol{\beta})] + \boldsymbol{\delta^T} + \mathbb{J}^T \mathbb{J}\boldsymbol{\delta} \\ &= [\boldsymbol{y} - f(\boldsymbol{\beta})]^T [\boldsymbol{y} - f(\boldsymbol{\beta})] - 2[\boldsymbol{y} - f(\boldsymbol{\beta})]^T \mathbb{J}\boldsymbol{\delta} + \boldsymbol{\delta^T} + \mathbb{J}^T \mathbb{J}\boldsymbol{\delta} \end{aligned} \tag{36}$$

Taking the zero derivative of $S(\boldsymbol{\beta} + \boldsymbol{\delta})$ with respect to $\boldsymbol{\delta}$ and setting the result to zero gives

$$(\mathbb{J}^T \mathbb{J})\boldsymbol{\delta} = \mathbb{J}^T [\boldsymbol{y} - f(\boldsymbol{\beta})], \tag{37}$$

where $\mathbb{J}$ is the Jacobian matrix. It is a rectangular matrix of size $m \times n$, where n is the number of parameters, that is the size of vector $\boldsymbol{\beta}$. The matrix multiplication $(\mathbb{J}^T \mathbb{J})$ provides the required $n \times n$ matrix-vector product which can be solved for δ. A "damped version" of this equation has the form:

$$(\mathbb{J}^T \mathbb{J} + \lambda \mathbb{I})\boldsymbol{\delta} = \mathbb{J}^T [\boldsymbol{y} - f(\boldsymbol{\beta})], \tag{38}$$

where $\mathbb{I}$ is the identity matrix, giving the increment $\boldsymbol{\delta}$ to the estimated parameter vector $\boldsymbol{\beta}$.

The (non-negative) damping factor λ is adjusted at each iteration. If the reduction of S is rapid, a smaller value can be used, bringing the algorithm closer to the Gauss–Newton method. On the other hand, if an iteration gives insufficient reduction in the residual, λ can be increased, giving a step closer to the gradient-descent direction. Therefore, for large values of λ , the step will be taken approximately in the direction opposite to the gradient. If either the length of the calculated step $\boldsymbol{\delta}$, or the reduction of the sum of squares from the latest parameter vector $\boldsymbol{\beta} + \boldsymbol{\delta}$ fall below predefined limits, then the iteration stops, and the last parameter vector $\boldsymbol{\beta}$ is considered to be the solution.

3.4 Dogbox and trust region method

Trust region methods, the description which was taken from [15], fall into the cathegory of sequential quadratic programming. It is an iterative procedure. Let us assume objective function $f(x)$, which is represented by a quadratic model inside a suitable neighborhood, called the trust region of the current iterate, as implied by the Taylor series expansion. This model of $f(x)$ at the k^{th} iteration can be written as:

$$f(k_k + s) \approx m_k(s) = f(x_k) + g_k^T s + \frac{1}{2} s^T B_k s, \tag{39}$$

where $g_k = \nabla f(x_k)$ and B_k is the symmetric approximation to $\nabla^2 f(x_k)$. The trust region may be defined by:

$$T_k = \{x \in R^n \mid ||x - x_k|| \leq \Delta_k\}. \tag{40}$$

With the model and the trust region, a step s_k is to be found, with $||s_k|| \leq \Delta_k$, such that the model is sufficiently reduced in value. The reduction in the model is compared with the objective function. If they agree to a certain extend, the step is accepted and the trust region is either expanded or remains the same. Otherwise the step is rejected and the trust region is contracted. Algorithm below applies for the basic trust region:

- S0: Pick the initial point and trust region parameter x_0 and Δ_0 and set $k = 0$.
- S1: Construct a quadratic model: $m_k(s) \approx f(x_k + s)$
- S2: Calculate s_k with $||s_k|| \leq \Delta_k$, so as to sufficiently reduce m_k.
- S3: Compute the ratio of actual to expected reduction, $r_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(0) - m_k(s_k)}$. This value will determine if the step is accepted or not and it will determine the update of Δ_k.
- S4: Increment $k \leftarrow k + 1$ and repeat $S1$.

The dogbox algorithm uses a rectangular trust region. In each iteration, the following constrained quadratic problem is to be figured out:

$$\min m(p) = \frac{1}{2} p^T B p + g^T p \tag{41}$$

such that:

$$l \leq \tilde{p} \leq \tilde{u}$$

The dogleg method can be applied when B is positive definite. In this case, the Gauss-Newton step $p^N = -B^{-1} g$ and the Cauchy step, which is the unconstrained minimizer of the appropriate quadratic model along anti-gradient can be expressed as:

$$p^C = -\frac{g^T g}{g^T B g} g. \tag{42}$$

For the dogleg path it applies:

$$p(\tau) = \tau p^0 \tag{43}$$

for

$$0 \leq \tau \leq 1$$

and

$$p(\tau) = p^C + (\tau - 1)(p^N - p^C) \tag{44}$$

for

$$1 < \tau \leq 2.$$

First, the algorithm moves towards the Cauchy point and then to the Newton point. It is proved that $m(p(\tau))$ is a decreasing function of τ, which means that as long as the algorithm stays in the trust region, it should follow the dogleg path. There is only one intersection of the dogleg path and the boundary. This makes the method simple and clear. This statement is not rule for a rectangular trust region; however, it is not hard to find the optimum along the dogleg path within the trust region. If any variable hits the initial boundary and the component of the anti-gradient point outside the operable region during the algorithm, the dogleg algorithm would not make any progress. Such variables satisfy the first-order optimality and they should be excluded before taking a next dogleg step. If B is positive semidefinite, there is no proper Newton step and the regularized Newton step should be computed. This is done by increasing the elements of B by a proper amount. The dogleg method works well for the not-rank-deficient Jacobian in least squares problem.

3.5 Simplex method

3.5.1 Problems of classical optimization algorithms

A slightly different approach that can be used when fitting data is the simplex algorithm. The description of this algorithm is taken from [16]. This method that was proposed by J. A. Nedler and R. Mead in 1965 has been used for data fitting since the 1980s. Some equations that are not linear are usually an attempt to relinearearize. The methods listed above can then be used. However, the main disadvantage of them is that many functions cannot be reduced to linear equations. In some cases, it is not considered appropriate to try to transform a nonlinear system into a linear one, even when it is possible. Not only is this approach cumbersome, but the error distribution and statistical weight of data points can change after these transformations. If these issues are not resolved properly, the fit result may be inaccurate.

The second disadvantage is a common feature of the above methods - their recursiveness. It is usually not clear at the beginning how many steps will need to be taken to obtain a solution. Sometimes it is not even clear whether any solution can be achieved at all. Another disadvantage is that the above methods require an initial estimate of the parameters (in this work, estimation of the parameters of the PC-SAFT EoS for the investigated substance). If this estimate is incorrect, the optimization algorithm may not find the correct solution.

Other specific disadvantages of these commonly used optimization algorithms are as follows: The stepwise descent strategy, which consist of adjusting one parameter at a time, sequentially, until a minimum is found and subsequent repetition of processes unitil the values are stable, is easy to program, however, it can be very slow. The steepest descent method adjusts the parameter values along the direction of the fastest decrease. It involves fewer iterations, but it requires a knowledge or computation of the new function's first derivatives. For this purpose, a numerical differentiation is used (see 3.3.1). The Gauss-Newton method (see 3.2) is fast and relatively efficient, but with a poor initial estimate it may not converge to a solution. Considerable truncation errors might occur in the partial derivatives calculations and in the matrix inversions, particulary, when the installation does not support double-precision formats. Moreover, the numerical calculation of partial derivatives at each repetition requires massive amounts of computations.

Although the Levenberg-Marquardt algorithm is used in this work, it is not perfect and suitable for all cases. It avoids the problem of divergence from Newton's method, and its speed is still acceptable. The amount and complexity of code generated in this way can, however, become substantial.

3.5.2 Simplex algorithm

A simplex is a geometric figure. It has one more vertex than is the number of the dimensions of the corresponding space. For example, a simplex on a two-dimensional surface is a triangle; a simplex in three-dimensional space is tetrahedon, and so on. Let M be the number of simplex vertices. The idea of the simplex algorithm is to build a simplex in the M+1 dimensional space described by the parameters which will further be fitted. To obtain the smallest value of the function, the program shifts the simplex "downhill", accelerating and slowing down as needed. It finds a vertex which has the highest (worst) response and a vertex with the lowest (best) response. Then, it rejects the highest response and substitutes another one for it. Then, the new vertex is computed, according to mechanisms, such as the reflection, expansion, contraction and shrinkage.

The description of the algorithm is described in the items below:

1. To create the reflected vertex, let d be the distance from the worst vertex to the midpoint of all the other vertices M. The reflected vertex is located at a distance d from M on the line continuation that joins the rejected vertex to M.
2. If the response of reflected vertex is neither worse than the rejected one nor better than the best in the simplex, then the algorithm accepts it. If it is lower (better) than the previous best response, the algorithm tests an expansion by reflecting the distance d twice.
3. The expanded vertex is accepted if it has a better response than the rejected one; otherwise the program accepts the reflected one.
4. If the response of the reflected vertex is worse than the response of the rejected vertex, the program tests a contraction by moving the rejected vertex a distance of half of d towards the midpoint M. This contracted vertex is accepted if it produces a better response than the rejected one; otherwise a shrinkage occurs and all vertices , except the best one, move directly toward it by half of their original distance from it.

These four items are illustrated in the figure 13.

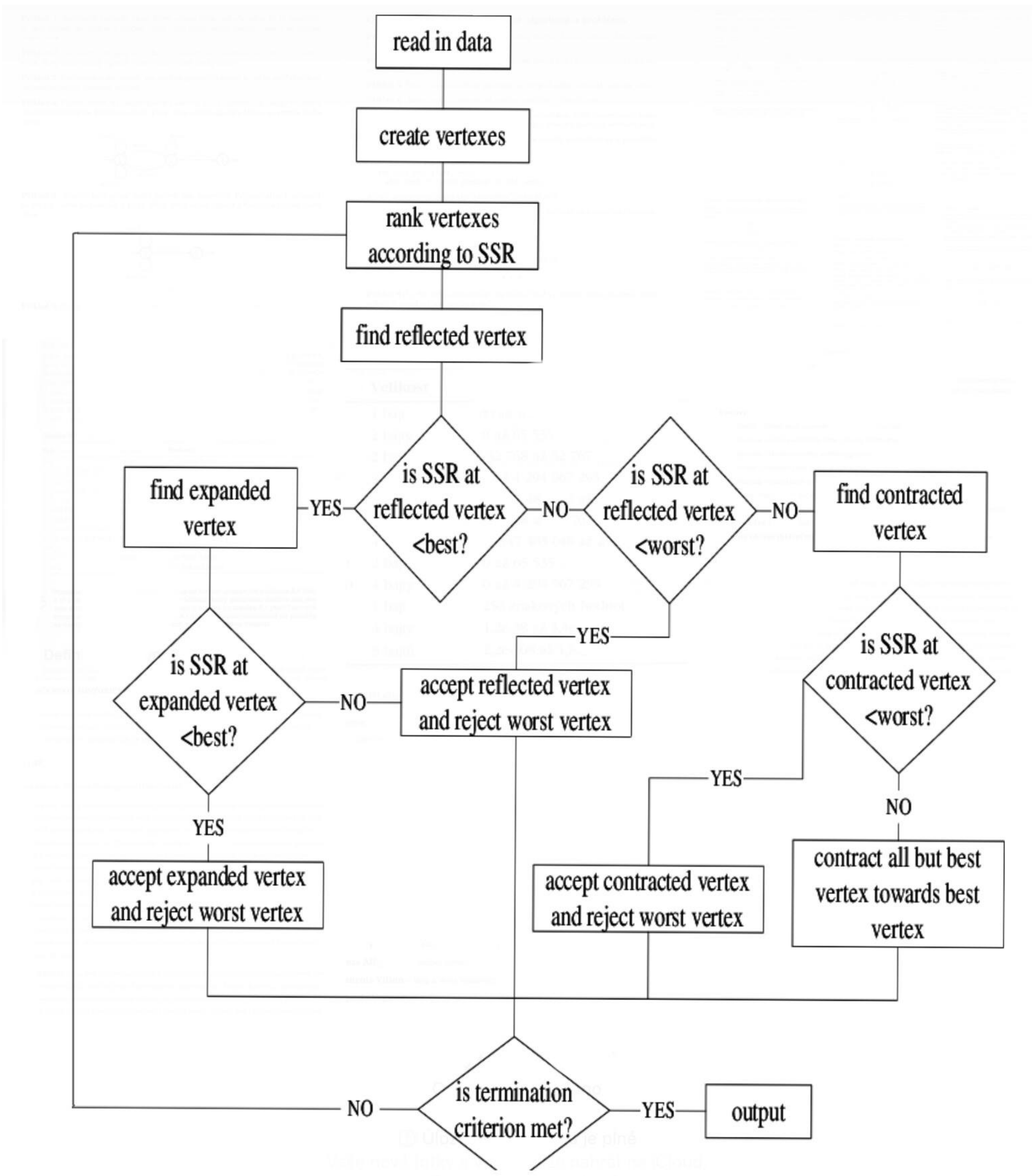
read in data
create vertexes
rank vertexes according to SSR
find reflected vertex
is SSR at reflected vertex <best?
YES
find expanded vertex
NO
is SSR at reflected vertex <worst?
NO
find contracted vertex
YES
is SSR at expanded vertex <best?
NO
accept reflected vertex and reject worst vertex
is SSR at contracted vertex <worst?
YES
NO
YES
accept expanded vertex and reject worst vertex
accept contracted vertex and reject worst vertex
contract all but best vertex towards best vertex
NO
is termination criterion met?
YES
output

Figure 13: Flowchart of simplex algorithm (https://www.researchgate.net/)

Various modifications of the simplex method can be implemented, for example the substitution of the shrinkage with a contraction beyond point M. The coefficient of the reflection, expansion and contraction can also be changed, within certain limits. The biggest advantages of Simplex method are listed below:

- Divergence is impossible

- The response value is needed to be computed only once or at most a few times for each iteration
- No knowledge of derivatives or numerical differentiation is necessary. This avoids rounding-off errors and allows the handling of non-continuous functions.
- No matrix operation is involved.

4 Introduction to optimization in Python programming language

4.1 Python programming language

As mentioned, the optimization of the PC-SAFT equation of state parameters is performed here in the Python programming language. It has obvious advantages in usage and is very suitable even for beginners. It is a high level programming language, which is very similar in structure to everyday English. The code written in Python is visually clear because the individual blocks of statements are separated by a tab. However, this can sometimes be problematic in terms of proper formatting when transmitting a code. Another advantage of this language is that there are a number of resources that can be found on the Internet, which can be helpful in any difficulty. At present, these resources include internet forums, where programmers exchange experiences and help each other, for example Geeks for Geeks, or stackowerflow.com. These forums were also helpful in working on this project.

But what turns out to be the biggest advantage of Python over other programming languages is a wide variety of libraries that Python has. The program in this work works with the $SciPy$ library, which is very good for scientific work. It is free and an open-source library, which contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. It consists of many sub-packages, from which *linalg* was used in this project for calculations in linear algebra and *optimize*, where optimization algorithms can be found, such as the Levenberg-Marquardt algorithm.

When working with the Python programming language, it is possible to choose from a number of development environments, some of which are free. Probably the best known open-source development environment is called Spyder, which was initially used in this project. Then it was replaced by the similarly known PyCharm platform. The PyCharm platform has proven to be very compatible with the Git project development environment and its Bitbucket online repository. After understanding this system of working on the project, it turns out that it is very advantageous in terms of clarity, mutual sharing and backup of the entire project.

In the Python programming language, mathematical problems are easy to solve from the programming point of view, because Python contains a whole list of libraries that make it easier for the programmer to work with the mathematical apparatus. For example, in this program, the library *numpy* is used from which the functions *inner*, *diag*, *eye* and *solve* are taken, which offers the programmer the opportunity to work effectively in the field of linear algebra

The use of the Levenberg-Marquardt algorithm is in the Python programming language possible in basically two ways. Either one of the functions from the built-in scipy library, or our own function can be used to solve the Levenberg-Marquardt algorithm. The second option does not seem to be a very complicated problem, especially because of the libraries that can be used in Python. In this program, it was decided to write our own code that will be able to effectively use the Levenberg-Marquardt algorithm and then compare its results with the results obtained after using the SciPy library function called $least\_squares$.

4.2 Optimization of quadratic equation

For correct understanding and easier implementation of the algorithm itself, a simple optimization of the quadratic function

$$f(x) = ax^2 + bx + c$$

is first described here. It was used for testing the individual properties of the Levenberg-Marquardt algorithm. As default parameters were choosen $a = -2.5$, $b = 15.3$ and $c = 1$. Troughout the whole chapter, all graphs and text will relate to these parameters.

4.2.1 Program description

The whole program, called *test_main.py*, is included to this thesis as an attachement. At the very beginning, two functions are defined: *line* and similar *line_with_noise*. In the former, the curve to be fited is entered and it is also the only place in the program where this functional rule needs to be changed. The *line_with_noise* function has a slightly modified functional rule so that its individual points are positively or negatively deviated . This is basically done as a simulation of separate points, which will later be obtained from real measurements and which will be interpolated with a correctly chosen curve. In the definition of another function called *line_error*, the difference between observations and estimates is calculated. Also here, observation means a broken original function.

Numerical differentiation has been described in the previous section of this text. The $numer-ical\_differentiation$ function is built on these principles. This function returns the Jacobian of numerical derivatives for the $error\_function$. The parameters a, b, c of the selected equation are updated here by $delta\_factor$. It can be set differently, in this program it was initially set to 1e(-4). The choice of the delta factor could be important in working with more extensive functions. However, when fitting a simple function, such as the quadratic function here, it turned out that no matter how it is chosen, its change does not affect the accuracy of the resulting parameter values. As clarified in 3.3.1, either forward or central numerical differentiation can be used. The program, originally written with the forward numerical differentiation, did not give results with good accuracy, as can be seen from the absolute deviation plot shown in figure 14

Due to our aim to achieve the most accurate results, this discrepancy had to be eliminated. This was achieved by changing the forward numerical differentiation to a central one. The program modified in this way gives better results as can be seen in figure 15. Here, the curves of absolute deviations after the use of our function and library function overlaps. In the program itself, forward numerical differentiation remained commented, for the possibility of comparison with the central one.

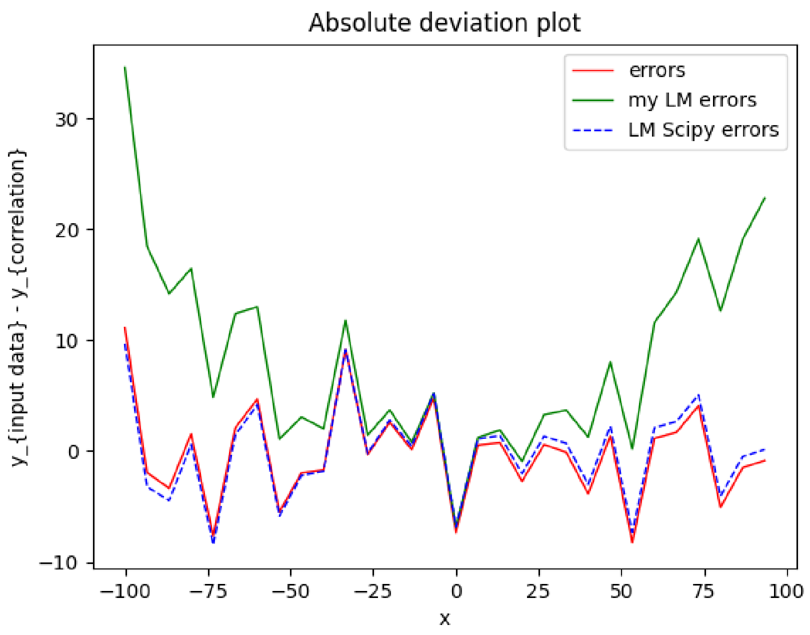

Figure 14: Absolute deviation plot after using forward numerical differentiation, for $np.linspace = (-100, 100, 30)$

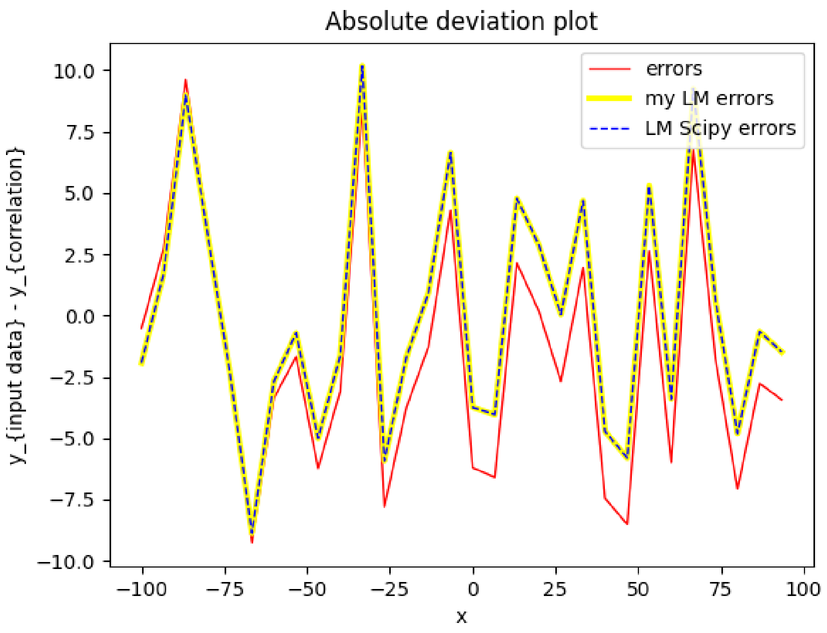

Figure 15: Absolute deviation plot after using central numerical differentiation, for $np.linspace = (-100, 100, 30)$

The Levenberg-Marquardt algorithm itself is hidden in the *LM* function, which takes several arguments:

- *initial_guess*, which is the initial estimate for the parameters to be found
- *args*, which are the inputs x and observations y
- *error_function*
- *jacobian_function*, by which the Jacobian of function gradients with respect to the parameters is obtained.
- *llambda*, which is the initial dampening factor.
- *lambda_multiplier*, which is the scale used to increase/decrease lambda
- *kmax*, which is the maximal number of iterations. A relatively small number is enough to achieve the optimal result (for simple function)

As in the function *numerical_differentiation*, the change of various factors, here specifically *llambda*, *lambda_multiplier* or *kmax* does not affect the course or result of fitting a simple function.

The *LM* function works exactly according to the principles described in the theoretical part of the description of the Levenberg-Marquardt algorithm. The function returns the optimized parameters a, b, c. The algorithm finds almost the exact parameters even in its third iteration. The parameters in its third step are $[-2.5232, 15.2757, 1.8393]$. Figure 16 demonstrates the third step of Levenberg-Marquardt algorithm:

In the test function *main*, the option of *np.linspace*, the entered parameters of the equation and their initial estimate is determined at the beginning. Then the library and non-library functions calculating the Levenberg-Marquardt algorithm are called. In figure 17 can be seen, how the measured data (or in this case broken line) are fitted by curves obtained by our *LM* function and *least_squares* library function. A small *np.linspace*$(-10, 10, 20)$ is selected here, for better orientation in the graph.

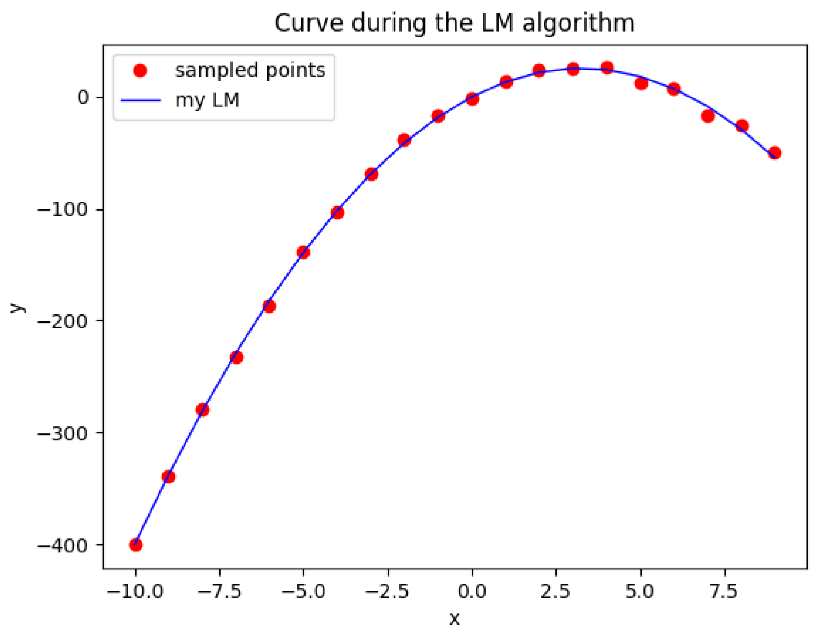

Figure 16: Graph shoving the third step of *LM* function, the actual parameters are very close to the desired ones

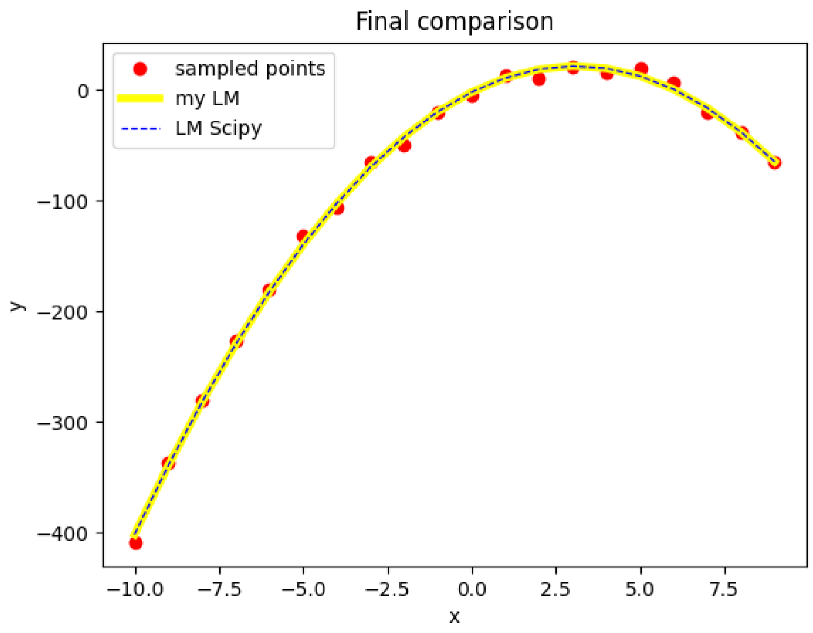

Figure 17: Graph shoving the broken function and the line with parameters obtained from *LM* function and *least_squares* function from scipy

4.2.2 Conclusions

After comparing the library *least_squares* function and our own *LM* function, it turned out that both functions return parameters almost identical for all ranges *np.linspace*.

For clarity, a table is inserted here, with several recorded experiments:

np.linspace	Type of differentiation	Initial guess	parameters given	parameters obtained LM	parameters obtained least_squares
(-100, 100, 30)	Forward	(-5, 10, 3)	(-2.5, 15.3, 3.1)	(-2.5, 15.28250763, 0.53947241)	(-2.49992851, 15.28781812, 0.51666433)
(-100, 100, 30)	Central	(-5, 10, 3)	(-2.5, 15.3, 3.1)	(-2.50007505, 15.32171975, 1.21011208)	(-2.50007505, 15.32171973, 1.2101013084)
(-10000, 10000, 10000)	Forward	(-5, 10, 3)	(-2.5, 15.3, 3.1)	(-2.5025473, 15.29469932, 203.8504393)	(-2.5, 15.299990, 1.12169254)
(-10000, 10000, 10000)	Central	(-5, 10, 3)	(-2.5, 15.3, 3.1)	(-2.5, 15.3000115, 0.9800674)	(-2.5, 15.30001151, 0.91009507)

Due to the great similarity between the results of our function and the built-in function, it can be said that our function has been programmed correctly for this simple quadratic equation. The next chapters will describe the application of this part of the program to more complex PC-SAFT equation of state and the optimization of its parameters.

5 Optimization of PC-SAFT EoS parameters for various substances

5.1 PC-SAFT EoS implementation

When optimizing the parameters of the PC-SAFT equation of state, we worked with several sub-libraries provided by colleagues from IT CAS, which are listed below for better orientation:

- *database_reader.py*, which retrieves data from Excel file
- *interface.py*, which is an interface for PC-SAFT EoS
- *example_call.py*, where the PC-SAFT EoS is tested
- *root_iter.py*, which has to be included to interface for some calculations

On the other hand, the program we implemented is devided into two parts:

- *utilities.py*, where the Levenberg-Marquardt algorithm is implemented
- *main.py*, where the Levenberg-Marquardt algorithm is used and tested

Programs received from colleagues from IT CAS must naturally be linked to our programs. All of those programs and libraries are included as an attachement to the thesis.

Statistical associating fluid theory was described in detail in chapter 2. PC-SAFT EoS itself is passed to the program as *.dll* file. Therefore, it is not necessary to deal with the theory of its calculations in terms of programming. It can be said that after a brief introduction to the issue of SAFT calculations, this *.dll* library can be taken as a kind of blackbox from which the values needed for the next optimization can be obtained. However, it is important what results can be obtained with the programmed PC-SAFT EoS and how to get them. In *interface.py* it is possible to see which values can be obtained and in the test section *example_call.py* is seen how they can be approached in practice.

For calculations, it is always necessary to set the parameters that enter the PC-SAFT EoS. Of course, this is done with regard to what outputs are needed, so some parameters can be set globally, some locally and some may not be set at all. In any case, here is a clear list of parameters (set by functions listed) that can enter the PC-SAFT EoS:

- *pcsaft.set_fluid*(), where the test substance is entered, it was methane when testing the program
- *pcsaft.set_molar_ratio*()
- *pcsaft.set_temperature*()
- *pcsaft.set_density*()
- *pcsaft.set_pressure*(), which is used for example for calculating the density by recalculating pressure

The output data include the saturation pressure(*p_sat*) and the density of the saturated liquid(*d_liq*), which are worked on further. These are obtained together from the *pcsaft.get_equil*() function, in which the pressure estimate for each iteration is entered as its parameter. This estimate is determined as the value of the previous saturation pressure calculated in the previous use of the *pcsaft.get_equil*() function.

Before implementing the Levenberg-Marquardt algorithm for PC-SAFT EoS, several questions arose at the beginning. The first of them being, whether the Levenberg Marquardt algorithm for the quadratic function was really well written. Of course, from the result of this simple optimization, it could be concluded that it was. However, in any case, some hidden errors could appear with a more complex optimization. Fortunately, this did not happen and the *numerical_differentiation*() and *LM*() functions could remain the same as in the example of the quadratic equation.

The Levenberg Marquardt algorithm was designed in such a way that different weights of the input parameters of the PC-SAFT equations could be taken into account when using it. Because, unlike, the quadratic function fitted above, here all parameters do not have to be equally "valuable" in the sense of influencing the results of the PC-SAFT equation of state. The value of the weight varies between 0 and 1. However, in this project, the different weights have not yet been considered and they all have a value of 1.

The quadratic function $ax^2 + bx + c$, that was implemented in the *line*() function, was replaced by the *pc_saft*() function. This function goes through the data from Excel, from which it gradually reads temperatures, for which it can then calculate the density of the saturated liquid and the saturation pressure. At the beginning of the optimization, it calculates them for the initial estimate of the parameters. It must be said that even a small change in this estimate has a large effect on the shape of the resulting function, as will later be seen in Figures 18 and 19 and that the estimated parameters should not differ much from the resulting ones, otherwise the calculation will fail. However, this fact has nothing to do with the accuracy or inaccuracy of PC-SAFT EoS calculations. It is given by the overall sensitivity of calculations of a similar type to any small change in conditions.

The function (or rather a series of data points) *line_with_noise*(), which was previously artificially created, is now replaced by the actual measured data for methane. The data here means the already mentioned saturated liquid density (referred to as *d_liq*) and saturated vapor pressure (referred to as *p_vap*).

As mentioned previously, the numerical differentiation and the Levenberg-Marquardt algorithm itself remained unchanged.

5.2 Levenberg-Marquardt optimization of methane and its comparison with other optimization algorithms

The first substance on which the optimization was tested is methane. The first initial estimate for methane parameters was $[0.9, 3.9, 160]$; the other parameters do not change and are zero. Figures 18 and 19 show the measured data for saturated pressure and saturated liquid density depending on the temperature (from 92 K to 180 K) and its PC-SAFT calculation with initially estimated parameters.

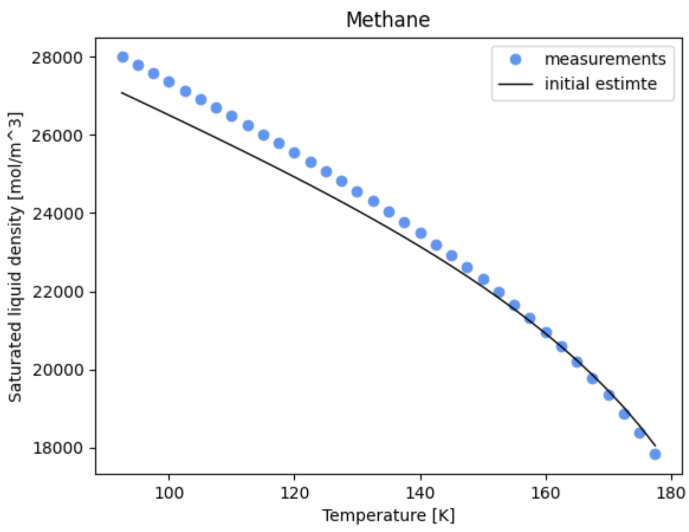

Figure 18: Saturated liquid density depending on temperature for methane: measured data and PC-SAFT calculations with initially estimated parameters [0.9, 3.9, 160]

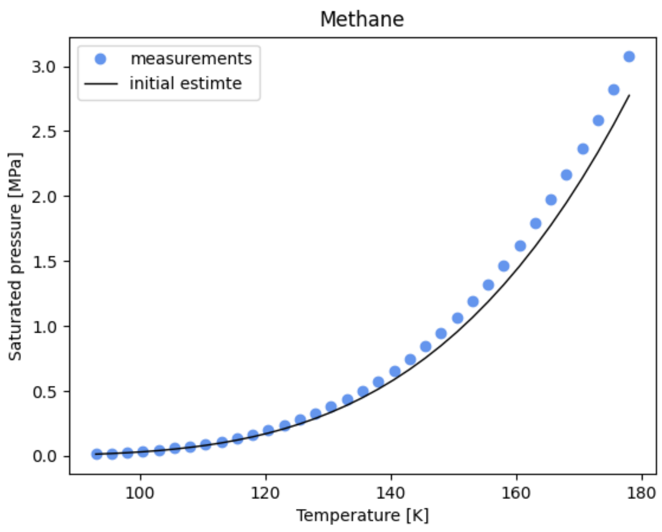

Figure 19: Saturated pressure depending on temperature for methane: measured data and PC-SAFT calculations with initially estimated parameters [0.9, 3.9, 160]

After running the Levenberg-Marquardt algorithm, which had 64 iterations and lasted approximately 9 minutes, these parameters were reached:[1.0026, 3.7023, 149.8159]. In Figures 20 and 21, the measured data and PC-SAFT curves with final parameters are shown.

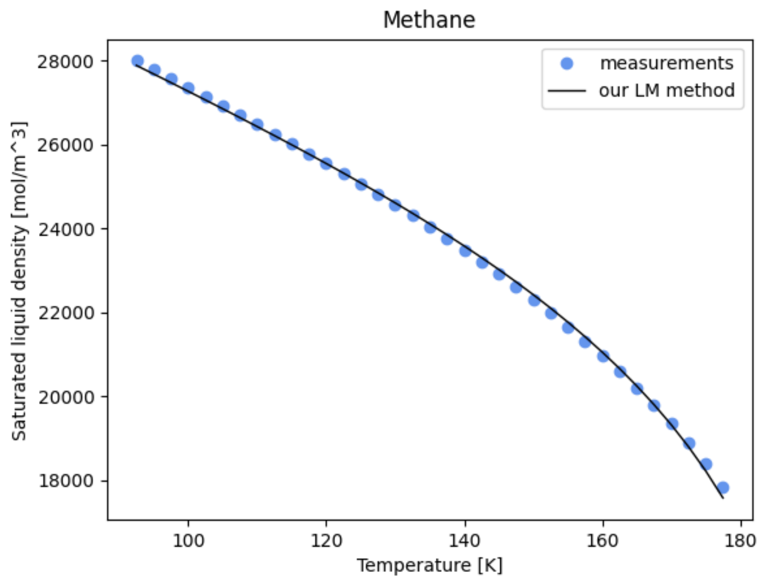

Figure 20: Saturated liquid density depending on temperature for methane: measured data and PC-SAFT calculations after our LM algorithm with final parameters [1.0026, 3.7023, 149.8159]

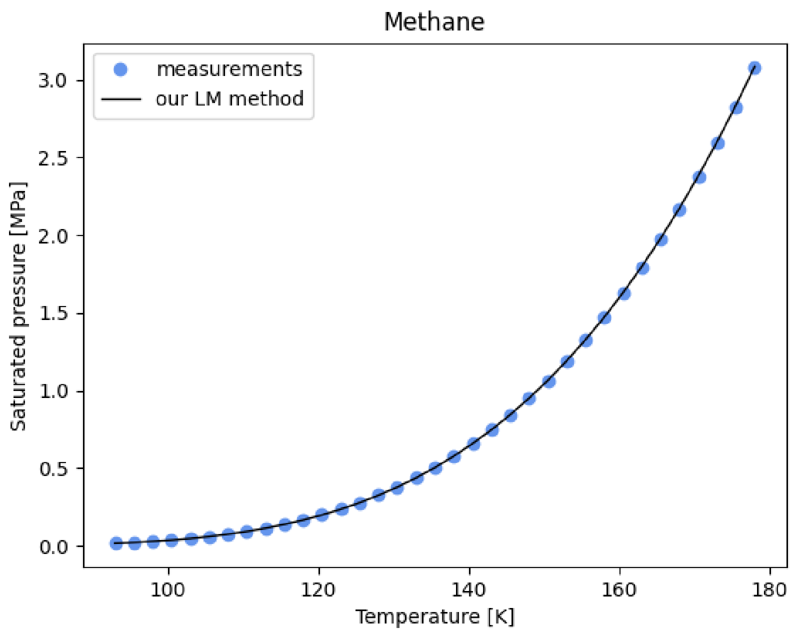

Figure 21: Saturated pressure depending on temperature for methane: measured data and PC-SAFT calculations after our LM algorithm with final parameters [1.0026, 3.7023, 149.8159]

The Figure 22 shows the course of the sums of errors, for each step of our Levenberg-Marquardt algorithm. It can be seen how this sum decreases during 64 iterations and how its values oscillate around zero from positive to negative values.

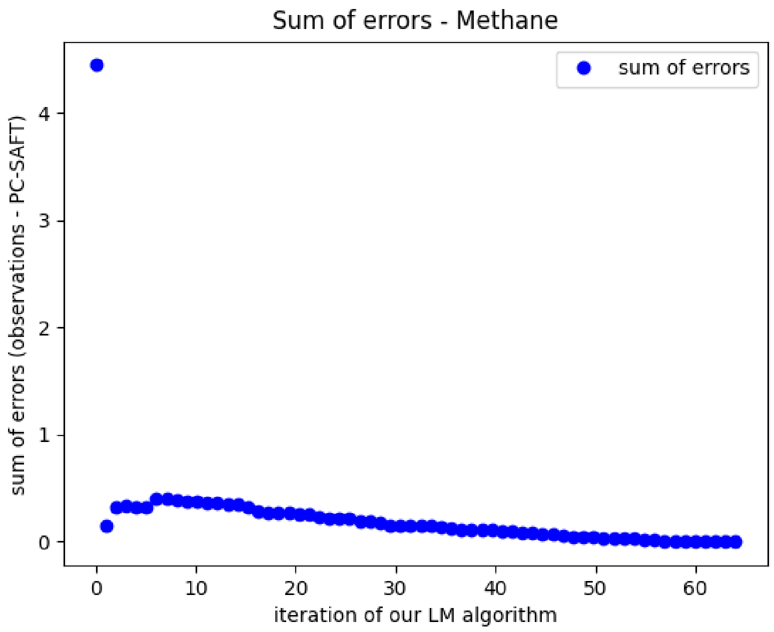

Figure 22: Sum of errors for each step of our LM algorithm for Methane

One of the worst estimates that can still be entered without PC-SAFT EoS ending up in non-physical states is e.g. [0.87, 4.3, 156.0]. Even for such a "bad" estimate, the Levenberg-Marquardt algorithm converges to almost the same final parameters [1.0026, 3.7022, 149.8112]. However, if the estimate is significantly worsened, PC-SAFT EoS can no longer calculate it correctly for some temperatures, so it is always necessary to have at least a general knowledge of the initial parameters. The problem is that when working with a set of some pre-prepared data, one can easily slip to artificially refining and adjusting the initial estimate to finally get where it is needed. Such a tendency can be felt especially by a person who has not yet encountered the practice. However, in practise, the initial estimate can never be so accurate, and, for example, in this case, the parameters [0.9, 3.9, 160] are a kind of maximum that can be reached in determining the initial estimate. The problem, therefore, needs to be sought elsewhere than in the initial estimate, which in practice cannot be improved indefinitely.

When comparing fits, it is necessary to have some uniform way to evaluate how much the resulting function differs from the measured data. One of the good and non-misleading options is to calculate the mean square error. This is calculated according to the formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)^2, \tag{45}$$

where n is the total number of terms for which the error is to be calculated, y_i is the observed value of the variable and $\bar{y}_i$ is the predicted value of variable. For the resulting parameters [1.0026, 3.7023, 149.8159], the mean square error for the saturated liquid density is $1.3088x10^{-5}(\mathrm{mol/m^3})^2$ and for the saturated vapor pressure is 8159.7606 MPa^2. Another thing that can be examined is the absolute deviation plot, which shows how much the resulting function deviates from the measured data for individual temperatures. This will be shown below, as part of a comparison of different fitting algorithms.

The Scipy library basically has three predefined algorithms that can be used for the non-linear least squares problem: the Levenberg-Marquardt algorithm, trust region method and dogbox method. These methods, described in detail in section 3, are chosen in a single function *scipy.optimize.least_squares*(), which requires several mandatory and optional parameters, some of which are listed below:

- *fun*, which is a one-dimensional field whose items correspond to the differences of the function with the current parameters with the measured data.In this program, the output of the *pc_saft_error* function corresponds to this.
- *X*0, which is an initial estimate of PC-SAFT parameters
- *jac*, which is a two, or three-point schema of partial derivatives. It is an optional argument.
- *method*, which is a method that is used:
  - *lm*: Levenberg-Marquardt algorithm as implemented in MINPACK. It does not handle bounds and sparse Jacobians. It is usually the most efficient method for small unconstrained problems.
  - *trf*: Trust Region Reflective algorithm, suitable for large-space problems with bounds. It is not suitable for this problem.
  - *dogbox*: dogleg algorithm with rectangular trust regions, typical of small problems with bounds. It is not recomended for problems with rank-deficient Jacobians.
- *bounds*, which is method to fix the lower and upper bounds of calculated parameters. It is used for 'trf' and 'dogbox' method
- *tr_solver*, which is method for solving trust-region subproblems, relevant only for 'trf' and 'dogbox' methods. In this program 'lsmr' is used, suitable for problems with sparse and large Jacobian matrices
- *verbose*, which can decide what progress optimization information to send to the user. The number of iterations was read here. Unfortunately this is not supported by the 'lm' function

These are the most imporatnt parameters of *least_squares* function. There are several additional parameters, but they weren't use in this project and their description in detail can be found i official *SciPy* documentation.

Let us assume the same input parameters for methane as at the beginning, that is $[0.9, 3.9, 160]$. The final parameters after using the 'lm' method are $[0.9194, 3.8385, 156.3382]$. In this case, the exact result was not obtained because the 'lm' method used in the *least_squares*() function only supports two-point numerical differentiation, (see the third point in the *least_squares*()

parameter list). A possible modification is the insertion of a handwritten three-point numerical derivative, as the 'jac' argument, according to the consensus of the definition of this function.

After testing and comparison of the library and our Levenberg-Marquardt algorithm, it is interesting to compare together other easily testable methods that the scipy library offers. The Dogbox and trust region reflective methods are based on a principle described in the section 3.4 .

For both, it is equally possible to use a three-point numerical differentiation scheme. After calling them for initial estimate of [0.9, 3.9, 160], the following parameters were obtained: [1.0027, 3.7021, 149.8064] for 'trf' method and [1.0022, 3.7027, 149.8405] for 'dogbox' method. The mean square error for the 'trf' method is $1.4534^{-5}(\text{mol/m}^3)^2$ for the saturated liquid density and 8171.7979 MPa^2 for the saturated vapor pressure. For the 'dogbox' method, there is $1.5885^{-5}(\text{mol/m}^3)^2$ for the saturated liquid density and 8194.4644 MPa^2 for the saturated vapor pressure.

The figures 23 and 24 show a graphical comparison of our Levenberg-Marquardt algorithm, the 'trf' and 'dogbox' methods with the same initial parameters [0.9, 3.9, 160].

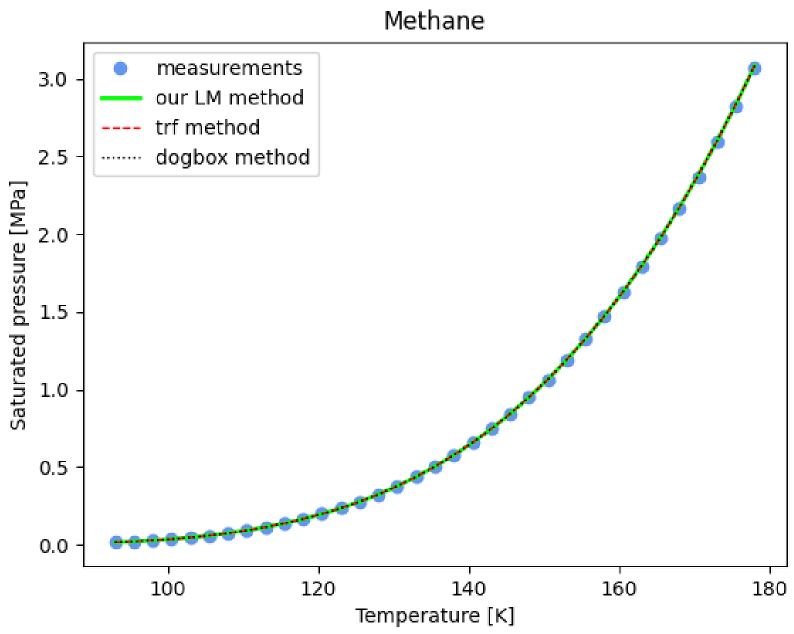

Figure 23: Saturated pressure curve of methane. Comparison of different fitting algorithms for initial estimate [0.9, 3.9, 160]

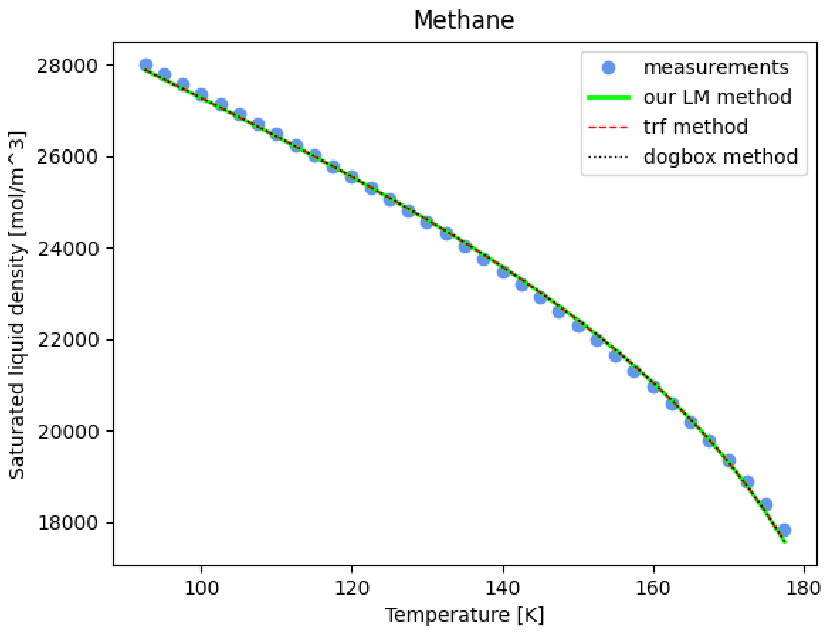

Figure 24: Saturated liquid density curve of methane. Comparison of different fitting algorithms for initial estimate [0.9, 3.9, 160]

As already mentioned, the deviation plots are also an important observation. This can be seen in Figures 25 and 26 for the methods described above.

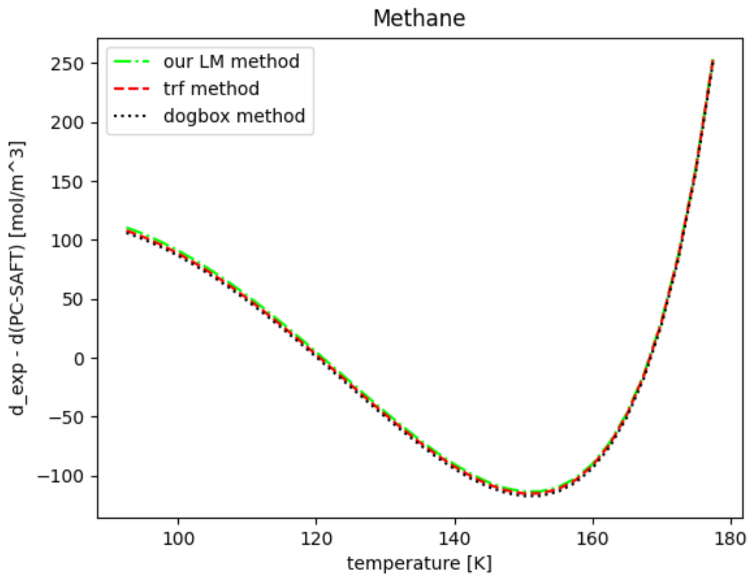

Figure 25: Absolute deviation plot for saturated liquid density. Comparison of different fitting algorithms

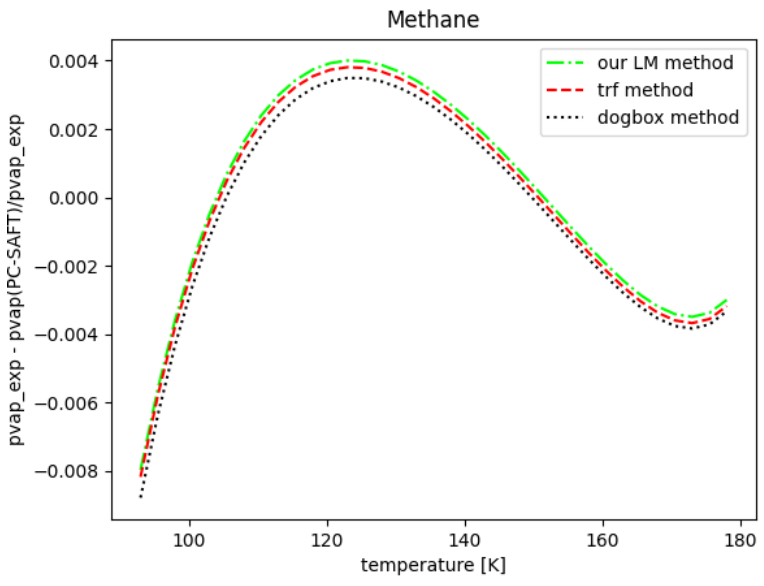

Figure 26: Relative deviation plot for saturated pressure. Comparison of different fitting algorithms

Since it might be difficult to observe the differences between individual curves in the graphs, previous observations are summarized in the table below:

Initial estimate	Method	Obtained parameters $m, \sigma, \epsilon/k$	Mean squared error pvap	Mean squared error dliq
[0.9, 3.9, 160]	Our LM	[1.0026, 3.7023, 149.8159]	$1.3088^{-5}(\text{mol/m}^3)^2$	8159.7606 MPa^2
[0.9, 3.9, 160]	Trf	[1.0027, 3.7021, 149.8064]	$1.4534^{-5}(\text{mol/m}^3)^2$	8171.7979 MPa^2
[0.9, 3.9, 160]	Dogbox	[1.0022, 3.7027, 149.8405]	$1.5885^{-5}(\text{mol/m}^3)^2$	8194.4644 MPa^2

It can be seen from the table that all tested method provides the results with very similar accuracy. The results with the slightest error are given by our Levenberg-Marquardt algorithm. Although the accuracy is not significantly higher than the accuracy of the other algorithms, the biggest advantage of the self-implemented function undoubtedly remains its comprehensibility and easy modifiability.

5.3 Substances used in industry investigated by PC-SAFT EoS

Methane is a simple and already very well researched substance that is very well suited for testing. Substances, whose parameters were optimized below, are more complex and widely used in practice, for example in cooling systems. Predicting their behavior in different conditions is very important. As an example of what can be investigated, the problem of gas solubility in liquids can be mentioned here. Each gas is soluble in liquid, either in great or small. Then serious damage to the cooling circuit can occur. Finding the equilibrium phase of the substances used in these circuits can be very helpful in minimizing the resulting degradation.

The first substance that was studied is "pentafluoroethane". This fluorocarbon is now used as a refrigerant and as a fire suppression agent in fire suppression systems. It has a zero ozone depletion potential, but on the other hand it has a high global warming potential. It is used as a common substitute for CFCs.

Pentafluoroethane is a substance with polarity, i.e. it has a non-zero dipole. The initial estimate for this substance is [3.1, 3.2, 150.0, 0.0, 0.0, 1.563, 1.0, 0.0, 0.0, 0.0, 0.0]. The dipole 1.563 and this parameter does not enter the optimization.

After using our LM algorithm, the following resulting parameters were obtained: [3.12214636, 3.11571653, 153.37571606]. The mean square error here is $7.930038^{-6}(\text{mol/m}^3)^2$ for the saturated pressure and 237.78636 MPa^2 for the saturated liquid density. If the trust region method was used, the resulting parameters were: [3.1887247, 3.09376850, 154.702824], with the mean squared error of $0.0084048(\text{mol/m}^3)^2$ for the saturated pressure and 23964.5531 MPa^2 for the saturated liquid density. The dogbox method gave very similar results, where the resulting parameters were: [3.18960408, 3.09342953, 154.680499] and the mean squared error was $0.008386(\text{mol/m}^3)^2$ for the saturated pressure and 24004.28783 MPa^2 for the saturated liquid density.

The table and the resulting graphs are included for greater clarity.

Initial estimate	Method	Obtained parameters $m, \sigma, \epsilon/k$	Mean squared error pvap	Mean squared error dliq
[3.1, 3.2, 150.0]	Our LM	[3.12214636, 3.11571653, 153.37571606]	$7.930038^{-6}(\text{mol/m}^3)^2$	237.78636 MPa^2
[3.1, 3.2, 150.0]	Trf	[3.1887247, 3.09376850, 154.702824]	$0.0084048(\text{mol/m}^3)^2$	23964.5531 MPa^2
[3.1, 3.2, 150.0]	Dogbox	[3.18960408, 3.09342953, 154.680499]	$0.008386(\text{mol/m}^3)^2$	24004.28783 MPa^2

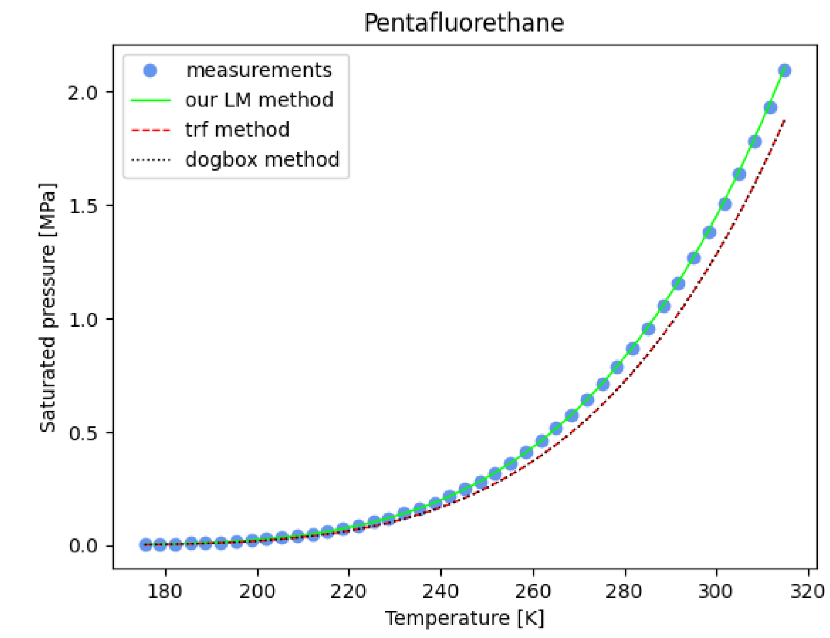

Figure 27: Saturated pressure curve for pentafluorethane. Comparison of different fitting algorithms for initial estimate of [3.1, 3.2, 150.0, 1.563]

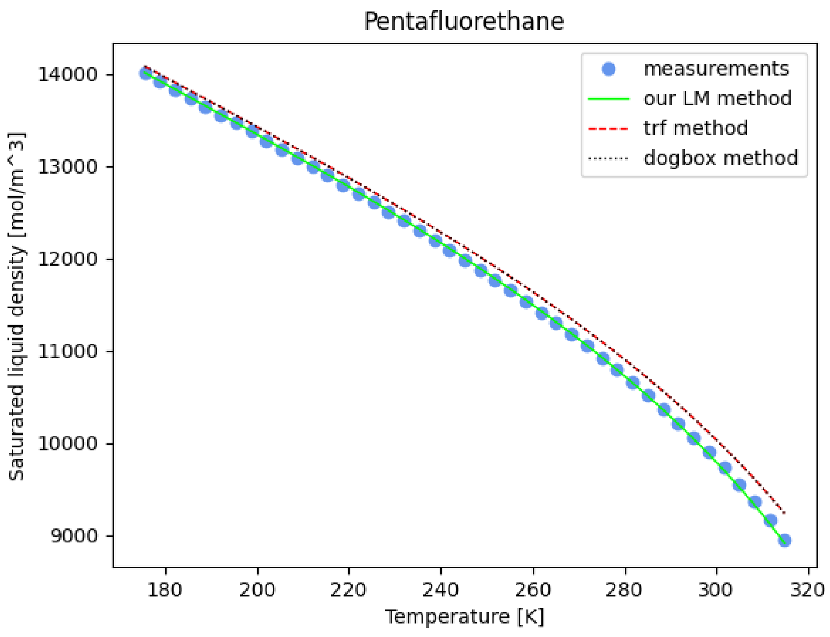

Figure 28: Saturated liquid density curve for pentafluorethane. Comparison of different fitting algorithms for initial estimate of [3.1, 3.2, 150.0, 1.563]

The third substance that has been studied is pentane. Pentane is an organic compound that is

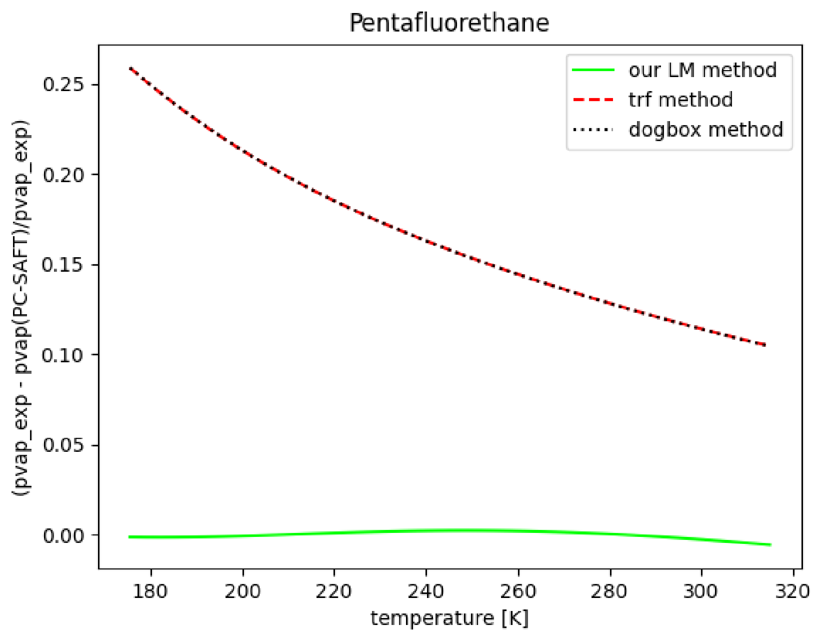

Figure 29: Relative deviation plot of pentafluorethane for saturated pressure. Comparison off different fitting algorithms.

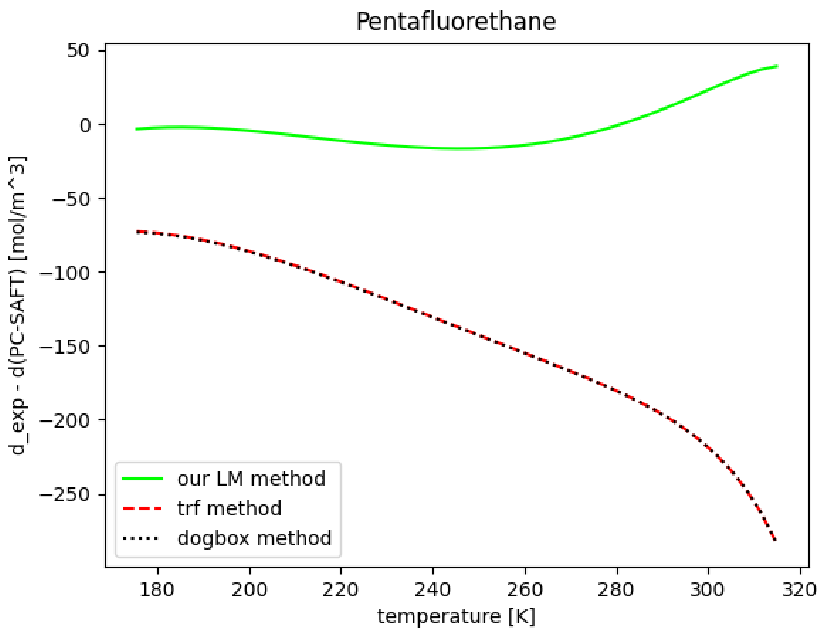

Figure 30: Absolute deviation plot of pentafluorethane for saturated liquid density. Comparison off different fitting algorithms.

used as part of some fuels and as a special solvent in laboratories. Because it is non-polar, it is very difficult to dissolve and it has readily soluble alkyl-rich compounds. It is also used as a blowing agent in the production of polystyrene foam, or as a working medium in geothermal power plants.

The initial estimate for pentane is [2.7, 3.8, 230.0]. After running our Levenberg-Marquardt algorithm, the following parameters were obtained: [2.70677932, 3.75206582, 230.50177492]. The mean square error here is $1.5402^{-5}(\text{mol/m}^3)^2$ for the saturated pressure and 907.0051 MPa^2 for the saturated liquid density. If the trust region method was used, the resulting parameters were: [2.694448, 3.758595, 231.08131], with the mean squared error of $3.9478^{-6}(\text{mol/m}^3)^2$ for the saturated pressure and 1013.1883 MPa^2 for the saturated liquid density. The resulting parameters from dogbox method were [2.69971, 3.75707, 230.784] and the mean squared error was $9.3072^{-6}(\text{mol/m}^3)^2$ for the saturated pressure and 1405.2818 MPa^2 for the saturated liquid density.

The table and the resulting graphs are included below.

Initial estimate	Method	Obtained parameters $m, \sigma, \epsilon/k$	Mean squared error pvap	Mean squared error dliq
[2.7, 3.8, 230.0]	Our LM	[2.70677932, 3.75206582, 230.50177492]	$1.5402^{-5}(\text{mol/m}^3)^2$	907.0051 MPa^2
[2.7, 3.8, 230.0]	Trf	[2.694448, 3.758595, 231.08131]	$3.9478^{-6}(\text{mol/m}^3)^2$	1013.1883 MPa^2
[2.7, 3.8, 230.0]	Dogbox	[2.69971, 3.75707, 230.784]	$9.3072^{-6}(\text{mol/m}^3)^2$	1405.2818 MPa^2

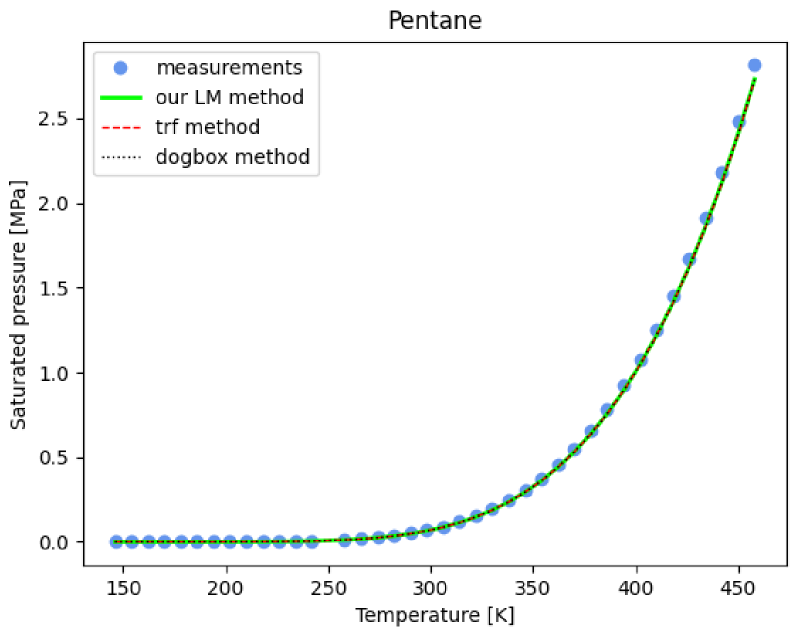

Figure 31: Saturated pressure curve for pentane. Comparison of different fitting algorithms for initial estimate of [2.7, 3.8, 230.0]

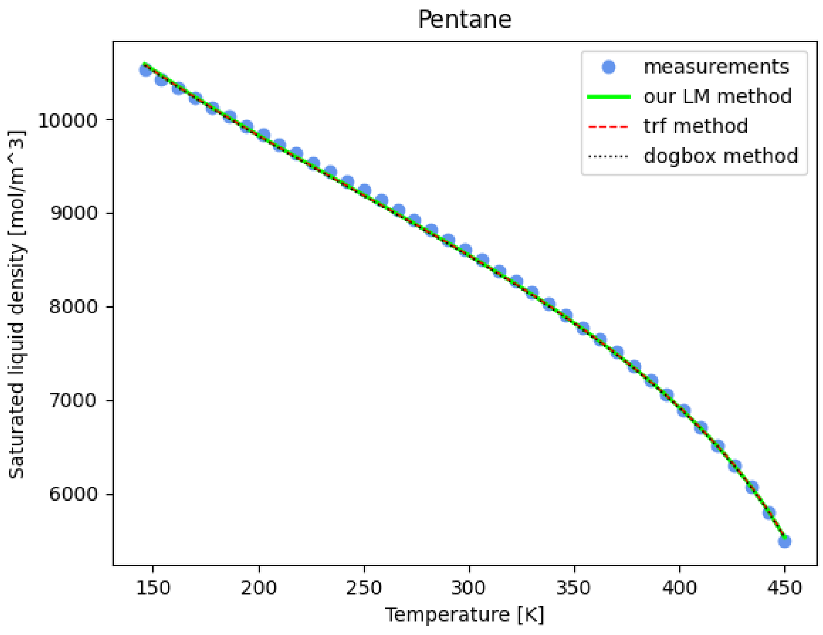

Figure 32: Saturated liquid density curve for pentane. Comparison of different fitting algorithms for initial estimate of [2.7, 3.8, 230.0]

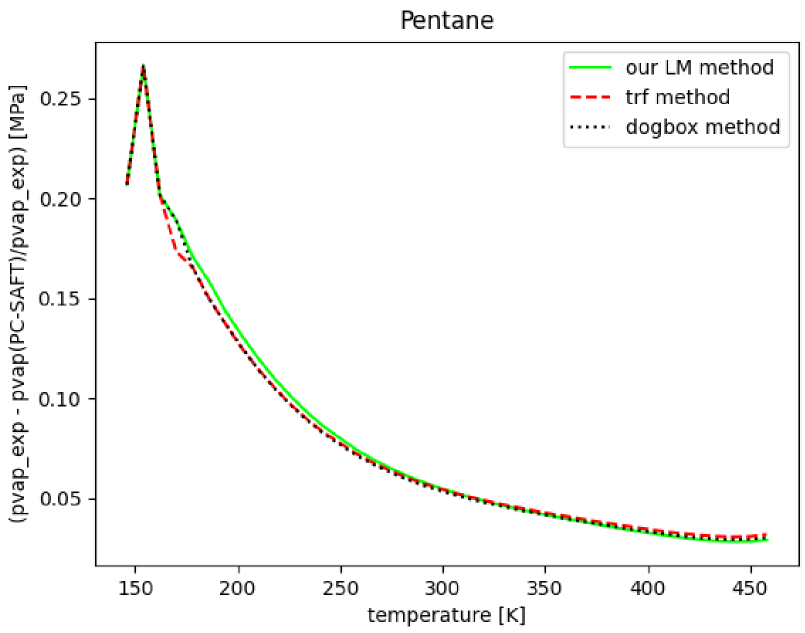

Figure 33: Relative deviation plot of pentane for saturated pressure. Comparison of different fitting algorithms.

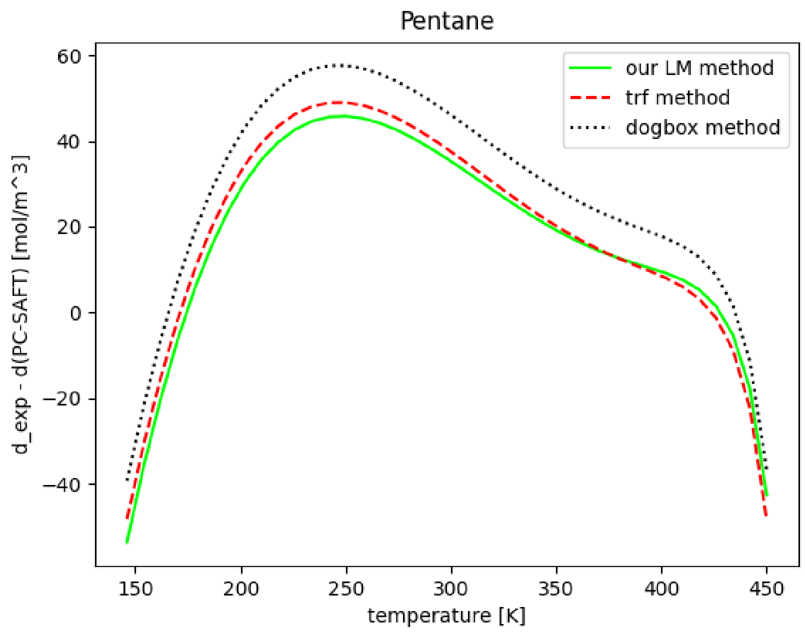

Figure 34: Absolute deviation plot of pentane for saturated liquid density. Comparison of different fitting algorithms.

6 Summary and conclusions

In this project, the Levenberg-Marquardt algorithm implemented by us was first tested for a quadratic function. Here it was compared with the library Levenberg Marquardt method from

$SciPy$. The sensitivity of the method to the applied numerical differentiation was shown; it is advisable to use the central, not the forward scheme.

After testing the Levenberg-Marquardt method on this simple example, the algorithm was tested and compared with the others on three substances: methane, pentane and pentafluoroethane. Methane served more as a test substance, while pentane and pentafluoroethane are already widely used in practice. On these substances, the Levenberg-Marquardt algorithm implemented by us was compared with the trust region field ('trf') and the dogbox methods from the $SciPy$ library. The 'lm' method from the $SciPy$ library could be used provided that three-point numerical diferentiation was inserted into it, which, however, would have to be written manually by the programmer. In this case, if the programmer already wants to embark on handwritten implementations, it may be more advantageous to write the whole by hand. Then, it is not just a "blackbox", but a program with great possibilities of partial individual adjustments, understandable to the person who wrote it.

However, both, the 'trf' and the dogbox method proved to be applicable to simple substances. By the word "simple substance" is meant here, for example, the tested methane or pentane, which have a zero dipole. For these substances, all tested methods work with similar accuracy. The Levenberg-Marquardt algorithm implemented by us showed slightly more accurate results in all cases, except for vapor pressure for pentane. There, the library methods, on the other hand, were slightly closer to the measured data. Significant differences between the tested functions occurred only in the investigated more complex pentafluoroethane, where the 'trf' and 'dogbox' methods proved to be undoubtedly less accurate (see Fig. 27 and 28).

As part of further research, it would be interesting to write an implementation of a simplex method that solves various problems of "classical" optimization, as described in the 3.5 section and which we also encountered in this work.

List of Figures

References

[1] Dina Zhabinskaya *Three-Phase Model of Pure Sunstances*, UC Davis, departement of physics and astronomy, 2020

[2] K.V.Narayanan *A Textbook of Chemical Engineering Thermodynamics, 2th edition*, Delhi-110092, ISBN-978-81-203-4747-2, 2013

[3] Gross J, Sadowski G,*Perturbed-chain SAFT: An equation of state based on a perturbation theory for chain molecules. Industrial and engineering chemistry research*, Ind. Eng. Chem. Res. 2001, 40, 1244-1260, 2001.

[4] V.Vinš, B.Planková, J.Hrubý, D.Celný, *Density gradient theory combined with the PC-SAFT equation of state for modeling the surface of associating systems*, EPJ Web of Conferences 67, 02129, DOI: 10.1051/epjconf/20146702129, 2014

[5] Marc D.Donohue, P.Vimalchand, *The perturbed-hard-chain theory. Extensions and applications*, Department of Chemical Engineering, The Johns Hopkins University, Baltimore, Maryland 21218 USA, Fluid Phase Equilibria journal, 1987

[6] Zucker, Robert D., Biblarz, Oscar, *Fundamentals of Gas Dynamics (2nd ed.).* Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, California, Wiley Books, 2002

[7] C.T. Kelley *Iterative Methods for Optimization. SIAM Frontiers in Applied Mathematics*, North Carolina State University, Raleigh, North Carolina, Society for Industrial Applied Mathematics, Philadelphia, 1999.

[8] K. Madsen, H.B. Nielsen, O. Tingleff, *Methods for non-linear least squares problems, 2th Edition*, Informatics and Mathematical Modelling, Technical University of Denmark, 2004

[9] A. Björck, A. *Numerical methods for least squares problems*, SIAM, Philadelphia, 1996

[10] Nadim Khalil *ULSI Characterization with Technology Computer-Aided Design*, Dissertation, 55 Brookdale Circle Shrewsbury, MA 01545, USA, Matr.-Nr. 9127879, geboren am 18. Dezember 1954 in Zouk Mikael, Libanon, Wien, im Mai, 1995

[11] K. M. Brown, J.E. Dennis Jr., *Derivative free analogues of the Levenberg-Marquardt and Gauss algorithms for nonlinear least squares approximation* Numerical Methods, 18:289-297, 1972

[12] P.E. Gill, W. Murray, M. H. Wright., *Practical Optimization*, Systems Optimization Laboratory, Department of Operations Research, Stanford University, California, USA, 1981

[13] Kenneth Levenberg *A method for the solution of certain non-linear problems in least squares*, Quart. Appl. Math. 2, 164 - 168, 1944

[14] V.Vinš, J.Hrubý, *Solubility of nitrogen in one-component refrigerants, Prediction by PC-SAFT EoS and correlatin of Henry's law constant*, International journal of refrigeration 131, 956-969, 2021

[15] C.Voglis, I.E.Lagaris *A Rectangular Trust Region Dogleg Approach for Unconstrained and Bound Constrained Nonlinear Optimization*, Department of Computer science, university of Ioannina, Greece, 2000

[16] Marco S. Caceci, William P. Cacheris, *Fitting curves to data - the simplex algorithm is the answer*, Florida State University, popular science magasine Byte, 1984