Diplomarbeit

Development of a Robust Algorithm for the Numerical Description of Gas Permeation Systems

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs unter der Leitung von

Ass.Prof. Dipl.-Ing. Dr. Michael Harasek

und der Betreuung von

Projektass. Dipl.-Ing. Werner Liemberger

E166
Institut für Verfahrenstechnik, Umwelttechnik und Technische Biowissenschaften

eingereicht an der Technischen Universität Wien
Fakultät für Maschinenwesen und Betriebswissenschaften

von

Daniel Halmschlager
Matrikelnummer 0926984

Wien, November 2017

Abstract

Within the project HylyPure®, a process for the separation of hydrogen from a mixture with methane has been developed at the Institute of Chemical, Environmental, and Biological Engineering at TU Wien. This enables the usage of the natural gas grid for the co-transport of hydrogen. The HylyPure® process includes gas permeation and pressure swing adsorption steps. A test facility for both steps was developed at TU Wien. For further optimization of the process, a solution to efficiently simulate large parameter sets and complex flow sheets is required. An algorithm for the numerical solution of gas permeation was developed and published at the institute some years ago, but does not fulfill all the current needs.

This thesis presents vectorized gas permeation (VECGP), an enhanced algorithm based on the finite difference method. VECGP not only ensures stable calculations for a wide range of gas permeation cases, but also significantly improves performance and is able to solve multi component systems and configurations with high recovery rates efficiently.

The new object oriented and modular implementation in MATLAB is designed to allow easy expansion and re-usage of code while simultaneously improving user-friendliness. Besides evaluating the mass transfer through membrane modules, VECGP also provides pressure drop and energy balance calculation and allows inclusion of real gas properties based on the best currently available models.

The core feature set is well tested and validated against the existing algorithm and has successfully been used to simulate complex multi component processes without any stability issues.

Kurzfassung

Im Rahmen des Projektes HylyPure® wurde am Institut für Verfahrenstechnik, Umwelttechnik und Technische Biowissenschaften ein Verfahren entwickelt um Wasserstoff aus einer Mischung mit Erdgas zurückzugewinnen und so das Erdgasnetz zum Wasserstoffkotransport zu nutzen. Der HylyPure®-Prozess besteht aus Gaspermeations- und Druckwechseladsorptionsschritten. Eine Versuchsanlage für beide Teile wurde and der TU Wien entwickelt. Zur weiteren Optimierung des Prozesses wird eine Lösung zur effizienten Simulation großer Parameterfelder und komplexer Fließschemata erforderlich. Ein Algorithmus zur Simulation von Gaspermeation wurde vor einigen Jahren am Insistut entwickelt und publiziert, welcher jedoch nicht alle derzeitigen Anforderungen erfüllt.

Diese Diplomarbeit stellt vecgp vor, einen aktualisierten und verbesserten Algorithmus basierend auf der Finite-Differenzen-Methode. Vecgp ermöglicht nicht nur stabile Berechnungen für eine große Anzahl an möglichen Prozesskonfigurationen, sondern ist auch deutlich schneller und kann Multikomponentenfälle, sowie Membranmodule mit hohen Rückgewinnungsraten effizient simulieren.

Die neue objektorientierte und modulare Implementierung in MATLAB stellt sicher, dass der Code einfach erweitert und wiederverwendet werden kann. Neben der Simulation des Stofftransportes in Membranmodulen erlaubt vecgp auch Druckverlust- und Energiebilanzberechnungen und ermöglicht die Verwendung von Realgasdaten basierend auf den genauesten derzeit verfügbaren Modellen.

Die Kernfunktionen sind ausgiebig getestet und gegen den existierenden Algorithmus validiert. Außerdem wurde vecgp auch schon erfolgreich zur Simulation von komplexen mehrstufigen Prozessen eingesetzt.

Acknowledgement

I wish to thank Werner Liemberger for coming up with the idea and making this thesis possible. Thank you for challenging and pushing my ideas, for providing guidance when I was stuck, and for making sure that the code is well-tested (yes, I am talking about the 100 000-something cases that kept the servers busy).

Additionally, I would like to thank Michael Harasek for giving me the freedom to define the focus of the work myself while making sure that it is up to scientific standards and has potential for impact.

Furthermore, I want to express my gratitude to the colleagues of the research group Computational Fluid Dynamics for providing me with an inspiring place to work and great conversations while the code was running – or while it was producing errors.

A large thank you is also due for my family. To my parents Susanne and Günter for supporting me on every step despite all distractions and delays, and to my sister Verena who impressively showed me that a great master thesis does not need to take one and a half years.

Last but not least to Eva. Thank you for all the emotional support, for keeping me grounded, and for enduring my weird ways in (occasionally) stressful times.

Contents

1 Introduction

1.1 Motivation and Problem Statement

A critical challenge the world is facing today is global warming, mainly driven by human-caused green house gas emissions. One way to mitigate (or at least slow down) this situation is to gradually shift energy sources away from fossil fuels towards renewables. The generation of energy from renewable sources like wind and solar power typically fluctuates, which makes balancing supply and demand increasingly difficult. Amongst various other possible ways, excess electric energy can be stored with the power-to-gas concept by producing hydrogen through electrolysis of water.

In order to efficiently transport hydrogen, the HylyPure® process concept was developed at the Institute of Chemical, Environmental, and Biological Engineering at TU Wien in cooperation with OMV AG. The core of the concept is injecting hydrogen into the existing natural gas grid at allowed concentrations (4 % (v/v) in Austria, ÖVGW 2001, and 5 % (v/v) in Germany, DVGW 2013) and transporting it with the stream. To efficiently extract the hydrogen, a hybrid process based on membrane separation and pressure swing adsorption was designed. Additionally, a test facility was developed that can operate with gas mixtures provided by OMV's Auersthal site (Liemberger et al. 2016; BMVIT 2015). Based on this test facility, extensive experimental trials were conducted that showed promising results (Liemberger et al. 2017) . Additionally to the experiments, it was an objective to be able to accurately simulate the process in order to allow easy parameter variations and interconnection of multiple stages.

For the simulation of separation of gas mixtures with membranes, a solver exists within the research group. This was developed by Makaruk and Harasek (2009), focusing on a slightly different process. While this code works well for easy cases, there are issues with stability in more complex cases (especially multiple components) and at high recovery rates of single components. Additionally, the implementation is not designed to allow simple interconnectivity with simulation code for the adsorption stage, as no externally usable interfaces are provided.

While the focus of this work is on the membrane separation step of the HylyPure® process, membranes are widely used for gas separation processes in various renewable energy applications (Makaruk et al. 2010; Scholz et al. 2013b; Abels et al. 2013; Chen et al. 2015).

Simulation of gas permeation in general is a recently intensively researched topic. Some authors use a very general numerical approach (Ahmad et al. 2015; Binns et al. 2016; Feichtinger et al. 2014; Lock et al. 2014; Sharifian et al. 2016), while others apply computational fluid dynamics (CFD) methods to membrane separation processes (Haddadi Sisakht et al. 2016; Ahmad et al. 2015; Alkhamis et al. 2013; Alkhamis et al. 2015). Additional research is focused on very specific separation applications (often biogas) (Ahsan and Hussain 2015; T. G. Lassmann 2015; T. Lassmann et al. 2016; Lock et al. 2015a; Lock et al. 2015b; Ohenoja and Sorsa 2015; Scholz et al. 2015). And there are also publications addressing specific membrane shapes or aspects of the transport process (Hosseini et al. 2016a; P. K. Kundu et al. 2013; Magnanelli et al. 2016; M. Szwast and Z. Szwast 2015; Xu and Agrawal 1996b).

Overall, it seems a well-performing and flexible simulation algorithm might be of high interest in many fields.

1.2 Objectives of this Work

The main objective of this work was to develop a new algorithm (and a reference implementation), that addresses the shortcomings of the solver by Makaruk and Harasek (2009). Given the circumstances, the most important factor to consider was calculation stability for a wide range of cases, but in general the goals were defined as:

- Algorithmic improvements to address complex cases
- Implementation with focus on modularity and performance
- Handling of multiple components, sweep gas streams, and variable permeate outlet locations
- Capability to simulate complex multi-stage processes
- Development and inclusion of new features (like pressure drop and energy balance calculation)

The aim of this thesis is to describe the developed algorithm. Its intention is to thoroughly show the underlying mathematics and assumptions as well as some key aspects of the implementation. While there are some usage examples and test cases given in later chapters, this shall not be considered as a full documentation of the implementation.

2 Basics of Gas Permeation

2.1 Principles of Membrane Separation

The following chapter focuses on the basics of gas permeation that are required to derive the model used in chapter 3. A full introduction into membrane separation would be out of scope for this thesis, but good coverage of these topics is available in standard literature such as, e.g., Baker (2012) or Melin and Rautenbach (2007).

In general, membranes are layers that are able to separate substances by allowing some components to permeate easier than others. For a transport process to occur through a membrane, some kind of driving force is required. These driving forces can be gradients in (partial) pressure, concentration, electrostatic charge, or, more generalized, chemical potential. Depending on the application and material, membranes can be dense or microporous. Electrically or chemically active membranes are also possible (Baker 2012).

Various different types of membrane separation can be distinguished, depending on the nature of the driving force and the type of substance to separate. Commonly used examples besides gas permeation are reverse osmosis, ultrafiltration, microfiltration, pervaporation, and electrodialysis.

Membrane materials can be organic or inorganic and are a heavily researched topic (Baker and Low 2014). For commercial membrane manufacturers, the exact material compositions are typically strictly guarded trade secrets.

The final categorization criterion for membrane modules is the type of module construction. While many different shapes are available, a primary distinction can be made between flat and tubular modules (Melin and Rautenbach 2007).

Going forward, the focus of this work will only be on the type of module that is modeled in the subsequent chapters, namely a hollow fiber module for gas permeation based on a dense, organic membrane. Typical schematics of such a hollow fiber module are shown in Fig. 2.1.

2.2 Mathematical Description

The transport process in dense membranes is typically described by the solution-diffusion model (Baker 2012). Within this model it is assumed that when a component permeates through the membrane, it is first dissolved in the membrane

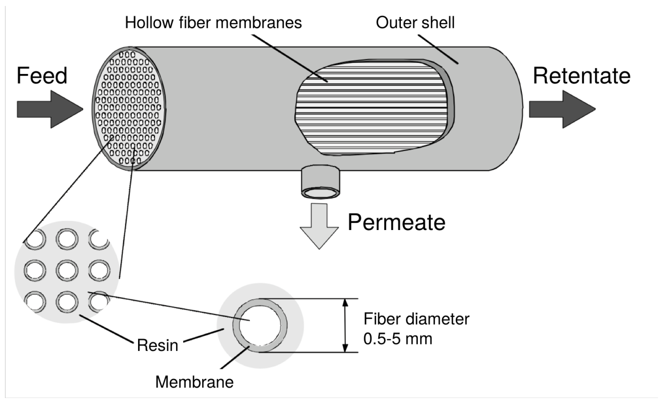


Figure 2.1: Scheme of a Hollow Fiber Module (adapted from Melin and Rautenbach 2007, p. 163)

material and then transported through the membrane via diffusion. On the other side of the membrane the component desorbs then again.

The diffusive transport is generally described by Fick's law

$$\dot{Q}''_j = -D_j \frac{dc_j}{dz} \tag{2.1}$$

where $\dot{Q}''_j$ is the diffusive flux of component j, D_j is the diffusion coefficient, c_j is the concentration of component j in the membrane, and z is the coordinate through the membrane.

Integrating over the thickness of the membrane, one obtains

$$\dot{Q}''_j = D_j \frac{c_{F,j} - c_{P,j}}{d_f} \tag{2.2}$$

where $c_{F,j}$ and $c_{P,j}$ denote the concentration of the respective component in the membrane.

At the interface between the membrane and the fluid, there has to be an equilibrium of component j in the fluid and in the membrane. Choosing a sorption coefficient K in the correct units, the concentration in the membrane can be related to the partial pressure in the fluid:

$$c_{F,j} = K_j \cdot p_{F,j} \tag{2.3}$$

$$c_{P,j} = K_j \cdot p_{P,j} \tag{2.4}$$

The partial pressures $p_{F,j}$ and $p_{P,j}$ follow Dalton's law for ideal gases:

$$p_{F,j} = x_{F,j}\, p_F \qquad\qquad p_{P,j} = x_{P,j}\, p_P \tag{2.5}$$

It is also possible to handle non-ideal cases by introducing fugacity coefficients φ:

$$p_{F,j} = \varphi_{F,j} x_{F,j} p_F \qquad\qquad p_{P,j} = \varphi_{P,j} x_{P,j} p_P \tag{2.6}$$

Substituting the concentrations in (2.2) with the terms from (2.3) and (2.4), one obtains:

$$\dot{Q}''_j = D_j \frac{K_j \cdot p_{F,j} \cdot \varphi_{F,j} - K_j \cdot p_{P,j} \cdot \varphi_{P,j}}{d_f} = D_j K_j \frac{p_{F,j} \cdot \varphi_{F,j} - p_{P,j} \cdot \varphi_{P,j}}{d_f} \tag{2.7}$$

or in the ideal case

$$\dot{Q}''_j = D_j K_j \frac{p_{F,j} - p_{P,j}}{d_f} \tag{2.8}$$

The diffusion coefficient D, the sorption coefficient K, and the membrane thickness d_f can be combined into a single value, the membrane permeance Π:

$$\Pi_j = \frac{D_j K_j}{d_f} \tag{2.9}$$

Thus the transmembrane flux can be described as

$$\dot{Q}''_j = \Pi_j \left(p_{F,j} \cdot \varphi_{F,j} - p_{P,j} \cdot \varphi_{P,j} \right) \tag{2.10}$$

or in the ideal case

$$\dot{Q}''_j = \Pi_j \left(p_{F,j} - p_{P,j} \right) \tag{2.11}$$

Equation (2.10) describes the local transmembrane flux based on the local partial pressures. In order to apply this local equation to full membrane module, some additional steps are required. This is shown in detail in chapter 3.

2.3 Concentration Polarization

Concentration polarization describes the effect that a concentration gradient forms in the fluid phases (Baker 2012). This means that the fluid that interacts with the membrane does not necessarily have the same composition as the fluid in the bulk, as is shown in Fig. 2.2.

The effect occurs when the transport of components within the fluid is slower or not significantly faster than the mass transport through the membrane and therefore a boundary layer is formed. Concentration polarization affects separation processes with liquids (e.g., reverse osmosis) more severely, as the rate of diffusion in liquids is much lower than in gases. Nevertheless, the effect is observable in gas

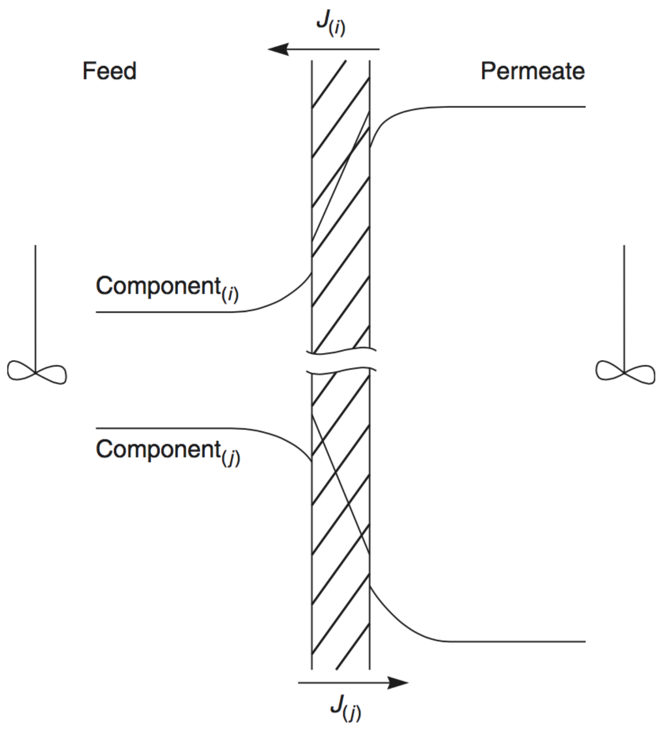


Figure 2.2: Concentration Polarization: Concentration gradients exist in the fluid phases (Baker 2012, p. 180)

phases as well. A detailed description including mathematical models is giving in Baker (2012).

The algorithm presented in this work does not directly consider concentration polarization effects, but it might be possible to emulate the behavior using the flux limit introduced in section 3.3.4.

3 Numerical Approach

3.1 Basics of Numerical Modeling

3.1.1 Classes of Differential Equations

In order to "calculate" real world processes, mathematical models have to be found that describe the relevant phenomena. Many of these processes (especially in mass transfer operations) are based on some quantities changing over space and/or time. The mathematical description for these changes typically involves differential equations (DES).

Generally, DES can be split into ordinary differential equations (ODES), which only contain derivatives in one variable, and partial differential equations (PDES), which have partial derivatives in two or more variables. An additional way of classifying DES is into linear and nonlinear ones. Linear DES only consist of linear combinations of the unknown variable and its derivatives, whereas for nonlinear DES any combination is possible. Finally, DES can be classified by the highest order of the occurring derivatives and whether a single DE or a system of equations is considered. As shown in Bärwolff (2016, p. 233), any DE of higher order can easily be transformed into a first order DE system.

3.1.2 Initial Value Problems and Boundary Value Problems

Solutions for DES themselves are not unique, as they typically include arbitrary constants (for ODES) or arbitrary functions (for PDES). In order to obtain a unique solution it is required to impose some additional restrictions on the solution. There are two possible ways to specify these additional restrictions: One is to define conditions for the solution on one single location in the domain of the independent variable, which leads to an initial value problem (IVP). The other is to define the conditions on (different) boundaries of the domain, which leads to a boundary value problem (BVP).

Note that a single first order ODE cannot really be transformed into a BVP, as only a single constant exists in the solution. Therefore, only one "boundary" condition can be specified, which leads to a plain IVP. This changes when a first order DE system is considered, as a condition for every single equation can be set there.

3.1.3 Solutions for Differential Equations

While it is possible to solve some classes of DEs analytically (especially simple linear ones), there are many DEs which are either unfeasible or downright not possible to solve analytically. Rather than trying to find an analytical solution, many disciplines of engineering now focus on calculating approximate numerical solutions for their models.

The methods by which numerical solutions can be acquired are manyfold. Bärwolff (2016) and Bornemann (2016) give concise introductions to numerical methods in general, whereas Deuflhard and Hohmann (2003), Freund and Hoppe (2007), and Dahmen and Reusken (2008) are somewhat more detailed. Alternatively, Lindfield and Penny (2012) and Mathew and Fink (2004) introduce the topic in a practical way using MATLAB.

The basic numerical methods (e.g. the Euler method or Runge-Kutta solvers) typically cannot be used directly to solve BVPs, as these solvers start at one specific point in the domain and calculate the solution from there. A simple approach to solve BVPs is found in the shooting method, which starts with guesses for the initial values, calculates the solution using some basic numerical method and checks if the boundary conditions are met. If not, new initial values are chosen and the solution is calculated again. While this method is quite efficient for linear problems, as linear interpolation between two guesses is possible, the efficiency for nonlinear problems can vary greatly. An alternative, widely-used approach for solving BVP is the finite element method (see Grossmann et al. 2007; Larsson and Thomée 2009).

The algorithm presented in the following sections employs another commonly used approach, the finite difference method (FDM). In previous work of the research group that this thesis originates in, Makaruk and Harasek (2009) also used the FDM in their algorithm. Their algorithm serves as a basis for this work and is subsequently referred to as (MAKARUK).

3.1.4 The Finite Difference Method

When applying the FDM to solve a BVP, the domain of the solution is discretized into a finite number of grid points. The derivatives in the DEs are approximated with finite differences, which transforms the DE system into a system of algebraic equations. It is possible to derive the finite differences to substitute arbitrary derivatives by using the Taylor Series expansion (see Lindfield and Penny 2012, p. 187). As the DEs under consideration only contain first order derivatives, higher order derivatives will not be considered in the following part.

There exist three simple finite difference approximations for first order differences:

$$\begin{aligned}
&\text{Forward differencing:} && f'(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x} \\
&\text{Backward differencing:} && f'(x) \approx \frac{f(x) - f(x-\Delta x)}{\Delta x} \\
&\text{Central differencing:} && f'(x) \approx \frac{f(x+\frac{1}{2}\Delta x) - f(x-\frac{1}{2}\Delta x)}{\Delta x}
\end{aligned}$$

When deriving these approximations using Taylor Series expansion it can be shown that forward and backward differencing have errors of $\mathcal{O}(\Delta x)$, while central differencing has an error of $\mathcal{O}(\Delta x^2)$.

A simplification in notation can be made if it is assumed that all grid points are spaced evenly (with a distance of Δx):

$$\begin{aligned}
f(x_i) &= f_i \\
f(x_i + \Delta x) &= f_{i+1} \\
f(x_i - \Delta x) &= f_{i-1} \\
f(x_i + 2\Delta x) &= f_{i+2} \\
f(x_i + \frac{1}{2}\Delta x) &= f_{i+\frac{1}{2}} \\
&\dots
\end{aligned}$$

$$\text{Forward differencing:} \quad f_i' \approx \frac{f_{i+1} - f_i}{\Delta x} \tag{3.1}$$

$$\text{Backward differencing:} \quad f_i' \approx \frac{f_i - f_{i-1}}{\Delta x} \tag{3.2}$$

$$\text{Central differencing:} \quad f_i' \approx \frac{f_{i+\frac{1}{2}} - f_{i-\frac{1}{2}}}{\Delta x} \tag{3.3}$$

When these approximations are applied at all n grid points, one obtains a system of algebraic equations in $f_1, f_2, \dots, f_n$, where the specified boundary conditions can be included easily. If a DE system has to be solved, the approximations have to be applied to each of the coupled equations, which in turn also results in a system of algebraic equations. Whether the obtained system of equations is linear or not depends directly on the underlying DE (a linear DE leads to a linear system of equations and vice versa). As the differencing schemes mentioned above only depend on the values of the neighboring grid points, the resulting system of equations is typically sparse.

3.1.5 Iterative Solvers

There are many methods available for solving systems of algebraic equations, both linear and nonlinear (see e.g. Bärwolff 2016). Commonly used iterative methods are the Jacobi method and the Gauss-Seidel method. Both of these methods have initially been developed for solving systems of linear equations, but have also been successfully applied to solve nonlinear systems. The basic approach for both methods is similar: Each of the n variables can be expressed in terms of the other variables (and the boundary conditions). For nonlinear cases, it is often not possible to explicitly express a variable, but rather using an implicit equation. In consequence, this means the variable can be expressed in terms of itself, the other variables, and the boundary conditions.

So for the linear case, one can write

$$f_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} \left(c_{i,j} \cdot f_j \right) + b_i \tag{3.4}$$

where $c_{i,j}$ describes the coefficients with which the other variables are considered and b_i describes the boundary conditions (if applicable). In a more general, nonlinear case, the equation can be written as

$$f_i = g\left(f_1, f_2, \dots, f_n, b_1, b_2, \dots, b_n\right) \tag{3.5}$$

where b_i again represents the boundary conditions.

For one single iteration step $k+1$, each of the n variables is calculated one after the other, where the right-hand side (RHS) of (3.4) or (3.5) is assumed constant at previously calculated values. The difference between the Jacobi method and the Gauss-Seidel method lies in the choice of the values for the RHS. While the Jacobi method always uses all values from the previous (k^{th}) iteration, the Gauss-Seidel method uses the values already calculated in the current ($(k+1)^{\text{th}}$) iteration for variables where these values are available. Assuming that the iteration step starts at variable 1 and moves towards n this means that once variable i is reached, the values of variables 1 to $i-1$ are already calculated in the current iteration step and can be used.

Mathematically, one iteration step can be described as follows:

Jacobi:

Linear: $$f_i^{(k+1)} = \sum_{\substack{j=1 \\ j \neq i}}^{n} \left(c_{i,j} \cdot f_j^{(k)} \right) + b_i \tag{3.6}$$

Nonlinear: $$f_i^{(k+1)} = g\left(f_1^{(k)}, f_2^{(k)}, \dots, f_n^{(k)}, b_1, b_2, \dots, b_n\right) \tag{3.7}$$

Gauss-Seidel:

Linear: $$f_i^{(k+1)} = \sum_{j=1}^{i-1} \left(c_{i,j} \cdot f_j^{(k+1)} \right) + \sum_{j=i+1}^{n} \left(c_{i,j} \cdot f_j^{(k)} \right) + b_i \tag{3.8}$$

Nonlinear: $$f_i^{(k+1)} = g\left(f_1^{(k+1)}, \dots, f_{i-1}^{(k+1)}, f_i^{(k)}, f_{i+1}^{(k)}, \dots, f_n^{(k)}, b_1, \dots, b_n\right) \tag{3.9}$$

3.1.6 Relaxation

For systems of linear equations, a sufficient condition for the convergence of (3.6) and (3.8) is an irreducible diagonally dominant matrix, although both methods may also converge if these conditions are not fulfilled (Bärwolff 2016, pp. 197–199). In the nonlinear case, a general condition for convergence cannot be given as this depends on the local behavior of the iteration function at the desired solution. In order to improve the convergence behavior, an additional step called relaxation can be introduced into both the Jacobi and the Gauss-Seidel method. First, let the result $f_i^{(k+1)}$ of (3.6) to (3.8) be denoted as $\hat{f}_i^{(k+1)}$. Then the relaxation step can be described as

$$f_i^{(k+1)} = \omega \cdot \hat{f}_i^{(k+1)} + (1-\omega) \cdot f_i^{(k)} \tag{3.10}$$

where ω is called the relaxation parameter. Using an $\omega > 1$ the resulting step is called over-relaxation, whereas $\omega > 1$ results in under-relaxation. Generally, convergence is *possible* for $0 < \omega < 2$ Bärwolff (see 2016, p. 201), but as stated above, not all systems do actually converge for any given ω. There is no universal recommendation for values of ω as this is heavily dependent on the specifics of the system to solve. Over-relaxation can significantly improve convergence speed for systems where convergence is not a problem. Under-relaxation, on the other hand, can result in convergence for otherwise not converging systems (at the price of lower convergence speeds).

Considering the effects stated above, it can be seen that the choice of ω heavily impacts the convergence speed. It is therefore advisable to make sure the selected ω results in favorable convergence behavior.

3.2 General Approach

3.2.1 Basics of Algorithm

Based on the foundations presented in section 3.1, the remainder of this chapter will be used to introduce an algorithm that is able to solve the flux equations introduced in section 2.2 for hollow fiber membrane modules. Additionally, calculation of pressure drop is handled as well as solving the energy balance (also taking into account the Joule-Thomson effect if required). In this chapter the focus will be on a general description of the algorithm, whereas specific details of the reference implementation will be described in Chapter 4.

In the Algorithm, the basic working element is a *stream*. Each *stream* consists of a set of properties: the molar flows for each component, temperature, pressure, and a list of components (i.e. the single pure gases). All other properties required by calculations are computed, either using formulae or by looking them up in interpolation tables. A detailed description of property calculation is given in section 4.2.

3.2.2 Flow Regimes

As stated by Melin and Rautenbach (2007, pp. 152 sqq.), there is a variety of possible flow configurations in a module. This algorithm only focuses on co-current and counter-current flow regimes (as shown in Fig. 3.1 and does not provide means for direct calculation of cross-flow regimes. The selection of co-current or counter-current flow is not binary, though. Rather, it is possible to specify the permeate outlet location as a continuous value between 0 (at feed inlet) and 1 (at retentate outlet). This splits the module in two parts: The first part between the feed inlet and the permeate outlet is considered to be co-current flow, whereas the second part from the permeate outlet to the retentate outlet is considered to be counter-current flow. Pure co-current or counter-current flow can be specified by setting the outlet parameter location to 1 or 0 respectively. A visual representation of the possible configurations is given in Fig. 3.1.

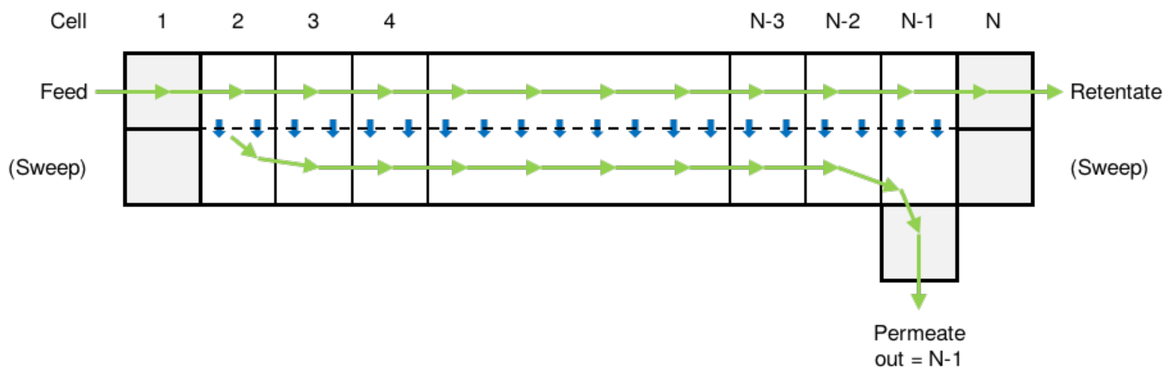
3.2.3 Discretization

The idea behind the discretization in this algorithm is to split the membrane module into n cells on both the feed and permeate side. In each cell, all properties are assumed to be constant and ideally mixed. Both the first (index 1) and last (index n) cell do not represent parts of the membrane module, but are only used to store boundary conditions. This means that the membrane area and length of the module are split into $n-2$ equally sized cells. The index of the permeate outlet cell is calculated from the (continuous) permeate outlet o as:

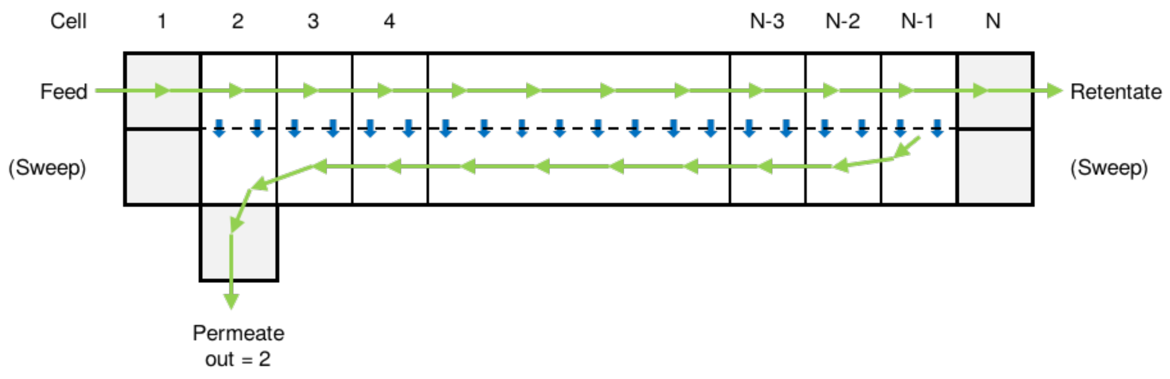
$$out = \lfloor 2 + o \cdot (n-3) \rceil \text{ with } o \in [0,1] \tag{3.11}$$

The model of cells by itself is, though, not suitable to describe a discretization that can be used with the FDM. As stated in section 3.1.4, the FDM does use grid points (i.e. locations in the domain) for discretization rather than cells (i.e. elements that span finite parts of the domain).

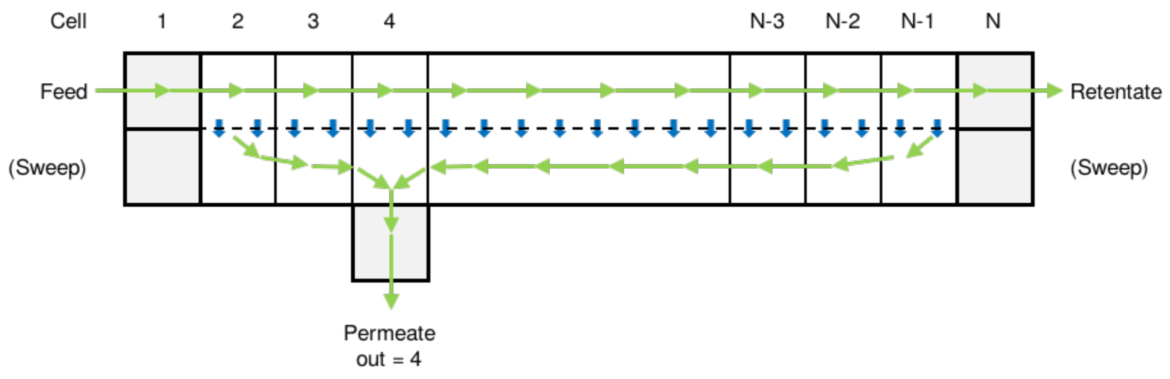
The link between the easy to understand cell model and a mathematically suitable grid model is now defined as follows: Grid points are always located at the downstream end of a cell. This means that on the feed side, the grid point i is located at the interface between cells i and $i+1$. On the permeate side, the relation is different for the co-current and the counter-current section. In the co-current section the grid point i is also located at the interface between cells i and $i+1$. In the counter-current section, though, the grid point i is located at the interface between cells i and $i-1$. As the permeate outlet cell marks the border between the



(a) Pure Co-Current Flow



(b) Pure Counter-Current Flow



(c) Partly Co-Current / Partly Counter-Current Flow

Figure 3.1: Possible Flow Regime Configurations in the Modeled Module

co-current and the counter-current section, both faces to the neighboring cells are considered to be upstream. Therefore, the grid point *out* represents the permeate leaving the module (which actually is the "downstream" value of the outlet cell). The location of storage points is visualized in Fig. 3.2.

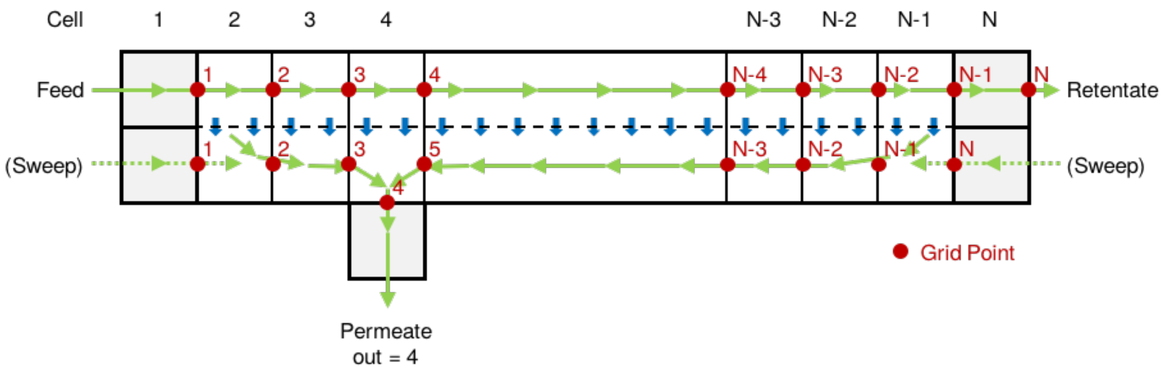


Figure 3.2: Location of Grid Points in Discretization

3.2.4 Application of Finite Difference Schemes

Having defined a mathematically suitable grid, one can now apply the finite difference approximations on it. By shifting the evaluation points, and assuming the approximations to be equations one can rewrite (3.1) to (3.3) as:

$$\text{Forward differencing:} \quad f'_{i-1} = \frac{f_i - f_{i-1}}{\Delta x}$$

$$\text{Backward differencing:} \quad f'_{i} = \frac{f_i - f_{i-1}}{\Delta x}$$

$$\text{Central differencing:} \quad f'_{i-\frac{1}{2}} = \frac{f_i - f_{i-1}}{\Delta x}$$

Compared to the original equations now the RHS is the same for all three differencing schemes, but the evaluation point of the derivative has changed in the left-hand side (LHS). Rearranging the equations in order to isolate f_i on the LHS leads to:

$$\text{Forward differencing:} \quad f_i = f_{i-1} + \Delta x \cdot f'_{i-1}$$

$$\text{Backward differencing:} \quad f_i = f_{i-1} + \Delta x \cdot f'_{i}$$

$$\text{Central differencing:} \quad f_i = f_{i-1} + \Delta x \cdot f'_{i-\frac{1}{2}}$$

These expressions are now in a form that can be used by an iterative solver. It can be seen that the difference between the schemes now only lies in the evaluation point of the derivative in the RHS. In the previous expressions, the values were expressed in terms of the previous grid point. Alternatively, it is also possible to express them in terms of the following grid point to allow calculating backwards:

$$\begin{aligned}
&\text{Forward differencing:} && f_i = f_{i+1} - \Delta x \cdot f'_i \\
&\text{Backward differencing:} && f_i = f_{i+1} - \Delta x \cdot f'_{i+1} \\
&\text{Central differencing:} && f_i = f_{i+1} - \Delta x \cdot f'_{i+\frac{1}{2}}
\end{aligned}$$

The transport equations under consideration typically define the term f'_i as a function of properties $\mathbf{x}$ at location i and location independent properties $\mathbf{y}$:

$$f'_i = g\left(\mathbf{x}_i, \mathbf{y}\right)$$

Evaluating the function g is trivial for forward and backward differencing, as all location-dependent properties $\mathbf{x}$ are stored at each grid point and are therefore available at location i. For central differencing, no values of $\mathbf{x}$ are available at location $i - \frac{1}{2}$ (or $i + \frac{1}{2}$). Central differencing might be preferable to use, though, as it provides second order accuracy, whereas forward and backward differencing only provide first order accuracy. In order to bypass the issue, $g\left(\mathbf{x}_{i-\frac{1}{2}}, \mathbf{y}\right)$ can be estimated using $\mathbf{x}_{i-1}$ and $\mathbf{x}_i$ by averaging the result. Two possible options for averaging come into mind:

$$\begin{aligned}
\bar{g}\left(\mathbf{x}_{i-\frac{1}{2}}, \mathbf{y}\right) &= \operatorname{avg}\left(g\left(\mathbf{x}_{i-1}, \mathbf{y}\right), g\left(\mathbf{x}_i, \mathbf{y}\right)\right) \\
\bar{g}\left(\mathbf{x}_{i-\frac{1}{2}}, \mathbf{y}\right) &= g\left(\operatorname{avg}\left(\mathbf{x}_{i-1}, \mathbf{x}_i\right), \mathbf{y}\right)
\end{aligned}$$

The first option evaluates the function g for both locations $i - 1$ and i and then averages the result. The second option averages the parameters at locations $i-1$ and i and then evaluates the function g with the averaged parameters. If the averaging operation (e.g. arithmetic mean) and the function g are linear, then both options lead to identical results. This is not the case when either the averaging operation (e.g. logarithmic mean) or the function g are nonlinear.

Although the first option shows some ideas similar to the tried and tested Crank-Nicolson method (Crank et al. 1947), it cannot be decided upfront which of the two options is preferable, as the results depend heavily on the specific functions. Under the assumption that the function g is computationally significantly more expensive than the averaging operation, applying the averaging operation on all properties is probably cheaper than calling the function g twice.

The presented algorithm allows the choice of 4 different options for evaluating the function g:

Upstream The function g is evaluated at the upstream end of a cell, which means forward differencing in flow direction. As the flow is directed in negative direction in the counter-current section of the permeate side, this implies backward differencing in terms of the grid coordinates in this section.

Downstream The function g is evaluated at the downstream end of a cell, which means backward differencing in flow direction. As the flow is directed in negative direction in the counter-current section of the permeate side, this implies forward differencing in terms of the grid coordinates in this section.

Arithmetic average The function g is evaluated with an arithmetic average of the downstream and upstream properties. A compact notation for an arithmetic average is:

$$\overline{\mathbf{x}_{\langle i-1,\, i\rangle}}^{\text{arithm.}} = \frac{\mathbf{x}_{i-1} + \mathbf{x}_i}{2}$$

Logarithmic average The function g is evaluated with an logarithmic average of the downstream and upstream properties. A compact notation for a logarithmic average is:

$$\overline{\mathbf{x}_{\langle i-1,\, i\rangle}}^{\text{log.}} = \begin{cases} \dfrac{\mathbf{x}_{i-1} - \mathbf{x}_i}{\ln \mathbf{x}_{i-1} - \ln \mathbf{x}_i} & \text{if } \mathbf{x}_{i-1} \neq \mathbf{x}_i \\ \mathbf{x}_i & \text{if } \mathbf{x}_{i-1} = \mathbf{x}_i \end{cases}$$

For both averaging options, averaging is typically applied at the level of intermediary properties. This means that if the function g depends on for example the partial pressures, then the averaging is applied directly for the partial pressures and not for the underlying properties (total pressure and molar fraction). Detailed descriptions which properties are averaged are given in the respective sections.

As the choice of evaluation location is present in many equations throughout the algorithm, a notation is introduced which indicates that the property is calculated according to the selected method: $\mathbf{x}_{\langle i\rangle}$. This notation evaluates to a different expression depending on the choice of evaluation, whether it is a feed side or a permeate side property, and the location of i. It is only defined for $2 \leq i \leq n-1$, as $i = 1$ and $i = n$ do not correspond to cells, but are only the boundary values. For the **feed side**, any property $\mathbf{x}_{F,\langle i\rangle}$ evaluates to:

$$\mathbf{x}_{F,\langle i\rangle} = \begin{cases} \mathbf{x}_{F,i-1} & \text{if upstream} \\ \mathbf{x}_{F,i} & \text{if downstream} \\ \overline{\mathbf{x}_{F,\langle i-1,\, i\rangle}}^{\text{arithm.}} & \text{if arithmetic average} \\ \overline{\mathbf{x}_{F,\langle i-1,\, i\rangle}}^{\text{log.}} & \text{if logarithmic average} \end{cases} \tag{3.12}$$

On the **permeate side**, the expression depends on the location of i. The evaluation location for the outlet cell is always the downstream value, as there is no unique upstream side.

For $i < out$:

$$\mathbf{x}_{P,\langle i\rangle} = \begin{cases} \mathbf{x}_{P,i-1} & \text{if upstream} \\ \mathbf{x}_{P,i} & \text{if downstream} \\ \overset{\text{arithm.}}{\overline{\mathbf{x}_{P,\langle i-1,\, i\rangle}}} & \text{if arithmetic average} \\ \overset{\text{log.}}{\overline{\mathbf{x}_{P,\langle i-1,\, i\rangle}}} & \text{if logarithmic average} \end{cases} \tag{3.13}$$

For $i = out$:

$$\mathbf{x}_{P,\langle i\rangle} = \mathbf{x}_{P,i} \tag{3.14}$$

For $i > out$:

$$\mathbf{x}_{P,\langle i\rangle} = \begin{cases} \mathbf{x}_{P,i+1} & \text{if upstream} \\ \mathbf{x}_{P,i} & \text{if downstream} \\ \overset{\text{arithm.}}{\overline{\mathbf{x}_{P,\langle i+1,\, i\rangle}}} & \text{if arithmetic average} \\ \overset{\text{log.}}{\overline{\mathbf{x}_{P,\langle i+1,\, i\rangle}}} & \text{if logarithmic average} \end{cases} \tag{3.15}$$

3.3 Transmembrane Flux and Mass Balance

3.3.1 Governing Equations

The formulation of the equations for transmembrane flux are based on Makaruk and Harasek (2009). One general change is that the unit used to describe the flows in the feed and permeate channel is mole flows ($\mathrm{mol\,s^{-1}}$) rather than standard temperature and pressure (STP) volume flows.

If it can be assumed that the only change happening inside the feed and permeate channel is due to transmembrane flux, the local mass conservation equations can be written as

$$\frac{d\dot{n}_{F,j}}{dl} = -\dot{Q}'_j \qquad\qquad \frac{d\dot{n}_{P,j}}{dl} = \dot{Q}'_j \tag{3.16}$$

where $\dot{Q}'_j$ denotes the length-specific transmembrane flux. As described in section 2.2, the transmembrane flux can be described using the solution-diffusion model:

$$\dot{Q}'_j = \dot{Q}''_j \cdot A' = \Pi_j \cdot \left(p_{F,j} - p_{P,j}\right) \cdot A' \tag{3.17}$$

In general, the permeance Π of component j can be a function of pressure, temperature, and composition on both the feed and permeate side:

$$\Pi_j = \Pi_j\left(p_F, p_P, T_F, T_P, x_{F,1}, \dots, x_{F,m}, x_{P,1}, \dots, x_{P,m}\right)$$

There have been many attempts to model variable permeances, but these models typically depend on various parameters that have to be determined experimentally (Bounaceur et al. 2017; Scholz et al. 2013a; Ahmad et al. 2013; Hosseini et al. 2016b; P. Kundu et al. 2013).

As conducting experiments to find these parameters is not part of the scope of this work and using such models without sensible values for the parameters would not lead to results that can be validated, the per-component permeances were modeled to be constant. The implementation was, however, designed to be adaptable so that any variable permeance model can easily be implemented in the future.

The partial pressures of component j can be expressed as

$$p_{F,j} = x_{F,j}\, p_F \qquad\qquad p_{P,j} = x_{P,j}\, p_P \tag{3.18}$$

for ideal gases that follow Dalton's law. As stated in section 2.2, non-ideal cases can be handled by introducing fugacity coefficients φ:

$$p_{F,j} = \varphi_{F,j}\, x_{F,j}\, p_F \qquad\qquad p_{P,j} = \varphi_{P,j}\, x_{P,j}\, p_P \tag{3.19}$$

The local mass conservation equations for single components are coupled by the definition of the molar fraction x_j of a component:

$$x_{F,j} = \frac{\dot{n}_{F,j}}{\sum\limits_{l=1}^{m} \dot{n}_{F,l}} \qquad\qquad x_{P,j} = \frac{\dot{n}_{P,j}}{\sum\limits_{l=1}^{m} \dot{n}_{P,l}} \tag{3.20}$$

These expressions are essential to the algorithm as they link the component specific mass conservation equations (3.16) to the other components. While there may be other sources of nonlinearity, like variable permeance models, non-ideal gas behavior, or the calculation of pressure drop and energy balance introduced in Sections 3.4 and 3.5, these expressions are the primary reason that even the most simple calculation results in a nonlinear DE system.

Finally, A' denotes the length specific membrane area and can be calculated as:

$$A' = \frac{A}{l} = n_f \cdot D_A \cdot \pi \tag{3.21}$$

It has to be noted that these equations only cover transport across the membrane from the feed channel to the permeate channel (and vice versa). Inside both channels, the only transport considered is the flow. Diffusion alongside the channels is not considered. For the geometries under consideration, though, the speed of the convective transport with the flow is typically much higher than the speed of diffusion. Thus the neglect of these effects is justified.

3.3.2 Boundary Conditions

The relevant boundary conditions for the equations describing transmembrane flux are of Dirichlet type and located at all three module inlets (feed, co-current sweep gas – $Sweep_0$, counter-current sweep gas – $Sweep_1$). There the molar flows for each component are fully specified:

$$\dot{n}_{F,j,1} = \dot{n}_{Feed,j} \qquad \dot{n}_{P,j,1} = \dot{n}_{Sweep_0,j} \qquad \dot{n}_{P,j,n} = \dot{n}_{Sweep_1,j} \tag{3.22}$$

Therefore, the direction of calculation has to be from the inlets towards the outlets, which means from the feed inlet to the retentate outlet on the feed side and from both sweep gas inlets to the permeate outlet on the permeate side.

3.3.3 Discretization

Using the notation introduced in section 3.2.4, the mass conservation equations (3.16) can be discretized as

$$\dot{n}_{F,j,i} = \begin{cases} \dot{n}_{F,j,i-1} - \dot{Q}_{j,i} & \text{for } 2 \leq i \leq n-1 \\ \dot{n}_{F,j,i-1} & \text{for } i = n \end{cases} \tag{3.23}$$

$$\dot{n}_{P,j,i} = \begin{cases} \dot{n}_{P,j,i-1} + \dot{Q}_{j,i} & \text{for } 2 \leq i \leq out-1 \\ \dot{n}_{P,j,i-1} - \dot{n}_{P,j,i+1} + \dot{Q}_{j,i} & \text{for } i = out \\ \dot{n}_{P,j,i+1} - \dot{Q}_{j,i} & \text{for } out+1 \leq i \leq n-1 \end{cases} \tag{3.24}$$

where

$$\dot{Q}_{j,i} = \dot{Q}'_{j,i}\Delta l = \Pi_{j,\langle i \rangle} \cdot \left(p_{F,j,\langle i \rangle} - p_{P,j,\langle i \rangle}\right) \cdot A'\Delta l \tag{3.25}$$

As $\Pi_{j,i}$ is assumed to be a constant Π_j in the current implementation, dependency on location specific property does not exist. If one wants to integrate such dependency, the properties to be used are to be subject to the selection of evaluation location like $p_{F,j,\langle i \rangle}$ and $p_{P,j,\langle i \rangle}$.

The remaining variables substitutions can be transferred directly from equations (3.18), (3.19), (3.20), and (3.21). Thus the discretization is fully described.

3.3.4 Ensuring Mass Balance by Limiting Flux

The discretized version can lead to specific issues when solved iteratively. During a single iteration step, $\dot{Q}_{j,i}$ is assumed to be constant. There are parameters in equation (3.25) that can reach arbitrarily high values (e.g. A' or $\Pi_{j,i}$), which means that $\dot{Q}_{j,i}$ can reach arbitrarily high values. For the feed side, this leads to a problematic state when $\dot{Q}_{j,i}$ becomes larger in value than $\dot{n}_{F,j,i-1}$. This would mean that the amount of mass flowing through the membrane is higher than the amount of mass reaching the segment of the membrane, which is a clear violation of the mass balance. The result would be that $\dot{n}_{F,j,i}$ becomes negative and therefore, due to the equations linking molar fractions to component flows (3.20), the calculated molar fractions would become negative. In turn, severe numerical issues would arise, which typically result in a non-converging solution.

Furthermore, the transmembrane flux $\dot{Q}_{j,i}$ is not necessarily positive, as situations where the permeate side partial pressure is higher than the feed side partial pressure are possible. Therefore the same adverse effects are also possible on the permeate side.

It has to be noted, that $\dot{n}_{P,j,i}$ is typically negative in the region of counter-current flow (for $i \geq out + 1$) due to the way the streams are described mathematically. There the issues arise when single values of $\dot{n}_{P,j,i}$ become positive.

A possible solution for these issues would be to algorithmically force the signs of $\dot{n}_{F,j,i}$ and $\dot{n}_{P,j,i}$, but this would lead to inconsistent solution behavior as the value of $\dot{Q}_{j,i}$ would not correspond to the change of $\dot{n}_{F,j,i}$ and $\dot{n}_{P,j,i}$ anymore.

The approach taken in the presented algorithm is to introduce a new value, $\hat{\dot{Q}}_{j,i}$ which is a limited transmembrane flux that avoids the numerical issues described above. The basic idea is to compare the calculated transmembrane flux to the corresponding available amount on the feed or permeate side.

The algorithm allows for two possible options to calculate the limited flux:

- applying a hard cut at the reference value
- dampening the transmembrane flux with a smooth function

First, the reference value has to be selected:

$$\dot{Q}_{ref,F,j,i} = \dot{n}_{F,j,i-1} \tag{3.26}$$

$$\dot{Q}_{ref,P,j,i} = \begin{cases} \dot{n}_{P,j,i-1} & \text{for } 2 \leq i \leq out - 1 \\ \dot{n}_{P,j,i-1} - \dot{n}_{P,j,i+1} & \text{for } i = out \\ -\dot{n}_{P,j,i+1} & \text{for } out + 1 \leq i \leq n - 1 \end{cases} \tag{3.27}$$

The hard limit can then be applied as:

$$\hat{\dot{Q}}_{j,i} = \begin{cases} \min\left(\dot{Q}_{j,i}, \dot{Q}_{ref,F,j,i}\right) & \text{if } \dot{Q}_{j,i} \geq 0 \\ \max\left(\dot{Q}_{j,i}, -\dot{Q}_{ref,F,j,i}\right) & \text{if } \dot{Q}_{j,i} < 0 \end{cases} \tag{3.28}$$

The soft limit is applied as:

$$\hat{\dot{Q}}_{j,i} = \left(\frac{1}{1 + {R_{j,i}}^{q}}\right)^{-q} \cdot \dot{Q}_{j,i} \tag{3.29}$$

where

$$R_{j,i} = \begin{cases} \dfrac{\dot{Q}_{j,i}}{\dot{Q}_{ref,F,j,i}} & \text{if } \dot{Q}_{j,i} \geq 0 \\ \dfrac{\dot{Q}_{j,i}}{-\dot{Q}_{ref,P,j,i}} & \text{if } \dot{Q}_{j,i} < 0 \end{cases}$$

and $q \in \,]0, \infty[$ denotes the dampening exponent.

Fig. 3.3 shows the effects of the hard limit and soft limits with different dampening parameters. As can be seen from the equations, the soft limit is computationally more expensive. It might lead to more favorable convergence behavior as the transition from non-limited to limited flux is smoother. Setting low values for the dampening parameter can significantly change the calculations, as is shown in section 5.2. Universally applicable recommendations cannot be given, though, as the convergence behavior heavily depends on the specific case.

3.4 Pressure Drop Calculation

3.4.1 Governing Equations

In contrast to the fluxes described in the previous section, pressures are not described by a transport equation and there is also no "conservation" of pressures. Rather, pressure drop occurs as a "by-product" of mass flow and is dependent on the viscosity of the fluid and the geometry (e.g., for a pipe the inner diameter).

The equations implemented in the current algorithm are based on the assumption of laminar flow. During the development of the algorithm, no case was observed where the Reynolds number would indicate turbulence. It has to be noted, though, that a check of the local Reynolds numbers is advisable on a case by case basis.

The pressure drop in a hollow fiber module is calculated for sides (bore and shell). Bore side, the model assumes flow through a pipe and shell side, flow parallel to tube bundles is considered.

Laminar flow in a pipe is described by the Hagen-Poiseuille law (VDI 2013, p. 1223). Using the viscosity of the bore side fluid (μ_b), the volumetric flow rate ($\dot{V}_b$), and the inner fiber diameter (D_I), this can be applied to the bore side channel

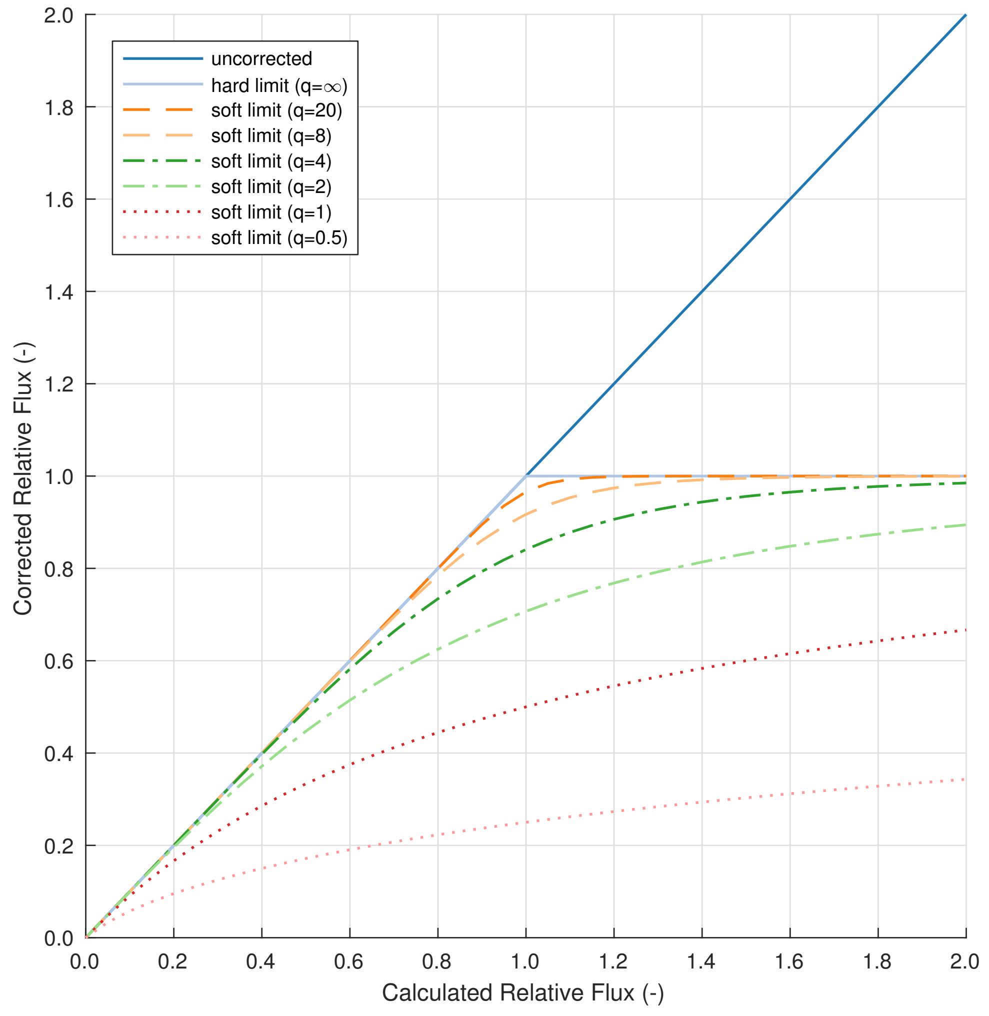


Figure 3.3: Effect of Different Settings for Flux Limit on the Calculated Flux

as

$$\frac{dp_b}{dl} = \frac{32\,\mu_b\,\dot{V}_b}{{D_I}^2\,{A^x}_b}$$

with the cross sectional area

$${A^x}_b = n_f \cdot \frac{{D_I}^2\,\pi}{4}$$

which leads to

$$\frac{dp_b}{dl} = \frac{128\,\mu_b\,\dot{V}_b}{n_f\,\pi\,{D_I}^4}\,. \tag{3.30}$$

For the flow in a tube bundle the approach is based on the calculation of pressure drop in the shell of a heat exchanger without baffles in VDI (2013, p. 1271). This leads to similar equations, where a hydraulic diameter is used instead of the pipe inner diameter.

$$\frac{dp_s}{dl} = \frac{32\,\mu_s\,\dot{V}_s}{{D_H}^2\,{A^x}_s} \tag{3.31}$$

with the cross sectional area

$${A^x}_s = \frac{\left({D_M}^2 - n_f {D_O}^2\right)\,\pi}{4} \tag{3.32}$$

and the hydraulic diameter

$$D_H = \frac{{D_M}^2 - n_f {D_O}^2}{D_M + n_f D_O} \tag{3.33}$$

These equations were only written in terms of bore side and shell side properties. In order to apply these to the properties of the feed and permeate side, it has to be distinguished between bore side feed and shell side feed. For **bore side feed**, equations (3.30) and (3.31) transform into

$$\frac{dp_F}{dl} = \frac{128\,\mu_F\,\dot{V}_F}{n_f\,\pi\,{D_I}^4} \qquad \frac{dp_P}{dl} = \frac{32\,\mu_P\,\dot{V}_P}{{D_H}^2\,{A^x}_s} \tag{3.34}$$

whereas for **shell side feed** the correct equations are

$$\frac{dp_F}{dl} = \frac{32\,\mu_F\,\dot{V}_F}{{D_H}^2\,{A^x}_s} \qquad \frac{dp_P}{dl} = \frac{128\,\mu_P\,\dot{V}_P}{n_f\,\pi\,{D_I}^4} \tag{3.35}$$

The equations describing the shell side cross sectional area (3.32) and hydraulic diameter (3.33) remain the same for both cases.

3.4.2 Boundary Conditions

For the calculation of pressure drop, the relevant boundary conditions are the feed pressure and the permeate pressure:

$$p_{F,1} = p_{Feed} \qquad\qquad p_{P,out} = p_{Perm} \tag{3.36}$$

It has to be noted that compared to the flux equations (see section 3.3.2), here the boundary condition is set at the permeate outlet rather than the sweep gas inlets. This has been chosen to make typical calculation scenarios easily achievable, as there the permeate pressure is typically a given parameter.

Therefore the direction of calculation remains from the feed inlet to the retetnate outlet on the feed side, but a different direction of calculation is required on the permeate side. There the direction of calculation has to be from the permeate outlet towards the two sweep gas inlets.

3.4.3 Discretization

Discretization of equations (3.34) and (3.35) leads to:

$$p_{F,i} = \begin{cases} p_{F,i-1} - \Delta p_{F,i} & \text{for } 2 \le i \le n-1 \\ p_{F,i-1} & \text{for } i = n \end{cases} \tag{3.37}$$

$$p_{P,i} = \begin{cases} p_{P,i+1} & \text{for } i = 1 \\ p_{P,i+1} - \Delta p_{P,i} & \text{for } 2 \le i \le out-1 \\ p_{P,i-1} - \Delta p_{P,i} & \text{for } out+1 \le i \le n-1 \\ p_{P,i-1} & \text{for } i = n \end{cases} \tag{3.38}$$

The pressure drop terms depend on whether the feed is bore side or shell side. For **bore side feed**, one obtains

$$\Delta p_F = \frac{128\,\Delta l}{n_f \pi D_I^4} \cdot \mu_{F,\langle i\rangle} \dot{V}_{F,\langle i\rangle} \qquad\qquad \Delta p_P = \frac{32\,\Delta l}{D_H^{\,2} A^x{}_s} \cdot \mu_{P,\langle i\rangle} \dot{V}_{P,\langle i\rangle} \tag{3.39}$$

whereas for **shell side feed** the equations are:

$$\Delta p_F = \frac{32\,\Delta l}{D_H^{\,2} A^x{}_s} \cdot \mu_{F,\langle i\rangle} \dot{V}_{F,\langle i\rangle} \qquad\qquad \Delta p_P = \frac{128\,\Delta l}{n_f \pi D_I^4} \cdot \mu_{P,\langle i\rangle} \dot{V}_{P,\langle i\rangle} \tag{3.40}$$

For the cell specific properties μ and $\dot{V}$, again the notation introduced in section 3.2.4 was used.

The (constant) geometry properties $A^x{}_s$ and D_H can be taken directly from equations (3.32) and (3.33). So far undefined are the viscosity μ and volume flow $\dot{V}$ of the mixed gas. No universally applicable descriptions can be given. The volume flow can be derived from the other properties using an equation of state. For an ideal gas mixture this can simply be the ideal gas law using a mixed molar mass, but once non-ideal effects are assumed, both the properties of single components and interaction parameters have to be considered. For viscosities the situation is even more complex, no easily applicable equation was found in literature, not even for ideal gases and the usable approximations all heavily depend on the specific case. The reference implementation therefore relies on setting fixed pure gas viscosities and relies on these to calculated mixed gas viscosities. The calculation of properties in the reference implementation is described in detail in section 4.2.

Assuming a method for calculating the mixed gas properties is specified, the discretization is now fully described.

3.5 Energy Balance

3.5.1 Governing Equations

Energy balance, like flux, is described by conservation and transport equations. The difference is that only one quantity (enthalpy) is transported, whereas in flux there are multiple distinct components that are transported. Furthermore, there are two possible distinct means of enthalpy transport, namely enthalpy transported with mass flow and heat transfer. Heat transfer is described according to Section B2 4.2 of VDI (2013, pp. 34 sqq.). Thermal radiation is not considered, as it is negligibly small compared to the other means of heat transport for the typical operating conditions of hollow fiber modules (temperatures well below 100 °C). Additionally, as is the case in transmembrane flux, no thermal conduction is considered in longitudinal direction in the feed and permeate channels.

Convective transport is considered by calculating the enthalpy of the transmembrane flow. As transmembrane flux is not restricted in its direction, the case can happen that some components can travel from the feed to the permeate side, whereas simultaneously other components can travel from the permeate to the feed side at a given location in the module. This case is handled in the algorithm by calculating the enthalpy of both the "forward" and the "backward" stream separately.

It is important to note, that as stated in section 3.2.1, the algorithm internally stores temperatures and pressures, but not enthalpies. This means that both enthalpies have to be calculated from the stored temperatures and in the end the resulting enthalpies have to be converted back to temperatures. Depending on the underlying model for enthalpy, these conversions can lead to additional mathematical challenges, as is described in section 4.2.

Given that the energy balance equations are written in terms of enthalpy, a conversion from and to the basic stream properties has to be defined:

$$\dot{H}_F = \dot{n}_F \cdot {h^m}_F \qquad \dot{H}_P = \dot{n}_P \cdot {h^m}_P \tag{3.41}$$

$$ {h^m}_F = h^m \left(p_F, T_F, x_{F,1}, \cdots, x_{F,m}\right) \qquad {h^m}_P = h^m \left(p_P, T_P, x_{P,1}, \cdots, x_{P,m}\right) \tag{3.42}$$

The conversion from molar enthalpies back to temperatures is also required:

$$T_F = T\left(p_F, {h^m}_F, x_{F,1}, \cdots, x_{F,m}\right) \qquad T_P = T\left(p_P, {h^m}_P, x_{P,1}, \cdots, x_{P,m}\right) \tag{3.43}$$

The underlying differential energy conservation equations for the feed and permeate channel can be written as

$$\frac{d\dot{H}_F}{dl} = -\dot{q}' \qquad \frac{d\dot{H}_P}{dl} = \dot{q}' \tag{3.44}$$

where the length specific enthalpy flow rate $\dot{q}'$ consists of the conduction and convection terms:

$$\dot{q}' = {\dot{q}'}_{\overrightarrow{FP}} - {\dot{q}'}_{\overrightarrow{PF}} + {\dot{q}'}_h \tag{3.45}$$

${\dot{q}'}_{\overrightarrow{FP}}$ denotes the enthalpy transported with mass flow from the feed channel to the permeate channel, whereas ${\dot{q}'}_{\overrightarrow{PF}}$ denotes the enthalpy transported with mass flow in the opposite direction. The overall heat transfer is described by ${\dot{q}'}_h$. The enthalpy transported with mass primarily depends on the transmembrane flux, but considers both directions separately.

$${\dot{q}'}_{\overrightarrow{FP}} = {\dot{Q}'}_{\overrightarrow{FP}} \cdot {h^m}_{\overrightarrow{FP}} \qquad {\dot{q}'}_{\overrightarrow{PF}} = {\dot{Q}'}_{\overrightarrow{PF}} \cdot {h^m}_{\overrightarrow{PF}} \tag{3.46}$$

As this calculation should be self contained and available separated from a transmembrane flux calculation, a definition of transmembrane flux is included here again, that only depends on stream properties:

$${\dot{Q}'}_j = -\frac{d\dot{n}_{F,j}}{dl} = \frac{d\dot{n}_{P,j}}{dl} \tag{3.47}$$

$${\dot{Q}'}_{\overrightarrow{FP},j} = \max\left(0, {\dot{Q}'}_j\right) \qquad {\dot{Q}'}_{\overrightarrow{PF},j} = \min\left(0, {\dot{Q}'}_j\right) \tag{3.48}$$

$${\dot{Q}'}_{\overrightarrow{FP}} = \sum_{j=1}^{m} {\dot{Q}'}_{\overrightarrow{FP},j} \qquad {\dot{Q}'}_{\overrightarrow{PF}} = \sum_{j=1}^{m} {\dot{Q}'}_{\overrightarrow{PF},j} \tag{3.49}$$

As the molar enthalpy h^m is dependent on the stream properties, it has to be defined, which values are to be assumed for these properties:

$${h^m}_{\overrightarrow{FP}} = h^m \left(p_F, T_F, x_{\overrightarrow{FP},1}, \cdots, x_{\overrightarrow{FP},m}\right) \tag{3.50}$$

$${h^m}_{\overrightarrow{PF}} = h^m \left(p_P, T_P, x_{\overrightarrow{PF},1}, \cdots, x_{\overrightarrow{PF},m}\right) \tag{3.51}$$

$$x_{\overrightarrow{FP},j} = \frac{\dot{Q}'_{\overrightarrow{FP},j}}{\dot{Q}'_{\overrightarrow{FP}}} \qquad x_{\overrightarrow{PF},j} = \frac{\dot{Q}'_{\overrightarrow{PF},j}}{\dot{Q}'_{\overrightarrow{PF}}} \tag{3.52}$$

This means that the composition is calculated separately for both part streams. Temperature and pressure are from the channel side where the flow originates from.

As there is a significant difference between the pressures in the feed and permeate channels in typical gas permeation applications, often a change of temperature can be observed due to the Joule-Thomson effect (Melin and Rautenbach 2007, p. 477). This is handled implicitly in the underlying property model linking enthalpies to pressures and temperatures. The algorithm just assumes that the gas passes through the membrane isenthalpicly.

In order to calculate heat transfer through the membrane, the hollow fiber module is assumed as a tube bundle, as it is for the pressure drop calculation. For a single fiber the heat transfer is computed as for a single pipe. The overall heat transfer is then evaluated as the sum of the single fiber heat transfers. Heat transfer for a single pipe is defined by three components (VDI 2013, pp. 34 sqq.):

- Convective heat transfer on the inside (bore side)
- Heat conduction through the wall (membrane)
- Convective heat transfer on the outside (shell side)

Heat conduction through the membrane is straightforward, as long as the thermal conductivity of the membrane material λ_f is known.

For the convective heat transfer on both sides of the membrane, the heat transfer coefficient is calculated using Nusselt numbers. On the inside, the Nusselt number to choose is relatively clear: Assuming a laminar flow through a pipe with a low Reynolds number and a long (compared to the diameter) pipe, it takes a constant value (VDI 2013, pp. 785 sqq.):

$$\mathrm{Nu}_b = \frac{\alpha_b D_I}{\lambda_b} = 3.66 \tag{3.53}$$

On the outside of the membrane, the situation is not that straightforward. VDI do not give values for flow in a tube bundle parallel to the tubes. The assumption taken for this algorithm was to model the hollow fiber module using the correlations for concentric annular ducts (VDI 2013, pp. 793 sqq.). The employed model assumes one single fiber in the shell and considers this the concentric annular. Assuming an adiabatic outside shell, the correlation describing heat transfer is:

$$\mathrm{Nu}_s = \frac{\alpha_s (D_M - D_O)}{\lambda_s} = 3.66 + 1.2 \left(\frac{D_O}{D_M} \right)^{-0.8} \tag{3.54}$$

An alternative approach would be to take the actual hydraulic diameter of the shell D_H as reference length instead of the difference between the module and single fiber diameter $(D_M - D_O)$.

$$\mathrm{Nu}_s = \frac{\alpha_s D_H}{\lambda_s} = 3.66 + 1.2\left(\frac{D_O}{D_M}\right)^{-0.8} \tag{3.55}$$

For $n_f = 1$, both (3.54) and (3.55) have the same result. With growing numbers of fibers, the heat transfer calculated with (3.55) is continuously higher than the one calculated with (3.54). Neither of these equations could be validated so far. The reference implementation uses (3.54) until some kind of validation (e.g. using CFD) is possible. If one wants to use (3.55), the expression $(D_M - D_O)$ has to be replaced with D_H in the following equations (3.56) and (3.57).

As these values are related to bore side and shell side properties, it has to be distinguished between bore side feed and shell side feed.

For **bore side feed**, the bore side properties correspond with the feed side properties and the shell side properties correspond with permeate side properties:

$$\alpha_b = \frac{\mathrm{Nu}_b \lambda_F}{D_I} \qquad \alpha_s = \frac{\mathrm{Nu}_s \lambda_P}{(D_M - D_O)} \tag{3.56}$$

For **shell side feed**, the opposite is the case:

$$\alpha_b = \frac{\mathrm{Nu}_b \lambda_P}{D_I} \qquad \alpha_s = \frac{\mathrm{Nu}_s \lambda_F}{(D_M - D_O)} \tag{3.57}$$

Combining the equations for heat conduction through the membrane and convective transport on both sides, a product of overall heat transfer coefficient and membrane area can be defined (VDI 2013, p. 34):

$$\alpha A = \frac{l\, n_f \pi}{(\alpha_b D_I)^{-1} + \dfrac{\ln (D_O/D_I)}{\lambda_f} + (\alpha_s D_O)^{-1}} \tag{3.58}$$

As the differential equations use length specific heat transfer, a product overall heat transfer coefficient and length specific area is required. This can be described as:

$$\alpha A' = \frac{n_f \pi}{(\alpha_b D_I)^{-1} + \dfrac{\ln (D_O/D_I)}{\lambda} + (\alpha_s D_O)^{-1}} \tag{3.59}$$

Using this value, the (differential) overall heat transfer is calculated as:

$$\dot{q}'_h = \alpha\, A'\; (T_F - T_P) \tag{3.60}$$

3.5.2 Boundary Conditions

The boundary conditions for the energy balance calculation are defined at the stream inlets, as for the flux calculation. On a basic stream level, the temperatures are prescribed:

$$T_{F,1} = T_{Feed} \qquad T_{P,1} = T_{Sweep_0} \qquad T_{P,n} = T_{Sweep_1} \tag{3.61}$$

As the algorithm uses enthalpy flows for calculation, the boundary conditions for temperature can be translated into boundary conditions for enthalpy flows:

$$\dot{H}_{F,1} = \dot{H}_{Feed} \qquad \dot{H}_{P,1} = \dot{H}_{Sweep_0} \qquad \dot{H}_{P,n} = \dot{H}_{Sweep_1} \tag{3.62}$$

3.5.3 Discretization

The discretization of the energy balance equations is not as straightforward as it is for the flux and pressure drop calculations. This is mainly affected by two aspects:

1. When performing energy balance calculations before the flux equations are fully converged (see section 3.6), mass balance is not necessarily fulfilled. Even more so, as the calculation is subject to numerical errors, mass balance will never be exactly fulfilled.

   This especially means that in the discretized form of equation (3.47) ($\dot{Q}'_j = -\frac{d\dot{n}_{F,j}}{dl} = \frac{d\dot{n}_{P,j}}{dl}$) the two RHS expressions are not equal. One approach would be to just take one of the expressions and calculate the enthalpy flows with that value. This would mean, though, that on the other side the assumed enthalpy flows do not correspond with the actual mass flows, which can lead to severe errors in the calculation, especially when the mass flows are small. In the reference implementation this is circumvented by calculating the energy balance separately for the feed and permeate side – each time using the transmembrane flux calculated from that side's mole flows. This means that the energy balance is fulfilled inside each cell on both the feed and permeate side, but might not be fulfilled across the membrane. The amount of enthalpy "leaving" the feed side might not be equal to the amount of enthalpy "entering" the permeate side (and vice versa). Once the mass balance closes, the energy balance will close as well.

2. Heat transfer is subject to similar issues as mass transfer when being discretized: the transferred heat influences the temperature and therefore its own driving force. In contrast to the transmembrane flux, though, only one quantity (enthalpy) is transferred. As this leads to simpler equations, direct formulae to account for this effect are given in VDI (2013).

> These corrections, though, are designed in pure heat transfer scenarios and do not deliver accurate results when being applied simultaneously with calculation enthalpy transported with mass. Therefore, both effects have to be considered separately. Furthermore, heat transfer directly depends on the temperature difference as driving force, but enthalpy transport with mass acts on the enthalpies. In order to apply both calculations after each other, a linking step between temperatures and enthalpies is required.

In contrast to the flux and pressure drop calculations, here often both values of cell contents and borders of some properties are used. As introduced in section 3.2.4, an index noted as $\langle i \rangle$ refers to the value of the cell contents and is subject to the chosen method for selection of the evaluation location. Plain indices i always refer to the value at the downstream border of a cell (as it is stored).

The discretization of the governing conservation equations (3.44) is then given as follows:

$$\dot{H}_{F,i} = \begin{cases} \dot{H}_{F,i-1} - \dot{q}_{F,i} & \text{for } 2 \leq i \leq n-1 \\ \dot{H}_{F,i-1} & \text{for } i = n \end{cases} \tag{3.63}$$

$$\dot{H}_{P,i} = \begin{cases} \dot{H}_{P,i-1} + \dot{q}_{P,i} & \text{for } 2 \leq i \leq out-1 \\ \dot{H}_{P,i-1} - \dot{H}_{P,i+1} + \dot{q}_{P,i} & \text{for } i = out \\ \dot{H}_{P,i+1} - \dot{q}_{P,i} & \text{for } out+1 \leq i \leq n-1 \end{cases} \tag{3.64}$$

where

$$\dot{q}_{F,i} = \dot{q}'_{F,i}\,\Delta l = \dot{q}_{\overrightarrow{FP},F,i} - \dot{q}_{\overrightarrow{PF},F,i} + \dot{q}_{h,i} \tag{3.65}$$

$$\dot{q}_{P,i} = \dot{q}'_{P,i}\,\Delta l = \dot{q}_{\overrightarrow{FP},P,i} - \dot{q}_{\overrightarrow{PF},P,i} + \dot{q}_{h,i} \tag{3.66}$$

The enthalpy flows are connected to other stream properties as described in the governing equations:

$$\dot{H}_{F,i} = \dot{n}_{F,i} \cdot {h^m}_{F,i} \qquad\qquad \dot{H}_{P,i} = \dot{n}_{P,i} \cdot {h^m}_{P,i} \tag{3.67}$$

$${h^m}_{F,i} = h^m\left(p_{F,i}, T_{F,i}, x_{F,i,1}, \cdots, x_{F,i,m}\right) \tag{3.68}$$

$${h^m}_{P,i} = h^m\left(p_{P,i}, T_{P,i}, x_{P,i,1}, \cdots, x_{P,i,m}\right) \tag{3.69}$$

$$T_{F,i} = T\left(p_{F,i}, {h^m}_{F,i}, x_{F,i,1}, \cdots, x_{F,i,m}\right) \tag{3.70}$$

$$T_{P,i} = T\left(p_{P,i}, {h^m}_{P,i}, x_{P,i,1}, \cdots, x_{P,i,m}\right) \tag{3.71}$$

First the expressions for enthalpy transported with mass flow ($\dot{q}_{\overrightarrow{FP},F,i}$, $\dot{q}_{\overrightarrow{PF},F,i}$, $\dot{q}_{\overrightarrow{FP},P,i}$, $\dot{q}_{\overrightarrow{PF},P,i}$) are covered. For the reasons stated above, these are calculated separately for the feed and the permeate side. Corresponding to the governing equations, the discretization for the feed side is as follows:

$$\dot{q}_{\overrightarrow{FP},F,i} = \dot{Q}_{\overrightarrow{FP},F,i} \cdot {h^m}_{\overrightarrow{FP},F,i} \qquad \dot{q}_{\overrightarrow{PF},F,i} = \dot{Q}_{\overrightarrow{PF},F,i} \cdot {h^m}_{\overrightarrow{PF},F,i} \tag{3.72}$$

where

$$\dot{Q}_{F,j,i} = \begin{cases} \dot{n}_{F,j,i-1} - \dot{n}_{F,j,i} & \text{for } 2 \leq i \leq n-1 \\ 0 & \text{for } i = n \end{cases} \tag{3.73}$$

$$\dot{Q}_{\overrightarrow{FP},F,j,i} = \max\left(0,\ \dot{Q}_{F,j,i}\right) \qquad \dot{Q}_{\overrightarrow{PF},F,j,i} = \min\left(0,\ \dot{Q}_{F,j,i}\right) \tag{3.74}$$

$$\dot{Q}_{\overrightarrow{FP},F} = \sum_{j=1}^{m} \dot{Q}_{\overrightarrow{FP},F,j,i} \qquad \dot{Q}_{\overrightarrow{FP},F} = \sum_{j=1}^{m} \dot{Q}_{\overrightarrow{PF},F,j,i} \tag{3.75}$$

The properties defining the molar enthalpy h^m are defined according to the governing equations (3.50) to (3.52).

$${h^m}_{\overrightarrow{FP},F,i} = h^m\left(p_{F,\langle i\rangle},\ T_{F,\langle i\rangle},\ x_{\overrightarrow{FP},F,1,i},\ \cdots,\ x_{\overrightarrow{FP},F,m,i}\right) \tag{3.76}$$

$${h^m}_{\overrightarrow{PF},F,i} = h^m\left(p_{P,\langle i\rangle},\ T_{P,\langle i\rangle},\ x_{\overrightarrow{PF},F,1,i},\ \cdots,\ x_{\overrightarrow{PF},F,m,i}\right) \tag{3.77}$$

$$x_{\overrightarrow{FP},F,j,i} = \frac{\dot{Q}_{\overrightarrow{FP},F,j,i}}{\dot{Q}_{\overrightarrow{FP},F,i}} \qquad x_{\overrightarrow{PF},F,j,i} = \frac{\dot{Q}_{\overrightarrow{PF},F,j,i}}{\dot{Q}_{\overrightarrow{PF},F,i}} \tag{3.78}$$

The permeate side is discretized analog to the feed side:

$$\dot{q}_{\overrightarrow{FP},P,i} = \dot{Q}_{\overrightarrow{FP},P,i} \cdot {h^m}_{\overrightarrow{FP},P,i} \qquad \dot{q}_{\overrightarrow{PF},P,i} = \dot{Q}_{\overrightarrow{PF},P,i} \cdot {h^m}_{\overrightarrow{PF},P,i} \tag{3.79}$$

$$\dot{Q}_{P,j,i} = \begin{cases} \dot{n}_{P,j,i} - \dot{n}_{P,j,i-1} & \text{for } 2 \leq i \leq out-1 \\ \dot{n}_{P,j,i+1} + \dot{n}_{P,j,i} - \dot{n}_{P,j,i-1} & \text{for } i = out \\ \dot{n}_{P,j,i+1} - \dot{n}_{P,j,i} & \text{for } out+1 \leq i \leq n-1 \end{cases} \tag{3.80}$$

$$\dot{Q}_{\overrightarrow{FP},P,j,i} = \max\left(0,\ \dot{Q}_{P,j,i}\right) \qquad \dot{Q}_{\overrightarrow{PF},P,j,i} = \min\left(0,\ \dot{Q}_{P,j,i}\right) \tag{3.81}$$

$$\dot{Q}_{\overrightarrow{FP},P} = \sum_{j=1}^{m} \dot{Q}_{\overrightarrow{FP},P,j,i} \qquad \dot{Q}_{\overrightarrow{FP},P} = \sum_{j=1}^{m} \dot{Q}_{\overrightarrow{PF},P,j,i} \tag{3.82}$$

$${h^m}_{\overrightarrow{FP},P,i} = h^m\left(p_{F,\langle i\rangle},\ T_{F,\langle i\rangle},\ x_{\overrightarrow{FP},P,1,i},\ \cdots,\ x_{\overrightarrow{FP},P,m,i}\right) \tag{3.83}$$

$${h^m}_{\overrightarrow{PF},P,i} = h^m\left(p_{P,\langle i\rangle},\ T_{P,\langle i\rangle},\ x_{\overrightarrow{PF},P,1,i},\ \cdots,\ x_{\overrightarrow{PF},P,m,i}\right) \tag{3.84}$$

$$x_{\overrightarrow{FP},P,j,i} = \frac{\dot{Q}_{\overrightarrow{FP},P,j,i}}{\dot{Q}_{\overrightarrow{FP},P,i}} \qquad x_{\overrightarrow{PF},P,j,i} = \frac{\dot{Q}_{\overrightarrow{PF},P,j,i}}{\dot{Q}_{\overrightarrow{PF},P,i}} \tag{3.85}$$

In order to calculate heat transfer, the intermediary temperatures are required. First the intermediary enthalpy flows are calculated:

$$\dot{H}_{F,i,inter} = \begin{cases} \dot{H}_{F,i-1} - \dot{q}_{F,i,inter} & \text{for } 2 \leq i \leq n-1 \\ \dot{H}_{F,i-1} & \text{for } i = n \end{cases} \tag{3.86}$$

$$\dot{H}_{P,i,inter} = \begin{cases} \dot{H}_{P,i-1} + \dot{q}_{P,i,inter} & \text{for } 2 \leq i \leq out-1 \\ \dot{H}_{P,i-1} - \dot{H}_{P,i+1} + \dot{q}_{P,i,inter} & \text{for } i = out \\ \dot{H}_{P,i+1} - \dot{q}_{P,i,inter} & \text{for } out+1 \leq i \leq n-1 \end{cases} \tag{3.87}$$

with

$$\dot{q}_{F,i,inter} = \dot{q}_{\overrightarrow{FP},F,i} - \dot{q}_{\overrightarrow{PF},F,i} \qquad \dot{q}_{P,i,inter} = \dot{q}_{\overrightarrow{FP},P,i} - \dot{q}_{\overrightarrow{PF},P,i} \tag{3.88}$$

Then, using molar enthalpies, the intermediary temperatures can be calculated:

$$h^m{}_{F,i,inter} = \frac{\dot{H}_{F,i,inter}}{\dot{n}_{F,i}} \qquad h^m{}_{P,i,inter} = \frac{\dot{H}_{P,i,inter}}{\dot{n}_{P,i}} \tag{3.89}$$

$$T_{F,i,inter} = T\left(p_{F,i}, h^m{}_{F,i,inter}, x_{F,i,1}, \cdots, x_{F,i,m}\right) \tag{3.90}$$

$$T_{P,i,inter} = T\left(p_{P,i}, h^m{}_{P,i,inter}, x_{P,i,1}, \cdots, x_{P,i,m}\right) \tag{3.91}$$

The product of overall heat transfer coefficient and membrane area (3.58) is discretized for a specific cell as:

$$(\alpha A)_i = \frac{\Delta l \, n_f \, \pi}{\left(\alpha_{b,i} D_I\right)^{-1} + \frac{\ln\left(D_O/D_I\right)}{\lambda_f} + \left(\alpha_{s,i} D_O\right)^{-1}} \tag{3.92}$$

For **bore side feed**, heat transfer coefficients $\alpha_{b,i}$ and $\alpha_{s,i}$ are defined as

$$\alpha_{b,i} = \frac{\mathrm{Nu}_b \lambda_{F,\langle i \rangle}}{D_I} \qquad \alpha_{s,i} = \frac{\mathrm{Nu}_s \lambda_{P,\langle i \rangle}}{(D_M - D_O)} \tag{3.93}$$

and for **shell side feed** as:

$$\alpha_{b,i} = \frac{\mathrm{Nu}_b \lambda_{P,\langle i \rangle}}{D_I} \qquad \alpha_{s,i} = \frac{\mathrm{Nu}_s \lambda_{F,\langle i \rangle}}{(D_M - D_O)} \tag{3.94}$$

Under the flow conditions outlined in section 3.5.1, the Nusselt numbers are constant and can be taken directly from (3.53) and (3.54).

The limit of heat transfer described above is applied by correcting the temperature difference with a factor Θ. The details are described in Chapter C1 of *VDI-Wärmeatlas* (VDI 2013, pp. 37 sqq.) using multiple variables and equations. For the assumptions made in the algorithm (constant properties in each cell), the equivalent model is a two-sided stirred tank. Using equations 8, 11, 13, 16 and Table 2 of Chapter C1 of *VDI-Wärmeatlas*, a compact equation for a cell specific Θ is:

$$\Theta_i = \frac{1}{1 + \frac{(\alpha A)_i}{\dot{W}_{F,\langle i \rangle}} + \frac{(\alpha A)_i}{\dot{W}_{P,\langle i \rangle}}} \tag{3.95}$$

with

$$\dot{W}_{F,i} = c^{m}_{p\,F,i} \cdot \dot{n}_{F,i} \tag{3.96}$$

$$\dot{W}_{P,i} = c^{m}_{p\,P,i} \cdot \dot{n}_{P,i} \tag{3.97}$$

In contrast to the enthalpies transported with mass, the overall heat transfer is equal for both the feed and permeate side and can be calculated as:

$$\dot{q}_{h,i} = (\alpha A)_i \cdot \Theta_i \cdot \left(T_{F,\langle i \rangle,inter} - T_{P,\langle i \rangle,inter}\right) \tag{3.98}$$

As now all expressions in equations (3.63) to (3.66) are defined, the enthalpy flows can be calculated. Using the relations in equations (3.67) to (3.71), these enthalpy flows can be converted back to the corresponding temperatures. Thus the discretization is complete.

3.6 Full Algorithm

3.6.1 Definition of the Problem

As stated in section 2.2 and further described in section 3.3.1, the governing DEs for transmembrane flux are a system of nonlinear first order ODEs. Although the simple case of co-current flow (without any pressure drop calculation) would result in an IVP, in general a BVP has to be solved. Section 3.1.1 shows that a BVP cannot be solved directly and that some sort of iterative solution method has to be applied. For the DEs describing pressure drop and energy balance, the case is similar. Pressure drop is nonlinear due to the fact that the volume flow $\dot{V}$ depends on the pressure in a nonlinear way even for ideal gases. The energy balance equations, actually could result in linear equations in the simplest case (constant heat capacities, linear correlation of enthalpy and temperature), but in general a nonlinear system has to be assumed as well.

To summarize, a BVP on a complex set of generally nonlinear DEs has to be solved. This algorithm employs the FDM as described in section 3.1.4 to discretize the problem into a defined number of cells. In order to solve the resulting (nonlinear) equation system, the Jacobi method is used (see section 3.1.6). As the underlying equation system is nonlinear in nature, it is typically required to under-relax the iteration in order to achieve a stable solution. The process of under-relaxation is straightforward and applied as described in section 3.1.6. In terms of nomenclature, the direct results of the evaluation are marked with a hat (e.g. $\hat{\dot{n}}^{k+1}$) whereas the relaxed results are written without additional accents (e.g. $\dot{n}^{k+1}$).

Additionally, it has to be noted, that the calculation is split into the three separate parts transmembrane flux, pressure drop, and energy balance. This was implemented mainly due to practical reasons, as this way the calculations are independent entities that can be called individually. The choice of separating the three parts has some implications on the solution behavior though: Each of the three calculations generally depends on all basic properties (molar flows, pressure, and temperature). While the first performed calculation (in the reference implementation this is transmembrane flux) uses the actual values at the start of any given iteration, the other subsequent calculations do not use the "virgin" values, but rather the ones already updated by the previous calculation(s). It is assumed that this situation is actually beneficial for the convergence speed, but this could not be validated so far.

3.6.2 Initialization

The initialization procedure of the algorithm defines that every cell on the feed side is initialized with the molar flows, pressure, and temperature of the feed. The cells on the permeate side are initialized with the pressures and temperature of the permeate outlet. For the molar flows on the permeate side, the values of the "left" (feed side) sweep gas are used in the co-current part, whereas the flows of the "right" (permeate side) sweep gas are used to initialize the counter-current part. If no sweep gas is configured, the permeate side is initialized with zero molar flows for each component.

Early advances to find better initializations by, e.g., using simple formulas to estimate the transmembrane flux or interpolating based on the solution of a case with very few cells did not improve the convergence behavior. Both trivial and more sophisticated initialization procedures led to a converged result in a very similar time. As trials have also shown that the algorithm is fairly robust regarding the initial state, the decision was made to implement a very simple initialization procedure.

3.6.3 Solution Procedure

Based on the single calculation steps defined in the previous sections, now a complete solution procedure can be assembled. The full procedure is visualized in

Fig. 3.4. Two different starting conditions are possible:

- Initial calculation without previous results
- Continuing a previous calculation calculation with possibly changed boundary conditions

The first case is typical when starting new calculations. There it is required to initialize the grid based on the boundary conditions before the actual solution process starts. An approach for initialization is described in section 3.6.2.

The second case can occur when a previous calculation exited, but not necessarily with the final results. This may be the case when it is desirable to limit the number of iterations in one run. A typical situation where this happens is the simulation of multi-stage processes where multiple single membrane modules have to be simulated (see e.g. section 4.8). It is possible that the boundary conditions have changed since the previous calculation. As long as these changes are not substantial, it is probably faster to continue with the result of the previous calculation while enforcing the new boundary conditions during the following iterations. Therefore, it is required to update the boundary conditions based on the external values.

Once the model is in a state that reflects the currently applicable boundary conditions, the first step is to calculate the fluxes as described in in section 3.3. Based on the results of the step, the current convergence value is updated:

$$\delta\left(\dot{n}_{F,j,i}\right) = |\hat{\dot{n}}^{k+1}_{F,j,i} - \dot{n}^{k}_{F,j,i}| \qquad \delta\left(\dot{n}_{P,j,i}\right) = |\hat{\dot{n}}^{k+1}_{P,j,i} - \dot{n}^{k}_{P,j,i}| \tag{3.99}$$

Afterwards, relaxation (see section 3.1.6) is applied to the results.

Next, pressure drop calculation as described in section 3.4 is performed if it is enabled. Also for the updated pressure values the convergence value is calculated:

$$\delta\left(p_{F,i}\right) = |\hat{p}^{k+1}_{F,i} - p^{k}_{F,i}| \qquad \delta\left(p_{P,i}\right) = |\hat{p}^{k+1}_{P,i} - p^{k}_{P,i}| \tag{3.100}$$

Relaxation is also applied to the results of the pressure drop calculation.

The following step is calculating the energy balance as defined in section 3.5 if enabled. Convergence is also updated for the resulting temperatures:

$$\delta\left(T_{F,i}\right) = |\hat{T}^{k+1}_{F,i} - T^{k}_{F,i}| \qquad \delta\left(T_{P,i}\right) = |\hat{T}^{k+1}_{P,i} - T^{k}_{P,i}| \tag{3.101}$$

The results of the energy balance calculation are then also relaxed. Once all values are calculated, it is checked whether defined convergence criteria ε are fulfilled for all calculations:

$$\varepsilon\left(\dot{n}\right) \geq \max_{\substack{2\leq i\leq n-1 \\ 1\leq j\leq k}} \left(\delta\left(\dot{n}_{F,j,i}\right), \delta\left(\dot{n}_{P,j,i}\right)\right) \tag{3.102}$$

$$\varepsilon\left(p\right) \geq \max_{2\leq i\leq n-1} \left(\delta\left(p_{F,i}\right), \delta\left(p_{P,i}\right)\right) \tag{3.103}$$

$$\varepsilon\left(T\right) \geq \max_{2\leq i\leq n-1} \left(\delta\left(T_{F,i}\right), \delta\left(T_{P,i}\right)\right) \tag{3.104}$$

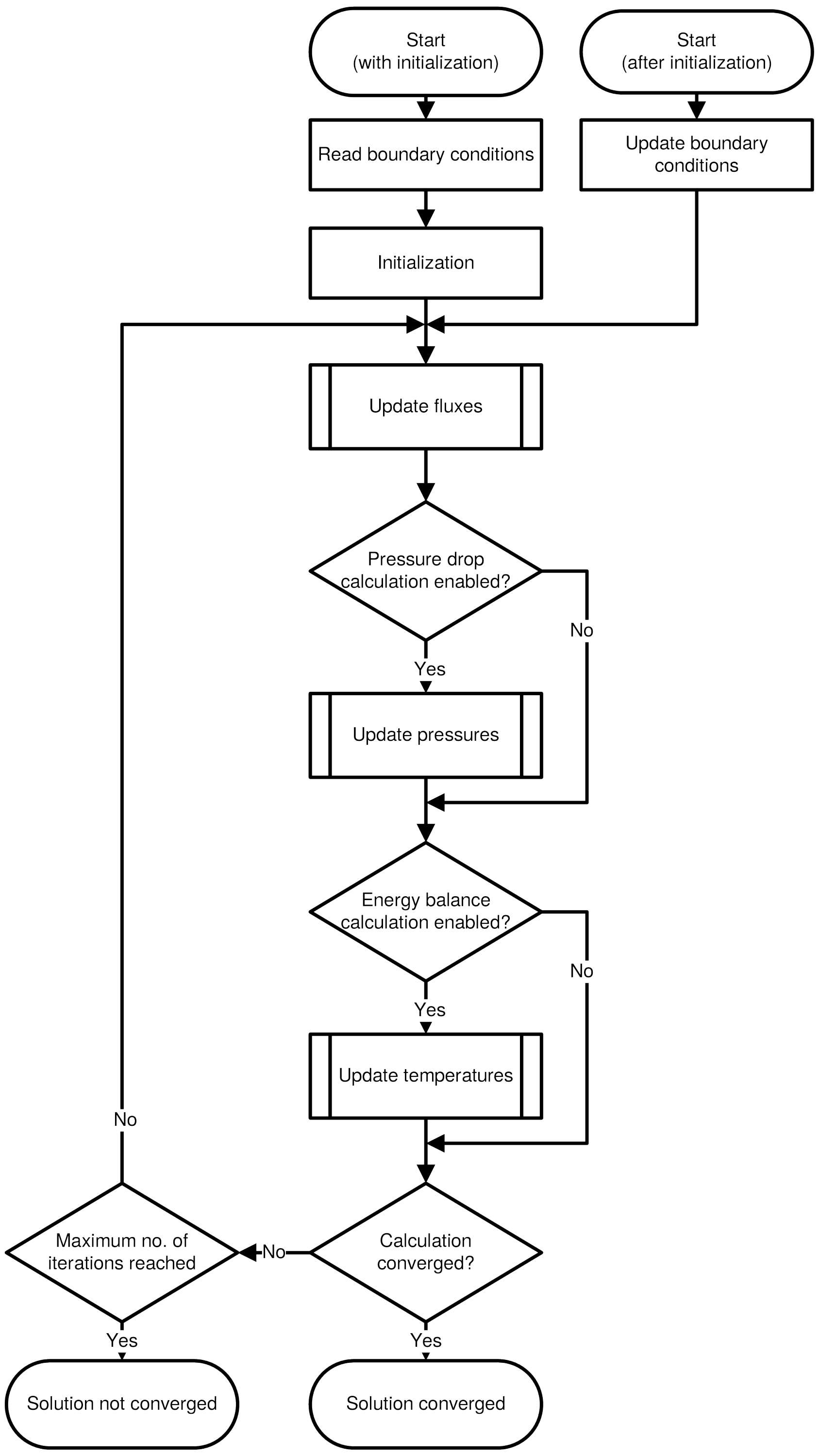


Figure 3.4: Flowchart of the Solution Procedure

Additionally, a check for a relative convergence of the flux (compared to the feed) is possible as well as a check for a closed mass balance:

$$\varepsilon\left(\dot{n}_{rel}\right) \geq \max_{\substack{2\leq i\leq n-1\\1\leq j\leq k}} \left(\frac{\delta\left(\dot{n}_{F,j,i}\right)}{\dot{n}_{F,j,1}}, \frac{\delta\left(\dot{n}_{P,j,i}\right)}{\dot{n}_{F,j,1}}\right) \tag{3.105}$$

$$\begin{aligned}\varepsilon\left(\dot{n}_{bal}\right) &\geq \max_{1\leq i\leq k} \left|\left(\dot{n}_{Perm,j} + \dot{n}_{Ret,j}\right) - \left(\dot{n}_{Feed,j} + \dot{n}_{Sweep_0,j} + \dot{n}_{Sweep_1,j}\right)\right| \\ &= \max_{1\leq i\leq k} \left|\left(\dot{n}_{P,j,out} + \dot{n}_{F,j,n}\right) - \left(\dot{n}_{F,j,1} + \dot{n}_{P,j,1} + \dot{n}_{P,j,n}\right)\right|\end{aligned} \tag{3.106}$$

If all criteria are fulfilled, the algorithm exits with a converged solution. Otherwise, it is checked whether the maximum number of iterations has been reached. If this is the case, the algorithm exits without a converged solution. If not, the next iteration is started by returning to the calculation of transmembrane flux.

4 Implementation Details

4.1 Adaption of Under-Relaxation

As stated in section 3.6.1, the underlying DES are nonlinear and therefore the Jacobi method typically requires under-relaxation to reach convergent solutions. The algorithm implements this as a core feature and allows for a custom selection of the under-relaxation factor. This comes with a trade-off, though, as low relaxation factors typically lead to stable solutions, but at the same time drastically reduce the convergence speed (see section 3.1.6). It is therefore advisable to select the highest possible relaxation factor that still leads to a stable and convergent solution. The choice of such a value is not trivial, though, as the maximum stable value depends heavily on the specific case and varies greatly with different input parameters.

An example of unfavorable convergence behavior can be observed in a simple gas permeation calculation (with disabled pressure drop and energy balance calculation). Changing the module area (in order to reach a target stage cut) and keeping all other parameters constant, the solution procedure might easily converge with a relaxation factor of $\omega = 0.9$ for target stage cuts of 0.4 and 0.6. For a target stage cut of 0.5, though, a lower relaxation factor (around $\omega = 0.5$) might be required to reach a target convergence criterion. The reasons for this behavior might be manyfold and often not directly observable. One effect could be identified though: In (partial) counter-current flow configurations, regions with oscillating molar fractions can occur. A generic version of the impact of this effect is shown in Fig. 4.1.

Looking only at the results of one iteration, it can be observed that while the molar fraction for this component generally decreases along the module (with increasing cell numbers), there are alternating cells with high and low molar fractions in the downstream part of the module. Even more so, with every single iteration, the highs and lows switch positions. This behavior definitely does not represent the reality — there are no sudden jumps of molar fractions in real gas permeation modules. In an attempt to explain the behavior, it can be assumed that substance circulates numerically between cells. In cells with a high partial pressure on the feed side and low partial pressure, the substance permeates from the feed to the permeate side. The neighboring cell shows exactly the opposite behavior: a low molar fraction (and therefore partial pressure) on the feed side and a high one on the permeate side, thus resulting in permeation from the permeate

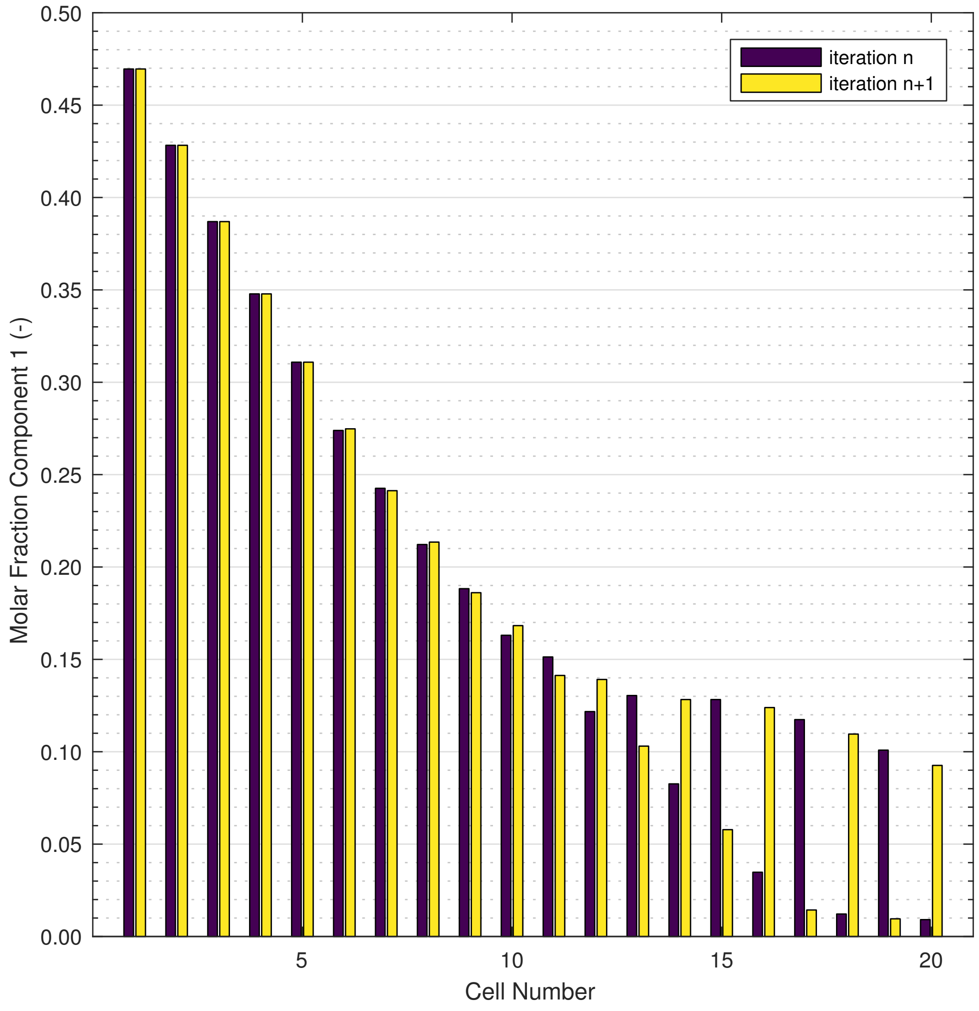


Figure 4.1: Exemplary Oscillation of Molar Fractions in a Two-Component Case

to the feed side. It can easily be reasoned, that by “dampening” the amount of substance permeating, this numerical misbehavior can be suppressed. This is where under-relaxation comes into play, as it does provide exactly this numerical “dampening”.

It can be observed though, that there is a rather sharp threshold value for the under-relaxation factor. As long as the factor is higher than the threshold, the convergence behavior is not significantly improved. After setting an under-relaxation factor below the threshold, though, the oscillating behavior disappears within few iterations. In order to reach optimal calculation speeds, the main challenge is to find this threshold value.

Ideally, the relaxation factor would be calculated analytically from the parameters of the case to calculate. As stated above, though, this is not easily possible, as small changes of the input parameters can lead to largely differing threshold values of the relaxation factor. Another approach would be to reduce the under-relaxation factor after a set number of iterations if convergence could not be reached. While this approach leads to convergent solutions eventually, there is an efficiency issue:

- When setting the number of iterations too high, cases with numerical issues may iterate “on the spot”, i.e. without any significant progress, for many iterations.
- Setting the number too low adversely affects well-behaved cases. These cases would reach convergent solutions with the higher under-relaxation factor, but due to the factor being lowered, convergence speed is reduced significantly.

In order to circumvent these issues, the reference implementation (denoted as VECGP) uses heuristic measures to identify situations where convergence is stalled. Only if this is the case, the under-relaxation factor is reduced. Then the reduction happens rather quickly, until a favorable convergence behavior is reached again. The indicator used to determine whether a system is converging accurately or running into numerical issues (like the oscillating molar fractions) is the rate of change of the convergence values. The rate of change is the difference between the convergence values (as defined in section 3.6.3) of two consecutive iterations. As long as the convergence values decrease with iterations, the system converges well. Increasing convergence values, in contrast, indicate issues in the calculation and are a sign that the chosen under-relaxation factor is too high.

There are, though, situations where temporarily increasing convergence values have to be tolerated: Both during the first iterations after initialization and after changing the under-relaxation factor, properties in the cells may change significantly. In these cases, using a single increasing convergence value as trigger to reduce the relaxation factor would lead to premature reduction. Therefore, a minimum number of iterations without changing the under-relaxation factor after initialization and after relaxation factor changes can be set.

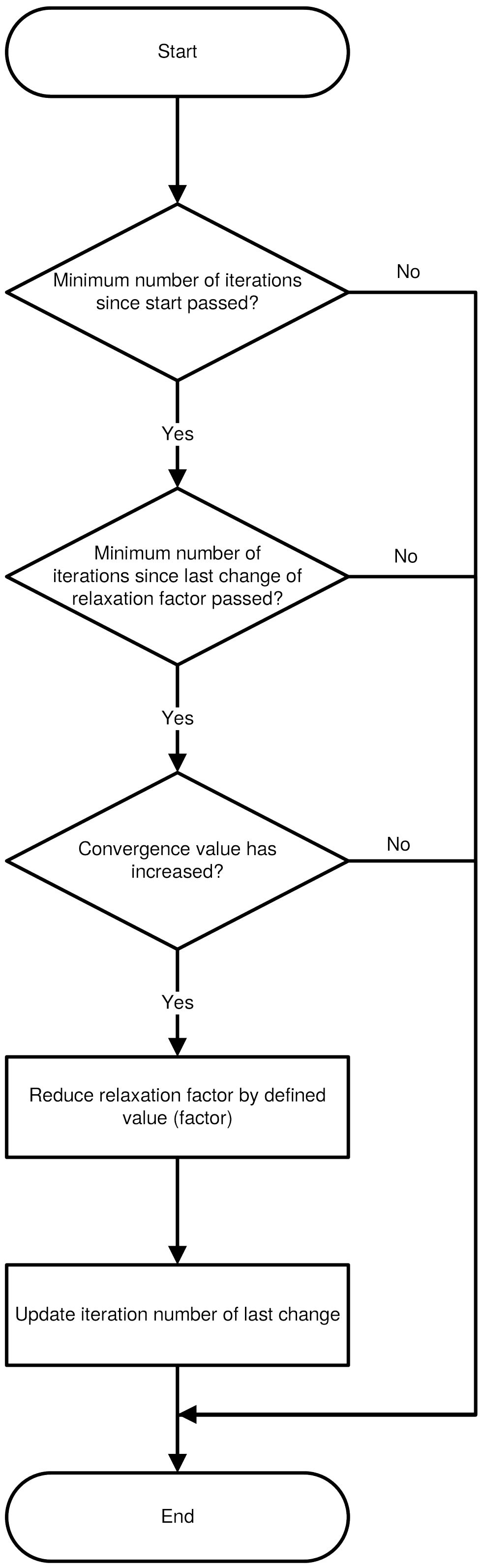

Start
Minimum number of iterations since start passed?
No
Yes
Minimum number of iterations since last change of relaxation factor passed?
No
Yes
Convergence value has increased?
No
Yes
Reduce relaxation factor by defined value (factor)
Update iteration number of last change
End

Figure 4.2: Flowchart of the Algorithm for Reducing the Under-Relaxation Factor

The full algorithm for deciding whether a reduction of the under-relaxation factor is required is shown in Fig. 4.2. First it is checked if a minimum number of iterations has passed since initialization and since the last reduction of the under-relaxation factor. If both conditions are fulfilled and the convergence value has increased compared to the previous iteration, the under-relaxation factor is decreased and the number of the iteration where the last reduction occurred is updated.

Veccp treats under-relaxation separately for flux, pressure drop, and energy balance calculations. All three sets of calculations can use different under-relaxation factors and parameters to configure the reduction. Trial runs have shown that this approach leads to reliably converging solutions even for complex cases involving e.g. multiple components with real gas properties and enabled pressure drop and energy balance calculation.

Recently proposals have been made to define formalized patterns for changing under-relaxation factors using analytical reasoning. Yang and Mittal (2014) proposed a *Scheduled Relaxation Jacobi method* that attempts to find optimal schedules for the under-relaxation factor. This approach was further improved by Adsuara et al. (2016). Both publications focus on linear equations and employ far more sophisticated mathematical methods to determine optimal under-relaxation factors than used in the algorithm presented in this thesis. While both these approaches are not directly applicable to the cases handled in this thesis, they still show that changing the under-relaxation factor between iterations can significantly improve convergence speeds.

4.2 Calculation of Mixed Gas Properties

4.2.1 Basics of Property Calculation

As stated in Chapter 3, the only properties permanently stored by the algorithm are molar fractions, pressures, and temperatures. All other properties that are required by calculations have to be somehow derived from these basic properties. As gas permeation operations typically involve more than one substance, the calculations have to use properties of the mixed gases. The means by which to calculate properties are manyfold, and there are standard works like *The Properties of Gases and Liquids* (Poling et al. 2000) and *Prediction of Transport and Other Physical Properties of Fluids* (Bretsznajder 1971) that cover this topic thoroughly. For a practically usable implementation, some choice has to be made, how to implement property calculations.

In light of the fact that property calculations will be called thousands of times in typical calculation scenarios (properties are required for every cell in every iteration), it is required that the retrieval of a single property value is computationally cheap. Therefore, the decision has been made to support two distinct ways of calculating properties in the reference implementation: very simple equations assuming ideal gas behavior and interpolation in a given table of properties. Detailed descriptions of the equations used for the simple case are given in the following sections. When using the interpolation method, a table of property values for all possible combinations of molar fractions, pressures, and temperatures is required. VECGP is agnostic to the source of this data, as long as it is provided in the correct format. Possible sources are described in section 4.2.2. The size of the table varies greatly depending on the amount of data points. It severely depends on the number of pure substances and the resolution of the molar fractions, as all possible combinations have to be covered. The reference implementation extracts the data points from the property table and builds axes for each dimension (molar fractions of all but one component, pressure, temperature) as well as a grid for interpolation. The interpolation itself is performed by MATLAB's own `griddedInterpolant` class based on the defined axes and grid. This class allows for various settings like choosing the interpolation method (linear or spline-based) and whether to enable extrapolation or not. Additionally, it facilitates querying multiple points at once efficiently, which allows for fast calculations. This is described in detail in section 4.5.

4.2.2 Sources of Real Gas Properties

In general, there are to possible ways to obtain real gas properties: property databases and property models. Whereas databases typically provide access to a large set of measured data points, property models typically involve sets of equations using various parameters that can be fitted to optimally represent the real behavior of a substance. As gas permeation often deals with multiple components in various compositions it cannot be assumed that all these data points are available in a property database. While single component properties might be available, it is required to employ some kind of interaction model to calculate mixed gas properties based on the pure component properties.

While Poling et al. (2000) and Bretsznajder (1971) provide many different models to calculate various properties, these are not considered as state of the art today. A software package using state of the art equations is *REFPROP* by the National Institute of Standards and Technology (Lemmon et al. 2013). The open source library *CoolProp* (Bell et al. 2014) provides data of similar quality, but does not reliably cover as many substances, mixtures, and properties yet. One advantage of *CoolProp* is that it can either use its own property model or use *REFPROP* as a backend. Therefore, implementing an interface to *CoolProp* gives access to both property models. VECGP includes a Python script that systematically queries data points in *CoolProp* in order to generate a table of property values that can be used by the algorithm.

4.2.3 Fugacity Coefficients and Partial Pressures

Fugacity coefficients are different from all the other properties covered in the reference implementation. While they do depend on the composition of the mixture (i.e. the molar fractions of all components), they are not properties of the mixture itself, but rather properties of every single component in a mixture.

The simple case assumes ideal gas behavior where fugacity coefficients are defined as 1. Therefore the partial pressures are just calculated according to Dalton's law as the product of molar fraction and pressure:

$$p_j = p \cdot x_j$$

When using the interpolation variant, the fugacity coefficients are queried from the interpolation table and the resulting partial pressure is calculated as a product of the ideal gas partial pressure and the fugacity coefficient:

$$p_j = p \cdot x_j \cdot \varphi_{j,int}\,(p,\, T,\, x_1,\, \cdots,\, x_m)$$

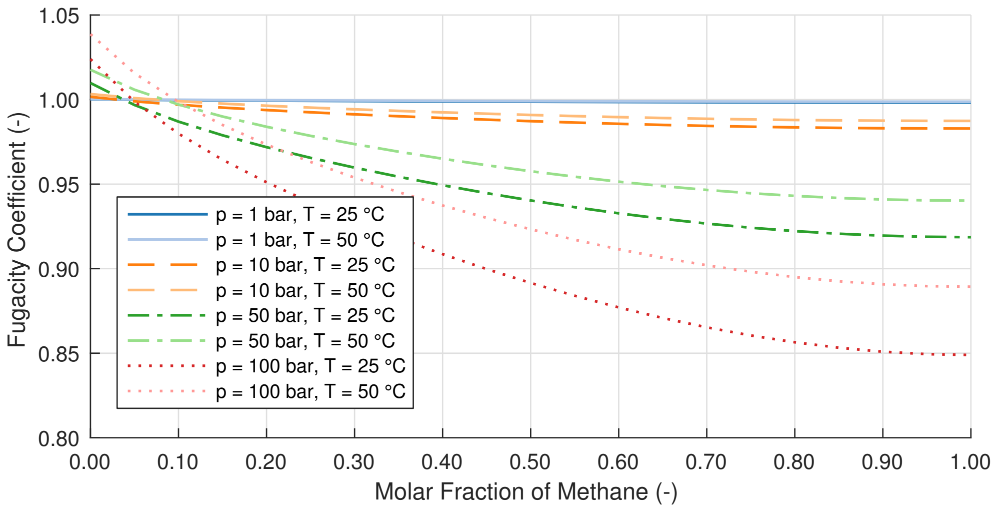
As shown in section 3.3, partial pressures play a central role in the calculation of transmembrane flux. Therefore, varying fugacity coefficients might have a significant impact on the overall results (e.g. stage cut, recoveries). The deviation of real gas fugacity coefficients from the ideal value of 1 depends heavily on the specific substances as well as temperature and pressure. In general, the deviation is low for low pressures and high temperatures. As an example, the fugacity coefficients (calculated using *REFPROP*) for mixtures of methane and hydrogen and mixtures of methane and carbon dioxide are shown in Fig. 4.3 and Fig. 4.4. Note that for the system of methane and carbon dioxide, only mixtures with a molar fraction of methane greater than 0.5 are shown, as pure carbon dioxide would be liquid at 25 °C and 100 bar.

It can be observed that the deviations reach significant values at high pressures, especially for hydrogen and carbon dioxide. In section 7.5, the differences between calculations with ideal partial pressures and real gas data are shown in some examples.

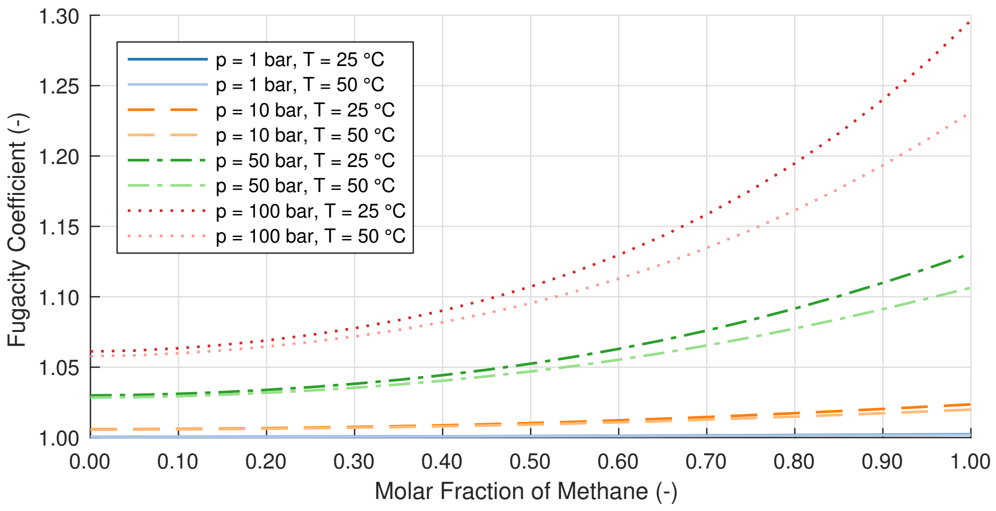
4.2.4 Molar Volume and Density

Molar volumes are required to calculate volume flows from mole flows and therefore needed for pressure drop calculations (section 3.4). For the simple case assuming ideal gases, the molar volume can be directly calculated from the ideal gas law:

$$v^m = \frac{R \cdot T}{p}$$

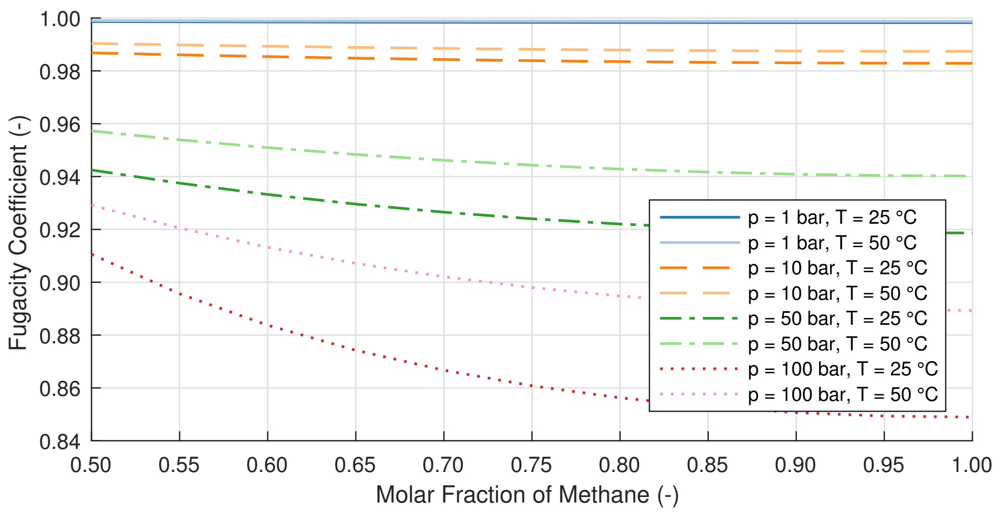


(a) Fugacity Coefficients of Methane

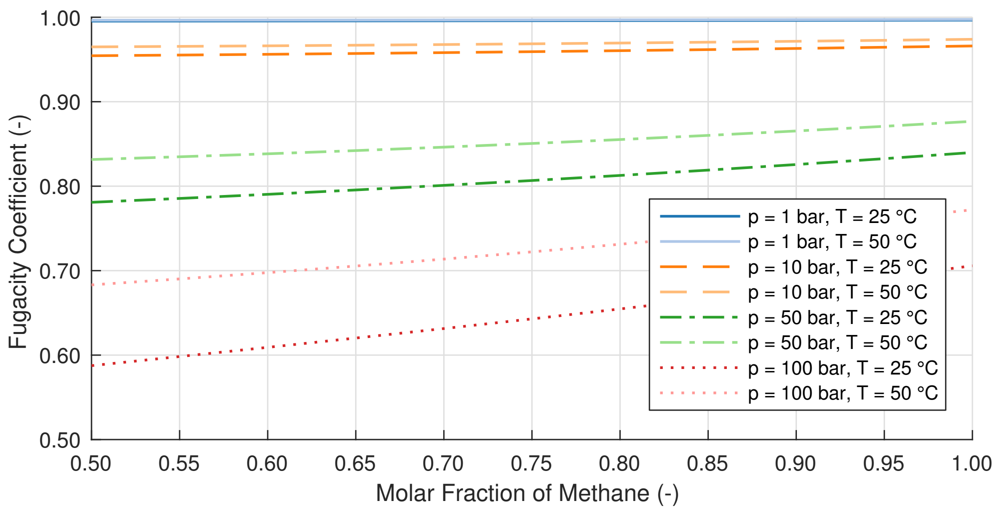


(b) Fugacity Coefficients of Hydrogen

Figure 4.3: Fugacity Coefficients for Mixtures of Methane and Hydrogen



(a) Fugacity Coefficients of Methane



(b) Fugacity Coefficients of Carbon Dioxide

Figure 4.4: Fugacity Coefficients for Mixtures of Methane and Carbon Dioxide

When using interpolation, molar volume is evaluated as the reciprocal value of molar density which is stored as part of the interpolation table:

$$v^m = \frac{1}{{\rho^m}_{int}\left(p, T, x_1, \cdots, x_m\right)}$$

Given the molar volume, the volume flow can be calculated as:

$$\dot{V} = \dot{n} \cdot v^m$$

Additionally, the density might be of interest. It is derived by dividing the molar mass of the mixture by the molar volume.

$$\rho = \frac{\sum\limits_{j=1}^{m} x_j M_j}{v^m}$$

When using interpolation, the molar mass of the mixture is stored in the table and therefore can be used directly:

$$\rho = M_{int}\left(p, T, x_1, \cdots, x_m\right) \cdot {\rho^m}_{int}\left(p, T, x_1, \cdots, x_m\right)$$

4.2.5 Viscosity

Viscosities are required for the pressure drop calculation (section 3.4) as well.

For the simple case it is assumed that the pure component viscosities μ_j are constant, neglecting temperature and pressure dependency. These constant values have to be defined manually. It has to be noted, that the viscosity is a function of temperature even under ideal gas assumptions (White 2005, p. 25). Therefore, the model of constant viscosities will only lead to acceptable results when no large variations of temperature are expected.

Based on the constant pure component viscosities, the viscosity of the mixed gas is calculated using the mixing formula of Wilke as described in equations 9-5.13 and 9-5.14 of *The Properties of Gases and Liquids* (Poling et al. 2000). In this formula a mixing coefficient ϕ_{j_1,j_2} is used which is defined as:

$$\phi_{j_1,j_2} = \frac{\left[1 + \left(\frac{\mu_{j_1}}{\mu_{j_2}}\right)^{\frac{1}{2}} \left(\frac{M_{j_2}}{M_{j_1}}\right)^{\frac{1}{4}}\right]^2}{\left[8\left(1 + \frac{M_{j_1}}{M_{j_2}}\right)\right]^{\frac{1}{2}}} \tag{4.1}$$

Using (4.1), the mixed gas viscosity is then calculated as:

$$\mu = \sum_{j_1=1}^{m} \frac{x_{j_1}\mu_{j_1}}{\sum\limits_{j_2=1}^{m} x_{j_2}\phi_{j_1,j_2}} \tag{4.2}$$

When using interpolation, the mixed gas viscosities can be taken directly from the interpolation table:

$$\mu = \mu_{int}\,(p, T, x_1, \cdots, x_m)$$

4.2.6 Thermal Conductivity

Thermal conductivities are required to calculate the heat transfer through the membrane (see section 3.5).

The simple calculation uses the formula of Eucken to calculate the pure component thermal conductivities based on the pure component viscosities. The formulation given in equation 10-3.3 of *The Properties of Gases and Liquids* (Poling et al. 2000) can be transformed into:

$$\lambda_j = \frac{\mu_j \cdot R}{M_j} \cdot \left(\frac{1}{\kappa_j - 1} + 2.25 \right) \tag{4.3}$$

The pure component heat capacity ratios κ_j are assumed to be constant and have to be specified manually.

The mixed gas thermal conductivities are then calculated according to the mixing formula of Mason and Saxena as given in equations 10-6.1 to 10-6.4 of *The Properties of Gases and Liquids*:

$$\lambda = \sum_{j_1=1}^{m} \frac{x_{j_1} \lambda_{j_1}}{\sum\limits_{j_2=1}^{m} x_{j_2} \phi_{j_1,j_2}} \tag{4.4}$$

The mixing coefficient ϕ_{j_1,j_2} is the same as for the calculation of the mixed gas viscosity and is specified in (4.1).

When using interpolation, the mixed gas thermal conductivities are again part of the interpolation table and can therefore be taken directly from there:

$$\lambda = \lambda_{int}\,(p, T, x_1, \cdots, x_m)$$

4.2.7 Enthalpy and Heat Capacities

Enthalpies and heat capacities are used throughout the energy balance calculation (section 3.5).

For the simple calculation, ideal gases with constant heat capacity ratios κ_j are assumed. The molar heat capacities can then be calculated as:

$$c_p^m = \sum_{j=1}^{m} \left(\frac{x_j \cdot \kappa_j}{\kappa_j - 1} \right) \cdot R \tag{4.5}$$

$$c_v^m = \sum_{j=1}^{m} \left(\frac{x_j}{\kappa_j - 1} \right) \cdot R \tag{4.6}$$

Based on the heat capacity at constant pressure c_p^m, the enthalpy can be simply calculated as:

$$h^m = c_p^m \cdot T \tag{4.7}$$

In order to calculate the temperature from a given enthalpy, the calculation just has to be inverted:

$$T = \frac{h^m}{c_p^m} \tag{4.8}$$

When using interpolation, the heat capacities as well as the molar enthalpy are stored in the interpolation table:

$$\begin{aligned} c_p^m &= c_{p\,int}^m \left(p, T, x_1, \cdots, x_m\right) \\ c_v^m &= c_{v\,int}^m \left(p, T, x_1, \cdots, x_m\right) \\ h^m &= {h^m}_{int} \left(p, T, x_1, \cdots, x_m\right) \end{aligned}$$

Calculating the temperature from a given molar enthalpy is not that trivial though, as the molar enthalpy cannot be looked up in the interpolation table. Using the fact that the partial derivative of the molar enthalpy in regard to the temperature is the molar heat capacity at constant pressure

$$\frac{\partial h^m}{\partial T} = c_p^m$$

and given a guess for the target temperature T_{guess}^k, a refined guess for the target temperature can be acquired by using the method of Newton:

$$T_{guess}^{k+1} = T_{guess}^k - \frac{{h^m}_{int}\left(p, T_{guess}^k, x_1, \cdots, x_m\right) - h^m}{c_{p\,int}^m\left(p, T_{guess}^k, x_1, \cdots, x_m\right)} \tag{4.9}$$

In VECGP, the temperature of the previous iteration (or the initialization temperature in the first iteration) is used as initial guess. If the initial guess is not too far off, this iteration typically leads to accurate results in very few (1-3) iterations.

4.3 Delayed Activation of Pressure Drop and Energy Balance Calculation

While all three modes of calculation (transmembrane flux, pressure drop, and energy balance) are interrelated, the dependency of the latter two on the first is much higher than the other way round. The transmembrane flux calculation is directly responsible for calculating the flows in each cell and this has a direct impact on the volume flows used in the pressure drop calculation and the enthalpy flows used in the energy balance calculation. As long as permeances are assumed pressure and temperature independent, the effect of changed pressures and temperatures on the flux calculation is limited.

This leads to the situation that both the pressure drop and the energy balance calculation require *reasonably* correct molar flow values in order to deliver sensible results. Therefore, it might be practical to approach a calculation case by first only solving the transmembrane flux equations and only activating the other calculations once convergence has been reached for the molar flows. Furthermore, it can be argued that the effect of the pressure drop calculation is greater than that of the energy balance calculation, as it directly affects the driving force in the flux calculation. The energy balance calculation, on the other hand, only affects the other calculations indirectly via the used properties. In light of this "order of dependency", a calculation in three steps might show favorable convergence behavior:

1. Only transmembrane flux calculation
2. Transmembrane flux and pressure drop calculation
3. Transmembrane flux, pressure drop, and energy balance calculation

4.4 Numerical Effects

When dealing with numerical methods, one always has to consider numerical effects that might lead to unexpected solution behavior. Two situations that commonly appear are divisions by zero and effects of the numerical accuracy. Divisions by zero often appear when a quantity is divided by a sum of quantities, as it is the case for the calculation of the molar fractions based on molar flows:

$$x_j = \frac{\dot{n}_j}{\sum\limits_{j=1}^{m} \dot{n}_j} \tag{4.10}$$

In the case that all $\dot{n}_j$ are zero, this equation leads to a $\frac{0}{0}$ which is mathematically undefined. If one tries to evaluate this in MATLAB, the result is `NaN` (Not a Number). The calculation is adversely affected by the appearance of `NaN` values, as they spread virally, i.e. every mathematical operation that involves a `NaN` also results in a `NaN`. This means that once a single `NaN` appears, after a few iterations typically all values become `NaN`s. Therefore it can easily be seen that these `NaN`s have to be avoided by any means possible. For equations such as (4.10), the desired result is typically zero when all values are zero. In VECGP checks are implemented throughout the code that detect these $\frac{0}{0}$ cases and replace them with zero.

The other case where positive or negative numbers are divided by zero typically does not lead to as severe problems, as these calculations result in values of positive or negative infinity which are mostly handled fine by subsequent mathematical operations.

The second big area of unexpected behavior is due to the limited numerical accuracy present in calculation environments. Typical double precision float representations have a relative accuracy of around 10^{-16}. This means that for any x, values between $x \cdot (1 - 10^{-16})$ and $x \cdot (1 + 10^{-16})$ cannot be differentiated by the numerical representation. This is especially important where values have to obey exact limits.

One example is the flux limiting applied as part of the transmembrane flux calculation as described in section 3.3.4. When applying the soft limit according to (3.29), it is not possible for the limited flux to reach a value higher than the reference value as long as exact mathematics can be assumed. Due to numerical effects, though, the result may be larger by a small margin within the numerical accuracy. This might lead to the situation that subtracting the soft-limited flux from the reference value results in a negative value. If this shift of sign happens, it adversely affects all subsequent calculations. Therefore, this situation has to be avoided. In the described situation, the case is handled in the reference implementation by also applying a hard limit after the soft limit.

Another case where numerical accuracy plays a role is the energy balance calculation as described in section 3.5. This calculation mainly operates on enthalpy flows which are calculated based on the molar flows, pressure, and temperature in the beginning. After the calculation, the resulting enthalpy flows are divided by the molar flows in order to obtain molar enthalpies which are subsequently converted back to temperatures. One problem that could arise is division by zero if the original molar flows are zero. Additionally, though, molar flows that are very small — to be exact: indistinguishable from zero within numerical accuracy using the largest volume flows as reference — can lead to problems when dividing the enthalpy flows by them. Therefore, in these cells the energy balance calculation has to be overridden. The phenomenological reasoning behind this is straightforward: if no mass is transported through a cell, then there is nothing that could transport enthalpy.

4.5 Performance Optimization

In order to optimize the performance of any code, the environment in which it is executed has to be taken into account. As the reference implementation is implemented in MATLAB, the specifics of this programming environment have to be considered. MATLAB is optimized for performing calculations on matrices and vectors, whereas loops are comparatively slow (MathWorks 2016). Therefore a goal of VECGP was to vectorize the code as much as feasibly possible — hence the name.

As a result, all the properties are not only stored in arrays covering all cells and components, but calculations are also performed on all cells simultaneously using matrix operations. Due to this vectorization it is not possible to use the Gauss-Seidel method as described in section 3.1.5, as this would require the cells to be calculated one after the other. Thus, VECGP only uses the Jacobi method.

Additionally, all the code dealing with mixed gas properties is vectorized as well. This means that the properties can also be calculated simultaneously for all cells. This also applies to the interpolation method, as a vector of query points can be passed to the used `griddedInterpolant` class.

Other means of performance optimization that are implemented are preallocation of memory for variables, caching of frequently repeated calculations, and inlining of frequently called code.

4.6 Structure of Code

The primary aim of VECGP is to allow easy, robust, and fast calculations of gas permeation processes in hollow fiber membrane modules. Additionally, it was a goal to make the helper routines (properties, handling of streams, etc.) reusable outside the pure calculation of gas permeation processes. In order to allow an easy set up of multi-stage calculations, VECGP was also designed to allow easy connection of single gas permeation modules. The code is self contained and does not have any external dependencies besides MATLAB. Most of the testing has been done with MATLAB R2016b, but to best knowledge it should also work with versions back to at least R2015a.

VECGP is coded as an object oriented library consisting of seven classes, each of which performs distinct tasks:

`Component` represents a single pure component and holds its properties like name, molar mass, (constant) viscosity, and constant heat capacity ratio (isentropic exponent).

`Components` represents a set of components. Therefore, it holds all the pure components that the set of components is made up from. The main use of this class is to provide an easy way to access mixed gas properties. As described in section 4.2, the properties can be derived in two ways:

- Calculation from constant properties of the pure components assuming ideal gases and using simple semi-empirical equations.
- Interpolation of properties in a provided grid. This allows for easy use of real gas properties for all parts of the calculation.

`AbstractStream` is an abstract class (which means that it just serves as a parent for inheritance) that provides methods to access properties of streams. It accesses `Components` to calculate said properties. Classes inherited from `AbstractStream` (`Stream` and `CalculationStream`) make use of these generally defined methods.

`Stream` represents a single stream, defining components, composition, flow, pressure and temperature. As the methods from `AbstractStream` are available, various properties of the stream can easily be accessed. Additionally, addition and subtraction of `Stream`s is possible as well as multiplication with a scalar.

`CalculationStream` is used internally by `Calculation` and represents a stream with multiple cells. Rather than storing total flows and composition, a `CalculationStream` stores the flows for each component for fast calculations. Total flow and composition are calculated by calling a caching method. Additionally, convergences of component flows, pressure, and temperature are stored.

`Module` represents a membrane module with all properties necessary to fully describe it. Besides storing the module geometry, the connected streams, and providing a method to calculate permeances, this class also allows easy access to module level properties like stage cut, recoveries, and mass balances. Besides setting streams, components, and parameters for calculation of permeances, the two important properties for regular flux calculations are the total membrane area and the location of the permeate outlet. As described in section 3.2.2, the outlet can be set as 0 (at feed side, counter-current), 1 (at retentate side, co-current), and all values in between. Values other than 0 and 1 are only possible when bore side feed is specified, as an outlet in the middle of the module is physically impossible with shell side feed. For the calculation of pressure drop and energy balance, additional geometry parameters (diameters, etc.) can be set.

`Calculation` actually performs the calculation. It holds a `Module` object, stores all settings necessary for calculation, and provides methods for initialization, single calculation steps, automated iteration loops, and status displays.

4.7 Sanitization of Inputs

In order to increase user-friendliness of the reference code, measures have been implemented to ensure “sane” inputs. This is typically done using the MATLAB method `validateattributes`. The checks include minimal and maximal values, especially positivity or non-negativity conditions. Additionally, it is enforced whether a number has to be an integer or can be a floating point number. The dimension of inputs is also checked to conform with the target value to be set.

All these measures are there to ensure, that attempted misconfigurations of the calculation fail early and loudly instead of silently returning wrong results. While input sanitization is applied for all class constructors and most exposed methods, internal interfaces often do not check their input parameters due to performance reasons.

4.8 Usage Examples

4.8.1 Basic Setup of a Calculation

This section shows which steps are required to perform a basic calculation. It has to be noted, that this is not a full documentation of the code and does not show all possible configuration options.

Components

At first, the used `Component` objects have to be defined. As this involves manual input of data (component names, molar masses, ...) which usually stays the same over multiple calculations, a script is provided that creates components for the currently most used substances: hydrogen, methane, carbon dioxide, carbon monoxide, and water (vapor).

```
define_components
```

This script creates a struct array with one field for each of the five components. Single components can easily be accessed by their short name (formula), e.g.

```
cmp.h2
```

For all further calculations, a `Components` object with all currently used components has to be created. In order to do that, the constructor has to be called with a vector of `Component`-objects. For a simple case of methane and hydrogen this can be achieved as:

```
components = Membrane.Components([cmp.ch4; cmp.h2]);
```

Permeances

The next step is to define the parameters for the calculation of permeance. As the reference implementation uses constant permeances, the only “parameter” for each component is the permeance itself. There are two variants how the permeances can be passed to the constructor of the `Module` class:

- A struct with a field for each component
- A column vector with the values in the order of the components

The struct-variant typically is less prone to errors because it does not depend on the order of the components. So if the permeance for hydrogen is given as $5 \times 10^{-8}\,\mathrm{mol\,m^{-2}\,s^{-1}\,Pa^{-1}}$ and a selectivity hydrogen/methane of 100:

```
permeances = struct();
permeances.h2 = 5e-8;
permeances.ch4 = permeances.h2 / 100;
```

Streams

Next, the streams for the calculation have to be defined. The parameters are mostly scalars, composition being the exception, which has to be passed values for every component. The composition shall be given as molar fractions. A feed composition of 10 % (mol/mol) hydrogen and 90 % (mol/mol) methane can be set as:

```
x = struct();
x.h2 = 0.1;
x.ch4 = 1 - x.h2;
```

For the other properties of the stream a pressure of 50 bar, a temperature of 300 K, and a flow of $0.01\,\mathrm{mol\,s^{-1}}$ is assumed.

```
feed = Membrane.Stream(components, 'name', 'Feed', ...
    'p', 50*1e5, 'T', 300, 'n', 0.01, 'x', x);
```

It can be seen that the first parameter is always the `Components` object. The stream has to "know", which components it is made up from.

For the permeate and retentate not that many parameters have to be defined. The main important parameter for the permeate is the pressure (here: 1 bar). The retentate does not require any parameters at all, but setting a name is nice anyway:

```
permeate = Membrane.Stream(components, 'name', 'Permeate', ...
    'p', 1*1e5);

retentate = Membrane.Stream(components, 'name', 'Retentate');
```

In general, the feed has to be fully defined and for the permeate, the pressure has to be defined. All other values are not necessary and will be overwritten by the calculation afterwards.

Module

Using the previously defined objects, a `Module` object can now be created. Required parameters are the total membrane area of the module and the location of the permeate outlet. For 0.2 m^2 and counter-current configuration the setup can be made as:

```
module = Membrane.Module(components, permeances, ...
    'A', 0.2, 'outlet', 0, ...
    'feed', feed, 'permeate', permeate, 'retentate', retentate );
```

Calculation

The final step is to create a `Calculation` object based on the defined `Module`.

```
calc = Membrane.Calculation(module);
```

As this `Calculation`-object is initialized with somewhat sensible defaults, the calculation can instantly be initialized:

```
calc.initialize();
```

This by default shows a summary of the calculation settings as well as a semi-graphical representation of the configured module. After the calculation is initialized, the iteration can be started:

```
calc.iterate();
```

While the code is iterating, some status lines are displayed. These include basic calculation information like elapsed time, current convergence values, stage cut, and flows and compositions of the permeate and retentate streams. Additionally, information is given about the change of under-relaxation parameters (as described in section 4.1). After reaching convergence, information about mass balance and possible issues with the result (due to e.g. the limitation of flux as described in section 3.3.4) are shown.

4.8.2 Using Real Gas Properties

As stated above and described in section 4.2, the class `Components` provides two modes for calculating required properties. Changing between these two modes is easily possible by setting a property of the `Components` object. The following seven properties / groups of properties can be set to the simple calculation or interpolation mode separately.

- Partial Pressure / Fugacity

- Molar Mass
- Molar Volume
- Density
- Viscosity
- Thermal Conductivity
- Enthalpy / Heat Capacities

In order to use the interpolation mode, well defined axes and a grid are required. The easiest way to generate these from a given properties table is the included script `load_interpolation_grid`.

4.8.3 Using External Loops to Solve for Non-Default Parameters

Usually, the library performs calculations as described in section 3.6: Input parameters that have to be defined are a fully specified module and feed as well as the permeate pressure. Output values are the permeate and retentate.

Many cases exist where this is not the desired direction of calculation. A frequently used case for this is varying some parameters (e.g., membrane area, feed flow, permeances) to achieve a defined stage cut. These types of calculation require an outer loop to solve for the given target. A suitable way to achieve this is the MATLAB function `fzero`, which takes a function of one variable and tries to find the value for the input variable so that the output becomes zero. In order to use `fzero`, one has to write a wrapper function that takes the value to vary as input and returns the difference between the actual and the target output value. This is described in detail in the tutorial accompanying the code.

4.8.4 Example of a Two-Stage Process

Actual technical applications typically use more complex separation setups than a single gas permeation stage. Pathare and Agrawal (2010), Xu and Agrawal (1996a), and Agrawal and Xu (1996) show examples of possible multi-stage membrane cascades. While calculating complex multi-stage processes is possible with the reference code, a detailed description of the setup of such a case is out of scope of this work.

The basics of calculating multi-stage processes are shown, though, in the following simple two-stage example: The system consists of two identical membrane modules, where the permeate of the first module is used as the feed for the second stage.

The setup of components and permeances is the same as for a simple example.

```
define_components
used_cmp = [cmp.ch4; cmp.h2];
components = Membrane.Components(used_cmp);

permeances = struct();
permeances.h2 = 5e-8;
permeances.ch4 = permeances.h2 / 100;
```

For the definition of the streams, permeate and retentate have to be defined for both modules. The main additional parameter required is the intermediate pressure between the two modules. Assuming a intermediate pressure of 5 bar, the streams can be defined:

```
x = struct();
x.h2 = 0.6;
x.ch4 = 1-x.h2;

feed = Membrane.Stream(components, 'name', 'Feed', ...
    'p', 50*1e5, 'T', 300, 'n', 0.01, 'x', x);
permeate1 = Membrane.Stream(components, 'name', 'Permeate 1', ...
    'p', 5*1e5, 'T', 300);
permeate2 = Membrane.Stream(components, 'name', 'Permeate 2', ...
    'p', 1*1e5, 'T', 300 );
retentate1 = Membrane.Stream(components, 'name', 'Retentate 1');
retentate2 = Membrane.Stream(components, 'name', 'Retentate 2');
```

As the example consists of two modules, both have to be defined:

```
module1 = Membrane.Module( ...
    components, permeances, 'name', 'Module 1', ...
    'A', 0.2, 'outlet', 0 );

module1.feed = feed;
module1.permeate = permeate1;
module1.retentate = retentate1;

module2 = Membrane.Module(components, permeances, ...
    'name', 'Module 2', 'A', 0.2, 'outlet', 0);

module2.feed = permeate1;
module2.permeate = permeate2;
module2.retentate = retentate2;
```

After defining the `Calculation` objects and initializing them, the iterations can be started:

```
% Module 1
calc1.initialize();
calc1.iterate();

% Module 2
calc2.initialize();
calc2.iterate();
```

As Module 2 depends on the Module 1 in this example, it is important, that the iterations are performed on Module 1 before Module 2. This example does not contain recycles, therefore the solution process is straightforward. If streams are recycled, an outer loop is required.

5 Validation of Algorithm

5.1 Comparison with Existing MATLAB Code

5.1.1 Description of Existing MATLAB Code

As the algorithm presented in this thesis is new, the correctness of its results has to be assessed. Therefore, it is validated against an implementation of the algorithm described by Makaruk and Harasek (2009). This implementation is already validated against experimental data and is actively used in research and industry.

In order to simplify nomenclature, the algorithm presented in this thesis and its reference implementation are referred to as VECGP from now on, whereas the algorithm presented by Makaruk and Harasek is referred to as MAKARUK. As VECGP is based on the algorithm published by Makaruk and Harasek, the general approach to the problem is inherently similar. When comparing both algorithms, though, differences in various parts clearly exist. One main difference is that MAKARUK uses a Gauss-Seidel iteration, whereas VECGP uses a Jacobi iteration. Both algorithms use under-relaxation and have features to reduce the relaxation factor, but these work very differently.

While the underlying algorithm of VECGP is based on the MAKARUK-algorithm, the actual code of VECGP, is completely independent of the MAKARUK-code. Therefore, the existing implementation of MAKARUK serves as an ideal reference to validate VECGP against.

5.1.2 Definition of Test Cases

In order to compare various approaches to solve the transmembrane flux equations, a set of test cases has been defined at the research division Thermal Process Engineering & Simulation of the Institute of Chemical, Environmental & Biological Engineering at TU Wien. These test cases are also used to validate the new algorithm. They consist of two parts:

- A set of 16 two-component (hydrogen and carbon dioxide) cases which are derived from a base case by changing various parameters.
- A set of 4 five-component (methane, carbon monoxide, carbon dioxide, hydrogen, and water) cases which are derived from a base case by changing

the target stage cut.

The cases are described in detail in Table 5.1 and Table 5.2. Note that as the case IDs were defined in an arbitrary order, the tables are not fully sorted by these values. Rather, a more "logical" sorting is applied.

The two-component are split into two groups. In the first group, various physical parameters are changed (feed composition, feed pressure, target stage cut, and permeances). In the second group, only the number of cells is varied, while the physical parameters stay the same. As in these cases the same physical conditions are calculated with different settings, the impact of discretization error can be observed. This topic is also covered in more detail in section 7.2. As case C011 served as the base case for the variation of cell numbers, it fits in both groups and is therefore listed twice in the table.

As can be seen in the tables, the units that were used to define the test cases are not consistent with the units used throughout this thesis: Nm^3 instead of mol, °C instead of K, and bar instead of Pa.

For the calculations with VECGP, a script was used that calculated all test cases in a batch. This script automatically converted the input values from these units into the ones used by the reference implementation and also converted the results back to the units used in the test cases.

It has to be noted that the test cases were initially defined with a stage cut as input parameter. In both algorithms the stage cut is not a direct input parameter, but rather a value that is calculated based on the inputs. Therefore, some input parameter has to be varied in order to reach the target stage cut. For the test cases this variable input parameter is the membrane area.

In order to actually compare the algorithms themselves and not the external loops that perform the parameter variation, the required membrane areas were calculated upfront using VECGP. These calculated areas are included in Table 5.1 and Table 5.2.

The actual reference calculations were done using the pre-calculated areas as inputs for both algorithms. This means that the results can only vary in terms of the calculated streams, but not in terms of the membrane area.

For both algorithms, the configuration options were set to the same values where possible. This includes:

- a convergence criterion of 10^{-10} (applied for each component)
- pure counter-current configuration
- no pressure drop calculation

Configuration options that are only present in VECGP were set so that they most closely match the assumptions made in MAKARUK. This includes:

Table 5.1: Specification of Two-Component Test Cases

(a) Constant Parameters

$\dot{n}_{Feed}$	T_{Feed}	p_{Perm}	Π_{H_2}
($\mathrm{Nm^3\,s^{-1}}$)	(°C)	(bar)	($\mathrm{Nm^3\,s^{-1}\,m^{-2}\,bar^{-1}}$)
1	25	1	2.5×10^{-4}

(b) Case-Specific Parameters

Case	x_{Feed,H_2}	x_{Feed,CO_2}	p_{Feed}	Π_{CO_2}	SC	A	n
	(–)	(–)	(bar)	($\mathrm{Nm^3\,s^{-1}\,m^{-2}\,bar^{-1}}$)	(–)	($\mathrm{m^2}$)	(–)
C001	0.5	0.5	10	2.5×10^{-3}	0.5	60.092	100
C002	0.2	0.8	10	2.5×10^{-3}	0.5	30.135	100
C003	0.8	0.2	10	2.5×10^{-3}	0.5	144.801	100
C004	0.5	0.5	7	2.5×10^{-3}	0.5	93.392	100
C005	0.5	0.5	4	2.5×10^{-3}	0.5	204.086	100
C006	0.5	0.5	3	2.5×10^{-3}	0.5	327.771	100
C007	0.5	0.5	10	2.5×10^{-3}	0.2	19.001	100
C008	0.5	0.5	10	2.5×10^{-3}	0.8	155.741	100
C009	0.5	0.5	10	7.5×10^{-4}	0.5	123.403	100
C010	0.5	0.5	10	6.3×10^{-3}	0.5	34.005	100
C011	0.5	0.5	10	2.5×10^{-2}	0.5	15.560	100
C017	0.5	0.5	10	2.5×10^{-2}	0.5	17.177	10
C018	0.5	0.5	10	2.5×10^{-2}	0.5	15.740	50
C011	0.5	0.5	10	2.5×10^{-2}	0.5	15.560	100
C019	0.5	0.5	10	2.5×10^{-2}	0.5	15.452	250
C020	0.5	0.5	10	2.5×10^{-2}	0.5	15.416	500
C012	0.5	0.5	10	2.5×10^{-2}	0.5	15.397	1000

Table 5.2: Specification of Five-Component Test Cases

(a) Constant Parameters

$\dot{n}_{Feed}$ (Nm^3 s^{-1})	x_{Feed,CH_4} (–)	$x_{Feed,CO}$ (–)	x_{Feed,CO_2} (–)	x_{Feed,H_2} (–)	x_{Feed,H_2O} (–)	T_{Feed} (°C)	p_{Feed} (bar)	p_{Perm} (bar)	n (–)
1	0.5630	0.0064	0.0807	0.3489	0.0010	25	10	1	200

Π_{CH_4} (Nm^3 s^{-1} m^{-2} bar^{-1})	Π_{CO} (Nm^3 s^{-1} m^{-2} bar^{-1})	Π_{CO_2} (Nm^3 s^{-1} m^{-2} bar^{-1})	Π_{H_2} (Nm^3 s^{-1} m^{-2} bar^{-1})	Π_{H_2O} (Nm^3 s^{-1} m^{-2} bar^{-1})
2.775×10^{-6}	2.885×10^{-6}	6.976×10^{-5}	2.175×10^{-4}	7.500×10^{-4}

(b) Case-Specific Parameters

Case	SC (–)	A (m^2)
C015	0.05	82.56
C013	0.50	3578.43
C014	0.80	15 088.16
C016	0.90	19 090.06

- no relative convergence or mass balance criteria
- no calculation of the energy balance
- usage of simple ideal gas properties (no interpolated real gas data — this is the default setting of the implementation)
- a hard flux limit as described in section 3.3.4 (this is the default setting of the implementation)
- the evaluation location for cell properties (as described in section 3.2.4) is set to *downstream* (this is the default setting of the implementation)

5.1.3 Comparison of Results

The results are shown in Table 5.3 to Table 5.6. For both the two-component and the five-component cases, there are nearly no visible differences in the first three digits of the flows and composition of permeate and retentate. Only in case C017, there is a slight deviation of the stage cut and the retentate composition. C017 is the case with only 10 cells, which is subject to large discretization errors anyway.

The difference between both algorithms is typically in the order of 10^{-5}, which in terms of concentrations corresponds to 10 ppm. No plots of the results are shown, as there would be no visible differences between VECGP and MAKARUK. Even for the comparatively large deviation of case C017 (which is in the order of 10^{-3}), the difference would only be around 0.1 mm when using bars of 100 mm length. Given the conceptual differences between VECGP and MAKARUK, these small variation of the results are well within an expected range of uncertainty.

Mass balance is closed up to at least an order of 10^{-8} for all cases and both algorithms.

5.1.4 Comparison of Calculation Performance

As stated above, both VECGP and MAKARUK calculate very similar results. They do, however, show a very different behavior in terms of results. There are two parameters that can be compared:

- calculation time
- number of iterations

Whereas the number of iterations remains constant when performing the same calculation multiple times, the calculation time is dependent on various factors like system utilization and other running programs. In order to minimize external

Table 5.3: Key Simulation Results of Two-Component Test Cases – VECGP

Case	$\dot{n}_{Perm}$ ($\mathrm{Nm^3\,s^{-1}}$)	x_{Perm,H_2} (–)	$\dot{n}_{Ret}$ ($\mathrm{Nm^3\,s^{-1}}$)	x_{Ret,H_2} (–)	Bal_{H_2} ($\mathrm{Nm^3\,s^{-1}}$)	Bal_{CO_2} ($\mathrm{Nm^3\,s^{-1}}$)
C001	0.500	0.189	0.500	0.811	-3.55×10^{-10}	3.55×10^{-9}
C002	0.500	0.040	0.500	0.360	-2.41×10^{-10}	2.41×10^{-9}
C003	0.500	0.613	0.500	0.987	-5.33×10^{-10}	5.33×10^{-9}
C004	0.500	0.200	0.500	0.800	-4.06×10^{-10}	4.06×10^{-9}
C005	0.500	0.229	0.500	0.771	-5.74×10^{-10}	5.74×10^{-9}
C006	0.500	0.253	0.500	0.747	-7.59×10^{-10}	7.59×10^{-9}
C007	0.200	0.126	0.800	0.593	-2.81×10^{-10}	2.81×10^{-9}
C008	0.800	0.376	0.200	0.998	-3.53×10^{-10}	3.53×10^{-9}
C009	0.500	0.333	0.500	0.667	-1.24×10^{-9}	3.71×10^{-9}
C010	0.500	0.118	0.500	0.882	-1.31×10^{-10}	3.28×10^{-9}
C011	0.500	0.061	0.500	0.939	-3.45×10^{-11}	3.45×10^{-9}
C017	0.500	0.068	0.500	0.932	2.27×10^{-13}	-2.27×10^{-11}
C018	0.500	0.061	0.500	0.939	-1.69×10^{-11}	1.69×10^{-9}
C011	0.500	0.061	0.500	0.939	-3.45×10^{-11}	3.45×10^{-9}
C019	0.500	0.060	0.500	0.940	-8.70×10^{-11}	8.70×10^{-9}
C020	0.500	0.060	0.500	0.940	-1.75×10^{-10}	1.75×10^{-8}
C012	0.500	0.060	0.500	0.940	-3.50×10^{-10}	3.50×10^{-8}

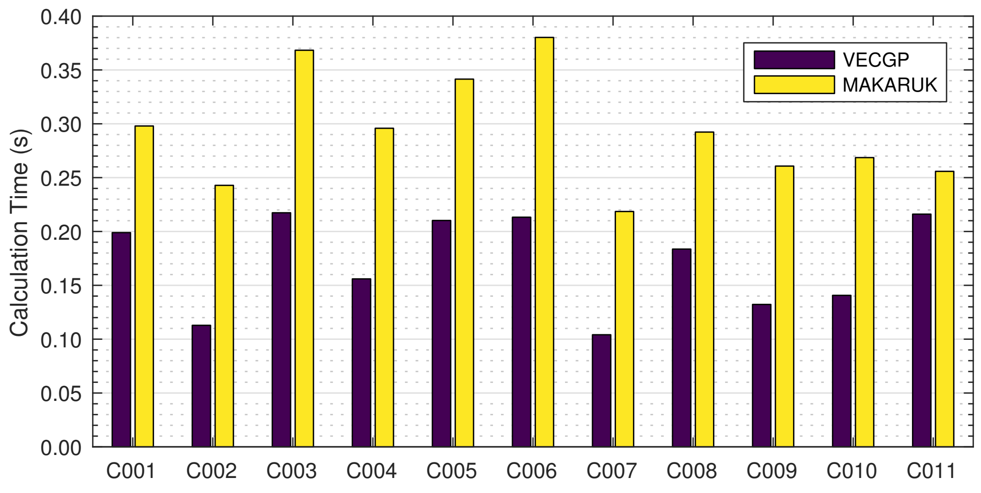


Figure 5.1: Comparison of Calculation Performance – Calculation Time Two-Component Cases with Constant Number of Cells

Table 5.4: Key Simulation Results of Two-Component Test Cases – MAKARUK

Case	$\dot{n}_{Perm}$ (Nm^3 s^{-1})	x_{Perm,H_2} (–)	$\dot{n}_{Ret}$ (Nm^3 s^{-1})	x_{Ret,H_2} (–)	Bal_{H_2} (Nm^3 s^{-1})	Bal_{CO_2} (Nm^3 s^{-1})
C001	0.500	0.189	0.500	0.811	1.96×10^{-10}	-1.80×10^{-9}
C002	0.500	0.040	0.500	0.360	1.39×10^{-10}	-1.30×10^{-9}
C003	0.500	0.613	0.500	0.987	-3.02×10^{-10}	2.68×10^{-9}
C004	0.500	0.200	0.500	0.800	2.32×10^{-10}	-2.13×10^{-9}
C005	0.500	0.229	0.500	0.771	3.17×10^{-10}	-2.93×10^{-9}
C006	0.500	0.253	0.500	0.747	4.14×10^{-10}	-3.83×10^{-9}
C007	0.200	0.126	0.800	0.593	1.98×10^{-10}	-1.76×10^{-9}
C008	0.800	0.376	0.200	0.998	-1.93×10^{-10}	1.73×10^{-9}
C009	0.500	0.333	0.500	0.667	6.75×10^{-10}	-1.96×10^{-9}
C010	0.500	0.118	0.500	0.882	7.66×10^{-11}	-1.65×10^{-9}
C011	0.500	0.061	0.500	0.939	2.57×10^{-11}	-1.67×10^{-9}
C017	0.499	0.068	0.501	0.930	4.54×10^{-12}	-1.01×10^{-10}
C018	0.500	0.061	0.500	0.938	1.61×10^{-11}	-8.05×10^{-10}
C011	0.500	0.061	0.500	0.939	2.57×10^{-11}	-1.67×10^{-9}
C019	0.500	0.060	0.500	0.940	5.27×10^{-11}	-4.30×10^{-9}
C020	0.500	0.060	0.500	0.940	9.70×10^{-11}	-8.71×10^{-9}
C012	0.500	0.060	0.500	0.940	1.92×10^{-10}	-1.83×10^{-8}

Table 5.5: Key Simulation Results of Five-Component Test Cases – VECGP

Case	$\dot{n}_{Perm}$ (Nm^3 s^{-1})	x_{Perm,CH_4} (–)	$x_{Perm,CO}$ (–)	x_{Perm,CO_2} (–)	x_{Perm,H_2} (–)	x_{Perm,H_2O} (–)
C015	0.050	0.026	0.000	0.083	0.885	0.005
C013	0.500	0.162	0.002	0.142	0.692	0.002
C014	0.800	0.456	0.005	0.101	0.436	0.001
C016	0.900	0.516	0.006	0.090	0.388	0.001

Case	$\dot{n}_{Ret}$ (Nm^3 s^{-1})	x_{Ret,CH_4} (–)	$x_{Ret,CO}$ (–)	x_{Ret,CO_2} (–)	x_{Ret,H_2} (–)	x_{Ret,H_2O} (–)
C015	0.950	0.591	0.007	0.081	0.321	0.001
C013	0.500	0.964	0.011	0.019	0.006	0.000
C014	0.200	0.989	0.011	0.000	0.000	0.000
C016	0.100	0.989	0.011	0.000	0.000	0.000

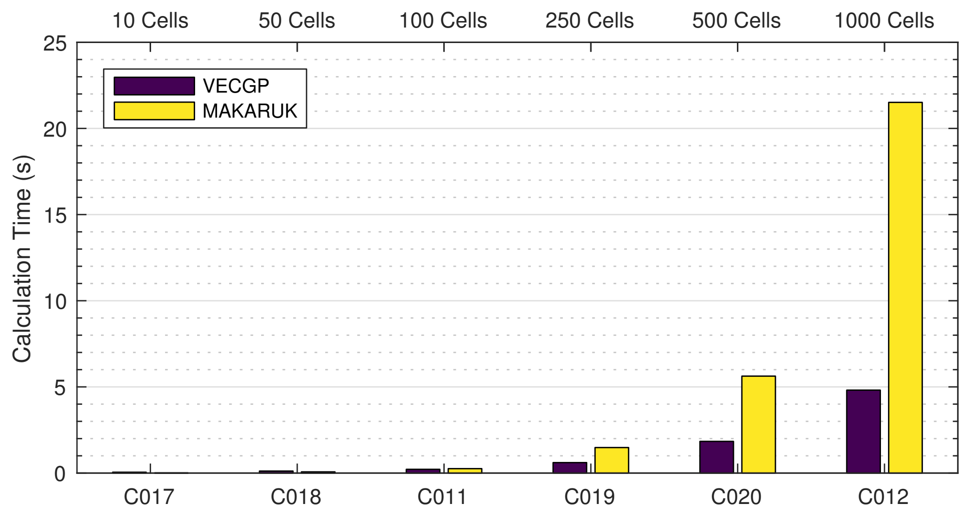
Case	Bal_{CH_2} (Nm^3 s^{-1})	Bal_{CO} (Nm^3 s^{-1})	Bal_{CO_2} (Nm^3 s^{-1})	Bal_{H_2} (Nm^3 s^{-1})	Bal_{H_2O} (Nm^3 s^{-1})
C015	-3.72×10^{-11}	-4.41×10^{-13}	-1.04×10^{-9}	3.68×10^{-9}	8.66×10^{-9}
C013	-3.83×10^{-11}	-2.49×10^{-13}	-4.58×10^{-9}	1.71×10^{-8}	1.34×10^{-9}
C014	2.04×10^{-12}	-1.30×10^{-11}	1.15×10^{-10}	-4.37×10^{-11}	4.39×10^{-9}
C016	-5.21×10^{-9}	5.44×10^{-9}	1.87×10^{-10}	2.81×10^{-9}	1.04×10^{-9}

Table 5.6: Key Simulation Results of Five-Component Test Cases – MAKARUK

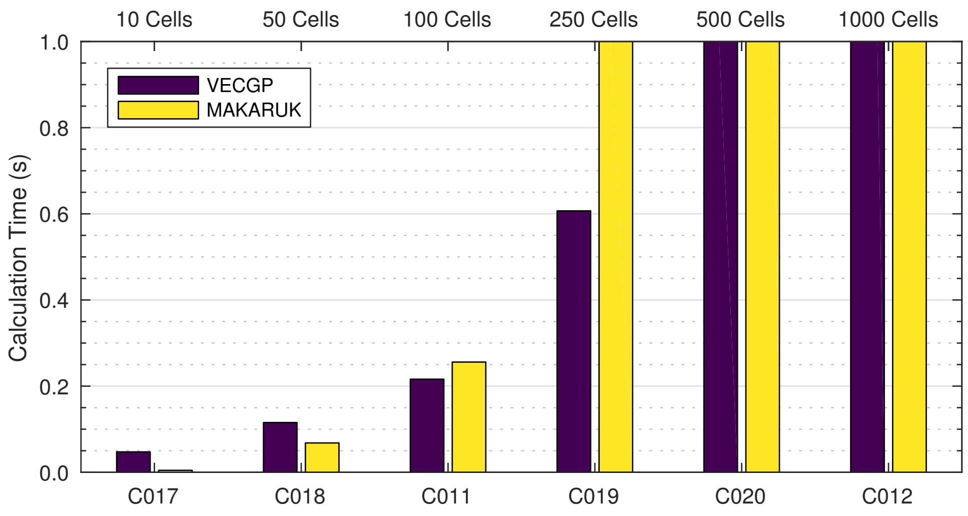
Case	$\dot{n}_{Perm}$	x_{Perm,CH_4}	$x_{Perm,CO}$	x_{Perm,CO_2}	x_{Perm,H_2}	x_{Perm,H_2O}
	(Nm^3 s^{-1})	(–)	(–)	(–)	(–)	(–)
C015	0.050	0.026	0.000	0.083	0.885	0.005
C013	0.500	0.162	0.002	0.142	0.692	0.002
C014	0.800	0.456	0.005	0.101	0.436	0.001
C016	0.900	0.516	0.006	0.090	0.388	0.001

Case	$\dot{n}_{Ret}$	x_{Ret,CH_4}	$x_{Ret,CO}$	x_{Ret,CO_2}	x_{Ret,H_2}	x_{Ret,H_2O}
	(Nm^3 s^{-1})	(–)	(–)	(–)	(–)	(–)
C015	0.950	0.591	0.007	0.081	0.321	0.001
C013	0.500	0.964	0.011	0.019	0.006	0.000
C014	0.200	0.989	0.011	0.000	0.000	0.000
C016	0.100	0.989	0.011	0.000	0.000	0.000

Case	Bal_{CH_2}	Bal_{CO}	Bal_{CO_2}	Bal_{H_2}	Bal_{H_2O}
	(Nm^3 s^{-1})	(Nm^3 s^{-1})	(Nm^3 s^{-1})	(Nm^3 s^{-1})	(Nm^3 s^{-1})
C015	4.38×10^{-11}	5.06×10^{-13}	3.16×10^{-10}	-9.43×10^{-10}	-4.39×10^{-9}
C013	2.12×10^{-12}	2.90×10^{-14}	1.36×10^{-11}	-9.21×10^{-11}	3.63×10^{-12}
C014	-3.50×10^{-11}	7.87×10^{-11}	-1.22×10^{-9}	-9.08×10^{-11}	8.71×10^{-10}
C016	8.76×10^{-11}	-3.63×10^{-11}	-1.36×10^{-9}	-9.16×10^{-11}	8.08×10^{-10}

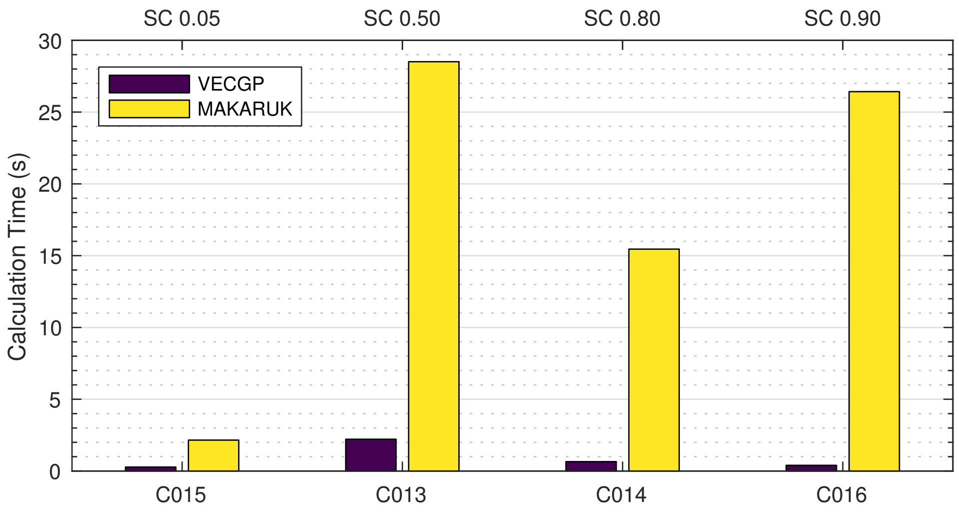


(a) Wide Y-Axis Range

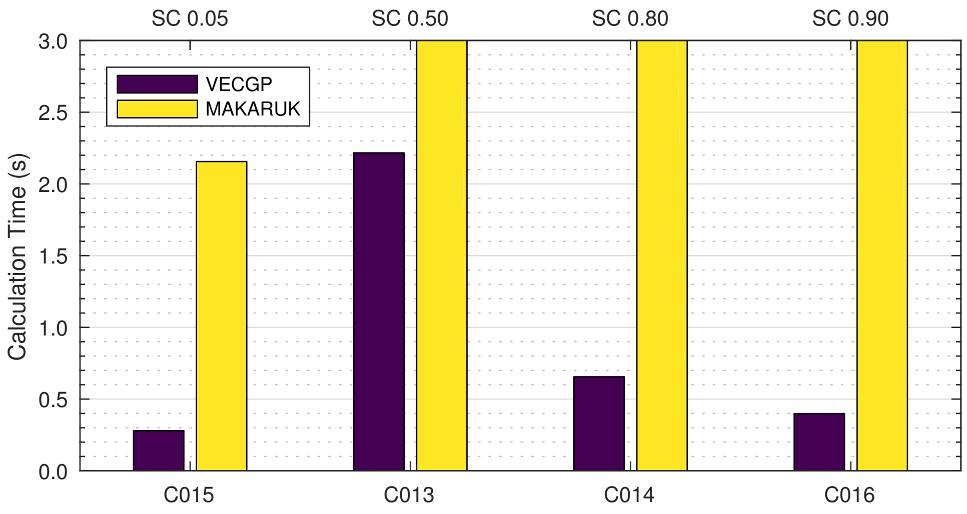


(b) Narrow Y-Axis Range

Figure 5.2: Comparison of Calculation Performance – Calculation Time Two-Component Cases with Varying Number of Cells



(a) Wide Y-Axis Range



(b) Narrow Y-Axis Range

Figure 5.3: Comparison of Calculation Performance – Calculation Time Five-Component Cases

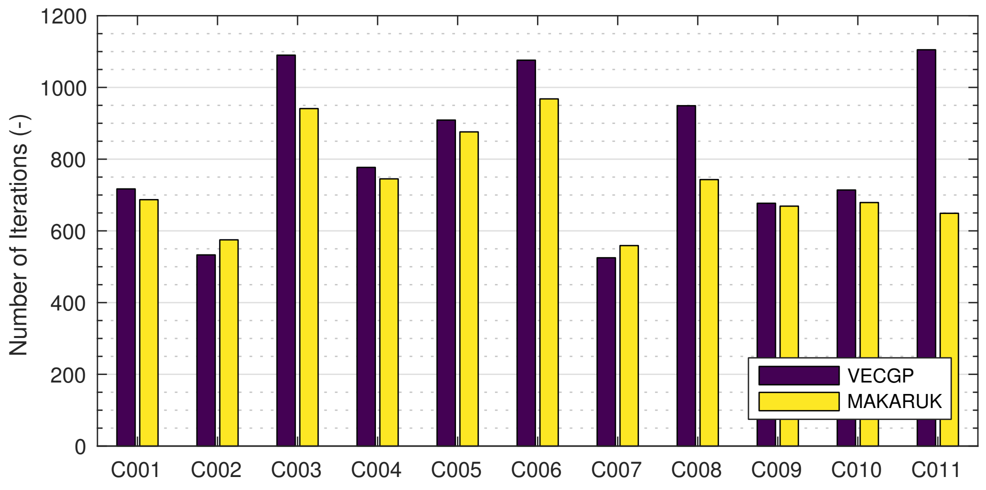


Figure 5.4: Comparison of Calculation Performance – Number of Iterations Two-Component Cases with Constant Number of Cells

impacts, all calculations were run 100 times and average calculation times were calculated.

The calculation times are shown in Fig. 5.1 to Fig. 5.3. As the calculation times vary in large ranges, the two-component cases with varying cell numbers and the five-component cases are shown twice with different scales on the y-axis.

It can bee seen that for the two-component cases with a constant number of cells of 100, MAKARUK is typically slower by a factor of 1.5 to 2. The notable exception is case C011, which is defined with a relatively high selectivity (100) compared to the other cases (10 for C001 to C008, 3 for C009, and 25 for C010). In this case, MAKARUK is only about 18% slower.

For the two-component cases with varying numbers of cells, this picture changes slightly. While MAKARUK is faster by a factor of 10 for the case with only 10 cells, this advantage disappears with increasing numbers of cells. For the 1000-cells case, VECGP is faster by a factor of around 4.5. This means that VECGP spends more time calculating independent of the number of cells, which probably can be attributed to overhead introduced by the object oriented code structure and a high number of function calls. With growing cell numbers, though, the required calculation time grows much faster for MAKARUK than for VECGP. This effect is explained in more detail further down when taking into account the iteration numbers.

With the five-component cases, the speed advantage of VECGP increases further. For these cases, MAKARUK is 7 to over 60 times slower than VECGP.

Fig. 5.4 to Fig. 5.6 show the number of iterations required to reach the set convergence criterion. For the two-component cases with a constant number of cells, both algorithms require a similar number of iterations to reach convergence, except for case C011, where MAKARUK requires about 40% less iterations. For the cases with

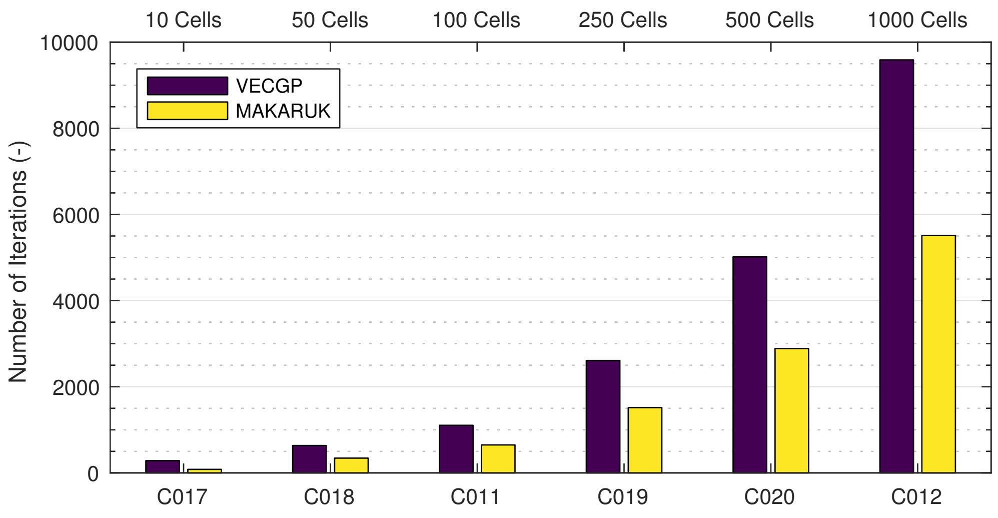


Figure 5.5: Comparison of Calculation Performance – Number of Iterations Two-Component Cases with Varying Number of Cells

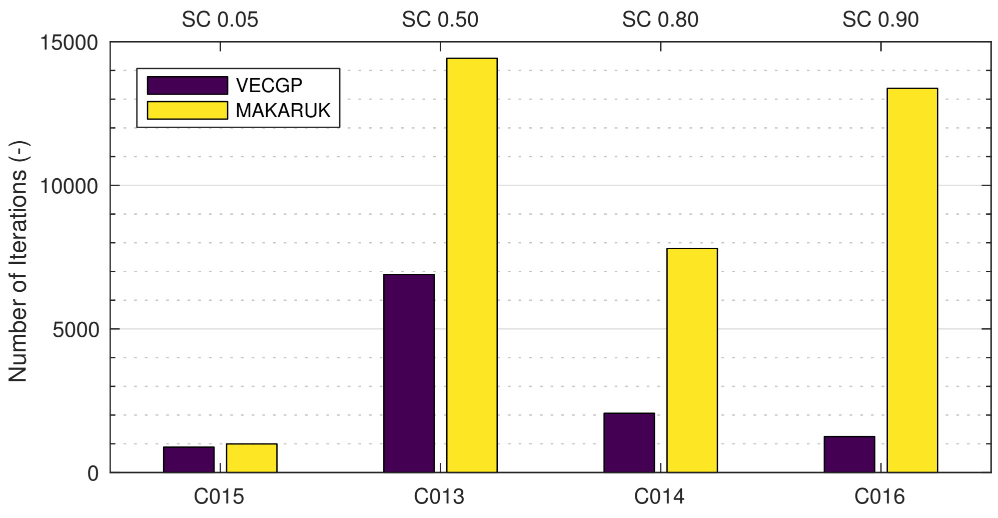


Figure 5.6: Comparison of Calculation Performance – Number of Iterations Five-Component Cases

varying cell numbers based on C011, this 40%-advantage in number of iterations stays mostly constant, except for the 10-cell case, where MAKARUK requires only around a third of the iterations required by VECGP.

For the five-component cases, the situation changes again, as can be seen in Fig. 5.6. Especially for high stage cuts (cases C013, C014, and C016), MAKARUK requires 2 to 10 times more iterations than VECGP.

This significant difference in the number of iterations can be explained by two factors. One is the general conceptual difference between the algorithms. MAKARUK uses Gauss-Seidel iteration, whereas VECGP uses Jacobi iterations. In general, a Gauss-Seidel iteration reaches convergence in fewer iterations than a Jacobi iteration. (Bärwolff 2016, p. 200) Taking into account only this difference, MAKARUK should consistently require fewer iterations than VECGP, which is not the case. The second factor affecting the number of iterations is the different approach to reducing the relaxation factor. As VECGP limits transmembrane flux (see section 3.3.4), the algorithm can typically start with a higher initial relaxation factor (0.9 by default), whereas in MAKARUK typically a rather low relaxation factor is selected to avoid overshooting flux calculations. Additionally, MAKARUK reduces the relaxation factor linearly over a large number of iterations, whereas VECGP makes few discrete steps, as described in section 4.1. It can therefore be assumed, that in the "easy" two-component cases, MAKARUK generally performs well, whereas VECGP can make up its conceptual disadvantage by using higher relaxation factors in most cases. For the "difficult" five-component cases, though, MAKARUK requires a high number of iterations until a relaxation factor is reached which allows convergence.

The difference in the number of iterations alone cannot fully explain the significant difference in calculation time that can be observed in many cases. In order to further investigate this issue, the average time per iteration was calculated. This is shown as time per 1000 iterations in Fig. 5.7 to Fig. 5.9.

It can be observed that the time per iteration for MAKARUK is influenced far more by the number of cells and the number of components than for VECGP. This can be explained by the difference of the per-cell handling between the two algorithms. While MAKARUK loops over all cells and components, VECGP uses vectorized computation as described in section 4.5. Additionally, the object oriented approach of VECGP leads to some overhead. This is especially visible in case C017, where the time per iteration for MAKARUK is significantly lower than for VECGP. With higher cell numbers (or more components), the time advantage of vectorized calculation over looping increases significantly.

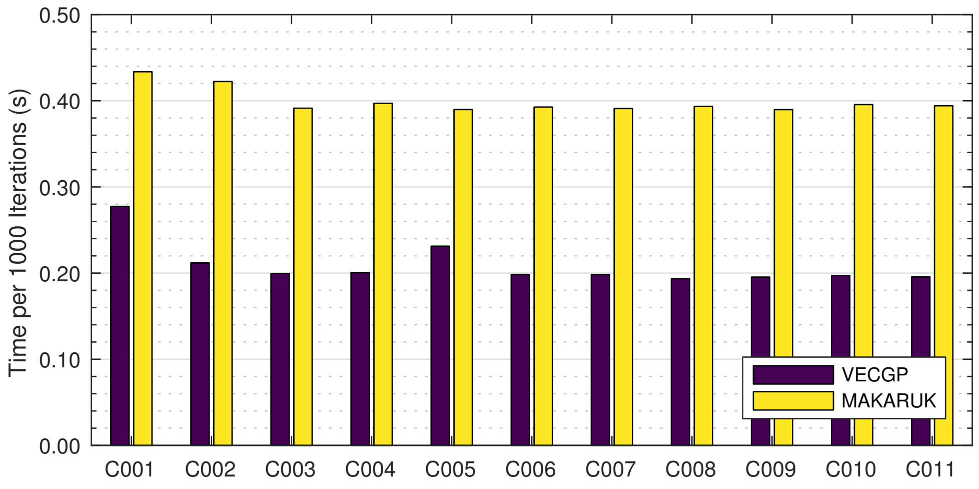


Figure 5.7: Comparison of Calculation Performance – Time per Iteration Two-Component Cases with Constant Number of Cells

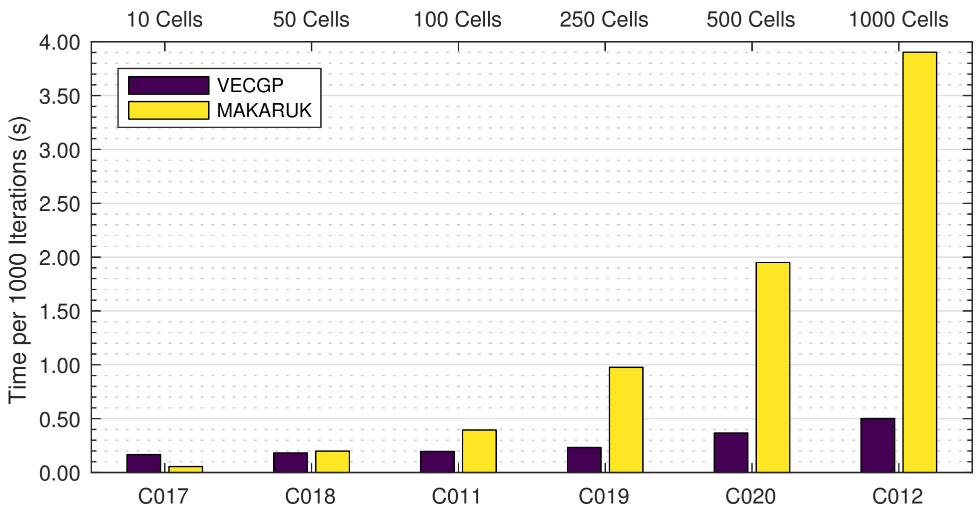


Figure 5.8: Comparison of Calculation Performance – Time per Iteration Two-Component Cases with Varying Number of Cells

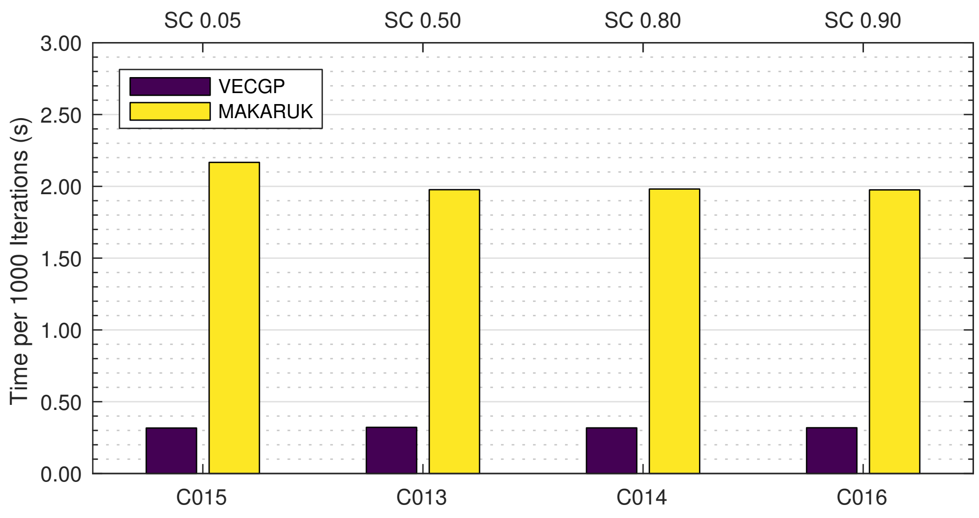


Figure 5.9: Comparison of Calculation Performance – Time per Iteration Five-Component Cases

5.2 Comparison with CFD

5.2.1 Description of CFD Code

In section 5.1, the core functionality of VECGP was validated against a conceptually similar code. The following Section will, in contrast, compare the results of VECGP with an algorithm based on CFD, as described by Schretter (2016) and Haddadi Sisakht et al. (2016). As this CFD based algorithm is conceptually very different, it is expected that the results will not be as close as with MAKARUK.

The approach taken in the CFD based algorithm is to use well established and tested CFD solvers in order to simulate the flow in the feed and permeate channels. As these solvers typically do not cover mass transport through solid walls, they were extended to support transmembrane flux which lead to the OpenFOAM solvers *membraneFoam* and *LTSMembraneFoam* which are both developed at TU Wien.

5.2.2 Definition of Test Cases

Schretter (2016) and Haddadi Sisakht et al. (2016) describe a variety of cases. For this comparison, the seven fiber module with five different permeate outlets will be used. This allows for comparison of both the variable permeate outlet and the pressure drop calculation of VECGP.

The module is shown in Fig. 5.10 and described in Schretter (2016, p. 36). It contains seven fibers with a diameter of 1 mm and a length of 500 mm. The inner diameter of the module is 6 mm. The five possible outlets are distributed evenly between the feed side and the retentate side every 125mm.

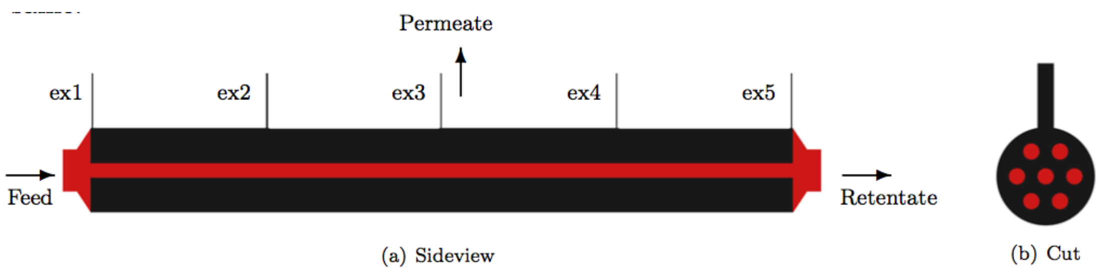


Figure 5.10: Module used for Comparison with CFD code (Schretter 2016, p. 36, Figure 13)

As the approach in the CFD based solver is rather different, the used units and parameters differ significantly. For example, VECGP users molar fractions, whereas the CFD based solver uses mass fractions. Many of the necessary conversions could be performed by the variable interface that VECGP provides, as it for example allows to use mass fractions as input variables which are automatically converted to the internally used molar fractions. In the cases where this automatic conversion was not possible, the values were converted manually.

Table 5.7 shows the case definition both in the original units defined in Schretter (2016, p. 37) and in the base units used by VECGP.

For VECGP, the number of cells was specified as 500. Pressure drop calculation was turned on and the property calculation was configured to use ideal gas behavior and simple equations (as described in section 4.2). The constant pure component viscosities were obtained from REFPROP for a temperature of 316.5 K and a pressure of 9 bar. The different outlet positions are modeled in VECGP by setting the outlet to 0, 0.25, 0.5, 0.75, and 1 respectively.

5.2.3 Comparison of Results

Fig. 5.11 to Fig. 5.14 show the results of the comparison. It can be seen that in its default configuration (hard limit), VECGP calculates a slightly higher stage cut than the CFD based algorithm. The CO_2 mass flow in the permeate (Fig. 5.11) is higher for VECGP than for the CFD based code, whereas the CO_2 mass fraction (Fig. 5.13) is very similar.

It is assumed that this difference can be attributed to the different approach that the CFD based algorithm uses. More specifically, CFD also models changes in concentration across the feed and permeate channel cross sections, and therefore inherently calculates concentration polarization effects. A definitive verification of this assumptions is not possible based on the available data, though.

As described in section 3.3.4, the default setting of VECGP is to use a hard flux limit, but it may be possible to emulate concentration polarization effects by setting low values for the soft limiting exponent. In the second result by VECGP shown in

Table 5.7: CFD Test Cases

(a) Original Parameters

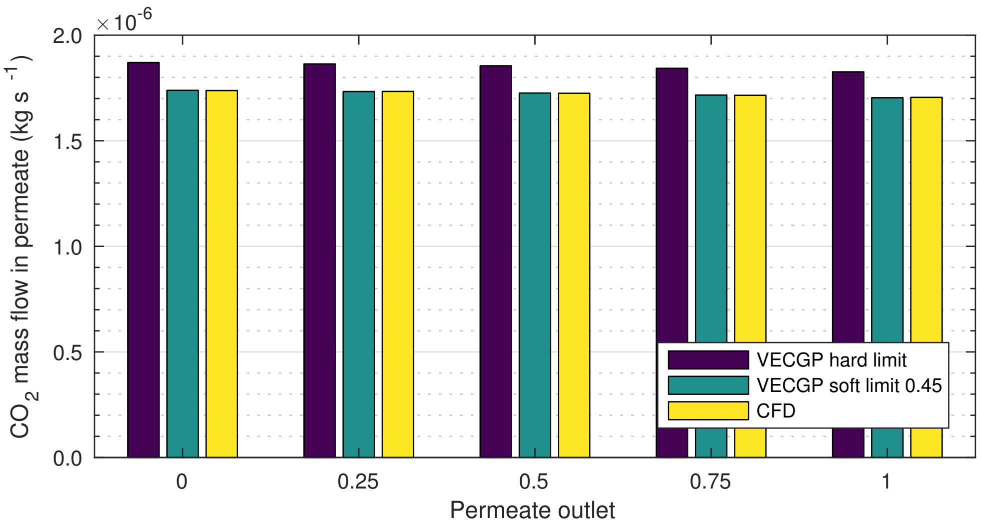
$\dot{m}_{Feed}$ (kg s^{-1})	w_{Feed,CO_2} (–)	w_{Feed,CH_4} (–)	w_{Feed,O_2} (–)	T_{Feed} (K)	p_{Feed} (bar)	p_{Perm} (bar)
6.2×10^{-6}	0.582	0.406	0.012	316.5	9.0	1.1

Π_{CO_2} (Nm^3 s^{-1} m^{-2} Pa^{-1})	Π_{CH_4} (Nm^3 s^{-1} m^{-2} Pa^{-1})	Π_{O_2} (Nm^3 s^{-1} m^{-2} Pa^{-1})
5.91×10^{-10}	1.59×10^{-11}	1.36×10^{-10}

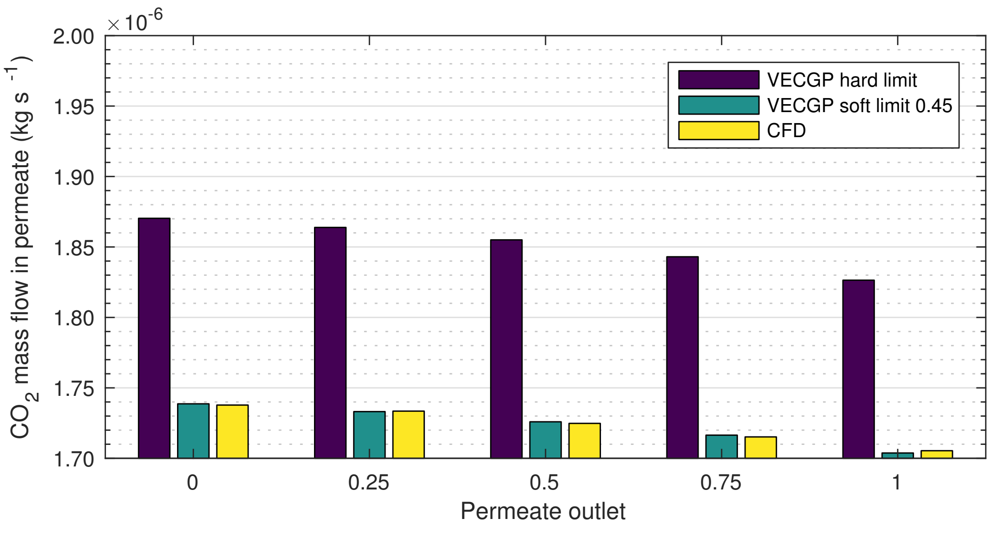
(b) Converted Parameters

$\dot{n}_{Feed}$ (mol s^{-1})	x_{Feed,CO_2} (–)	x_{Feed,CH_4} (–)	x_{Feed,O_2} (–)
2.412×10^{-4}	0.340	0.651	0.010

Π_{CO_2} (mol s^{-1} m^{-2} Pa^{-1})	Π_{CH_4} (mol s^{-1} m^{-2} Pa^{-1})	Π_{O_2} (mol s^{-1} m^{-2} Pa^{-1})
6.071×10^{-9}	7.098×10^{-10}	6.071×10^{-9}

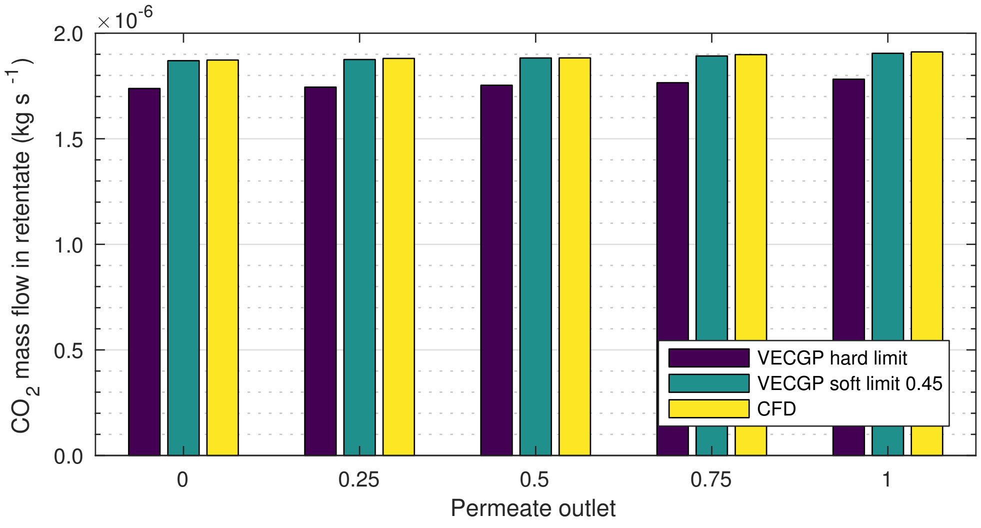


(a) Wide Y-Axis Range

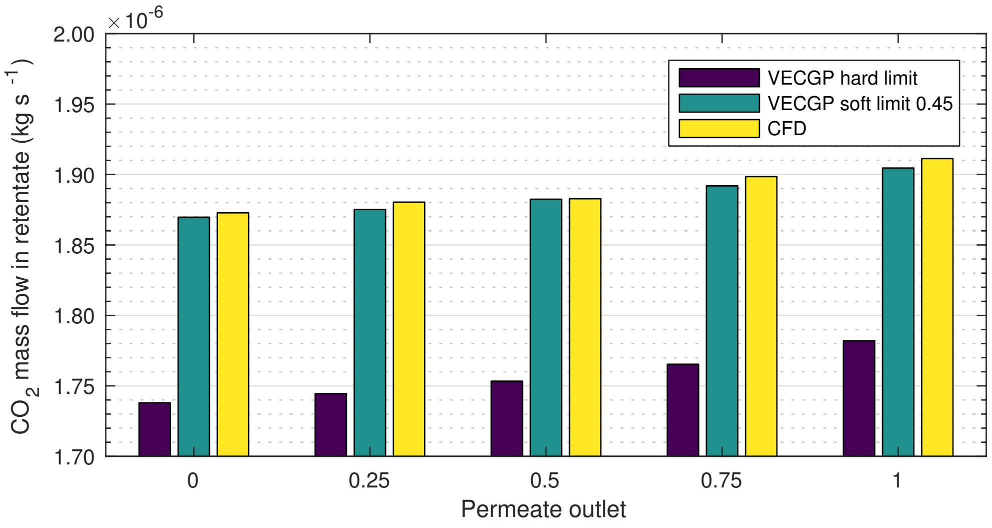


(b) Narrow Y-Axis Range

Figure 5.11: Comparison of Results – CO_2 Mass Flow in Permeate

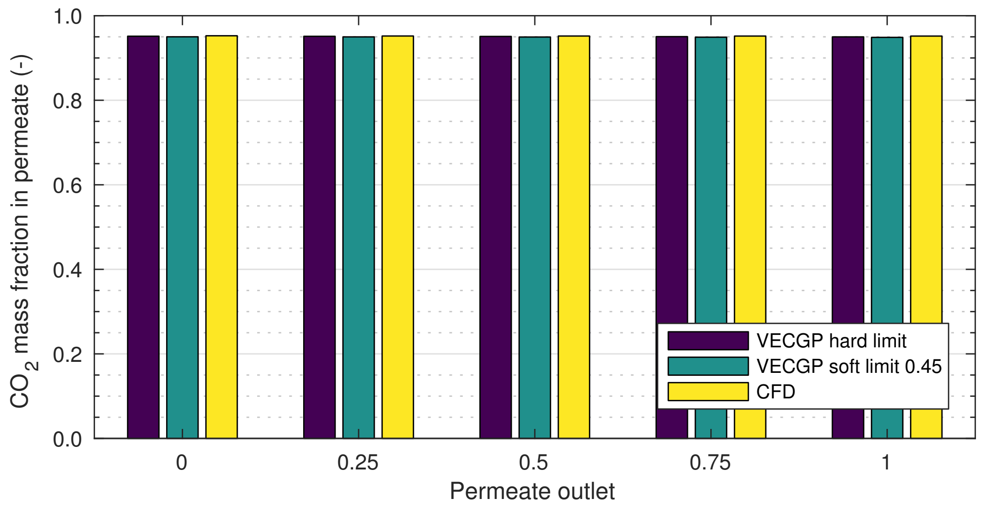


(a) Wide Y-Axis Range

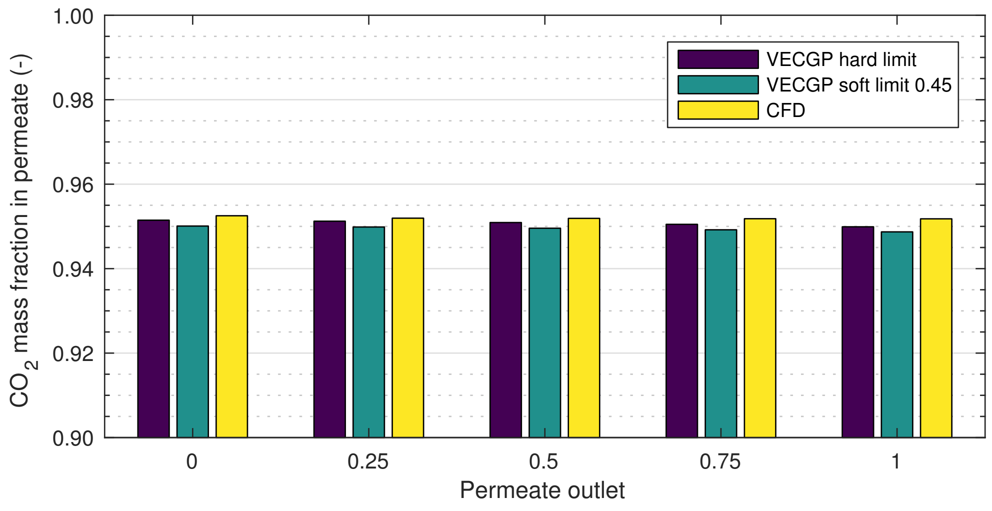


(b) Narrow Y-Axis Range

Figure 5.12: Comparison of Results – CO_2 Mass Flow in Retentate

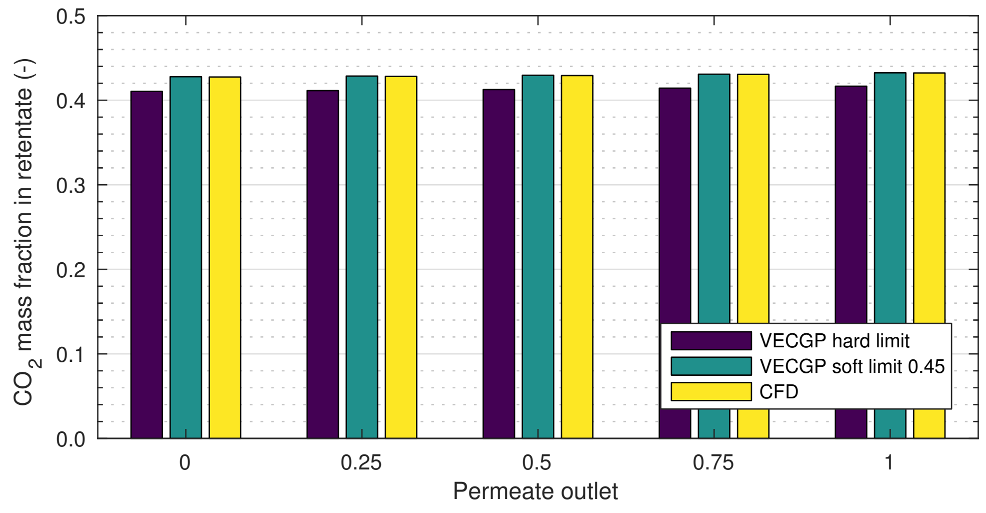


(a) Wide Y-Axis Range

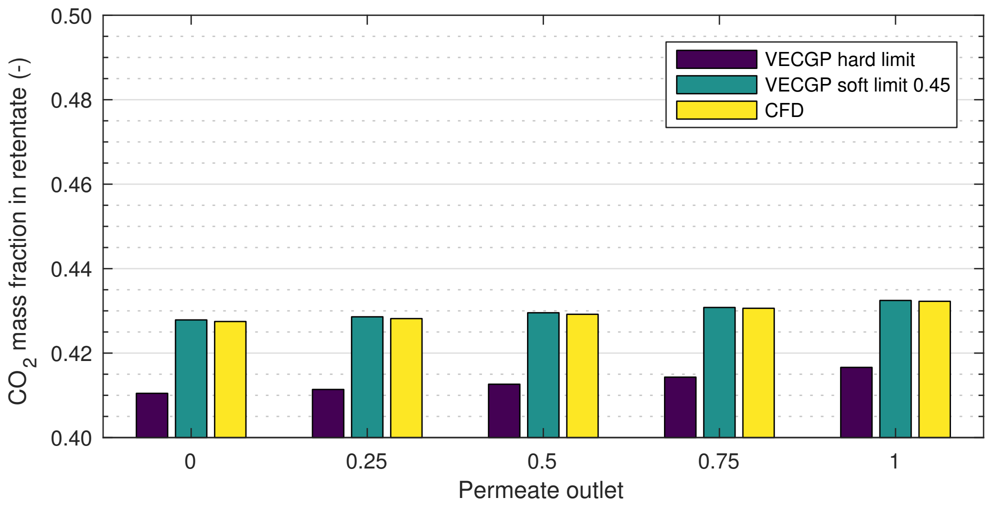


(b) Narrow Y-Axis Range

Figure 5.13: Comparison of Results – CO_2 Mass Fraction in Permeate



(a) Wide Y-Axis Range



(b) Narrow Y-Axis Range

Figure 5.14: Comparison of Results – CO_2 Mass Fraction in Retentate

Fig. 5.11 to Fig. 5.14, a soft limit with an exponent of 0.45 was used. This value was fitted in order to closely resemble the results of the CFD based algorithm.

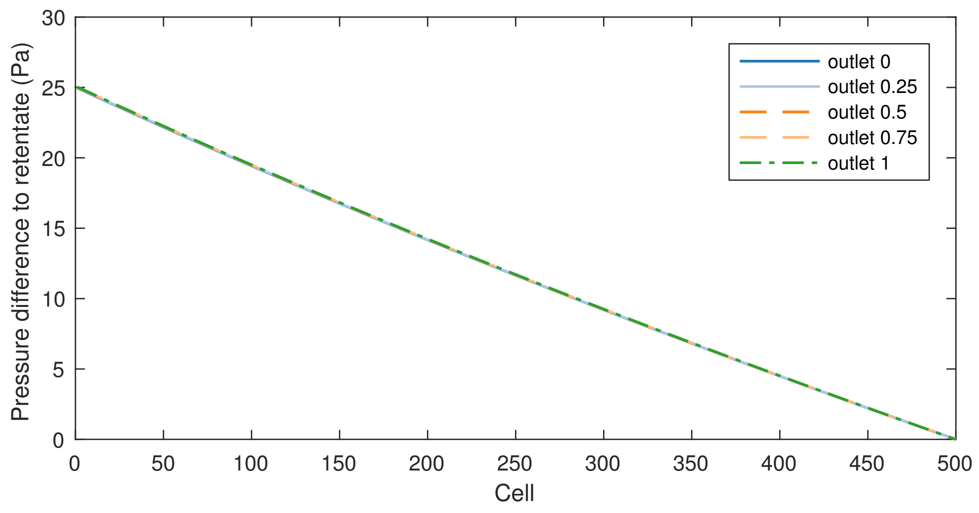
While it can be argued that setting parameters arbitrarily to fit the results does not validate the fitted results, it has to be noted that adjusting this single parameter led to significantly closer matches for all observed variables in all five cases. Whereas the results with the hard limit differed by about 8 %, the adjusted soft limit led to results of VECGP that match the results of the CFD based code within 0.5 %.

It can also be observed that the CO_2 mass flow matches very closely at the permeate outlet (Fig. 5.11) between soft limited VECGP and the CFD based algorithm, whereas there are visible differences at the retentate outlet (Fig. 5.12). In order to fulfill the CO_2 mass balance, both results should actually match, as the amount of CO_2 in the feed is the same for both algorithms. This means that the CFD based algorithm shows visible mass balance errors.

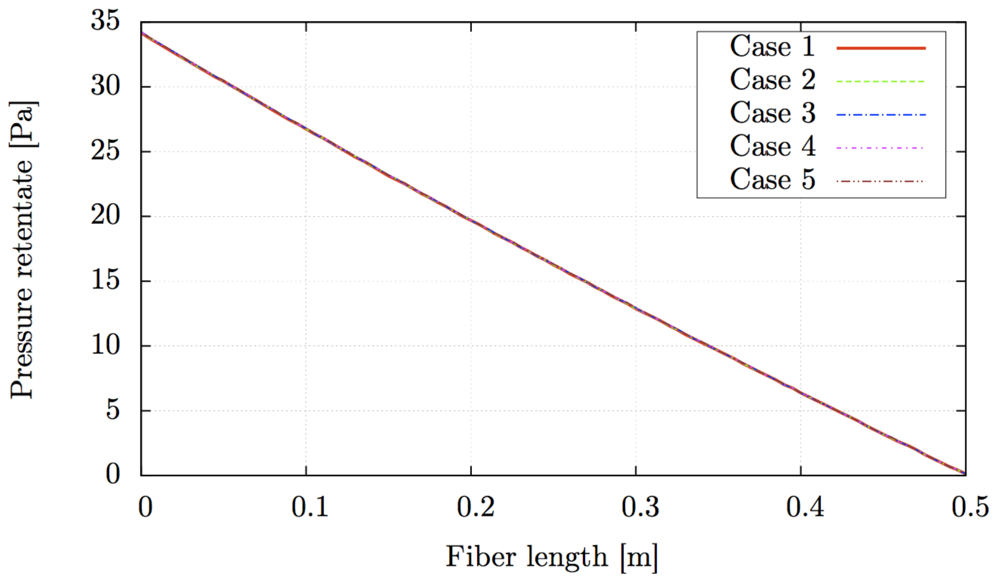
For the calculation of pressure drop, the results differ significantly more, as can be observed in Fig. 5.15 and Fig. 5.16. As the data for the results of the CFD based algorithm is shown neither in Schretter (2016), nor in Haddadi Sisakht et al. (2016), the comparison is solely based on the shown figures. Figures that closely resemble the ones published in Schretter (2016) have been produced with the results of VECGP (using a hard limit). The results of VECGP with hard limit and soft limit do not differ significantly from each other, therefore the soft-limited results are excluded for this comparison.

In general, it can be observed that the pressure drop calculated by VECGP is significantly smaller than the one calculated by the CFD based code, both in the feed and in the permeate channel. For both channels, the results look qualitatively similar. In the feed channel the maximum pressure drop is 25 Pa for VECGP and around 34 Pa for the CFD based algorithm. As the CFD based algorithm calculates the pressure difference compared to the actual value at the permeate outlet and not compared to the outlet location in the permeate channel, there is a constant "base" pressure drop of approximately 4 Pa that can be attributed to the outlet nozzle itself and is therefore not considered for the comparison with VECGP. Disregarding the outlet nozzle pressure drop, the maximum pressure difference is 1.4 Pa for VECGP and around 2.2 Pa for the CFD based code and therefore still substantially different.

Providing a clear explanation for these differences is not easily possible, as the details of the pressure drop calculation (e.g. which viscosities were used) are not described in Schretter (2016) and Haddadi Sisakht et al. (2016). Reasons for the observed discrepancies could be differently assumed viscosities, the CFD calculation leading to unequal distribution of flow between the seven fibers, or other differing underlying assumptions. Further investigation and/or experimental validation are required to determine the causes for the observed differences.

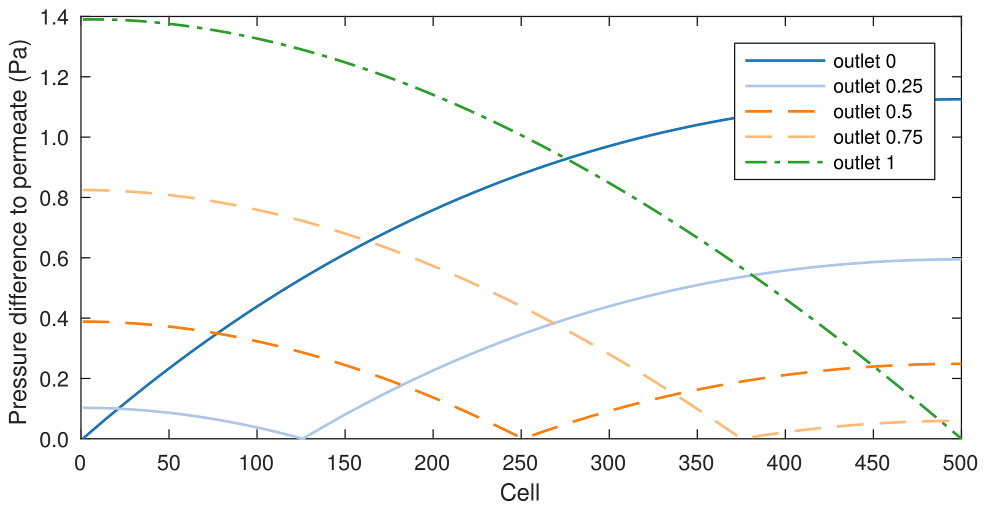


(a) VECGP with hard flux limit

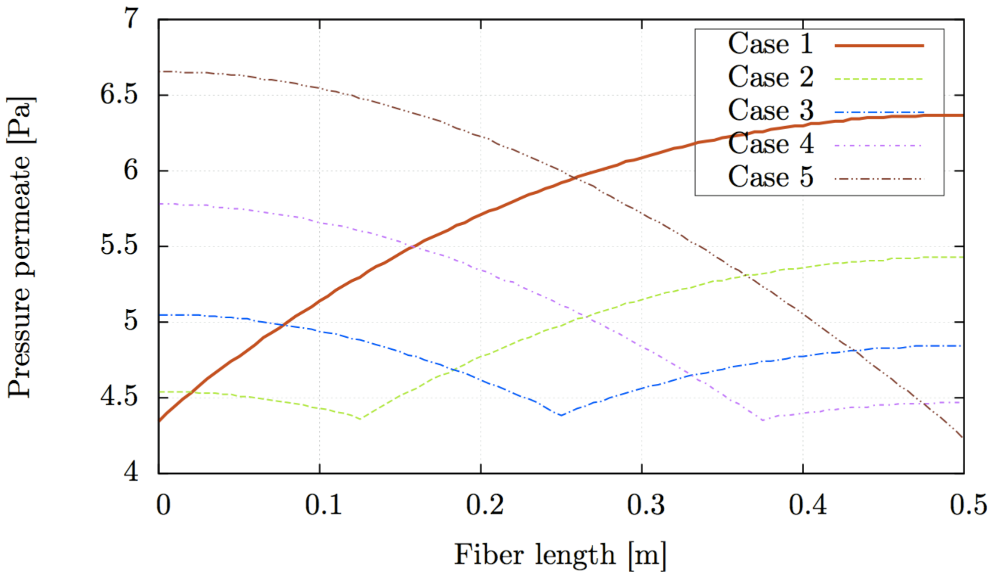


(b) CFD based algorithm (Schretter 2016, p. 72, Figure 37)

Figure 5.15: Comparison of Results – Pressure Difference in Feed Channel



(a) VECGP with hard flux limit



(b) CFD based algorithm (Schretter 2016, p. 72, Figure 37)

Figure 5.16: Comparison of Results – Pressure Difference in Permeate Channel

6 Parameter Variation

6.1 Description of Test Cases

In order to understand how changes of the algorithm parameters affect the calculation results, a common test case has been defined that represents the setup of the HylyPure test facility at the Institute of Chemical, Environmental & Biological Engineering at TU Wien. While validation against experimental data is out of scope for this thesis, this common case allows for easier comparison with experimental results in further research. The case selected for evaluation is a feed composed of 95 % (mol/mol) methane and 5 % (mol/mol) hydrogen and a membrane with a higher permeance for hydrogen (selectivity 50). The simulated module is used in counter-current configuration with shell-side feed and also the selective layer of the membrane on the shell-side. Sweep gas flows were not simulated. To distinguish these cases from the ones used in chapter 5, the case IDs are starting with C100, which is the base case. Additionally, the IDs are not numbered consecutively, but grouped by the type of variation that is performed which is visible in the tens digit.

The full stream parameters for the base case are given in Table 6.1a and the module permeances and geometry are described in Table 6.1b. These parameters remain constant in all the following test cases except for C151-C156 where the feed pressure is varied. All other variations described in this chapter are based on changing the calculation parameters of VECGP, with the defaults for the base case listed in Table 6.1c.

The variations are listed in Table 6.2a to Table 6.2e, where the changes compared to the base case are shown.

All calculations in this chapter are performed in the standard direction, i.e., with a fully defined module and solving for the permeate and retentate flows. No outer search loops (as described in section 4.8.3) are used.

6.2 Variation of Cell Numbers

The first parameter variation to consider is the number of cells, which is done in cases C111-C115. This is similar to some of the 2-component cases used for validation against MAKARUK in section 5.1. In the validation cases, though, the stage cut was defined and kept constant by varying the total membrane area, while here the area remains constant and leads to possibly different stage cuts.

Table 6.1: Base Case

(a) Stream Configuration

$\dot{n}_{Feed}$ (mol s^{-1})	x_{Feed,H_2} (–)	x_{Feed,CH_4} (–)	T_{Feed} (K)	p_{Feed} (Pa)	p_{Perm} (Pa)
2.9×10^{-3}	0.05	0.95	300	51×10^5	5×10^5

(b) Module Parameters

Π_{H_2} (mol s^{-1} m^{-2} Pa^{-1})	Π_{CH_4} (mol s^{-1} m^{-2} Pa^{-1})	A (m^2)	d_f (m)	D_I (m)	D_M (m)	n_f (–)	λ_f (W m^{-1} s^{-1})
2.5×10^{-8}	5.0×10^{-10}	0.17	1×10^{-4}	1.34×10^{-4}	0.02	750	0

(c) Calculation Settings

Parameter	Value
Number of Cells	200
Flux Limit	hard
Property Calculation	ideal
Pressure Drop Calculation	off
Energy Balance Calculation	off
Cell Property Evaluation Mode	
(Flux, Pressure, and Energy Balance)	downstream (0)
Under-Relaxation Factor	
(Flux, Pressure, and Energy Balance)	
Start Value	0.9
Reduction Factor	0.9
No. of Iterations Before First Reduction	= Number of Cells (200)
No. of Iterations Between Reductions	$= \frac{1}{2} \times$ Number of Cells (100)
Convergence Criterion	
Absolute (Flux)	10^{-9}
Relative (Flux)	10^{-9}
Absolute (Pressure)	n/a
Absolute (Temperature)	n/a
Balance Criterion	
Absolute (Flux)	10^{-7}
Relative (Flux)	10^{-7}

Table 6.2: Parameter Variation Cases

(a) Variation of Cell Numbers

Case	Number of Cells
C111	10
C112	50
C113	100
C114	500
C115	1000

(b) Variation of Flux Limit Application

Case	Flux Limit
C121	off
C122	soft ($q = 0.2$)
C123	soft ($q = 0.5$)
C124	soft ($q = 1$)
C125	soft ($q = 2$)
C126	soft ($q = 4$)
C127	soft ($q = 8$)

(c) Variation of Cell Property Evaluation Mode

Case	Number of Cells	Property Evaluation
C131	10	arithmetic avg.
C132	50	arithmetic avg.
C133	100	arithmetic avg.
C134	500	arithmetic avg.
C135	1000	arithmetic avg.

Case	Number of Cells	Property Evaluation
C141	10	logarithmic avg.
C142	50	logarithmic avg.
C143	100	logarithmic avg.
C144	500	logarithmic avg.
C145	1000	logarithmic avg.

(d) Ideal and Real Gas Properties at Different Pressure Levels

Case	p_{Feed} (Pa)	Property Calculation
C151	11×10^5	ideal
C152	11×10^5	real
C153	51×10^5	ideal
C154	51×10^5	real
C155	91×10^5	ideal
C156	91×10^5	real

(e) Pressure Drop and Energy Balance Calculation

Case	Pressure Drop	Energy Balance
C161	off	off
C162	on	off
C163	off	on
C164	on	on

Case	Property Calculation
C161-C164	real

The considered numbers of cells are 10, 50, 100, 500, 1000, and 200 in the base case. First trials have shown that some of the parameters for the reduction of the relaxation factor (as described in section 4.1) should be linked to the number of cells for best calculation performance — this is also reflected in the definition of the base case in Table 6.1c. Therefore, these parameters are scaled with the cell numbers. This subsequently also applies for cases C131-C135 and C141-C145.

As with higher cell numbers the discretization errors shrink and therefore the results become more accurate, the aim of this variation is to identify the degree of error imposed by using lower numbers of cells. Taking into account that higher cell numbers significantly increase the computational effort, there is always a trade-off between accuracy and speed.

6.3 Variation of Flux Limit Application

Vecgp includes different methods of flux limitation. The reasoning why this flux limitation is necessary and how it is applied is described in detail in section 3.3.4. Additionally, using soft limiting with low values for the limiting exponent might allow emulation of concentration polarization effects, which are otherwise currently not considered mathematically in vecgp. section 5.2 shows that using a low limiting exponent allows fitting the simulation results nicely to those of a cfd based code that inherently includes concentration polarization effects.

It is therefore of interest to investigate the effect of different limiting exponents on the simulation results. This is done in cases C121-C127. The base case C100 represents the simulation with hard flux limit, C121 is calculated with disabled flux limit, and cases C122-C127 use a soft flux limit with increasing limiting exponents (0.2 to 8).

6.4 Variation of Cell Property Evaluation Mode

This parameter variation is similar to the variation of the cell numbers covered in section 6.2, the main difference being that now the cell property evaluation method is also varied. The different possibilities and the mathematical background for cell property evaluation methods are described in section 3.2.4. What is decided here is basically which value is used to represent the “contents” of a cell. This is especially important for, e.g., the calculation of transmembrane flux as this is a function of the cell contents on both sides of the membrane. As vecgp is based on the finite difference method (fdm), property values are only stored at nodes (between two cells), but not for cells. The default setting is to use the value of the downstream node for the cell, but it is also possible to select other evaluation modes within vecgp. Both arithmetic (cases C131-C135) and logarithmic (cases

C141-C145) averaging between the downstream and upstream node is covered in this variation and compared to the calculations with the default downstream evaluation (cases C111-C115). Each of the cases is calculated with cell numbers ranging from 10 to 1000 as described in section 7.2.

6.5 Ideal and Real Gas Properties at Different Pressure Levels

As described in section 4.2, VECGP allows the calculation to be performed with real gas properties by using interpolation tables. The main properties where real gas properties affect flux calculations (when exempting conversions to other units like volume flows) are the component partial pressures or fugacities. To better understand the degree of impact of these real gas properties, a comparison between calculations with ideal and real properties is performed in cases C151 to C156. As the non-ideality of gas properties is heavily dependent on the total pressure, the comparison is made at three feed pressure levels: 11 bar (C151 and C152), 51 bar (C153 and C154), and 91 bar (C155 and C156).

6.6 Pressure Drop and Energy Balance Calculation

The final parameter variation is enabling pressure drop and energy balance calculation, as described in detail in sections 3.4 and 3.5. As both these calculations require various properties of the gases, these calculations were only performed with enabled real gas properties. It has to be noted though, that simulations without real gas properties are possible as well, as VECGP allows calculating the required properties with simple empirical formulae (see section 4.2).

Case C161 serves as the reference here with only the flux calculation enabled (and thus being essentially the same as case C154). In case C162, pressure drop calculation is enabled, whereas in case C163, energy balance calculation is enabled. Finally, case C164 shows the results with all calculations enabled.

It has to be noted, that pressure drop and energy balance calculation are enabled delayed, as is described in section 4.3. Specifically, this means that first solely the flux calculation is performed and once this reaches convergence, the additional calculations are enabled. This allows for a more stable calculation. In the case with both pressure drop and energy balance calculation, they are enabled successively: first just flux, then flux and pressure drop, and finally all three calculations.

7 Results and Discussion

7.1 Overview

The relevant calculation results (stage cut, hydrogen recovery, flow and composition of permeate and retentate) as well as some performance characteristics (calculation time and number of iterations) for all cases defined in chapter 6 are listed in Table 7.1 and will be referenced in the following sections. Additionally, Fig. 7.1 to Fig. 7.3 visually show the required calculation times and numbers of iterations to reach convergence as well as time per 1000 iterations.

Case C100 represents the base case as defined in chapter 6 — all the results in the following sections should be compared to this case.

7.2 Variation of Cell Numbers

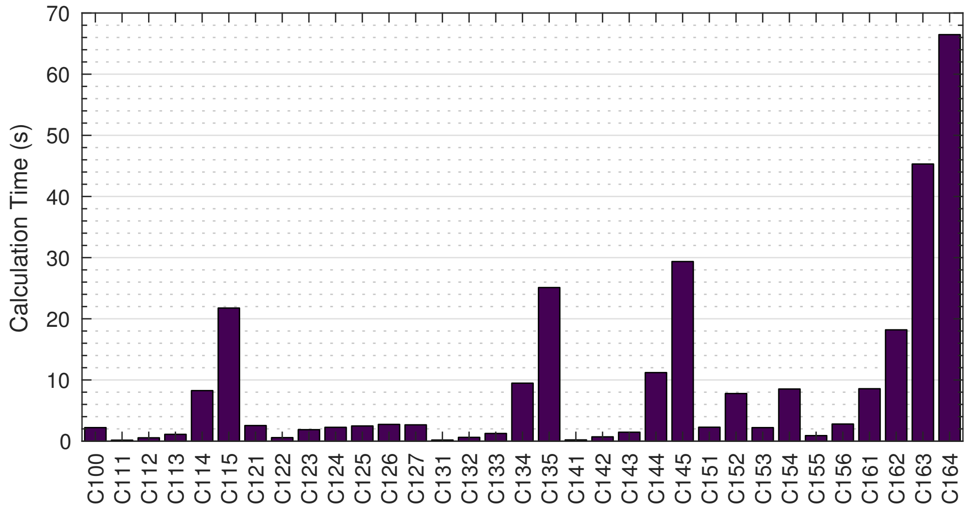
The variation of cell numbers was described in section 6.2. It can be seen in Table 7.1 that both the permeate flow as well as the hydrogen fraction in the permeate increase with higher cell numbers leading to higher stage cuts and hydrogen recoveries. This means that by using small cell numbers, the separation performance of the module is underestimated. Compared to 1000 cells, the calculated hydrogen recovery is around 6 % lower when using 10 cells and 0.6 % lower when using 100 cells. The stage cut is affected as well but to a lesser extent.

From a performance perspective, it can clearly be seen in Fig. 7.1 and Fig. 7.2 that higher cell numbers significantly increase the required computational effort. Going from 100 to 1000 cells increases the required time by a factor of nearly 20. As already identified in section 5.1.4, the calculation itself scales very well with increasing cell numbers. The required time per iteration only roughly doubles when increasing the number of cells from 100 to 1000 even though ten times more calculations are required. The main factor driving the longer overall calculation times is the higher number of iterations required to reach convergence. As cell values can only "travel" by one cell per iteration, an increased number of cells means it takes more iterations for information to propagate from one end to the other. Therefore, the required number of iterations scales mostly linear with the number of cells.

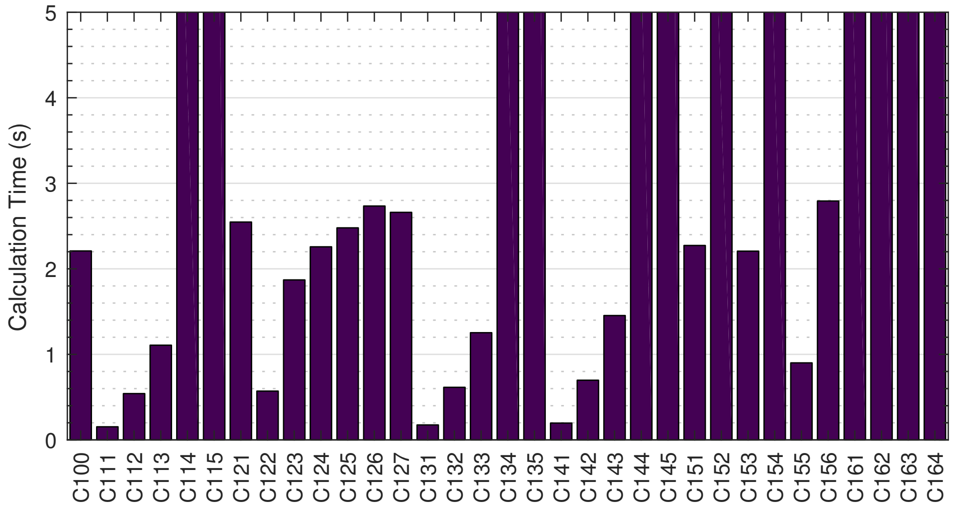
In light of the significant impact of the cell number on the performance, it has to be once more emphasized that before performing a large number of calculations

Table 7.1: Simulation Results – Base Case and Parameter Variation

Case	SC	Rec_{H_2}	$\dot{n}_{Perm}$	x_{Perm,H_2}	$\dot{n}_{Ret}$	x_{Ret,H_2}	Time	Iterations
	(–)	(–)	(mol s^{-1})	(–)	(mol s^{-1})	(–)	(s)	(–)
C100	0.179	0.910	5.20×10^{-4}	0.254	2.38×10^{-3}	0.005	2.21	6814
C111	0.177	0.859	5.13×10^{-4}	0.243	2.39×10^{-3}	0.009	0.15	508
C112	0.179	0.901	5.19×10^{-4}	0.252	2.38×10^{-3}	0.006	0.54	1925
C113	0.179	0.907	5.20×10^{-4}	0.253	2.38×10^{-3}	0.006	1.11	3697
C114	0.180	0.912	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	8.26	17 064
C115	0.180	0.913	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	21.76	34 217
C121	0.179	0.910	5.20×10^{-4}	0.254	2.38×10^{-3}	0.005	2.55	8549
C122	0.065	0.382	1.89×10^{-4}	0.293	2.71×10^{-3}	0.033	0.57	1411
C123	0.170	0.871	4.94×10^{-4}	0.256	2.41×10^{-3}	0.008	1.87	4920
C124	0.179	0.908	5.20×10^{-4}	0.253	2.38×10^{-3}	0.006	2.26	6372
C125	0.179	0.910	5.20×10^{-4}	0.254	2.38×10^{-3}	0.005	2.48	6604
C126	0.179	0.910	5.20×10^{-4}	0.254	2.38×10^{-3}	0.005	2.73	6742
C127	0.179	0.910	5.20×10^{-4}	0.254	2.38×10^{-3}	0.005	2.66	6814
C131	0.180	0.912	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	0.18	523
C132	0.180	0.913	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	0.61	1939
C133	0.180	0.913	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	1.25	3762
C134	0.180	0.913	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	9.47	17 146
C135	0.180	0.913	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	25.11	34 277
C141	0.179	0.911	5.20×10^{-4}	0.254	2.38×10^{-3}	0.005	0.20	515
C142	0.180	0.913	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	0.70	1938
C143	0.180	0.913	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	1.45	3761
C144	0.180	0.913	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	11.20	17 147
C145	0.180	0.913	5.21×10^{-4}	0.254	2.38×10^{-3}	0.005	29.36	34 277
C151	0.020	0.042	5.70×10^{-5}	0.107	2.84×10^{-3}	0.049	2.27	6917
C152	0.019	0.042	5.58×10^{-5}	0.108	2.84×10^{-3}	0.049	7.78	6948
C153	0.179	0.910	5.20×10^{-4}	0.254	2.38×10^{-3}	0.005	2.21	6814
C154	0.169	0.927	4.89×10^{-4}	0.275	2.41×10^{-3}	0.004	8.52	7530
C155	0.301	1.000	8.73×10^{-4}	0.166	2.03×10^{-3}	0.000	0.90	2836
C156	0.265	1.000	7.69×10^{-4}	0.188	2.13×10^{-3}	0.000	2.79	2517
C161	0.169	0.927	4.89×10^{-4}	0.275	2.41×10^{-3}	0.004	8.57	7530
C162	0.169	0.927	4.89×10^{-4}	0.275	2.41×10^{-3}	0.004	18.20	12 169
C163	0.169	0.927	4.90×10^{-4}	0.275	2.41×10^{-3}	0.004	45.31	14 628
C164	0.169	0.927	4.89×10^{-4}	0.275	2.41×10^{-3}	0.004	66.47	20 039

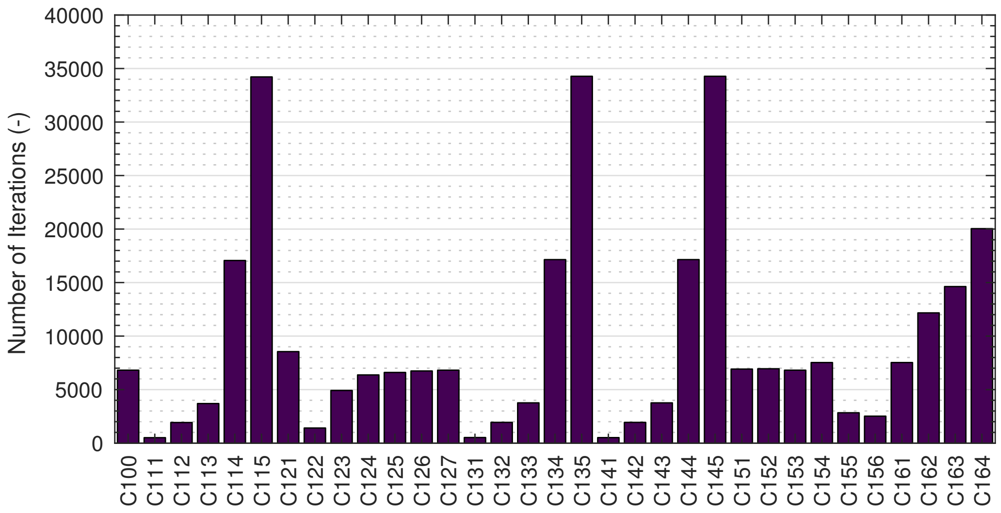


(a) Wide Y-Axis Range

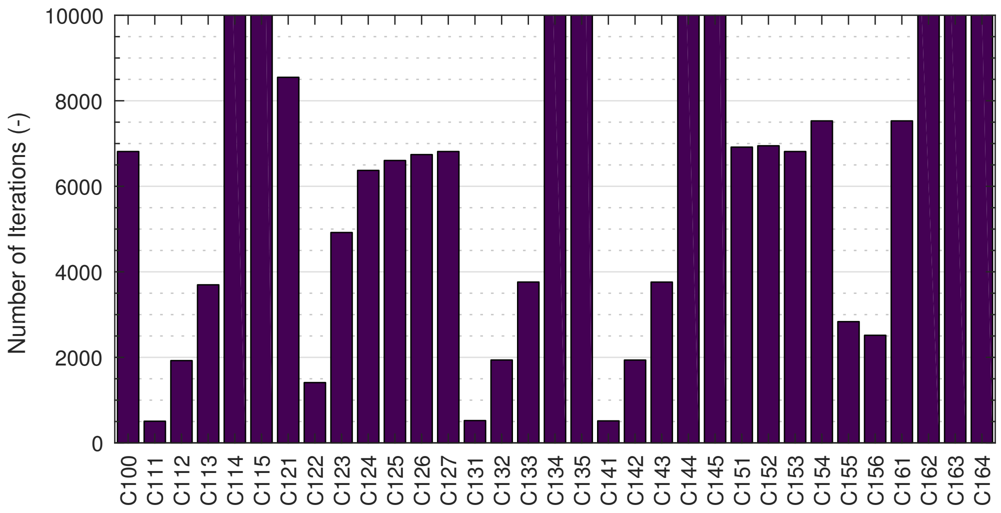


(b) Narrow Y-Axis Range

Figure 7.1: Comparison of Calculation Performance – Calculation Time

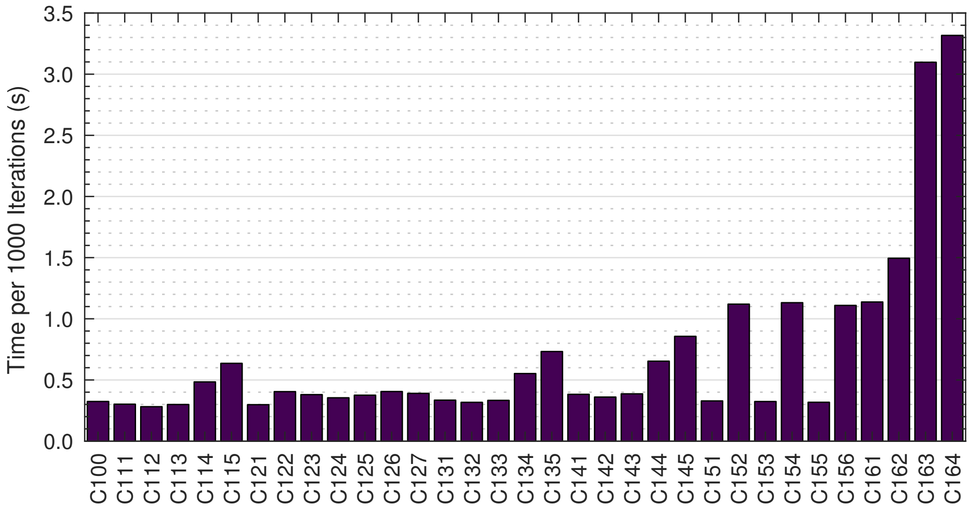


(a) Wide Y-Axis Range

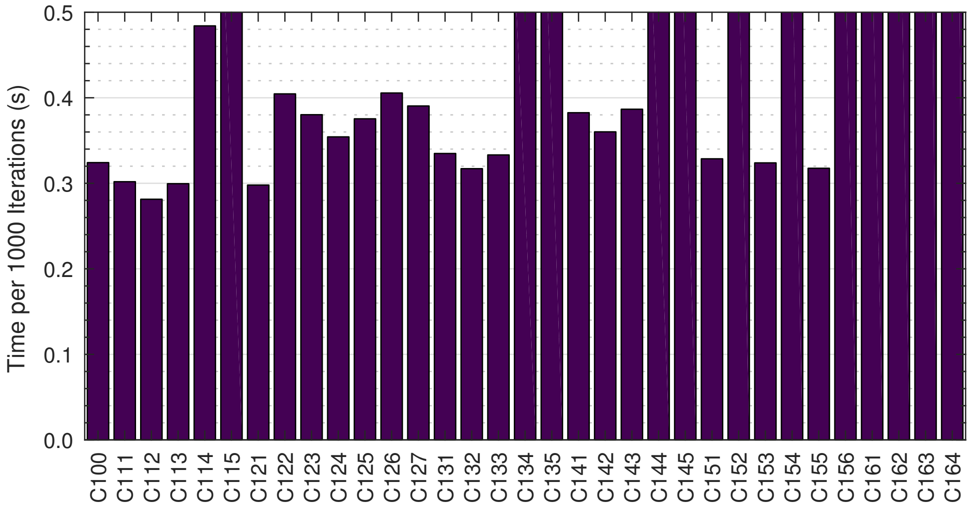


(b) Narrow Y-Axis Range

Figure 7.2: Comparison of Calculation Performance – Number of Iterations



(a) Wide Y-Axis Range



(b) Narrow Y-Axis Range

Figure 7.3: Comparison of Calculation Performance – Time per Iteration

(especially when using outer search loops), a sensible cell number has to be chosen. It should be suitable for the required level of accuracy but not unnecessarily large.

Finally, it has to be noted that here the evaluation of cell properties (as described in detail in section 3.2.4) is always performed on the downstream node. In section 7.4, the variation of cell numbers is performed with different methods of cell property evaluation which shows somewhat different results.

7.3 Variation of Flux Limit Application

The calculation results for the variation of flux limit application (described in section 6.3) can be seen as cases C121-C127in Table 7.1. The first observation to be made is that for this case, flux limit is not required to reach a convergent solution, as both case C100 (hard limit) and C121 (no limit) lead to identical results (within the shown precision). As the recovery of the faster permeating component (hydrogen) is less than 1 and the stage cut is moderate at around 0.2, no component runs "empty" in either the feed or permeate channel. It can be seen, though, that the hard limit leads to a significantly lower number of iterations and also an overall shorter calculation time. Thus it can be concluded that the computational effort required for the hard flux limit also pays off in easy cases, which means that leaving (at least) the hard limit on by default is recommended for all cases.

Taking into account the soft limit as well, it can be observed that cases with an exponent smaller than 1, namely C122 and C123, show significantly different results, whereas exponents greater than 2 (C125-C127) lead to essentially the same results as with hard limiting (or without any limit).

While leading to the same results with higher exponents, the soft limit is computationally slightly more expensive per iteration, as can be seen in Fig. 7.3. The additional "dampening" applied by the soft limit on the other hand leads to a lower number of iterations, which can partially compensate for the additional effort. Given that the results are very close, no universal recommendation can be given whether a hard or soft limit is more favorable. The decision has to be made on a case to case basis, but as a general starting point the hard limit seems to provide overall good and fast results.

7.4 Variation of Cell Property Evaluation Mode

The results for the variation of cell property evaluation modes are shown as cases C131 to C135, C141 to C145 and for reference C111 to C115 of Table 7.1. It can be observed, that the results between 50 and 1000 cells are indistinguishable at the shown level of precision if any kind of averaging is applied. Compared to the results of the downstream evaluation, a significantly higher level of accuracy can be reached with low cell numbers.

While the averaging leads to a somewhat increased calculation time at the same number of cells, this disadvantage is far outweighed by the fact that lower cell numbers can be used. Given that arithmetic averaging is computationally less expensive than logarithmic averaging while providing the better results, it is safe to say that arithmetic averaging with a rather low number of cells (in the range of 50 to 100) provides for a reasonably accurate and fast default setting.

7.5 Ideal and Real Gas Properties at Different Pressure Levels

Cases C151-C156 of Table 7.1 show that at 11 bar (C151 and C152), the differences between ideal and real gas properties are very small, with the permeate flow being reduced by around 2 % when using real gas properties. At 51 bar (C153 and C154), this difference grows to approximately 6 % and at 91 bar (C155 and C156) to 12 %.

For this specific system of hydrogen and methane, the recovery of hydrogen typically increases when using real gas properties, as the fugacity coefficients are greater than 1 for hydrogen and smaller than 1 for methane. Therefore, the actual driving force through the membrane compared to the ideal case is higher for hydrogen and lower for methane. As the majority of the feed is methane, this also explains the lower overall stage cut when using real gas properties.

From a performance perspective, enabling real gas properties significantly increases the required computational effort, as the properties have to be looked up for every single cell in every iteration. Overall, this leads to 3 to 4 times longer computation times, whereas the impact on the required number of iterations does not show a clear trend. Iteration numbers can both increase or decrease by a small margin.

The experimental validation of MAKARUK was only performed at relatively low pressure levels of 9 bar (Makaruk and Harasek 2009) where, as can be seen above, real gas effects do not play a big role. As VECGP has so far only been validated against MAKARUK, these results have to be considered unvalidated. Therefore, further comparison with experimental data at high pressures will be required in subsequent work.

7.6 Pressure Drop and Energy Balance Calculation

The results for pressure drop and energy balance calculation are shown as cases C161 to C164 in Table 7.1. It can be seen that the additional enabled calculations do not change the main results within the shown order of accuracy for this case. Enabling energy balance calculation does change the permeate temperature significantly though. Instead of just "copying" the 300 K from the feed, a value of

288 K is calculated for the permeate outlet. At the retentate end of the module, an even lower temperature of 281 K is calculated in the permeate channel. As the current permeance model does not employ temperature dependency, though, these changes in temperature do not really affect the flux. The sole impact of the changed temperature on the fluid properties (e.g., viscosities and densities) has only very little effect on the calculated results.

The observations from this case might not be applicable in general, though, as the effect of the additional calculations depends *heavily* on the specific case. For example, small changes to the module geometry like making the fiber walls slightly thicker and the inner fiber diameter smaller severely affect the calculated pressure drop in the fiber. As there are a large number of geometry parameters, a thorough evaluation of how the calculation is affected by different module geometries would go beyond the scope of this thesis.

It can be clearly observed, though, that enabling pressure drop and energy balance calculation significantly increases both the required computation time and number of iterations. While pressure drop calculation roughly doubles the required calculation time, energy balance alone leads to an increase of the required time by a factor of 5. With all calculations enabled, this factor grows to over 7.

In Figures 7.2 and 7.3 it can be seen that the time increase for pressure drop calculation is mainly driven by the higher number of required iterations, whereas the energy balance drastically increases the time per iteration.

8 Conclusion and Outlook

In chapter 5 it is shown, that VECGP with basic settings (without enabled real gas properties, pressure drop, or energy balance calculation), delivers results practically identical to the existing MAKARUK. As the existing code is experimentally validated, it can be concluded, that VECGP is validated within the same limits. In terms of calculation performance, VECGP is significantly faster than MAKARUK for complex cases and its advantage generally increases with higher complexity (that is higher numbers of cells or components).

Compared to the CFD-based code, there are noticable differences between the calculated mass fractions and mass flow rates when using VECGP with the default flux limit settings. These are assumed to be attributable to the significantly different calculation approach (as described in section 5.2). By setting the flux limit parameter to rather low values, it is possible to closely match the calculated results for the specific case.

The pressure drop calculated by VECGP was, though qualitatively similar, substantially lower than the one obtained by the CFD-based code. The reasons for this difference are not yet fully clear and further investigation is required.

The most notable conclusion that can be drawn from the parameter variation is, that the number of cells drastically impacts calculation performance and that choosing a suitable value is essential to ensure efficient calculation. Using averaging within cell property evaluation allows setting far lower cell numbers while still resulting in good accuracy. In the defined test case, 50 cells with arithmetic averaging led to results as accurate as 1000 cells without averaging.

In terms of advanced calculation, the test cases show that using real gas properties can significantly change the results. The impact of the pressure drop and energy balance calculation was low, but it was established that these depend strongly on the specific parameters. The findings from this single common base case can therefore not be assumed to cover all possible results. All these additional calculations do drastically increase the required computational effort and should therefore only be enabled when required.

Looking forward, quite a few areas can be identified, where further research and (code) development is desirable. Additional validation of the algorithm is required for the substantial new features of the code (real gas properties, pressure drop, and energy balance calculation) as well as many of the small elements (e.g., flux limit, evaluation of cell properties, variable permeate outlet, convergence settings). These validations will most likely have to involve comparisons with other simulation results and experimental data.

For adapting or expanding the algorithm there are a few logical next steps, for which the groundwork has already partly been laid in this work. The consideration of real gas properties and the calculation of pressure drops and energy balances all enable or simplify the inclusion of advanced permeance models. These models could include temperature and pressure dependencies as well as flux coupling (dependency of the permeance for one substance on the presence of other substances). The strict focus on calculation stability in VECGP also acts as an enabler here, as it has been observed that reaching a stable calculation becomes more challenging when increasing the complexity of the simulation.

Additionally, the models for pressure drop and especially energy balance calculation are still rather rudimentary, and slight expansions might be required to allow for versatile calculations. Further modifications might be required if additional issues can be identified during more extensive validation.

In terms of adaption and expansion of the code, some ideas for improvements have also been identified. Many of these are in the area of usability. Currently, the implementation consists mostly of a library of classes that perform the calculations. The front-end is just a few example files that describe possible ways how to use these libraries. In order to allow easier use, a significant expansion of the front-end might be desired, especially considering the calculation of multi-stage processes and networks of multiple differently linked stages.

Overall, the main objectives of this have been reached clearly: Many new features have been implemented and are in various states of maturity. The core of VECGP, the simulation of membrane separation, is well tested and serves as a stable and fast solution for the calculation of multi-component cases. The code is clearly structured and can be expanded easily.

List of Figures

List of Tables

List of Symbols

Symbol	Unit	Description
$\dot{n}$	$\mathrm{mol\,s^{-1}}$	mole flow rate
$\dot{Q}$	$\mathrm{mol\,s^{-1}}$	transmembrane mole flow rate
$\dot{V}$	$\mathrm{m^3\,s^{-1}}$	volume flow rate
$\dot{m}$	$\mathrm{kg\,s^{-1}}$	mass flow rate
$\dot{H}$	$\mathrm{J\,s^{-1}}$	enthalpy flow rate
$\dot{q}$	$\mathrm{J\,s^{-1}}$	transmembrane enthalpy flow rate
p	Pa	pressure
T	K	temperature
x	–	molar fraction
w	–	mass fraction
c	$\mathrm{mol\,m^{-3}}$	concentration
M	$\mathrm{kg\,kmol^{-1}}$	molar mass
φ	–	fugacity coefficient
v^m	$\mathrm{m^3\,mol^{-1}}$	molar volume
ρ^m	$\mathrm{mol\,m^{-3}}$	molar density
ρ	$\mathrm{kg\,m^{-3}}$	density
h^m	$\mathrm{J\,mol^{-1}}$	molar enthalpy
R	$\mathrm{J\,mol^{-1}\,K^{-1}}$	universal gas constant
c_p^m	$\mathrm{J\,mol^{-1}\,K^{-1}}$	heat capacity at constant pressure
c_v^m	$\mathrm{J\,mol^{-1}\,K^{-1}}$	heat capacity at constant volume
κ	–	heat capacity ratio
D	$\mathrm{m^2\,s^{-1}}$	diffusion coefficient
K	$\mathrm{mol\,m^{-3}\,Pa^{-1}}$	diffusion coefficient
Π	$\mathrm{mol\,s^{-1}\,m^{-2}\,Pa^{-1}}$	permeance of membrane
l	m	length of module
A	$\mathrm{m^2}$	total membrane area
D_I	m	inner fiber diameter (lumen)
D_O	m	outer fiber diameter
D_A	m	active layer diameter of fiber
D_H	m	shell side hydraulic diameter
D_M	m	inner module diameter
d_f	m	thickness of fiber
n_f	–	number of fibers

Symbol	**Unit**	**Description**
A^x	$\mathrm{m^2}$	cross sectional area
λ_f	$\mathrm{W\,m^{-1}\,s^{-1}}$	thermal conductivity of fiber
μ	$\mathrm{Pa\,s}$	dynamic viscosity
λ	$\mathrm{W\,m^{-1}\,s^{-1}}$	thermal conductivity of fluid
ϕ	$-$	mixing coefficient
α	$\mathrm{W\,m^{-2}\,K^{-1}}$	heat transfer coefficient
Nu	$-$	Nusselt number
Re	$-$	Reynolds number
Pr	$-$	Prandtl number
SC	$-$	stage cut
Rec	$-$	recovery
Bal	$\mathrm{mol\,s^{-1}}$	mass balance
δ	$-$	convergence
ε	$-$	convergence criterion
ω	$-$	relaxation factor

Sub-/Superscript	**Description**
F	value at feed side
P	value at permeate side
b	bore side value
s	shell side value
$Feed$	value of feed
$Perm$	value of permeate
Ret	value of retentate
$Sweep_0$	value of co-current sweep gas
$Sweep_1$	value of counter-current sweep gas
i	cell index
j	component index
k	iteration index
n	total number of cells
m	total number of components
out	index of permeate outlet cell
$'$	length specific
$''$	area specific
int	value taken from interpolation table

List of Acronyms

BVP boundary value problem

CFD computational fluid dynamics

DE differential equation

FDM finite difference method

IVP initial value problem

LHS left-hand side

MAKARUK refers to the algorithm presented by Makaruk and Harasek (2009)

ODE ordinary differential equation

PDE partial differential equation

RHS right-hand side

STP standard temperature and pressure

VECGP refers to the algorithm presented in this thesis and its reference implementation (vectorized gas permeation)

References

Abels, C. et al. (2013). “Membrane processes in biorefinery applications”. In: *Journal of Membrane Science* 444, pp. 285–317. DOI: 10.1016/j.memsci.2013.05.030.

Adsuara, J. E. et al. (2016). “Scheduled Relaxation Jacobi method: Improvements and applications”. In: *Journal of Computational Physics* 321, pp. 369–413. DOI: 10.1016/j.jcp.2016.05.053. URL: http://linkinghub.elsevier.com/retrieve/pii/S002199911630198X.

Agrawal, R. and Xu, J. (1996). “Gas separation membrane cascades II. Two-compressor cascades”. In: *Journal of Membrane Science* 112.2, pp. 129–146. DOI: 10.1016/0376-7388(95)00273-1.

Ahmad, F. et al. (2013). “Temperature and pressure dependence of membrane permeance and its effect on process economics of hollow fiber gas separation system”. In: *Journal of Membrane Science* 430, pp. 44–55. DOI: 10.1016/j.memsci.2012.11.070.

Ahmad, F. et al. (2015). “Hollow fiber membrane model for gas separation: Process simulation, experimental validation and module characteristics study”. In: *Journal of Industrial and Engineering Chemistry* 21. DOI: 10.1016/j.jiec.2014.05.041.

Ahsan, M. and Hussain, A. (2015). “Mathematical Modeling of Helium Recovery from a Multicomponent Fuel Gas with Polymeric Membrane”. In: *International Journal of Chemical Engineering and Applications* 6.3, pp. 173–178. DOI: 10.7763/IJCEA.2015.V6.476.

Alkhamis, N. et al. (2013). “Gas Separation Using a Membrane”. In: *ASME 2013 International Mechanical Engineering Congress and Exposition*. ASME, V07AT08A039. DOI: 10.1115/IMECE2013-62764.

Alkhamis, N. et al. (2015). “Computational study of gas separation using a hollow fiber membrane”. In: *International Journal of Heat and Mass Transfer* 89, pp. 749–759. DOI: 10.1016/j.ijheatmasstransfer.2015.05.090.

Baker, R. W. (2012). *Membrane Technology and Applications*. 3rd ed. Chichester: Wiley. DOI: 10.1002/0470020393.

Baker, R. W. and Low, B. T. (2014). “Gas Separation Membrane Materials: A Perspective”. In: *Macromolecules* 47.20, pp. 6999–7013. DOI: 10.1021/ma501488s.

Bärwolff, G. (2016). *Numerik für Ingenieure, Physiker und Informatiker*. 2. Aufl. Berlin Heidelberg: Springer Spektrum, S. 364. DOI: 10.1007/978-3-662-48016-8.

Bell, I. H. et al. (2014). “Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp”.

In: *Industrial & Engineering Chemistry Research* 53.6, pp. 2498–2508. DOI: 10.1021/ie4033999.

Binns, M. et al. (2016). "Strategies for the simulation of multi-component hollow fibre multi-stage membrane gas separation systems". In: *Journal of Membrane Science* 497, pp. 458–471. DOI: 10.1016/j.memsci.2015.08.023.

BMVIT – Bundesministerium für Verkehr, Innovation und Technologie, Hrsg. (2015). *HylyPure - Grünen Wasserstoff energieeffizient rückgewinnen*. URL: http://www.energy-innovation-austria.at/wp-content/uploads/2015/07/eia_02_15_D_FIN.pdf.

Bornemann, F. (2016). *Numerische lineare Algebra*. Springer Spektrum, S. 145. DOI: 10.1007/978-3-658-12884-5.

Bounaceur, R. et al. (2017). "Rigorous variable permeability modelling and process simulation for the design of polymeric membrane gas separation units: MEMSIC simulation tool". In: *Journal of Membrane Science* 523, pp. 77–91. DOI: 10.1016/j.memsci.2016.09.011.

Bretsznajder, S. (1971). *Prediction of Transport and Other Physical Properties of Fluids*. 1st ed. Oxford: Pergamon Press.

Chen, X. Y. et al. (2015). "Membrane gas separation technologies for biogas upgrading". In: *RSC Adv.* 5.31, pp. 24399–24448. DOI: 10.1039/C5RA00666J.

Crank, J. et al. (1947). "A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 43.01, p. 50. DOI: 10.1017/S0305004100023197. URL: http://www.journals.cambridge.org/abstract%7B%5C_%7DS0305004100023197.

Dahmen, W. und Reusken, A. (2008). *Numerik für Ingenieure und Naturwissenschaftler*. 2. Aufl. Berlin Heidelberg: Springer. DOI: 10.1007/978-3-540-76493-9.

Deuflhard, P. and Hohmann, A. (2003). *Numerical Analysis in Modern Scientific Computing*. New York: Springer. DOI: 10.1007/978-0-387-21584-6.

DVGW – Deutscher Verein des Gas- und Wasserfaches e.V. (2013). *G 260*.

Feichtinger, A. et al. (2014). "Collocation method for the modeling of membrane gas permeation systems". In: *International Journal of Nonlinear Sciences and Numerical Simulation* 15.5, pp. 307–316. DOI: 10.1515/ijnsns-2013-0113.

Freund, R. W. und Hoppe, R. H. (2007). *Stoer/Bulirsch: Numerische Mathematik 1*. 10. Aufl. Berlin Heidelberg New York: Springer.

Grossmann, C. et al. (2007). *Numerical Treatment of Partial Differential Equations*. Berlin Heidelberg New York: Springer.

Haddadi Sisakht, B. et al. (2016). "Designing Better Membrane Modules Using CFD". In: *Chemical Product and Process Modeling* 11.1, pp. 57–66. DOI: 10.1515/cppm-2015-0066.

Hosseini, S. S. et al. (2016a). "Gas permeation and separation in asymmetric hollow fiber membrane permeators: Mathematical modeling, sensitivity analysis and optimization". In: *Korean Journal of Chemical Engineering* 33.11, pp. 3085–3101. DOI: 10.1007/s11814-016-0198-z.

Hosseini, S. S. et al. (2016b). "Mathematical Modeling and Investigation on the Temperature and Pressure Dependency of Permeation and Membrane Separation Performance for Natural gas Treatment". In: *Chemical Product and Process Modeling* 11.1, pp. 7–10. DOI: 10.1515/cppm-2015-0051.

Kundu, P. K. et al. (2013). "Modelling of multicomponent gas separation with asymmetric hollow fibre membranes-methane enrichment from biogas". In: *The Canadian Journal of Chemical Engineering* 91.6, pp. 1092–1102. DOI: 10.1002/cjce.21721.

Kundu, P. et al. (2013). "Analysis of permeate pressure build-up effects on separation performance of asymmetric hollow fiber membranes". In: *Chemical Engineering Science* 104, pp. 849–856. DOI: 10.1016/j.ces.2013.10.003.

Larsson, S. and Thomée, V. (2009). *Partial Differntial Equations with Numerical Methods*. Berlin Heidelberg: Springer. DOI: 10.1007/978-3-540-88706-5.

Lassmann, T. G. (2015). "The purification of fermentatively produced hydrogen using gas permeation: A practical and simulative approach". PhD thesis. TU Wien. URL: http://www.ub.tuwien.ac.at/diss/AC13025950.pdf.

Lassmann, T. et al. (2016). "The purification of fermentatively produced hydrogen using membrane technology: a simulation based on small-scale pilot plant results". In: *Clean Technologies and Environmental Policy* 18.1, pp. 315–322. DOI: 10.1007/s10098-015-0997-7.

Lemmon, E. W. et al. (2013). *NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties - REFPROP, Version 9.1*. Gaithersburg. URL: https://www.nist.gov/srd/refprop.

Liemberger, W. et al. (2016). "Extraction of green hydrogen at fuel cell quality from mixtures with natural gas". In: *Chemical Engineering Transactions* 52, pp. 427–432. DOI: 10.3303/CET1652072.

Liemberger, W. et al. (2017). "Experimental analysis of membrane and pressure swing adsorption (PSA) for the hydrogen separation from natural gas". In: *Journal of Cleaner Production* 167, pp. 896–907. DOI: 10.1016/j.jclepro.2017.08.012.

Lindfield, G. R. and Penny, J. E. T. (2012). *Numerical Methods using MATLAB*. 3rd ed. Academic Press, p. 552.

Lock, S. S. M. et al. (2015a). "Effect of recycle ratio on the cost of natural gas processing in countercurrent hollow fiber membrane system". In: *Journal of Industrial and Engineering Chemistry* 21, pp. 542–551. DOI: 10.1016/j.jiec.2014.03.017.

Lock, S. S. M. et al. (2015b). "Modeling, simulation and economic analysis of CO2 capture from natural gas using cocurrent, countercurrent and radial crossflow hollow fiber membrane". In: *International Journal of Greenhouse Gas Control* 36, pp. 114–134. DOI: 10.1016/j.ijggc.2015.02.014.

Lock, S. S. M. et al. (2014). "Mathematical Modeling of the Radial Crossflow Hollow Fiber Membrane Module for Multi-Component Gas Separation". In: *Applied*

Mechanics and Materials 625, pp. 726–729. DOI: 10.4028/www.scientific.net/AMM.625.726.

Magnanelli, E. et al. (2016). “Enhancing the understanding of heat and mass transport through a cellulose acetate membrane for CO2 separation”. In: *Journal of Membrane Science* 513, pp. 129–139. DOI: 10.1016/j.memsci.2016.04.021.

Makaruk, A. and Harasek, M. (2009). “Numerical algorithm for modelling multicomponent multipermeator systems”. In: *Journal of Membrane Science* 344.1-2, pp. 258–265. DOI: 10.1016/j.memsci.2009.08.013.

Makaruk, A. et al. (2010). “Membrane biogas upgrading processes for the production of natural gas substitute”. In: *Separation and Purification Technology* 74.1, pp. 83–92. DOI: 10.1016/j.seppur.2010.05.010.

Mathew, J. H. and Fink, K. D. (2004). *Numerical Methods using MATLAB*. Pearson Prentice Hall, p. 680.

Melin, T. und Rautenbach, R. (2007). *Membranverfahren*. 3. Aufl. VDI-Buch. Berlin Heidelberg: Springer. DOI: 10.1007/978-3-540-34328-8.

Ohenoja, M. and Sorsa, A. (2015). “Simulation as a Tool for Evaluating Biogas Purification Processes”. In: *Proceedings of the 56th SIMS*, pp. 55–61. DOI: 10.3384/ecp1511955.

ÖVGW – Österreichische Vereinigung für das Gas- und Wasserfach (2001). *G 31*.

Pathare, R. and Agrawal, R. (2010). “Design of membrane cascades for gas separation”. In: *Journal of Membrane Science* 364.1, pp. 263–277. DOI: 10.1016/j.memsci.2010.08.029.

Poling, B. E. et al. (2000). *The Properties of Gases and Liquids*. 5th ed. New York: McGraw-Hill, p. 793. DOI: 10.1036/0070116822.

Scholz, M. et al. (2013a). “Modeling Gas Permeation by Linking Nonideal Effects”. In: *Industrial & Engineering Chemistry Research* 52.3, pp. 1079–1088. DOI: 10.1021/ie202689m.

Scholz, M. et al. (2013b). “Transforming biogas into biomethane using membrane technology”. In: *Renewable and Sustainable Energy Reviews* 17, pp. 199–212. DOI: 10.1016/j.rser.2012.08.009.

Scholz, M. et al. (2015). “Dynamic process simulation and process control of biogas permeation processes”. In: *Journal of Membrane Science* 484, pp. 107–118. DOI: 10.1016/j.memsci.2015.03.008.

Schretter, P. (2016). “Simulation of Membrane Modules with OpenFOAM”. Master thesis. TU Wien. URL: http://www.ub.tuwien.ac.at/dipl/2016/AC13009611.pdf.

Sharifian, S. et al. (2016). “Simulation of Membrane Gas Separation Process Using Aspen Plus® V8.6”. In: *Chemical Product and Process Modeling* 11.1, pp. 67–72. DOI: 10.1515/cppm-2015-0067.

Szwast, M. and Szwast, Z. (2015). “A Mathematical Model of Membrane Gas Separation with Energy Transfer by Molecules of Gas Flowing in a Channel to

Molecules Penetrating this Channel from the Adjacent Channel". In: *Chemical and Process Engineering* 36.2, pp. 151–169. DOI: 10.1515/cpe-2015-0012.

The MathWorks, Inc. (2016). *MATLAB Documentation on Vectorization*. URL: https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html.

VDI – Gesellschaft Verfahrenstechnik und Chemieingenieurwesen, Hrsg. (2013). *VDI-Wärmeatlas*. Berlin Heidelberg: Springer. DOI: 10.1007/978-3-642-19981-3.

White, F. M. (2005). *Viscous Fluid Flow*. New York: McGraw-Hill.

Xu, J. and Agrawal, R. (1996a). "Gas separation membrane cascades I. One-compressor cascades with minimal exergy losses due to mixing". In: *Journal of Membrane Science* 112.2, pp. 115–128. DOI: 10.1016/0376-7388(95)00272-3.

Xu, J. and Agrawal, R. (1996b). "Membrane separation process analysis and design strategies based on thermodynamic efficiency of permeation". In: *Chemical Engineering Science* 51.3, pp. 365–385. DOI: 10.1016/0009-2509(95)00262-6.

Yang, X. I. and Mittal, R. (2014). "Acceleration of the Jacobi iterative method by factors exceeding 100 using scheduled relaxation". In: *Journal of Computational Physics* 274, pp. 695–708. DOI: 10.1016/j.jcp.2014.06.010. URL: http://linkinghub.elsevier.com/retrieve/pii/S0021999114004173.