Brandenburg University of Technology

Master Thesis

Numerical Simulations of Multicomponent Mixture Vaporization

Author:
Hannes Seidel

Supervisor:
Prof. Fabian Mauss, Corinna Netzer, Lars Seidel

A thesis submitted in Cottbus of the requirements for the degree of Master of Science

in the

Faculty 3 Mechanical Engineering, Electrical and Energy Systems
Chair of Thermodynamics of Reactive Systems

20. September 2016

Declaration of Authorship

I, Hannes Seidel, declare that this thesis titled, 'Numerical Simulations of Multicomponent Mixture Vaporization' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

BRANDENBURG UNIVERSITY OF TECHNOLOGY

Abstract

Faculty 3 Mechanical Engineering, Electrical and Energy Systems
Chair of Thermodynamics of Reactive Systems

Master of Science

Numerical Simulations of Multicomponent Mixture Vaporization

by Hannes SEIDEL

Conventional internal combustion engines are powered with diesel or gasoline fuels, distilled from crude oil. This liquids are complex mixtures of different types of poison, cancerous and lung damaging hydrocarbon species. The combustion of this hydrocarbons with air can produce additional hazardous species. The prediction of the vapor liquid equilibrium for this fuels is indispensable for the secure handling and combustion. Because of the complexity, the attempting of numerical simulations of real fuels is highly unrealistic. The use of surrogate fuels whose overall behavior mimics the characteristics of the target fuels, is the compromise until now. The defined mixtures offers the possibility for various numerical simulations.
In this work a numerical tool for the prediction of distillation curves for multicomponent mixtures with any number of substances is developed. This tool is based on a real gas equation of state and mixing rules for real liquid mixtures. The prediction of distillation curves and the comparison with samples has been realized with a vaporisation model for multicomponent mixtures. The use of two methods for the calculation of the saturation pressures and the applying of the real phase characteristics offers results to contrast. The real liquid phase has been modeled with activity coefficients from UNIFAC. The real gas phase takes respect with the cubic Soave Redlich Kwong equation of state. The blending of mixtures until the preferred distillation curve results, has been established by solving a multidimensional optimisation problem.
For validation the tool has been applied to, in literature well known, fuels for advance combustion engines (FACE) experiments with good agreement.

Acknowledgements

First and foremost, thank you Prof. Fabian Mauss for given me the chance to carry out my thesis on this interesting topic. I offer my sincerest gratitude to my supervisors, Corinna Netzer and Lars Seidel, who have supported me throughout my thesis with patience and knowledge whilst allowing me the room to work in my own way. I attribute the level of my Masters degree to their encouragement and effort and without them this thesis, would not have been completed or written.

Contents

Symbols

m	Mass	kg
p	Pressure	Pa (N/m^2)
T	Temperature	K
V	Volume	m^3
w	Mass fraction	kg / kg
M	Molare mass	g / mol
v	Molare volume	m^3 / mol
x	Molare liquid fraction	mol / mol
y	Molare gas Fraction	mol / mol
γ	Activity coefficient	
φ	Fugacity coefficient	
ω	Acentric factor	

Chapter 1

Introduction

Conventional internal combustion engines are powered with diesel or gasoline fuels, distilled from crude oil. This liquids are complex mixtures of different types of poison, cancerous and lung damaging hydrocarbon species. The combustion of this hydrocarbons with air can produce additional hazardous species, like ozone and nitrogen oxides. The prediction of the vapor liquid equilibrium of this fuels is indispensable for the secure handling, the transportation and of course, for the combustion. Because of the complexity, the attempting of numerical simulations of real fuels is highly unrealistic. The compromise for this problem is the use of surrogate fuels whose overall behavior mimics the characteristics of the target fuels. This mixtures contains clear numbers of substances of two or ten, instead of two- or ten-thousand as the real fuel. The defined mixtures offers the possibility for numerical simulations of the combustion process, which may delivers directives to enhance the performance and to reduce the emissions of engines. The used surrogate fuels referencing to the fuels for advanced combustion engines (FACE).
There are various methods to describe diesel or gasoline fuels. Popular numbers are the octane number or the e10 prefix which pointing to the content of octane or ethanol. This numbers are mostly true for the amount of ethanol, but the octane number results from experiments and can not simply adapted for the description of surrogates. Furthermore is the h / c ratio of interest for the hydrogen - carbon distribution, which is to universal for numerical applications. One incorruptible characteristic is the distillation curve of fuels. The substances have different boiling temperatures resulting in a curve from the initial to the final boiling point. The last point can be good predicted from the boiling point of the highboiler. The first and all points between, are not trivial predictable. The modeling of this curve has to result in a method where only the substances, the fractions and the pressure are the calculation base. This again can serve for the answering of a composition problem, to predict the needed fractions based on a given distillation curve.

Calculations on binary mixtures are frequently used and comparatively easy to implement because the molar fractions are dependent, so one amount supplies the other. Multicomponent mixtures requires more sophisticated algorithms for equilibrium calculations. The level of complexity rises again respects to the real behavior of the phases. The boiling temperature of a substance can be calculated using Antoine constants or cubic equations of state (eos). While the Antoine equation can be solved by hand, a numeric framework is helpful for cubic equations. All algorithms are written in R script interpreter language. The real phases are taken into account by using UNIFAC for the liquid and cubic eos for the gas phase. The vaporization model based on open system conditions and the bubble point temperature of multicomponent mixtures. This temperature can be calculated, using five models with and without real phases. The accuracy of the estimated saturation pressures are compared for the Antoine and cubic Soave Redlich Kwong equation. crude oil

1.1 Surrogate Fuels

Fuel surrogates are mixtures of one or more simple fuels that are designed to emulate either the physical properties or combustion properties of a more complex fuel. Gasoline is a widely used fuel, but its molecular complexity makes it unattractive to experimentally and computationally study. Therefore, surrogate fuels with a manageable molecular composition that represent real fuel behavior in one or more aspects are needed to enable repeatable experimental and computational combustion investigations. The most widely used surrogates for gasoline fuels are binary mixtures of n-heptane and iso-octane termed primary reference fuel (PRF). Ternary mixtures of PRFs plus toluene (i.e., TRF) have also been proposed as surrogates for gasoline fuels with satisfactory reproduction of important target properties [2]. The fuels for advanced combustion engines (FACE) have been developed in a joint project between the U.S. Department of Energy and the Coordinating Research Council (CRC), which is a research body composed of automobile, engine, and energy companies for studying advanced combustion engine concepts, among many problems of relevance. The mission of the FACE working group is to recommend a set of test fuels that are well-suited for research so that the researchers evaluating advanced combustion systems may compare results from different laboratories using the same set of fuels for consistency [1].

Chapter 2

Methods

2.1 Thermodynamic Equations of State

A system in equilibrium can be described by measurable values and properties like pressure, temperature, volume, mass and composition. For the description of states are the energy, enthalpy and entropy. Because of their functional context are this units called state variables. The functions which present the relationship between those variables are called equations of state (eos).

2.1.1 Critical and Reduced Values

The critical point marks the state above whom no pure liquid exists. All upcoming vapor liquid equilibrium (vle) calculations are located between the triple and the critical point. The pressure (p_c) and temperature (T_c) at this point are readily available properties of substances. With this values, the reduced can be calculated:

$$p_r = \frac{p}{p_c} \tag{2.1}$$

$$T_r = \frac{T}{T_c} \tag{2.2}$$

$$v_r = \frac{v\,p_c}{R\,T_c} \tag{2.3}$$

2.1.2 Acentric Factor

The acentric factor ω is a conceptual number introduced by Pitzer in 1955, it is a measure of the non-sphericity of molecules and defined at the reduced temperature of 0.7.

$$\omega = -\log_{10}(p_r^S) - 1, \text{ at } T_r = 0.7. \tag{2.4}$$

For many mono atomic fluids p_r^S at $T_r = 0.7$ is close to 0.1, therefore $\omega \to 0$ [3].

2.2 Ideal Fluids and Mixtures

2.2.1 Ideal Gas Law

The model of ideal gases by Émile Clapeyron is a straightforward eos, resulting from a combination of the empirical Boyle's law, Charles' law and Avogadro's law [4]. It describes the limiting case of all thermodynamic eos for a low density or rather small pressure at high temperature. In this case the volume and the cohesion of the gas molecules can be neglected. For some gases it provides a good approach and it forms a starting point for all upcoming derivations.

$$p = \frac{RT}{v} \tag{2.5}$$

$$v = \frac{V}{n} \tag{2.6}$$

2.2.2 Dalton's Law

Dalton's law stating that the pressure exerted by a mixture of gases in a fixed volume is equal to the sum of the pressures that would be exerted by each gas alone in the same volume.

$$p_i = y_i\, p \tag{2.7}$$

2.2.3 Raoult's Law

Raoult's law stating that the vapor pressure of an ideal solution is proportional to the product of the mole fraction of the solvent and their saturation pressure in pure state.

$$p_i = x_i\, p_i^S \tag{2.8}$$

2.3 Real Gases

2.3.1 Compressibility Factor

For the describing of properties of real gases some auxiliary quantities are common for represent the real behavior. The compressibility factor z is defined as follows:

$$z = \frac{pv}{RT} = \left(\frac{v}{v^{id}}\right)_{T,p} \tag{2.9}$$

For ideal gases $z^{id} = 1$ and for all further fluids it results in the limit case:

$$\lim_{p \to 0} z = 1$$

The difference of a thermodynamic attribute between a real fluid and an ideal gas is a criterion for the molecular interaction. It is called departure function and defined as the difference of an attribute of a real fluid at a temperature and pressure according to them of an ideal gas at that temperature and pressure.

$$f_{departure} = \left(m - m^{id}\right)_{T,p} \tag{2.10}$$

The departure function of the molar volume can express by using equation 2.9 as follows:

$$(v - v^{id}) = \frac{RT}{p}(z - 1) \tag{2.11}$$

Furthermore are the departure functions of the entropy, enthalpy and the Gibbs enthalpy points of interest.

departure f	$f\,(p)$	$f\,(v)$
$(s - s^{id})_{T,p}$	$\int_0^p \left[-\left(\frac{\delta v}{\delta T}\right)_p + \frac{R}{p}\right] dp$	$\int_\infty^g \left[\left(\frac{\delta p}{\delta T}\right)_v - \frac{R}{v}\right] dv + R \ln z$
$(h - h^{id})_{T,p}$	$\int_0^p \left[v - T\left(\frac{\delta v}{\delta T}\right)_p\right] dp$	$-\int_\infty^g \left[p - T\left(\frac{\delta p}{\delta T}\right)_v\right] dv + pv - RT$
$(g - g^{id})_{T,p} = RT \ln \varphi$	$\int_0^p \left(v - \frac{RT}{p}\right) dp$	$RT(z - 1 - \ln z) + \int_\infty^g \left(\frac{RT}{v} - p\right) dv$

2.3.2 Virial Equation of State

The virial equation is a full theoretically equation of state and derived from statistical mechanics [5].

$$\frac{pv}{RT} = 1 + \frac{b}{v} + \frac{c}{v^2} + \frac{d}{v^3} + \ldots \tag{2.12}$$

$$b = -v_c, \; c = \frac{v_c^2}{3} \tag{2.13}$$

Theoretical expressions can be developed for each of the coefficients, if appropriate assumptions of intermolecular forces are made. In this case b corresponds to interactions between pairs of molecules, c to triplets, and so on. The accuracy can be increased indefinitely by considering higher order terms.

2.3.3 Cubic Equations of State

Cubic eos are called such because they can be rewritten as a cubic function of the molar volume or the compressibility factor. The coefficients A and B are normalized for cubic eos and include the acentric factor, processed in the α term.

$$A = \frac{\alpha\, a\, p}{(R\,T)^2} \tag{2.14}$$

$$B = \frac{b\, p}{R\,T} \tag{2.15}$$

2.3.3.1 Van der Waals EoS

The Van der Waals eos, published in 1873, describes the properties of fluids over a wide pressure range [6]. It includes values from the critical point (T_c, p_c), to reduce the condensation and the critical phenomenons on molecular conceptions. The compressibility factor or the pressure are parted in a repulsing and an attraction term.

$$z = z^r + z^a \tag{2.16}$$

The repulsing forces are reflected by the volume of the molecules and processed inside the constant b. The a parameter includes the attraction interactions between the molecules [7].

$$z = \frac{v}{v-b} - \frac{a}{R\,T\,v} \tag{2.17}$$

$$p = \frac{R\,T}{v-b} - \frac{a}{v^2} \tag{2.18}$$

$$a = \frac{27(R\,T_c)^2}{64p_c},\ b = \frac{R\,T_c}{8p_c} \tag{2.19}$$

$$A = \frac{a\,p}{(R\,T)^2} \tag{2.20}$$

[6]

Polynomial form:

$$v^3 - \left(b + \frac{R\,T}{p}\right) v^2 + \frac{a}{p} v - \frac{ab}{p} = z^3 - (1 + \frac{b\,p}{R\,T})\, z^2 + \frac{a\,p}{(R\,T)^2}\, z - \frac{a\,b\,p^2}{(R\,T)^3} = 0 \tag{2.21}$$

Simplified with $A\,B$ coefficients:

$$z^3 - (1+B)\, z^2 + A\, z - A\, B = 0 \tag{2.22}$$

2.3.3.2 Redlich Kwong EoS

The Redlich Kwong eos is an improvement of the Van der Waals eos from 1949. The a parameter has obtained a square root of T dependence and the denominator of the second term got a small b dependence. It is quite accurate for predicting molar volumes of pure substances. However, calculations of the properties of mixtures and predictions of vapor-liquid equilibrium using this model are not particularly accurate [8].

$$p = \frac{R\,T}{v-b} - \frac{a}{\sqrt{T}\, v\,(v+b)} \tag{2.23}$$

$$a = \frac{0.42748\, R^2\, {T_c}^{5/2}}{p_c},\ b = \frac{0.08664\, R\,T_c}{p_c} \tag{2.24}$$

$$A = \frac{a\,p}{\sqrt{T}\ (R\,T)^2} \tag{2.25}$$

Polynomial form:

$$v^3 - \frac{R\,T}{p} v^2 + \left(\frac{a}{p\sqrt{T}} - \frac{b\,R\,T}{p} - b^2\right) v - \frac{a\,b}{p\,\sqrt{T}} = z^3 - z^2 + (A - B - B^2)\, z - AB = 0 \tag{2.26}$$

2.3.3.3 Soave Redlich Kwong EoS

In 1972, Soave replaced the square root of T term with a function $\alpha(T,\omega)$ involving the temperature and the acentric factor ω. Essentially, the parameter a is given a more complicated temperature dependence than that assumed in the Redlich-Kwong equation [9]. This equation is still frequently used for predicting the properties of pure substances, mixtures and vapor-liquid equilibrium. It is not expected to be accurate for highly polar species or molecules that exhibit hydrogen bonding [10].

$$p = \frac{RT}{v-b} - \frac{a\,\alpha}{v\,(v+b)} \tag{2.27}$$

$$a = \frac{0.427\,R^2\,T_c^2}{p_c},\ b = \frac{0.08664\,R\,T_c}{p_c} \tag{2.28}$$

$$\alpha = \left(1 + \left(0.48508 + 1.55171\,\omega - 0.15613\,\omega^2\right)\left(1-\sqrt{T_r}\right)\right)^2 \tag{2.29}$$

Polynomial form:

$$v^3 - \frac{RT}{p}v^2 - \left(b^2 - \frac{\alpha\,a}{p}\right)v - \frac{\alpha\,a\,b}{p} - \frac{R\,T\,b}{p} = z^3 - z^2 + (A - B - B^2)\,z - AB = 0 \tag{2.30}$$

2.3.3.4 Peng Robinson EoS

The Peng-Robinson equation, published 1976, is particularly accurate for predicting the properties of hydrocarbons, including the behavior of mixtures and vapor-liquid equilibrium. It is fairly similar to the Soave-Redlich-Kwong equation, but with a slightly different denominator for the second term [11]. It is not expected to be accurate when predicting properties of highly polar molecules, particularly those that are capable of hydrogen bonding [10].

$$p = \frac{RT}{v-b} - \frac{a\,\alpha}{v(v+b)+b(v-b)} \tag{2.31}$$

$$a = \frac{0.45724\,R^2\,T_c^2}{p_c}, b = \frac{0.07780\,R\,T_c}{p_c} \tag{2.32}$$

$$\alpha = \left(1 + \left(0.37464 + 1.54226\,\omega - 0.26992\,\omega^2\right)\left(1-\sqrt{T_r}\right)\right)^2 \tag{2.33}$$

Polynomial form:

$$0 = z^3 - (1 - B)\, z^2 + (A - 2B - 3B^2)\, z - (AB - B^2 - B^3) \tag{2.34}$$

2.3.3.5 Peng Robinson Gasem EoS

This modification from 2001 is more accurate for heavy hydrocarbons and works better at temperatures above the critical point than the normal Peng-Robinson equation [12].

$$p = \frac{RT}{v - b} - \frac{a}{v(v + b) + b(v - b)} \tag{2.35}$$

$$a = \frac{0.45724\, R^2\, T_c^2}{p_c} \exp\left((2 + 0.836\, T_r)\, (1 - T_r^{\alpha})\right) \tag{2.36}$$

$$\alpha = 0.134 + 0.508\, \omega - 0.0467\, \omega^2 \tag{2.37}$$

$$b = \frac{0.07780\, R\, T_c}{p_c} \tag{2.38}$$

Polynomial form:

$$0 = z^3 - (1 - B)\, z^2 + (A - 2B - 3B^2)\, z - (AB - B^2 - B^3) \tag{2.39}$$

2.3.3.6 Solving Cubic Equations

Cubic equations can be solved in a general manner, presupposed the cubic polynomial with real coefficients is given as: $z^3 + c_2\, z^2 + c_1\, z + c_0 = 0$. The first step is to calculate the parameters:

$$q = \frac{c_2^2 - 3c_1}{9}, \; r = \frac{2c_2 - 9c_2 c_1 - 27c_2}{54}$$

In the next step the discriminant $m = r^2 - q^3$ is computed to consider the following cases:

1. $m < 0$ ($r^2 < q^3$), the polynomial has three real roots and

   $$\cos(\theta) = \frac{r}{\sqrt{-q^3}}$$

   Than the three roots are:

   $x_1 = 2\sqrt{-q} \cos\left(\frac{\theta}{3}\right) - \frac{c_2}{3}$

   $x_2 = 2\sqrt{-q} \cos\left(\frac{\theta}{3} + 120°\right) - \frac{c_2}{3}$

$x_3 = 2\sqrt{-q}\cos\left(\frac{\theta}{3} + 240°\right) - \frac{c_2}{3}$

2. $m > 0$ ($r^2 > q^3$), the polynomial has only one real root and

$$s = \sqrt[3]{-r + \sqrt{m}},\ t = \sqrt[3]{-r - \sqrt{m}}$$

Than the root is:

$x_1 = s + t - \frac{c_2}{3}$

[13]

2.4 Real Mixtures

2.4.1 Mixing Rules

The mixing rules are empiric for cubic equations of state. In case of mixtures, the parameters a and b must be determined from the data of the pure substances. The interactions between the molecules are reflected by a quadratic concentration dependence [7].

$$a = \sum_i \sum_j x_i\, x_j a_{ij} \tag{2.40}$$

The cross coefficients a_{ij} are the geometric medium of the parameters of pure substances (a_{ii}, a_{jj}), corrected with the binary interaction parameter k_{ij}.

$$a_{ij} = \sqrt{a_{ii}\, ajj}\,(1 - k_{ij}) \tag{2.41}$$

The proper volume of the mixture can be computed by the summation of the product of the parameters from the pure substances and the molar fraction.

$$b = \sum_i x_i\, b_i \tag{2.42}$$

This rules are similar valid for the parameters of the gas phase.

2.4.2 Fugacity and Activity

The relationship of the differential of the Gibbs enthalpy to the pressure explicit equation for ideal gases $dg^{id} = RT \ln p$ ($T = const$), is valid for real fluids by using the auxiliary quantity fugacity (f) to substitute the pressure. The ratio of the fugacity to the pressure is called fugacity coefficient and becomes one for ideal gases respectively at zero pressure, for all fluids.

$$\varphi = \frac{f}{p} \tag{2.43}$$

The equilibrium between the different phases of mixtures comply if the temperature, pressure and chemical potential is equal for every phase. The chemical potential equals the molar Gibbs enthalpy, which can be expressed with the fugacity. It is useful to split the Gibbs enthalpy, or any other property of the mixture in a term for the ideal mixture (g_i^{id}) and an excess term (g_i^e).

$$g_i = g_i^{id} + g_i^e = g_i^{pure}(T, p^0) + RT \ln y_i + RT \ln \frac{f_i}{y_i f_i^0} \tag{2.44}$$

The fraction of the fugacity to the fugacity at standard condition is called activity.

$$a_i = \frac{f_i}{f_i^0} \tag{2.45}$$

$$\gamma_i = \frac{a_i}{\zeta_i} \tag{2.46}$$

The ratio of the activity to an arbitrary concentration criterion (ζ) is the activity coefficient (γ). By choosing as standard condition the pure substance at system temperature, a reference pressure and the coefficient concentration of the liquid phase, the equation 2.44 transforms to:

$$g_i = g_i^{id} + g_i^e = g_i^{pure}(T, p^0) + RT \ln x_i + RT \ln \gamma_i \tag{2.47}$$

The properties of pure substances and mixtures can be estimated with a couple of methods. To distinguish between the strategies of estimation, it is useful to separate in parameter based methods and theoretical equations. The first are only valid for special mixtures and can be separated in one and two parameter models. They are simple and fast to use, but for every mixture data are needed.

2.4.3 Activity Coefficient Models

2.4.3.1 Van Laar

The Van Laar equation is a two parameter model to calculate the activity coefficients for binary liquid mixtures. The equation was derived from the Van der Waals equation. The parameters A_{12} and A_{21} are needed for every mixture and listed in databases, like Dechema [14].

$$\begin{cases} \ln\ \gamma_1 = A_{12}\left(\frac{A_{21}x_2}{A_{12}x_1+A_{21}x_2}\right)^2 \\ \ln\ \gamma_2 = A_{21}\left(\frac{A_{12}x_1}{A_{12}x_1+A_{21}x_2}\right)^2 \end{cases} \tag{2.48}$$

2.4.3.2 Margules

The Margules activity model was introduced in 1895. It model the excess Gibbs free energy of a liquid mixture as a power series of the mole fraction x_i.

$$\frac{g^e}{RT} = x_1x_2(A_{21}x_1 + A_{12}x_2) + x_1^2x_2^2(B_{21}x_1 + B_{12}x_2) + ... + x_1^m x_2^m(M_{21}x_1 + M_{12}x_2)$$

The constants A and B are derived from regression of experimental data. In praxis the B and higher order parameters are set to zero. The leading term $x_1\,x_2$ assures that the excess Gibbs energy becomes zero at $x_1 = 0$ and $x_1 = 1$. The activity coefficient of component i is found by differentiation of the excess Gibbs energy towards x_i. This yields, when applied only to the first term and using the Gibbs–Duhem equation.

$$\begin{cases} \ln\ \gamma_1 = \left(A_{12} + 2\left(A_{21} - A_{12}\right)x_1\right)x_2^2 \\ \ln\ \gamma_2 = \left(A_{21} + 2\left(A_{12} - A_{21}\right)x_2\right)x_1^2 \end{cases} \tag{2.49}$$

2.4.3.3 Universal Quasi chemical

UNIQUAC as been derived from a first order approximation of interacting molecule surfaces in statistical thermodynamics. It requires two basic underlying parameters:

1. Relative surface and volume fractions are chemical constants, which must be known for all chemicals.
2. An empirical parameter between components that describes the intermolecular behavior. This parameter must be known for all binary pairs in the mixture. This parameters are derived from experimental activity coefficients, or from phase diagrams, from which the activity coefficients themselves can be calculated.

2.4.3.4 Universal Quasichemical Functional Group Activity Coefficients

The UNIFAC method is an extension of the UNIQUAC model. The liquid mixture is no longer a solution of molecules, than a mix of functional groups. The advantage is the much smaller amount of possible functional groups than molecules [5]. The method based on the solvents molecular structure and its resulting interaction forces. It is a semi-empirical system for the prediction of non-electrolyte activity in non-ideal mixtures [15].

The model splits up the activity coefficient for each species in the system in two parts, a combinatorial γ^c and a residual γ^r part. For the i^{th} molecule, the activity coefficients are broken down as per the following equation: $\ln \gamma_i = \ln \gamma_i^c + \ln \gamma_i^r$.

2.5 Representation of thermodynamic properties

The common way to represent thermodynamic data are two dimensional diagrams. The functional context of any property can be displayed. For binary mixtures the molar ratio against the pressure, temperature, activity coefficients or the phase can be plotted.

2.5.1 Pressure Composition Diagram p(T)-xy

The functional relation between the composition and the pressure of a binary mixture can be shown in a diagram with a pressure range of the boiling points from the low-, to the highboiler at a given temperature and the molar ratio of the liquid and gas phases from zero to one. The resulting pressure at a molar liquid fraction is the sum of the saturation pressures times the fractions.

$$p = \sum_{i}^{nos} x_i \, p_i^S \,(T) \tag{2.50}$$

The saturation pressure can be calculate using the Antoine equation.

$$\log_{10} p^S = A - \frac{B}{C+T} \tag{2.51}$$

This equation is valid for ideal fluids. The behavior of real liquids and gases can be reflected using models for the activity and fugacity coefficients. For practical reasons and because of the small difference of the fugacity coefficients to one, only activity coefficient models for the liquid phase are used. The term dilate as follows:

$$p = \sum_{i}^{nos} \gamma_i, \, x_i \, p_i^S \tag{2.52}$$

The molar gas fraction result from the product of the molar liquid fractions, the activity coefficient and the saturation pressure against the pressure.

$$y_i = \gamma_i \, x_i \, p_i^S / p \tag{2.53}$$

Homogeneous methods for creation of binary pressure or temperature explicit composition diagrams can be performed with cubic eos. The equilibrium obtain if the sum of the calculated molar gas fractions equals one. The gas fractions result from the product of the liquid fractions and the separation factors. These factors yield from the devision of the fugacity coefficients of the liquid and gas phases.

2.5.2 Temperature Composition Diagram T(p)-xy

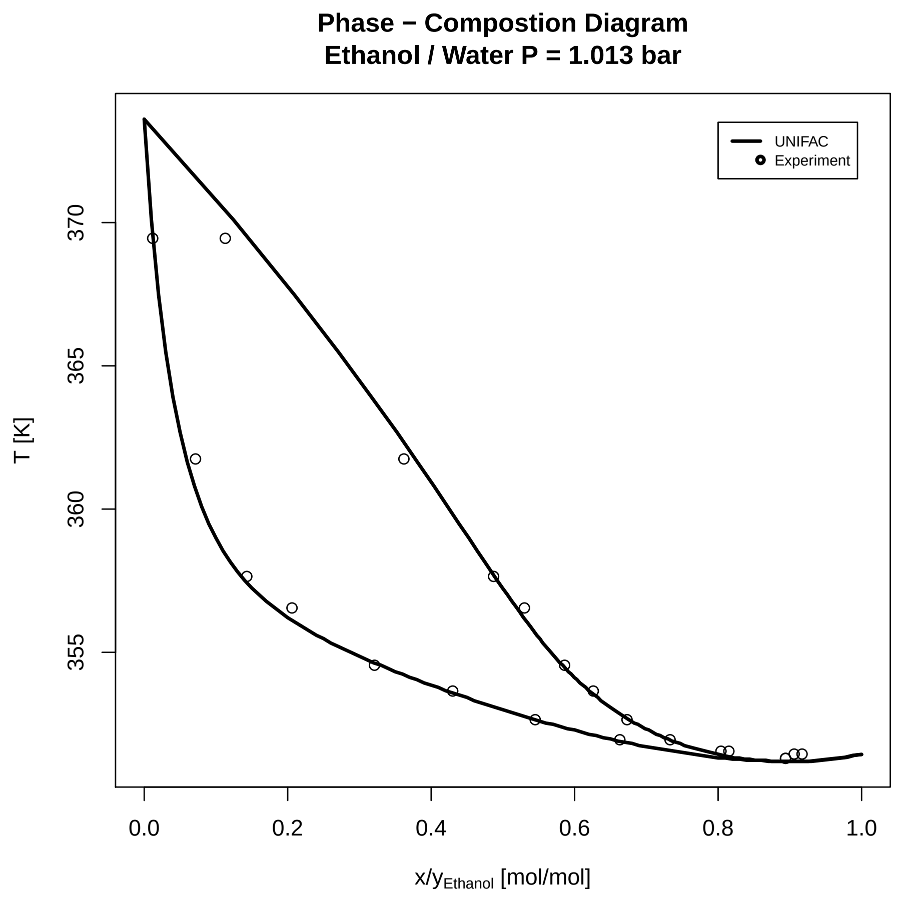


FIGURE 2.1 T(x)-y

Similarly to the pressure composition diagram at a given temperature, can be displayed the equilibrium relation between the composition and temperature at a given pressure.

2.5.3 Phase Composition Diagram xy

The vapor liquid equilibrium of a binary mixture at a specific temperature can be shown by plotting the molar fractions of the liquid- against the gas phase. All mentioned diagrams and the progression of the activity coefficients while the equilibrium states can are illustrated in Fig. 3.2 at page 35.

2.5.4 Homogeneous and heterogeneous methods

For the representation of the real behavior of the gas phase for pure substances or mixtures, are fugacity coefficients (φ) required. The equation of state has to be capable of to describe equilibrium state homogeneous for the liquid and the gas phase.

$$x_i\,\varphi_i^l = y_i\,\varphi_i^g \tag{2.54}$$

The heterogeneous characterization of the vapor liquid equilibrium can be done by using activity coefficients (γ) and the standard fugacity (f_i^0) for the liquid phase and fugacity coefficients for the gas phase. This for an activity model for the liquid and an eos for the gas phase are required.

$$x_i\,\gamma_i\,f_i^0 = y_i\,\varphi_i^g\,p \tag{2.55}$$

$$f_i^0 = \varphi_i^S\,p_i^S\,\exp\left(\frac{v_i\left(p - p_i^S\right)}{R\,T}\right) = \varphi_i^S\,p_i^S\,\mathrm{Poy}_i \tag{2.56}$$

$$\phi_i = \frac{\varphi_i^S\,\mathrm{Poy}_i}{\varphi_i^g} \tag{2.57}$$

$$x_i\,\gamma_i\,\phi_i\,p_i^S = y_i\,p \tag{2.58}$$

The term for the Poynting factor in equation 2.56 is only valid for calculations far enough from the critical point. With the assumption of an ideal gas phase, ϕ_i becomes one. It can also be neglected for substances which associates not too much, at not too high pressures. This for a small error must be accepted and the equation simplifies to a form which only includes the activity model [7].

$$x_i\,\gamma_i\,p_i^S = y_i\,p \tag{2.59}$$

For an ideal gas and liquid phase, the equation reduced to Raoult's law (Eq. 2.8).

2.6 Distillation Curve

The previous representations of thermodynamic properties are only valid for binary mixtures. Multicomponent mixtures can't be represent in a two dimensional diagram by mapping the molar fractions. The focus in this work is to reflect the behavior of real fuels with surrogate fuels. The data of fuels are very bare, but the major information is the distillation curve. The diagram results from an test procedure, where at an increasing temperature the evaporated vapor is condensed and recovered. The simulation of this

line is not insignificant, because the system is not in equilibrium. The test conditions are at atmospheric pressure, but the gas phase disappears in the condenser until no liquid is left.
In the following section are two strategies of simulations of the distillation curve demonstrated. The closed system is the first point of interest, because of the thermodynamic entireness. A thermodynamic incomplete but practicable way to assume the distillation curve of a multicomponent mixture is implemented in the open system model.

2.6.1 Closed System

No exchange of matter and isobaric conditions are required for the full description of the vaporization and condensation of multicomponent mixtures. The gas and liquid phase are ideal, to reduce the degree of freedom of this system so far as the vapor liquid equilibrium can be solved at every temperature point. At the starting point of the boilingline almost everything is liquid until the first gas bubble forms. This point is called bubble point with the convergence criteria that the sum of the calculated gas fractions equals one.

$$y_i = x_i \, p_i^S / p$$
$$\sum_i^{nos} y_i = 1$$

The end of the line characterize that nearly all becomes gas and the first drop of liquid condenses. In this case the point of view switch to the gas phase, where the original molar fractions are now gas fractions. The dew point is achieved at the temperature where the calculated sum of the liquid fractions equals one.

$$x_i = y_i \, p / p_i^S$$
$$\sum_i^{nos} x_i = 1$$

The saturated liquid line pass among this to points with a steady progression. The amount of liquid decrease continuously and the part of gas increase. The mass balance of gas and liquid ratio stays one.

$$1 = L + V \tag{2.60}$$
$$z_i = x_i L + y_i V \tag{2.61}$$

Where z_i is the original molar fraction, L notes the liquid and V the vapor amount. Solving the unknown liquid and gas fractions, following equation shows the dependence of the values.

$$z_i = x_i\left(1 - V\right) + y_i V = x_i L + y_i\left(1 - L\right) \tag{2.62}$$

Using the separation factor K_i,

$$K_i = y_i / x_i = p_i^S / p \tag{2.63}$$

the number of unknowns decrease to one as follows:

$$z_i = x_i L + K_i x_i\left(1 - L\right) = x_i\left(L + K_i\left(1 - L\right)\right) = x_i\left(L\left(1 - K_i\right) + K_i\right) \tag{2.64}$$

Because of the criteria that,

$$\sum_i^{nos} x_i = \sum_i^{nos} y_i = \sum_i^{nos} K_i x_i = 1 \tag{2.65}$$

we have $1 + N$ equations with 1 unknowns, L. By using the Gaussian elimination algorithm to solve the linear equation system (A, b), only L must be found to get all x_i.

$$\underbrace{\begin{pmatrix} L(1-K_1)+K_1 & 0 & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & L(1-K_N)+K_N \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_i \\ x_N \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} z_1 \\ z_i \\ z_N \end{pmatrix}}_{b} \tag{2.66}$$

The liquid ratio L can be found iteratively or by solving the one-dimensional optimisation problem, with the limitations between 0 and 1, by minimization of the objective function:

$$\min\{f(L)|L \in \mathbb{R}\} = \left[\sum_i^{nos} x_i - 1\right] + \left[\sum_i^{nos} K_i x_i - 1\right]. \tag{2.67}$$

The calculation is only valid for ideal fluids. Because of the dependency for homogeneous or heterogeneous methods on the liquid fractions, the equation system becomes underdetermined with no possible solution.

2.6.2 Open System Model

The open system is characterized by the transport of energy and matter through the system boundaries. This conditions are as opposed to vapor liquid equilibrium calculations, since no boundaries exists, the vapor composition stays unclear. The correct conditions are quasi open, with the following assumptions:

- equilibrium conditions of the bubble point
- mass transport from 100 to 0 gram liquid
- no heat transport
- the bubble point of the original mixture effects as start point
- the saturation temperature of the high boiler builds the final point

The model simulate the dynamic of the batch distillation by a stepwise calculation of the bubble point at an alternating mixture. The mass of the mixture starts with 100g, which equals 100% liquid. The dynamic is emulated by the subtraction of one gram liquid after the bubble point calculation. The temperature results from the closed system scenario, where the sum of the gas fractions becomes one. These fractions imply the contribution of the one gram vaporized liquid. After the conversion to mass fractions they are subtracted from the mixture. The new compound weights one gram less than the original. The mass calculations makes it necessary to know the molar mass of each species. This mass can be calculated from the chemical sum formula. The way back to mass fractions simply arise from the mass of the substance against the overall mass. The molar fractions comes from the conversion:

$$x_i = \frac{w_i/M_i}{\sum_i^{nos} w_i/M_i} \tag{2.68}$$

The mass fractions result from the inverse formula:

$$w_i = \frac{x_i\, M_i}{\sum_i^{nos} x_i\, M_i} \tag{2.69}$$

The simulation calculates the temperatures, until no mass is left. One step before are almost all substances vaporized and only the highboiler is left. Consequentially the boiling point of it provide the final temperature.

2.7 Numerical Methods

2.7.1 The R Project for Statistical Computing

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. It consists of a language plus a run-time environment with graphics, a debugger, access to certain

system functions, and the ability to run programs stored in script files. The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions [16].
All attached functions are written in R because of the comfort of the script interpreter in combine with the R Studio$^{\text{TM}}$ suite. So it facilitate interested readers to apply them and to maintain the algorithms for future students. The repository with all necessary functions is placed at `https://github.com/seihan/vlecalc`.

2.7.2 UNIFAC

The UNIFAC approach is that the knowledge about the functional groups inside the mixture substitutes the properties of the single chemicals. This requires a pre-processing. A new set of data is needed to figure out the functional groups inside the species. In a second step is this knowledge processed to calculate the surface (unu) and interaction parameters (aij) matrices. The *UNIFAC Structural Groups and Parameters* tables are required, which are free accessible on a web page from the University of East Anglia [17]. With this parameters, the molar fractions and the temperature, the activity coefficients can be summarized. The first two steps are assisted by the self defined function *UNIFAC.gen* which compares the input species with the database and generates the unu and aij matrices. The final calculations of this matrices with the fractions and the temperature takes place in the function script *UNIFAC* which returns the activity coefficients for each substance. The tables (*UNIFAC-tbl*.csv*) linked in the program, are necessary to work with it. Table 3 serves as database, where the functional groups for each species are stored. The unu data frame is structured in *Group, k, Rk, Qk,* $substance_i$ with the size of the number of subgroups and the number of substances. The aij matrix is a square matrix with the size of the number of subgroups. The number of substances is not limited, as long all group contributions are known. The matrices for the water ethanol mixture can be seen below.

```
$unu
  Group k     Rk    Qk water ethanol
1   H2O 7 0.9200 1.400     1       0
2   CH3 1 0.9011 0.848     0       1
3   CH2 1 0.6744 0.540     0       1
4    OH 5 1.0000 1.200     0       1

$aij
       [,1]  [,2]  [,3]   [,4]
[1,]    0.0 300.0 300.0 -229.1
[2,] 1318.0   0.0   0.0  986.5
[3,] 1318.0   0.0   0.0  986.5
[4,]  353.5 156.4 156.4     0.
```

Code 2.1 Generated UNIFAC matrices of the water ethanol mixture.

2.7.3 Bubble point - Ideal Phases

The state of a saturated liquid where the first bubble of vapor forms is called *bubblepoint*. The pressure of a gas is build by the partial pressures of the substances. Dalton's law stated that this pressure equals the molar fraction of a substance inside the gas and Raoult's that this equals the product of the saturation pressure and the molar fraction inside the liquid phase. Combining this laws delivers a statement for the equivalent of the saturation pressure and the pressure of the surrounding atmosphere.

$$p^S_{mixture}\left(T_{bubble}\right) = p_{system} = \sum_i^{nos} p_i = \sum_i^{nos} x_i\, p_i^S = \sum_i^{nos} y_i \tag{2.70}$$

Consequently are the molar fractions inside the gas the result of the following transformation:

$$y_i = x_i\, p_i^S / p_{system} \tag{2.71}$$

The calculation can be temperature- or pressure explicit performed [18]. While at a known temperature the saturation pressure of the mixture provides directly, an iterative algorithm is required to find the temperature where the pressures of the mixture and the system are equal. Pressures above this point hold the mixture in a liquid state and below, the state of aggregation changes to gas. The iteration is a key point for the performance and accuracy of the simulation. The manipulation can be done by optimisation or mutation. R comes with several one- or multidimensional optimisation models, but such routines mostly have the differences of alteration for the converge criterion, which also encounter without reaching the desired criterion.
To avoid this, an estimated temperature is taken to compute an initial pressure for comparison with the given. This returns a value which indicates the distance to the solution of the problem and in addition it acts to manipulate the temperature. The advantage of this mutation is the quick conversion because of the recursively changing of the manipulation value.
The following five steps are needed to solve this problem:

1. Temperature estimation from saturation temperatures (inverse Antoine).

$$T_i^S = \frac{B_i}{A_i - \log_{10}\left(p\right)} - C_i,\, T = \sum_i^{nos} x_i\, T_i^S$$

2. Calculate mixture pressure from saturation pressures (Antoine).

$$\log_{10}\left(p_i^S\right) = \frac{A_i - B_i}{C_i + T},\; p_{mixture} = \sum_i^{nos} x_i\, p_i^S$$

3. Calculate gas fractions (y).

$$y_i = x_i\, p_i^S / p$$

4. Compare the pressures.

$$S = p_{mixture}/p$$

5. If the result is close to one, return the temperature. Else mutate the it and repeat from step 2.

$$T = T + 0.1\, T\, \left(1 - S\right)/S$$

The estimated temperature result from the saturated temperatures at the system pressure. This temperature outcome from the inverse calculation of the saturated pressure and can be done from Antoine or cubic equations. This allows to uses this algorithm with the saturated values from cubic equations.

2.7.4 Bubble point - Real Liquid, Ideal Gas Phase

The extension of Raoult's and Dalton's law with activity coefficients forms the equation for real liquids.

$$p^S_{mixture} = \sum_i^{nos} \gamma_i\; x_i\, p_i^S \tag{2.72}$$

The Algorithm is close to the previous one, except the step to calculate the activity coefficients.

1. Temperature estimation.

$$T = \sum_i^{nos} x_i\, T_i^S$$

2. Calculate the activity coefficients γ using UNIFAC.

3. Calculate the mixture pressure.

$$p_{mixture} = \sum_i^{nos} \gamma_i\, x_i\, p_i^S$$

4. Calculate the molar gas fractions.

$$y_i = \gamma_i \, x_i \, p_i^S \, / p$$

5. Compare the pressures.

$$S = p_{mixture}/p$$

6. If the result is close to one, return the temperature. Else mutate it and repeat from step 2.

$$T = T + 0.1 \, T \, \left(1 - S\right) / S$$

For all upcoming calculations of real liquid phases serves the UNIFAC method for the calculating of activity coefficients.

2.7.5 Dew point - Ideal Fluids

The temperature where the first drop of liquid condense is called dew point. The point of view switches to the gas phase with the assumption that the mixture fractions are now gas fractions. The liquid composition is unknown and the solution is valid if the sum of the liquid fractions equals one.
The loop below and the diagram on page 50 illustrates the solving algorithm.

1. Temperature of the highboiler serves as initial value.

$$T = T^S_{\text{highboiler}}$$

2. Calculate equilibrium constants (K).

$$K_i = p_i^S / p$$

3. Calculate the molar liquid fractions.

$$x_i = y_i / K_i$$

4. Summarize the the molar liquid fractions.

$$S = \sum_i^{nos} x_i$$

5. If the result is close to one, return the temperature. Else mutate it and repeat from step 2.

$$T = T + 0.01\,T\,(S - 1)/S$$

2.7.6 Multidimensional Lists

The clear handling of mixtures with any number of substances with a couple of properties requires a generic, multidimensional object. The *Lists* inside R are mostly generic vectors [19]. After the initialisation and allocation of values, all objects inside the list can be accessed by number or the assigned name. New fields can be attached and existing ones removed. A list can contain lists or other object types.
The *Substances* object is a list of substances, which are also lists. Each substance is defined by name and contains various properties. This properties are stored in two tables, because the Antoine coefficients take one row for each validity range. The critical properties and the sum formulas are unique per substance and placed in an extra comma separated values (csv) table. This table could be attached with more properties in additional columns, which would be available by the name of the column. The molar mass result from the sum formula and the element masses, which are extra stored. The function *Substance* returns this properties as a list by a given name. This list will be extended with input values like the mass fraction and calculated ones like the molar fractions. Every attribute can serve for sorting, which is used to reorder the substances by the saturation temperature.

2.7.7 SRK

The homogeneous representation of the pressure-volume-temperature behavior of vapors and liquids from pure substances or mixtures can be performed using the SRK eos. The critical properties in *bar* and *K* and the acentric factor are required for each substance. For multicomponent mixtures, improves adapted binary interaction parameters (k_{12}) the accuracy of the results, which are 0.01 by default. Inside the *srk* function script are the basic functions integrated. The *absrk* handles the mixing rules. The *volsrk* returns the molar volumes and the *phisrk* the fugacities. Later are the functions *hsrk* and *ssrk* present to calculate the departure functions of the enthalpy and entropy. The templates of this programs are served from FORTRAN codes from the Book *Thermodynamik* by *Jürgen Gmehling* [5]. This functions facilitates the common equilibrium calculations for pure substances and mixtures. The most advanced and well adapted algorithm solves the cubic equation of the molar volume. The units inside the auxiliary functions are *bar*, *K* and cm^3 because of numerical reasons. The translation into SI units inside the algorithms

```
$water
$water$Formula
[1] "H2O"
$water$Antoine
       A        B        C T.min  T.max  References..P..bar..T..K..
 5.190496 1730.630 -39.724 274.15 373.15 https://en.wikipedia.org
 5.259376 1810.940 -28.665 372.15 647.15 https://en.wikipedia.org
$water$Ac
[1] 0.344
$water$Pc
[1] 221.2
$water$Tc
[1] 647.3
$water$MolarMass
[1] 18
$water$MassFraction
[1] 0.4
$water$Mass
[1] 40
$water$Fraction
[1] 0.630137
$water$Tsat
[1] 378.8919
$water$TsatSRK
[1] 375.1988
$water$MolarAmount
[1] 0.630137

$ethanol
$ethanol$Formula
[1] "C2H6O"
$ethanol$Antoine
        A       B       C T.min  T.max  References..P..bar..T..K..
 5.323356 1642.890 -42.85 216.15 353.15 https://en.wikipedia.org
 4.800357 1332.040 -73.95 350.15 516.15 https://en.wikipedia.org
$ethanol$Ac
[1] 0.635
$ethanol$Pc
[1] 63.83
$ethanol$Tc
[1] 516.2
$ethanol$MolarMass
[1] 46
$ethanol$MassFraction
[1] 0.6
$ethanol$Mass
[1] 60
$ethanol$Fraction
[1] 0.369863
$ethanol$Tsat
[1] 352.6689
$ethanol$TsatSRK
[1] 351.1713
$ethanol$MolarAmount
[1] 0.369863
```

Code 2.2 Multidimensional list example for the binary ethanol water mixture and its properties at 1e+5 Pa.

failed by an increased error for the iterated results of T and p, it takes respect after the input and before the return inside the main routine. The calculations for the bubble- and dew point are pressure or temperature explicit possible. In addition are routines for the saturation temperature and pressure of pure substances implemented. This initiates three thermodynamic models. At first the homogeneous method for mixtures to get their vapor-liquid conditions. The circumstance to get the saturation pressure and temperature of pure substances enables calculations according Raoult's and Dalton's law. The last model is the extension of the second by inserting an activity model. Common to all of the following algorithms is the fact to estimate at first the state variables of interest. This prevent the automation and integration of this models into simulations. The estimated value must be very close to the resulting one and can't be something like the mean saturation values. The University of Cambridge offers a very good example for vle calculations in form of an online calculator for pure substances and binary mixtures, using cubic eos [10]. The estimation routines are migrated from that function script.

2.7.8 Bubblepoint - Ideal Liquid, Real Gas Phase

The prediction of the vapor liquid equilibrium temperature at a given pressure, with cubic equations, operates with the capabilities to describe the vapor and the liquid phase. The following algorithm has been successfully proofed with the SRK eos.

1. Estimate the temperature and the gas fractions.
2. Calculate the mixture parameters a and b for the liquid phase.
3. Calculate the molar volumes and fugacities for the liquid phase.
4. Calculate the mixture parameters a and b for the gas phase.
5. Calculate the molar volumes and fugacities for the gas phase.
6. Calculate the equilibrium factors K:

$$K = \varphi^l/\varphi^g,\ y = x\,K$$

7. Summate the molar gas fractions:

$$S = \sum_{i}^{nos} y_i$$

8. If the sum is very close to one, return the temperature. Else mutate it and the gas fractions and repeat from step 2.

$$T = T + T\,0.1\,(1-S)/S,\ y_i = K_i\,x_i/S$$

2.7.9 Dewpoint - Real Gas, Ideal Liquid Phase

The routine to calculate the dew point temperature with the SRK eos is quite similar to the bubble point algorithm, with the distinction of the unknown liquid fractions.

1. Estimate the temperature and the liquid fractions.
2. Calculate the mixture parameters a and b for the liquid phase.
3. Calculate the molar volumes and fugacities for the liquid phase.
4. Calculate the mixture parameters a and b for the gas phase.
5. Calculate the molar volumes and fugacities for the gas phase.
6. Calculate the equilibrium factors K:

$$K = \varphi^l / \varphi^g, \ x = y/K$$

7. Summate the liquid fractions:

$$S = \sum_{i}^{nos} x_i$$

8. If the sum is very close to one, return the temperature. Else mutate it and the liquid fractions and repeat from step 2.

$$T = T + T\,0.1\,(S-1)/S, \, x = x/S$$

2.7.10 Saturation State of Pure Substances

The point where the state of aggregation changes from liquid to gas can be calculated or predicted for pure substances with various approaches. The pressure and temperature explicit terms of the Antoine equation 2.51 and its inverse are previously shown inside the bubble point algorithm for ideal phases on page 20. This equation follows an empirical relationship and requires coefficients, fitted to experimental data. Cubic eos offers a complete thermodynamic model to predict the state of the liquid and the gas phase. The condition for the equilibrium between two phases are valid if the pressure, the temperature and the chemical potential are equal in both phases. It can be deduced that the fugacity coefficients in this case must be equal, too [7]. An iterative procedure alternates the wanted state variable until this condition has been accomplished.

Saturation pressure:

1. Estimate the saturation pressure.
2. Calculate the parameters a and b for the liquid phase.
3. Calculate the molar volume for the liquid phase.
4. Calculate the fugacity coefficient for the liquid phase.
5. Calculate the parameters a and b for the gas phase.
6. Calculate the molar volume for the gas phase.
7. Calculate the fugacity coefficient for the gas phase.
8. Compare the fugacity coefficients:
$$S = \varphi^l - \varphi^g$$
9. If the result is close to zero, return the pressure. Else mutate it and repeat from step 3.
$$p = p + p\,S$$

Because of the temperature dependence of the parameters a and b includes the iteration loop for the saturation pressure not this step, while the saturation temperature loop alternate all values.

Saturation temperature:

1. Estimate the saturation temperature.
2. Calculate the parameters a and b for the liquid phase.
3. Calculate the molar volume for the liquid phase.
4. Calculate the fugacity coefficient for the liquid phase.
5. Calculate the parameters a and b for the gas phase.
6. Calculate the molar volume for the gas phase.
7. Calculate the fugacity coefficient for the gas phase.
8. Compare the fugacity coefficients:
$$S = \varphi^g / \varphi^l$$
9. If the result is close to one, return the temperature. Else mutate it and repeat from step 2.
$$T = T + 0.1\,T\,(S-1)/S$$

2.7.11 Distillation Curve - Closed System

The instance for the representation of the vapor liquid equilibrium inside a closed vessel can be separeted in two parts.

1. Compute the bubble- and the dew point temperatures - $f(p, x)$.
2. Solve an one dimensional problem between this points - $f(p, x, T)$

The first part returns the limitations of the temperature interval, which decrease the degree of freedom of this system. This makes it possible to solve for the unknown vapor liquid composition.

Algorithm 1 Calculate boiling line in a closed system

```
Require: $nos > 1, b = fractions, p_{system} > 0, e = 1^{-7}$
  function OBJF(l)
      $x_f = l\,(1 - K) + K$
      $A = \text{matrix}(0, nos, nos)$
      for $i = 0; i < nos; i++$ do
          $A[i, i] = x_f[i]$
      end for
      $x = \text{solve}(A, b)$
      return $\|\text{sum}(x) - 1\| + \|\text{sum}(K\,x) - 1\|$
  end function

  $T_{bubble} = \text{calc.bubble}(x, p)$
  $T_{dew} = \text{calc.dew}(x, p)$
  $T_{range} = \text{seq}(T_{bubble}, T_{dew}, 1)$
  $steps = \text{length}(T_{range})$
  for $i = 0; i < steps; i++$ do
      $K = p^S(T)/p_{system}$
      $l = 1$
      $l\,[i] = \text{optimize}(f = \text{OBJF}, lower = 0, upper = 1, tol = e)$
  end for
```

2.7.12 Distillation Curve - Open System

The distillation curve model mimic the behavior of a boiling multicomponent mixture with a steady lost of vapor. This dynamic is implemented by a virtual mass of 100g, which comes from the mass fractions shifted by two decimal points. This mass is the inverse of the recovered vapor. The several bubble point models operate with molar fractions, wherefore the transformation between this kind of fractions is necessary. This step is distinguished with little errors because of finite internal decimal places in computers, which become noticeable by divisions. The subtracted gas fractions outcome from an iteration process with a limited accuracy. In sum they are only very close to one and deviate from it after the transformation. This circumstances can be affected by setting the mass of a substance to zero, if it is smaller than 10^{-10} g. The stepwise simulation has the characteristic of gaps between one and the next step. It don't effect the quality of the diagrams, but prevent various analytic post-processings. The function *approxfun* returns a function from a set of data points, which performs the interpolation on any point inside the given interval. This makes it possible to compare the simulated results with measured values. The simulation algorithm can be expressed with the following steps:

1. 1st initialisation step: Get the properties of the substances and compute the molar weight, the mass and the saturation temperatures (Antoine, SRK).
2. 2nd initialisation step: Sort the list by T^S, calculate the overall and the molar mass to compute the molar amount per substance, which correspond to the molar fractions. Fill the first column of the mass stack.
3. Do the simulation loop while the mass of the mixture stays over zero gram.
    - Get the bubblepoint temperature and the molar gas fractions from one of the five models.
    - Subtract the one gram gas from the liquid, store the masses and the temperature.
    - Calculate and store the vapor $= 100 - mass$.
4. Finally set the T^S of the highboiler for the last temperature point at 100% vapor.
5. Call the approximation routine to return the distillation curve function and build the massstack to draw the progression diagram.

2.7.13 Progression Diagram

The progress of the substances during the distillation is a useful by-product. It illustrates from the overall mass at the top of the y-axis and the initial temperature on the x-axis until zero gram at the final boiling point, the degeneration of each substance. It is processed by recording the mass from the substances during the simulation and a post-processed summation.

1. Recording the masses into the *massstack*.

Substances mass per row

Temperature step per column

2. Building the stack by reverse summation of the masses from the last to the first row.

3. Plot the progression line for each substance (masses, temperatures).

4. Draw polygons from the substance masses and temperatures, to zero mass on reverse temperatures.

2.7.14 Blending by Optimisation

The previous models for the representation of distillation curves and their functional context, provide the possibility to create mixtures fitted to measured values. The problem is based on the differences of the curves.

$$\min\{f(x)|x \in \mathbb{R}\} = \sum \|T\left(\%\,\mathrm{evap}_{samples}\right) - T\left(\%\,\mathrm{evap}_{simulated}\right)\| \tag{2.73}$$

To solve this problem, the constraints on the unknown fractions (x) must be respected.

$$0 < x_i < 1\,, \sum_{i}^{nos} x_i = 1\,.$$

The simulation of the distillation curve and the post-processed functional context between the evaporated fractions and their temperature, is called inside the objective function. The returned approximated function enables the calculation of temperatures at every point inside the interval from 0 to 100% evaporated fractions. The results are compared with measured as described in equation 2.73 and returned to the solver. The handling of this multidimensional problem performs the *optim(par, fn)* function, which is part of the base *stats* package in R [20]. The input consists of a initial set of fractions and the objective function. The output includes the best set of parameters found and the value corresponding to the parameters. The limitations are observed inside the objective function by two steps.

1. Make the parameters absolute.

$$x = \|x\|$$

2. Normalize the fractions.

$$x = x/\sum_{i}^{nos} x_i$$

The whole solving of this mixture problem can be expressed in a short and concise manner by the following sequence:

- Call the routine with the measured values, the substances and the pressure.
- Inside the function starts the initialisation for the distillation function, with the initial or the default equimolar fractions and the chosen or the default Antoine model, as well as the samples.
- This variables are global, in point of view from the objective function, which is complete with the samples and does two jobs.
    1. Get the distillation function.
    2. Compare the samples with the approximated (Eq. 2.73) and return the difference.
- Finally the optimisation goes until the differences of the returned and the previous values are tiny.

Chapter 3

Results

3.1 Models for Real Phases

The non-ideal behavior of mixtures is observed in two strategies, a homogeneous model and UNIFAC methods are realized. The SRK equation handle with the fugacity, which pointing more on the gas side, while the UNIFAC acts on the liquid. The differences of this methods are small on regular fuels, but huge at polar mixtures like ethanol-water. The binary interaction parameter (k_{12}) inside the SRK equation influence the equilibrium calculations in a significant manner. Calculations with the ethanol-water mixture indicates how close or far the results reach experimental data. Solving the problem of the tightest equilibrium line in compare to the measured one deliver a fitted k_{12} for this mixture as shown in figure 3.3. The big variety of results depending on this value advise to proof the competence of the SRK equation. For multicomponent mixtures with N substances are N - *1* k_{12} needed to cover the whole mixture. Because of the high number of species inside the FACE fuels a fixed value of 0.01 works as interaction parameter for all mixtures. The binary mixture of water and ethanol serves as a good benchmark to proof the precision of the single models. At figure 3.2, the individual methods are compared on the basis of usual equilibrium calculations. It demonstrates the basic functionality and the slightly differences of the single models.

3.1.1 Binary Mixtures

The prediction of the real behavior of binary mixtures is possible with a couple of methods. Most of them are rich of empirical parameters which makes them unhandy for general algorithms. The Van Laar, Margules, UNIQUAC, SRK methods are compared on figure 3.1.

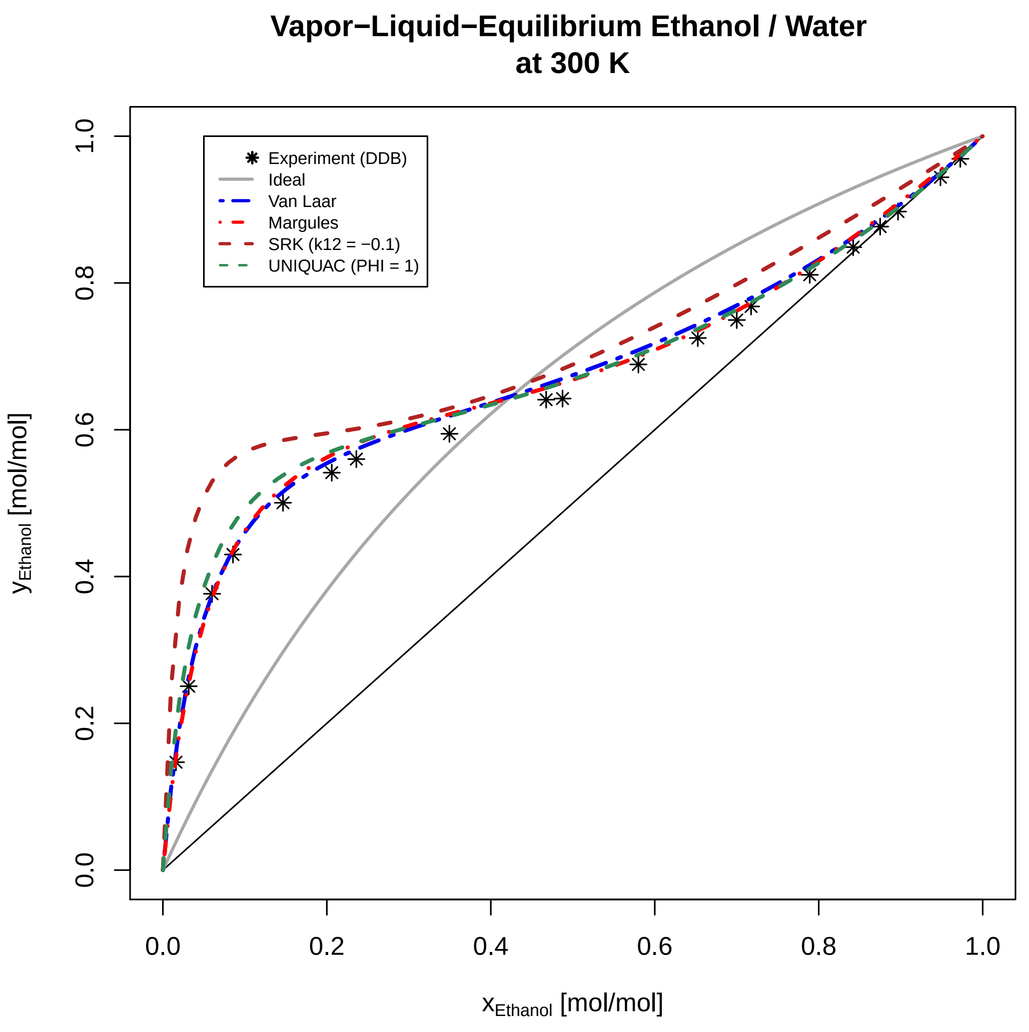


FIGURE 3.1 Vapor liquid equilibrium line of the binary ethanol water mixture, processed with different methods.

3.1.2 The SRK Function Script

The SRK cubic eos is implemented in the *SRK.R* function script and contains the following routines for the pressure- or temperature explicit vle state:

- Saturation state of pure substances
- Bubblepoint state of multicomponent mixtures
- Dewpoint state of multicomponent mixtures
- Molare volumes of pure substances and mixtures

The returned values contain the fugacity coefficients of the liquid and the gas phase, the wanted molar fractions, the departure functions of the enthalpy and entropy, the

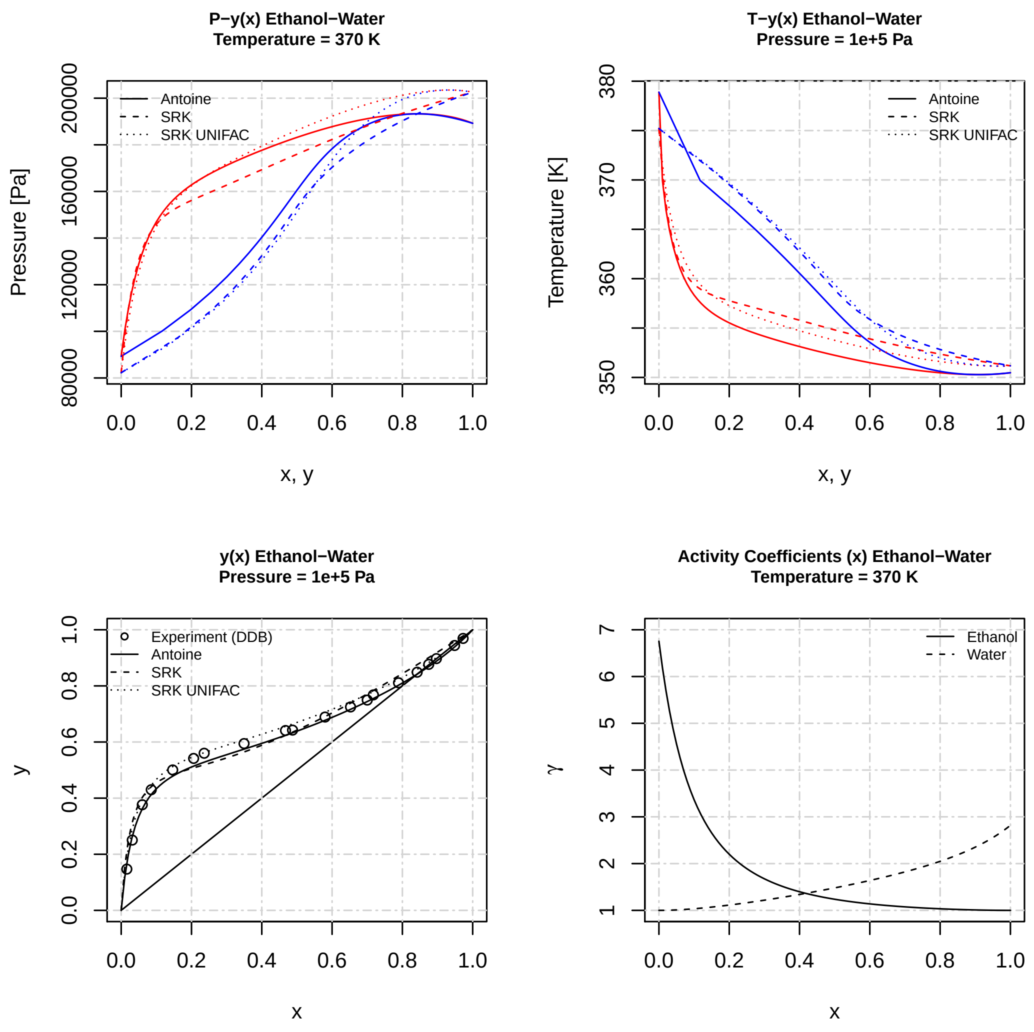


FIGURE 3.2 Equilibrium calculations with the Antoine UNIFAC, SRK and SRK UNIFAC models.

conversion criterion number (S) and the number of iterations. The function is modulare structured. First are the routines of the cubic equation. The paramater summation (a, b, am, aij, alpha), the solving of the cubic equation of the molar volume, the fugacity coefficients calculation and the departure functions of the enthalpy and entropy. Next are the algorithms for the named problems, which can be used by choosing the method at the function input. This choice and the critical values, the acentric factors (Tc[K], Pc[bar], Ac) stored in single vectors with the length of the number of substances, the state variables (T[K], p[Pa]) and the molar fractions are the required input parameters. The molar fractions are obsolete for the saturation state values. The algorithms can be easily expanded with other vle calculations by the combination of the srk functions and the adding of the new method inside the switch statement.

3.1.3 Binary Interaction Parameter

The use of the SRK eos on mixtures returns more accurate results with adapted binary interaction parameters. This values are not widespread available in the literature or online databases. This motives are responsible for the prediction method of this parameter, which base on the differences of equilibrium curves as shown in Fig. 3.3.

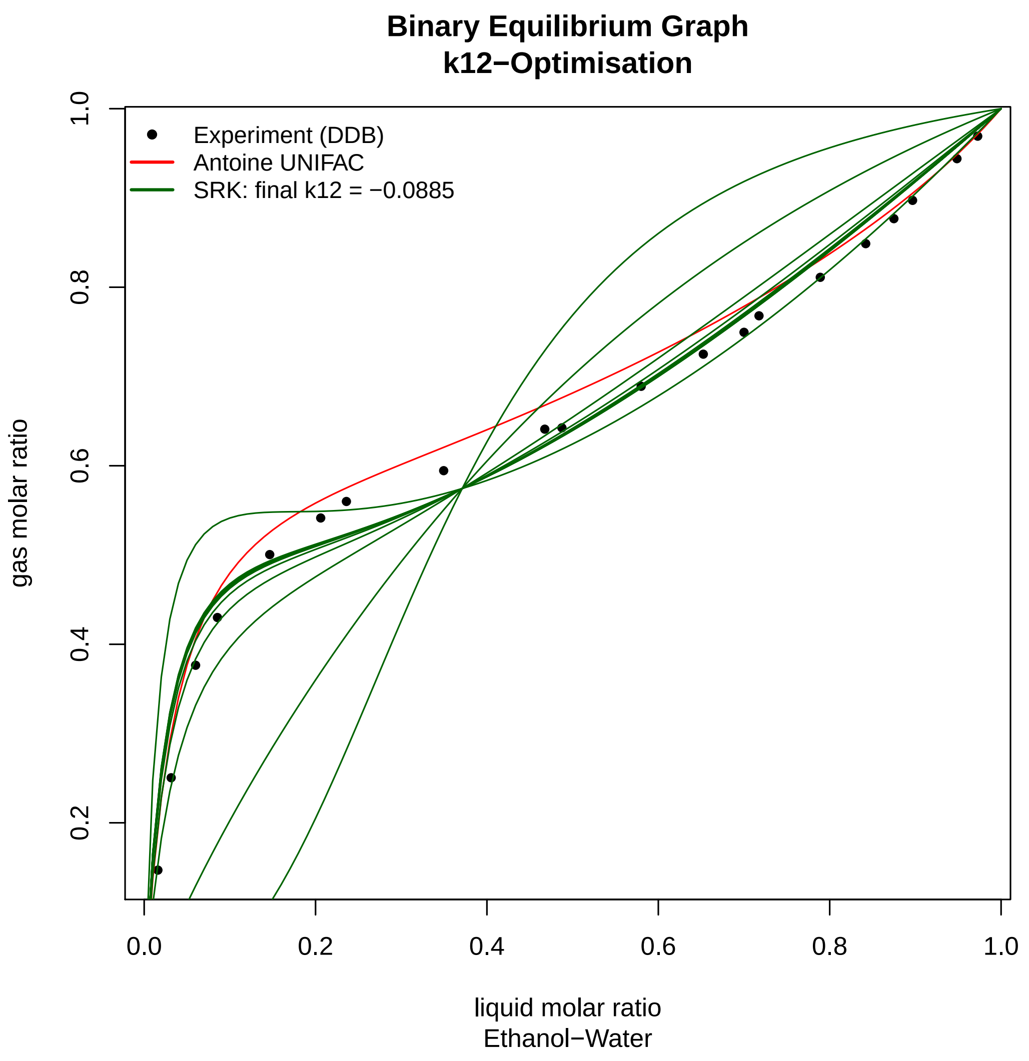


FIGURE 3.3 Optimisation progression while finding the closest k_{12} value, which is necessary for the cubic SRK eos.

3.2 Distillation Curves

The distillation curves of multicomponent mixtures, simulated by the open system model, results from a vaporization model. One goal is the relative small amount of data needed to calculate the saturation pressures. For pointing out of similarities and

differences, two ways of prediction are realized. The *Antoine* equation and the use of cubic, SRK eos. The first is very fast and accurate but limited by a temperature range. Such data of substances are very rare and stored in three different formats. The cubic equations potentialize vapor liquid equilibrium calculations for multicomponent mixtures in a homogenous manner. The critical properties and the binary interaction parameters are required. This parameters are rare, but the influence depends on the distinction of the substances, like polarity. In a mixture of very similar hydrocarbons decrease it's influence, by adding polar substances like ethanol, it increase. The possibility of calculating the saturation states of pure substances using cubic equations, opens a third way of prediction, by the combination of the cubic and *Raoult's* equation, similar like this with Antoines.
The non ideal behaviour of mixtures increase with the dissimilarity of the substances. One common model to encounter this problem is the use of UNIFAC. This method is combined with the Antoine and cubic model. The circumstance to predict the distillation curve of multicomponent mixtures potentialize the blending of mixtures to obtain a desired distillation curve, by solving a multidimensional optimisation problem.

3.2.1 Closed System

The distillation curves from a closed system scenario are a by product from the research for vaporization methods. They are more for future interests, like simulations in batch reactors. The function is a good example for using the lever rule on ideal multicomponent mixtures and to illustrade the not linear but steady progression. On every temperature point along the curve are the molar fractions for the liquid and the gas phase available and could post-processed to progression curves.

3.2.2 Open System

The open system model works very close to experimental data. The dynamic of vaporization is fitted by a regression method which changes the mixture step by step until only the highboiler is left. It handle Antoine and cubic equations with or without UNIFAC. The modular construction makes it possible to include other prediction methods or adjust the included ones. The simulations are quick enough for the use on common PC's or smart devices, the Antoine model for example takes less than 1s on a 1.87GHz CPU for a mixture with 11 compounds.
The data from experiments are taken from *Surrogate Model Development for Fuels for Advanced Combustion Engines* to compare with their results. Because sometimes the

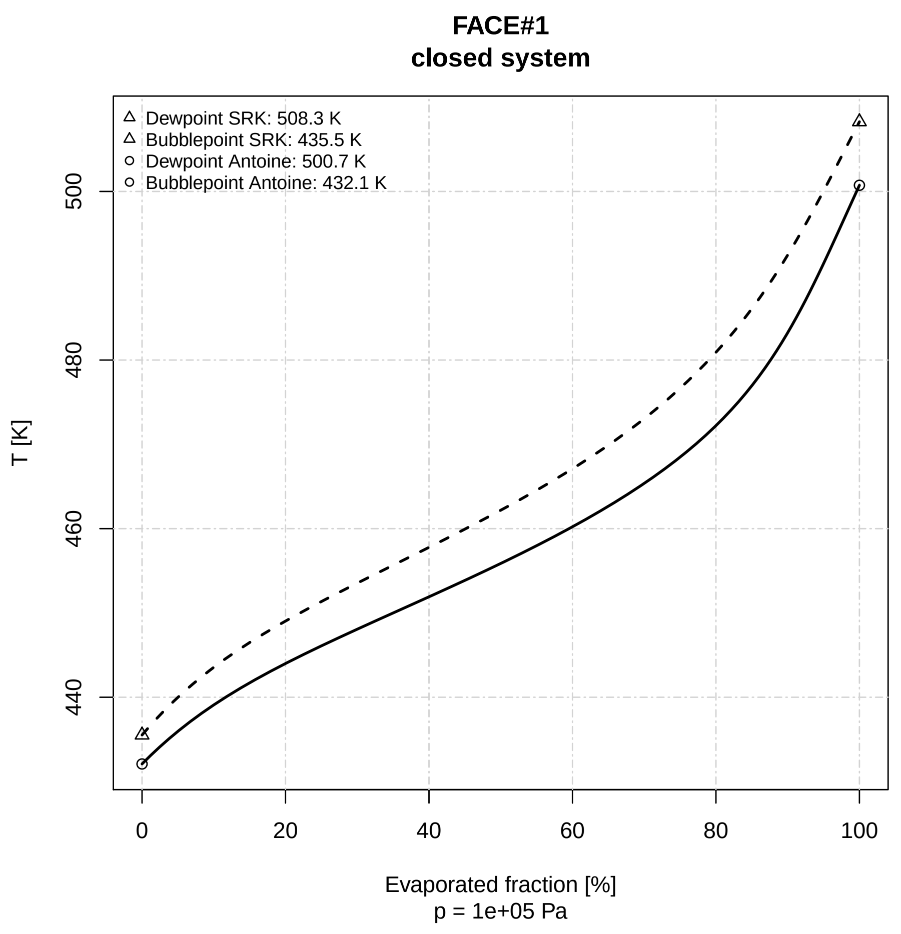


FIGURE 3.4 Boilingline in a closed system scenario

data is captured from diagrams with less resolution but thick lines, the deviation is of quality nature only.

The output is not limited at distillation curves. The mass progression of the included substances enable a view into the changing amount of the species inside the mixture.

3.2.3 Ethanol Fuels

Ethanol is a highly polare substance which effects the models very sensitive. The use of activity coefficient methods are irreplaceable for mixtures containing such polare compounds. To take a picture of it's influence, the distillation curve of an randomly produced surrogate mixture containing 10% ethanol can be found on Fig. 3.7. It certain illustrade the comparison of the models with and without UNIFAC, noticeable in the differences of the initial boilingpoint of more than $50K$.

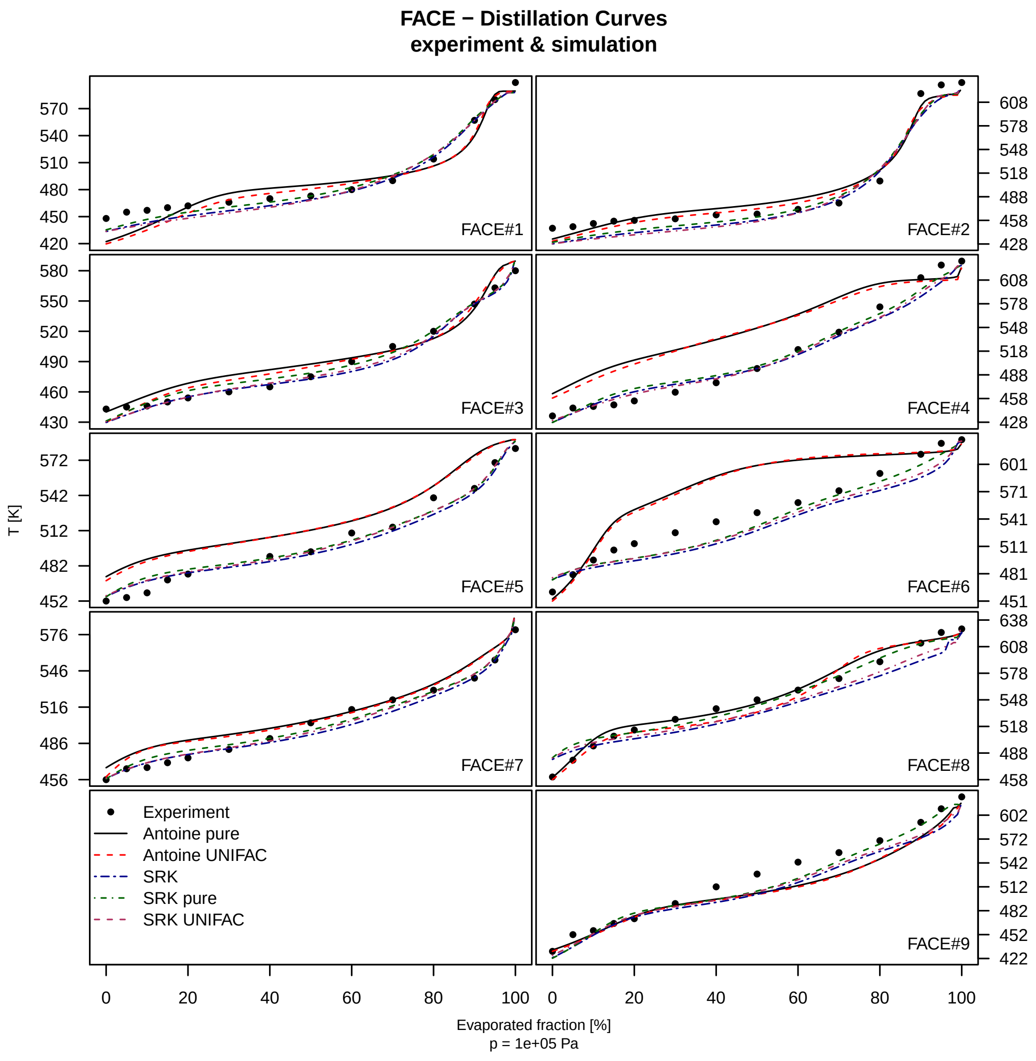


FIGURE 3.5 Distillation curves in a open system scenario.

3.3 Optimal Blending

The blending of mixtures using optimisation routines based on the comparison of the distillation curves. The simulated curve is converted in a function which returns the temperature for every point of the x-axis. The sum of the sample temperatures minus the simulated is the minimisation value returned by the objective function. The *optim* function generates new parameters until the conversion criterion or the iteration limit is reached. This criteria based on the differences of the returned minimisation values. The principle works for any number of substances and for any temperature ranges. The output highly depends on the selected substances and their saturation temperatures. On figure 3.8 is the optimised distillation curve based on FACE#9 illustrated.

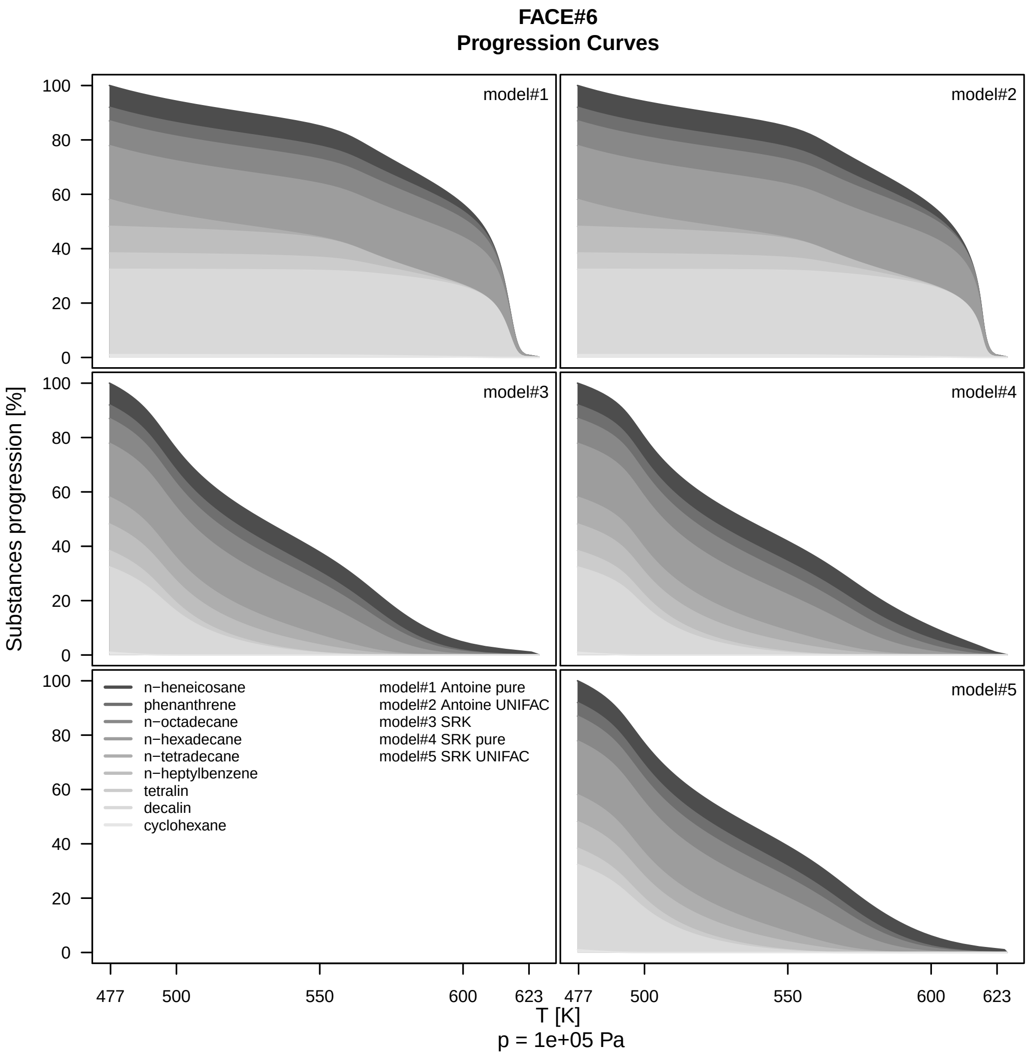


FIGURE 3.6 Substances progression during the vaporization for the different models.

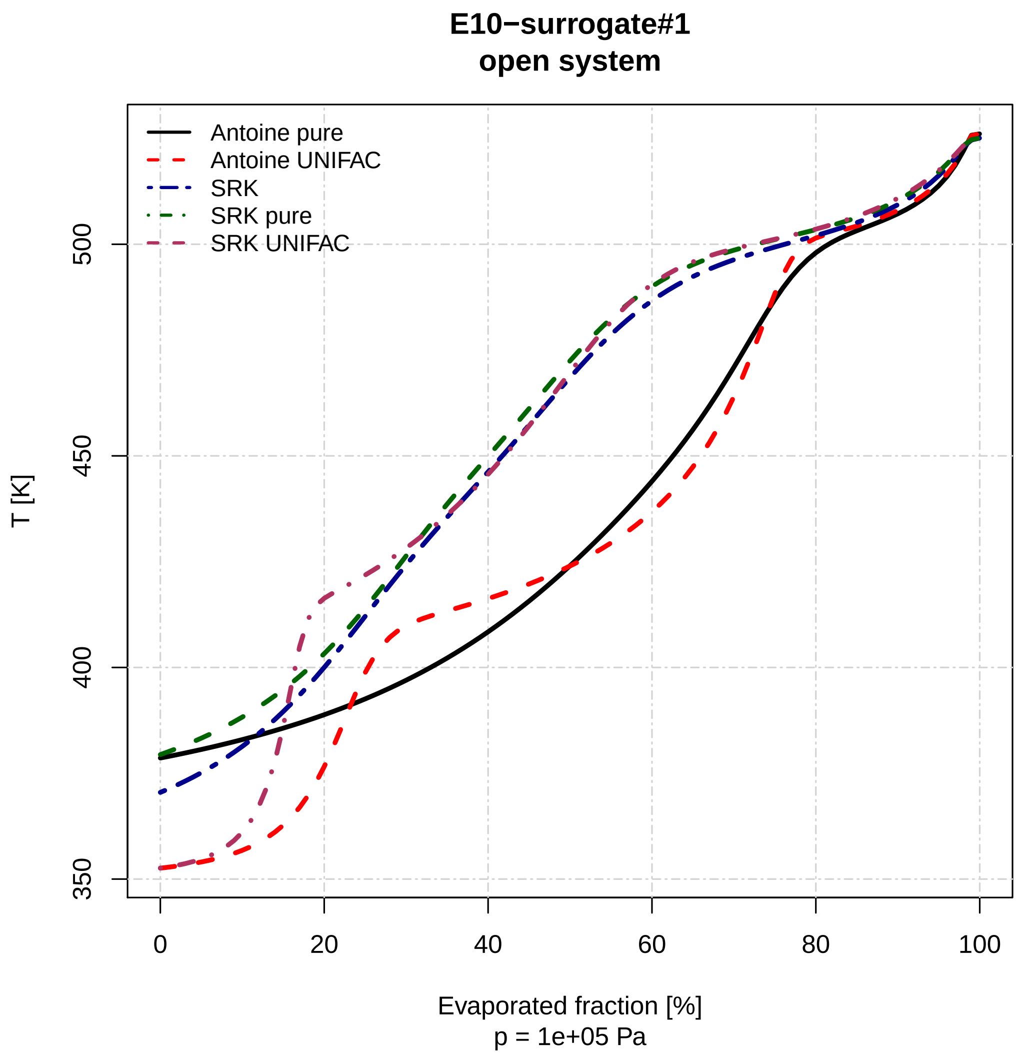


FIGURE 3.7 Distillation curves from a randomly blended surrogate fuel, containing 10% ethanol.

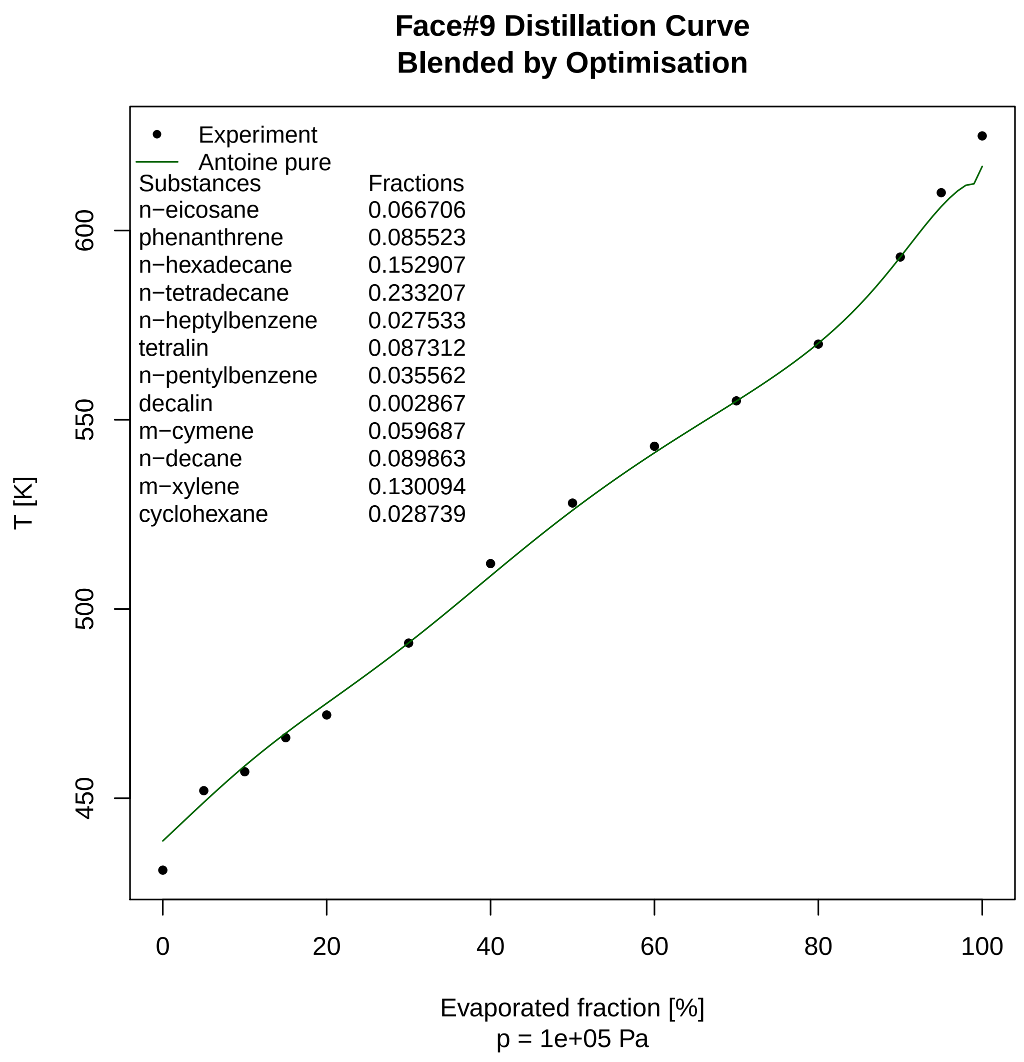


Figure 3.8 FACE#9 distillation curve - resulting from multidimensional optimisation of the mass fractions.

Chapter 4

Discussion

The prediction of thermodynamic properties of multicomponent mixtures is already possible using simple models in combine with experimental data. The challenge to increase the accuracy and decrease the dependency and the number of material properties has successfully passed, with the three parameter correspondence principle of the cubic eos. The comparison based on calculations of saturation pressures shows values, wich are very close on a wide range. The differences increase close to the critical point to the detriment of the Antoine constants routine, while the SRK eos hit this point. The higher computation complexity of the cubic equations serves not longer as argument in this decade. Some results of the method comparison study is shown on figure 4.1. The last point deviation names the distance between the cubic and the result of the Antoine model. The most relevant result is the simplification of the prediction technology for distillation curves. The modelisation of the open system on the most uncomplicated way initiate a basic setup with satisfying results. This could be improved with several adaptions to enhance the accuracy. First before, questions about the sense of such improvements has to be discussed. Do the measured samples reflect the nature of vaporization with a disregarded error? The experimental setup for the ASTM D86 testing method include some sources for errors. For example are the monitoring of the temperature and the recovered vapor not coupled. The lost of heat by radiation and conduction through the faces of the vessels are neglected too. One significant of the sampling occurs in the final boilingpoint, which is mostly higher than the saturation temperature of the highboiler. The simulation don't reflect the real transport of matter as it happens in a boiling mixture. The diffusion of liquid molecules in rising gas bubbles and the transport by the friction and adhesion between both phases are neglected. The assumption the highboiler exists until the and of the vaporization only an ideal. Small concentrations would cause it disappears because of the phenomena, while the model predicts 100% in the last gram. On the opposite are the long durability of the lowboilers not only representative too. The initial boilingpoint of the simulation is very close to that of the samples, but almost always different between the single models. This and the gap of modeled curves highly depends on the differences of the substances properties, which is significant pictured on the method comparison diagrams on page 45. Summarily are the errors part of the sampling and the predicting. The Antoine constants are rare values and limited in a temperature range too. Some of this ranges are below the simulation temperature, which withhold precise predictions. Most constants are provided by the National Institute of Standards and Technology (NIST). Because of the legal regulations of the NIST, constants can not be presented for discussion here. For some mixtures the temperatures close to the final boilingpoint (evap. frac. $> 90\%$) are not steady increasing. The reasons are unclear, but probably the conversion step from molar to mass and back to molar fractions produces rounding errors. It happens not at every mixture and mostly by using the Antoine models. The progression diagram gives an other view inside the vaporization process. One surprise is the long endurance of the highboiler at FACE#6 utilized with the first two models. Because of the significant differences of the distillation curves pointing it that failures with the saturation pressures blow up the complete simulation. The first attempt is that the values for cyclohexane are wrong, but the comparison neglect this. Ones again strikes the cubic eos. All SRK models produce the expected regression of the lowboilers at first and the highboilers at last.

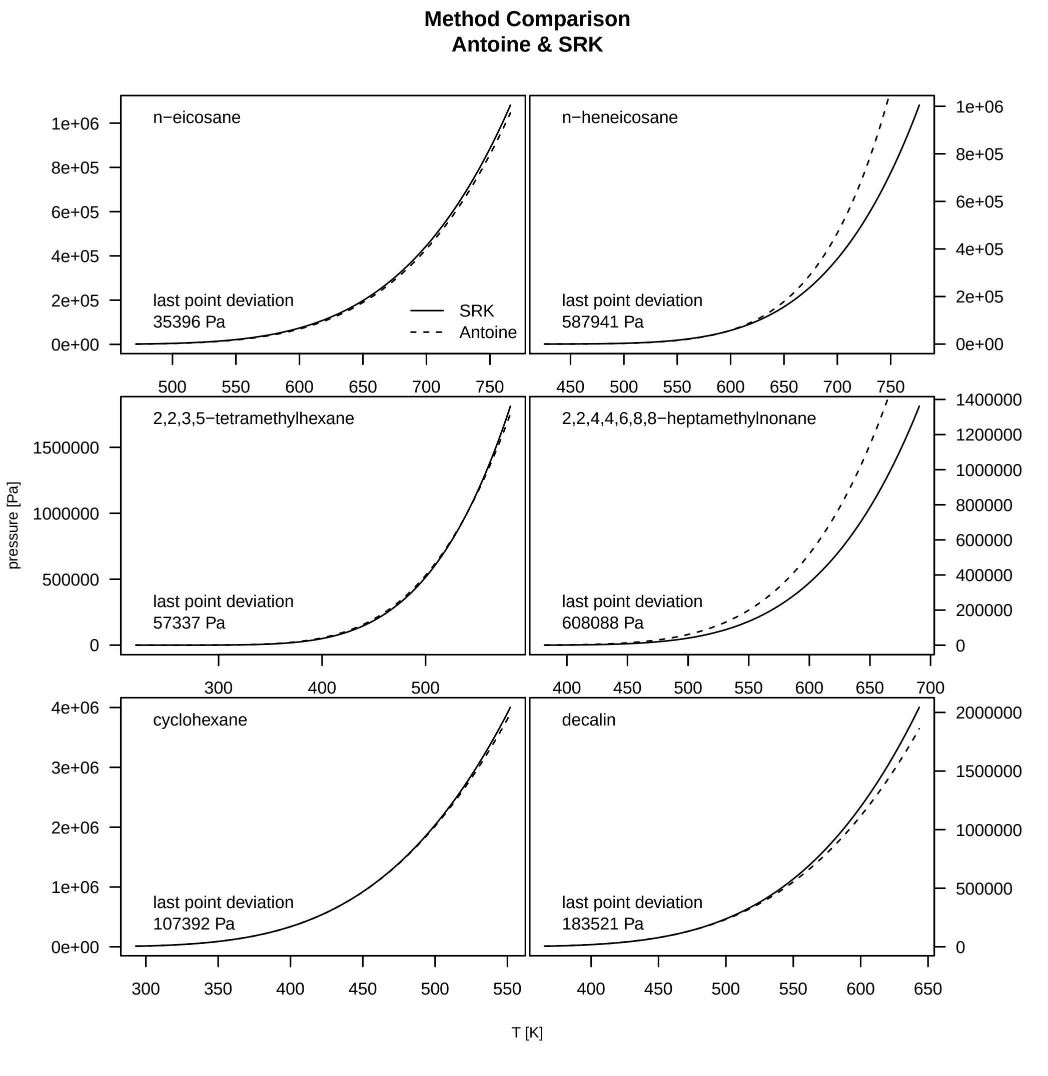


Figure 4.1 Saturation pressure calculation for pure substances from the triple to the critical point.

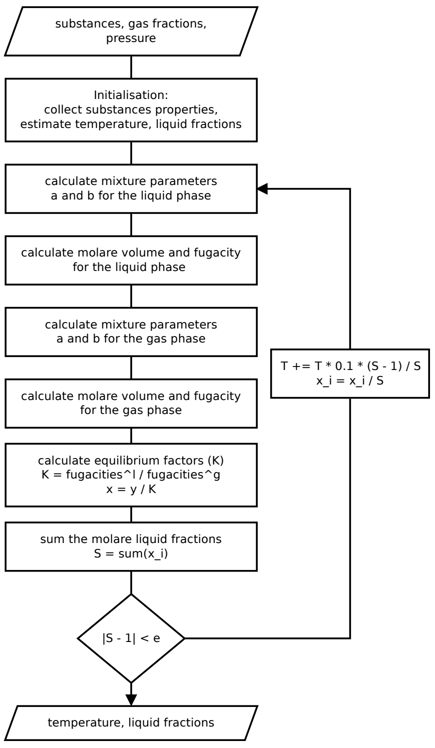
Chapter 5

Conclusions

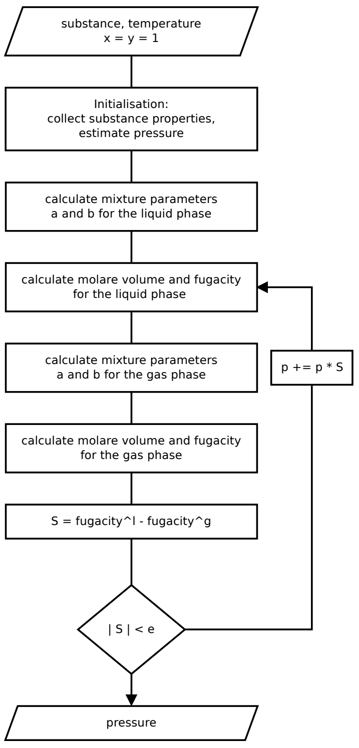
The handling of multicomponent mixtures with an arbitary number of substances is quite possible using open source software in combine with public available data. The surrogate fuels serves as a good example for the demonstration of the capability of modern equations of state. For further investigations are strategies for thermodynamic consistent tests useful to adapt this constants, or better to improve the results from cubic eos with valid critical properties and an adapted acentric factor. Some test of changing this factor supplies some better set of outcomes at single species. To focus the use of cubic eos is recommended because of the quality in prediction and of course the more available critical values. If the acentric factor is not present, but valid Antoine constants, the calculation of it with equation 2.4 is advised to become independent of this constants and to widespread the validity range of vle calculations. The modular structure of the object oriented programming enables some usefull combinations of the thermodynamic models like previously named. Furthermore are automatisations possible to predict the binary interaction parameter using UNIFAC. All in all are a lot of thermodynamic problems manageable using the major programs the SRK eos, UNIFAC and the Substance framework. Examples are predictions of liquid liquid equilibrium calculations, liquid solid predictions or gas solubility in liquids. The expansion of the closed system boilingline with respect to real phases, could be also of interest for a couple of problems.

Appendix A

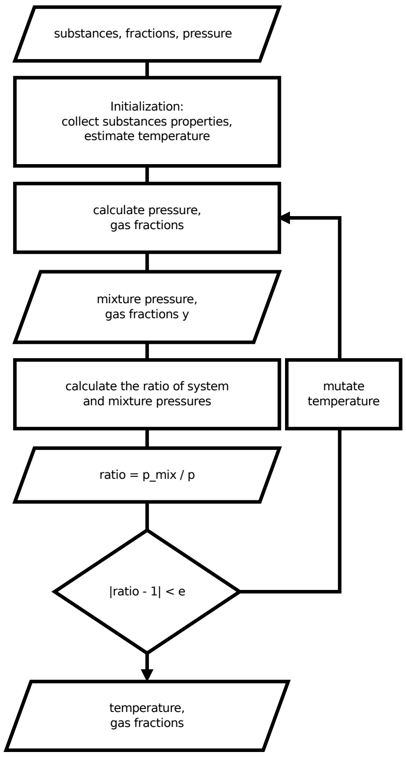
Appendix


substances, gas fractions, pressure
Initialisation: collect substances properties, estimate temperature, liquid fractions
calculate mixture parameters a and b for the liquid phase
calculate molare volume and fugacity for the liquid phase
calculate mixture parameters a and b for the gas phase
calculate molare volume and fugacity for the gas phase
calculate equilibrium factors (K) K = fugacities^l / fugacities^g x = y / K
sum the molare liquid fractions S = sum(x_i)
|S - 1| < e
T += T * 0.1 * (S - 1) / S x_i = x_i / S
temperature, liquid fractions

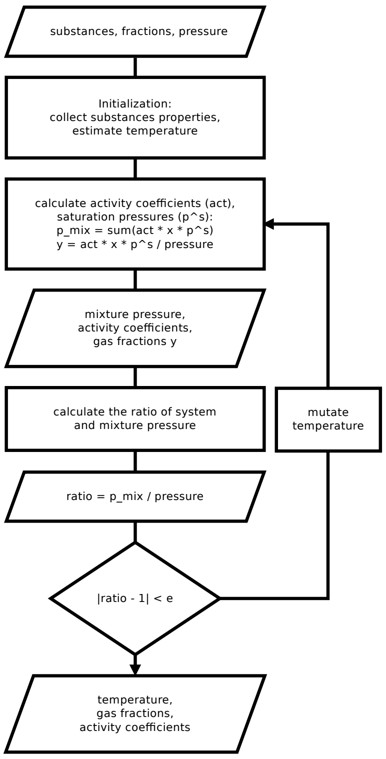
(A) Algorithm to calculate the dewpoint temperature using the SRK eos.


substance, temperature x = y = 1
Initialisation: collect substance properties, estimate pressure
calculate mixture parameters a and b for the liquid phase
calculate molare volume and fugacity for the liquid phase
calculate mixture parameters a and b for the gas phase
calculate molare volume and fugacity for the gas phase
S = fugacity^l - fugacity^g
| S | < e
p += p * S
pressure

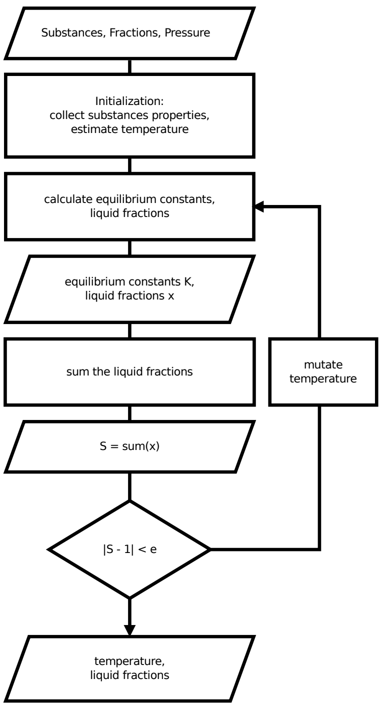
(B) Algorithm to get the saturation pressure of pure substances using the SRK eos.


substances, fractions, pressure
Initialization: collect substances properties, estimate temperature
calculate pressure, gas fractions
mixture pressure, gas fractions y
calculate the ratio of system and mixture pressures
mutate temperature
ratio = p_mix / p
|ratio - 1| < e
temperature, gas fractions

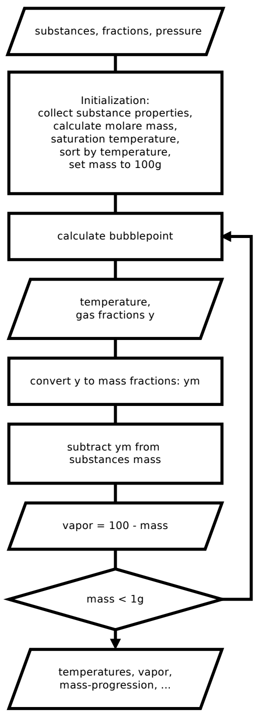
(A) Algorithm to calculate the bubblepoint temperature using the saturation pressure.


substances, fractions, pressure
Initialization: collect substances properties, estimate temperature
calculate activity coefficients (act), saturation pressures (p^s): p_mix = sum(act * x * p^s) y = act * x * p^s / pressure
mixture pressure, activity coefficients, gas fractions y
calculate the ratio of system and mixture pressure
mutate temperature
ratio = p_mix / pressure
|ratio - 1| < e
temperature, gas fractions, activity coefficients

(B) Algorithm to calculate the bubblepoint with real liquid phase using UNIFAC.


Substances, Fractions, Pressure
Initialization: collect substances properties, estimate temperature
calculate equilibrium constants, liquid fractions
equilibrium constants K, liquid fractions x
sum the liquid fractions
mutate temperature
S = sum(x)
|S - 1| < e
temperature, liquid fractions

(C) Algorithm to calculate the dewpoint temperature using the saturation pressure.


substances, fractions, pressure
Initialization: collect substance properties, calculate molare mass, saturation temperature, sort by temperature, set mass to 100g
calculate bubblepoint
temperature, gas fractions y
convert y to mass fractions: ym
subtract ym from substances mass
vapor = 100 - mass
mass < 1g
temperatures, vapor, mass-progression, ...

(D) Algorithm to model the distillation curve of multicomponent liquids in a open system.

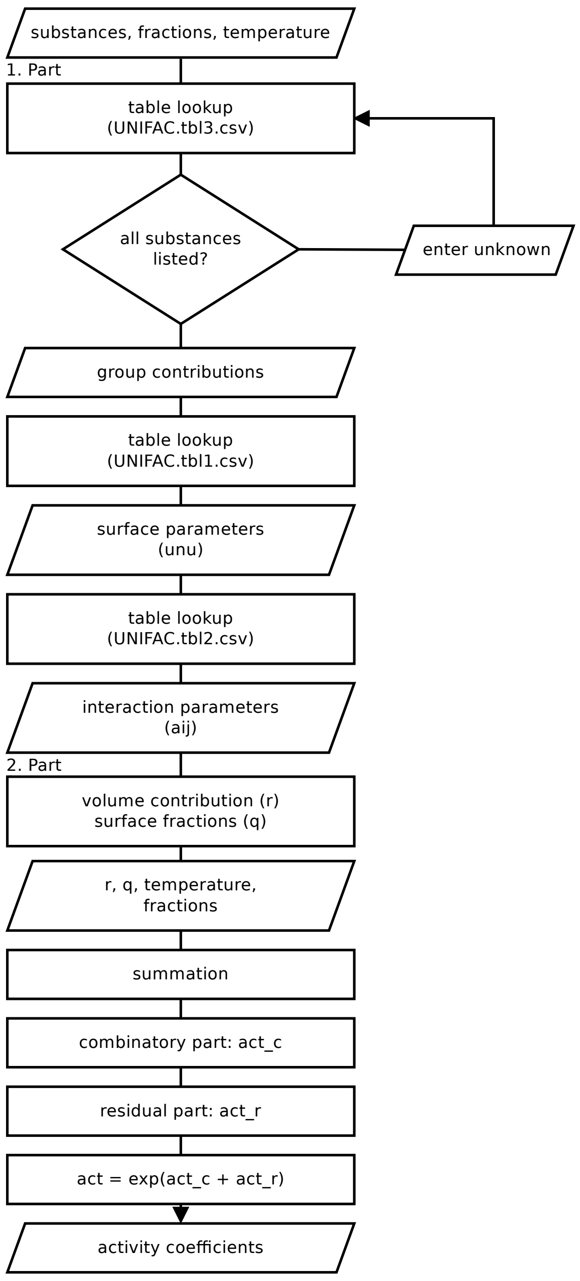

substances, fractions, temperature
1. Part
table lookup (UNIFAC.tbl3.csv)
all substances listed?
enter unknown
group contributions
table lookup (UNIFAC.tbl1.csv)
surface parameters (unu)
table lookup (UNIFAC.tbl2.csv)
interaction parameters (aij)
2. Part
volume contribution (r) surface fractions (q)
r, q, temperature, fractions
summation
combinatory part: act_c
residual part: act_r
act = exp(act_c + act_r)
activity coefficients

Figure A.3 Algorithm to calculate the activity coefficients using the UNIFAC model.

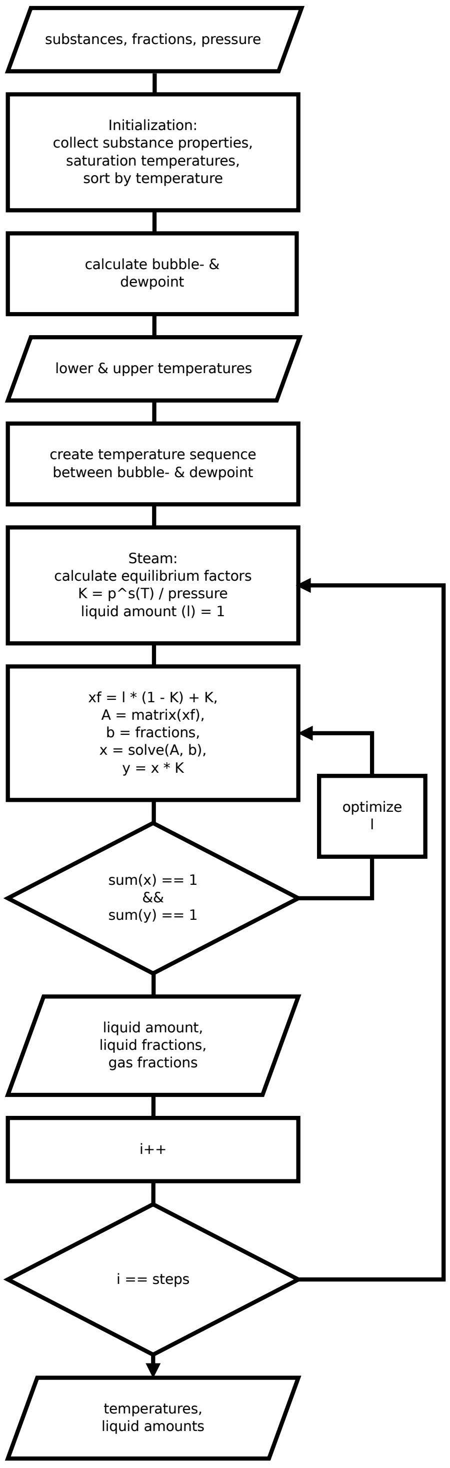

substances, fractions, pressure
Initialization: collect substance properties, saturation temperatures, sort by temperature
calculate bubble- & dewpoint
lower & upper temperatures
create temperature sequence between bubble- & dewpoint
Steam: calculate equilibrium factors K = p^s(T) / pressure liquid amount (l) = 1
xf = l * (1 - K) + K, A = matrix(xf), b = fractions, x = solve(A, b), y = x * K
optimize l
sum(x) == 1 && sum(y) == 1
liquid amount, liquid fractions, gas fractions
i++
i == steps
temperatures, liquid amounts

FIGURE A.4 Algorithm to model the distillation curve of multicomponent liquids in a closed system.

```
os.boilingline = function(substances=NULL,
                          fractions=NULL,
                          pressure=NULL,
                          mixname=NULL,
                          verbose=F){
  source('Substance.R');
  source('Antoine.P.R');
  source('Antoine.T.R');
  source('UNIFAC.R');
  source('UNIFAC.gen.R');
  source('SRK.R');
  Percent = function(x, digits = 0, format = "f", ...) {
    paste0(formatC(100 * x, format = format, digits = digits, ...), "%")
  }
  PsAntoine = function(temperature, ...) {# list with saturated pressures
    ps = Antoine.P(A, B, C, temperature);
    return(ps);
  }
  TsAntoine = function(pressure, ...){
    ts = Antoine.T(A, B, C, pressure);
    return(ts);
  }
  PsSRK = function(temperature, ...){
    ps = rep(0, nos);
    for(i in 1:nos){
      ps[i] = SRK(temperature=temperature, method = 'ps',
                  x=1, Ac=Ac[i], Pc=Pc[i], Tc=Tc[i])$pressure;
    }
    return(ps)
  }
  TsSRK = function(pressure, ...){
    ts = rep(0, nos);
    for(i in 1:nos){
      ts[i] = SRK(pressure=pressure, method = 'ts',
                  x=1, Ac=Ac[i], Pc=Pc[i], Tc=Tc[i])$temperature;
    }
    return(ts)
  }
  calc.bubble.T = function(...){
    ts = TsAntoine(pressure);
    temperature = sum(x * ts);
    c = 0;
    while(c < 100){
      ps = PsAntoine(temperature);
      p = sum(x * ps);
      y = abs(x * ps / pressure);
      rat = p / pressure;
      if(abs(1 - rat) < e){break;}
      else{
        c = c + 1;
        temp = 0.1 * temperature * (1 - rat) / rat;
        temperature = temperature + temp;
      }
    }
    if(c == 100){
      printf('Warning: Iteration limit (Antoine).
             Calculated pressure = %3.3f Pa.\n', p);
    }
    result = list(temperature=temperature,
                  y=y,
                  pressure=p);
  }
  calc.bubbleUNIFAC.T = function(...){
    ts = TsAntoine(pressure);
    temperature = sum(x * ts);
    c = 0;
    while(c < 100){
      act = UNIFAC(fractions=x, unu=unu, aij=aij, temperature=temperature);
      ps = PsAntoine(temperature);
      p = sum(act * x * ps);
      y = act * x * ps / pressure;
      rat = p / pressure;
      if (abs(1 - rat) < e){break;}
      else{
        c = c + 1;
        temp = 0.1 * temperature * (1 - rat) / rat;
        temperature = temperature + temp;
      }
    }
    if(c == 100){
      printf('Warning: Iteration limit (Antoine).
             Calculated pressure = %3.3f Pa.\n', p);
    }
    result = list(temperature=temperature,
                  y=abs(y))
    return(result);
  }
  calc.bubbleSRKU.T = function(...){
    ts = TsSRK(pressure);
    temperature = sum(x * ts);
    c = 0;
    while(c < 100){
      act = UNIFAC(fractions=x, unu=unu, aij=aij, temperature=temperature);
      ps = PsSRK(temperature);
      p = sum(act * x * ps);
      y = act * x * ps / p;
      rat = p / pressure;
      if (abs(1 - rat) < e){break;}
      else{
        c = c + 1;
        temp = 0.1 * temperature * (1 - rat) / rat;
        temperature = temperature + temp;
      }
    }
    if(c == 100) printf('Warning: Iteration limit (cubic).
                        Calculated pressure = %3.3f Pa.\n', p);
    result = list(temperature=temperature,
                  y=abs(y));
    return(result);
```

```
}
calc.bubbleSRK.T = function(...){
  ts = TsSRK(pressure);
  temperature = sum(x * ts);
  c = 0;
  while(c < 100){
    ps = PsSRK(temperature);
    p = sum(x * ps);
    y = x * ps / p;
    rat = p / pressure;
    if (abs(1 - rat) < e){break;}
    else{
      c = c + 1;
      temp = 0.1 * temperature * (1 - rat) / rat;
      temperature = temperature + temp;
    }
  }
  if(c == 100) printf('Warning: Iteration limit (cubic pure).
                      Calculated pressure = %3.3f Pa.\n', p);
  result = list(temperature=temperature,
                y=abs(y));
  return(result);
}
printf = function(...) cat(sprintf(...));
ptm = proc.time(); # count the calc. time
e = 1e-7;  # accuracy
nos = length(substances); # count number of substances
if(is.null(pressure)){
  stop('No pressure [Pa] specified. Abort.');
}
if (nos == 0){
  stop('No substance specified. Abort.');
}
if (length(fractions) != nos){ # check: N substances == N fractions
  stop('Number of components differ their mole fractions. Abort.');
}
# check for mole fraction consistent (sum(xi = 1)?)
if (abs(1 - sum(fractions)) > e){
  stop('Sum of mole fractions not equal to 1. Abort.');
}
if (is.null(mixname)){
  warning('No name specified, named "unknown"');
  mixname='unknown'
}
subtitle = c('p =', pressure, 'Pa'); # subtitle for plotting
Substances = rep(list(''), nos); # initiate the list of lists
for(i in 1:nos){ # collect substances properties
  Substances[[i]] = Substance(substances[i]); # get properties
  Substances[[i]]$MassFraction = fractions[i];
  Substances[[i]]$Mass = fractions[i] * 100;
  Substances[[i]]$Fraction = as.numeric(fractions[i] /
                             Substances[[i]]$MolarMass); # temporary value
  A = Substances[[i]]$Antoine$A[1];
  B = Substances[[i]]$Antoine$B[1];
  C = Substances[[i]]$Antoine$C[1];
  Tsat = Antoine.T(A, B, C, pressure); # calc boiling temperature
  Substances[[i]]$Tsat = Tsat;
  Ac = Substances[[i]]$Ac;
  Pc = Substances[[i]]$Pc;
  Tc = Substances[[i]]$Tc;
  TsatSRK = SRK(pressure, x=1, Ac=Ac, Pc=Pc, Tc=Tc, method = 'ts');
  Substances[[i]]$TsatSRK = TsatSRK$temperature;
}
abc=sapply(Substances, function(abc) abc$Antoine[1,1:3]);
A = as.numeric(abc[1,]);
B = as.numeric(abc[2,]);
C = as.numeric(abc[3,]);
names(Substances) = substances; # add the names, next reorder by Ts
Substances = Substances[order(-sapply(Substances, function(tsat) tsat$Tsat))];
m = sum(sapply(Substances, function(mass) mass$Fraction));
masses = sapply(Substances, function(mass) mass$Mass);
mass = sum(masses)
for(i in 1:nos){
  Substances[[i]]$MolarAmount = as.numeric(Substances[[i]]$Fraction / m);
  Substances[[i]]$Fraction = as.numeric(Substances[[i]]$MolarAmount);
} # Substances list initiation is complete
Suborg = Substances;
Ac = sapply(Substances, function(ac) ac$Ac);
Pc = sapply(Substances, function(pc) pc$Pc);
Tc = sapply(Substances, function(tc) tc$Tc);
substances = names(Substances)
u = UNIFAC.gen(substances); # load UNIFAC values
unu = u[[1]];
aij = u[[2]];
filename=paste(c(mixname,'-mass-progression.pdf'), collapse='');
pdf(file=filename);
par(mfrow=c(3,2), oma=c(4.5, 4, 4, 2.5), mar=rep(.1, 4), cex=0.7, las=1)
for(r in 1:5){ # start model loop
  fractions = as.numeric(sapply(Substances, function(frac) frac$Fraction));
  x = fractions;
  temperatures = c(); # initiate temperature table
  vapor = 0; # initiate vapor table
  massstack = masses; # initiate masses table
  printf('\n\nModel %d of 5\n\nProgress: ',r);
  while(round(mass, 0) > 0){ # simulation start from 100 to 0g substances
    if(r == 1){
      result = calc.bubble.T();
    }
    if(r == 2){
      result = calc.bubbleUNIFAC.T();
    }
    if(r == 3){
      result = SRK(pressure=pressure,x=x,Tc=Tc,Pc=Pc,Ac=Ac,method='bubbleT');
    }
    if(r == 4){
      result = calc.bubbleSRK.T();
    }
    if(r == 5){
      result = calc.bubbleSRKU.T();
    }
    temperature = result$temperature;
    y = result$y; # molar vapor fractions
    Mm = as.numeric(sapply(Substances, function(molmass) molmass$MolarMass));
    ym = abs(y * Mm / sum(y * Mm)); # convert molar vapor to mass fraction
    for(i in 1:nos){ # substract 1g vapor
      Substances[[i]]$Mass=round(abs(as.numeric(Substances[[i]]$Mass -
                                              ym[i])), 9);
    }
    masses = sapply(Substances, function(mass) mass$Mass); # get masses
    mass = sum(masses);
    xm = as.numeric(masses / mass); # calculate mass fractions
    x = xm / Mm / sum(xm / Mm); # convert mass to molar fractions
    for(i in 1:nos){
      Substances[[i]]$Fraction = x[i]; # store new molar fractions
    }
    if(mass < 10){ # last points controll
      temp = length(temperatures);
      if(temperature < temperatures[temp]){ # check for rising temperature
        warning('Model#', r, ' temperature from ', round(temperature,2),
                '%K to ',round(temperatures[temp],2), 'K corrected.');
        temperature = temperatures[temp]; # workaround
      }
      temp = length(vapor);
      if (round(vapor[temp], 0) == round(100 - mass, 0)){
        warning('Model#', r, ' vapor from ', round(100 - mass,0),
                '% to ', round(101 - mass, 0),'% corrected.');
        mass = mass - 1;
      }
    }
    temperatures = c(temperatures, temperature); # store temperature
    vapor = c(vapor, 100 - mass); # store vapor
    massstack = cbind(massstack,masses); # store masses
    printf('%s ', Percent((100 - mass) * 1e-2));
  } # simulation end
  if((r == 1) || (r == 2)){  # set last temperature
    temperatures = c(temperatures, Substances[[1]]$Tsat); # Antoine
  }
  else{
    temperatures = c(temperatures, Substances[[1]]$TsatSRK); # SRK
  }
  if(r == 1) BlAntoine = list(vapor=vapor,
                              temperature=temperatures);
  if(r == 2) BlUNIFAC = list(vapor=vapor,
                             temperature=temperatures);
  if(r == 3) BlSRK = list(vapor=vapor,
                          temperature=temperatures);
  if(r == 4) BlSRKpure = list(vapor=vapor,
                              temperature=temperatures);
  if(r == 5) BlSRKU = list(vapor=vapor,
                           temperature=temperatures);
  for(i in (nos - 1):1){# sum masses -> build the stack
    massstack[i,] = massstack[(i + 1),] + massstack[i,];
  }
  grays = gray.colors(nos);
  if((r %% 2) && (r != 5)){ # alternating plotting
```

```
    plot(x = temperatures, y = massstack[1,],
         type='n',
         ann=FALSE,
         xaxt='n');
    legend('topright',
           legend=paste(c('model#',r),collapse = ''),
           bty='n',
           cex=0.9);
  }
  if(!(r %% 2)){
    plot(x = temperatures, y = massstack[1,],
         type='n',
         ann=FALSE,
         xaxt='n',
         yaxt='n');
    legend('topright',
           legend=paste(c('model#',r),collapse = ''),
           bty='n',
           cex=0.9);
  }
  if(r == 5){
    plot(x = temperatures, y = massstack[1,], # empty bottom left plot
         type='n',
         ann=FALSE);
    legend('topleft',
           legend=substances,
           col=grays,
           lwd=2,
           bty='n', cex=0.8);
    legend('topright',
           legend=c('model#1 Antoine pure',
                    'model#2 Antoine UNIFAC',
                    'model#3 SRK',
                    'model#4 SRK pure',
                    'model#5 SRK UNIFAC'),
           bty='n',
           cex=0.8);
    axis(1, at=c(round(temperatures[1],0),round(Substances[[1]]$Tsat,0)));
    plot(x = temperatures, y = massstack[1,],
         type='n',
         ann=FALSE,
         yaxt='n');
    legend('topright',
           legend=paste(c('model#',r),collapse = ''),
           bty='n',
           cex=0.9);
    axis(1, at=c(round(temperatures[1],0),round(Substances[[1]]$Tsat,0)));
  }
  for(i in 1:nos){
    lines(x = temperatures, y = massstack[i,], col=grays[i]);
    xx = c(temperatures, rev(temperatures));
    yy = c(massstack[i,], rep(0,101));
    polygon(xx, yy, col=grays[i], border=NA);
  }
  Substances = Suborg; # reset substances list
```

```
    masses = sapply(Substances, function(mass) mass$Mass);
    mass = sum(masses);
  } # end model loop -> start post processing
  title(paste(c(mixname,'\nProgression Curves'), collapse=''),
        outer=TRUE, cex=0.8);
  mtext(paste(c('T [K]\n',subtitle), collapse=' '),
        1, 3, outer=TRUE,cex=0.8); # x-axis
  mtext('Substances progression [%]', 2, 3,
        outer=TRUE, las=0, cex=0.8); # y-axis
  dev.off();
  duration = proc.time() - ptm; # calculation time
  printf('\nCalculation takes %3.3f s\n', as.numeric(duration[1]));
  temp1 = length(BlAntoine$temperature);
  temp2 = length(BlUNIFAC$temperature);
  temp3 = length(BlSRK$temperature);
  temp4 = length(BlSRKpure$temperature);
  temp5 = length(BlSRKU$temperature);
  mint0 = min(c(BlAntoine$temperature[1],
                BlUNIFAC$temperature[1],
                BlSRK$temperature[1],
                BlSRKpure$temperature[1],
                BlSRKU$temperature[1]));
  maxt100 = max(c(BlAntoine$temperature[temp1],
                  BlUNIFAC$temperature[temp2],
                  BlSRK$temperature[temp3],
                  BlSRKpure$temperature[temp4],
                  BlSRKU$temperature[temp5]));
  pdf(file=paste(c(mixname,'-osbl.pdf'),collapse=''));
  plot(c(0,100), c(mint0, maxt100),
       main=paste(c(mixname,'\nDistillation Curves'),collapse = ''),
       xlab='Evaporated fraction [%]',
       ylab='T [K]',
       sub=paste(subtitle, collapse=' '),
       type='n');
  grid(NULL,NULL, lty = 6);
  legend('topleft', legend=c('Antoine pure',
                             'Antoine UNIFAC',
                             'SRK',
                             'SRK pure',
                             'SRK UNIFAC'),
         lty=c(1,2,6,4,8),
         col=c('black', 'red', 'darkblue', 'darkgreen', 'maroon'),
         lwd=2,
         bty='n', cex=0.9)
  lines(BlAntoine$vapor, BlAntoine$temperature, lwd=3);
  lines(BlUNIFAC$vapor, BlUNIFAC$temperature, lty=2,
        lwd=3, col='red');
  lines(BlSRK$vapor, BlSRK$temperature, lty=6, lwd=3,
        col='darkblue');
  lines(BlSRKpure$vapor, BlSRKpure$temperature, lty=8, lwd=3,
        col='darkgreen');
  lines(BlSRKU$vapor, BlSRKU$temperature, lty=4, lwd=3,
        col='maroon');
  dev.off();
  result = list(Antoine=BlAntoine,
                UNIFAC=B1UNIFAC,
                SRK=B1SRK,
                SRKpure=B1SRKpure,
                SRKUNIFAC=B1SRKU);
  return(result);
}
```

Code A.1 Distillation curve simulation using 5 models for bubblepoint temperature calculations with automated plotting of the distillation curves and the substances progression diagrams.

```
Substance = function(name){
  AntoineVals = read.csv('Substances.Antoine-tbl.csv', sep=',');
  Antoine = AntoineVals[grep(paste('^', name, sep='', ' '),
                           AntoineVals$Substance),];
  if(nrow(Antoine) == 0){
    Antoine = AntoineVals[grep(paste('^', name, sep='', '$'),
                             AntoineVals$Substance),];
  }
  if(nrow(Antoine) == 0){
    stop('\nAntoine coefficients for ', name,' not found.');
  }
  Antoine$Substance = NULL; # remove name field
  criticalVals = read.csv('Substances.critical-tbl.csv', sep=',');
  criticals = criticalVals[grep(paste('^', name, sep='', ' '),
                              criticalVals$Substance), ];
  if(nrow(criticals) == 0){
    criticals = criticalVals[grep(paste('', name),
                                criticalVals$Substance), ];
  }
  if(nrow(criticals) == 0){
    stop('\nCritical values for ', name,' not found.');
  }# load element masses file to calc. the molar mass
  EleMass = read.csv('ElementMasses.csv', sep=',');
  # capture  and split formula
  string = tail(unlist(strsplit(gsub('',' ', criticals$Formula), ' ')), -1);
  elements = string[1]; # take elements
  mass = NULL;
  factors = NULL;
  # 2. start from index two (first is upper letter), check for upper letter
  i = 2
  while (i <= length(string)){
    if(grepl('[A-Z]', string[i])){ # is upper?
      elements = append(elements, string[i]);
      if(!grepl('[0-9]', string[i - 1])){
        factors = append(factors, 1); # factor = 1
      }
      if(i == length(string)){
        factors = append(factors, 1);
      }
    }
    else if(grepl('[a-z]', string[i])){ # is lower?
      elements[i - 1] = paste(elements[i - 1], string[i], sep='');
    }
    else{ # is number!
```

```
      tmp = '';
      while(grepl('[0-9]', string[i])){
        tmp = paste(tmp, string[i], sep='')
        i = i + 1;
      }
      i = i - 1;
      factors = append(factors, as.numeric(tmp));
    }
    i = i + 1;
  }
  for (i in 1:length(elements)){
    mass = append(mass, EleMass[grep(elements[i], EleMass$Element), ]$Mass);
  }
  M = sum(factors * mass);
  result = list(Formula = as.character(criticals$Formula),
                Antoine = Antoine,
                Ac = criticals$Ac,
                Pc = criticals$Pc,
                Tc = criticals$Tc,
                MolarMass = M);
  return(result)
}
```

Code A.2 Substance function to return substance properties from csv tables and to calculate the molare mass from the chemical formula.

```
UNIFAC.gen = function(Substances=NULL,
                      verbose=FALSE){
  # tool to sum the UNIFAC parameters and to generate the surface and
  # the interaction parameters matrizes
  #
  # input: vector with substances or NULL
  #
  # return: list with 2 matrices: $unu = surface parameters
  #                               $aij = interaction parameters
  # file: UNIFAC-tbl3.csv contains example substances with group contributions
  #       all new substances are stored in it
  # define functions:
  # 'printf' Formated printing, example: printf('Something equals %d',1)
  printf = function(...) cat(sprintf(...));
  # 'get.aij'
  get.aij = function(i,j){# read UNIFAC interaction paramters from csv file
    atbl = read.csv('UNIFAC-tbl2.csv',sep=',');
    # search insid the UNIFAC interaction parameter table
    for (k in 1:nrow(atbl)){
      if ((atbl[k,1] == i) && (atbl[k,2]== j)){
        return(atbl[k,3]);
        break;
      }
      else if (k == nrow(atbl)){
        printf('Combination of group %d and group %d not found.',i,j);
        return(0);
      }
    }
  }
```

```
# 'get.UNIFAC' Get UNIFAC paramters from table get.unifac('formula')
# -> k, Group, Rk, Qk
get.UNIFAC = function(formula){
  utbl = read.csv('UNIFAC-tbl1.csv',sep=','); # read UNIFAC surface paramters
  # search the UNIFAC table for functional group by formula and
  # return the line or 0
  # formula = case sensitive string e.g. H2O, OH, CH2=CH, Morpholine
  line = utbl[grep(paste('^',formula,sep='',' '),utbl$Group),];
  if (nrow(line)==0){
    line = utbl[grep(paste('^',formula,sep='','$'),utbl$Group),];
  }
  if (nrow(line)==0){
    stop('\nGroup ',formula,' not found.\n');
  }
  return(line);
}
# 'get.sub' Read the functional group contribution, by name
get.sub = function(name){
  subtbl = read.csv('UNIFAC-tbl3.csv',sep=',');
  sub = subtbl[grep(paste('^',name,sep='',' '),subtbl$Name),];
  if (nrow(sub)==0){
    sub = subtbl[grep(paste('^',name,sep='','$'),subtbl$Name),];
  }
  if (nrow(sub)==0){
    printf('\nSubstance "%s" not found.\n',name);
  }
  else{
    return(sub)
  }
}
# define variables
sn = Substances;
k = c(); # subgroup number
if(verbose){
    printf('Welcome to the "UNIFAC MATRIZES GENERATOR"\n\n');
    printf('All known substances are stored in file "UNIFAC-tbl3.csv",\n');
    printf('new ones can be defined by name and group contribution.');
}
if(is.null(sn)){
  printf('Generate the UNIFAC Matrix in two steps:\n\n');
  printf('1.Type the names of all substances.\n')
  printf('2. Type the group contribution for unknown substances.\n\n');
  printf('Type the names of all substances (case sensitive)!');
  # 1. read the names from user input
  sn = readline(); # substances names
  sn = strsplit(sn,',')[[1]]; # split names at seperator ','
}
if(verbose){
  printf('The following substances are in use:\t');
  printf('%s\t', sn);
}
nos = length(sn); # number of substances
tmp1 = data.frame();
for (i in 1:nos){
  tmp2 = get.sub(sn[i]); # check for known substances in file 'UNIFAC-tbl3.csv'
```

```
    if(!is.null(tmp2)){
      tmp1 = rbind(tmp1,tmp2); # one substance per row
    }
    else{ # define substance
      tmp2 = data.frame(Name=sn[i], Groups=NA)[, ]
      printf('\nEnter the group contribution for %s:\t[1*CH3 1*CH2 1*OH]',sn[i]);
      tmp2[2] = readline();
      tmp1 = rbind(tmp1,tmp2);
      write.table(tmp2,file='UNIFAC-tbl3.csv',sep=',',
                  row.names=F,col.names=F,append=T);
    }
  } # transform listed functional groups in matrix
  groups = strsplit(paste(tmp1[,2]),'\\t'); # 1st split -> groups per substances
  tmp1 = data.frame();
  for (i in 1:length(groups)){ # from 1st to last substance
    groups[i] = strsplit(groups[[i]],'\\ '); # 2nd split between groups
    nu = rep(0,1);
    gr = rep(NA,1);
    for (j in 1:length(groups[[i]])){ # from 1st to last group per substance
      tmp1 = strsplit(groups[[i]][j],'\\*')[[1]]; # 3rd split -> amount of groups
      nu[j] = tmp1[1];
      gr[j] = tmp1[2];
    }
    if (i == 1){ # 1st substance
      num = cbind(gr,nu); # matrix of group and amount per group
    }
    else{
      tmp2 = rep(0,nrow(num)); # zero column
      num = cbind(num,tmp2); # new substance -> new column
      for(l in 1:length(gr)){ # from 1st to last group per substance
        pos = grep(paste('^',gr[l],sep='','$'),num[,1]); # group exists?
        if (length(pos) > 0){
          num[pos,i+1] = nu[l]; # hit -> set value
        }
        else{ # new group
          tmp2 = rep(0,ncol(num)-2); # repeat zeros for the new line
          tmp3 = c(gr[l],tmp2,nu[l]); # vector with group, 0, amount of group
          num = rbind(num,tmp3); # add the new line
        }
      }
    }
  }
  nog = nrow(num); # number of groups
  tmp1 = data.frame();
  for (i in 1:nog){
    tmp2 = get.UNIFAC(num[i,1]); # get the UNIFAC parameters
    tmp1 = rbind(tmp1,tmp2);
  }
  if((nrow(tmp1) == nog) || (length(tmp1) == nog) ){
    unu = cbind(tmp1,num[,-1]);
  }# data frame with UNIFAC values and amount per substance
  else{
    stop('Not all groups have been identified.
         May check for group with get.UNIFAC(group)')
  }
  for(i in 5:ncol(unu)) unu[,i] = as.numeric(as.character(unu[,i]));
  colnames(unu)[5:ncol(unu)] = sn;
  unu = data.frame(unu, row.names = NULL);
  # 3. Generate the interaction parameters matrix for the subgroups
  aij = matrix(0,nog,nog);
  for (i in 1:nog){
    for (j in 1:nog){
      aij[j,i] = get.aij(unu[j,2],unu[i,2]);
    }
  }
  if(verbose){
    printf('\n\nThe UNIFAC matrix is ready for use.\n\n');
    print(unu);
    printf('\n\nThe interaction parameters matrix is ready for use.\n\n');
    print(aij);
  }
  result = list(unu=unu,
                aij=aij);
  return(result);
}
```

Code A.3 UNIFAC matrices generator.

```
UNIFAC = function(fractions=c(),
                  unu=c(),
                  aij=c(),
                  temperature=NULL){
  # function to calculate the activity coefficients
  # with UNIFAC
  # arguments:
  # x: vector with molar fractions
  # unu: UNIFAC surface parameters and component matrix
  # aij: UNIFAC interaction parameters matrix [k,k]
  # temperature: [K]
  #
  # return:
  # activity coefficients
  nos = length(fractions);# count number of substances
  e = 1e-7;
  # check for mole fraction consistent (sum(xi = 1)?)
  if (abs(sum(fractions) - 1) > e){
    stop('Sum of mole fractions not equal to 1. Abort.')
  }
  x = fractions;
  # 1. create UNIFAC matrices
  r = rep(0, nos);  # r volume contribution, c(r[1], ... ,r[n])
  for (i in 1:nos){
    r[i] = sum(unu[, 3] * unu[, 4 + i]);
  }
  q = rep(0, nos);  # q surface fractions, c(q[1], ... , q[n])
  for (i in 1:nos){
    q[i] = sum(unu[, 4]*unu[, 4 + i]);
  }
  nog = nrow(unu); # number of groups
  eij = matrix(0, nog, nos); # surface fractions for
  for (i in 1:nos){ # each subgroup in each molecule
    for (j in 1:nog){
      eij[j, i] = unu[j, 4 + i] * unu[j, 4] / q[i];
    }
  }
  # 1. summation
  # determine the overall surface area fraction
  # for each subgroup in the mixture
  S = rep(0, nog);
  for (i in 1:nog){
    S[i] = sum(x * q * eij[i, ]) / sum(x * q);
  }
  tau = exp(-aij / temperature);
  beta = matrix(0, nos, nog);
  for (i in 1:nos){
    for (j in 1:nog){
      beta[i, j] = sum(eij[, i] * tau[, j]);
    }
  }
  s = rep(0, nog);
  for (i in 1:nog){
    s[i] = sum(S * tau[, i]);
  }
  J = rep(0, nos)
  for (i in 1:nos){
    J[i] = r[i] / sum(x * r);
  }
  L = rep(0, nos);
  for (i in 1:nos){
    L[i] = q[i] / sum(x * q);
  } # 2. combinatory part
  actc = rep(0, nos);
  for (i in 1:nos){
    actc[i] = 1 - J[i] + log(J[i]) - 5 * q[i] *
      (1 - J[i]/ L[i] + log(J[i] / L[i]));
  } # 3. residual part
  actr = rep(0,nos);
  for (i in 1:nos){
    actr[i] = q[i] * (1- sum(S * beta[i,] / s -
                                eij[,i] * log(beta[i,] / s)));
  } # 4. sum the parts
  act = exp((actc + actr));
  return(act);
}
```

Code A.4 UNIFAC calculation.

```
FACE# substances and mass fractions

FACE1=c('n-dodecane','n-octadecane','2,2,3,5-tetramethylhexane',
'2,2,4,4,6,8,8-heptamethylnonane','cyclohexane','decalin','tetralin',
'naphthalene','m-cymene','n-pentylbenzene','n-heptylbenzene')
Fractions1=c(0.162,0.101,0.26,0.1,0.03,0.12,0.008,0.015,0.152,0.042,0.01)

FACE2=c('n-dodecane','n-octadecane','n-eicosane','n-heneicosane',
'2,2,3,5-tetramethylhexane','2,2,4,4,6,8,8-heptamethylnonane','cyclohexane',
'decalin','m-xylene','tetralin','naphthalene','m-cymene','n-pentylbenzene',
'n-heptylbenzene')
Fractions2=c(0.09,0.02,0.05,0.05,0.367,0.07,0.03,0.12,0.01,0.007,0.012,
0.13,0.033,0.011)

FACE3=c('n-hexadecane','n-octadecane','2,2,3,5-tetramethylhexane',
'2,2,4,4,6,8,8-heptamethylnonane','cyclohexane','decalin','m-xylene',
'tetralin','naphthalene','m-cymene','n-pentylbenzene','n-heptylbenzene')
Fractions3=c(0.08,0.040,0.095,0.1,0.05,0.2,0.020,0.075,0.058,0.18,0.052,0.050)

FACE4=c('n-tridecane','n-hexadecane','n-octadecane','n-heneicosane',
'2,2,3,5-tetramethylhexane','2,2,4,4,6,8,8-heptamethylnonane',
'cyclohexane','decalin','tetralin','phenanthrene','m-cymene','n-heptylbenzene')
Fractions4=c(0.050,0.05,0.07,0.08,0.051,0.11,0.06,0.16,0.006,0.017,0.246,0.100)

FACE5=c('n-decane','n-tridecane','n-tetradecane','n-hexadecane','n-octadecane',
'cyclohexane','decalin','m-xylene','tetralin','naphthalene',
'm-cymene','n-pentylbenzene','n-hexylbenzene','n-heptylbenzene')
Fractions5=c(0.1,0.166,0.13,0.06,0.071,0.02,0.27,0.007,0.029,0.01,0.034,
0.032,0.021,0.05)

FACE6=c('n-tetradecane','n-hexadecane','n-octadecane','n-heneicosane',
'cyclohexane','decalin','tetralin','phenanthrene','n-heptylbenzene')
Fractions6=c(0.099,0.198,0.091,0.08,0.01,0.314,0.06,0.05,0.098)

FACE7=c('n-decane','n-dodecane','n-tetradecane','n-hexadecane','n-octadecane',
'cyclohexane','decalin','naphthalene','m-cymene','n-pentylbenzene',
'n-hexylbenzene','n-heptylbenzene')
Fractions7=c(0.161,0.09,0.1,0.102,0.041,0.02,0.14,0.024,0.06,0.062,0.040,0.16)

FACE8=c('n-dodecane','n-tetradecane','n-hexadecane','n-octadecane',
'n-heneicosane','cyclohexane','decalin','tetralin','naphthalene','phenanthrene',
'n-hexylbenzene','n-heptylbenzene')
Fractions8=c(0.027,0.169,0.122,0.051,0.046,0.011,0.162,0.04,0.111,0.169,
0.022,0.07)

FACE9=c('n-decane','n-tetradecane','n-hexadecane','n-eicosane','cyclohexane',
'decalin','m-xylene','tetralin','phenanthrene','m-cymene','n-pentylbenzene',
'n-heptylbenzene')
Fractions9=c(0.050,0.05,0.181,0.061,0.08,0.26,0.01,0.069,0.055,0.02,0.04,0.124)
```

Bibliography

[1] COORDINATING RESEARCH COUNCIL. Fuels for advanced combustion engines working group, accessed on Sep 1, 2016. URL `https://crcao.org/publications/advancedVehiclesFuelsLubricants/FACE/index.html`.

[2] J.C.G. Andrae. Development of a detailed kinetic model for gasoline surrogate fuels. *Fuel*, 87(10-11):2013–2022, aug 2008. doi: 10.1016/j.fuel.2007.09.010.

[3] G. Saville. ACENTRIC FACTOR. In *A-to-Z Guide to Thermodynamics, Heat and Mass Transfer, and Fluids Engineering*. Begell House, 2006. doi: 10.1615/atoz.a.acentric_factor.

[4] E. Clapeyron. Mémoire sur la puissance motrice de la chaleur. *Journal de l'École Polytechnique*, 1834.

[5] J. Gmehling and B. Kolbe. *Thermodynamik*. Lehrbuchreihe Chemieingenieurwesen, Verfahrenstechnik. VCH Verlagsgesellchaft mbH, 2 edition, 1992. ISBN 3527285474.

[6] Johannes Diderik Van der Waals. Over de continuiteit van den gas- en vloeistoftoestand, 1873.

[7] J. Gmehling. Potential of group contribution methods for the prediction of phase equilibria and excess properties of complex mixtures. *Pure and Applied Chemistry*, 75(7):875–888, 2003. ISSN 0033-4545. doi: 10.1351/pac200375070875. URL `http://iupac.org/publications/pac/75/7/0875/`.

[8] Otto. Redlich and J. N. S. Kwong. On the thermodynamics of solutions. v. an equation of state. fugacities of gaseous solutions. *Chemical Reviews*, 44(1):233–244, feb 1949. doi: 10.1021/cr60137a013.

[9] Giorgio Soave. Equilibrium constants from a modified redlich-kwong equation of state. *Chemical Engineering Science*, 27(6):1197–1203, jun 1972. doi: 10.1016/0009-2509(72)80096-4.

[10] 2008 Patrick J. Barrie, University of Cambridge. Javascript programs for solving cubic equations of state, accessed on Sep 1, 2016. URL `http://people.ds.cam.ac.uk/pjb10/thermo/`.

[11] Ding-Yu Peng and Donald B. Robinson. A new two-constant equation of state. *Ind. Eng. Chem. Fund.*, 15(1):59–64, feb 1976. doi: 10.1021/i160057a011.

[12] K.A.M Gasem, W Gao, Z Pan, and R.L Robinson. A modified temperature dependence for the peng–robinson equation of state. *Fluid Phase Equilibria*, 181(1-2): 113–125, may 2001. doi: 10.1016/s0378-3812(01)00488-5.

[13] W.H. Press. *Numerical Recipes in FORTRAN Example Book: The Art of Scientific Computing*. Number Bd. 1-2 in Fortran numerical recipes. Cambridge University Press, 1992. ISBN 9780521437219.

[14] H Knapp and Dechema. *Vapor-liquid Equilibria for Mixtures of Low-boiling Substances: without special title*. Chemistry data series. DECHEMA, 1989.

[15] Aage Fredenslund, Russell L. Jones, and John M. Prausnitz. Group-contribution estimation of activity coefficients in nonideal liquid mixtures. *AIChE Journal*, 21 (6):1086–1099, 1975. ISSN 1547-5905. doi: 10.1002/aic.690210607.

[16] The R Foundation. The r project for statistical computing, accessed on Sep 1, 2016. URL `https://www.r-project.org`.

[17] Simon L. Clegg and Anthony S. Wexler Peter Brimblecombe. UNIFAC structural groups and parameters, accessed on Sep 1, 2016. URL `http://www.aim.env.uea.ac.uk/aim/info/UNIFACgroups.html`.

[18] Milo D. Koretsky. *Engineering and Chemical Thermodynamics*. Wiley, 2004.

[19] R Documentation. Lists – generic and dotted pairs, accessed on Sep 1, 2016. URL `https://stat.ethz.ch/R-manual/R-devel/library/base/html/list.html`.

[20] The R Foundation. General-purpose optimization, accessed on Sep 1, 2016. URL `https://stat.ethz.ch/R-manual/R-devel/library/stats/html/optim.html`.