NTNU
Norwegian University of
Science and Technology

Nonsmooth Modelling of Multiphase Multicomponent Heat Exchangers with Phase Changes

Marius Reed

Chemical Engineering and Biotechnology
Submission date: June 2018
Supervisor: Johannes Jäschke, IKP

Norwegian University of Science and Technology
Department of Chemical Engineering

Summary

Developing proper and consistent dynamic models of processes with phase changes can be challenging. Appearance and disappearance of phases result in changes in the system equations and is not known a priori. Dynamic models of such processes have to detect and adapt to these changes. In this thesis, a nonsmooth dynamic model, formulated as a DAE of index 1, of a multi-component heat exchanger is presented. The nonsmooth formulation both detects and adapts to phase changes by including a continuous extension of the phase mole fractions into the phase regime where the corresponding phase does not exist. In addition, the model is formulated in a way which allows reversal flow inside the heat exchanger. Similar to the handling of phase changes, this is done by using the nonsmooth functions $\min$ and $\max$. The heat exchanger is modeled as a number of flash tanks with constant volume in series.

This thesis presents the development of a multiphase multicomponent flash tank model. Further a single-sided heat exchanger with given heat transfer as well as a countercurrent heat exchanger is modeled as flash tanks in series. Results from dynamic simulations of a single flash tank, one side of a heat exchanger with given heat and a countercurrent heat exchanger are presented. During the simulations the heat exchanged, the inlet flow rates and the inlet temperatures are varied. By doing so, the phases both appears and disappears during the simulations. The results show that the proposed model is both able to detect and handle phase changes. Further, a shutdown of a heat exchanger has been performed, showing the models abilities to allow reversal flow.

For simulation purposes, the model is written and implemented in MATLAB®. To simulate the model, an implicit Euler integrator with fixed time step is used. At the nonsmooth points, the Jacobian is not defined. As a result, the generalized derivatives are calculated through automatic differentiation and is used as a replacement for the Jacobian. The generalized derivatives are computed by using a user-defined MATLAB®-object. These derivatives are supplied to the MATLAB®-solver *fsolve* which is used in the Euler integrator.

Sammendrag

Utvikling av dynamiske modeller for prosesser med faseendringer kan være utfordrende. Når en fase forsvinner eller oppstår vil det medføre endringer i ligningene som beskriver systemet. Problemet med slike faseendringer er at det ikke er kjent når disse skjer a priori. Dynamiske modeller av slike prosesser må oppdage og tilpasse seg disse endringene. I denne oppgaven presenteres en ikke-glatt dynamisk model, formulert som en DAE av indeks 1, av en mulitkomponentvarmeveksler. Den ikke-glatte formuleringen både oppdager og tilpasser seg til faseendringer ved å inkludere en kontinuerlig forlengelse av fasemolfraksjonene inn i faseområdet der den tilsvarende fasen ikke eksisterer. I tillegg er modellen formulert på en måte som tillater tilbakestrømning i varmeveksleren. På samme måte som håndtering av faseendringer, gjøres dette ved å bruke de ikke-glatte funksjonene $\min$ og $\max$. Varmeveksleren er modellert som en rekke flashtanker med konstant volum i serie.

Denne oppgaven presenterer utviklingen av en model av en multifase multikomponent flash tank. Videre er en enkeltsidig varmeveksler med gitt varmeoverføring, samt en motstrømsvarmeveksler modellert som flash tanker i serie. Resultatene fra dynamiske simulering av en enkelt flash-tank, en side av en varmeveksler med gitt varmeoverføring og en motstrømsvarmeveksler presenteres. Under simuleringene blir varmeoverføringen, innløpsstrømningsraten og innløpstemperaturene variert. Slik vil både faser oppstå og forsvinne i løpet av simuleringene. Resultatene viser at den foreslåtte modellen både kan oppdage og håndtere faseendringer. Videre er avstengning av en varmeveksler utført, og viser modellens evner til å tillate tilbakestrømning.

Modellen er skrevet og implementert i MATLAB®. For å simulere modellen, brukes en implisitt Euler integreringsmetode med konstant tidsteg. I ikke-glatte punkter er Jacobian ikke definert. På grunn av dette beregnes de generaliserte deriverte gjennom automatisk differensiering og brukes som en erstatning for Jacobian. De generaliserte deriverte beregnes ved å bruke et brukerdefinert MATLAB®-objekt. Disse deriverte sendes til MATLAB®-løseren *fsolve* som brukes i Euler integreringsmetoden.

Preface

This master thesis was written in the spring of 2018. The thesis concludes the 5-year master's degree programme Chemical Engineering and Biotechnology at the Norwegian University and Technology. The thesis is written at the Department of Chemical Engineering, more specifically in the Process Systems Engineering group.

I would like to thank my supervisor Associate Professor Johannes Jäschke for all the guidance and good discussions during the last year. I am really grateful for the opportunity to work with this exciting project. I would also like to thank Marlene L. Lund for helping me during the introduction to nonsmooth analysis as part of my specialization project. It has been very helpful during this master thesis.

Further, I would like to thank all of my friends here in Trondheim for all the good memories, both in and outside of the study halls. Finally, I want to express my gratitude for all the encouragement from my family and Monica.

Declaration of Compliance

I hereby declare that this is an independent work according to the exam regulations of the Norwegian University of Technology.

Trondheim, June 2018
Marius Reed

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

DAE	Differential algebraic equation
HEX	Heat exchanger
LD	Lexicographic directional
NC	Number of components
OOP	Object-oriented programming
VLE	Vapor-liquid equilibrium

Mathematical notation

$(\mathbf{f} \circ \mathbf{g})$	Composite function, $\mathbf{f}(\mathbf{g}(\mathbf{x})$
$\equiv$	Equivalent to
$\exists$	Exists
$\forall$	For all
$\in$	Element of
κ	Condition number of a matrix.
$\mathbb{N}$	Space of natural numbers
$\mathbb{R}^n$	Eucledian space of dimension n
$\mathbf{A}$	Bold upper case letter denotes a matrix
$\mathbf{a}$	Bold lower case letter denotes a vector
$\mathbf{A^T}$	Transpose of matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	Inverse of matrix $\mathbf{A}$

$\mathbf{a}_{(i)}$	Column i of a matrix $\mathbf{A}$
$\mathbf{f}'(\mathbf{x})$	Derivative of $\mathbf{f}$ at $\mathbf{x}$
$\mathbf{f}'(\mathbf{x};\mathbf{M})$	Lexicographic directional derivative of $\mathbf{f}$ at $\mathbf{x}$ in direction $\mathbf{M}$
$\mathbf{Jf}(\mathbf{x})$	Jacobian of $\mathbf{f}$ at $\mathbf{x}$
$\mathbf{Jf}(\mathbf{x};\mathbf{M})$	Lexicographic derivative of $\mathbf{f}$ at $\mathbf{x}$ in direction $\mathbf{M}$
$\mathcal{C}^1$	Continuous differentiable
$\mathcal{PC}^1$	Piecewise differentiable
$\mathcal{PL}$	Piecewise linear
$\mathrm{det}\mathbf{M}$	Determinant of $\mathbf{M}$
$\|\cdot\|$	Unspecified norm
$\stackrel{\mathrm{def}}{=}$	Defined as
$\partial\mathbf{f}$	Clarke Jacobian of $\mathbf{f}$ at $\mathbf{x}$
$\partial_B\mathbf{f}(\mathbf{x})$	B-subdifferential of $\mathbf{f}$ at $\mathbf{x}$
$\partial_L\mathbf{f}(\mathbf{x})$	Lexicographic subdifferential of $\mathbf{f}$ at $\mathbf{x}$
$\partial_P\mathbf{f}(\mathbf{x})$	The plenary Jacobian
π	Projection
$\subset$	Subset of
sup	Supremum of
$\supset$	Superset of
$A \to B$	Mapping from A to B
A	Upper case letter denotes a set
a	Lower case letter denotes a scalar
$a^{(i)}$	Order of directional derivative or iteration i
a_{ij}	Element in row i, column j in matrix $\mathbf{A}$
$conv(S)$	Convex hull of the set S

Symbols

γ	Activity coefficient	-
$\mathbf{z}$	Mole fraction in stream	-
μ	Chemical potential	mol
Φ	A general quantity	-

ϕ	Fugacity coefficient	-
ρ	Molar density	mol m^{-3}
A	Antoine parameter	-
B	Antoine parameter	-
C	Antoine parameter	-
C	Heat capacity	J mol^{-1} K^{-1}
c	Vapor valve coefficient	mol s^{-1} $Pa^{-0.5}$
C_0	Compressability factor	mol m^{-3} Pa^{-1}
F	Flow rate	mol·s^{-1}
G	Gibbs energy	J
H	Enthalpy	J
h	Molar enthalpy	J mol^{-1}
K	Phase equilibrium constant	-
M	Molar hold up	mol
m	Mass	kg
N	Mole	mol
p	Pressure	Pa
Q	Heat	J s^{-1}
R	Gas constant	J mol^{-1} K^{-1}
S	Entropy	J K^{-1}
T	Temperature	K
U	Internal Energy	J
UA	Heat transfer coefficient times the surface area	J s^{-1} K^{-1}
V	Volume	m^3
v	Molar volume	m^3 mol^{-1}
x	Mole fraction in liquid phase	-
y	Mole fraction in vapor phase	-

Superscripts

α	Phase α
β	Phase β

C	Cold side of a HEX
H	Cold side of a HEX
j	Cell j of a HEX
$Raoult$	Using Raoults law
sat	At saturation

Subscripts

0	Outlet
n	Constant composition
D	Discretized function
i	Component i
in	Inlet
L	Liquid phase
max	Maximum
out	Outlet
p	Constant pressure
ref	Reference value
T	Constant temperature
t	At time t
tot	Total
V	Vapor phase
v	Constant volume
vap	Vaporization

Chapter 1

Introduction

Heat exchangers are a very common unit within process systems. Processes like refrigeration cycles, petroleum refining, and wastewater treatment are only a few examples of where heat exchangers are used. As they are very common they are present in many models of parts of a process and entire plants. Therefore, making a good model of a heat exchanger can increase the overall quality of many process models. Dynamic models are used both in process operations and -design. The dynamic models can be used to study the behavior of processes during startup and shutdown, perform optimization, train operators in addition to other applications[1]. In specific, models of heat exchangers are important as they can be used to minimize the energy consumption of refrigeration cycles, thereby reducing the operational costs[2].

Within modeling of heat exchangers, there are several discrete phenomena that can take place. The stream can be susceptible to phase transition moving into another phase region and reversal of the flow is something that has to be taken into consideration. Refrigeration cycles, which are used in a wide range of industrial processes such as liquefaction of natural gas, are examples of where phase transition should be modeled. There are two major challenges with including these phenomena in a heat exchanger. First of all, where and when a phase transition or a flow reversal takes place is not known beforehand. In addition, when a phase appears or disappears the topology of the heat exchangers changes. This results in a change in the number of equations needed to describe the system.

A model of a heat exchanger includes solving phase equilibrium between coexisting phases. These calculations are commonly referred to as flash calculations[3]. Flash calculations are of big importance as they are not only present in a mathematical model of a heat exchanger. Condensers, evaporators and distillation columns are other process units where flash calculations are included when developing dynamic models. Therefore, the flash calculations included in a dynamic model of a heat exchanger can be reused when developing a model for these other units.

1.1 Scope of Work

The main goal of this thesis is to develop a model of a heat exchanger which is valid for all the different phase regions (liquid-only, vapor-liquid, vapor-only). In addition, the model should allow reversal flow inside the heat exchanger. First, a model of single flash tank is to be developed. The reason for this is that the heat exchanger will be modeled as several flash tanks, with constant volume, in series. Secondly a single side of a heat exchanger with given heat transfer is to be developed by reusing the flash tank model. After this is accomplished, the model is to be extended to include both sides of a countercurrent heat exchanger which is the final objective.

As the main objective is to simulate the phase transition and handling changes in the equation set a specific heat exchanger is not considered nor is nonideal gases and liquids. There are several approaches that can be chosen to achieve the objective. In this thesis, it is desired to formulate nonsmooth DAEs of index 1, where the phase region detection is handled by a nonsmooth equation. This decision is taken as the author has experience using nonsmooth formulations to detect flow reversal in previous work done in a specialization project[4]. As there are nonsmooth functions in the model, the generalized derivatives are to be computed by automatic differentiation. This is done by implementing a MATLAB®-object using object-oriented programming (OOP) where the operators are overloaded.

1.2 Previous Work

In the last decades, solving phase equilibrium between coexisting phases has been discussed[3]. Such phase equilibrium is present in a heat exchanger. A well-established way of developing a heat exchanger model is to add the algebraic phase equilibrium equations to the differential equations of the conserved quantities in a differential algebraic equation (DAE) formulation. This was done for a multiphase heat exchanger already in 1989 by Pingaud[5]. However, this formulation is only valid in the vapor-liquid regime and not for the single phase regimes. During the last recent years there has been a development where the discrete phenomena of appearing and disappearing phases have been incorporated in the HEX models.

Kamath et al.[6] presented an equation-oriented framework using the information about the roots of a cubic equation of state to determine which phase region the system is in and to handle the phase transitions. The model is formulated as a mathematical program with complementarity constraints. To be valid in all the phase regions a relaxation variable, β, and two slack variables, s_V and s_L, are introduced. To find a solution of the model, an optimization problem has to be solved to determine these variables. The drawback with this formulation is that it is only a steady state model, as well as solving the optimization problem gets computational demanding with an increase in the number of variables.

Wilhelmsen et al.[3] presented a framework for solving flash calculations called the Thermodynamic Differential Algebraic Equation (TDAE) method which handles phase changes. They proposed using a hybrid model, using different sets of equations for the different

phases, and using an event detection routine to stop the integrator at different roots. The suggested roots are the following,

$$0 = \begin{cases} w(1-w) & \text{for changes from two-phase to single-phase} \\ T - T_{inc} & \text{for changes from single-phase to two-phase} \end{cases} \tag{1.1}$$

Here w is the vapor fraction, T is temperature and T_{inc} is the incipient temperature. The drawback of this formulation is that the integrator has to be stopped when the system moves into another phase region before the model is initialized with the end conditions using another set of equations. The issue with this is that the there is a limited theory on the existence, sensitivity, and uniqueness of their solutions[1].

Sahlodin et al.[1] presented a dynamic extension of a steady state formulation for handling phase changes proposed by Watson and Barton[7]. In their nonsmooth formulation, they introduce a modification, defining a continuous extension of the phase mole fraction into the regimes where they are normally not defined. The reason for introducing this extension is that the disappearance and appearance of a phase will not lead to discontinuities in the corresponding phase mole fractions. In the paper, it is described how the complementarity conditions, in the model proposed by Kamath et al.[6], can be reformulated as nonsmooth equations. In contrast to the hybrid formulation proposed by Wilhemlsen et al.[3], there are available theory on the sensitivity analysis, existence and uniqueness presented by Khan and Barton[8, 9].

1.3 Outline

Chapter 2 starts with the mathematical theory of nonsmooth analysis, covering lexicographic differentiation and how this can be included in automatic differentiation. The chapter ends with the theory of differential algebraic equations (DAEs), including different types and index of DAEs as well as how they can be solved. In Chapter 3 the thermodynamic model of the heat exchanger is presented. Chapter 4 covers the development of the dynamic model of a single flash tank, a single-sided HEX, and a countercurrent HEX. Following this, the simulation methods used in this thesis are introduced in Chapter 5. Then, Chapter 6 presents the results from dynamic simulations of the models. This chapter also discusses these results and their validity. Lastly, Chapter 7 concludes the thesis before recommendations for further work are given.

Chapter 2

Mathematical Preliminaries

In this chapter mathematical theory regarding the calculation of the *generalized derivatives* needed to solve the heat exchanger model, described in Chapter 4, is presented. This chapter is important as it explains how the derivatives of the nonsmooth functions are computed, which is essential to this thesis. The heat exchanger is modeled by using nonsmooth functions (in the form of *piecewise differentiable* functions), meaning that some equations in the model are not differentiable at certain points. For such functions, generalized derivative information is required as a replacement for the ordinary derivative of continuously differentiable functions. This information can be obtained from elements of the *Clarke Jacobian* or the *B-subdifferential*[10]. The problem with these subdifferentials is that they do not obey strict calculus rules, meaning that it is not possible to calculate them automatically[10]. During the last recent years, it has been proven that piecewise differentiable functions are *lexicographically smooth*, as well as that *lexicographic derivatives* of the same type of functions are a superset of the Clarke Jacobian[11]. These lexicographic derivatives are possible to compute through the *lexicographic-directional derivatives*, which obeys strict calculus rules. This development makes it possible to obtain an element of the Clarke Jacobian by *automatic differentiation*.

This chapter will define piecewise differentiable functions, *convex* functions, sets and hulls before the B-subdifferential and Clarke Jacobian are introduced. After this introduction, the lexicographic and lexicographic directional derivative will be defined. Further, the *automatic differentiation*, with the extension to the computation of lexicographic derivative of the absolute value function, will be covered. The part introducing nonsmooth analysis and automatic differentiation in this chapter is adapted from the specialization project written by the author in the autumn of 2017[4].

After the nonsmooth analysis is covered, the chapter will introduce differential algebraic equations (DAEs). This is relevant to the thesis as the heat exchanger is model as DAEs. First, DAE formulations are introduced, including different types of DAEs. This part will also define the index of a DAE, and an example of how specifications of different variables

in a system can affect the index will be given. Finally, a way of solving a Semi-Explicit index 1 DAE is presented.

2.1 Piecewise Differentiable ($\mathcal{PC}^1$) Functions and Convexity

In the heat exchanger model given in Chapter 4 the valve equations as well as the differential equations of the conserved quantities includes the $\mathcal{PC}^1$-functions $\min$ and $\max$. In addition, the equation which both handles and detects the phase changes uses the $\mathcal{PC}^1$-function mid. $\mathcal{PC}^1$-functions are defined in Definition 2.1.

Definition 2.1. *(From [10]) Consider an open set $X \subset \mathbb{R}^n$ and a function $\mathbf{f} : X \to \mathbb{R}^m$. As defined by Scholtes[12], $\mathbf{f}$ is* piecewise differentiable *($\mathcal{PC}^1$) at $\mathbf{x} \in X$ if there exists a neighborhood $N \subset X$ of $\mathbf{x}$ and a finite collection of continuous differentiable ($\mathcal{C}^1$) functions $\mathbf{f}_{(1)}$,...,$\mathbf{f}_{(q)}$: $\mathrm{N} \to \mathbb{R}^m$ such that $\mathbf{f}$ is continuous on N and such that*

$$\mathbf{f}(\mathbf{y}) \in \{\mathbf{f}_{(\mathbf{i})}(\mathbf{y}) \ : \ i \in \{1, ..., q\}\}, \qquad \forall\, \mathbf{y} \in \mathrm{N}. \tag{2.1}$$

If, in addition, each $\mathbf{f}_{(i)}$ is linear, then $\mathbf{f}$ is piecewise linear ($\mathcal{PL}$) at $\mathbf{x}$.

Example 2.1. *The function*

$$f(x) : \mathbb{R} \to \mathbb{R} : x \to \max\{0, x\}$$

consists of two linear (and thereby also $\mathcal{C}^1$) functions at x = 0:

$$f_{(1)}(x) = 0, \quad \forall\, x \in (-\infty, 0]$$
$$f_{(2)}(x) = x, \quad \forall\, x \in [0, \infty)$$

$f(x)$ is therefore $\mathcal{PL}$ (and thereby also $\mathcal{PC}^1$) on $\mathbb{R}$.

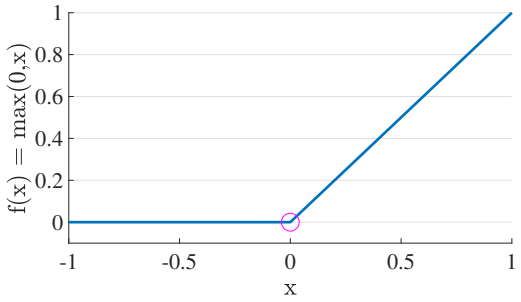


Figure 2.1: Graph of $\max\{0,x\}$. The circle marks the nonsmooth point.

To define the B-subdifferential and Clarke Jacobian, and express the relationship between them, the convex hull is needed. Before defining the convex hull, convex sets and functions are defined.

Definition 2.2. *(From [13]) A set $S \in \mathbb{R}^n$ is a* convex set*, if for any two point $\mathbf{x} \in S$ and $\mathbf{y} \in S$, the following relation holds,*

$$\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in S, \quad \forall\, \alpha \in [0,1]. \tag{2.2}$$

In other words, for a set to be convex, any straight line between two points within the set cannot cross the boundary of the set. In Figure 2.2 an example of both a convex and non-convex set is illustrated.

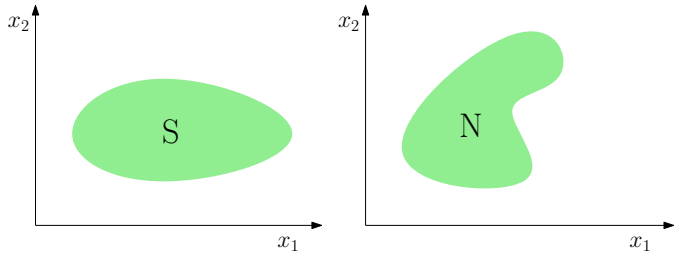


Figure 2.2: Illustration of a convex set, S, and a non-convex set N. The figure is adapted from Lund[14] and Vikse[15].

Definition 2.3. *(From [13]) The function $\mathbf{f}$ is a* convex function *if its domain S is a convex set and if for any to points $\mathbf{x} \in S$ and $\mathbf{y} \in S$, the following property is satisfied:*

$$\mathbf{f}(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha\mathbf{f}(\mathbf{x})(1-\alpha)\mathbf{f}(\mathbf{y}), \quad \forall\alpha \in [0,1]. \tag{2.3}$$

Definition 2.4. *(From [13]) A convex combination of a finite set of vectors $\{\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_m\}$ $\in \mathbb{R}^m$ is any vector $\mathbf{x}$ of the form*

$$\mathbf{x} = \sum_{i=1}^{m} \alpha_i x_i, \quad \textit{where} \sum_{i=1}^{m} \alpha_i = 1, \quad \textit{and } \alpha \geq 0\ \forall i = 1,2,...,m \tag{2.4}$$

The convex hull *of $\{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_m\}$ is the set of all convex combinations of these vectors.*

Definition 2.4 states that a convex hull of a non-convex set, is the smallest convex superset of S. This means that a convex hull of a set will always be larger than the original set unless the original set is convex, then the two will be equal. A non-convex set, S, and its corresponding convex hull is presented in Figure 2.3.

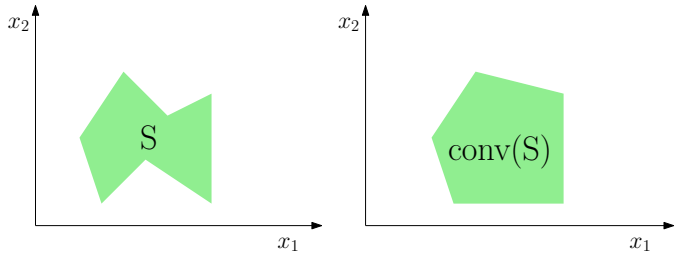


Figure 2.3: Illustration of a non-convex set (S) and its corresponding convex hull (conv(S)). The figure is adapted from Lund[14] and Vikse[15].

2.2 B-subdifferential & Clarke Generalized Jacobian

Set-valued generalized derivatives are as mentioned a replacement of the ordinary derivative of continuously differentiable functions for nonsmooth functions. However, both the B-subdifferential and the Clarke generalized Jacobian requires some continuity, namely *local Lipschitz continuity*.

Definition 2.5. *(From [13]) Given an open set $X \subset \mathbb{R}^n$ and a function $\mathbf{f}$: $X \to \mathbb{R}^m$. The function $\mathbf{f}$ is said to be* Lipschitz continuous *at $\mathbf{x} \in X$ if there exists a $L > 0$ such that.*

$$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|, \quad \forall \mathbf{x}, \mathbf{y} \in N. \tag{2.5}$$

Further, if property 2.5 only holds for a $\mathbf{x}$, $\mathbf{y}$ in a neighborhood $N \subset X$ of $\mathbf{x}$, the function is locally Lipschitz continuous.

Definition 2.5 states that the derivatives of a function needs to be bounded from above for it to be Lipschitz continuous. An example of a function that often appears in physical systems that is **not** Lipshitz continuous is $f(x) = \sqrt{x}$. This is because as $x \to 0$, $f'(x) \to \infty$. However, Lipschitz continuity can be established for $f(x) = \sqrt{|x|}$ by introducing a small value $0 < \epsilon \ll 1$, $f(x) = \sqrt{|x| + \epsilon}$. This property is used in the valve equation in the heat exchanger model introduced in Chapter 4.

Definition 2.6. *(From [10]) Given an open set $X \subset \mathbb{R}^n$ and a locally Lipschitz continuous function $\mathbf{f} : X \to \mathbb{R}^m$, let $S \subset X$ be the set on which $\mathbf{f}$ is differentiable. The* B-subdifferential *of $\mathbf{f}$ at $\mathbf{x} \in X$ is then*

$$\partial_B \mathbf{f}(\mathbf{x}) := \left\{ \mathbf{H} \in \mathbb{R}^{m \times n} : \mathbf{H} = \lim_{n \to \infty} \mathbf{J}\mathbf{f}(\mathbf{x}_{(\mathbf{i})}), \quad \mathbf{x} = \lim_{n \to \infty} \mathbf{x}_{(i)}, \quad \mathbf{x}_{(i)} \in S, \forall i \in \mathbb{N} \right\}. \tag{2.6}$$

The Clarke (generalized) Jacobian *of $\mathbf{f}$ at $\mathbf{x}$ is $\partial \mathbf{f}(\mathbf{x})$:= conv($\partial_B \mathbf{f}(\mathbf{x})$)*

In other words, the B-subdifferential is a set which consists of all Jacobians that arises when $\mathbf{x}$ is approached from every possible direction. The Clarke Jacobian is, from Def-

inition 2.6, a larger set than the B-subdifferential as it is the convex hull of it. However, both the B-subdifferential and the Clarke Jacobian reduces to the singleton of the Jacobian when $\mathbf{f}$ is continuously differentiable at $\mathbf{x}$.

Example 2.2. *(Adapted from Lund[14]) Let f(x) =* mid*(-x, x, 0.5), which is a* $\mathcal{PC}^1$*-function on* $\mathbb{R}$*, and therefore also Lipschitz continuous. The graph of f(x) is shown in Figure 2.4. The nonsmooth points are marked with a circle.*

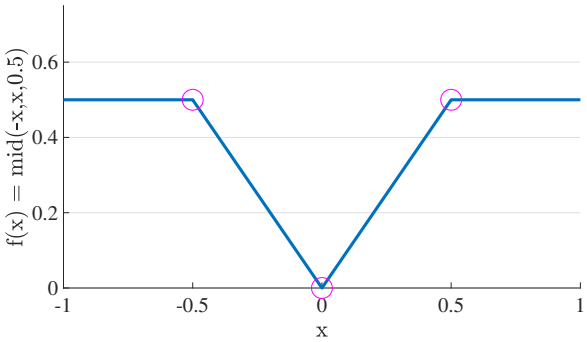


Figure 2.4: The figure shows the graph of f(x) = mid(-x, x, 0.5). The circles marks the nonsmooth points.

f(x) has three nondifferentiable points, x = -0.5, x = 0 and x = 0.5. In these points the Jacobian depends on from which direction the nondifferentiable point is approach. Consider the points x = $\{$*-1, -0.25, 0.25, 1*$\}$*, the B-subdifferentiable of f at these points are:*

$$\begin{aligned} \partial_B f(-1) &= 0 \\ \partial_B f(-0.25) &= -1 \\ \partial_B f(0.25) &= 1 \\ \partial_B f(1) &= 0 \end{aligned}$$

The B-subdifferentiable at these points is a single-valued set as the slope of f(x) does not change depending on from which direction the point is approached. In contrast, at the nondifferentiable, the Jacobian of f(x) changes depending on whether the function is approached from left or right. The B-subdifferential at these three points are as following:

$$\begin{aligned} \partial_B f(-0.5) &= \{0, -1\} \\ \partial_B f(0) &= \{-1, 1\} \\ \partial_B f(0.5) &= \{1, 0\}. \end{aligned}$$

The corresponding Clarke generalized Jacobian is:

$$\begin{aligned}\partial f(-0.5) &= [0,-1]\\ \partial f(0) &= [-1,1]\\ \partial f(0.5) &= [1,0].\end{aligned}$$

The shortcoming of B-subdifferentials is that they do not obey the general calculus rules, and there is no general approach on how to calculate them[14]. A disadvantage with the Clarke Jacobian is that it only satisfies some classical calculus rules as inclusion and not equality. One of these, the chain rule, has to be satisfied as equality to be used in automatic vector forward differentiation[10], which is used in this thesis. In calculus, the chain rule is the formula for calculating the derivative of the composition of two or more functions,

$$(\mathbf{f} \circ \mathbf{g})' = (\mathbf{f}' \circ \mathbf{g}) \cdot \mathbf{g}', \tag{2.7}$$

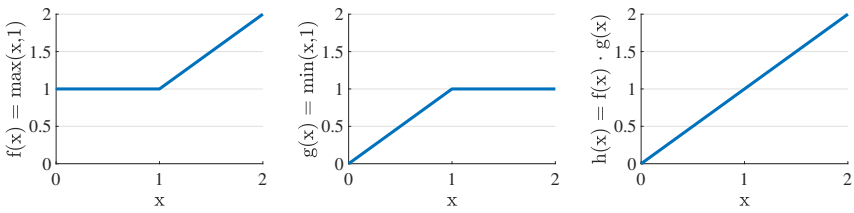
with $(\mathbf{f} \circ \mathbf{g})$ being $\mathbf{f}(\mathbf{g}(\mathbf{x}))$.

Example 2.3. *(Adapted from Lund[14]) Consider two functions, f(x) =* $\max\{x,1\}$ *and g(x) =* $\min\{x,1\}$*, that are both* $\mathcal{PC}^1$ *on* $\mathbb{R}$*. The functions are both nondifferentiable at x = 1, and has the following Clarke Jacobians at this point:*

$$\begin{aligned}\partial f(1) &= [0,1]\\ \partial g(1) &= [0,1]\end{aligned}$$

To show that the Clarke Jacobian only satisfies the chain rule by inclusion, consider the function h(x) = f(x) · g(x) = x, which is smooth and its Clarke Jacobian at x = 1 is $\partial h(1) = 1$*. This shows that the Clarke Jacobian only satisfies the chain rule by inclusions as*

$$\begin{aligned}\partial h(1) &\subset \partial(f(1)\cdot g(1))\\ &= \partial f(1)\cdot g(1) + \partial g(1)\cdot f(1)\\ &= [0,1]\cdot 1 + [0,1]\cdot 1\\ &= [0,2]\\ \partial h(1) &\neq \partial(f(1)\cdot g(1)).\end{aligned}$$



(a) f(x) = max(x,1) **(b)** g(x) = min(x,1) **(c)** h(x) = f(x) · g(x)

Figure 2.5: The graphs of f(x) = max(x,1), g(x) = min(x,1) and h(x) = f(x) · g(x)

Due to the lack of strict calculus rules for the Clarke Jacobian, another type of generalized derivatives has to be used. Lexicographic derivatives can be obtained through the calculation of Lexicographic directional derivatives (LD-derivatives), which follow strict calculus rules. It has been proved, by Khan and Barton[11], that the Lexicographic derivative are elements of the plenary hull of the Clarke Jacobian, meaning that such elements are equally useful as elements from the Clarke Jacobian.

Definition 2.7. *(From [16]) A set* $A \subset \mathbb{R}^{m\times n}$ *is* plenary *if it includes every* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *satisfying* $\mathbf{Au} \in A\mathbf{u}$ *for all* $\mathbf{u} \in \mathbb{R}^n$*. Further, the* plenary hull*, denoted* plen *A, is the smallest plenary set containing A.*

Definition 2.8. *(From [10]) The* plenary Jacobian $\partial_P \mathbf{f}(\mathbf{x})$ *is the plenary hull of the Clarke Jacobian, and satisfies:*

$$\partial_P \mathbf{f}(\mathbf{x}) = \{\mathbf{M} \in \mathbb{R}^{m\times n} : \forall \mathbf{v} \in \mathbb{R}^n, \quad \exists \mathbf{H} \in \partial \mathbf{f}(\mathbf{x}) \quad \text{s.t} \mathbf{Mv} = \mathbf{Hv}\} \supseteq \partial \mathbf{f}(\mathbf{x})$$

This means that the lexicographic derivatives can be used in nonsmooth Newton-type solvers on the form,

$$\mathbf{G}(\mathbf{x}^k)\left(\mathbf{x}^{(k+1)} - \mathbf{x}^k\right) = -\mathbf{f}(\mathbf{x}^k), \tag{2.8}$$

where G is a generalized derivative element [17].

2.3 Lexicographic Derivatives

The lexicographic derivatives were first proposed and developed by Nesterov[18], and, as mentioned earlier, proved to be a part of the plenary hull of the Clarke generalized Jacobian by Khan and Barton[11]. For a function to have a well defined lexicographic derivative at a point $\mathbf{x}$, it must be lexicographically smooth (L-smooth) at $\mathbf{x}$. The lexicographic derivative is calculated through the lexicographic directional derivatives (LD-derivatives) as they obey strict calculus rules, meaning that it is possible to compute them automatically. These LD-derivatives are the lexicographic analogues to the directional derivative of continuous

differentiable functions ($\mathcal{C}$).

Definition 2.9. *(From [13]) The* directional derivative *of a function* $\mathbf{f}(\mathbf{x})$*:* $\mathbb{R}^n \to \mathbb{R}^m$ *in the direction* $\mathbf{d} \in \mathbb{R}^n$ *is given by,*

$$\mathbf{f}'(\mathbf{x}) \equiv \lim_{h \to 0} \frac{\mathbf{f}(\mathbf{x} + h\mathbf{d}) - \mathbf{f}(\mathbf{x})}{h} \tag{2.9}$$

Definition 2.10. *(From [10]) Given an open set* $X \subset \mathbb{R}^n$ *and a locally continuous function* $\mathbf{f} : X \to \mathbb{R}^m$*,* $\mathbf{f}$ *is* lexicographically smooth *at* $\mathbf{x} \in X$ *if it is directionally differentiable at* $\mathbf{x}$ *and if, for any* $k \in \mathbb{N}$ *and* $\mathbf{M} \in \mathbb{R}^{n \times p}$*, the following functions are well-defined:*

$$\begin{aligned}
&\mathbf{f}^{(0)}_{\mathbf{x},\mathbf{M}} : \mathbb{R}^n \to \mathbb{R}^m : \mathbf{d} \mapsto \mathbf{f}'(\mathbf{x}; \mathbf{d}),\\
&\mathbf{f}^{(1)}_{\mathbf{x},\mathbf{M}} : \mathbb{R}^n \to \mathbb{R}^m : \mathbf{d} \mapsto [\mathbf{f}^{(0)}_{\mathbf{x},\mathbf{M}}]'(\mathbf{m}_{(1)}; \mathbf{d}),\\
&\mathbf{f}^{(2)}_{\mathbf{x},\mathbf{M}} : \mathbb{R}^n \to \mathbb{R}^m : \mathbf{d} \mapsto [\mathbf{f}^{(1)}_{\mathbf{x},\mathbf{M}}]'(\mathbf{m}_{(2)}; \mathbf{d}),\\
&\quad \vdots\\
&\mathbf{f}^{(k)}_{\mathbf{x},\mathbf{M}} : \mathbb{R}^n \to \mathbb{R}^m : \mathbf{d} \mapsto [\mathbf{f}^{(k-1)}_{\mathbf{x},\mathbf{M}}]'(\mathbf{m}_{(k)}; \mathbf{d}).
\end{aligned}$$

There are many types of functions that are L-smooth. Some of these groups of functions are $\mathcal{C}^1$ functions, convex functions and, most importantly for this project work, $\mathcal{PC}^1$ functions[10].

Definition 2.11. *(From [10]) Given the function* $\mathbf{f} : X \to \mathbb{R}^m$*, with* $X \subset \mathbb{R}^n$ *and* $\mathbf{f}$ *lexicographically smooth at* $\mathbf{x}$*. The* lexicographic derivative *of* $\mathbf{f}$ *at* $\mathbf{x}$ *is defined as*

$$\mathbf{J}_L\mathbf{f}(\mathbf{x}; \mathbf{M}) \equiv \mathbf{J}\mathbf{f}^{(n)}_{\mathbf{x},\mathbf{M}}(0) \tag{2.10}$$

for any nonsingular $\mathbf{M} \in \mathbb{R}^{nxn}$*. Further, the* lexicographic subdifferential *of* $\mathbf{f}$ *at* $\mathbf{x}$ *is given by*

$$\partial_L\mathbf{f}(\mathbf{x}) = \{\mathbf{J}_L\mathbf{f}(\mathbf{x}; \mathbf{M}) : \mathbf{M} \in \mathbb{R}^{nxn}, \quad \det\mathbf{M} \neq 0\} \tag{2.11}$$

Definition 2.12. *(From [10]) Given an open set* $X \subset \mathbb{R}^n$*, a locally Lipschitz continuous function* $\mathbf{f}$*:* $X \to \mathbb{R}^m$ *that is lexicographically smooth at* $\mathbf{x} \in X$*, and a matrix* $\mathbf{M} \equiv [\mathbf{m}_{(1)} \ \ldots \ \mathbf{m}_{(k)}] \in \mathbb{R}^{n \times k}$*, the* LD-derivative *of* $\mathbf{f}$ *at* $\mathbf{x}$ *in the directions* $\mathbf{M}$ *is*

$$\mathbf{f}'(\mathbf{x}; \mathbf{M}) \equiv \left[\mathbf{f}^{(0)}_{\mathbf{x},\mathbf{M}}(\mathbf{m}_{(1)})\mathbf{f}^{(1)}_{\mathbf{x},\mathbf{M}}(\mathbf{m}_{(2)}) \ldots \mathbf{f}^{(k-1)}_{\mathbf{x},\mathbf{M}}(\mathbf{m}_{(k)})\right] \tag{2.12}$$

$$= \left[\mathbf{f}^{(k)}_{\mathbf{x},\mathbf{M}}(\mathbf{m}_{(1)}) \ldots \mathbf{f}^{(k)}_{\mathbf{x},\mathbf{M}}(\mathbf{m}_{(k)})\right] \tag{2.13}$$

Example 2.4. *(Adapted from Lund[14]) Let $f(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}$ be the $\mathcal{PC}^1$ function: f(x_1,x_2) = $\min\{x_1,x_2\}$. The function is not differentiable at any point where $x_1 = x_2$. However the function is $\mathcal{PC}^1$, meaning that it is L-smooth, and therefore, the LD-derivative is well defined. In this example the lexicographic derivative of f is computed using Definition 2.10 at (x_1,x_2) = (0,0).*

Given the direction matrix

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.14}$$

which is square and nonsingular. The first order directional derivative is computed as defined in Definition 2.10:

$$\begin{aligned} f^{(0)}_{\mathbf{0},\mathbf{M}}(\mathbf{d}) &= f'(\mathbf{0};\mathbf{d}) \\ &= \lim_{h\to 0} \frac{f(x_1 + hd_1, x_2 + hd_2) - f(x_1, x_2)}{h} \\ &= \lim_{h\to 0} \frac{\min\{x_1 + hd_1, x_2 + hd_2\} - \min\{x_1, x_2\}}{h} \\ &= \lim_{h\to 0} \frac{\min\{0 + hd_1, 0 + hd_2\} - \min\{0, 0\}}{h} \\ &= \lim_{h\to 0} \frac{h\min\{d_1, d_2\}}{h} \\ &= \min\{d_1, d_2\}. \end{aligned}$$

The second order directional derivative is further calculated by using Definition 2.10:

$$\begin{aligned} f^{(\mathbf{1})}_{\mathbf{x},\mathbf{M}}(\mathbf{d}) &= \left[f^{(0)}_{\mathbf{0},\mathbf{M}}\right]' \left(\mathbf{m}_{(1)};\mathbf{d}\right) \\ &= \lim_{h\to 0} \frac{f^0_{\mathbf{x},\mathbf{M}}\left(m_{11} + hd_1, m_{21} + hd_2\right) - f^0_{\mathbf{x},\mathbf{M}}\left(m_{11}, m_{21}\right)}{h} \\ &= \lim_{h\to 0} \frac{\min\{m_{11} + hd_1, m_{21} + hd_2\} - \min\{m_{11}, m_{21}\}}{h} \\ &= \lim_{h\to 0} \frac{\min\{1 + hd_1, 0 + hd_2\} - \min\{1, 0\}}{h} \\ &= \lim_{h\to 0} \frac{hd_2}{h} \\ &= d_2. \end{aligned}$$

The third order directional derivative is calculated in the same way. Equation

$$
\begin{aligned}
f^{(\mathbf{2})}_{\mathbf{x},\mathbf{M}}(\mathbf{d}) &= \left[f^{(1)}_{\mathbf{0},\mathbf{M}}\right]'(\mathbf{m}_{(2)};\mathbf{d}) \\
&= \lim_{h\to 0} \frac{f^1_{\mathbf{x},\mathbf{M}}(m_{12}+hd_1, m_{22}+hd_2) - f^1_{\mathbf{x},\mathbf{M}}(m_{12}, m_{22})}{h} \\
&= \lim_{h\to 0} \frac{m_{22}+hd_2 - m_{22}}{h} \\
&= d_2.
\end{aligned}
$$

Using Definition 2.12, the LD-derivative can be calculated in the two presented alternatives. Equation 2.12 gives

$$
f'(\mathbf{0};\mathbf{M}) = \begin{bmatrix} f^0_{\mathbf{0},\mathbf{M}}(\mathbf{m}_{(1)}) & f^1_{\mathbf{0},\mathbf{M}}(\mathbf{m}_{(2)}) \end{bmatrix} = \begin{bmatrix} \min\{m_{11}, m_{21}\} & m_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix}.
$$

Alternative 2, Equation 2.13 gives the same generalized derivative:

$$
f'(\mathbf{0};\mathbf{M}) = \begin{bmatrix} f^2_{\mathbf{0},\mathbf{M}}(\mathbf{m}_{(1)}) & f^2_{\mathbf{0},\mathbf{M}}(\mathbf{m}_{(2)}) \end{bmatrix} = \begin{bmatrix} m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix}.
$$

The graph of f(x,y) = $\min\{x,y\}$ *is presented in Figure 2.6.*

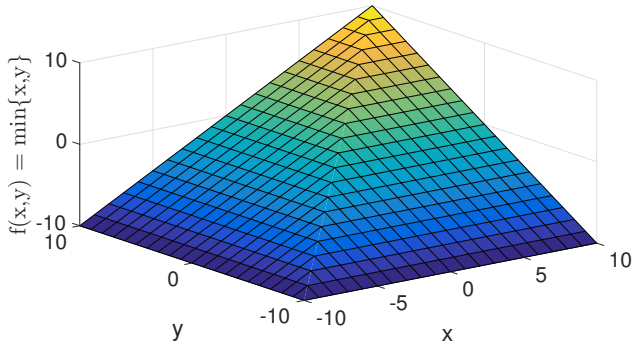


Figure 2.6: Graph of f(x,y) = $\min\{x,y\}$

2.4 Automatic Differentiation

In this section, an introduction to automatic differentiation (AD) of smooth and L-smooth functions, with additional focus on $\mathcal{PC}^1$ functions, will be given. The AD of $\mathcal{PC}^1$ uses the theory of generalized derivatives presented in the previous sections, as well as further work done by Khan and Burton[10].

Automatic differentiation is an alternative to calculating the derivatives through symbolic or numerical differentiation. There are two different kinds of AD, forward automatic differentiation and reverse automatic differentiation. In this thesis, only the forward mode will be introduced as it is how the AD used in this project is implemented. The thought of AD is that strict calculus rules such as the chain rule, Equation 2.7, can be implemented in a numerical environment[19]. This is done by exploiting the procedural operations that a computer performs when evaluating functions. When a computer evaluates a function, elemental operations are executed in a sequence. After each execution, a temporary variable is returned which is then used in the next operation. The elemental operations referred to are simple operations such as subtraction, addition, multiplication, division as well as simple functions (trigonometric functions, exponential function etc.). Similar to calculating the function value, the derivative of a function is possible to calculate by using this stepwise evaluation. In this thesis, this has been implemented using object-oriented programming (OOP) in MATLAB®. The objects have both a value and a derivative. By overwriting the elemental functions, both the value and derivative is updated after each elemental operation. The AD-class *valder.m* is presented in Appendix D.1. The forward AD is illustrated in Table 2.1.

Table 2.1: Forward AD of the function f(x,y) = $x^2y + y\sin(x)$ at (x,y) = (1,2)

Function Value		**Derivate expression**	**Derivative**
$u_1 = x$	$= 1$	$\mathbf{J}u_1$	$= [1\ 0]$
$u_2 = y$	$= 2$	$\mathbf{J}u_2$	$= [0\ 1]$
$u_3 = u_1^2$	$= 1$	$\mathbf{J}u_3 = 2u_1\mathbf{J}u_1$	$= [2\ 0]$
$u_4 = u_3 \cdot u_2$	$= 2$	$\mathbf{J}u_4 = u_2 \cdot \mathbf{J}u_3 + u_3 \cdot \mathbf{J}u_2$	$= [4\ 1]$
$u_5 = \sin(u_1)$	≈ 0.8415	$\mathbf{J}u_5 = \cos(u_5) \cdot \mathbf{J}u_1$	$\approx [0.5403\ 0\ ]$
$u_6 = u_2 \cdot u_5$	≈ 1.6829	$\mathbf{J}u_6 = u_5 \cdot \mathbf{J}u_2 + u_2 \cdot \mathbf{J}u_5$	$\approx [1.0806\ 0.8415]$
$u_7 = u_4 + u_6$	≈ 3.6829	$\mathbf{J}u_7 = \mathbf{J}u_4 + \mathbf{J}u_6$	$\approx [5.0806\ 1.8415]$

2.4.1 AD of $\mathcal{PC}^1$-functions

The example of AD presented in Table 2.1 was done using a $\mathcal{C}^1$ functions. It is also possible to apply the same principle to a nonsmooth function, given that they are L-factorable.

Definition 2.13. *(From [10]) A factorable function* $\mathbf{f}$ *is* **L**-factorable *if the elemental library* $\mathcal{L}$ *contains only lexicographically smooth functions whose LD-derivatives are known or computable.*

In Definition 2.13 the library $\mathcal{L}$ refers to the elemental functions that the function $\mathbf{f}$ can be broken down to. Important to this project, $\mathcal{PC}^1$-functions are L-factorable. In this project, only the abs-function is differentiated by exploiting the strict calculus rules of the lexicographic directional derivatives. Also the $\mathcal{PC}^1$-functions $\min$, $\max$ and mid are used in the model formulation presented in Chapter 4, but these are expressed as a function of the absolute function in the AD-class. This means that it is actually the abs-function that is evaluated when the $\min$, $\max$ and mid functions are called. $\min\{x, y\}$ and $\max\{x, y\}$

can be expressed as a function of abs. $\mathrm{mid}\{x, y, z\}$ can be expressed by the use of $\min$ and $\max$, making it a function of the absolute value function,

$$\min\{x, y\} = \frac{x + y - |x - y|}{2}, \tag{2.15}$$

$$\max\{x, y\} = \frac{x + y + |x - y|}{2}, \tag{2.16}$$

$$\mathrm{mid}\{x, y, z\} = \min\{\max\{x, y\}, \max\{x, z\}, \max\{y, z\}\}. \tag{2.17}$$

Algorithm 1 Computes the LD-derivative, $\mathbf{u}'(\mathbf{x}; \mathbf{M})$ for the absolute value function u = $|x|$ [10]

Require: Function f : $\mathbb{R} \to \mathbb{R} : |x|$ that admits a scalar argument.

```
if x ≠ 0 then
    Set V̇ ← (sign x)M
else
    Set s_1 ← 1
    for k = 1 to p do
        if m_(k) ≠ 0 then
            Set s_1 ← sign m_k
            Break out of for-loop
        end if
    end for
    V̇ ← s_1 M
end if
return V̇
```

In Algorithm 1 the procedure of calculating the LD-derivative of the absolute function. If x is not equal to 0, the procedure reduces to returning the well-defined derivative of $\mathrm{abs}(x)$. With x = 0, the LD-derivative is determined by the sign of the first non-zero direction.

2.5 Differential-Algebraic Equations (DAEs)

Many dynamic process models consist of differential- and algebraic equations (DAEs). This is also the case for the heat exchanger model presented in this thesis. In this section, an introduction to DAEs and the solution of such equation systems is given. First the theory for smooth DAE systems will be covered. After, some theory for nonsmooth DAE systems will be introduced.

2.5.1 DAE Formulations

The most general type of differential-algebraic equations is the fully-implicit nonlinear DAE[20]:

$$\mathbf{f}(\mathbf{z}', \mathbf{z}, t) = 0 \qquad \mathbf{z}(t_0) = \mathbf{z_0}. \tag{2.18}$$

Here $\mathbf{f}$ is a vector of nonlinear functions, $\mathbf{z}$ is the vector of unknown variables, $\mathbf{z}'$ is the time derivative of $\mathbf{z}$, and t is time. For a system with n differential- and m algebraic variables the dimensions are as follow:

$$\mathbf{f} = [f_1 \dots f_n \dots f_{n+m}]^T \;\; \mathbf{z} = [z_1 \dots z_n \dots z_{n+m}]^T \;\; \mathbf{z}' = [z_1' \dots z_n' \dots 0 \dots 0]^T \tag{2.19}$$

All types of DAEs can be represented by Equation 2.18, but less theory exists for this formulation compared to other DAE formulations[20]. Within process modeling the models are mostly formulated as semi-explicit DAEs which are on the following form:

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{y}, t), \quad \mathbf{x}(t_0) = \mathbf{x_0}, \tag{2.20}$$

$$0 = \mathbf{g}(\mathbf{x}, \mathbf{y}, t). \tag{2.21}$$

Here the $\mathbf{z}$-vector has been divided into two vectors $\mathbf{x}$ and $\mathbf{y}$, where $\mathbf{x}$ and $\mathbf{y}$ is the differential- and algebraic variables respectively. $\mathbf{f}(\mathbf{x}, \mathbf{y}, t)$ is the differential equations, while $\mathbf{g}(\mathbf{x}, \mathbf{y}, t)$ is the algebraic equations. This formulation of DAEs is called semi-explicit as it is not possible to express $\mathbf{y}$ explicitly from the algebraic equations. In a case where this is possible ($\mathbf{y} = \mathbf{g}(\mathbf{x}, t)$) the DAE formulation in Equation 2.20 is reduced into a set of ordinary differential equations[20],

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}, t) \qquad \mathbf{x}(t_0) = x_0 \tag{2.22}$$

2.5.2 Index of DAEs

For a steady state model to be numerically well-behaved it must **be nonsingular** and **have small condition numbers**[20]. For a system of equations to be nonsingular, it must have full rank, meaning that the number of variables and linearly independent equations is equal. Further, the condition (κ) number is defined as,

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \left\|\mathbf{A^{-1}}\right\| \tag{2.23}$$

If $\kappa(\mathbf{A}) \gg 1$ the matrix is ill-conditioned[21]. To find the condition number of a model the $\mathbf{A}$-matrix in Equation 2.23 is substituted with the Jacobian or, in the nonsmooth case,

the generalized derivative matrix. Solving a system of equation on the form $\mathbf{Ax} = \mathbf{b}$, the numerical solution will not be exact. This is due to the rounding errors during the calculation. The numerical solution can be written as $\mathbf{x} + \delta\mathbf{x}$, and normally this will be the solution of equation $\mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$. Here the elements of $\delta\mathbf{b}$ is very small. In the case where $\kappa(\mathbf{A})$ is large this might not be the case for $\delta\mathbf{x}$, leading to that the solution of the model is highly affected by the numerical rounding errors[21].

For a dynamic model, nonsingularity and small conditions numbers do not guarantee that it is numerically well-behaved. In addition, the DAE must have a **low index**. *The index is a measure of the problems related to initialization and integration of dynamic process models*[20]. Dynamic models are divided into *low index* (0 or 1) and *high index* (2 or larger). Most commercial solvers, such as *ode15s* in MATLAB®, only handle low index problems. Because of this, it is advantageous to develop a low index model whenever possible. The index of a model can be found by investigating how many differentiations have to be done to express the algebraic variables as a function of the specified variables[20]. Which variables that are specified can affect the index of the model. An example of how the index of a model is calculated, and how a specification of a variable can affect the index of a model is given in Example 2.5.

Example 2.5. *(This example is based on an example in [20]). Consider a ideal gas system with one inlet and outlet stream where heat is transferred to the system. Assume a reference temperature T_{ref} = 0 K and constant heat capacity. The inlet flows F_0, inlet enthalpy h_0 and heat transfer Q are specified and are functions of time, t.*

$$\frac{dM}{dt} = F_0(t) - F_1(t) \tag{2.24}$$

$$\frac{dU}{dt} = Q(t) + F_0(t)h_0(t) - F_1(t)h_1 \tag{2.25}$$

$$pV = MRT \tag{2.26}$$

$$U = MC_vT \tag{2.27}$$

$$h_1 = C_pT \tag{2.28}$$

M is the number of moles, h_1 is enthalpy in the outlet flow, p is pressure, T is temperature and U is internal energy. In the model, there are balance equations for M and U, and algebraic equations for pressure, temperature, and enthalpy. The given model is of index 1 and is semi-explicit. That the model is of index 1 can be shown by differentiating the algebraic equations,

$$p'V = M'RT + MRT' \tag{2.29}$$

$$U' = M'C_vT + MC_vT' \tag{2.30}$$

$$h_1' = C_pT' \tag{2.31}$$

By solving Equation 2.29 for p', 2.30 for T' and 2.31 forh_1' and inserting Equation 2.24 and 2.25, the following ordinary differential equations are obtained,

$$\frac{dp}{dt} = \frac{R}{V}\left((F_0(t) - F_1(t))(T-1) + \frac{Q(t) + F_0(t)h_0(t) - F_1(t)h_1}{C_v}\right), \tag{2.32}$$

$$\frac{dT}{dt} = \frac{Q(t) + F_0(t)h_0(t) - F_1(t)h_1 - F_0 + F_1}{MC_v}, \tag{2.33}$$

$$\frac{h_1}{dt} = \frac{C_p}{M}\left(\frac{Q(t) + F_0(t)h_0(t) - F_1(t)h_1}{C_v} - F_0 + F_1\right). \tag{2.34}$$

As only one differentiation of the algebraic is needed to convert the DAE into a set of ordinary differential equations the DAE is of index 1.

Now consider the same system but instead of specifying Q, the temperature T is specified. The specification of the temperature will impose constraints on the two differential variables M and U and will lead to a higher index DAE. This can be shown in the same manner as for the index 1 model. Differentiating the algebraic equations,

$$p'V = M'RT, \tag{2.35}$$

$$U' = M'C_vT, \tag{2.36}$$

$$h_1' = 0. \tag{2.37}$$

Inserting Equation 2.25 and 2.24 into Equation 2.36 the following equation is obtained,

$$Q = [C_vT - h_0(t)]\,F_0(t) + [h_1 - C_vT]\,F_1(t). \tag{2.38}$$

To express Q' as a function of the other variables Equation 2.38 has to be differentiated. This means that two differentiations are needed to obtain a system of ODEs when T is constant. The index of the system is therefore 2. The same will be the case if the pressure, p, is specified.

A semi-explicit DAE is of index 1 if the following holds,

$$\det\left(\frac{\partial \mathbf{g}}{\partial \mathbf{y}}\right) \neq 0 \tag{2.39}$$

.

To show this the system in Example 2.5 can be used. First the algebraic variable set $\mathbf{y}$ = [p, M, T] is considered. As shown in the example, using this variable set, the DAE is of index 1. The Jacobian of the algebraic function $\mathbf{g}$ is given as,

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}} = \begin{bmatrix} V & -RT & -MR \\ 0 & -C_v T & -MC_v \\ 0 & 0 & -C_p \end{bmatrix}, \tag{2.40}$$

and the determinant is,

$$\det\left(\frac{\partial \mathbf{g}}{\partial \mathbf{y}}\right) = \det\left(\begin{bmatrix} V & -RT & -MR \\ 0 & -C_v T & -MC_v \\ 0 & 0 & -C_p \end{bmatrix}\right) = VTC_V C_p \neq 0, \tag{2.41}$$

which is true as $V > 0$, $T > 0$, $C_V > 0$, and $C_p > 0$.

Now considering the other algebraic variable set, $\mathbf{y}$ = [p, M, Q]. The Jacobian is given as,

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}} = \begin{bmatrix} V & -RT & -MR \\ 0 & -C_v T & -MC_v \\ 0 & 0 & 0 \end{bmatrix}, \tag{2.42}$$

and the determinant is,

$$\det\left(\frac{\partial \mathbf{g}}{\partial \mathbf{y}}\right) = \det\left(\begin{bmatrix} V & -RT & -MR \\ 0 & -C_v T & -MC_v \\ 0 & 0 & 0 \end{bmatrix}\right) = 0. \tag{2.43}$$

Thereby, the index of the DAE system is not of index 1 if the temperature is specified.

2.5.3 Nonsmooth DAEs

The mathematical theory behind nonsmooth DAEs has not been investigated thoroughly in this work as it is considered to be out of scope. However, a brief introduction on how it is possible to verify that a nonsmooth DAE system has a generalized differentiation index of 1 is presented. The theory in this section assumes that the nonsmooth functions are $\mathcal{PC}^1$. During the last recent years there has been a development on computationally relevant theory of nonsmooth DAEs. Stechlinski, Patrascu and Barton have established theory on the well-posedness and sensitivity analysis of nonsmooth DAEs[22, 23]. A nonsmooth DAE system has generalized differentiation index of 1 if the projection of the Clarke Jacobian of the algebraic equation $\mathbf{g}$ has full rank for all $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$ at all times t. This projection is the nonsmooth analogue of $\partial \mathbf{g} / \partial \mathbf{y}$, in the smooth case[23]. That the system is of index 1 means that it admits a unique regular solution[23]. How to verify this index will be illustrated by using an example from Stechlinski et al.(2017)[23]. This example is based on simple flash calculations and includes the mid-function which is also present in the heat exchanger model presented in Chapter 4.

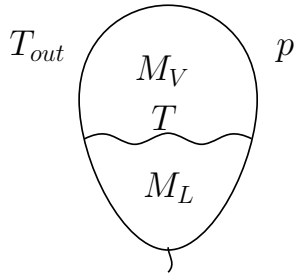


Figure 2.7: Sketch of a balloon with one species distributed between a liquid and a vapor phase. The content exchange heat with the environment. The balloon can either be in the vapor-only, vapor-liquid or liquid-only phase region.

Example 2.6. *Consider a balloon, illustrated in Figure 2.7, where a single species is held at a constant pressure. The balloon system can be described by a simple dynamic model formulated as a nonsmooth DAE,*

$$\frac{dH}{dt} = UA(T_{out} - T(t)), \tag{2.44}$$

$$H(t) = MC_p(T(t) - T_0) - M_L(t)\Delta_{vap}h, \tag{2.45}$$

$$0 = \text{mid}\left(1 - \frac{M_L(t)}{M}, 1 - \frac{aT(t) + b}{p}, \frac{-M_L(t)}{M}\right), \tag{2.46}$$

$$T(0) = T_0. \tag{2.47}$$

Here UA is the overall heat transfer coefficient times the surface area, H is enthalpy, T_{out} is the outside temperature, T is the temperature inside the balloon, M is the total molar holdup, M_L is the liquid molar holdup, C_p is the heat capacity, $\Delta_{vap}h$ is the heat of vaporization, and the expression aT(t) + b is the saturation pressure, p^{sat}. The nonsmoothness of the DAE arises in the mid*-function. The three terms are active in different phase regimes. The first term is active in the liquid phase only ($M_L = M$), the second term is active in the two-phase regime ($p^{sat} = p$), while the third term is active in the vapor-only ($M_L = 0$). The terms are formulated such that one of the nonactive is non-negative, while the other nonactive term is non-positive. Each of the three terms are $\mathcal{C}$-functions. To verify that the system is of generalized differentiation index 1, the projection of the Clarke Jacobian is calculated. The function* g *can be divided into three selection functions. At most two of these can be active as the first and third term in the* mid*-function cannot be equal,*

$$\mathbf{g}_1 = \begin{bmatrix} -H(t) + MC_p(T(t) - T_0) - M_L(t)\Delta_{vap}h \\ 1 - {}^{M_L}\!/_{M} \end{bmatrix} \tag{2.48}$$

$$\mathbf{g}_2 = \begin{bmatrix} -H(t) + MC_p(T(t) - T_0) - M_L(t)\Delta_{vap}h \\ 1 - \frac{aT(t)+b}{p} \end{bmatrix} \tag{2.49}$$

$$\mathbf{g}_3 = \begin{bmatrix} -H(t) + MC_p(T(t) - T_0) - M_L(t)\Delta_{vap}h \\ -{}^{M_L(t)}\!/_{M} \end{bmatrix} \tag{2.50}$$

As $\mathbf{Jg}_1 = \mathbf{Jg}_3$ *for any points in the domain of the function, the Clarke Jacobian of* $\mathbf{g}$ *at any point is given by,*

$$\partial\mathbf{g}(H, T, M_L) = \left\{ \begin{bmatrix} -1 & MC_p & -\Delta_{vap}h \\ 0 & \frac{(\lambda-1)a}{p} & -{}^{\lambda}\!/_{M} \end{bmatrix} : \lambda \in [0, 1] \right\} \tag{2.51}$$

If the determinant of the projection of the Clarke Jacobian of $\mathbf{g}, \partial\mathbf{g}$*, with respect to* $\mathbf{y}$*,* $\pi_{\mathbf{y}}\partial g(H, T, M_L)$ *is non zero the nonsmooth DAE system has index 1.* $\pi_{\mathbf{y}}\partial\mathbf{g}(H, T, M_L)$ *is given as,*

$$\pi_{\mathbf{y}}\partial g(H, T, M_L) = \left\{ \begin{bmatrix} MC_p & -\Delta_{vap}h \\ \frac{(\lambda-1)a}{p} & -{}^{\lambda}\!/_{M} \end{bmatrix} : \lambda \in [0, 1] \right\}, \tag{2.52}$$

which is the nonsmooth analogue of ${}^{\partial\mathbf{g}}\!/_{\partial\mathbf{y}}$*, in the smooth case. The determinant of* $\pi_{\mathbf{y}}\partial\mathbf{g}(H, T, M_L)$ *is,*

$$\det\left(\begin{bmatrix} MC_p & -\Delta_{vap}H \\ \frac{(\lambda-1)a}{p} & -{}^{\lambda}\!/_{M} \end{bmatrix} \right) = -\lambda C_p + (\lambda - 1)\frac{a\Delta_{vap}h}{p} \neq 0, \, \forall\lambda \in [0, 1], \tag{2.53}$$

as $C_p > 0$ *and* $a\Delta_{vap}h/p > 0$*.*

For a nonsmooth DAE system where the nonsmooth functions are $\mathcal{PC}^1$, and each of the selection functions of $\mathbf{g}$ is of index 1, than the index of the nonsmooth DAE will be 1.

2.6 Solving Semi-Explicit Index 1 DAE

Given a semi-explicit index 1 DAE system, on the form as presented in Equation 2.20, it can be integrated using an explicit integration code. The algorithm for doing such an integration is given in Algorithm 2[20].

Algorithm 2 Explicit Euler integration algorithm for a semi-explicit index 1 DAE system.[20]

1: Specify initial values at t_0 : $\mathbf{x_0} = \mathbf{x}(t_0)$
2: **while** t =< t_{end} **do**
3: Calculate algebraic variables, g($\mathbf{x_n}, \mathbf{y_n}, t_n$) = 0: $\mathbf{y_n} = \mathbf{G}(\mathbf{x_n}, t_n)$
4: Calculate $\mathbf{x}_{n+1}$ from $\mathbf{x_n}$ and $\mathbf{y_n}$: $\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta t \cdot \mathbf{f}(\mathbf{x}_n, \mathbf{y}_n, t_n)$
5: **end while**

In this algorithm, a Euler integration is used, but it is also possible to use other explicit integration algorithms, such as Runge Kutta methods[20].

For stiff models, the explicit integration method must be replaced by an implicit one[20]. An implicit integration requires the algebraic equations to be solved repeated times. This will make the algorithm more computational expensive but is necessary if the model is stiff. The implicit algorithm is given below.

Algorithm 3 Implicit Euler integration algorithm for a semi-explicit index 1 DAE system.[20]

1: Specify initial values at t_0 : $\mathbf{x_0} = \mathbf{x}(t_0)$
2: **while** t =< t_{end} **do**
3: **while** error $> \epsilon$ **do**
4: error = $\mathbf{y}_{n+1}$ - $\mathbf{y}_n$ + $\Delta t \cdot \mathbf{f}\,(\mathbf{y}_{n+1}, \mathbf{x}_{n+1}, t_{n+1})$
5: $\mathbf{g}(\mathbf{y}_{n+1}, \mathbf{x}_{n+1}, t_{n+1}) = 0$
6: **end while**
7: **end while**

Chapter 3

Thermodynamic Theory

In this chapter, the thermodynamic model used in this thesis is presented. It covers the derivation of the thermodynamic equations used in the HEX model. First, the definition of thermodynamic phase equilibrium is presented before the K-value method is discussed. Afterward, the equations used for enthalpy calculations are derived.

3.1 Vapor-liquid Equilibrium

To any thermodynamic equilibrium system an energy function is minimized[24]. For a closed system where temperature and pressure are kept constant, the Gibbs energy is minimized at equilibrium. As the Gibbs energy is minimized at equilibrium the following are a necessary condition for a two phase equilibrium[24],

$$(dG)_{T,p,\mathbf{n}} = \sum_{i=1}^{NC} \mu_i^{\alpha} dn_i^{\alpha} + \sum_{i=1}^{NC} \mu_i^{\beta} dn_i^{\beta}, \tag{3.1}$$

$$dn_i^{\alpha} + dn_i^{\beta} = 0. \tag{3.2}$$

Here $(dG)_{T,p,\mathbf{n}}$ is the differential of Gibbs energy at constant temperature, T, pressure, p, and total amount of each component. μ_i^{α} and μ_i^{β} is the chemical potential of component i in phase α and β respectively. dn_i^{α} and dn_i^{β} is the change in the molar holdup of component i in phase α/β. NC is the number of components in the system. Equation 3.2 comes from the total mass balance as the total amount of each component is conserved assuming no reactions. Eliminating dn_i^{β} from Equation 3.1 gives,

$$(dG)_{T,p,\mathbf{n}} = \sum_{i=1}^{NC} (\mu_i^\alpha - \mu_i^\beta) dn_i^\alpha = 0. \tag{3.3}$$

Close to the equilibrium point, all dn_i^α is independent leading to that the equilibrium condition can be written as[24],

$$\mu_i^\alpha = \mu_i^\beta, \qquad \forall i \in [1, NC] \tag{3.4}$$

In addition, both the pressure and temperature is equal in the two phases at equilibrium,

$$T^\alpha = T^\beta, \tag{3.5}$$
$$p^\alpha = p^\beta. \tag{3.6}$$

In Figure 3.1 an illustration of equilibrium in a tank containing a liquid and a vapor phase is presented. In the tank the intensive properties will be equal in both phases for all the components. The arrows indicate mass transfer which maintain the equilibrium.

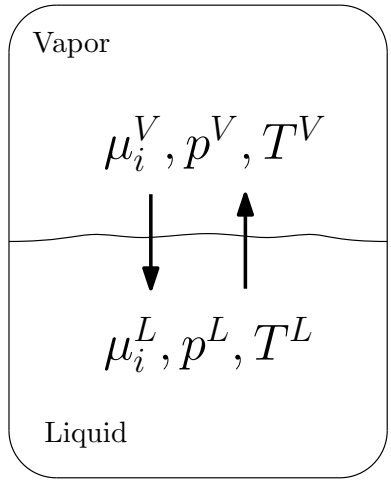


Figure 3.1: Illustration of vapor-liquid equilibrium in a tank. Thermodynamic equilibrium is defined by equal intensive properties in both phases for all the components. The arrows indicates mass transfer to maintain the equilibrium.

3.1.1 K-value Method

Vapor-liquid equilibriums are traditionally solved by the Rashford-Rice method[25]. This method is also referred to as the K-value method as the equilibrium equations are solved as a set of K-value problems on the form[24],

$$y_i \overset{\text{def}}{=} K_i x_i. \tag{3.7}$$

Here y_i and x_i is the molar fraction of component i in the vapor and liquid phase respectively. K_i is the equilibrium constant for component i. Referring to K_i as a constant is actually misguiding as K_i is a function of temperature, pressure and the composition of both phases, $K_i(T, p, \mathbf{x}, \mathbf{y})$[24]. As the necessary condition of thermodynamic equilibrium is expressed in the terms of chemical potential and not by K-values and molar fractions, it is important to know the relation between them.

The K-value and the chemical potential is related through the component fugacity coefficient, ϕ_i and/or activity coefficient, γ_i. In this thesis, the vapor phase is assumed to behave as an ideal gas, while the liquid is assumed to be an ideal mixture. With these assumptions, the K-value for component i can be calculated from Raoult's law[24]. Raoult's law can be derived by using a fugacity model for the chemical potential of the vapor phase and an activity model for the chemical potential of the vapor phase. By doing so the K-value can be expressed as[24],

$$K_i^{vle} = \frac{\phi_i^{sat} p_i^{sat} \gamma_i}{\phi_i p} \exp\left[\frac{v_i^{sat}(p - p_i^{sat})}{RT}\right]. \tag{3.8}$$

ϕ_i^{sat}, p_i^{sat} and v_i^{sat} is the fugacity coefficient, pressure and molar volume of component i at saturation. ϕ_i and γ_i is the fugacity and activity coefficient of component i at the current condition and R is the gas constant. Assuming a pure ideal liquid component the following relations hold,

$$\phi_i = \phi_i^{sat}, \tag{3.9}$$
$$\gamma_i = 1, \tag{3.10}$$
$$p = p_i^{sat}. \tag{3.11}$$

By doing so the mixture is per definition behaving ideal[24]. Equation 3.8 reduces to Raoult's law,

$$K_i^{Raoult} = \frac{p_i^{sat}}{p}, \tag{3.12}$$

which is used for the VLE calculations in this thesis.

The saturation pressure for a component i can be calculated in different ways, one of them being the integrated Clausius-Clapeyron equation[25],

$$p^{sat}(T) = p^{sat}(T_{ref}) \exp\left[-\frac{\Delta_{vap}h}{R}\left(\frac{1}{T} - \frac{1}{T_{ref}}\right)\right]. \tag{3.13}$$

T_{ref} is the reference temperature and $\Delta_{vap}h$ is the molar enthalpy of vaporization. However, in practical calculations, Equation 3.13 is not sufficiently accurate and an empirical relationship is used[25]. One of these is the Antoine equation,

$$log_{10}\left(p^{sat}(T)\right) = A - \frac{B}{T+C}. \tag{3.14}$$

Here A, B, and C are empirical coefficients which are component specific and can be found in the literature. The Antoine equation is used for calculating the saturation pressure of a component in this thesis.

3.2 Enthalpy Calculations

There are several different methods for calculating the enthalpy of a stream, holdup or phase. In this thesis, the molar enthalpy, h, is calculated as a function of the ideal molar enthalpy of each components, h_i, weighted with the molar fractions, z_i.

$$h^{\alpha} = \sum_{i}^{NC} z_i^{\alpha} h_i^{\alpha}. \tag{3.15}$$

The calculation of the molar enthalpy for a component used in this thesis can be derived from the total differential of molar internal energy $U(S, V, \mathbf{n})$. This is done as equilibrium calculations in the flash tank model is based on minimization of the Gibbs energy which is a function of temperature and pressure. In the derivation below it is shown how to go from $U(S, V, \mathbf{n}) \rightarrow U(T, p, \mathbf{n})$[24].

$$(dU)_{\mathbf{n}} = -pdV + TdS \tag{3.16}$$

$$(dV)_{\mathbf{n}} = \left(\frac{\partial V}{\partial T}\right)_{p,\mathbf{n}} dT + \left(\frac{\partial V}{\partial p}\right)_{T,\mathbf{n}} dp \tag{3.17}$$

$$(dS)_{\mathbf{n}} = \left(\frac{\partial S}{\partial T}\right)_{p,\mathbf{n}} dT + \left(\frac{\partial S}{\partial p}\right)_{T,\mathbf{n}} dp \tag{3.18}$$

$$(dU)_{\mathbf{n}} = -p\left[\left(\frac{\partial V}{\partial T}\right)_{p,\mathbf{n}} dT + \left(\frac{\partial V}{\partial p}\right)_{T,\mathbf{n}} dp\right] + T\left[\left(\frac{\partial S}{\partial T}\right)_{p,\mathbf{n}} dT + \left(\frac{\partial S}{\partial p}\right)_{T,\mathbf{n}} dp\right] \tag{3.19}$$

$$= \left[T\left(\frac{\partial S}{\partial T}\right)_{p,\mathbf{n}} - p\left(\frac{\partial V}{\partial T}\right)_{p,\mathbf{n}}\right] dT + \left[T\left(\frac{\partial S}{\partial p}\right)_{T,\mathbf{n}} - p\left(\frac{\partial V}{\partial p}\right)_{T,\mathbf{n}}\right] dp \tag{3.20}$$

$$= \left[C_p - p\left(\frac{\partial V}{\partial T}\right)_{p,\mathbf{n}}\right] dT - \left[T\left(\frac{\partial V}{\partial T}\right)_{p,\mathbf{n}} + p\left(\frac{\partial V}{\partial p}\right)_{T,\mathbf{n}}\right] dp \tag{3.21}$$

Here S is the entropy, V is volume and C_p is the heat capacity at constant pressure. The subscripts related to the differentials denotes which variables that are kept constant. To go from Equation 3.20 to 3.21 the following definition and Maxwell relation is used[24],

$$\left(\frac{\partial S}{\partial T}\right)_{p,n} \overset{\text{def}}{=} \frac{C_p}{T}$$
$$\left(\frac{\partial S}{\partial p}\right)_{T,n} = \left(\frac{\partial V}{\partial T}\right)_{p,n}.$$

From the expression of internal energy with pressure and temperature as free variables, Equation 3.21, the differential of enthalpy, H, can be derived. Starting with the definition of enthalpy[24],

$$H \overset{\text{def}}{=} U + pV, \tag{3.22}$$

The total differential of H can be expressed as,

$$(dH)_n = (dU)_n + p(dV)_n + V(dp)_n. \tag{3.23}$$

$(dU)_n$ is known from Equation 3.21, and $V(dp)_n$ is already a function of p as is desired. The total differential of V, $(dV)_n$, is given in Equation 3.17. Inserting Equation 3.21 and 3.17 into 3.23 the differential of H, $(dH)_n$ can be expressed as,

$$\begin{aligned}(dH)_n &= \left[C_p - p\left(\frac{\partial V}{\partial T}\right)_{p,n} + p\left(\frac{\partial V}{\partial T}\right)_{p,n}\right] dT \\ &- \left[T\left(\frac{\partial V}{\partial T}\right)_{p,n} + p\left(\frac{\partial V}{\partial p}\right)_{T,n} - p\left(\frac{\partial V}{\partial p}\right)_{T,n} - V\right] dp \\ &= C_p dT - \left[T\left(\frac{\partial V}{\partial T}\right)_{p,n} - V\right] dp.\end{aligned} \tag{3.24}$$

Assuming ideal gas Equation 3.24 reduces to,

$$(dH)_n = C_p dT. \tag{3.25}$$

Similarly, for the liquid phase, the change in volume as a function of change in the temperature and/or pressure is assumed to be negligible, resulting in the same equation for the change in enthalpy, Equation 3.25.

Further, in the calculation of enthalpy, a reference condition has to be chosen[24]. In this thesis a pure component in liquid phase at T_{ref} = 298.15 K is chosen. In addition,

it is assumed that the heat capacity at constant pressure, C_p, is constant. By setting this reference the calculation of the enthalpy in the vapor and liquid phase is done by the use of the following equations,

$$h_L = \sum_i^{NC} x_i C_{p,L,i} \cdot (T - T_{ref}) \tag{3.26}$$

$$h_V = \sum_i^{NC} y_i (\Delta h_{vap,i} + C_{p,V,i} \cdot (T - T_{ref})). \tag{3.27}$$

Chapter 4

Model Development

In this chapter, the development of the heat exchanger (HEX) model is presented. First, the model of a flash tank is described before the full heat exchanger model, consisting of several flash tanks in series, is introduced. In the end, an extension to a countercurrent heat exchanger is presented. The main challenges when developing the model is to make a set of equations that are fulfilled in all the different phase regions. The three phase regions are liquid-only, vapor-liquid and vapor-only. The model developed in this thesis is a semi-explicit index 1 DAE (see Section 2.5.1) which consists of differential equations for the internal energy and the component molar holdup and algebraic equations for the algebraic variables. A summary of the model equations for both the single flash tank and the HEX can be found in Appendix B.

4.1 Flash Tank

The flash tank presented in this section is modeled as a UV-flash, meaning that the internal energy U, molar holdups, $\mathbf{M}$, and total volume of the flash tank, V_T are given. With these values known, a flash calculation can be done to find the composition in the liquid and vapor phase ($\mathbf{x}$, $\mathbf{y}$), the temperature, T, the pressure, p, and vapor fraction, f[26].

$$(U, V, \mathbf{M}) \rightarrow (\mathbf{x}, \mathbf{y}, T, p, f) \tag{4.1}$$

The model is based on the evaporator model proposed by Sahlodin et al.[1]. Figure 4.1 shows a sketch of the flash tank as it is modeled in this thesis. The total molar inlet flow, F_{in}, its composition, $\mathbf{z}_{in}$, and the molar enthalpy, h_{in}, in the inlet flow is given. In addition, the total volume of the tank and the heat flow is defined. The flash tank has one liquid and one vapor outlet. Both outlets flow towards a given outlet pressure, p_0.

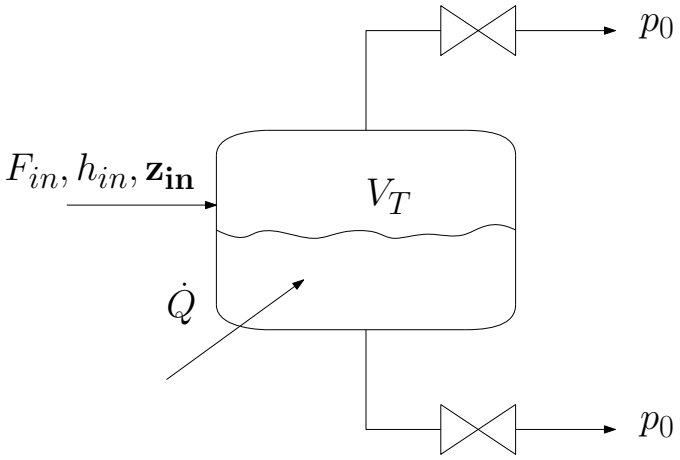


Figure 4.1: Sketch of the flash tank modeled in this thesis.

4.1.1 Differential Equations

To perform the flash calculations the internal energy, U, as well as the molar holdups, $\mathbf{M}$, needs to be known. At the initial condition these variables are given, but as they will change in time they are expressed as differential equations. The differential equations are developed through the general balance equation for a quantity Φ,

$$\frac{d\Phi}{dt} = \Phi_{in} - \Phi_{out} + \Phi_{generated} - \Phi_{loss}. \tag{4.2}$$

Assuming that there are no reactions taking place in the flash tank and that the tank is adiabatic, the generation and the loss term can be neglected. Therefore, the simplified balance equation used in the flash tank model can be written as,

$$\frac{d\Phi}{dt} = \Phi_{in} - \Phi_{out}. \tag{4.3}$$

The flash tank model includes dynamic component molar holdup and energy balance equation. The component molar holdup equation can be derived from the component mass conservation equation. The dynamic component mass balance equation is formulated from Equation 4.3,

$$\frac{dm_i}{dt} = m_{in} - m_{out} \qquad\qquad i = 1, ..., NC, \tag{4.4}$$

where m is the mass holdup in the flash tank. As it is assumed that no reactions are taking place, the mass balance can be substituted with a molar balance,

$$\frac{dM_i}{dt} = M_{in} - M_{out} \qquad i = 1, ..., NC. \tag{4.5}$$

Here M_i is molar holdup of component i. The molar feed flow rate and the composition is given as an input to the model. In addition, the vapor and liquid outlet flow rate are modeled as separate flows. By deciding to use these input and state variables the molar balance equation is written as:

$$\frac{dM_i}{dt} = z_i F_{in} - x_i F_L - y_i F_V \quad i = 1, ..., NC, \tag{4.6}$$

where z_i, x_i and y_i is the molar fraction of component i in the feed, liquid and vapor outlet flow respectively. F_{in} is the total molar flow rate into the flash tank, while F_L and F_V is the liquid and vapor outlet flow rates.

In addition to the dynamic mass balance, a dynamic energy balance is included in the flash tank model. The kinetic and potential energy is neglected, meaning that only the internal energy is balanced. The same strategy is used for the internal energy as for the mass balance, starting with Equation 4.3.

$$\frac{dU}{dt} = U_{in} - U_{out}. \tag{4.7}$$

Same as for the mass balance equation, the inlet/outlet internal energy are not system variables in the flash tank model. To include only input and system variables in the internal energy balance Equation 4.7 is rewritten,

$$\frac{dU}{dt} = F_{in} h_{in} - F_L h_L - F_V h_V + Q. \tag{4.8}$$

In Equation 4.8 h_{in}, h_L and h_V is the molar enthalpy in the inlet and liquid/vapor outlet flow. Q is the heat transferred to the system. The inlet enthalpy is given as a input by calculating the enthalpy through the temperature and composition in the inlet.

4.1.2 Algebraic Equations

All of the other system variables are calculated through algebraic equations. Given the value of the input variables and the differential variables, the value of the algebraic variables can be calculated through these algebraic equations. Even though some of the algebraic variables, such as vapor and liquid fractions, do not have a physical interpretation in all the three different phase regions, all equations have to be satisfied for any given condition.

Enthalpy Calculations

The enthalpy calculations in the flash tank model are performed by use of the equations derived in Section 3.2. The equations used to calculate the molar enthalpies in the liquid and vapor is Equation 3.26 and 3.27. To include all the model equation in this section the equation is restated below,

$$h_L = \sum_i^{NC} x_i C_{p,L,i} \cdot (T - T_{ref}), \tag{4.9}$$

$$h_V = \sum_i^{NC} y_i (\Delta H_{vap,i} + C_{p,V,i} \cdot (T - T_{ref})). \tag{4.10}$$

It is assumed that the phases are perfectly mixed, meaning that the temperature and composition are the same in the entire phase. Further, the total enthalpy holdup, H, in the flash tank is calculated by,

$$H = M_L h_L + M_V h_V. \tag{4.11}$$

Here M_L and M_V is the molar holdup of vapor and liquid respectively. In addition to these equations the enthalpy, H and internal energy, U, needs to be related as the enthalpy is relative to the selected reference state[24]. This relation is given by the definition of enthalpy, stated in Equation 3.22 as,

$$H \stackrel{\text{def}}{=} U + pV. \tag{4.12}$$

Vapor-Liquid Equilibrium

The vapor-liquid equilibrium (VLE) equations used in this section is thoroughly derived and discussed in Section 3.1. For sake of completeness, the equations used for the VLE calculations is restated below,

$$y_i = K_i x_i \qquad i = 1, ..., NC, \tag{4.13}$$

$$K_i = \frac{p_i^{sat}}{p} \qquad i = 1, ..., NC, \tag{4.14}$$

$$log_{10}(p_i^{sat}) = A_i - \frac{B_i}{T - C_i} \qquad i = 1, ..., NC. \tag{4.15}$$

As already mentioned, the proposed flash tank model should be valid for all the phase regions, meaning that there is not necessarily a vapor-liquid equilibrium present in the

flash tank. Even though, the VLE equation is continuously a part of the model equations. This will result in non-physical values for either the vapor, y_i, or liquid, x_i molar fractions at conditions resulting in a single phase being present in the flash tank. How this is handled is described in the section **Nonsmooth Formulation Approach**.

Flow Rates

The liquid and vapor flow rates are modeled as two separate flows. This is to make the flow rate of each phase to be dependent on the fraction of the volume holdup of each phase. It can be argued, that the outflow should be a function of the cross-sectional area that the phase covers, rather than the volume. However, it is assumed that using the volume is a good enough approximation to reduce the nonlinearity of the model.

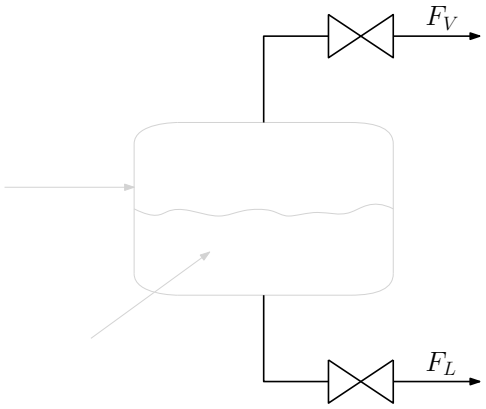


Figure 4.2: A sketch of the flash tank with the outflows highlighted.

The proposed nonsmooth equations for the flow rates are,

$$F_V = c_V \cdot \frac{V_V}{V_{tot}} \max\left(0, \frac{p - p_0}{\sqrt{|p - p_0| + \epsilon}}\right), \tag{4.16}$$

$$F_L = c_L \cdot \frac{V_L}{V_{tot}} \max\left(0, \frac{p - p_0}{\sqrt{|p - p_0| + \epsilon}}\right). \tag{4.17}$$

Here c_V and c_L is the valve coefficients for the vapor and liquid flow. V_V, V_L and V_{tot} is the vapor volume, liquid volume and total volume of the flash tank. p_0 is the outlet pressure. In addition to adjusting the flow rate for each phase, the volume of each phase are used to trigger a liquid-blocking and vapor-blocking valve when the corresponding phase volume goes to zero. The $\max$-function is used to model a check valve to prevent reverse flow. When the pressure, p, inside the tank is less than the outflow pressure, p_0,

both the outflows are shut down. Further a small value $\epsilon > 0$ is used to establish Lipschitz continuity at $p = p_0$[1]. This is important as it is a requirement to calculate the generalized derivative which is discussed in Chapter 2.

Volume, Density and Pressure Calculations

The total volume of the flash tank is fixed, and all of the volume is occupied by either a mixture of liquid and vapor or just one of the phases. This is incorporated in the model by the following equation,

$$V_{tot} = V_V + V_L. \tag{4.18}$$

As already mentioned in the derivation of the enthalpy calculation in Section 3.2, the vapor phase is assumed to behave as an ideal gas. Therefore the ideal gas law is included in the model to relate the pressure, temperature, vapor holdup, and vapor volume,

$$pV_V = M_V RT. \tag{4.19}$$

The liquid volume holdup in the tank is calculated by the relation of the liquid molar holdup and the density of the liquid, ρ_L,

$$V_L = \frac{M_L}{\rho_L}. \tag{4.20}$$

To calculate the density a modification of the equation proposed by Sahlodin et al. is used[1],

$$\frac{1}{\rho_L} = \sum_i^{NC} \frac{x_i}{\rho_i(p)}, \tag{4.21}$$

where ρ_i is the density of component i. The modification is that the liquid density of component i is a function of the pressure, making the liquid compressible. The density, as a function of pressure, is given by,

$$\rho_i(p) = \rho_i(p_{ref}) \cdot (1 + C_0(p - p_{ref})), \tag{4.22}$$

where $\rho_i(p_{ref})$ is the density of component i at the reference pressure p_{ref} and C_0 is a compressibility factor. This leads to the following model for the liquid density,

$$\frac{1}{\rho_L} = \sum_i^{NC} \frac{x_i}{\rho_i(p_{ref})\,(1 + C_0(p - p_{ref}))}. \tag{4.23}$$

By introducing this modification the index of the flash tank model is reduced from 2 to 1 for the liquid-only phase region. Thereby the DAE system is of index one in all the three phase regions. If incompressible liquid was to be assumed, information about the pressure is lost when there is no vapor present and the derivative of the algebraic equations would become rank deficient. The reason for this is that $M_V = 0$ and $V_V = 0$, leading to a trivial solution of Equation 4.19.

Molar Holdup

In addition to the differential equation for the component molar holdups, Equation 4.6, the component molar holdups need to be related to the vapor and liquid holdups as well as the composition in the different phases. These relations are included in the model by the following equations,

$$M_i = x_i M_L + y_i M_V, \qquad i = 1, ..., NC, \tag{4.24}$$

$$\sum_i^{NC} M_i = M_L + M_V. \tag{4.25}$$

Nonsmooth Formulation Approach

The main objective of this master thesis has been to develop a single model that is valid for all the three different phase regions. To achieve this all the system equations has to be satisfied at any point in the domain of the system. In this suggested model formulation this has been done by using a nonsmooth formulation suggested by Sahlodin et al[1]. This formulation includes the mid-function, described in Example 2.2. The mid-function is used to coop with the fact the VLE calculations is not physically satisfied when there is only one existing phase. This is done by extending the phase molar fraction into the phase region where they physically do not exists. The suggested formulation is as follows,

$$0 = \mathrm{mid}\left(\frac{M_V(t)}{M_V(t)+M_L(t)}, \sum_i^{NC} x_i(t) - \sum_i^{NC} y_i(t), \frac{M_V(t)}{M_V(t)+M_L(t)} - 1\right) \tag{4.26}$$

Here, the first term is valid for the liquid-only regime as the term will set the vapor holdup to be zero. The second term is valid when there is both vapor and liquid present in the flash tank, while the last term is for the vapor-only regime as it will only be satisfied as long as the liquid holdup is zero. The second term can be replaced by a dynamic extension of the Rashford-Rice equation,

$$0 = -\sum_{i=1}^{NC} \frac{M_i(k_i - 1)}{\left(\sum_{j=1}^{NC} M_j\right) + M_V(k_i - 1)}. \tag{4.27}$$

The proof of the nonsmooth formulation is based on minimization of Gibbs energy at constant temperature and pressure and can be found in Sahlodin et al[1]. In this thesis, the formulation will be justified by showing the behavior of the three equations and how the mid-function chooses the correct term given the condition of the system. To do so, a single component flash tank system will be considered.

Liquid-Only Region

When the system is below the bubble point, there will only be liquid in the flash tank. When this is the case solving the system equation should result in that the vapor holdup, M_V, is equal to zero. This means that mid-function in Equation 4.26 should select the first term. For this to be the case one of the two other terms has to be non-positive, whilst the other term should be non-negative. Looking at the third term, when M_V is zero, the term is equal to -1, meaning that it is less than zero. For the nonsmooth formulation to choose the correct the term, the second term should be greater than zero below the bubble point.

As the system is below the bubble point the saturation pressure calculated by Equation 4.15 will be less than the pressure of the system. Further, the K-value calculated by Equation 4.14 is less than 1. This will result in that the calculated vapor fraction, y, will be less than 1. Inserting this into the expression in the middle of Equation 4.26, gives that the expression is larger than zero. To sum up the values for the three different expressions for temperatures and pressures below the bubble point,

$$\begin{aligned} \frac{M_V(t)}{M_V(t) + M_L(t)} &= 0, \\ \sum_i^{NC} x_i(t) - \sum_i^{NC} y_i(t) &> 0, \\ \frac{M_V(t)}{M_V(t) + M_L(t)} - 1 &< 0. \end{aligned}$$

This will result in that the first term will active, forcing the vapor holdup to be zero when the system conditions are below the bubble point. **Note** that the vapor fraction, y, calculated at such conditions is not 1. For a multicomponent system, the molar fraction in the non-existing phase will neither sum up to 1. This is physically incorrect as per definition the sum of molar fractions in a phase should add up to 1. Therefore it should not be called a molar fraction but rather a pseudo molar fraction y'. But to simplify the notation in this thesis the molar fraction of the non-existing phase will still be referred to as a molar fraction and will be denoted as x and y.

Vapor-Liquid Equilibrium

When the system is above the bubble point and below the dew point, both phases will exist. For a single component system, the pressure and temperature of the system will be

such that the saturation pressure of the component is equal to the pressure of the system. Both the vapor and liquid holdup, will be greater than 0 but less than 1, $0 < M_V < 1$ and $0 < M_L < 1$. This results in the following values for the three terms in the mid-function in Equation 4.26,

$$\begin{aligned} \frac{M_V(t)}{M_V(t) + M_L(t)} &> 0, \\ \sum_i^{NC} x_i(t) - \sum_i^{NC} y_i(t) &= 0, \\ \frac{M_V(t)}{M_V(t) + M_L(t)} - 1 &< 0. \end{aligned}$$

Vapor-Only Region

The last remaining phase region is the vapor-only phase region. At this point, the liquid holdup, M_L, is zero and the vapor fraction, y, is 1. Further, the system conditions are above the dew point, meaning that for the given temperature and pressure, the corresponding saturation pressure, p^{sat}, is greater than the system pressure. The K-value at this conditions will be greater than 1, leading to a liquid fraction, x, that is less than one. Here, the same applies for the liquid fraction as mentioned earlier for the vapor fraction, meaning that this calculated liquid molar fraction is a pseudo fraction, x', which do not have a physical interpretation. The three terms in Equation 4.26 will have the following values for conditions above the dew point,

$$\begin{aligned} \frac{M_V(t)}{M_V(t) + M_L(t)} &> 0, \\ \sum_i^{NC} x_i(t) - \sum_i^{NC} y_i(t) &< 0, \\ \frac{M_V(t)}{M_V(t) + M_L(t)} - 1 &= 0. \end{aligned}$$

Example of the Nonsmooth Formulation

The behavior of Equation 4.26 can be illustrated by an example. In this example, only the VLE calculations are included and the results shown are steady state values. Suppose a flash tank where the feed consists of water and methanol. By changing the temperature, while keeping the pressure constant, in the tank the system will move through the different phase regimes. Figure 4.3 shows the value of each of the three terms in Equation 4.3. The continuous parts of the lines are where the corresponding part is active, while the line is

dotted where it is not active. As the figure shows, the first term is active at low temperature (liquid-only regime), while the third term is active for high temperature (vapor-only regime). The second term is active for the temperature in between (vapor-liquid regime). This corresponds to the behavior which is desired and what has been explained is the most recent sections.

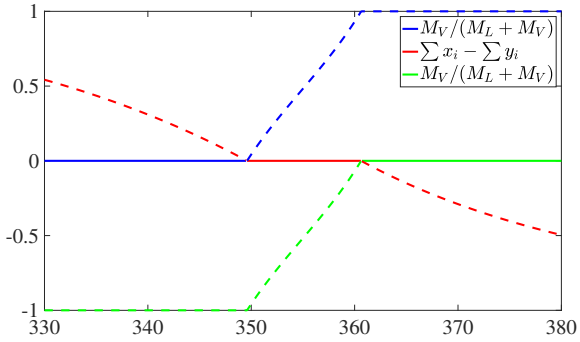


Figure 4.3: The graph shows which of the three terms in the nonsmooth mid-function is active as a function of the temperature T, with the pressure kept constant.

4.2 Heat Exchanger Model

The heat exchanger is modeled as flash tanks in series, with some modification. In this section, the HEX model, developed from the flash tank model, is presented. First, the model of one side of the HEX where the heat, Q, is considered to be fixed is presented. Secondly, a model where both sides of a countercurrent heat exchanger are included will be introduced.

4.2.1 Heat Exchanger - One Side Model

In this section, the development of one side of a heat exchanger is given. In this model, it is assumed that a given heat flow, Q, is exchanged from another thought flow. In Figure 4.4 the one-sided heat exchanger is shown. The heat exchanger is divided into M flash tanks, and it is assumed that the same amount $Q_{cell} = Q/M$ is exchanged to each of the different cells. Similar to the flash tank in Section 4.1, the inlet flow, F_{in}, inlet molar enthalpy, h_{in} as well as the composition in the inlet flow, $\mathbf{z}_{in}$ is given. In addition, the outlet pressure, p_0 is an input to the model. The flash tanks will be numbered from $j = 1, 2...M-1, M$, with $j = 1$ being the flash tank where the inlet goes into, and M is the last tank.

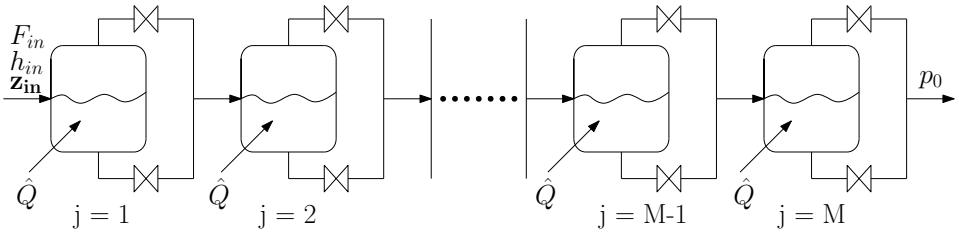


Figure 4.4: The figure shows how the heat exchanger is modeled as a set of flash tanks in series.

Differential Equations

The differential equations for the HEX is based on the differential equations for the flash tank presented in Section 4.1.1. However, to allow reversal flow inside the heat exchanger the differential equations has to be changed accordingly. If the flow reverses the composition and the molar enthalpy of the flow will be that of the cell where it normally would flow to. To include this in the model, the nonsmooth functions $\max$ and $\min$ are used to detect the direction of the flow. The formulations used in this thesis is based on the formulation proposed by Stechlinski et al.[27].

Consider a system consisting of several tanks. Each of the tanks has multiple flows entering/leaving, depending on the pressure difference. The system is presented in Figure 4.5. The accumulation of a quantity in the center tank, Φ^0, is dependent on the flow direction as well as the molar quantity of Φ, ξ, in the tank where it flows from. The accumulation of Φ over time can be expressed as the following differential equation,

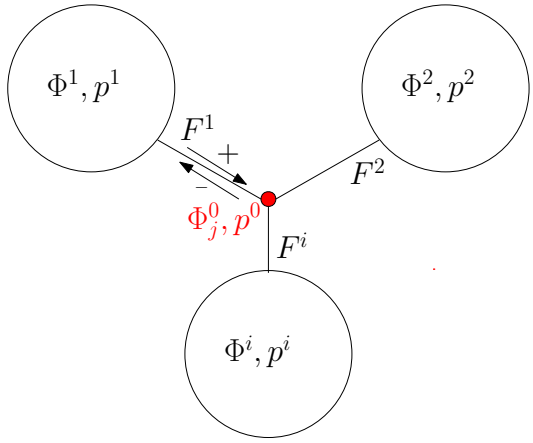


Figure 4.5: Illustration of connected tanks with multiple flows going in or out of the tanks. The flow rates are either positive or negative dependent on the pressure difference between the tanks, $p^i - p^0$. In this sketch, the flow rates are defined to be positive when going into tank 0. Φ is a general quantity. The figure is adapted from Stechlinski et al.[27].

$$\frac{d\Phi^0}{dt} = \sum_{i=1}^{n} \max\big(F^i, 0\big)\xi^i + \sum_{i=1}^{n} \min\big(F^i, 0\big)\xi^0 \tag{4.28}$$

The first sum in Equation 4.28 includes only the flows going into the center tank in Figure 4.5. The amount of Φ going into the tank is the positive flow rates multiplied with the molar quantity in the tank it comes from, ξ^i. The second sum covers the flows which are going out of the tank. This amount is equal to the flow rate of the flows leaving the tank multiplied by the molar quantity/composition in the given tank, ξ^0.

As Figure 4.4 shows, each of the tanks in the heat exchanger model has two flows entering (one vapor, one liquid) and two flows (one vapor, one liquid) leaving, except for the first tank which only has one inlet flow which is fixed. The flow directions indicated in Figure 4.4 is the positive directions, but reversal flow is possible. There are three different cases to consider. The first tank which has one fixed inlet flow. The last tank which has a check valve in the outlet flows, meaning that no reversal flow is allowed. The last case is all the other tanks. In the next illustrations and equations, only one flow in and out of the tank is considered as the vapor and liquid flow going from one tank to another will have the same direction.

Figure 4.6 shows the different flow directions for the three different cases. In Figure 4.6a the flow directions for the first tank is presented. The inlet flow, F_{in}, is given and is unidirectional. Based on Equation 4.28 the balance equation for the quantity Φ^1 can be formulated as,

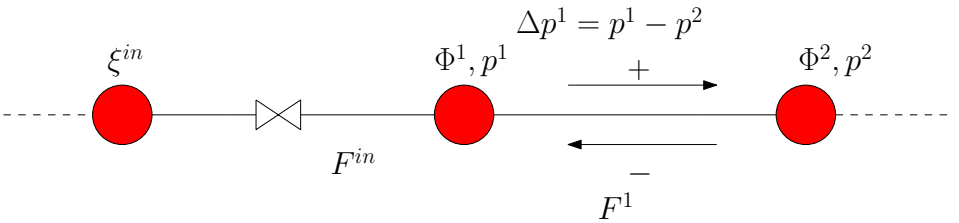
$$\frac{d\Phi^1}{dt} = F_{in}\xi_{in} - \max\big(0, F^1\big)\xi^1 - \min\big(0, F^1\big)\xi^2. \tag{4.29}$$

The $\max$-term in Equation 4.29 corresponds to when the flow goes from tank 1 to 2. In this case, the flow, F^1, is positive and the amount of quantity Φ **leaving** tank 1 is $F^1\xi^1$. The $\min$-term is non-zero when the flow direction is opposite, meaning that F^1 is below zero. When this is the case, an amount of quantity Φ equal to $-F^1\xi^2$ is **arriving** tank 1 from tank 2.

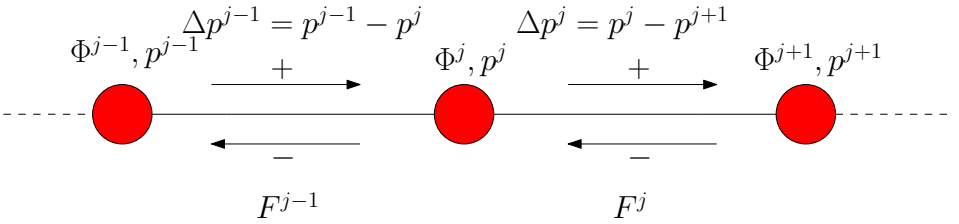
In Figure 4.6b, the flow directions for tank $j \neq \{1, M\}$ is illustrated. For these tanks, the direction of neither flows are fixed. The differential equation for quantity Φ^j can be written as,

$$\frac{d\Phi^j}{dt} = \max\big(0, F^{j-1}\big)\xi^{j-1} - \max\big(0, F^j\big)\xi^j + \min\big(0, F^{j-1}\big)\xi^j - \min\big(0, F^j\big)\xi^{j+1}. \tag{4.30}$$

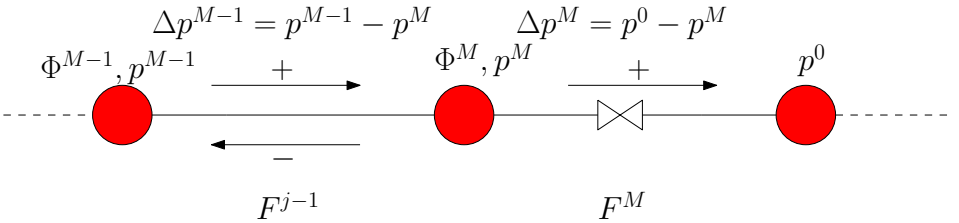
In Equation 4.30, the $\max$-terms are non-zero when the flows are positive, meaning that the flow is going from $j-1 \rightarrow j \rightarrow j+1$, whilst the $\min$-terms are non-zero when the directions are reversed. Note that only two of the terms are non-zero at the same time. This is the desired behavior as a specific flow cannot go both in and out of a tank at the same time.



(a) Flow directions into the first flash tank in the heat exchanger model.



(b) Flow directions into flash tank number $j \neq \{1, M\}$ in the heat exchanger model.



(c) Flow directions into the last flash tank, j = M, in the heat exchanger model.

Figure 4.6: The flow direction in and out of the different flash tanks which the heat exchanger consist of.

The last case is the last flash tank in the heat exchanger, $j = M$. The flow directions for this part of the HEX is presented in Figure 4.6c. The outlet from this tank F^M has a check valve meaning that no reverse flow is allowed. This is not handled in the differential equations but is included in the valve equations, Equation 4.52 and 4.53. The balance of quantity Φ^M is formulated as,

$$\frac{d\Phi^M}{dt} = \max\left(0, F^{M-1}\right)\xi^{M-1} - F^M\xi^M + \min\left(0, F^{M-1}\right)\xi^M. \tag{4.31}$$

The differential equation for the dynamic variables in the HEX model can be formulated using the balance equations for the general quantity, Φ. The differential equations for the internal energy in each part of the HEX, U^j is given as,

$$\frac{dU}{dt}^{1} = F_{in}h_{in} - \left(\max\left(0, F_L^1\right)h_L^1 + \max\left(0, F_V^1\right)h_V^1\right) \tag{4.32}$$
$$- \left(\min\left(0, F_L^1\right)h_L^2 + \min\left(0, F_V^1\right)h_V^2\right) + Q_{cell},$$

$$\frac{dU}{dt}^{j} = \left(\max\left(0, F_L^{j-1}\right)h_L^{j-1} + \max\left(0, F_V^{j-1}\right)h_V^{j-1}\right)$$
$$- \left(\max\left(0, F_L^{j}\right)h_L^{j} + \max\left(0, F_V^{j}\right)h_V^{j}\right) \tag{4.33}$$
$$+ \left(\min\left(0, F_L^{j-1}\right)h_L^{j} + \min\left(0, F_V^{j-1}\right)h_V^{j}\right)$$
$$- \left(\min\left(0, F_L^{j}\right)h_L^{j+1} + \min\left(0, F_V^{j}\right)h_V^{j+1}\right) + Q_{cell}, \quad j = 2...M-1,$$

$$\frac{dU}{dt}^{M} = \left(\max\left(0, F_L^{M-1}\right)h_L^{M-1} + \max\left(0, F_V^{M-1}\right)h_V^{M-1}\right) \tag{4.34}$$
$$- \left(F_L^M h_L^M + F_V^M h_V^M\right) + \left(\min\left(0, F_L^{M-1}\right)h_L^M + \min\left(0, F_V^{M-1}\right)h_V^M\right) + Q_{cell}.$$

Similarly, the differential of the component holdup is given as,

$$\frac{dM_i}{dt}^{1} = F_{in}z_{in,i} - \left(\max\left(0, F_L^1\right)x_i^1 + \max\left(0, F_V^1\right)y_i^1\right) \tag{4.35}$$
$$- \left(\min\left(0, F_L^1\right)x_i^2 + \min\left(0, F_V^1\right)y_i^2\right), \quad i = 1...NC$$

$$\frac{dM_i}{dt}^{j} = \left(\max\left(0, F_L^{j-1}\right)x_i^{j-1} + \max\left(0, F_V^{j-1}\right)y_i^{j-1}\right)$$
$$- \left(\max\left(0, F_L^{j}\right)x_i^{j} + \max\left(0, F_V^{j}\right)y_i^{j}\right) \tag{4.36}$$
$$+ \left(\min\left(0, F_L^{j-1}\right)x_i^{j} + \min\left(0, F_V^{j-1}\right)y_i^{j}\right)$$
$$- \left(\min\left(0, F_L^{j}\right)x_i^{j+1} + \min\left(0, F_V^{j}\right)y_i^{j+1}\right), \quad i = 1...NC, \quad j = 2...M-1,$$

$$\frac{dM_i}{dt}^{M} = \left(\max\left(0, F_L^{M-1}\right)x_i^{M-1} + \max\left(0, F_V^{M-1}\right)y_i^{M-1}\right) - \left(F_L^M x_i^M + F_V^M y_i^M\right)$$
$$+ \left(\min\left(0, F_L^{M-1}\right)x_i^M + \min\left(0, F_V^{M-1}\right)y_i^M\right), \quad i = 1...NC \tag{4.37}$$

Algebraic Equations

In every flash tank, which is part of the HEX model, the algebraic equations presented in Section 4.1.2 is solved. For the outflows, the equation is changed for all flash tanks except for the last tank to allow bi-directional flow. The equations for all the flash tanks are presented below.

$$M_i^j = M_L^j x_i^j + M_V^j y_i^j, \qquad i = 1, ..., NC, \; j = 1...M, \tag{4.38}$$

$$\sum_i^{NC} M_i^j = M_L^j + M_V^j, \qquad j = 1...M, \tag{4.39}$$

$$H^j = M_L^j h_L^j + M_V^j h_V^j, \qquad j = 1...M, \tag{4.40}$$

$$H^j = U^j + p^j V_T, \qquad j = 1...M, \tag{4.41}$$

$$y_i^j = k_i^j x_i^j, \qquad i = 1, ..., NC, \; j = 1...M, \tag{4.42}$$

$$V_T = V_L^j + V_V^j, \qquad j = 1...M, \tag{4.43}$$

$$\rho_L^j = \sum_i^{NC} \frac{x_i^j}{\rho_i \left(1 + C_0(p^j - p_{ref})\right)}, \qquad j = 1...M, \tag{4.44}$$

$$h_L^j = \sum_i^{NC} x_i^j c_{p,L,i}(T^j - T_0), \qquad j = 1...M, \tag{4.45}$$

$$h_V^j = \sum_i^{NC} y_i^j (\Delta_{vap} h_i + c_{p,V,i})(T^j - T_0), \qquad j = 1...M, \tag{4.46}$$

$$k_i^j = \frac{p_{sat,i}^j}{p^j}, \qquad j = 1...M, \tag{4.47}$$

$$\log_{10}(p_{sat,i}^j) = A_i - \frac{B_i}{T^j - C_i}, \qquad j = 1...M, \tag{4.48}$$

$$0 = \text{mid}\left(\frac{M_V^j}{M_V^j + M_L^j}, \sum_i^{NC} x_i^j - \sum_i^{NC} y_i^j, \frac{M_V^j}{M_V^j + M_L^j} - 1\right), \quad j = 1...M. \tag{4.49}$$

As mentioned, only the last tank has a reverse flow blocking valve. This leads to a different flow equation formulation for the different flash tanks. For the last tank, the flow equation is the same as in Section 4.1.2. For the other tanks, the nonsmooth $\max$ function is excluded to allow bi-directional flow, and the pressure gradient is the difference between the given tank and the next. For the last tank, $j = M$, the outlet flows towards the defined outlet pressure, p_0. To make the total pressure drop through the HEX independent on the number of cells it is divided into the valve coefficients are multiplied with $\sqrt{M}$.

$$F_V^j = c_V \sqrt{M} \cdot \frac{V_V^j}{V_T} \left(\frac{p^j - p^{j+1}}{\sqrt{|p^j - p^{j+1}| + \epsilon}}\right) \qquad j = 1...M-1 \tag{4.50}$$

$$F_L^j = c_L \sqrt{M} \cdot \frac{V_L^j}{V_T} \left(\frac{p^j - p^{j+1}}{\sqrt{|p^j - p^{j+1}| + \epsilon}}\right) \qquad j = 1...M-1 \tag{4.51}$$

$$F_V^M = c_V\sqrt{M}\cdot\frac{V_V^M}{V_T}\max\left(0,\frac{p^M-p_0}{\sqrt{|p^M-p_0|+\epsilon}}\right) \tag{4.52}$$

$$F_L^M = c_L\sqrt{M}\cdot\frac{V_L^M}{V_T}\max\left(0,\frac{p^M-p_0}{\sqrt{|p^M-p_0|+\epsilon}}\right) \tag{4.53}$$

4.2.2 Countercurrent Heat Exchanger

The model of one side of a heat exchanger can be extended to a countercurrent heat exchanger. Both sides, cold and hot side, of the HEX is divided into M parts, which is modeled as flash tanks in series like it was in the previous section. In Figure 4.7, the countercurrent HEX, as it is modeled, is illustrated. F_{in}^j, h_{in}^j, $\mathbf{z}_{in}^j$ are the input variables for one side, while F_{in}^k, h_{in}^k, $\mathbf{z}_{in}^k$ denotes the input variables for the second side. In addition, the outlet pressure is defined for both sides, p_0^j and p_0^k.

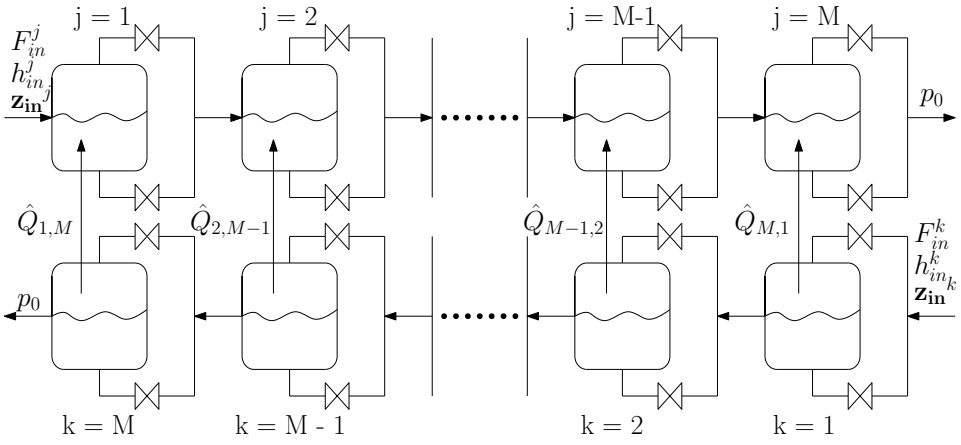


Figure 4.7: A sketch of the countercurrent heat exchanger as it is modeled with flash tanks in series exchanging heat with one another.

All equations presented in Section 4.2.1 are included for both sides. The two sides are connected by substituting the defined heat, Q, with the following equation,

$$Q = UA\Delta T. \tag{4.54}$$

Here U is the heat transfer coefficient, A is the surface area where the heat is transferred and ΔT is the temperature difference between the two flash tanks that exchange heat.

The heat transferred from one cell on one side of the HEX to the corresponding cell on the other side can be expressed by a general equation of Equation 4.54, using the indices in Figure 4.7.

$$Q_{j,k} = UA\left(T^k_{M-j+1} - T^j_j\right), \tag{4.55}$$

$$Q_{k,j} = -Q_{j,k} \tag{4.56}$$

Here $Q_{j,k}$ is the heat transferred from cell k on one side of the HEX to cell j on the other side. T^j_j is cell number j on side j, while T^k_{M-j+1} is cell number $M-j+1$ on side k. The index number $M-j+1$ will be the cell which transfer heat with cell j on side j. Equation 4.56 states that the heat transferred the other way is the same amount but with different sign.

Chapter 5

Simulation Methods

In this chapter, the simulation methods used for dynamic simulation of the models described in Chapter 4 are presented. The chapter will cover which program and solvers are used. In addition, how the simulations are initialized is described. Using SI units for the parameters and system variables makes the system ill-conditioned. Due to this, other units have been used in the simulations. The units used for the different variables can be found in Appendix A.

5.1 Solvers

The model, which is to be simulated, is formulated as a semi-explicit index 1 DAE. There are several commercial DAE solvers that can be used to simulate such systems[20], one of them being the *ode15s* solver in MATLAB®. *ode15s* is a solver that is well suited for stiff systems, something the flash tank and HEX model is due to the fast changes connected to the phase changes. However, to the best of the author's knowledge, there is no documentation that this solver can handle nonsmooth models. During simulation work done in relation to this thesis, problems using the *ode15s*-solver has occurred. The solver return an error message that it is not able to find a solution without reducing the time step below the limit at the nonsmooth points. A reason for this can be that *ode15s* uses variable time-stepping. This means that when the system changes rapidly, the time step is reduced. At a nonsmooth point, the gradient depends on which direction the system moves, as discussed in Chapter 2. Even if the time step is reduced to a very small value, the derivative will change discretely at the nonsmooth point. Therefore, using the numerically calculated derivative at the previous point will not move the system towards the correct solution at the next time which leads to the error message. It has also been tried to supply the generalized derivatives calculated by using AD-objects, see Appendix D.1. Still, the *ode15s*-solver returns the same error message. Due to these problems, a simple integrator

with a constant time step has been used to solve the system.

In this thesis, the model is simulated by using the implicit Euler integration algorithm presented in Algorithm 3. To do this, the differential equations are discretized using backward differentiation. This results in the following discrete functions for the differential equations for the single flash tank model,

$$U_{t+\Delta t} = U_t + (F_{in}h_{in} - F_{L,t+\Delta t}h_{L,t+\Delta t} - F_{V,t+\Delta t}h_{V,t+\Delta t} + Q)\Delta t, \tag{5.1}$$
$$M_{i,t+\Delta t} = M_{i,t} + (z_i F_{in} - x_{i,t+\Delta t}F_{L,t+\Delta t} - y_{i,t+\Delta t}F_{V,t+\Delta t})\Delta t, \tag{5.2}$$
$$i = 1, ..., NC.$$

Here, the value of the differential variables at the previous time, t, is known, while the variables denoted with $t + \Delta t$ is the variables that should be solved for to find the solution of the system at the next time. By rearranging Equation 5.1 and 5.2 so that they are equal to zero they can be solved as an algebraic equation. Combining these equations with the algebraic equations included in the model, a system of nonlinear algebraic equations is obtained. To simulate a time step, this system of equations can be solved by a nonlinear equation solver. In this thesis the Levenberg-Marquardt algorithm (LMA) in *fsolve* in MATLAB® has been used.

The LMA is used to solve non-linear least squares problems and can be thought of as a combination of steepest descent and the Gauss-Newton method[28]. The steepest descent is described in Equation 5.3[13], and the search direction in the Gauss-Newton method is the solution of Equation 5.5[13].

$$\mathbf{x}^{k+1} = \mathbf{x}^k - (\mathbf{p}_{SD}^k)^T \mathbf{Jf}(\mathbf{x}^k) \tag{5.3}$$
$$\mathbf{p}_{SD}^k = -\mathbf{Jf}(\mathbf{x}^k) / \left\|\mathbf{Jf}(\mathbf{x}^k)\right\| \tag{5.4}$$

$$\mathbf{Jf}(\mathbf{x}^k)^\mathbf{T}\mathbf{Jf}(\mathbf{x}^k)\mathbf{p}_{GN}^k = -\mathbf{Jf}(\mathbf{x}^k)^\mathbf{T} f(\mathbf{x}^k) \tag{5.5}$$
$$\mathbf{x}^{k+1} = \mathbf{x}^k - \left(\mathbf{Jf}(\mathbf{x}^k)^\mathbf{T}\mathbf{Jf}(\mathbf{x}^k)\right)^{-1} \mathbf{Jf}(\mathbf{x}^k)^\mathbf{T}\mathbf{f}(\mathbf{x}^k) \tag{5.6}$$

In the Gauss-Newton method, $\mathbf{Jf}(\mathbf{x}^k)^\mathbf{T}\mathbf{Jf}(\mathbf{x}^k)$, is an approximation of the Hessian of $\mathbf{f}(\mathbf{x}^k)$, $\nabla^2\mathbf{f}(\mathbf{x}^k)$. The difference between Gauss-Newton and the LMA is that LMA uses a trust-region strategy[13]. By doing so, the weakness of Gauss-Newton, which is its behavior when the Jacobian $\mathbf{Jf}$ is or is close to being rank-deficient is avoided[13]. The search direction in the LMA, $\mathbf{p}_{LM}$ is stated in Lemma 5.1,

Lemma 5.1. *(From [13]). The vector $\mathbf{p}_{LM}$ is a solution of the trust region subproblem*

$$\min_p \left\|\mathbf{Jf}(\mathbf{x}^k)\mathbf{p}^k + \mathbf{f}(\mathbf{x}^k)\right\|^2, \qquad \text{subject to } \left\|\mathbf{p}^k\right\| \leq \Delta^k,$$

if and only if $\mathbf{p}_{LM}$ *is feasible and there is a scalar* $\lambda \geq 0$ *such that*

$$\left(\mathbf{Jf}(\mathbf{x}^k)^{\mathrm{T}}\mathbf{Jf}(\mathbf{x}^k) + \lambda^k \mathbf{I}\right)\mathbf{p}_{LM}^k = -\mathbf{Jf}(\mathbf{x}^k)^{\mathrm{T}}\mathbf{f}(\mathbf{x}^k), \tag{5.7}$$

$$\lambda^k\left(\Delta^k - \left\|\mathbf{p}_{LM}^k\right\|\right) = 0. \tag{5.8}$$

In Lemma 5.1, $\Delta^k \geq 0$ is the trust-region radius[13]. If the Gauss-Newton direction, $\mathbf{p}_{GN}^k$ strictly lies inside the trust region, $\left\|\mathbf{p}_{GN}^k\right\| < \Delta^k$, and is a solution of the minimization problem in Lemma 5.1[13]. In such a case λ^k = 0. Otherwise, there exists a $\lambda^k \geq 0$ such that $\mathbf{p^k}$ is a solution of the same minimization problem. Far away from the solution, the LMA will behave as the steepest descent method, which is slow, but guarantees convergence[28]. Closer to the correct solution, it becomes Gauss-Newton method. The important difference between these two methods is that the steepest descent has a local linear convergence, while the Gauss-Newton has, or close to, local quadratic convergence[13]. For an explanation of rate of convergence see Appendix C.

In this thesis, the generalized derivative $\mathbf{G}$, calculated by the AD-objects, has been used as a replacement for the Jacobian in Equation 5.3 and 5.5. The $\left(\mathbf{Jf}(\mathbf{x}^k)^{\mathbf{T}}\mathbf{Jf}(\mathbf{x}^k)\right)^{-1}\mathbf{Jf}(\mathbf{x}^k)^{\mathbf{T}}$ part is the pseudo inverse of the Jacobian, meaning that the Gauss Newton method is the same as the Newton-type method proposed in Equation 2.8. This is due to the fact that $\left(\mathbf{Jf}(\mathbf{x}^k)^{\mathbf{T}}\mathbf{Jf}(\mathbf{x}^k)\right)^{-1}\mathbf{Jf}(\mathbf{x}^k)^{\mathbf{T}} = \mathbf{Jf}(\mathbf{x}^k)^{-1}$ if the Jacobian is square and has full rank.

A drawback of using LMA is that it does not support boundaries on the system variables. As the system equations are non-convex, this can lead to the solver finding alternative, non-physical solutions. However, using the solution at the previous time step as an initial guess makes the solver starting close to the physical solution as long as the time step Δt is sufficiently small. For the system solved in this thesis, this has been sufficient for the solver to converge to the correct solution. In addition, it is possible to include additional equations that can set a boundary on some variables that should be positive. This can be done by adding the following equation,

$$0 = a - |a|. \tag{5.9}$$

Here a is the variable that is to be constrained to be positive.

5.1.1 The Implicit Euler Integrator

The implicit Euler integrator is written in MATLAB® and can be found in Appendix D.3. All the system models are implemented as MATLAB® functions, all returning the residuals and the generalized derivative at each iteration. The calculation of the generalized derivative is done by automatic differentiation using AD-objects, see Section 2.4. By doing so the derivative can be supplied to *fsolve* and it will not be necessary to calculate the approximated derivatives by numerical differentiation which is the default approach. This reduces the amount of the function evaluation. However, creating the AD-objects at each

function call is computational demanding which increases the run time for one function evaluation.

At each time step, the following equations are solved,

$$0 = \mathbf{f}_D\left(\mathbf{x}_{t+\Delta t}, \mathbf{y}_{t+\Delta t}, t+\Delta t, \mathbf{x}_t\right), \tag{5.10}$$
$$0 = \mathbf{g}\left(\mathbf{x}_{t+\Delta t}, \mathbf{y}_{t+\Delta t}, t+\Delta t\right). \tag{5.11}$$

Here $\mathbf{f}_D$ is the discretized differential equations, presented in Equation 5.1 and 5.2, and $\mathbf{g}$ is the algebraic equations. In $\mathbf{f}_D$, $\mathbf{x}_t$ is the values of the differential variables at time t and is an input parameter in Equation 5.1 and 5.2. In figure 5.1 a scheme of a implicit Euler integration step is presented. The figure shows how the system variables are calculated at the next time t+Δt using the values of the differential variables at the previous time t.

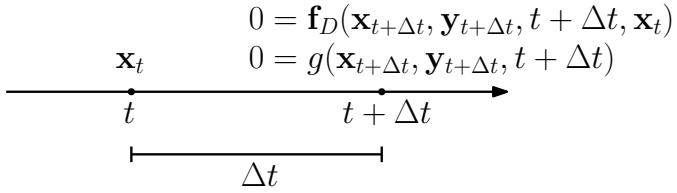


Figure 5.1: A scheme of a implicit Euler integration step from time t to time $t+\Delta t$. The function indicated at time $t+\Delta t$ is the equations solved at each time step.

To the best of the author's knowledge, there is no literature supporting that the LMA handles nonsmooth systems of equations. However, the author has good experience using this algorithm to solve systems of nonsmooth equations from previous work[4]. In addition, the solver has performed sufficiently in this thesis work. As there is little literature that supports this, a simple example can be used to illustrate that the LMA, at least in some cases, handles the nonsmooth equations. Further, the implicit Euler integration method is demonstrated in the example.

Example 5.1. *Consider the following DAE system,*

$$\frac{dx_1}{dt} = x_2, \tag{5.12}$$
$$x_2 = \max\left(1.1, t\right). \tag{5.13}$$

Performing a backward differentiation the DAE can be rewritten as a set of algebraic equations,

$$0 = x_{1,t+\Delta t} - \left(x_{1,t} + x_{2,t+\Delta t}\Delta t\right), \tag{5.14}$$
$$0 = x_{2,t+\Delta t} - \max\left(1.1, t+\Delta t\right). \tag{5.15}$$

The system is simulated using a implicit Euler integrator with an initial condition x_1 = 10, x_2 = 1.1. The generalized derivative is calculated using AD-objects and is supplied to fsolve. In Figure 5.2 the values of x_2 as a function of t as a result of simulating the equation system with different time steps are shown. As the graph shows, the LMA does not have any problems moving through the nonsmooth point. Nevertheless, the correctness of the results is sensitive to the time step. With a time step of $\Delta t = 0.2$ and $\Delta t = 0.3$ the implicit Euler integrator skips the nonsmooth point due to a too large step size. This leads to an inaccurate solution close to the nonsmooth points. However, the solver is able to find the correct solution at the next time, t. Simulating the system with a time step of $\Delta t = 0.1$ is efficiently small to capture the nonsmoothness of the system. With this step size, the solver ends up in the nonsmooth point at t = 1.1 s. Still it has no problems finding the solution at this point and move further to the next solution at t = 1.2 s. This is not a proof that the LMA handles all nonsmooth system, but in this example it has no problems moving through or ending up in nonsmooth points when provided with the generalized derivatives.

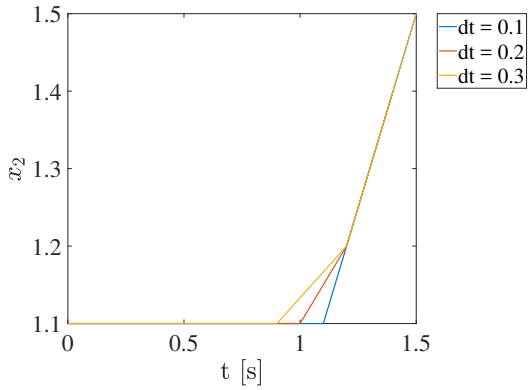


Figure 5.2: The graphs shows the values of x_2 when the DAE system is simulated with different time steps, Δt.

5.2 Initialization Methods

To start a dynamic simulation an initial condition of the system has to be provided to the solver. For semi-explicit DAEs it is possible to provide only the initial values of the differential variables and then calculate the corresponding values for the algebraic variables, see Equation 2.20. However, in this thesis the system is initialized at steady state. The steady state solution is found by setting the differential equations equal to zero, and then solve the set of algebraic equations,

$$0 = \mathbf{f}(\mathbf{x}, \mathbf{y}, t), \tag{5.16}$$

$$0 = \mathbf{g}(\mathbf{x}, \mathbf{y}, t). \tag{5.17}$$

In this section, the initialization process for the single flash, the single-sided HEX and countercurrent HEX model is described.

5.2.1 Flash tank

The flash tank model is the fundamental part of all the models in this thesis. Therefore the initialization of the other models is based on the initialization of the flash tank. Every initialization processes start with initialization of a flash tank in the vapor-only or liquid-only regime with no heating, Q = 0. With these conditions it is easier to give an initial guess for the steady state values. With no heating/cooling, the temperature, mole fractions, outlet flow and enthalpy of/in the existing phase are equal to the inlet values. In addition, the volume of the existing phase is equal to the total tank volume whilst the other phase volume is 0. Using *nonlinlsq* in MATLAB®, these variables can be constrained to be equal to these known values. By doing so, *nonlinlsq* is able to find the steady state solution.

If the system is to be initialized with another amount of heating/cooling the heating/cooling is increased/decreased step-wise using the previous solution as the initial guess. At this point, the LMA in *fsolve* is used as it has proven to have a better performance in solving the nonsmooth system.

5.2.2 Heat Exchanger

Initialization of a single side of the heat exchanger starts with initializing a single flash tank. Afterward, all the flash tanks that the HEX consists of is guessed to have the same steady state solution. As the pressure in each tank will be different due to the valve equations this will not be a correct solution for the HEX. To find the correct solution *fsolve* is used with the mentioned steady state solution of the single flash tank as an initial guess.

For the countercurrent HEX a steady state solution for both sides is first found separately. Afterward, the steady state solution for the total countercurrent HEX model is found with the heat transfer coefficient, U, equal to zero. This is done as the steady state solution of both sides of the HEX is found with no heating/cooling. Lastly, U is gradually increased finding a new steady state solution for each value of U until the steady state solution with the desired value of U is found.

Chapter 6

Results and Discussion

In this section, the results from simulating the flash tank and the HEX models, developed in Chapter 4, are presented. First, results from a dynamic simulation of a single flash tank with two components will be presented. Afterward, results from dynamic simulations of a two-component one-sided HEX will be shown and discussed. This part will also include results from a shutdown of a single-sided HEX. At the end of this chapter simulation results of a countercurrent HEX are presented.

6.1 Two Component Flash Tank

In this section the results from the dynamic simulation of a single flash tank with two components are presented. The flash tank is initialized at a steady state with zero heat flow, $Q = 0$. Component 1 is methanol, denoted with subscript 1, and component 2 is water, denoted with subscript 2. The inlet flow, F_{in} = 0.1 kmol/s, $\mathbf{z}_{in}$ = $[0.5\,0.5]$ and the inlet temperature, T_{in} = 410 K. At the initial state, the flash tank consists of vapor only. During the simulation the heat removal, Q, is increased to Q_{max} = -4 MW using the following algorithm,

Algorithm 4 Algorithm for varying the heat duty in the flash tank.

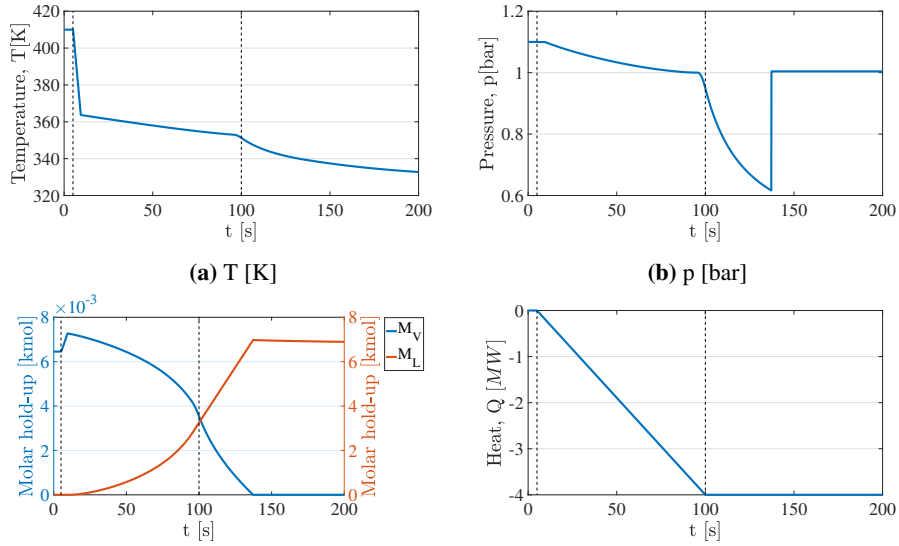
if $t < 5$ **then**
  $Q = 0$ MW
else if $5 < t < 100$ **then**
  $Q = \frac{-4}{(100-5)} \cdot (t-5)$ MW
else
  $Q = -4$ MW
end if

All other model parameters are given in table 6.1,

Parameter	Value	Parameter	Value
$c_{p,V,1}$	44.06 kJ kmol^{-1} K^{-1}	$c_{p,L,1}$	81.08 kJ kmol^{-1} K^{-1}
$c_{p,V,2}$	35 kJ kmol^{-1} K^{-1}	$c_{p,L,2}$	75 kJ kmol^{-1} K^{-1}
$h_{vap,1}$	35210 KJ kmol^{-1}	c_V	1 kmol s^{-1} MPa$^{-0.5}$
$h_{vap,2}$	40660 KJ kmol^{-1}	c_L	5 kmol s^{-1} MPa$^{-0.5}$
ρ_1	24.72 kmol m^{-3}	V_T	0.2 m^3
ρ_2	55.51 kmol m^{-3}	C_0	4.35e-4 MPa^{-1}
A_1	5.1585	A_2	4.6543
B_1	1569.613	B_2	1435.264
C_1	-34.846	C_2	-64.848

Table 6.1: Parameters used in the two component flash simulation. The Antoine parameter is taken from NIST[29]. Heat of vaporization is taken from Skogestad[25]. Density from PubChem[30].

The temperature, pressure, molar phase holdups, and the heat is presented in Figure 6.1. Initially, the system is in the vapor-only regime due to the high temperature and low pressure. At the initial condition the third term of the nonsmooth Equation 4.26 is active. At t = 5 s, when the heat removal is starting to gradually increase, the temperature starts decreasing, whilst the pressure increases. This, together with a slight increase in the molar



(a) T [K]

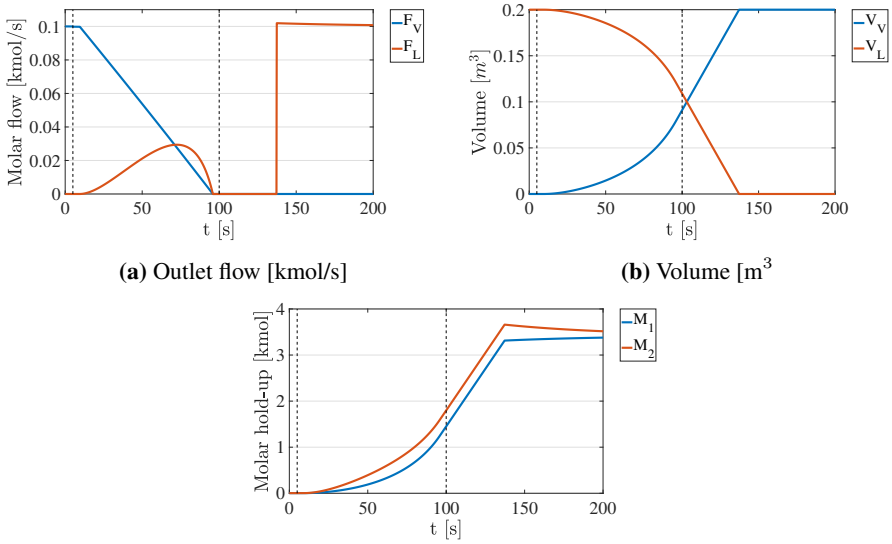
(b) p [bar]

(c) Phase holdup [kmol]

(d) Q [MW]

Figure 6.1: Temperature, pressure, molar phase holdups and the heat in the dynamic simulation of two component flash. The black dashed lines indicate where the heat removal starts to increase and where it stops increasing.

vapor holdup, happens as the total volume has to be maintained. At t $\approx$ 9.5 s, the rate at which the temperature decreases changes, and the pressure start to decrease. This change occurs as the vapor starts to condensate as the pressure is equal to the saturation pressure of water at the given temperature. This can also be seen in Figure 6.1c, as the liquid molar holdup starts to increase from 0. The system is now in the vapor-liquid equilibrium regime, meaning that the second term of Equation 4.26 is active. Further, at t $\approx$ 99 s, the temperature starts to decrease more rapidly. At the same time, the pressure goes below the outlet pressure, p_0. The reason for this is that only a small amount of vapor is present in the flash tank and therefore the temperature decreases. In addition, as the liquid has a lower density than the vapor, the pressure decreases to maintain the total volume in the tank, V_T. With a pressure in the tank below p_0, the nonsmooth formulation of the outflows, see Equation 4.16, sets the flows to be zero. This is illustrated in Figure 6.2a.



(a) Outlet flow [kmol/s]

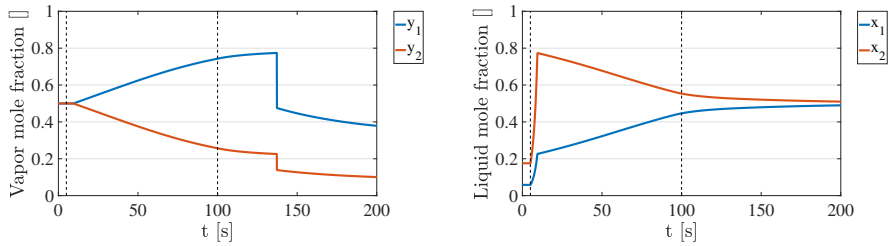
(b) Volume [m^3]

(c) Component molar holdup [kmol]

Figure 6.2: Outlet flows, F_V and F_L, phase volume, V_V and V_L, and molar component holdup, M_1 and M_2 in the dynamic simulation of two component flash. The black dashed lines indicate where the heat removal starts to increase and where it stops increasing.

The heat removal stops increasing at t = 100 s, but the system is not yet at a steady state. After t $\approx$ 135 the system enters the liquid-only regime. This can be seen both from the phase holdup in Figure 6.1c and the phase volume in 6.2b. Notice that $M_V = 0$. This is obtained by that the first term in the nonsmooth Equation 4.26 gets active. At the same time, the pressure increases and reestablishes a liquid outlet flow. Figure 6.2c shows the molar holdup of the two components. As the figure illustrates, the component molar holdup increases as the heat removal is increased. Which is, as earlier mentioned, because of the big difference in density between the two phases. The molar holdup of water is higher than of methanol for major parts of the simulation. This is explained by the higher fraction of wa-

ter in the liquid holdup (see Figure 6.3b) when the liquid phase starts to appear. This is due to the fact that water condensates at higher temperatures than methanol. Since the vapor outflow is larger than the liquid outflow at low heat duties, there is a higher accumulation of water in the tank than of methanol during this period. Close to the end of the simulation the component holdups are starting to approach each other, and when the system reaches its steady state they will be equal as the component inlet flow is equal ($\mathbf{z}_{in}$ = [0.5 0.5]).



(a) y [] **(b)** x []

Figure 6.3: Vapor composition, y_1 and y_2, and liquid composition, x_1 and x_2, in the dynamic simulation of two component flash. The black dashed lines indicate where the heat removal starts to increase and where it stops increasing.

In Figure 6.3 the composition in the vapor and liquid phase is presented. An important remark is that the molar fractions are only meaningful with the corresponding phase present in the flash tank. At the initial, condition there is no liquid present, and the flash tank model only calculates pseudo molar fraction in the liquid phase (discussed in Section 4.1.2). The mole fractions are called pseudo mole fractions as the sum of the fractions do not add up to 1, which is the definition of molar fractions. At this point the liquid mole fractions are nonphysical. Further, the system is in the vapor-only regime and therefore the two vapor molar fractions are equal and sums to 1. When the system enters the vapor-liquid regime (t $\approx$ 9.5) the liquid molar fraction adds up to 1 and are now meaningful. From this point, the fraction of water is highest in the liquid phase, while there is a higher mole fraction of methanol in the vapor phase. At the point where the last vapor disappears and the system reaches the liquid-only regime, the vapor molar fraction stops being meaningful as it was for the liquid molar fraction in the vapor-only regime.

The profile of the mole fractions reflects the nonsmooth behavior of the system. This nonsmooth behavior comes from the nonsmooth formulation given in Section 4.1.2. Even though not all variables are included in a nonsmooth equation, the nonsmoothness influences all variables as the entire system is connected. This is noticeable in the temperature profile as the gradient is dependent on which of the terms in Equation 4.26 is active. Which term is active implies whether vapor is cooled down, vapor is condensing or liquid is cooled down. Another nonsmooth formulation in the model is the outlet flow due to the $\max$-function. This nonsmoothness is illustrated by the profile of the outlet flows in Figure 6.2a. It is also important to notice that even though the outlet flows are nonsmooth at a point in time, this is not reflected by any kinks in the values of the other system variables.

6.2 Two Component HEX

In this section the results from dynamic simulation(s) of a single side of a HEX, as it is described in Section 4.2.1, are presented. The parameters used in this section are the same as for the single flash simulation, see Table 6.1. The input variables is also the same as in Section 6.1, but the simulation is initialized at a steady state with $Q = -0.08MW$. In the results presented in this section the number of flash tanks in series is set to M = 3. This number should be higher to get more exact results. However, simulating the HEX model is computational demanding. As the main objective is to show that phase changes are handled by the model, this number is assumed to be sufficient. At the end of the section simulation results of a shutdown of the HEX will be presented and discussed. An illustration of the HEX with M = 3 is presented in Figure 6.4. The variables which appear in the graphs in this section are included in the figure. The heat is changed during the simulations as described in Algorithm 4.

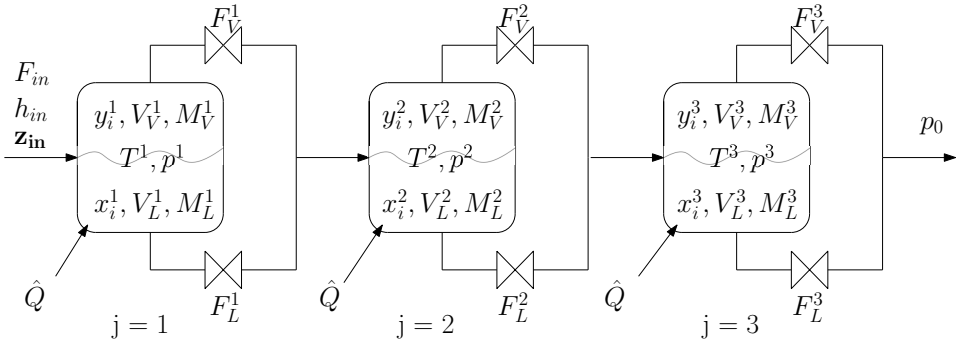
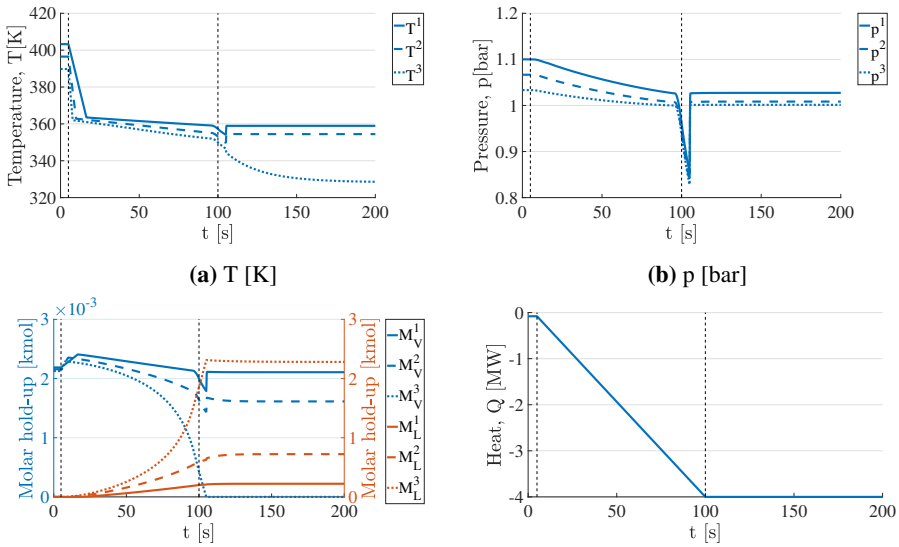


Figure 6.4: Illustration of the one-sided HEX modeled as three flash tanks in series. Variables and indices are included for each of the tanks and flows.

Figure 6.5 presents the temperature, pressure and molar holdup of liquid and vapor in each of the flash tanks in series. In addition, the total heat duty at time t is shown in Figure 6.5d. This means that at time t, the heat duty to each tank is a third of what the graph shows. At the initial condition, all the tanks are in the vapor-only regime. At t = 5 s the heat removal is starting to increase. Similar as for the single flash tank, the temperature in each cell decreases linearly the first period. This is due to the assumption of constant heat capacities and that there is only vapor present in the tanks. This means that all the heat removal only decrease the temperature of the vapor. At time t $\approx$ 8 s, vapor starts to condensate at the end of the HEX (j = 3). As the heat removal further increases vapor also starts to condensate in the middle of the HEX (j = 2, t $\approx$ 10 s), and even later at the first part of the HEX (j = 1, t $\approx$ 17 s). That condensation occurs can be seen both in that the rate of the temperature decreases and that the liquid molar holdup (see Figure 6.5c) starts to increase.

When the vapor starts to condensate the pressure decreases. The reason for this is that the pressure drop through the valves is lower with larger liquid flows as $c_L > c_V$. Up until t



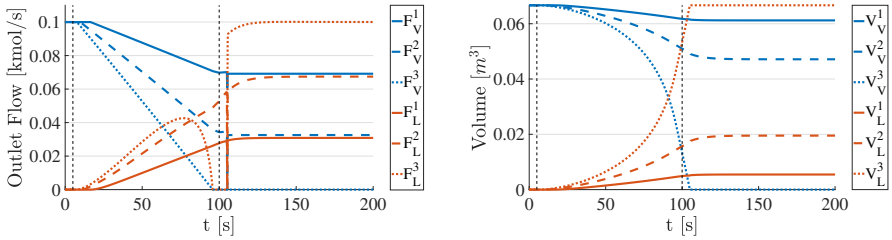
(a) T [K] **(b)** p [bar]

(c) Phase holdup [kmol] **(d)** Q [MW]

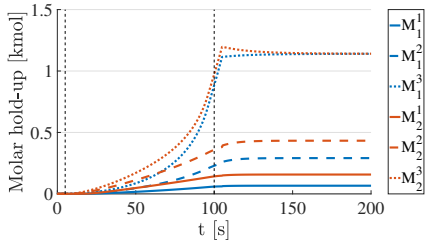
Figure 6.5: Temperature, pressure, molar phase holdups and the heat in the dynamic simulation of two component HEX one side. The black dashed lines indicate where the heat removal starts to increase and where it stops increasing.

$\approx$ 105 s the entire HEX is within the vapor-liquid equilibrium regime. At this point, the end of the HEX moves into the liquid-only regime. Prior to this, the temperature in all part of the HEX decreases significantly. This is due to the pressure drop, meaning that the temperature at which there still is vapor also decreases. When the pressure drops, the outlet flow is choked but there is still flow from the beginning of the HEX towards the end of the HEX, see Figure 6.6a. As mentioned in Section 4.1, this happens because of the big density difference between vapor and liquid. As the inlet flow does not increase at any point, it will take some time before the volume at the end of the HEX is completely filled with liquid, see Figure 6.6b. When this is achieved the pressure increases and the outlet flow is reestablished.

The outlet of the HEX is liquid only, and the temperature of the outlet keeps decreasing towards a steady state temperature as the heat duty stops changing at t = 100 s. The rest of the HEX stays within the vapor-liquid equilibrium regime for the rest of the simulation. As Figure 6.6b and 6.5c shows, there is more liquid present in the middle of the HEX than in the first part. This is the expected behavior for the same reasons as for the temperature difference at the beginning of the simulation. Figure 6.6c presents the component molar holdup in each part of the HEX. It can be seen that the differences in the component holdup are largest in the first part of the HEX. This due to a higher molar holdup of the liquid phase (see Figure 6.5c) compared to the vapor phase, and that the mole fraction of water is higher in the liquid. That this is the case is explained by the higher saturation pressure of water than for methanol at a given temperature, meaning that more water will condensate at



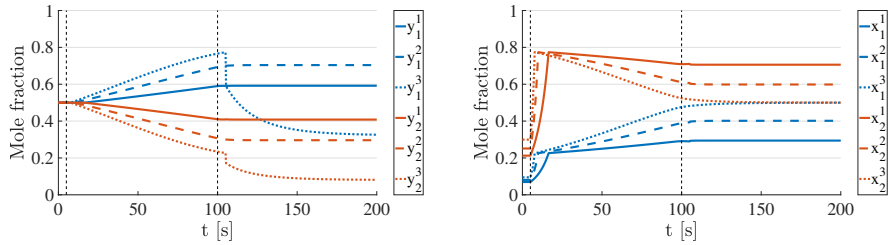
(a) Outlet flow [kmol/s] **(b)** Volume [m^3



(c) Component molar holdup [kmol]

Figure 6.6: Outlet flows, F_V and F_L, phase volume, V_V and V_L, and molar component holdup, M_1 and M_2 in the dynamic simulation of two component HEX one side. The black dashed lines indicate where the heat removal starts to increase and where it stops increasing.

the beginning of the HEX where the pressure is equal to the saturation pressure of water. Looking at the end of the HEX, the two component molar holdups are almost equal at the end of the simulation. When approaching steady state, the molar balance has to be satisfied. As there are equal amounts of the two components flowing into the system, the same amounts have to flow out at steady state. Since there is only liquid flowing out of the HEX the mole fractions must be equal to satisfy the molar balance, something they are, see Figure 6.7b.



(a) y [] **(b)** x []

Figure 6.7: Vapor composition, y_1 and y_2, and liquid composition, x_1 and x_2, in the dynamic simulation of two component HEX one side. The black dashed lines indicate where the heat removal starts to increase and where it stops increasing.

Figure 6.7 shows the composition in the vapor and liquid phase for the three parts of the HEX. As it was for the single flash tank in section 4.1, the mole fractions in the liquid phase are nonphysical at the initial condition as there is no liquid present in any part of the HEX. At t $\approx$ 8 s, when the vapor starts to condensate at the end of the HEX, the liquid fractions is meaningful in this part of the HEX. As more heat is removed and liquid starts to form in the other parts as well, the liquid mole fractions also get physical for the other parts. Both plots in Figure 6.7 illustrates the nonsmoothness of the model. The nonsmooth change in the liquid mole fractions appears when the liquid phase appears in the given cell, leading to a change in which term is active in Equation 4.26. The vapor mole fractions only have a nonsmooth behavior in the last part of the HEX. The reason for this is that this is the only part where the vapor phase disappears during the simulation.

6.2.1 Shutdown of HEX

As mentioned in Chapter 1 dynamic simulation is of specific interests when it comes to start-up and shut-down of process units. In this section, a simulation of a shutdown of a HEX is presented. The parameter values used in this section is the same as in Table 6.1. At the beginning of the simulation the heat exchanger is at steady state conditions with a inlet flow, F_{in}^0 = 0.1 kmol/s, inlet temperature, T_{in} = 410 K and a heat duty of Q = -2 MW. The HEX is divided into 3 flash tanks in series. With the initial conditions, all the parts of the HEX are in the vapor-liquid regime. The HEX is illustrated in Figure 6.4. During the shutdown the inlet flow is gradually decreased until it is completely shut down. The algorithm for doing so is presented in Algorithm 5, and the graph of F_{in} from the simulation results is illustrated in Figure 6.8.

Algorithm 5 Algorithm for gradually decreasing the inlet flow rate, F_{in}, into the HEX.

if $t < 5$ **then**
  $F_{in} = 0.1$ kmol/s
else if $5 < t < 50$ **then**
  F_{in} = 0.1 - 0.1/45 · (t - 5) kmol/s
else
  F_{in} = 0 kmol/s
end if

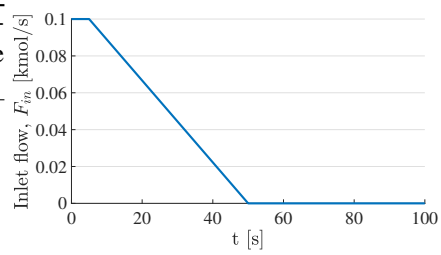


Figure 6.8: The inlet flow rate, F_{in}, into the HEX.

In addition to decreasing the inlet flow, the heat is gradually decreased. The heat is completely shut down 20 s after the inlet flow is shut. Algorithm 6 describes how the heat is decreased during the simulation, while Figure 6.9 shows the value of Q as a function of time, t.

Algorithm 6 Algorithm for gradually decreasing the heat, Q, into the HEX.

if $t < 5$ **then**
  Q = -2 MW
else if $5 < t < 70$ **then**
  Q = -2 + 2/65 · (t - 5) MW
else
  Q = 0 MW
end if

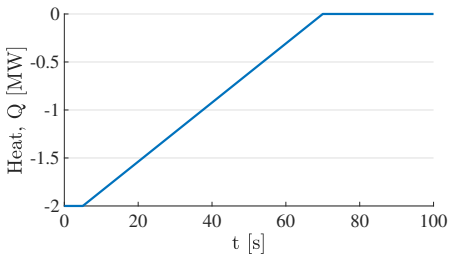
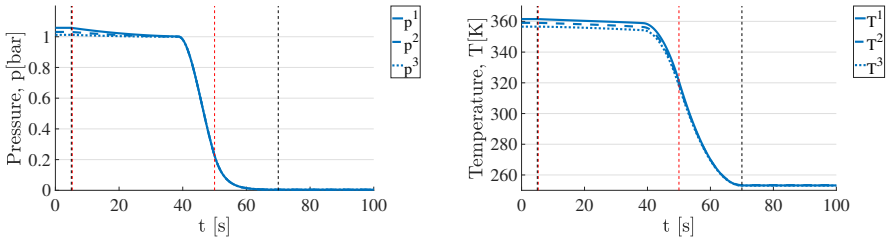


Figure 6.9: The heat, Q, into the HEX.

Figure 6.10a, 6.10b and 6.11 shows the pressure, temperature and the flow rates out of the 3 parts of the HEX. As the inlet flow rate decreases so do the pressure in the heat exchanger, leading to a decrease in the outlet flow rates. With a decrease in pressure, the temperature also decreases slightly. The reason for this is that the saturation pressure of the mixture is equal to the pressure in the vapor-liquid region. At approximately t = 39 s the pressure at the end of the HEX falls below the outlet pressure, p_0. This triggers the check valve and the outlet flow is shut. The pressure keeps falling, and so do the temperature. This can be explained by that the total volume of the tank needs to be filled with substance. As there is still some cooling, more vapor will condensate into liquid, see Figure 6.12c. With less vapor in the system, the pressure is reduced to meet the volume constraint. At the end there is almost a vacuum in the HEX with a steady state pressure of p = 426 Pa.



(a) Pressure in the different parts of the HEX from shutdown simulation.

(b) Temperature in the different parts of the HEX from shutdown simulation.

Figure 6.10: The pressure, p, temperature, T, in the different parts of the heat exchanger from shutdown simulation. The black and red dashed line indicates where the heat removal and the inlet flow rate starts decreasing and when it is completely shut down.

When the outlet flow is closed, there is still flow from the other parts of the HEX towards the end of the HEX. As this continues, the molar holdup increases in this part of the HEX. This leads to a higher pressure in this part of the HEX. Eventually, at t ≈ 47 s, the flow between the middle (j=2) and end part (j=3) of the HEX reverses, see Figure 6.11. First, the flow starts to flow from the third part of the HEX towards the middle part, while it still flows from the first part (j=1) to the middle part. But, at t ≈ 49 s, the pressure in the first part get lower than the middle part, making the flow between the first and the middle part

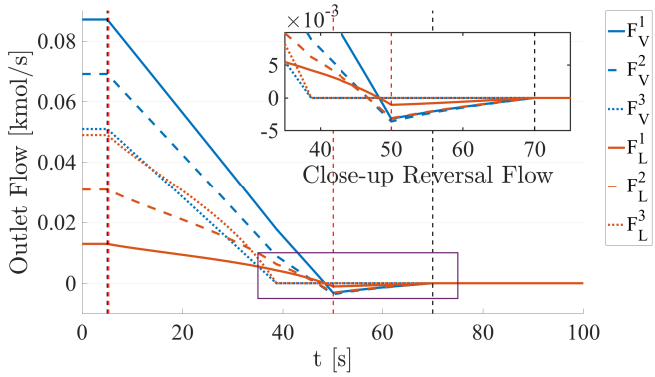
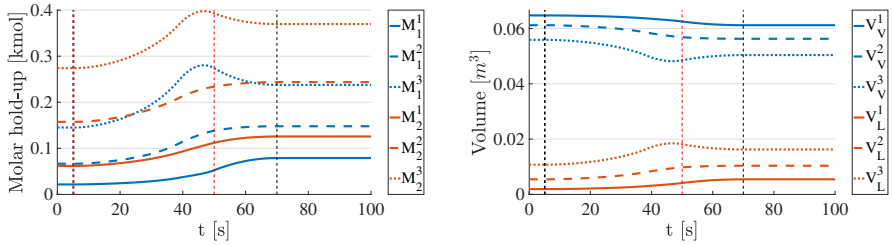


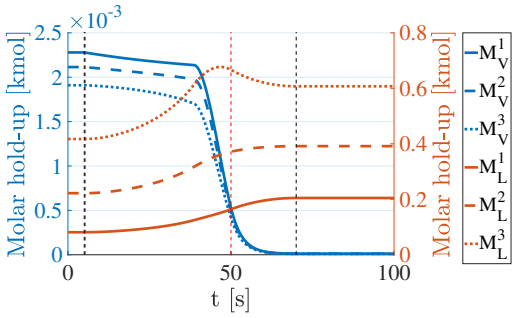
Figure 6.11: Flow rate out of the different parts of the HEX from shutdown simulation. The black and red dashed line indicates where the heat removal and the inlet flow rate starts decreasing and when it is completely shut down.

of the HEX to reverse as well. At t = 70 s the heat duty is zero, and the system reaches a steady state.



(a) Component holdups in the different parts of the HEX.

(b) Phase volume in the different parts of the HEX.



(c) Phase holdups in the different parts of the HEX.

Figure 6.12: The component holdups, M_i, phase volumes, V_L and V_V, and the phase holdups, M_V and M_L in the different parts of the HEX. The black and red dashed line indicates where the heat removal and the inlet flow rate starts decreasing and when it is completely shut down.

This case study shows that the model is able to simulate a shutdown. In addition, it shows that the model allows flow reversal. However, the numerical values obtained in this simulation can be questioned. The reason for this is that some of the assumption done in the development can affect the results. For example, the temperature in the HEX is lower than the region where the Antoine parameters, used in the simulation, is stated to be valid. This will affect the numerical value of the saturation pressure, and thereby the pressure and temperature in the HEX. Further, a linear approximation of the compressibility of the liquid phase is used. As the difference between the reference pressure (the pressure where the reference density is correct) and the pressure in the tank is significant, the assumption of a linear relation can be rather poor. However, the overall behavior of the system is not expected to change if these assumptions were to be changed. That the pressure and temperature will decrease, as well as reversal flow will occur is still the expected behavior.

6.3 Single Component Countercurrent HEX

This section will cover the results from simulating a countercurrent heat exchanger with one component on each side. In this section a superscript of C and H indicates that the variable is part of the cold or hot side respectively. The hot side is methanol with the following inlet conditions, F_{in}^{H} = 0.1 kmol/s and T_{in}^{H} = 410 K meaning that the inlet is vapor only. The cold side is water and has inlet conditions of F_{in}^{C} = 0.1 kmol/s and T_{in}^{C} = 300 K and is pure liquid. The heat transfer coefficient multiplied with the transfer area is set to be UA = 4 MW/kK. All other parameters have the same values as in Table 6.1. The countercurrent heat exchanger is illustrated in Figure 6.13.

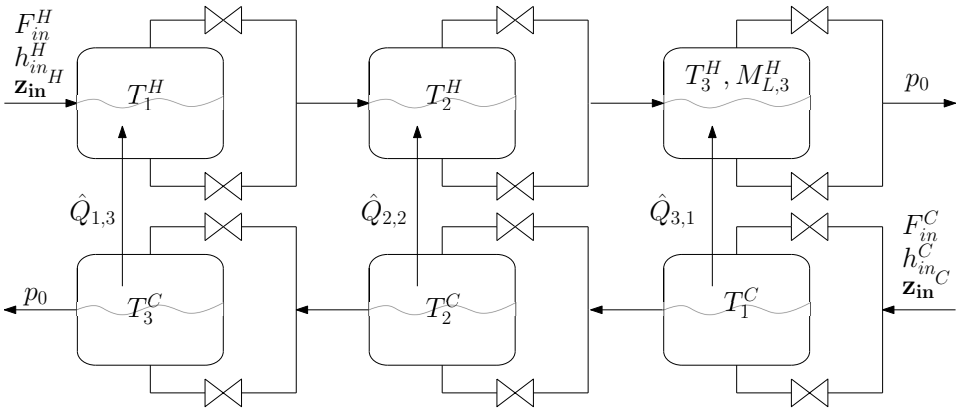


Figure 6.13: Illustration of countercurrent HEX with M = 3.

During the simulation the temperature in the inlet on the cold side changes by using Algorithm 7 and the value of the inlet temperature during the simulation are shown in Figure 6.14.

Algorithm 7 Algorithm for varying the temperature of the inlet in the cold side of the countercurrent heat exchanger.

if $t < 5$ **then**
  $T_{in}^{C} = 300$ K
else if $5 < t < 70$ **then**
  $T_{in}^{C} = 300 - 0.3(t - 5)$ K
else
  $T_{in}^{C} = 280.5$ K
end if

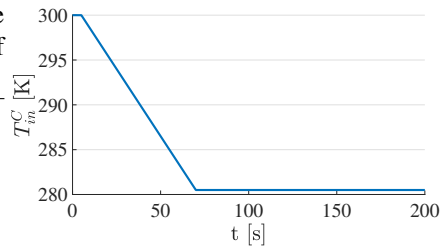


Figure 6.14: The temperature in the inlet flow on the cold side of the heat exchanger, T_{in}^{C}.

In addition to the temperature changes, the inlet flow rate on the cold side increases. The inlet flow rate does not change during the period where the temperature changes. This is to make it simpler to investigate which effect the two changes have individually. Algorithm 8 shows how the inlet flow rate is changed and Figure 6.15 presents the value during the simulation as a graph.

Algorithm 8 Algorithm for varying the inlet flow rate on the cold side of the countercurrent heat exchanger.

if $t < 100$ **then**
  $F_{in}^{C} = 0.1$ kmol/s
else if $100 < t < 150$ **then**
  $F_{in}^{C} = 0.1 - 0.001(t - 100)$ kmol/s
else
  $F_{in}^{C} = 0.15$ kmol/s
end if

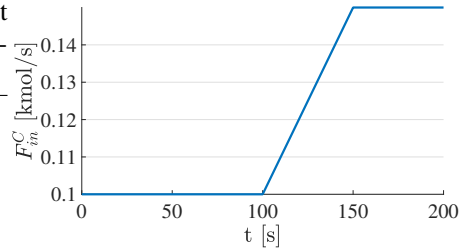


Figure 6.15: The inlet flow rate on the cold side of the heat exchanger, F_{in}^{C}.

Both when decreasing the inlet temperature and increasing the inlet flow rate on the cold side it is expected that the total heat transferred in the HEX should increase. In Figure 6.16 the total heat transferred from the cold to the hot side is presented. The values in the graph are negative, meaning that heat is removed from the hot side of the HEX during the simulation. As the graph shows, the total heat transfer has the expected behaviour. It is also interesting to inspect in which part the heat transfer increases most. As the different parts of the HEX can be in different phase regions during the simulation, it is expected that the amount of heat transferred in each part will be different.

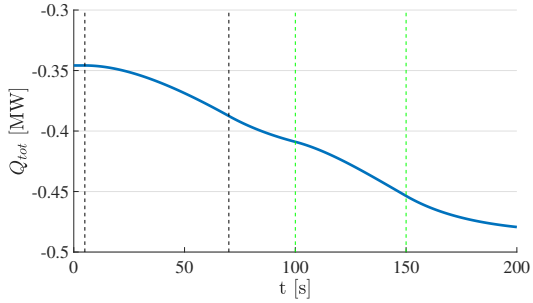
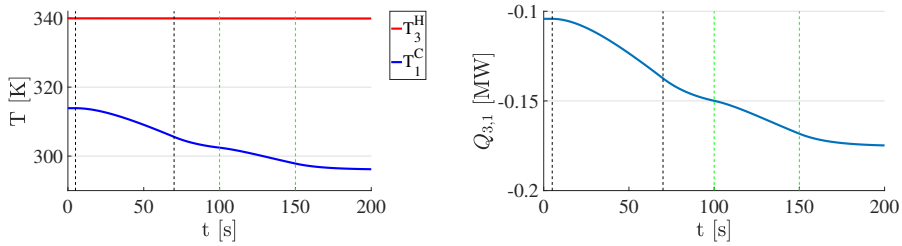


Figure 6.16: The total heat transferred from the hot to the cold side of the heat exchanger, Q_{tot}.

Figure 6.17a shows the temperature in the first part of the cold side and the end of the hot side, while Figure 6.17b shows the heat transfer from the two sides. Notice how the temperature of on the hot side is the same during the entire simulation. This is because it is in the vapor-liquid regime and all the cooling is used to condensate the methanol, increasing the molar holdup in the liquid phase, see Figure 6.18. As the temperature in the inlet on the cold side reduced, see Figure 6.14, the temperature in the first cell of the cold also decreases. This leads to that there is more heat transferred between the two cells as the temperature difference increases. The temperature in the cold side and the heat transfer is moving towards a steady state around t = 100 s, but at this point the inlet flow on the cold side starts to increase. The increase in the inlet flow makes the temperature decrease even further. The reason for this is that the residence time of the cold liquid is reduced as the flow out of the cells increases with the inlet flow, which allows more heat to be removed. When the inlet flow rate stops changing both the temperature and the heat transfer reaches a steady state.

In Figure 6.19a and 6.19b the temperature in middle of the HEX and the heat transfer between the sides are presented. In this part of the HEX the hot side is in the vapor-only



(a) Temperature at the cold side inlet and hot side outlet.

(b) Heat transfer at the cold side inlet and hot side outlet.

Figure 6.17: Temperature and heat transfer at the cold side inlet and the hot side outlet in the countercurrent HEX. The black and green dashed lines indicates where the temperature (black) in the inlet and the inlet flow rate (green) on the cold side starts and stops changing.

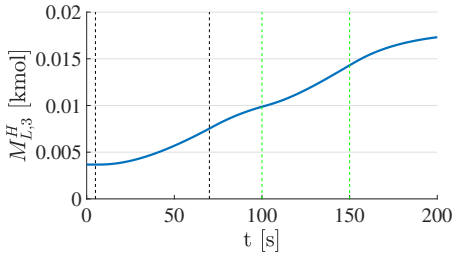
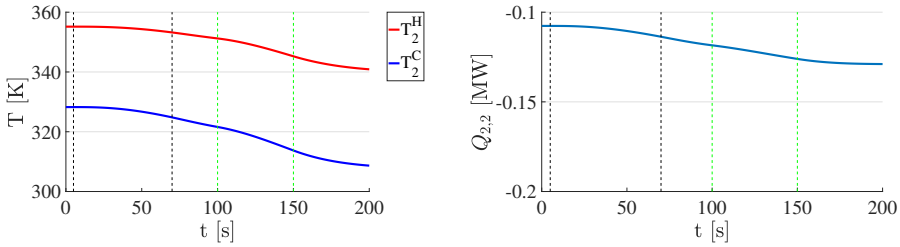


Figure 6.18: Liquid molar holdup at the cold side inlet and the hot side outlet in the countercurrent HEX. The black and green dashed lines indicates where the temperature (black) in the inlet and the inlet flow rate (green) on the cold side starts and stops changing.

regime whilst the cold side is in the liquid-only regime. From the previously described parts of the HEX it is known that the temperature of the cold side into the middle part of HEX reduces during the simulation due to the changes in the inlet temperature and flow rate. This also affects the heat transfer in the middle part. Both when the inlet temperature is reduced and the flow rate is increased the heat transfer in the middle part increases. The lower temperature in the inlet to the middle part of the HEX increases the temperature difference and thereby the heat transfer. The reduction in the temperature with increasing inlet flow rate is the same as previously discussed.

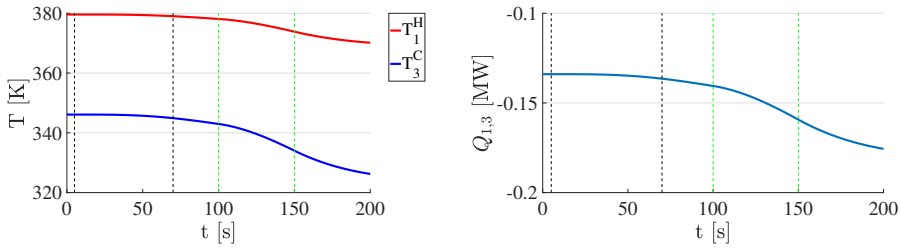


(a) Temperature of the cold and hot side at the middle of the HEX.

(b) Heat transfer between the middle part of the cold and hot side of the HEX.

Figure 6.19: Temperature and heat transfer at the middle of the countercurrent HEX. The black and green dashed lines indicates where the temperature (black) in the inlet and the inlet flow rate (green) on the cold side starts and stops changing.

The temperature and heat flow in the remaining part of the HEX are shown in Figure 6.20a and 6.20b. In this part the heat transfer increases less compared to the other parts when the inlet temperature on the cold side is decreased. The reason for this is that the temperature difference at the inlet on the cold side is increasing due to the constant temperature in the corresponding part on the hot side. As Figure 6.17b, this increase the heat transfer a lot in this part of the HEX. Therefore, the temperature difference between the two sides at the outlet on the cold side does not increase that much, leading to only a small increase in the heat transfer at this part. At t = 100 s, the flow rate on the cold side increases leading to a



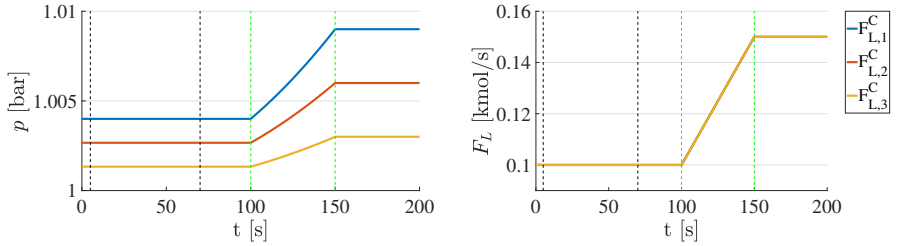
(a) Temperature at the hot side inlet and cold side outlet.

(b) Heat transfer at the hot side inlet and cold side outlet.

Figure 6.20: Temperature and heat transfer at the hot side inlet and the cold side outlet of the countercurrent HEX. The black and green dashed lines indicates where the temperature (black) in the inlet and the inlet flow rate (green) on the cold side starts and stops changing.

larger heat transfer in the heat exchanger. This is of the same reason as discussed for the other parts of the HEX. That an increase in the flow rate affects the heat transfer more in this part of the HEX is explained by that there is more substance to remove heat on the cold side throughout the HEX. This leads to a smaller increase in the temperature on the cold side of the HEX, resulting to a higher temperature difference at this part.

In addition to investigating the temperatures and the heat transfer in the HEX it also of interest to look at how the model of the HEX behave to a change in the inlet flow rate. Figure 6.21a and 6.21b shows the pressure and the outlet flow of each part of the HEX on the cold side. As the figures show, the flow rate throughout the HEX increases instantly as the inlet flow rate increases. The reason for this is that the dynamics of pressure is very fast, meaning that also the dynamics of the flow rate will be very fast. This is because the flow rates are only a function of the pressures as long as the HEX stays inside the same phase regime.



(a) Pressure in the different parts on the cold side of the HEX.

(b) Flow rate out of the different parts on the cold side of the HEX.

Figure 6.21: The pressure and liquid flow rate on the cold side of the countercurrent HEX. The black and green dashed lines indicates where the temperature (black) in the inlet and the inlet flow rate (green) on the cold side starts and stops changing.

6.4 Performance of Solver

In this thesis the results shown are based on lumping the heat exchanger into 3 cells. To obtain more exact results, this number should be higher. However, this number is chosen as the simulation run-time is rather long. This is mainly due to that MATLAB® solvers do not support other data types than doubles as initial guesses. Even if this was supported, MATLAB® is based on call-by-value, meaning that a copy of the inputs is created at each function evaluation. This is a problem as the creation of the AD-objects is computational demanding, leading to a longer run-time.

Since the AD-objects is relatively inefficient, the amount of iterations needed to find a solution is very important. To investigate the number of iterations that are needed for the solver to converge the model of one side of a heat exchanger is used. The parameters and inputs are the same as in Section 6.2. First a steady state solution of the model with no heat transfer, $Q = 0$ MW, is found. Afterwards the heat transfer is reduced with -0.01 MW and a new steady state solution is found. The number of iterations to find the steady state solution with the new value for Q is registered. In Figure 6.22 the number of iterations needed for the solver to converge as a function of the heat transfer is presented.

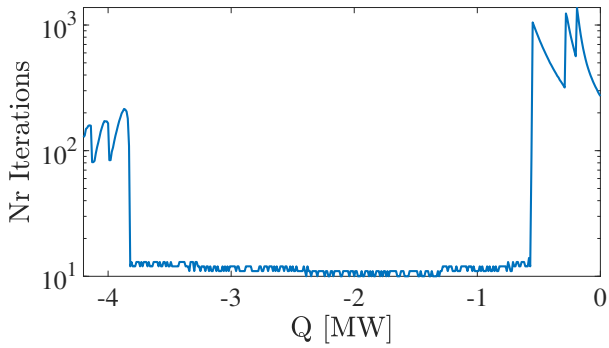


Figure 6.22: The amount of iterations used to solve the HEX model at different values of Q.

In Figure 6.22 it can be seen that the amount of iterations needed varies with the amount of heat transferred. There are 3 significant peaks from Q = 0...-0.7 MW, and the amount of iterations increases again around Q = -3.8 MW. In between only a small amount of iterations is needed. For these values of Q all the parts of the HEX are in the vapor-liquid region. The three peaks is due to the nonsmoothness which arises when a phase appears or disappears. In Figure 6.23 it shows the details for a heat transfer between Q = 0...-0.8 MW. It is also stated how many of the parts of the HEX which are in a given phase region. N_V is the number of parts in vapor-only region, N_{VL} is the number of parts in the vapor-liquid region and N_L is number of parts in the liquid-only region. The dotted lines indicate where a phase appears or disappears. The graph shows quite clearly that the number of iterations needed to find the solution increases close to, and at the nonsmooth points (where there is a phase change).

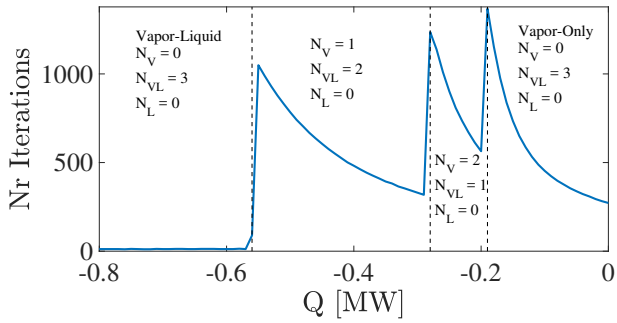


Figure 6.23: The amount of iterations used to solve the HEX model with some parts in the vapor-only region and other in the vapor-liquid region. The dotted lines indicates where there is a phase change. N_V is number in vapor-only region, N_{VL} is number in the vapor-liquid region and N_L is number the in liquid-only region.

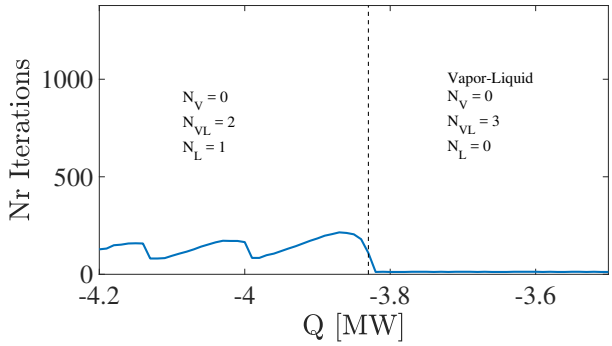
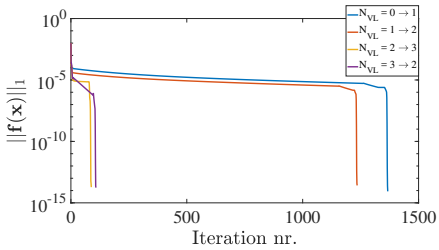


Figure 6.24: The amount of iterations used to solve the HEX model with some parts in the liquid-only and others in the vapor-liquid region. The dotted lines indicates where there is a phase change. N_V is number in vapor-only region, N_{VL} is number in the vapor-liquid region and N_L is number the in liquid-only region.

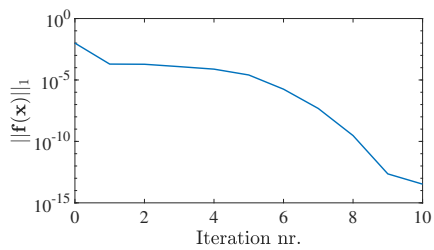
In Figure 6.24, the iterations needed to find the steady state solution for Q = -3.5...-4.2 MW is shown. For values close to Q = -3.5 MW, the entire HEX is in the vapor-liquid region. At $Q \approx$ -3.85 MW the last part of the HEX moves into the liquid-only region. At this point the number of iterations increases. This is expected as it moves through a nonsmooth point. However, the amount of iterations needed for the solver to converge at this point is less than when a part of the HEX moves from the vapor-only region to the vapor-liquid region. An indication of why there are these differences in the number of iterations the residuals and condition numbers can be investigated.

In Figure 6.25a the residuals, $\|\mathbf{f}(\mathbf{x})\|_1$, at each iteration when the solver moves toward the steady state solution is shown. From the previous steady solution to the one the solver

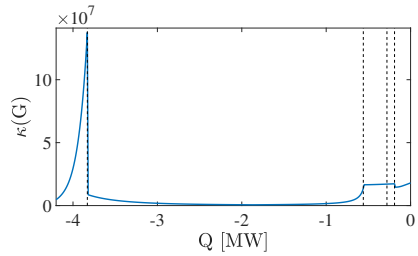
converges to there is a phase change. $N_{VL} = i \rightarrow j$ denotes that the number of parts which are in the vapor-liquid region changes from i to j. As it can be seen, the rate of convergence is close to linear and converges quite slowly at the beginning. However, as it approaches the correct solution the convergence rate improves a lot. This is the because solver gets within the neighborhood where the LMA has quadratic convergence. In Figure 6.25b the residual at each iteration when there are no phase changes between the previous and the current steady state solution is presented. In this case the solver has close to quadratic convergence from the beginning and converges after 10 iterations. It is important to notice that the residual at the initial guess is more or less the same for all the cases presented in Figure 6.25a and 6.25b. However, there is a significant difference between the number of iterations needed for the solver to converge. The condition number at the steady state solution can give an indication of why that is the case. When the entire HEX is in the vapor-liquid region (Q $\approx$ [-3.8,-0.6]) the condition number is significantly smaller than the other regions. With a high condition number, the system is said to be ill-conditioned and it is harder for the solver to find the correct solution[31]. Why there is such big difference between the different phase regions has not been thoroughly investigated. However, an explanation can be that since some of the variables are zero in the vapor-only and liquid-only region can make some of the equation to be close to linear dependent. This will make the generalized derivative matrix, $\mathbf{G}(\mathbf{x})$, close to singular and thereby the condition



(a) The residuals $\|\mathbf{f}(\mathbf{x})\|_1$ at each iteration when there occur a phase change from the previous steady state solution to the current.



(b) The residuals $\|\mathbf{f}(\mathbf{x})\|_1$ at each iteration when the entire HEX is in the vapor-liquid region both at the current and previous steady state solution.



(c) The condition number of the generalized derivative, $\kappa(\mathbf{G}(\mathbf{x}))$ at the solution as a function of the heat transfer, Q.

Figure 6.25: The residuals, $\|\mathbf{f}(\mathbf{x})\|_1$, at each iteration and the condition numbers $\kappa(\mathbf{G}(\mathbf{x})$ at the steady state solutions.

number increases. Another explanation can be that some of the variables are badly scaled. However, this will affect the condition number in the vapor-liquid region as well.

6.5 Further Discussion

The results presented in this thesis corresponds to the expected behavior of a heat exchanger. However, the results obtained are not compared to any real process data. As no specific heat exchanger is considered in this work comparing to real data is not possible. If a specific heat exchanger was to be considered, it is expected that the current dynamic model will deviate in some extent to real data. The reason for this is some of the assumptions made during the development of the model. Assumptions like ideal gas and liquid as well as constant heat capacity are something that should be revised to get results closer to real data. It is possible to include nonideal behavior for both the vapor and liquid phase by incorporating cubic equations of state (EOS) such as Peng-Robinson or Soave-Redlich-Kwong (SRK) to the model. There are some challenges related to incorporating such EOS, mainly associated with the number of real roots of the EOS depending on which phase regime the system finds itself in. The assumption of constant heat capacity can be replaced with a polynomial where the heat capacity of a component is a function of the temperature.

Chapter 7

Concluding Remarks

This thesis presents a nonsmooth formulation for a multiphase multicomponent heat exchanger which handles phase changes. The heat exchanger is modeled as multiple flash tanks in series with a fixed volume. To detect and adapt to appearance and disappearance of phases the mid function, which is a $\mathcal{PC}^1$-function, is used. To maintain the same number of equations in all the phase regimes (vapor-only, vapor-liquid, liquid-only) the phase mole fractions are extended into the phase-regime where they normally does not exist. The existence, uniqueness, and sensitivity of its solution are supported by the nonsmooth analysis thoroughly discussed in Chapter 2. Here it is explained how the generalized derivatives of $\mathcal{PC}^1$-functions can be calculated and that they can be used as a replacement for the Jacobian elements.

In this thesis dynamic simulations of a single flash tank, one side of a HEX in addition to a countercurrent HEX has successfully been performed. The model has proved to both detect and adapt to the appearance and disappearance of phases. Further, the behavior of the model coincides with the expected behavior of a HEX. However, simulating the model is relatively computational demanding. The reason for this is that the calculation of the generalized derivatives, which is used as a replacement for the Jacobian in the *fsolve*-solver in MATLAB®, is computational demanding. The automatic differentiation was implemented using operator overloading in MATLAB®. Therefore, the computation takes time as all the operations are redefined, together with the derivative rules. In addition, MATLAB® does not support other data types than doubles as input for the system variables to their solvers. This leads to that the AD-objects needs to be created at each function evaluation. Another remark regarding the simulation of the model is that the DAEs *ode15s* implemented in MATLAB® was not able to simulate the system. Because of this, an implicit Euler integrator with constant time step was used for simulation purposes. To solve the system of equations at each time, t, the Levenberg-Marquardt algorithm in *fsolve* was used. To the best of the author's knowledge, there is no literature supporting that this algorithm handles nonsmooth equations. However, the algorithm was able to solve the nonsmooth equations

in the HEX model. In addition, an example showing that the LMA handles nonsmooth, at least in some cases, is presented in this thesis.

7.1 Suggestion for Further Work

The proposed model can be used to simulate shut-down, start-up in addition to the effect of different disturbance. Therefore it is suitable to use for optimization studies, or it could be implemented as part of a larger dynamic model, like a refrigeration cycle. Such optimization studies are not within the scope of this thesis but are something that could be interesting to investigate further. This is suggested as optimization of HEXs could lead to a significant decrease in energy consumption. However, as the model is relatively computational demanding to solve. It should be considered to implement the model and the automatic differentiation in a low-level language such as C/C++ before integrating it in a larger system or performing optimization studies. As C/C++ support call-by-reference, it will not be necessary to create the AD-objects at each function evaluation.

Another improvement that should decrease the simulation time of the model is to make a integrator without a constant time step. As shown in Section 6.4 it is easier to solve the system when the entire HEX is in the vapor-liquid regime. In this region it should be possible to use larger time steps. However, using a constant time step, the time step is limited by that it is difficult to converge when approaching a phase change. If the time step is not fixed it is possible to adjust it to both how close it is to a phase change and which phase-region the system is in at the current time. A suggestion for how to make this work is to use the mid-function. If two of the terms are close to zero, the system is close to a nonsmooth point and the time step should be decreased. In addition, it can be used to detect which phase region the HEX is in by checking which of the three terms are zero.

Another suggestion for further work is to extend the model to include nonideal vapor and liquid. This can be done by including cubic equations of state such as Peng-Robinson or Soave-Redlich-Kwong. A challenge here will be to find the correct roots of the cubic equation for the different phase regions and it should be considered whether this should be done in the inner or outer loop. In addition, the assumption of constant heat capacity can be revised. The heat capacity can be implemented as a function of the temperature using a polynomial function with experimental parameters.

In this thesis, no specific heat exchanger has been considered, but rather a generic model was developed. The parameters of this model can be fitted to some real equipment. A possibility is to use sizing parameters from an existing heat exchanger and thereafter use real process data to tune the other model parameters. By doing this, the model could be used to simulate different cases as part of training for process operators. In addition, the model could be used for model predictive control (MPC) and thereby improve the overall control of the HEX. As a result, the energy consumption could be reduced.

Bibliography

[1] Ali M Sahlodin, Harry A. J. Watson, and Paul I. Barton. "Nonsmooth model for dynamic simulation of phase changes". In: *AIChE Journal* 62.9 (2016), pp. 3334–3351.

[2] Christina Florina Zotica. "Dynamic Simulation of Heat Exchanger with Multicomponent Phase Changes". MA thesis. Norwegian University of Technology, 2017.

[3] Øivind Wilhelmsen, Geir Skaugen, Morten Hammer, Per Eilif Wahl, and John Christian Morud. "Time Efficient Solution of Phase Equilibria in Dynamic and Distributed Systems with Differential Algebraic Equation Solvers". In: *Industrial & Engineering Chemistry Research* 52.5 (2013), pp. 2130–2140.

[4] Marius Reed. *Nonsmooth analysis of connected oil well system*. Project Thesis, Department of Chemical Engineering, Norwegian University of Science and Technology, 2017.

[5] H. Pingaud, J.M. Le Lann, B. Koehret, and M.C. Bardin. "Steady-state and dynamic simulation of plate fin heat exchangers". In: *Computers & Chemical Engineering* 13.4 (1989), pp. 577 –585.

[6] Ravindra S. Kamath, Lorenz T. Biegler, and Ignacio E. Grossmann. "An equation-oriented approach for handling thermodynamics based on cubic equation of state in process optimization". In: *Computers & Chemical Engineering* 34.12 (2010), pp. 2085 –2096.

[7] Harry A.J. Watson and Paul I. Barton. "Simulation and Design Methods for Multiphase Multistream Heat Exchangers". In: *IFAC-PapersOnLine* 49.7 (2016), pp. 839 –844.

[8] Kamil A. Khan and Paul I. Barton. "Generalized Derivatives for Solutions of Parametric Ordinary Differential Equations with Non-differentiable Right-Hand Sides". In: *Journal of Optimization Theory and Applications* 163.2 (2014), pp. 355–386.

[9] Kamil A. Khan and Paul I. Barton. "Switching behavior of solutions of ordinary differential equations with abs-factorable right-hand sides". In: *Systems & Control Letters* 84 (2015), pp. 27 –34.

[10] Kamil A. Khan and Paul I. Barton. “A vector forward mode of automatic differentiation for generalized derivative evaluation”. In: *Optimization Methods and Software* 30.6 (2015), pp. 1185–1212.
[11] Kamil A. Khan and Paul I. Barton. “Generalized Derivatives for Solutions of Parametric Ordinary Differential Equations with Non-differentiable Right-Hand Sides”. In: *Journal of Optimization Theory and Applications* 163.2 (2014), pp. 355–386.
[12] Stefan Scholtes. *Introduction to Piecewise Differentiable Equations*. Jan. 2012.
[13] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2nd. Springer, 2006.
[14] Marlene L. Lund. “Implementation and evaluation of a new solution approach for semidefinite programming”. MA thesis. Norwegian University of Technology, 2017.
[15] Mathias Vikse. “Design and Implementation of Modular Subroutines for Simulation of LNG Plants.” MA thesis. Norwegian University of Technology, 2016.
[16] Telma Caputti. “The plenary hull of the generalized Jacobian matrix and the inverse function theorem in subdifferential calculus”. In: *Colloquium Mathematicae*. Vol. 60. 1. Institute of Mathematics Polish Academy of Sciences. 1990, pp. 15–20.
[17] Liqun Qi and Jie Sun. “A nonsmooth version of Newton’s method”. In: *Mathematical Programming* 58.1 (1993), pp. 353–367.
[18] Yu. Nesterov. “Lexicographic differentiation of nonsmooth functions”. In: *Mathematical Programming* 104.2 (2005), pp. 669–700.
[19] Richard D. Neidinger. “Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming”. In: *SIAM Review* 52.3 (2010), pp. 545–563.
[20] Haavard I. Moe. “Dynamic Process Simulation - Studies on modeling and index reduction”. PhD thesis. Norwegian Institute of Technology, 1995.
[21] Endre Sli and David F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.
[22] Peter G. Stechlinski and Paul I. Barton. “Dependence of solutions of nonsmooth differential-algebraic equations on parameters”. In: *Journal of Differential Equations* 262.3 (2017), pp. 2254 –2285.
[23] Peter Stechlinski, Michael Patrascu, and Paul I. Barton. “Nonsmooth differential-algebraic equations in chemical engineering”. In: *Computers & Chemical Engineering* 114 (2018), pp. 52 –68.
[24] Tore Haug-Warberg. *Den termodynamiske arbeidsboken*. Kolofon Forlag, 2006.
[25] Sigurd Skogestad. *Chemical ans Energy Process Engineering*. CRC Press, 2008.
[26] P. Flatby, S. Skogestad, and P. Lundstrm. “Rigorous Dynamic Simulation of Distillation Columns Based on UV-Flash”. In: *IFAC Proceedings Volumes* 27.2 (1994), pp. 261 –266.
[27] Peter Stechlinski, Michael Patrascu, and Paul I. Barton. “Nonsmooth differential-algebraic equations in chemical engineering”. In: *Computers & Chemical Engineering* (2017).
[28] Manolis Lourakis. “A Brief Description of the Levenberg-Marquardt Algorithm Implemened by levmar”. In: 4 (Jan. 2005).
[29] The National Institute of Standards and Technology (NIST). *NIST Chemistry WebBook*. URL: `https://webbook.nist.gov/chemistry/` (visited on 06/20/2018).

[30] PubChem. *PubChem - Open Chemistry Database*. URL: https://pubchem.ncbi.nlm.nih.gov (visited on 06/20/2018).
[31] Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

Appendix A

Units Used in Simulations

State variable	Description	Unit
U	Internal energy	MJ
M_i	Molar hold-up of component i	kmol
T	Temperature	kK
P	Pressure	MPa
x_i	Molar fraction of component i in liquid phase	-
y_i	Molar fraction of component i in vapor phase	-
M_L	Liquid molar hold-up	kmol
M_V	Vapor molar hold-up	kmol
V_L	Liquid volume	m^3
V_V	Vapor volume	m^3
ρ_L	Liquid density	Mmol m^{-3}
$P_{s,i}$	Saturation pressure of component i	MPa
k_i	Vapor-liquid equilibrium coefficient for component i	-
H	Enthalpy	MJ
h_L	Liquid molar enthalpy	MJ $kmol^{-1}$
h_V	Vapor molar enthalpy	MJ $kmol^{-1}$
F_V	Vapor flow rate	kmol s^{-1}
F_L	Liquid flow rate	kmol s^{-1}

Table A.1: State variables in the flash tank model

Appendix B

Summary: Model Equations

In this appendix the model equations for the flash tank and HEX is presented. This to make it easier for the reader to see all the model equations that are included in the calculations in one place. The part with $i = 1...NC$ and $j = 1...M$ is not included in the equations to simplify the equations. However, where a subscript of i is included it is given that this equation is for $i = 1...NC$, while a superscript of j means that the equation is for $j = 1...M$.

B.1 Total Flash Tank Model Equations

$$\frac{dM_i}{dt}(t) = F_{in}(t)z_i - F_L(t)x_i(t) - F_V(t)y_i(t) \tag{B.1}$$

$$\frac{dU}{dt}(t) = F_{in}(t)h_{in}(t) - F_L(t)h_L(t) - F_V(t)h_V(t) + Q(t) \tag{B.2}$$

$$M_i(t) = M_L x_i(t) + M_V(t)y_i(t) \tag{B.3}$$

$$\sum_i^{NC} M_i(t) = M_L(t) + M_V(t) \tag{B.4}$$

$$H(t) = M_L(t)h_L(t) + M_V(t)h_V(t) \tag{B.5}$$

$$H(t) = U(t) + p(t)V_T \tag{B.6}$$

$$y_i(t) = k_i(t)x_i(t) \tag{B.7}$$

$$0 = mid\left(\frac{M_V(t)}{M_V(t) + M_L(t)}, \sum_i^{NC} x_i(t) - \sum_i^{NC} y_i(t), \frac{M_V(t)}{M_V(t) + M_L(t)} - 1\right) \tag{B.8}$$

$$V_T = V_L(t) + V_V(t) \tag{B.9}$$

$$F_V(t) = c_V \cdot \frac{V_V(t)}{V_T} \max\left(0, \frac{p(t) - p_0}{\sqrt{|p(t) - p_0| + \epsilon}}\right) \tag{B.10}$$

$$F_L(t) = c_L \cdot \frac{V_L(t)}{V_T} \max\left(0, \frac{p(t) - p_0}{\sqrt{|p(t) - p_0| + \epsilon}}\right) \tag{B.11}$$

$$\rho_L(t) = \sum_i^{NC} \frac{x_i(t)}{\rho_i \left(1 + C_0(p(t) - p_{ref})\right)} \tag{B.12}$$

$$h_L(t) = \sum_i^{NC} x_i(t) c_{p,L,i}(T(t) - T_0) \tag{B.13}$$

$$h_V(t) = \sum_i^{NC} y_i(t)(\Delta_{vap} h_i + c_{p,V,i})(T(t) - T_0) \tag{B.14}$$

$$k_i = \frac{p_{s,i}(t)}{p(t)} \tag{B.15}$$

$$\log_{10}(P_{s,i}(t)) = A_i - \frac{B_i}{T(t) - C_i} \tag{B.16}$$

B.2 One Side of Heat Exchanger

$$\begin{aligned} \frac{dM_i}{dt}^1 &= F_{in} z_{in,i} - \left(\max\left(0, F_L^1\right) x_i^1 + \max\left(0, F_V^1\right) y_i^1\right) \\ &\quad - \left(\left(\min\left(0, F_L^1\right) x_i^2 + \min\left(0, F_V^1\right) y_i^2\right), \quad i = 1...NC \end{aligned} \tag{B.17}$$

$$\begin{aligned} \frac{dM_i}{dt}^j &= \left(\max\left(0, F_L^{j-1}\right) x_i^{j-1} + \max\left(0, F_V^{j-1}\right) y_i^{j-1}\right) \\ &\quad - \left(\max\left(0, F_L^j\right) x_i^j + \max\left(0, F_V^j\right) y_i^j\right) \\ &\quad + \left(\min\left(0, F_L^{j-1}\right) x_i^j + \min\left(0, F_V^{j-1}\right) y_i^j\right) \\ &\quad - \left(\min\left(0, F_L^j\right) x_i^{j+1} + \min\left(0, F_V^j\right) y_i^{j+1}\right), \quad i = 1...NC, \quad j = 2...M-1, \end{aligned} \tag{B.18}$$

$$\begin{aligned} \frac{dM_i}{dt}^M &= \left(\max\left(0, F_L^{M-1}\right) x_i^{M-1} + \max\left(0, F_V^{M-1}\right) y_i^{M-1}\right) - \left(F_L^M x_i^M + F_V^M y_i^M\right) \\ &\quad + \left(\min\left(0, F_L^{M-1}\right) x_i^M + \min\left(0, F_V^{M-1}\right) y_i^M\right), \quad i = 1...NC \end{aligned} \tag{B.19}$$

$$\begin{aligned} \frac{dU}{dt}^1 &= F_{in} h_{in} - \left(\max\left(0, F_L^1\right) h_L^1 + \max\left(0, F_V^1\right) h_V^1\right) \\ &\quad - \left(\min\left(0, F_L^1\right) h_L^2 + \min\left(0, F_V^1\right) h_V^2\right) + Q_{cell}, \end{aligned} \tag{B.20}$$

$$\begin{aligned}\frac{dU^j}{dt} &= \left(\max\left(0, F_L^{j-1}\right)h_L^{j-1} + \max\left(0, F_V^{j-1}\right)h_V^{j-1}\right) \\ &- \left(\max\left(0, F_L^{j}\right)h_L^{j} + \max\left(0, F_V^{j}\right)h_V^{j}\right) \\ &+ \left(\min\left(0, F_L^{j-1}\right)h_L^{j} + \min\left(0, F_V^{j-1}\right)h_V^{j}\right) \\ &- \left(\min\left(0, F_L^{j}\right)h_L^{j+1} + \min\left(0, F_V^{j}\right)h_V^{j+1}\right) + Q_{cell}, \quad j = 2...M-1,\end{aligned} \tag{B.21}$$

$$\begin{aligned}\frac{dU^M}{dt} &= \left(\max\left(0, F_L^{M-1}\right)h_L^{M-1} + \max\left(0, F_V^{M-1}\right)h_V^{M-1}\right) \\ &- \left(F_L^M h_L^M + F_V^M h_V^M\right) + \left(\min\left(0, F_L^{M-1}\right)h_L^M + \min\left(0, F_V^{M-1}\right)h_V^M\right) + Q_{cell}.\end{aligned} \tag{B.22}$$

$$M_i^j(t) = M_L^j x_i^j(t) + M_V^j(t) y_i^j(t) \tag{B.23}$$

$$\sum_i^{NC} M_i^j(t) = M_L^j(t) + M_V^j(t) \tag{B.24}$$

$$H^j(t) = M_L^j(t) h_L^j(t) + M_V^j(t) h_V^j(t) \tag{B.25}$$

$$H^j(t) = U^j(t) + p^j(t) V_T \tag{B.26}$$

$$y_i^j(t) = k_i^j(t) x_i^j(t) \tag{B.27}$$

$$0 = mid\left(\frac{M_V^j(t)}{M_V^j(t) + M_L^j(t)}, \sum_i^{NC} x_i^j(t) - \sum_i^{NC} y_i^j(t), \frac{M_V^j(t)}{M_V^j(t) + M_L^j(t)} - 1\right) \tag{B.28}$$

$$V_T^j = V_L^j(t) + V_V^j(t) \tag{B.29}$$

$$F_V^j(t) = c_V \cdot \frac{V_V^j(t)}{V_T} \max\left(0, \frac{p^j(t) - p_0}{\sqrt{|p^j(t) - p_0| + \epsilon}}\right) \tag{B.30}$$

$$F_L^j(t) = c_L \cdot \frac{V_L^j(t)}{V_T} \max\left(0, \frac{p^j(t) - p_0}{\sqrt{|p^j(t) - p_0| + \epsilon}}\right) \tag{B.31}$$

$$\rho_L^j(t) = \sum_i^{NC} \frac{x_i^j(t)}{\rho_i \left(1 + C_0(p^j(t) - p_{ref})\right)} \tag{B.32}$$

$$h_L^j(t) = \sum_i^{NC} x_i^j(t) c_{p,L,i}(T^j(t) - T_0) \tag{B.33}$$

$$h_V^j(t) = \sum_i^{NC} y_i^j(t)(\Delta_{vap} h_i + c_{p,V,i})(T^j(t) - T_0) \tag{B.34}$$

$$k_i^j = \frac{p_{s,i}^j(t)}{p^j(t)} \tag{B.35}$$

$$\log_{10}(P_{s,i}^j(t)) = A_i - \frac{B_i}{T^j(t) - C_i} \tag{B.36}$$

B.3 Counter-Current Heat Exchanger

In this section all of the equation is for $i = 1...NC$, $j = 1...M$ and $k \in C, H$. Here C and H denotes the hot and cold side of the HEX.

$$\begin{aligned}\frac{dM_i}{dt}^{1,k} &= F_{in}z_{in,i} - \left(\max\left(0, F_L^1\right)x_i^1 + \max\left(0, F_V^1\right)y_i^1\right) \\ &\quad - \left(\left(\min\left(0, F_L^1\right)x_i^2 + \min\left(0, F_V^1\right)y_i^2\right), \quad i = 1...NC\end{aligned} \tag{B.37}$$

$$\begin{aligned}\frac{dM_i}{dt}^{j,k} &= \left(\max\left(0, F_L^{j-1}\right)x_i^{j-1} + \max\left(0, F_V^{j-1}\right)y_i^{j-1}\right) \\ &\quad - \left(\max\left(0, F_L^{j}\right)x_i^{j} + \max\left(0, F_V^{j}\right)y_i^{j}\right) \\ &\quad + \left(\min\left(0, F_L^{j-1}\right)x_i^{j} + \min\left(0, F_V^{j-1}\right)y_i^{j}\right) \\ &\quad - \left(\min\left(0, F_L^{j}\right)x_i^{j+1} + \min\left(0, F_V^{j}\right)y_i^{j+1}\right), \quad i = 1...NC, \quad j = 2...M-1,\end{aligned} \tag{B.38}$$

$$\begin{aligned}\frac{dM_i}{dt}^{M,k} &= \left(\max\left(0, F_L^{M-1}\right)x_i^{M-1} + \max\left(0, F_V^{M-1}\right)y_i^{M-1}\right) - \left(F_L^M x_i^M + F_V^M y_i^M\right) \\ &\quad + \left(\min\left(0, F_L^{M-1}\right)x_i^{M} + \min\left(0, F_V^{M-1}\right)y_i^{M}\right), \quad i = 1...NC\end{aligned} \tag{B.39}$$

$$\begin{aligned}\frac{dU}{dt}^{1,H} &= F_{in}h_{in} - \left(\max\left(0, F_L^1\right)h_L^1 + \max\left(0, F_V^1\right)h_V^1\right) \\ &\quad - \left(\min\left(0, F_L^1\right)h_L^2 + \min\left(0, F_V^1\right)h_V^2\right) + UA\left(T^{C,M} - T^{H,1}\right),\end{aligned} \tag{B.40}$$

$$\begin{aligned}\frac{dU}{dt}^{j,H} &= \left(\max\left(0, F_L^{j-1}\right)h_L^{j-1} + \max\left(0, F_V^{j-1}\right)h_V^{j-1}\right) \\ &\quad - \left(\max\left(0, F_L^{j}\right)h_L^{j} + \max\left(0, F_V^{j}\right)h_V^{j}\right) \\ &\quad + \left(\min\left(0, F_L^{j-1}\right)h_L^{j} + \min\left(0, F_V^{j-1}\right)h_V^{j}\right) \\ &\quad - \left(\min\left(0, F_L^{j}\right)h_L^{j+1} + \min\left(0, F_V^{j}\right)h_V^{j+1}\right) + UA\left(T^{C,M-j+1} - T^{H,j}\right), \\ &\quad j = 2...M-1,\end{aligned} \tag{B.41}$$

$$\begin{aligned}\frac{dU}{dt}^{M,H} &= \left(\max\left(0, F_L^{M-1}\right)h_L^{M-1} + \max\left(0, F_V^{M-1}\right)h_V^{M-1}\right) \\ &\quad - \left(F_L^M h_L^M + F_V^M h_V^M\right) + \left(\min\left(0, F_L^{M-1}\right)h_L^M + \min\left(0, F_V^{M-1}\right)h_V^M\right) \\ &\quad + UA\left(T^{C,1} - T^{H,M}\right),\end{aligned} \tag{B.42}$$

$$\begin{aligned}\frac{dU}{dt}^{1,C} &= F_{in}h_{in} - \left(\max\left(0, F_L^1\right)h_L^1 + \max\left(0, F_V^1\right)h_V^1\right) \\ &\quad - \left(\min\left(0, F_L^1\right)h_L^2 + \min\left(0, F_V^1\right)h_V^2\right) - UA\left(T^{C,M} - T^{H,1}\right),\end{aligned} \tag{B.43}$$

$$
\begin{aligned}
\frac{dU}{dt}^{j,C} &= \left(\max\left(0, F_L^{j-1}\right)h_L^{j-1} + \max\left(0, F_V^{j-1}\right)h_V^{j-1}\right) \\
&\quad - \left(\max\left(0, F_L^{j}\right)h_L^{j} + \max\left(0, F_V^{j}\right)h_V^{j}\right) \\
&\quad + \left(\min\left(0, F_L^{j-1}\right)h_L^{j} + \min\left(0, F_V^{j-1}\right)h_V^{j}\right) \\
&\quad - \left(\min\left(0, F_L^{j}\right)h_L^{j+1} + \min\left(0, F_V^{j}\right)h_V^{j+1}\right) - UA\left(T^{C,M-j+1} - T^{H,j}\right), \\
&\quad j = 2...M-1,
\end{aligned} \tag{B.44}
$$

$$
\begin{aligned}
\frac{dU}{dt}^{M,C} &= \left(\max\left(0, F_L^{M-1}\right)h_L^{M-1} + \max\left(0, F_V^{M-1}\right)h_V^{M-1}\right) \\
&\quad - \left(F_L^M h_L^M + F_V^M h_V^M\right) + \left(\min\left(0, F_L^{M-1}\right)h_L^M + \min\left(0, F_V^{M-1}\right)h_V^M\right) \\
&\quad - UA\left(T^{C,1} - T^{H,M}\right).
\end{aligned} \tag{B.45}
$$

$$M_i^{j,k}(t) = M_L^{j,k}x_i^{j,k}(t) + M_V^{j,k}(t)y_i^{j,k}(t) \tag{B.46}$$

$$\sum_i^{NC} M_i^{j,k}(t) = M_L^{j,k}(t) + M_V^{j,k}(t) \tag{B.47}$$

$$H^{j,k}(t) = M_L^{j,k}(t)h_L^{j,k}(t) + M_V^{j,k}(t)h_V^{j,k}(t) \tag{B.48}$$

$$H^{j,k}(t) = U^{j,k}(t) + p^{j,k}(t)V_T \tag{B.49}$$

$$y_i^{j,k}(t) = k_i^{j,k}(t)x_i^{j,k}(t) \tag{B.50}$$

$$0 = mid\left(\frac{M_V^{j,k}(t)}{M_V^{j,k}(t) + M_L^{j,k}(t)}, \sum_i^{NC} x_i^{j,k}(t) - \sum_i^{NC} y_i^{j,k}(t), \frac{M_V^{j,k}(t)}{M_V^{j,k}(t) + M_L^{j,k}(t)} - 1\right) \tag{B.51}$$

$$V_T^{j,k} = V_L^{j,k}(t) + V_V^{j,k}(t) \tag{B.52}$$

$$F_V^{j,k}(t) = c_V \cdot \frac{V_V^{j,k}(t)}{V_T}\max\left(0, \frac{p^{j,k}(t) - p_0}{\sqrt{|p^{j,k}(t) - p_0| + \epsilon}}\right) \tag{B.53}$$

$$F_L^{j,k}(t) = c_L \cdot \frac{V_L^{j,k}(t)}{V_T}\max\left(0, \frac{p^{j,k}(t) - p_0}{\sqrt{|p^{j,k}(t) - p_0| + \epsilon}}\right) \tag{B.54}$$

$$\rho_L^{j,k}(t) = \sum_i^{NC} \frac{x_i^{j,k}(t)}{\rho_i\left(1 + C_0(p^{j,k}(t) - p_{ref})\right)} \tag{B.55}$$

$$h_L^{j,k}(t) = \sum_i^{NC} x_i^{j,k}(t)c_{p,L,i}(T^{j,k}(t) - T_0) \tag{B.56}$$

$$\tag{B.57}$$

$$h_V^{j,k}(t) = \sum_i^{NC} y_i^{j,k}(t)(\Delta_{vap}h_i + c_{p,V,i})(T^{j,k}(t) - T_0) \tag{B.58}$$

$$k_i^{j,k} = \frac{p_{s,i}^{j,k}(t)}{p^{j,k}(t)} \tag{B.59}$$

$$\log_{10}(P_{s,i}^{j,k}(t)) = A_i - \frac{B_i}{T^{j,k}(t) - C_i} \tag{B.60}$$

Appendix C

Rate of Convergence

In this chapter the local rate of convergence of nonlinear programming algorithms are introduced. It is emphasized that the rate of convergence described in this is on local rate of convergence. This means that the convergence rate is only current in the neighboorhood of an optimal solution[31]. There are approaches that can provide information about the progress of a method far away from the convergence limit such as the *Computational complexity* and *Informational complexity* approach[31]. However, this information is often pessimistic as it considers the worst possible problem instance[31]. The theory presented in this Appendix is taken from *Nonlinear Programming - Bertsekas*[31] if not otherwise stated.

C.1 The Local Analysis Approach

The local analysis approach considers the local behavior of a method. It describes the behavior near the solution quite accurately by using Taylor series approximations. However, the behavior far from the solution is ignored. The rate of convergence of a method is obtained by investigating how an error function $\mathbf{e}(\mathbf{x})$ changes from one iteration to another. Typical choices for $\mathbf{e}(\mathbf{x})$ is,

$$\mathbf{e}(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}^*\|_2 , \tag{C.1}$$

$$\mathbf{e}(\mathbf{x}) = \|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}^*)\|_1 . \tag{C.2}$$

In Equation C.1 and C.2 $\mathbf{x}$ is the system variables, $\mathbf{x}^*$ is the optimal solution and $\mathbf{f}(\mathbf{x})$ is the residual of the system equations. $\|\|_1$ and $\|\|_2$ denotes the absolute value and the euclidean norm. $\sup$ denotes the *supremum*.

A method with *linear* convergence will satisfy the following inequality,

$$\limsup_{k\to\infty} \frac{e(\mathbf{x}^{k+1})}{e(\mathbf{x}^k)} \leq \beta, \tag{C.3}$$

where $\beta \in (0,1)$ is some scalar. The further away β is from unity, the faster the method converges. A method is said to converge *superlinearly* if,

$$\limsup_{k\to\infty} \frac{\mathbf{e}(\mathbf{x}^{k+1})}{\mathbf{e}(\mathbf{x}^k)} = 0, \tag{C.4}$$

Further it is possible to show that a method has *quadratic* convergence if,

$$\limsup_{k\to\infty} \frac{\mathbf{e}(\mathbf{x}^{k+1})}{\mathbf{e}(\mathbf{x}^k)^2} < \infty, \tag{C.5}$$

C.2 The Effect of the Condition Number

In Chapter 2 the condition number κ was defined in Equation 2.23. An alternative definition is[31],

$$\kappa(\mathbf{A}) = \frac{\lambda_{max}}{\lambda_{min}}. \tag{C.6}$$

Here λ_i is the eigenvalues of the matrix $\mathbf{A}$. To find the condition number of a system of equations, the matrix $\mathbf{A}$ is substituted with the Hessian, $\nabla^2\mathbf{f}(\mathbf{x})$. The best convergence rate for the steepest descent method is bounded by the following inequality,

$$\frac{\left\|x^{k+1}\right\|_2}{\left\|x^k\right\|_2} \leq \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} - \lambda_{min}}. \tag{C.7}$$

From Equation C.7 it can be seen that for a system with a high condition number will have a lower rate of convergence compared with a well conditioned system.

Appendix D

MATLAB® code

In this appendix, the automatic differentiation class, *valder*, the HEX models, and the initialization of values and solvers are presented as MATLAB® code.

D.1 valder.m

```
%*********************************************************************
% @author: Marius Reed, Marlene L. Lund
% @organization: Process Systems Engineering, NTNU
% @since: September 2017
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Stores double values and their LD-derivative
%*********************************************************************

classdef valder
    properties
        val  %function value
        der  %derivative value or gradient vector
    end
    methods
        %Constructor of a valder object.
        function obj = valder(a,b)
            if nargin == 0
                obj.val = [];
                obj.der = [];
            elseif nargin == 1
                obj.val = a;
                obj.der = 0;
            else
                obj.val = a;
                obj.der = b;
            end
```

```
end

function val = getVal(obj)
    val = obj.val;
end

function der = getDer(obj)
    der = obj.der;
end

% Creating a vector from the valder object.
function vec = double(obj)
    vec = [obj.val, obj.der];
end

% Overloading plus for the object
function h = plus(u,v)
    if ~isa(u,'valder')
        h = valder(u+v.val, v.der);
    elseif ~isa(v,'valder')
        h = valder(v+u.val, u.der);
    else
        h = valder(u.val+v.val, u.der+v.der);
    end
end

% Overloading negative for the object
function h = uminus(u)
    h = valder(-u.val, -u.der);
end

% Overloading minus for the object
function h = minus(u,v)
    if ~isa(u,'valder')
        h = valder(u-v.val, -v.der);
    elseif ~isa(v,'valder')
        h = valder(u.val-v, u.der);
    else
        h = valder(u.val-v.val, u.der-v.der);
    end
end

% Overloading multiplication for the object
function h = mtimes(u,v)
    if ~isa(u,'valder')
        h = valder(u*v.val, u*v.der);
    elseif ~isa(v,'valder')
        h = valder(v*u.val, v*u.der);
    else
        h = valder(u.val*v.val, u.der*v.val + u.val*v.der);
    end
end

% Overloading division for the object
function h = mrdivide(u,v)
    if ~isa(u,'valder')
        h = valder(u/v.val, -u*v.der/(v.val)^2);
```

```
    elseif ~isa(v,'valder')
        h = valder(u.val/v, u.der/v);
    else
        h = valder(u.val/v.val, (u.der*v.val - u.val*v.der)/v.val^2);
    end
end

% Overloading power for the object
function h = mpower(u,v)
    if ~isa(u,'valder')
        h = valder(u^v.val, u^v.val*log(u)*v.der);
    elseif ~isa(v,'valder')
        h = valder(u.val^v, v*u.val^(v-1)*u.der);
    else
        h = exp(v*log(u));
    end
end

% Overloading exponenital for the object
function h = exp(u)
    h = valder(exp(u.val), exp(u.val)*u.der);
end

% Overloading log for the object
function h = log(u)
    h = valder(log(u.val),(1/u.val)*u.der);
end
% Overloading log10 for the object
function h = log10(u)
    h = valder(log10(u.val),(1/u.val)*u.der);
end

% Overloading the square root for the object
function h = sqrt(u)
    h = valder(sqrt(u.val), u.der/(2*sqrt(u.val)));
end

% Overloading sine for the object
function h = sin(u)
    h = valder(sin(u.val), cos(u.val)*u.der);
end

% Overloading cosine for the object
function h = cos(u)
    h = valder(cos(u.val), -sin(u.val)*u.der);
end

% Overloading tan for the object
function h = tan(u)
    h = valder(tan(u.val), sec(u.val)^2*u.der);
end

% Overloading arcsine for the object
function h = asin(u)
    h = valder(asin(u.val), u.der/sqrt(1-u.val^2));
end
```

```
% Overloading arctan for the object
function h = atan(u)
    h = valder(atan(u.val), u.der/(1+u.val^2));
end

% Overloading the absolute function for the object using
% lexicographic derivatives
function u = abs(u)
    s.type = '()'; % reference type
    for i = 1:length(u.val)
        s.subs = {i};
        uvar = subsref(u,s); % store ith element
        x = double(uvar); % convert to double

        % assign abs value and derivative to output :
        v = abs(uvar.val);
        d = valder.fsign(x)*uvar.der ;
        u = subsasgn (u, s, valder(v, d));
    end
end

% Overloading the max function for the object
function h = max(u)
    s.type = '()';
    if length(u.val) > 2
        s.subs = {length(u.val)};
        u_l = subsref(u,s);
        s.subs = {1:length(u.val)-1};
        u_f = subsref(u,s);
        h = max2(u_l,max(u_f));
    else
        s.subs = {1};
        u_f = subsref(u,s);
        s.subs = {2};
        u_l = subsref(u,s);
        h = max2(u_f,u_l);
    end
end

% Overloading the min function for the object
function h = min(u)
    s.type = '()';
    if length(u.val) > 2
        s.subs = {length(u.val)};
        u1 = subsref(u,s);
        s.subs = {1:length(u.val)-1};
        u2 = subsref(u,s);
        h = min2(u1,min(u2));
    elseif length(u.val) == 1
        h = u;
    else
        s.subs = {1};
        u1 = subsref(u,s);
        s.subs = {2};
        u2 = subsref(u,s);
        h = min2(u1,u2);
    end
```

```
end

% Overloading the max function for two objects
function h = max2(u,v)
    h = (u + v + abs(u-v))/2;
end

% Overloading the min function for two objects
function h = min2(u,v)
    h = (u + v - abs(u-v))/2;
end

% Overloading the mid-function for the object
function h = mid(u)
    s.type = '()';
    s.subs = {1};
    u1 = subsref(u,s);
    s.subs = {2};
    u2 = subsref(u,s);
    s.subs = {3};
    u3 = subsref(u,s);
    s.subs = {1};
    u = subsasgn(u,s,min2(u1,u2));
    s.subs = {2};
    u = subsasgn(u,s,min2(u1,u3));
    s.subs = {3};
    u = subsasgn(u,s,min2(u2,u3));
    h = max(u);
end

% Overloading the midfunction for the three objects
function h = midobj(u1,u2,u3)
    s.type = '()';
    s.subs = {1};
    u = valder();
    u = subsasgn(u,s,min2(u1,u2));
    s.subs = {2};
    u = subsasgn(u,s,min2(u1,u3));
    s.subs = {3};
    u = subsasgn(u,s,min2(u2,u3));
    h = max(u);
end

% Overloading the 1 and inf norm for the object
function h = norm(u,p)
    switch p
        case 1
            h = sum(abs(u));
        case inf
            h = max(abs(u));
    end
end

% Overloading the mnorm of the function
function h = mnorm(u,p)
    nsq = length(u.val);
    n = sqrt(nsq);
```

```
    if mod(n,1) > 0
        n = nsq;
    end
    S.type = '()';
    D.type = '()';
    uabs = abs(u);
    h = double(zeros(n,1), zeros(n,length(getDer(u))));
    switch p
        case 1
            j = 1;
            for i = 1:nsq
                S.subs = {i};
                D.subs = {j};
                h = subsasgn(h, S, subsref(h,D) + subsref(uabs, S));
                if mod(1,n) == 0
                    j = j + 1;
                end
            end
            h = max(h);

        case inf
            j = 1;
            for i  = 1:nsq;
                S.subs = {i};
                D.subs = {j};
                h = subsasgn(h,S, subsref(h,D) + subsref(uabs, S));
                if j < n
                    j = j + 1;
                else
                    j = 1;
                end
            end
    end
end
% overloads indexed reference
function h = subsref (u,S)
    SD.type = S.type;
    SD.subs = {S.subs{1} , ':'}; % get row number 1
    h = valder(subsref(u.val,S), subsref(u.der,SD));
end
% overloads indexed assignment
function obj = subsasgn (obj ,S,u)
    SD. type = S.type ;
    SD. subs = {S.subs{1} ,':'}; % get row number i
    A = subsasgn (obj.val , S, u.val);
    B = subsasgn (obj.der , SD , u.der );
    obj = valder(A,B);
end

% Overloading the sum function for the object
function h = sum(u)
    s.type = '()';
    s.subs = {1};
    h = subsref(u,s);
    if length(getVal(u)) > 1
        for i = 2:length(getVal(u))
            s.subs = {i};
```

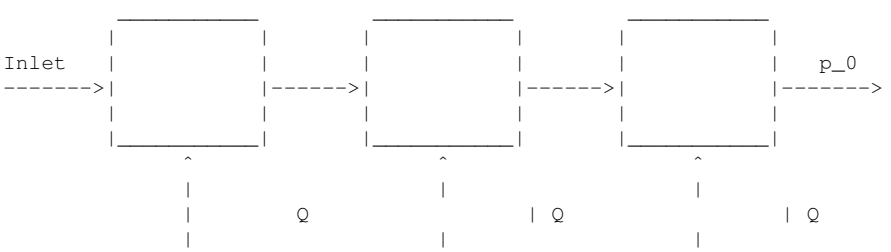
```
                    h = h + subsref(u,s);
                end
            end
        end

    end

    methods ( Static )
        % Returns sign of the first nonzero element in x
        function s = fsign (x)
            i = 1;
            while x(i) == 0 && i< length (x)
                i = i+1;
            end
            s = sign (x(i));
        end
    end
end
```

D.2 Shutdown of one-sided Heat Exchanger

This section includes all the MATLAB®-files needed to produce the results presented in Section 6.2.1. Only small adjustments is needed to produce the results in 6.2. The part which decreases the inlet flow rate and heat removal in HEX_implicit.m must be replaced by Algorithm 4.

D.2.1 Documentation of MATLAB®-files

```
************************************************************************
@Name of Folder: Shutdown
@author: Marius Reed
@organization: Process Systems Engineering, NTNU
@project: Master Thesis 2018
@since: June 2018
@requires: MATLAB R2017b (not tested in other releases)
@description: The folder cotains the files that are neceassary for
              simulating a single-sided HEX. As it is, the main.m
              file will simulate a shutdown of  a HEX with water and
              methanol The parameters can be altered to include other
              components and change the inlet conditions and more.
              This can be changed in the parameters functions. The
              results from these simulations are presented and discus-
              sed in the master thesis.
@estimated run time: 3 hours
************************************************************************
Sketch of one side of a heat exchanger
```




```
Variable list
Var1: M_1    Molar hold up of component 1 (Methanol)     [kmol]
Var2: M_2    Molar hold up of component 1 (Water)        [kmol]
Var3 : U      Internal Energy                            [MJ]
Var4 : F_V    Vapor outlet flow                          [kmol/s]
Var5 : F_L    Liquid outlet flow                         [kmol/s]
Var6 : M_V    Molar vapor hold-up                        [kmol]
Var7 : P      Pressure                                   [MPa]
Var8 : T      Temperature                                [K]
Var9 : V_L    Liquid volume                              [m^3]
Var10: x_1    Liquid fraction of component 1 (Methanol)  [-]
Var11: x_2    Liquid fraction of component 2 (Water)     [-]
Var12: y_1    Vapor fraction of component 1 (Methanol)   [-]
Var13: y_2    Vapor fraction of component 1 (Water)      [-]
Var14: H      Enthalpy                                   [MJ]
Var15: h_L    Liquid molar enthalpy                      [MJ/kmol]
```

```
Var16: h_V    Vapor molar enthalpy                          [MJ/kmol]
Var17: V_V    Vapor volume                                  [m^3]
Var18: rho_L  Liquid molar density                          [Mmol/m^3]
Var19: M_L    Liquid molar hold-up                          [kmol]
Var20: K_1    Equilibrium coeffiecient for component 1      [-]
Var21: K_2    Equilibrium coeffiecient for component 2      [-]
Var22: Ps_1   Saturation pressure for component 1           [MPa]
Var23: Ps_2   Saturation pressure for component 2           [MPa]

Component list:
1: Methanol       CH3OH
2: Water          H2O

**********************************************************************
Brief description of each files in the folder. [i] behind an input or
output denotes the dimension of the variable. If nothing is stated,
the variable is a scalar or a struct.
----------------------------------------------------------------------
main.m
input:
output:          Saves the results into HEX.mat file. In additi-
                 on it produces some plots from the results.
description:     The main.m script do necessary operations to find the
                 inital condition for the simulation of a HEX In addit-
                 ion it simulates the HEX and saves the results into a
                 .mat file and produces plots.
----------------------------------------------------------------------
Flash.m
input:           t = time, w = system variables[18], data = paramters
output:          F = function residuals[18], G = Generalized derivative
                 matrix[18x18]
description:     The flash.m function contains the model of a two
                 component flash. It includes 2 differential equations
                 (internal energy U, and component hold-up M_1). In ad-
                 dition, there are 16 algebraic equations.
----------------------------------------------------------------------
HEX.m
input:           t = time, w = system variables[18*M],data = parameters
output:          F = residuals[18*M], G = Generalized Derivative Matrix
                 [18*Mx18*M], Q = Heat transfer
description:     The HEX.m function contain the model of one side of a
                 two component HEX with given heat transfer.
----------------------------------------------------------------------
HEX_implicit.m
input:           t = time, w = system variables[2*M*18],
                 data = parameters, dynVar = dynamic variables values
                 at previous time step [4*M], dt = Time Step
output:          F = residuals [M*18], G = Generalized Derivative
                 Matrix[M*18xM*18]
description:     The HEX_implicit.m function contains the model of one
                 side of a heat exchanger where the differential equat-
                 ions are discretized.
----------------------------------------------------------------------
implicitSolverFull.m
input:           w0 = initial conditions[M*18], data = parameters, dt
                 = time step, tspan = [t_start, t_end]
output:          t_vec = time vector [(t_start-t_end)/dt + 1] ,
```

```
                w = System variables matrix[M*18x(t_start-t_end)/dt+1],
                Q = heat transfer [(t_start-t_end)/dt + 1],
description:    Simulates the HEX_implicit.m model using a implicit
                euler integrator with constant time step dt.
-------------------------------------------------------------------------
initialGuesses_gases.m
input:          par = parameters
output:         w0 = Initial guesses [18], lb = lower boundaries [18],
                ub = upper boundaries [18]
description:    Returns inital guesses and boundaries for a Flash tank
                given the parameters and that the flash is in the
                vapor-only regime.
-------------------------------------------------------------------------
parameters.m
input:
output:         par = parameters
description:    Returns the parameters for a one side of a HEX with
                methanol and water.
-------------------------------------------------------------------------
plotShutDown.m
In this function the dimension of the variables is equal to the once
produced by implicitSolverFull.m
input:          t = time, w = system variables, Q = heat transfer,
description:    Produces plots from the result obtained from the dyna-
                mic simulation of one side of a heat exchanger.
                The plots are the same as those included in the master
                thesis
-------------------------------------------------------------------------
valder.m
description:    The valder.m file contains the MATLAB class valder wh-
                ich is a user defined MATLAB object. These object cal-
                culates the function values as well as computing the
                generalized derivative using vector forward automatic
                differentiations. The class includes computation of
                generalized derivative information at nonsmooth points
                for piecewise continues functions such as abs, min,
                max and mid. This is done through the lexicographic
                directional derivative.
-------------------------------------------------------------------------
ShutDown.mat
description:    This .mat file contains the results from running the
                main.m file.
*************************************************************************
```

D.2.2 main.m

```
%*************************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @project: Master Thesis 2018
% @since: June 2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Main script for simulating shutdown of a two component HEX
% @estimated run time: 3 hours
```

```
%*************************************************************************
% Description of files
% @ Flash(t,w,data): Two component flash tank model
% @ HEX(t,w,data): Two component HEX model
% @ HEX_implicit(t,w,data): Two component HEX model with discretized diff-
%                           erential equaionts
% @ implicitSolver_Full(w,data,dt,tspan): Euler integrator for simulating
%                                         the two component HEX
% @ initialGuesses_gases(par): Initial guesses and boundaries for two
%                              component HEX in vapor-only region
% @ parmeters: Parameter values two component HEX with water and
%              methanol
% @ plotHEX_shutDown(t,w,Q): Function for plotting result from simulation
%                            of two component HEX
% @ valder: Matlab class for calculating the function residuals and
%           Generalized derivatives.
%*************************************************************************
clear all
clc
tic
%% Variable list
% Var1: M_1    Molar hold up of component 1 (Methanol)     [kmol]
% Var2: M_2    Molar hold up of component 1 (Water)        [kmol]
% Var3 : U      Internal Energy                            [MJ]
% Var4 : F_V    Vapor outlet flow                          [kmol/s]
% Var5 : F_L    Liquid outlet flow                         [kmol/s]
% Var6 : M_V    Molar vapor hold-up                        [kmol]
% Var7 : P      Pressure                                   [MPa]
% Var8 : T      Temperature                                [K]
% Var9 : V_L    Liquid volume                              [m^3]
% Var10: x_1    Liquid fraction of component 1 (Methanol)  [-]
% Var11: x_2    Liquid fraction of component 2 (Water)     [-]
% Var12: y_1    Vapor fraction of component 1 (Methanol)   [-]
% Var13: y_2    Vapor fraction of component 1 (Water)      [-]
% Var14: H      Enthalpy                                   [MJ]
% Var15: h_L    Liquid molar enthalpy                      [MJ/kmol]
% Var16: h_V    Vapor molar enthalpy                       [MJ/kmol]
% Var17: V_V    Vapor volume                               [m^3]
% Var18: rho_L  Liquid molar density                       [Mmol/m^3]
% Var19: M_L    Liquid molar hold-up                       [kmol]
% Var20: K_1    Equilibrium coeffiecient for component 1   [-]
% Var21: K_2    Equilibrium coeffiecient for component 2   [-]
% Var22: Ps_1   Saturation pressure for component 1        [MPa]
% Var23: Ps_2   Saturation pressure for component 2        [MPa]

% Component list:
% 1: Methanol       CH3OH
% 2: Water          H2O

%% Parameters and initial guesses
data.par = parameters();
[w0,lb,ub] = initialGuesses_gases(data.par);

%% Solver settings
options_lsq = optimoptions(@lsqnonlin,'Display','iter',...
     'MaxIterations',500,'MaxFunEvals',1e10,...
     'stepTolerance',1e-20,'FunctionTolerance',1e-20,...
```

```
     'specifyObjectiveGradient', true,...
     'OptimalityTolerance',1e-20);
options_fsolve = optimoptions(@fsolve,'Display','iter',...
     'MaxIterations',3e3,'MaxFunEvals',1e10,...
     'stepTolerance',1e-13,'FunctionTolerance',1e-6,...
     'specifyObjectiveGradient', true,...
     'OptimalityTolerance',1e-13, 'Algorithm','levenberg-marquardt');
%% Solving a two component flash tank with no heat transfer
data.par.Q = 0;
data.par.V_T = data.par.V_T/data.par.M; % Volume of Flash tank is equal to
                                        % 1/M of the HEX
lb(17) = data.par.V_T;                  % Lower boundary on volume
ub(17) = data.par.V_T;                  % Upper boundary on volume
w0 = lsqnonlin(@(w) Flash(1,w,data), w0,lb,ub,options_lsq);
%% Full heat exchanger steady state
data.par = parameters();
w_initial = [];
% Repeating the system variables M times
for i = 1:23
    w_initial = [w_initial;repmat(w0(i),data.par.M,1)];
end
%% Increasing heat removal until the desired initial value
Q = 0:-0.01:-2;
data.par.Q = 0;
w_initial(:,1) = fsolve(@(w)HEX(0,w,data)...
        ,w_initial,options_fsolve);
for i = 2:length(Q)
    data.par.Q = Q(i);
    w_initial(:,i) = fsolve(@(w) HEX(0,w,data)...
        ,w_initial(:,i-1),options_fsolve);
end
%% Simulating the HEX model
[t,w,Q,F_in] = implicitSolverFull(w_initial(:,end),data,0.01,[0 100]);
save('HEX_Shutdown.mat','t','w','Q','F_in'); % Saving results into .mat
% file
plotHEX_ShutDown(t,w,Q,F_in)
```

D.2.3 Flash.m

```
%*********************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Two component Flash tank model
% @input: t = time, w = system variables data = parameters
% @output: F = residuals, G = Generalized Derivative Matrix Q = Heat
% transfer
%*********************************************************************

function [F,G,Q] = Flash(t,w,data)

%% Unpacking parameters
par = data.par;
F_in = par.F_in;            % Inlet flow rate [kmol/s]
```

```
Q_0 = par.Q;                % Heat transfer [MW]
Q_max = par.Q_max;          % Maximum Heat transfer [MW]
R = par.R;                  % Gas constant [MJ/kmol*kK]
V_T = par.V_T;              % Total volume [m^3]
P_0 = par.P_0;              % Outlet pressure [MPa]
Cp_V_1 = par.Cp_V(1);       % Vapor Heat Capcity comp 1 [MJ/kmol*kK]
Cp_L_1 =  par.Cp_L(1);      % Liquid Heat Capcity comp 1 [MJ/kmol*kK]
Cp_V_2 = par.Cp_V(2);       % Vapor Heat Capcity comp 2 [MJ/kmol*kK]
Cp_L_2 =  par.Cp_L(2);      % Liquid Heat Capcity comp 2 [MJ/kmol*kK]
h_vap_1 = par.h_vap(1);     % Heat of vaporization comp 1 [MJ/kmol]
h_vap_2 = par.h_vap(2);     % Heat of vaporization comp 2 [MJ/kmol]
A_1 = par.Antoine(1,1);     % Antoine Parameter comp 1
B_1 = par.Antoine(2,1);     % Antoine Parameter comp 1
C_1 = par.Antoine(3,1);     % Antoine Parameter comp 1
A_2 = par.Antoine(1,2);     % Antoine Parameter comp 2
B_2 = par.Antoine(2,2);     % Antoine Parameter comp 2
C_2 = par.Antoine(3,2);     % Antoine Parameter comp 2
rho_1 = par.rho(1);         % Molar density comp 1 [Mmol/m^3]
rho_2 = par.rho(1);         % Molar density comp 2 [Mmol/m^3]
z_1 = par.z(1);             % Inlet composition comp 1
z_2 = par.z(2);             % Inlet composition comp 1
h_in = par.h_in;            % Inlet molar enthalpy [MJ/kmol]
C0 = par.C0;                % Compressability factor [1/MPa]
T_ref = par.T_ref;          % Reference temperature [kK]
c_V = par.c_V;              % Vapor valve coefficient [kmol/bar^0.5*s]
c_L = par.c_L;              % Liquid valve coefficient [kmol/bar^0.5*s]

% Defining the changes in heat transfer
if t < 5
    Q = Q_0;
elseif t < 100
    Q = Q_0 + (Q_max-Q_0)/(100-5)*(t-5);
else
    Q = Q_max;
end
%% Unpacking system variables
w = valder(w,eye(length(w)));
M_1 = w(1);         % Component 1 Molar hold-up [kmol]
M_2 = w(2);         % Component 2 Molar hold-up [kmol]
U = w(3);           % Internal Energy [MJ]
F_V = w(4);         % Vapor flow rate [kmol/s]
F_L = w(5);         % Liquid flow rate [kmol/s]
M_V = w(6);         % Vapor molar hold-up [kmol]
P = w(7);           % Pressure [MPa]
T = w(8);           % Temperature [kK]
V_L = w(9);         % Liquid Volume [m^3]
x_1 = w(10);        % Liquid mole fraction component 1
x_2 = w(11);        % Liquid mole fraction component 2
y_1 = w(12);        % Vapor mole fraction component 1
y_2 = w(13);        % Vapor mole fraction component 2
H = w(14);          % Enthalpy [MJ]
h_L = w(15);        % Molar enthalpy in liquid phase [MJ/kmol]
h_V = w(16);        % Molar enthalpy in vapor phase [MJ/kmol]
V_V = w(17);        % Vapor volume [m^3]
rho_L = w(18);      % Liquid molar density [Mmol/kK]
M_L = w(19);        % Liquid molar hold-up [kmol]
k_1 = w(20);        % Vapor-liquid equilibrium coefficent component 1
```

```
k_2 = w(21);        % Vapor-liquid equilibrium coefficent component 2
Ps_1 = w(22);       % Saturation pressure component 1
Ps_2 = w(23);       % Saturation pressure component 2


%% Differential equations
f1 = F_in*z_1 - F_L*x_1 - F_V * y_1;
f2 = F_in*z_2 - F_L*x_2 - F_V * y_2;
f3 = F_in*h_in - F_L*h_L - F_V*h_V + Q;

%% Algebraic equations
f4 = M_1 - (M_L*x_1 + M_V*y_1);
f5 = M_2 - (M_L*x_2 + M_V*y_2);
f6 = (M_1+M_2) - (M_L + M_V);
f7 = H - (M_L*h_L + M_V*h_V);
f8 = H - (U + P*V_T);
f9 = y_1 - k_1*x_1;
f10 = y_2 - k_2*x_2;
f11 = midobj(M_V/(M_V+M_L),(x_1+x_2) - (y_1+y_2), (M_V/(M_V+M_L))-1);
f12 = V_T - (V_L + V_V);
f13 = P*V_V - M_V*R*T;
f14 = V_L - M_L/(rho_L*1e3);
f15 = F_V*V_T - c_V*V_V*max2(0,(P-P_0)/sqrt(abs((P-P_0))+1e-10));
f16 = F_L*V_T - c_L*V_L*max2(0,(P-P_0)/sqrt(abs((P-P_0))+1e-10));
f17 = 1/rho_L - (x_1/(rho_1*(1+C0*(P-0.1))) + x_2/(rho_2*(1+C0*(P-0.1))));
f18 = h_L - ((x_1*Cp_L_1)*(T-T_ref) + (x_2*Cp_L_2)*(T-T_ref));
f19 = h_V - (y_1*(h_vap_1 + Cp_V_1*(T-T_ref)) + y_2*(h_vap_2 +...
    Cp_V_2*(T-T_ref)));
f20 = k_1*P - Ps_1;
f21 = k_2*P - Ps_2;
f22 = Ps_1 - 10^(A_1 - B_1/(T*1e3+C_1))/10;
f23 = Ps_2 - 10^(A_2 - B_2/(T*1e3+C_2))/10;

%% Extracting the values and the derivatives from the valder objects
F = [getVal(f1);getVal(f2);getVal(f3);getVal(f4);getVal(f5);getVal(f6);...
    getVal(f7);getVal(f8);getVal(f9);getVal(f10);getVal(f11);...
    getVal(f12);getVal(f13);getVal(f14);getVal(f15);getVal(f16);...
    getVal(f17);getVal(f18);getVal(f19);getVal(f20);getVal(f21);...
    getVal(f22);getVal(f23)];

G = [getDer(f1);getDer(f2);getDer(f3);getDer(f4);getDer(f5);getDer(f6);...
    getDer(f7);getDer(f8);getDer(f9);getDer(f10);getDer(f11);...
    getDer(f12);getDer(f13);getDer(f14);getDer(f15);getDer(f16);...
    getDer(f17);getDer(f18);getDer(f19);getDer(f20);getDer(f21);...
    getDer(f22);getDer(f23)];

end
```

D.2.4 HEX.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
```

```
% @description: Two component HEX model
% @input: t = time, w = system variables data = parameters
% @output: F = residuals, G = Generalized Derivative Matrix Q = Heat
% transfer, F_in = inlet flow rate
%*****************************************************************

function [F,G,Q,F_in] = HEX(t,w,data)

%% Unpacking parameters
par = data.par;
F_in = par.F_in;            % Inlet flow rate [kmol/s]
Q_0 = par.Q;                % Heat transfer [MW]
Q_max = par.Q_max;          % Maximum Heat transfer [MW]
R = par.R;                  % Gas constant [MJ/kmol*kK]
V_T = par.V_T;              % Total volume [m^3]
P_0 = par.P_0;              % Outlet pressure [MPa]
Cp_V_1 = par.Cp_V(1);       % Vapor Heat Capcity comp 1 [MJ/kmol*kK]
Cp_L_1 =  par.Cp_L(1);      % Liquid Heat Capcity comp 1 [MJ/kmol*kK]
Cp_V_2 = par.Cp_V(2);       % Vapor Heat Capcity comp 2 [MJ/kmol*kK]
Cp_L_2 =  par.Cp_L(2);      % Liquid Heat Capcity comp 2 [MJ/kmol*kK]
h_vap_1 = par.h_vap(1);     % Heat of vaporization comp 1 [MJ/kmol]
h_vap_2 = par.h_vap(2);     % Heat of vaporization comp 2 [MJ/kmol]
A_1 = par.Antoine(1,1);     % Antoine Parameter comp 1
B_1 = par.Antoine(2,1);     % Antoine Parameter comp 1
C_1 = par.Antoine(3,1);     % Antoine Parameter comp 1
A_2 = par.Antoine(1,2);     % Antoine Parameter comp 2
B_2 = par.Antoine(2,2);     % Antoine Parameter comp 2
C_2 = par.Antoine(3,2);     % Antoine Parameter comp 2
rho_1 = par.rho(1);         % Molar density comp 1 [Mmol/m^3]
rho_2 = par.rho(1);         % Molar density comp 2 [Mmol/m^3]
z_1 = par.z(1);             % Inlet composition comp 1
z_2 = par.z(2);             % Inlet composition comp 1
h_in = par.h_in;            % Inlet molar enthalpy [MJ/kmol]
C0 = par.C0;                % Compressability factor [1/MPa]
T_ref = par.T_ref;          % Reference temperature [kK]
c_V = par.c_V;              % Vapor valve coefficient [kmol/bar^0.5*s]
c_L = par.c_L;              % Liquid valve coefficient [kmol/bar^0.5*s]
M = par.M;                  % Number of flash tanks in series
% Adjusting the valve coeffcient to M
c_V = sqrt(M)*c_V;
c_L = sqrt(M)*c_L;

% Defining the changes in heat transfer
if t < 5
    Q = Q_0;
elseif t < 70
    Q = Q_0 - Q_0/(70-5)*(t-5);
else
    Q = 0;
end
if t < 5
    F_in = F_in;
elseif t < 50
    F_in = F_in - F_in/(50-5)*(t-5);
else
    F_in = 0;
end
```

```
%% Unpacking system variables
L = length(w);
w = valder(w,eye(length(w)));
M_1 = w(1:M);             % Component 1 Molar hold-up [kmol]
M_2 = w(1*M+1:2*M);       % Component 2 Molar hold-up [kmol]
U = w(2*M+1:3*M);         % Internal Energy [MJ]
F_V = w(3*M+1:4*M);       % Vapor flow rate [kmol/s]
F_L = w(4*M+1:5*M);       % Liquid flow rate [kmol/s]
M_V = w(5*M+1:6*M);       % Vapor molar hold-up [kmol]
P = w(6*M+1:7*M);         % Pressure [MPa]
T = w(7*M+1:8*M);         % Temperature [kK]
V_L = w(8*M+1:9*M);       % Liquid Volume [m^3]
x_1 = w(9*M+1:10*M);      % Liquid mole fraction component 1
x_2 = w(10*M+1:11*M);     % Liquid mole fraction component 2
y_1 = w(11*M+1:12*M);     % Vapor mole fraction component 1
y_2 = w(12*M+1:13*M);     % Vapor mole fraction component 2
H = w(13*M+1:14*M);       % Enthalpy [MJ]
h_L = w(14*M+1:15*M);     % Molar enthalpy in liquid phase [MJ/kmol]
h_V = w(15*M+1:16*M);     % Molar enthalpy in vapor phase [MJ/kmol]
V_V = w(16*M+1:17*M);     % Vapor volume [m^3]
rho_L = w(17*M+1:18*M);   % Liquid molar density [Mmol/kK]
M_L = w(18*M+1:19*M);     % Liquid molar hold-up [kmol]
k_1 = w(19*M+1:20*M);     % Vapor-liquid equilibrium coefficent component 1
k_2 = w(20*M+1:21*M);     % Vapor-liquid equilibrium coefficent component 2
Ps_1 = w(21*M+1:22*M);    % Saturation pressure component 1
Ps_2 = w(22*M+1:23*M);    % Saturation pressure component 2

%% Preallocating memory
f1 = valder(zeros(M,1),zeros(M,L));
f2 = valder(zeros(M,1),zeros(M,L));
f3 = valder(zeros(M,1),zeros(M,L));
f4 = valder(zeros(M,1),zeros(M,L));
f5 = valder(zeros(M,1),zeros(M,L));
f6 = valder(zeros(M,1),zeros(M,L));
f7 = valder(zeros(M,1),zeros(M,L));
f8 = valder(zeros(M,1),zeros(M,L));
f9 = valder(zeros(M,1),zeros(M,L));
f10 = valder(zeros(M,1),zeros(M,L));
f11 = valder(zeros(M,1),zeros(M,L));
f12 = valder(zeros(M,1),zeros(M,L));
f13 = valder(zeros(M,1),zeros(M,L));
f14 = valder(zeros(M,1),zeros(M,L));
f15 = valder(zeros(M,1),zeros(M,L));
f16 = valder(zeros(M,1),zeros(M,L));
f17 = valder(zeros(M,1),zeros(M,L));
f18 = valder(zeros(M,1),zeros(M,L));
f19 = valder(zeros(M,1),zeros(M,L));
f20 = valder(zeros(M,1),zeros(M,L));
f21 = valder(zeros(M,1),zeros(M,L));
f22 = valder(zeros(M,1),zeros(M,L));
f23 = valder(zeros(M,1),zeros(M,L));
%% Differential equations
f1(1) = F_in*z_1 - (F_L(1)*x_1(1) + F_V(1)*y_1(1));
f2(1) = F_in*z_2 - (F_L(1)*x_2(1) +  F_V(1)*y_2(1));
f3(1) = F_in*h_in - (F_L(1)*h_L(1) + F_V(1)*h_V(1)) + Q/M;

for j = 2:M-1
```

```
    f1(j) = (F_L(j-1)*x_1(j-1) + F_V(j-1)*y_1(j-1))...
        - (F_L(j)*x_1(j) + F_V(j)*y_1(j));
    f2(j) = (F_L(j-1)*x_2(j-1) + F_V(j-1)*y_2(j-1))...
        - (F_L(j)*x_2(j) + F_V(j)*y_2(j));
    f3(j) = (F_L(j-1)*h_L(j-1) + F_V(j-1)*h_V(j-1))...
        - (F_L(j)*h_L(j) + F_V(j)*h_V(j)) + Q/M;
end

f1(M) = (F_L(M-1)*x_1(M-1) + F_V(M-1)*y_1(M-1))...
    - (F_L(M)*x_1(M) + F_V(M)*y_1(M));
f2(M) = (F_L(M-1)*x_2(M-1) + F_V(M-1)*y_2(M-1))...
    - (F_L(M)*x_2(M) + F_V(M)*y_2(M));
f3(M) = (F_L(M-1)*h_L(M-1) + F_V(M-1)*h_V(M-1))...
    - (F_L(M)*h_L(M) + F_V(M)*h_V(M)) + Q/M;
%% Algebraic equations
for j = 1:M
    f4(j) = M_1(j) - (M_L(j)*x_1(j) + M_V(j)*y_1(j));
    f5(j) = M_2(j) - (M_L(j)*x_2(j) + M_V(j)*y_2(j));
    f6(j) = (M_1(j)+M_2(j)) - (M_L(j) + M_V(j));
    f7(j) = H(j) - (M_L(j)*h_L(j) + M_V(j)*h_V(j));
    f8(j) = H(j) - (U(j) + P(j)*V_T/M);
    f9(j) = y_1(j) - k_1(j)*x_1(j);
    f10(j) = y_2(j) - k_2(j)*x_2(j);
    f11(j) = midobj(M_V(j)/(M_V(j)+M_L(j)),...
        ((x_1(j)+x_2(j)) - (y_1(j)+y_2(j))), (M_V(j)/(M_V(j)+M_L(j)))-1);
    f12(j) = V_T/M - (V_L(j) + V_V(j));
    f13(j) = P(j)*V_V(j) - M_V(j)*R*T(j);
    f14(j) = V_L(j) - M_L(j)/(rho_L(j)*1e3);
    if j < M
        f15(j) = F_V(j)*V_T/M - c_V*V_V(j)*((P(j)-P(j+1))...
            /sqrt(abs((P(j)-P(j+1)))+1e-10));
        f16(j) = F_L(j)*V_T/M - c_L*V_L(j)*((P(j)-P(j+1))...
            /sqrt(abs((P(j)-P(j+1)))+1e-10));
    else
        f15(j) = F_V(j)*V_T/M - c_V*V_V(j)*((P(j)-P_0)...
            /sqrt(abs((P(j)-P_0))+1e-10));
        f16(j) = F_L(j)*V_T/M - c_L*V_L(j)*(((P(j)-P_0)...
            /sqrt(abs((P(j)-P_0))+1e-10)));
    end
    f17(j) = 1/rho_L(j) - (x_1(j)/(rho_1*(1+C0*(P(j)-0.1)))...
        +x_2(j)/(rho_2*(1+C0*(P(j)-0.1))));
    f18(j) = h_L(j) - (x_1(j)*Cp_L_1*(T(j)-T_ref)...
        + x_2(j)*Cp_L_2*(T(j)-T_ref));
    f19(j) = h_V(j) - (y_1(j)*(h_vap_1 + Cp_V_1*(T(j)-T_ref))...
        +y_2(j)*(h_vap_2 + Cp_V_2*(T(j)-T_ref)));
    f20(j) = k_1(j)*P(j) - Ps_1(j);
    f21(j) = k_2(j)*P(j) - Ps_2(j);
    f22(j) = Ps_1(j) - 10^(A_1 - B_1/(T(j)*1e3+C_1))/10;
    f23(j) = Ps_2(j) - 10^(A_2 - B_2/(T(j)*1e3+C_2))/10;
%% Extracting the values and the derivatives from the valder objects
F = [getVal(f1);getVal(f2);getVal(f3);getVal(f4);getVal(f5);getVal(f6);...
    getVal(f7);getVal(f8);getVal(f9);getVal(f10);getVal(f11);...
    getVal(f12);getVal(f13);getVal(f14);getVal(f15);getVal(f16);...
    getVal(f17);getVal(f18);getVal(f19);getVal(f20);getVal(f21);...
    getVal(f22);getVal(f23)];

G = [getDer(f1);getDer(f2);getDer(f3);getDer(f4);getDer(f5);getDer(f6);...
```

```
    getDer(f7);getDer(f8);getDer(f9);getDer(f10);getDer(f11);...
    getDer(f12);getDer(f13);getDer(f14);getDer(f15);getDer(f16);...
    getDer(f17);getDer(f18);getDer(f19);getDer(f20);getDer(f21);...
    getDer(f22);getDer(f23)];
end
```

D.2.5 HEX_implicit.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Implicit two component HEX model
% @input: t = time, w = system variables data = parameters, dynVar =
% Value of differential variables at previous time, dt = time step
% @output: F = residuals, G = Generalized Derivative Matrix
%*****************************************************************

function [F,G] = HEX_implicit(t,w,data,dynVar,dt)

%% Unpacking parameters
par = data.par;
F_in = par.F_in;            % Inlet flow rate [kmol/s]
Q_0 = par.Q;                % Heat transfer [MW]
R = par.R;                  % Gas constant [MJ/kmol*kK]
V_T = par.V_T;              % Total volume [m^3]
P_0 = par.P_0;              % Outlet pressure [MPa]
Cp_V_1 = par.Cp_V(1);       % Vapor Heat Capcity comp 1 [MJ/kmol*kK]
Cp_L_1 =  par.Cp_L(1);      % Liquid Heat Capcity comp 1 [MJ/kmol*kK]
Cp_V_2 = par.Cp_V(2);       % Vapor Heat Capcity comp 2 [MJ/kmol*kK]
Cp_L_2 =  par.Cp_L(2);      % Liquid Heat Capcity comp 2 [MJ/kmol*kK]
h_vap_1 = par.h_vap(1);     % Heat of vaporization comp 1 [MJ/kmol]
h_vap_2 = par.h_vap(2);     % Heat of vaporization comp 2 [MJ/kmol]
A_1 = par.Antoine(1,1);     % Antoine Parameter comp 1
B_1 = par.Antoine(2,1);     % Antoine Parameter comp 1
C_1 = par.Antoine(3,1);     % Antoine Parameter comp 1
A_2 = par.Antoine(1,2);     % Antoine Parameter comp 2
B_2 = par.Antoine(2,2);     % Antoine Parameter comp 2
C_2 = par.Antoine(3,2);     % Antoine Parameter comp 2
rho_1 = par.rho(1);         % Molar density comp 1 [Mmol/m^3]
rho_2 = par.rho(1);         % Molar density comp 2 [Mmol/m^3]
z_1 = par.z(1);             % Inlet composition comp 1
z_2 = par.z(2);             % Inlet composition comp 1
h_in = par.h_in;            % Inlet molar enthalpy [MJ/kmol]
C0 = par.C0;                % Compressability factor [1/MPa]
T_ref = par.T_ref;          % Reference temperature [kK]
c_V = par.c_V;              % Vapor valve coefficient [kmol/bar^0.5*s]
c_L = par.c_L;              % Liquid valve coefficient [kmol/bar^0.5*s]
M = par.M;                  % Number of flash tanks in series
% Adjusting the valve coeffcient to M
c_V = sqrt(M)*c_V;
c_L = sqrt(M)*c_L;

% Values of the differential variables at previous time.
```

```
M_1_0 = dynVar(1:M);
M_2_0 = dynVar(1*M+1:2*M);
U_0 = dynVar(2*M+1:3*M);

% Defining the changes in heat transfer
if t < 5
    Q = Q_0;
elseif t < 70
    Q = Q_0 - Q_0/(70-5)*(t-5);
else
    Q = 0;
end

if t < 5
    F_in = F_in;
elseif t < 50
    F_in = F_in - F_in/(50-5)*(t-5);
else
    F_in = 0;
end
%% Unpacking system variables
L = length(w);
w = valder(w,eye(length(w)));
M_1 = w(1:M);             % Component 1 Molar hold-up [kmol]
M_2 = w(1*M+1:2*M);       % Component 2 Molar hold-up [kmol]
U = w(2*M+1:3*M);         % Internal Energy [MJ]
F_V = w(3*M+1:4*M);       % Vapor flow rate [kmol/s]
F_L = w(4*M+1:5*M);       % Liquid flow rate [kmol/s]
M_V = w(5*M+1:6*M);       % Vapor molar hold-up [kmol]
P = w(6*M+1:7*M);         % Pressure [MPa]
T = w(7*M+1:8*M);         % Temperature [kK]
V_L = w(8*M+1:9*M);       % Liquid Volume [m^3]
x_1 = w(9*M+1:10*M);      % Liquid mole fraction component 1
x_2 = w(10*M+1:11*M);     % Liquid mole fraction component 2
y_1 = w(11*M+1:12*M);     % Vapor mole fraction component 1
y_2 = w(12*M+1:13*M);     % Vapor mole fraction component 2
H = w(13*M+1:14*M);       % Enthalpy [MJ]
h_L = w(14*M+1:15*M);     % Molar enthalpy in liquid phase [MJ/kmol]
h_V = w(15*M+1:16*M);     % Molar enthalpy in vapor phase [MJ/kmol]
V_V = w(16*M+1:17*M);     % Vapor volume [m^3]
rho_L = w(17*M+1:18*M);   % Liquid molar density [Mmol/kK]
M_L = w(18*M+1:19*M);     % Liquid molar hold-up [kmol]
k_1 = w(19*M+1:20*M);     % Vapor-liquid equilibrium coefficent component 1
k_2 = w(20*M+1:21*M);     % Vapor-liquid equilibrium coefficent component 2
Ps_1 = w(21*M+1:22*M);    % Saturation pressure component 1
Ps_2 = w(22*M+1:23*M);    % Saturation pressure component 2

%% Preallocating memory
f1 = valder(zeros(M,1),zeros(M,L));
f2 = valder(zeros(M,1),zeros(M,L));
f3 = valder(zeros(M,1),zeros(M,L));
f4 = valder(zeros(M,1),zeros(M,L));
f5 = valder(zeros(M,1),zeros(M,L));
f6 = valder(zeros(M,1),zeros(M,L));
f7 = valder(zeros(M,1),zeros(M,L));
f8 = valder(zeros(M,1),zeros(M,L));
f9 = valder(zeros(M,1),zeros(M,L));
```

```
f10 = valder(zeros(M,1),zeros(M,L));
f11 = valder(zeros(M,1),zeros(M,L));
f12 = valder(zeros(M,1),zeros(M,L));
f13 = valder(zeros(M,1),zeros(M,L));
f14 = valder(zeros(M,1),zeros(M,L));
f15 = valder(zeros(M,1),zeros(M,L));
f16 = valder(zeros(M,1),zeros(M,L));
f17 = valder(zeros(M,1),zeros(M,L));
f18 = valder(zeros(M,1),zeros(M,L));
f19 = valder(zeros(M,1),zeros(M,L));
f20 = valder(zeros(M,1),zeros(M,L));
f21 = valder(zeros(M,1),zeros(M,L));
f22 = valder(zeros(M,1),zeros(M,L));
f23 = valder(zeros(M,1),zeros(M,L));

%% Differential equations

f1(1) = M_1(1)  - (M_1_0(1)  + (F_in*z_1 - (max2(0,F_L(1))*x_1(1)...
    +  max2(0,F_V(1))*y_1(1)) - (min2(0,F_L(1))*x_1(2) +...
    min2(0,F_V(1))*y_1(2)))*dt);
f2(1) = M_2(1)  - (M_2_0(1)  + (F_in*z_2 - (max2(0,F_L(1))*x_2(1)...
    +  max2(0,F_V(1))*y_2(1)) - (min2(0,F_L(1))*x_2(2)...
    +  min2(0,F_V(1))*y_2(2)))*dt);
f3(1) = U(1) - (U_0(1) + (F_in*h_in - (max2(0,F_L(1))*h_L(1)...
    + max2(0,F_V(1))*h_V(1)) - (min2(0,F_L(1))*h_L(2)...
    + min2(0,F_V(1))*h_V(2)) + Q/M)*dt);

for j = 2:M-1
    f1(j) = M_1(j) - (M_1_0(j) + ((max2(0,F_L(j-1))*x_1(j-1)...
        + max2(0,F_V(j-1))*y_1(j-1)) - (max2(0,F_L(j))*x_1(j)...
        + max2(0,F_V(j))*y_1(j)) + (min2(0,F_L(j-1))*x_1(j)...
        + min2(0,F_V(j-1))*y_1(j)) - (min2(0,F_L(j))*x_1(j+1)...
        + min2(0,F_V(j))*y_1(j+1)))*dt);
    f2(j) = M_2(j) - (M_2_0(j) + ((max2(0,F_L(j-1))*x_2(j-1)...
        + max2(0,F_V(j-1))*y_2(j-1)) - (max2(0,F_L(j))*x_2(j)...
        + max2(0,F_V(j))*y_2(j)) + (min2(0,F_L(j-1))*x_2(j)...
        +  min2(0,F_V(j-1))*y_2(j)) - (min2(0,F_L(j))*x_2(j+1)...
        + min2(0,F_V(j))*y_2(j+1)))*dt);
    f3(j) = U(j) - (U_0(j) + ((max2(0,F_L(j-1))*h_L(j-1)...
        + max2(0,F_V(j-1))*h_V(j-1)) - (max2(0,F_L(j))*h_L(j)...
        + max2(0,F_V(j))*h_V(j)) + (min2(0,F_L(j-1))*h_L(j)...
        + min2(0,F_V(j-1))*h_V(j)) - (min2(0,F_L(j))*h_L(j+1)...
        + min2(0,F_V(j))*h_V(j+1))  + Q/M)*dt);
end

f1(M) = M_1(M) - (M_1_0(M) + ((max2(0,F_L(M-1))*x_1(M-1)...
    + max2(0,F_V(M-1))*y_1(M-1)) - (max2(0,F_L(M))*x_1(M)...
    + max2(0,F_V(M))*y_1(M)) + (min2(0,F_L(M-1))*x_1(M)...
    + min2(0,F_V(M-1))*y_1(M)))*dt);
f2(M) = M_2(M) - (M_2_0(M) + ((max2(0,F_L(M-1))*x_2(M-1)...
    + max2(0,F_V(M-1))*y_2(M-1)) - (max2(0,F_L(M))*x_2(M)...
    + max2(0,F_V(M))*y_2(M)) + (min2(0,F_L(M-1))*x_2(M)...
    + min2(0,F_V(M-1))*y_2(M)))*dt);
f3(M) = U(M) - (U_0(M) + ((max2(0,F_L(M-1))*h_L(M-1)...
    + max2(0,F_V(M-1))*h_V(M-1)) - (max2(0,F_L(M))*h_L(M)...
    + max2(0,F_V(M))*h_V(M)) + (min2(0,F_L(M-1))*h_L(M)...
    + min2(0,F_V(M-1))*h_V(M))  + Q/M)*dt);
```

```
%% Algebraic equations
for j = 1:M
    f4(j) = M_1(j) - (M_L(j)*x_1(j) + M_V(j)*y_1(j));
    f5(j) = M_2(j) - (M_L(j)*x_2(j) + M_V(j)*y_2(j));
    f6(j) = (M_1(j)+M_2(j)) - (M_L(j) + M_V(j));
    f7(j) = H(j) - (M_L(j)*h_L(j) + M_V(j)*h_V(j));
    f8(j) = H(j) - (U(j) + P(j)*V_T/M);
    f9(j) = y_1(j) - k_1(j)*x_1(j);
    f10(j) = y_2(j) - k_2(j)*x_2(j);
    f11(j) = midobj(M_V(j)/(M_V(j)+M_L(j)),...
        ((x_1(j)+x_2(j)) - (y_1(j)+y_2(j))), (M_V(j)/(M_V(j)+M_L(j)))-1);
    f12(j) = V_T/M - (V_L(j) + V_V(j));
    f13(j) = P(j)*V_V(j) - M_V(j)*R*T(j);
    f14(j) = V_L(j) - M_L(j)/(rho_L(j)*1e3);
    if j < M
        f15(j) = F_V(j)*V_T/M - c_V*V_V(j)*(P(j)-P(j+1))...
            /sqrt(abs((P(j)-P(j+1)))+1e-10);
        f16(j) = F_L(j)*V_T/M - c_L*V_L(j)*(P(j)-P(j+1))...
            /sqrt(abs((P(j)-P(j+1)))+1e-10);
    else
        f15(j) = F_V(j)*V_T/M - c_V*V_V(j)*max2(0,(P(j)-P_0)...
            /sqrt(abs((P(j)-P_0))+1e-10));
        f16(j) = F_L(j)*V_T/M - c_L*V_L(j)*max2(0,((P(j)-P_0)...
            /sqrt(abs((P(j)-P_0))+1e-10)));
    end
    f17(j) = 1/rho_L(j) - (x_1(j)/(rho_1*(1+C0*(P(j)-0.1)))...
        +x_2(j)/(rho_2*(1+C0*(P(j)-0.1))));
    f18(j) = h_L(j) - (x_1(j)*Cp_L_1*(T(j)-T_ref)...
        + x_2(j)*Cp_L_2*(T(j)-T_ref));
    f19(j) = h_V(j) - (y_1(j)*(h_vap_1 + Cp_V_1*(T(j)-T_ref))...
        +y_2(j)*(h_vap_2 + Cp_V_2*(T(j)-T_ref)));
    f20(j) = k_1(j)*P(j) - Ps_1(j);
    f21(j) = k_2(j)*P(j) - Ps_2(j);
    f22(j) = Ps_1(j) - 10^(A_1 - B_1/(T(j)*1e3+C_1))/10;
    f23(j) = Ps_2(j) - 10^(A_2 - B_2/(T(j)*1e3+C_2))/10;

%% Extracting the values and the derivatives from the valder objects
F = [getVal(f1);getVal(f2);getVal(f3);getVal(f4);getVal(f5);getVal(f6);...
    getVal(f7);getVal(f8);getVal(f9);getVal(f10);getVal(f11);...
    getVal(f12);getVal(f13);getVal(f14);getVal(f15);getVal(f16);...
    getVal(f17);getVal(f18);getVal(f19);getVal(f20);getVal(f21);...
    getVal(f22);getVal(f23)];

G = [getDer(f1);getDer(f2);getDer(f3);getDer(f4);getDer(f5);getDer(f6);...
    getDer(f7);getDer(f8);getDer(f9);getDer(f10);getDer(f11);...
    getDer(f12);getDer(f13);getDer(f14);getDer(f15);getDer(f16);...
    getDer(f17);getDer(f18);getDer(f19);getDer(f20);getDer(f21);...
    getDer(f22);getDer(f23)];
end
```

D.2.6 implicitSolverFull.m

```
%*********************************************************************
% @author: Marius Reed
```

```
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Implicit Euler integrator for two component flash tank
% @input: w0 = initial conditions, data = parameters, dt = time step,
% tspan = [t_start, t_end]
% @output: t = time vector, w = System variables matrix, Q = heat transfer,
%          F_in = inlet flow rate
%*********************************************************************

function [t,w,Q,F_in] = implicitSolverFull(w0,data,dt,tspan)
% Solver settings
options_fsolve = optimoptions(@fsolve,'Display','iter',...
     'MaxIterations',3e3,'MaxFunEvals',1e10,...
     'stepTolerance',1e-20,'FunctionTolerance',1e-20,...
     'specifyObjectiveGradient', true,...
     'OptimalityTolerance',1e-13, 'Algorithm','levenberg-marquardt');
% Preallocating memory and setting the first value equal to initial values
t = tspan(1):dt:tspan(2);
w = zeros(length(w0),length(t));
Q = zeros(1,length(t));
F_in = zeros(1,length(t));
w(:,1) = w0;
Q(1) = data.par.Q;
F_in(1) = data.par.F_in;
M = data.par.M;
% Solving the flash model and doing time steps integrations
for i = 2:length(t)
    disp(t(i));
    w0 = w(:,i-1);
    dynVar = w(1:3*M,i-1);
    w(:,i) = fsolve(@(w) HEX_implicit(t(i),w,data,dynVar,dt)...
        ,w0,options_fsolve);
    [~,~,Q(i),F_in(i)] = HEX(t(i),w(:,i),data);
end
end
```

D.2.7 initialGuesses_gases.m

```
%*********************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Returns inital guesses and boundaries for a Flash tank
% given the parameters and that the flash is in the vapor-only regime
% @input: par = parameters
% @output: w0 = Initial guesses, lb = lower boundaries, ub = upper
% boundaries
%*********************************************************************

function [w0,lb,ub] = initialGuesses_gases(par)

% M_1 = w(1);           % Component 1 molar hold-up [kmol]
w0(1) = 0.5;
```

```
lb(1) = 0;
ub(1) = inf;
% M_2 = w(2);           % Component 2 molar hold-up [kmol]
w0(2) = 0.5;
lb(2) = 0;
ub(2) = inf;
% U = w(3);             % Internal energy [MJ]
w0(3) = -1;
lb(3) = -inf;
ub(3) = inf;
% F_V = w(4);           % Vapor flow rate [kmol/s]
w0(4) = par.F_in;
lb(4) = par.F_in;
ub(4) = par.F_in;
% F_L = w(5);           % Liquid flow rate [kmol/s]
w0(5) = 0;
lb(5) = 0;
ub(5) = 0;
% M_V = w(6);           % Vapor molar hold-up [kmol]
w0(6) = 0.1;
lb(6) = 0;
ub(6) = inf;
% P = w(7);             % Pressure [MPa]
w0(7) = 0.11;
lb(7) = 0;
ub(7) = inf;
% T = w(8);             % Temperature [kK]
w0(8) = par.T_in;
lb(8) = par.T_in;
ub(8) = par.T_in;
% V_L = w(9);           % Liquid volume [m^3]
w0(9) = 0;
lb(9) = 0;
ub(9) = 0;
% x_1 = w(10);          % Liquid mole fraction component 1
w0(10) = 0.5;
lb(10) = 0;
ub(10) = inf;
% x_2 = w(11);          % Liquid mole fraction component 2
w0(11) = 0.5;
lb(11) = 0;
ub(11) = inf;
% y_1 = w(12);          % Vapor mole fraction component 1
w0(12) = par.z(1);
lb(12) = par.z(1);
ub(12) = par.z(1);
% y_2 = w(13);          % Vapor mole fraction component 2
w0(13) = par.z(2);
lb(13) = par.z(2);
ub(13) = par.z(2);
% H = w(14);            % Enthalpy [MJ]
w0(14) = 1;
lb(14) = -inf;
ub(14) = inf;
% h_L = w(15);          % Liquid molar enthalpy [MJ/kmol]
w0(15) = par.h_in;
lb(15) = -inf;
```

```
ub(15) = inf;
% h_V = w(16);          % Vapor molar enthalpy [MJ/kmol]
w0(16) = par.h_in;
lb(16) = par.h_in;
ub(16) = par.h_in;
% V_V = w(17);          % Vapor volume [m^3]
w0(17) = par.V_T;
lb(17) = par.V_T;
ub(17) = par.V_T;
% rho_L = w(18);        % Liquid molar density [Mmol/m^3]
w0(18) = 0.5;
lb(18) = 0;
ub(18) = inf;
% M_L = w(19);          % Liquid molar hold-up [kmol]
w0(19) = 0;
lb(19) = 0;
ub(19) = 0;
% K_1 = w(20);          % Vapor-liquid equilibrium coefficient comp 1
w0(20) = 1;
lb(20) = 0;
ub(20) = inf;
% K_2 = w(21);          % Vapor-liquid equilibrium coefficient comp 2
w0(21) = 1;
lb(21) = 0;
ub(21) = inf;
% Ps_1 = w(22);         % Saturation Pressure component 1 [MPa]
w0(22) = 0.1;
lb(22) = 0;
ub(22) = inf;
% Ps_2 = w(23);         % Saturation Pressure component 1 [MPa]
w0(23) = 0.1;
lb(23) = 0;
ub(23) = inf;
w0 = w0';
end
```

D.2.8 parameters.m

```
%*********************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Parameters for two component HEX
% @componentList: 1: Methanol   2: Water
%*********************************************************************

function par = parameters()
%% Literature
% https://webbook.nist.gov/cgi/cbook.cgi?ID=C67561&Mask=4&Type=ANTOINE&Plot=on
% https://webbook.nist.gov/cgi/cbook.cgi?ID=C7732185&Mask=4&Type=ANTOINE&Plot=on
% http://www.personal.utulsa.edu/~geoffrey-price/Courses/ChE7023/
% HeatCapacity-HeatOfFormation.pdf
%% Parameters
% Antoine Parameters log(p_sat[bar]) = A - B/(T[K]+C)
```

```
A = [5.15853 4.6543];
B = [1569.613 1435.264];
C = [-34.846 -64.848];

par.T_in = 0.410;       % Temperature inlet [kK]
par.F_in = 0.1;         % Inlet flow rate [kmol/s]
par.Antoine = [A;B;C];  % Antoine parameters
par.Q = 0;              % Heat transfer [MW]
par.Q_max = -4;         % Maximum heat transfer [MW]
par.h_vap = [35.210 40.660]; % Heat of vaporization [MJ/kmol]
par.Cp_V = [44.06, 35]; % Vapor Heat Capacity [MJ/kmol*kK]
par.Cp_L = [81.08, 75]; % Liquid Heat Capacity [MJ/kmol*kK]
par.R = 8.314;          % Gas constant [MJ/(kK * kmol)]
par.P_0 = 0.1;          % Outlet Pressure [MPa]
par.V_T = 0.2;          % Total volume [m^3]
par.rho = [0.792/32.04, 1/18.016]; % Liquid molar density[Mmol/m^3]
par.c_V = 1;            % Valve coefficient vapor[kmol/(bar^(-0.5) s)]
par.c_L = 5;            % Valve coefficient liquid [kmol/(bar^(-0.5) s)]
par.z = [0.5 0.5];      % Composition inlet [Hot Cold]
par.C0 = 3*145.0377e-6; % Liquid compressability factor [1/MPa]
par.T_ref = 0.29815;    % Reference temperature [kK]
par.M = 3;              % Number of flash tanks in series

%% Calculation of inlet enthalpy [MJ/kmol]
% This calculation assumes that the inlet is in vapor phase
par.h_in = par.z(1)*(par.h_vap(1) + par.Cp_V(1)*(par.T_in-par.T_ref))+...
        par.z(2)*(par.h_vap(2) + par.Cp_V(2)*(par.T_in-par.T_ref));
end
```

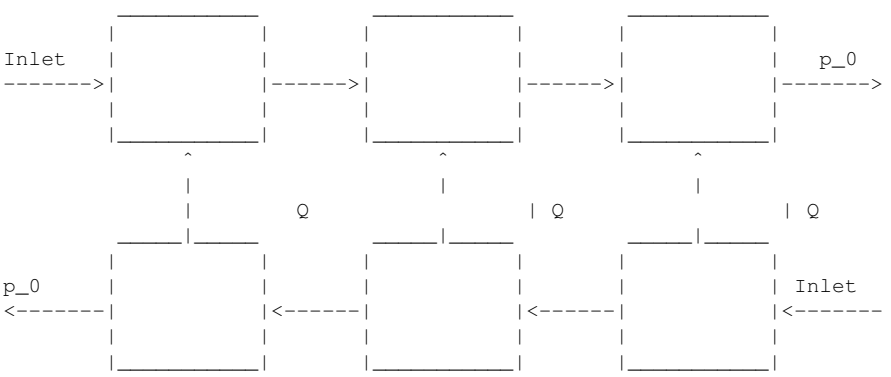
D.3 Counter-Current Heat Exchanger

This section includes all the MATLAB®-files needed to produce the results presented in Section 6.3. In addition the a description of each file is included.

D.3.1 Documentation of MATLAB®-files

```
************************************************************************
@Name of Folder: HEX_CounterCurrent
@author: Marius Reed
@organization: Process Systems Engineering, NTNU
@project: Master Thesis 2018
@since: June 2018
@requires: MATLAB R2017b (not tested in other releases)
@description: The folder cotains the files that are neceassary for
              simulating a counter current HEX. As it is, the main.m
              file will simulate a counter current HEX with methanol
              on the hot side and water on the cold side. The paramet-
              ers can be altered to include other components and chan-
              ge the inlet conditions and more. This can be change in
              the parameters functions. The results from these simula-
              tions are presented and discussed in the master thesis.
@estimated run time:
************************************************************************
```

Sketch of the counter current heat exchanger




```
**********************************************************************
System Variables:
Var1:  M_1    Molar hold up of component 1                [kmol]
Var2 : U      Internal Energy                             [MJ]
Var3 : F_V    Vapor outlet flow                           [kmol/s]
Var4 : F_L    Liquid outlet flow                          [kmol/s]
Var5 : M_V    Molar vapor hold-up                         [kmol]
Var6 : P      Pressure                                    [MPa]
Var7 : T      Temperature                                 [K]
Var8 : V_L    Liquid volume                               [m^3]
Var9 : x_1    Liquid fraction of component 1              [-]
Var10: y_1    Vapor fraction of component 1               [-]
Var11: H      Enthalpy                                    [MJ]
Var12: h_L    Liquid molar enthalpy                       [MJ/kmol]
Var13: h_V    Vapor molar enthalpy                        [MJ/kmol]
Var14: V_V    Vapor volume                                [m^3]
Var15: rho_L  Liquid molar density                        [Mmol/m^3]
Var16: M_L    Liquid molar hold-up                        [kmol]
Var17: K_1    Equilibrium coeffiecient for component 1    [-]
Var18: Ps_1   Saturation pressure for component 1         [MPa]
**********************************************************************
Brief description of each files in the folder. [i] behind an input or
output denotes the dimension of the variable. If nothing is stated,
the variable is a scalar or a struct.
----------------------------------------------------------------------
main.m
input:
output:         Saves the results into HEX_CC_UA4.mat file. In additi-
                on it produces some plots from the results.
description:    The main.m script do necessary operations to find the
                inital condition for the simulation of a counter curr-
                ent HEX. In addition it simulates the CC HEX and saves
                the results into a .mat file and produces plots.
----------------------------------------------------------------------
Flash.m
input:          t = time, w = system variables[18], data = paramters
output:         F = function residuals[18], G = Generalized derivative
                matrix[18x18]
description:    The flash.m function contains the model of a single
                component flash. It includes 2 differential equations
                (internal energy U, and component hold-up M_1). In ad-
```

```
                dition, there are 16 algebraic equations.
-------------------------------------------------------------------------
HEX.m
input:          t = time, w = system variables[18*M],data = parameters
output:         F = residuals[18*M], G = Generalized Derivative Matrix
                [18*Mx18*M]
description:    The HEX.m function contain the model of one side of a
                single component HEX with given heat transfer.
-------------------------------------------------------------------------
HEX_CC.m
input:          t = time, w = system variables [2*M*18],
                data = parameters.
output:         F = residuals [2*M*18], G = Generalized Derivative
                Matrix[2*M*18x2*M*18], Q = Heat transfer, F_in_2 =
                Inlet flow rate cold Side, F_in = Inlet flow rate hot
                side, T_in_2 = Inlet temperature cold side.
description:    The HEX_CC.m function contains the model of counter
                current heat exchanger with a single component on each
                side. The HEX is divided into M flash tanks in series.
                The model contains 2 differential equations for each
                part of the HEX on both sides as well as 16 algebraic
                equations.
-------------------------------------------------------------------------
HEX_CC_implicit.m
input:          t = time, w = system variables[2*M*18],
                data = parameters, dynVar = dynamic variables values
                at previous time step [4*M], dt = Time Step
output:         F = residuals [2*M*18], G = Generalized Derivative
                Matrix[2*M*18x2*M*18], Q = Heat transfer, F_in_2 =
                Inlet flow rate cold Side, F_in = Inlet flow rate hot
                side, T_in_2 = Inlet temperature cold side
description:    The HEX_CC.m function contains the model of counter
                current heat exchanger with a single component on each
                side. The HEX is divided into M flash tanks in series.
                The model contains 2 differential equations for each
                part of the HEX on both sides as well as 16 algebraic
                equations.
-------------------------------------------------------------------------
implicitSolverFull_CC.m
input:          w0 = initial conditions[2*M*18], data = parameters, dt
                = time step, tspan = [t_start, t_end]
output:         t_vec = time vector [(t_start-t_end)/dt + 1] ,
                w = System variables matrix[2*M*18x(t_start-t_end)/dt+1],
                Q = heat transfer [(t_start-t_end)/dt + 1],
                F_in_2 = Inlet flow rate cold side[(t_start-t_end)/dt+1],
                F_in = Flow Rate hot side [(t_start-t_end)/dt + 1],
                T_in_2 = Inlet temp. cold side[(t_start-t_end)/dt+1]
description:    Simulates the HEX_CC_implicit.m model using a implicit
                euler integrator with constant time step dt.
-------------------------------------------------------------------------
initialGuesses_gases.m
input:          par = parameters
output:         w0 = Initial guesses [18], lb = lower boundaries [18],
                ub = upper boundaries [18]
description:    Returns inital guesses and boundaries for a Flash tank
                given the parameters and that the flash is in the
                vapor-only regime.
```

```
------------------------------------------------------------------------
initialGuesses_liquid.m
input:          par = parameters
output:         w0 = Initial guesses [18], lb = lower boundaries [18],
                ub = upper boundaries [18]
description:    Returns inital guesses and boundaries for a Flash tank
                given the parameters and that the flash is in the
                liquid-only regime.
------------------------------------------------------------------------
parameters_Methanol.m
input:
output:         par = parameters
description:    Returns the parameters for a HEX with methanol as the
                single component. Here the inlet conditions are defin-
                ed as well.
------------------------------------------------------------------------
parameters_water.m
input:
output:         par = parameters
description:    Returns the parameters for a HEX with water as the si-
                ngle component. Here the inlet conditions are defined
                as well.
------------------------------------------------------------------------
parameters_CC.m
input:
output:         par = parameters
description:    Returns the parameters for a counter current HEX with
                methanol on the hot side and water on the cold side.
                The parameters for each of the sides are the same as
                defined in the two other parameters function.
------------------------------------------------------------------------
plotCC.m
In this function the dimension of the variables is equal to the once
produced by implicitSolverFull_CC.m
input:          t = time, w = system variables, Q = heat transfer,
                F_in_2 = Inlet flow rate on cold side,
                T_in_2 = Inlet temperature on cold side.
description:    Produces plots from the result obtained from the dyna-
                mic simulation of the counter current heat exchanger.
                The plots are the same as those included in the master
                thesis
------------------------------------------------------------------------
valder.m
description:    The valder.m file contains the MATLAB class valder wh-
                ich is a user defined MATLAB object. These object cal-
                culates the function values as well as computing the
                generalized derivative using vector forward automatic
                differentiations. The class includes computation of
                generalized derivative information at nonsmooth points
                for piecewise continues functions such as abs, min,
                max and mid. This is done through the lexicographic
                directional derivative.
------------------------------------------------------------------------
HEX_CC_UA4.mat
description:    This .mat file contains the results from running the
                main.m file.
************************************************************************
```

D.3.2 main.m

```
%*************************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @project: Master Thesis 2018
% @since: June 2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Main script for simulating a single component coun-
%               ter current HEX.
% @estimated run time:
%*************************************************************************
% Description of files
% @ Flash(t,w,data): Single component flash tank model
% @ HEX(t,w,data): Single component HEX model
% @ HEX_CC(t,w,data): Single component counter current HEX model
% @ HEX_CC_implicit(t,w,data,dynVar,dt): Single component counter current
%                       HEX model with discretized differential equations
% @ implicitSolverFull_CC(w,data,dt,tspan): Euler integrator for simulating
%                                           the HEX_CC_implicit model
% @ initialGuesses_gases(par): Initial guesses and boundaries for single
%                              component Flash in vapor-only region
% @ initialGuesses_liquid(par): Initial guesses and boundaries for single
%                               component Flash in liquid-only region
% @ parmeters_CC: Parameter values for counter current HEX
% @ parameters_Methanol: Parameters for HEX model with methanol
% @ parameter_water: Paramters for HEX model with water
% @ plotCC(t,w,Q,F_in_2,T_in_2): Function for plotting result from
%                                simulation of counter current HEX
% @ valder: Matlab class for calculating the function residuals and
%           Generalized derivatives.
%*************************************************************************
clear all
clc
tic
%% Variable list
% Cold side is Var 1-18, Hot Side has the same structure and is Var 19-36
% Var1:  M_1    Molar hold up of component 1              [kmol]
% Var2 : U      Internal Energy                           [MJ]
% Var3 : F_V    Vapor outlet flow                         [kmol/s]
% Var4 : F_L    Liquid outlet flow                        [kmol/s]
% Var5 : M_V    Molar vapor hold-up                       [kmol]
% Var6 : P      Pressure                                  [MPa]
% Var7 : T      Temperature                               [K]
% Var8 : V_L    Liquid volume                             [m^3]
% Var9 : x_1    Liquid fraction of component 1            [-]
% Var10: y_1    Vapor fraction of component 1             [-]
% Var11: H      Enthalpy                                  [MJ]
% Var12: h_L    Liquid molar enthalpy                     [MJ/kmol]
% Var13: h_V    Vapor molar enthalpy                      [MJ/kmol]
% Var14: V_V    Vapor volume                              [m^3]
% Var15: rho_L  Liquid molar density                      [Mmol/m^3]
% Var16: M_L    Liquid molar hold-up                      [kmol]
% Var17: K_1    Equilibrium coeffiecient for component 1  [-]
% Var18: Ps_1   Saturation pressure for component 1       [MPa]
```

```
%% Component list:
% Hot Side: Methanol       CH3OH
% Cold Side: Water          H2O

%% Solver Settings
options_lsq = optimoptions(@lsqnonlin,'Display','iter',...
     'MaxIterations',1000,'MaxFunEvals',1e10,...
     'stepTolerance',1e-20,'FunctionTolerance',1e-20,...
     'specifyObjectiveGradient', true,...
     'OptimalityTolerance',1e-20);
 options_fsolve = optimoptions(@fsolve,'Display','iter',...
     'MaxIterations',2e3,'MaxFunEvals',1e10,...
     'stepTolerance',1e-13,'FunctionTolerance',1e-13,...
     'specifyObjectiveGradient', true,...
     'OptimalityTolerance',1e-13, 'Algorithm','levenberg-marquardt');

%% Solving single Flash Tank with Methanol with no Heat Transfer
data.par = parameters_Methanol();             % Parameters
[w0,lb,ub] = initialGuesses_gases(data.par); % Initial guess and boundaries
data.par.Q = 0;                               % No Heat Transfer
data.par.V_T = data.par.V_T/data.par.M;    % Volume equal to one cell in HEX
lb(14) = data.par.V_T;                        % Lower Vapor Volume Boundary
ub(14) = data.par.V_T;                        % Upper Vapor  Volume Boundary
w0_methanol = lsqnonlin(@(w) Flash(1,w,data),...
    w0,lb,ub,options_lsq);                    % Solving the Flash
%% Solving a Full Single Side of HEX with no Heat Transfer
data.par = parameters_Methanol();          % Parameters
w_initial = [];
for i = 1:18                                  % Repeating each variable M times
    w_initial = [w_initial;repmat(w0_methanol(i),data.par.M,1)];
end
w_initial_methanol = fsolve(@(w)HEX(1,w,data),...
    w_initial,options_fsolve);              % Solving one Side HEX
%% Solving single Flash Tank with Methanol with no Heat Transfer
data.par = parameters_water();                % Parameters
[w0,lb,ub] = initialGuesses_liquid(data.par);% Initial guess and boundaries
data.par.Q = 0;                               % No Heat Transfer
data.par.V_T = data.par.V_T/data.par.M;  % Volume equal to one cell in HEX
lb(8) = data.par.V_T;                        % Lower Liquid Volume Boundary
ub(8) = data.par.V_T;                        % Upper Liquid Volume Boundary
w0_water = lsqnonlin(@(w) Flash(1,w,data),...
    w0,lb,ub,options_lsq);                    % Solving the Flash

%% Solving a Full Single Side of HEX with no Heat Transfer
data.par = parameters_water();             % Parameters
w_initial = [];                               % Clearing Var
for i = 1:18                                  % Repeating each Var M Times
    w_initial = [w_initial;repmat(w0_water(i),data.par.M,1)];
end
% Solving one Side HEX
w_initial_water = fsolve(@(w)HEX(1,w,data),w_initial,options_fsolve);

%% Finding inital values with desired UA (Here UA = 4 is used)
data.par = parameters_CC();                        % Parameters
w_initial = [w_initial_methanol;w_initial_water];% Concatenating Init. Val.
%%
UA = 0:0.1:4;                                      % UA values
```

```
data.par.UA = 0;                                    % First UA Value
% Solving CC HEX with UA = 0
w_initial_Full = fsolve(@(w)HEX_CC(1,w,data),...
    w_initial,options_fsolve);
% Stepwise changing UA until desired UA
%%
for i = 2:length(UA)
    data.par.UA = UA(i);
    w_initial_Full(:,i) = fsolve(@(w)HEX_CC(1,w,data),...
        w_initial_Full(:,i-1),options_fsolve);
end

%% Simulating the CC HEX
data.par = parameters_CC();             % Setting the parameters
[t_full,w_full,Q,F_in_2,F_in,T_in_2] = implicitSolverFull_CC(...
    w_initial_Full(:,end),data,0.05,[0 200]);
% Saving the result into a .mat file
save('HEX_CC_UA_4.mat','t_full','w_full','Q','F_in_2','F_in','T_in_2','data');
%% Generating Plots
plotCC(t_full,w_full,Q,F_in_2,T_in_2)
toc
```

D.3.3 Flash.m

```
%**********************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Single component Flash tank model
% @input: t = time, w = system variables data = parameters
% @output: F = residuals, G = Generalized Derivative Matrix
%**********************************************************************

function [F,G] = Flash(t,w,data)
%% Unpacking parameters
par = data.par;
F_in = par.F_in;            % Inlet flow rate [kmol/s]
Q = par.Q;                  % Heat transfer [MW]
R = par.R;                  % Gas constant [MJ/kmol*kK]
V_T = par.V_T;              % Total volume [m^3]
P_0 = par.P_0;              % Outlet pressure [MPa]
Cp_V_1 = par.Cp_V(1);       % Vapor Heat Capcity [MJ/kmol*kK]
Cp_L_1 =  par.Cp_L(1);      % Liquid Heat Capcity [MJ/kmol*kK]
h_vap_1 = par.h_vap(1);     % Heat of vaporization [MJ/kmol]
A_1 = par.Antoine(1,1);     % Antoine Parameter
B_1 = par.Antoine(2,1);     % Antoine Parameter
C_1 = par.Antoine(3,1);     % Antoine Parameter
rho_1 = par.rho(1);         % Molar density [Mmol/m^3]
z_1 = par.z(1);             % Inlet composition (Should be 1!)
h_in = par.h_in;            % Inlet molar enthalpy [MJ/kmol]
C0 = par.C0;                % Compressability factor [1/MPa]
T_ref = par.T_ref;          % Reference temperature [kK]
c_V = par.c_V;              % Vapor valve coefficient [kmol/bar^0.5*s]
c_L = par.c_L;              % Liquid valve coefficient [kmol/bar^0.5*s]
```

```
%% Unpacking system variables
w = valder(w,eye(length(w)));
M_1 = w(1);         % Component Molar hold-up [kmol]
U = w(2);           % Internal Energy [MJ]
F_V = w(3);         % Vapor flow rate [kmol/s]
F_L = w(4);         % Liquid flow rate [kmol/s]
M_V = w(5);         % Vapor molar hold-up [kmol]
P = w(6);           % Pressure [MPa]
T = w(7);           % Temperature [kK]
V_L = w(8);         % Liquid Volume [m^3]
x_1 = w(9);         % Liquid mole fraction
y_1 = w(10);        % Vapor mole fraction
H = w(11);          % Enthalpy [MJ]
h_L = w(12);        % Molar enthalpy in liquid phase [MJ/kmol]
h_V = w(13);        % Molar enthalpy in vapor phase [MJ/kmol]
V_V = w(14);        % Vapor volume [m^3]
rho_L = w(15);      % Liquid molar density [Mmol/kK]
M_L = w(16);        % Liquid molar hold-up [kmol]
k_1 = w(17);        % Vapor-liquid equilibrium coefficent
Ps_1 = w(18);       % Saturation pressure


%% Differential equations
f1 = F_in*z_1 - F_L*x_1 - F_V * y_1;
f2 = F_in*h_in - F_L*h_L - F_V*h_V + Q;

%% Algebraic equations
f3 = M_1 - (M_L*x_1 + M_V*y_1);
f4 = M_1 - (M_L + M_V);
f5 = H - (M_L*h_L + M_V*h_V);
f6 = H - (U + P*V_T);
f7 = y_1 - k_1*x_1;
f8 = midobj(M_V/(M_V+M_L),x_1 - y_1, (M_V/(M_V+M_L))-1);
f9 = V_T - (V_L + V_V);
f10 = P*V_V - M_V*R*T;
f11 = V_L - M_L/(rho_L*1e3);
f12 = F_V*V_T - c_V*V_V*max2(0,(P-P_0)/sqrt(abs((P-P_0))+1e-10));
f13 = F_L*V_T - c_L*V_L*max2(0,(P-P_0)/sqrt(abs((P-P_0))+1e-10));
f14 = 1/rho_L - (x_1/(rho_1*(1+C0*(P-0.1))));
f15 = h_L - (x_1*Cp_L_1)*(T-T_ref);
f16 = h_V - (y_1*(h_vap_1 + Cp_V_1*(T-T_ref)));
f17 = k_1*P - Ps_1;
f18 = Ps_1 - 10^(A_1 - B_1/(T*1e3+C_1))/10;

%% Extracting the values and the derivatives from the valder objects
F = [getVal(f1);getVal(f2);getVal(f3);getVal(f4);getVal(f5);getVal(f6);...
    getVal(f7);getVal(f8);getVal(f9);getVal(f10);getVal(f11);...
    getVal(f12);getVal(f13);getVal(f14);getVal(f15);getVal(f16);...
    getVal(f17);getVal(f18)];

G = [getDer(f1);getDer(f2);getDer(f3);getDer(f4);getDer(f5);getDer(f6);...
    getDer(f7);getDer(f8);getDer(f9);getDer(f10);getDer(f11);...
    getDer(f12);getDer(f13);getDer(f14);getDer(f15);getDer(f16);...
    getDer(f17);getDer(f18)];
```

```
end
```

D.3.4 HEX.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: One side of heat exchanger with given Q with one component
% @input: t = time, w = system variables, data = parameters
% @output: F = residuals, G = Generalized Derivative Matrix
%*****************************************************************

function [F,G] = HEX(t,w,data)
%% Unpacking parameters
par = data.par;
F_in = par.F_in;            % Inlet flow rate [kmol/s]
Q = par.Q;                  % Heat transfer [MW]
R = par.R;                  % Gas constant [MJ/kmol*kK]
V_T = par.V_T;              % Total volume [m^3]
P_0 = par.P_0;              % Outlet pressure [MPa]
Cp_V_1 = par.Cp_V(1);       % Vapor Heat Capcity [MJ/kmol*kK]
Cp_L_1 =  par.Cp_L(1);      % Liquid Heat Capcity [MJ/kmol*kK]
h_vap_1 = par.h_vap(1);     % Heat of vaporization [MJ/kmol]
A_1 = par.Antoine(1,1);     % Antoine Parameter
B_1 = par.Antoine(2,1);     % Antoine Parameter
C_1 = par.Antoine(3,1);     % Antoine Parameter
rho_1 = par.rho(1);         % Molar density [Mmol/m^3]
z_1 = par.z(1);             % Inlet composition (Should be 1!)
h_in = par.h_in;            % Inlet molar enthalpy [MJ/kmol]
M = par.M;                  % Number of flash tanks in series
C0 = par.C0;                % Compressability factor [1/MPa]
T_ref = par.T_ref;          % Reference temperature [kK]
c_V = par.c_V*sqrt(M);      % Vapor valve coefficient [kmol/bar^0.5*s]
c_L = par.c_L*sqrt(M);      % Liquid valve coefficient [kmol/bar^0.5*s]

%% Unpacking system variables
L = length(w);
w = valder(w,eye(length(w)));
M_1 = w(1:M);               % Component Molar hold-up [kmol]
U = w(1*M+1:2*M);           % Internal Energy [MJ]
F_V = w(2*M+1:3*M);         % Vapor flow rate [kmol/s]
F_L = w(3*M+1:4*M);         % Liquid flow rate [kmol/s]
M_V = w(4*M+1:5*M);         % Vapor molar hold-up [kmol]
P = w(5*M+1:6*M);           % Pressure [MPa]
T = w(6*M+1:7*M);           % Temperature [kK]
V_L = w(7*M+1:8*M);         % Liquid Volume [m^3]
x_1 = w(8*M+1:9*M);         % Liquid mole fraction
y_1 = w(9*M+1:10*M);        % Vapor mole fraction
H = w(10*M+1:11*M);         % Enthalpy [MJ]
h_L = w(11*M+1:12*M);       % Molar enthalpy in liquid phase [MJ/kmol]
h_V = w(12*M+1:13*M);       % Molar enthalpy in vapor phase [MJ/kmol]
V_V = w(13*M+1:14*M);       % Vapor volume [m^3]
rho_L = w(14*M+1:15*M);     % Liquid molar density [Mmol/kK]
```

```
M_L = w(15*M+1:16*M);        % Liquid molar hold-up [kmol]
k_1 = w(16*M+1:17*M);        % Vapor-liquid equilibrium coefficent
Ps_1 = w(17*M+1:18*M);       % Saturation pressure

%% Preallocating memory
f1 = valder(zeros(M,1),zeros(M,L));
f2 = valder(zeros(M,1),zeros(M,L));
f3 = valder(zeros(M,1),zeros(M,L));
f4 = valder(zeros(M,1),zeros(M,L));
f5 = valder(zeros(M,1),zeros(M,L));
f6 = valder(zeros(M,1),zeros(M,L));
f7 = valder(zeros(M,1),zeros(M,L));
f8 = valder(zeros(M,1),zeros(M,L));
f9 = valder(zeros(M,1),zeros(M,L));
f10 = valder(zeros(M,1),zeros(M,L));
f11 = valder(zeros(M,1),zeros(M,L));
f12 = valder(zeros(M,1),zeros(M,L));
f13 = valder(zeros(M,1),zeros(M,L));
f14 = valder(zeros(M,1),zeros(M,L));
f15 = valder(zeros(M,1),zeros(M,L));
f16 = valder(zeros(M,1),zeros(M,L));
f17 = valder(zeros(M,1),zeros(M,L));
f18 = valder(zeros(M,1),zeros(M,L));

%% Differential equations
f1(1) = F_in*z_1 - F_L(1)*x_1(1) - F_V(1) * y_1(1);
f2(1) = F_in*h_in - F_L(1)*h_L(1) - F_V(1)*h_V(1) + Q/M;
for j = 2:M
    f1(j) = F_L(j-1)*x_1(j-1) + F_V(j-1)*y_1(j-1)...
        - (F_L(j)*x_1(j) + F_V(j)*y_1(j));
    f2(j) = F_L(j-1)*h_L(j-1) + F_V(j-1)*h_V(j-1)...
        - (F_L(j)*h_L(j) + F_V(j)*h_V(j)) + Q/M;
end
%% Algebraic equations
for j = 1:M
    f3(j) = M_1(j) - (M_L(j)*x_1(j) + M_V(j)*y_1(j));
    f4(j) = (M_1(j)) - (M_L(j) + M_V(j));
    f5(j) = H(j) - (M_L(j)*h_L(j) + M_V(j)*h_V(j));
    f6(j) = H(j) - (U(j) + P(j)*V_T/M);
    f7(j) = y_1(j) - k_1(j)*x_1(j);
    f8(j) = midobj(M_V(j)/(M_V(j)+M_L(j)),x_1(j) - y_1(j),...
        (M_V(j)/(M_V(j)+M_L(j)))-1);
    f9(j) = V_T/M - (V_L(j) + V_V(j));
    f10(j) = P(j)*V_V(j) - M_V(j)*R*T(j);
    f11(j) = V_L(j) - M_L(j)/(rho_L(j)*1e3);
    if j < M
        f12(j) = F_V(j)*V_T/M - c_V*V_V(j)*((P(j)-P(j+1))...
            /sqrt(abs((P(j)-P(j+1)))+1e-10));
        f13(j) = F_L(j)*V_T/M - c_L*V_L(j)*((P(j)-P(j+1))...
            /sqrt(abs((P(j)-P(j+1)))+1e-10));
    else
        f12(j) = F_V(j)*V_T/M - c_V*V_V(j)*max2(0,(P(j)-P_0)...
            /sqrt(abs((P(j)-P_0))+1e-10));
        f13(j) = F_L(j)*V_T/M - c_L*V_L(j)*max2(0,((P(j)-P_0)...
            /sqrt(abs((P(j)-P_0))+1e-10)));
    end
    f14(j) = rho_L(j)*(x_1(j)) - (rho_1*(1+C0*(P(j)-0.1)));
```

```
    f15(j) = h_L(j) - (x_1(j)*Cp_L_1)*(T(j)-T_ref);
    f16(j) = h_V(j) - (y_1(j)*(h_vap_1 + Cp_V_1*(T(j)-T_ref)));
    f17(j) = k_1(j)*P(j) - Ps_1(j);
    f18(j) = Ps_1(j) - 10^(A_1 - B_1/(T(j)*1e3+C_1))/10;

%% Extracting the values and the derivatives from the valder objects
F = [getVal(f1);getVal(f2);getVal(f3);getVal(f4);getVal(f5);getVal(f6);...
    getVal(f7);getVal(f8);getVal(f9);getVal(f10);getVal(f11);...
    getVal(f12);getVal(f13);getVal(f14);getVal(f15);getVal(f16);...
    getVal(f17);getVal(f18)];

G = [getDer(f1);getDer(f2);getDer(f3);getDer(f4);getDer(f5);getDer(f6);...
    getDer(f7);getDer(f8);getDer(f9);getDer(f10);getDer(f11);...
    getDer(f12);getDer(f13);getDer(f14);getDer(f15);getDer(f16);...
    getDer(f17);getDer(f18)];
end
```

D.3.5 HEX_CC.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Counter current heat exchanger
% @input: t = time, w = system variables, data = parameters
% @output: F = residuals, G = Generalized Derivative Matrix,
%          Q = Heat transfer, F_in_2 = Inlet flow rate cold Side
%          F_in = Inlet flow rate hot side,
%          T_in_2 = Inlet temperature cold side
%*****************************************************************

function [F,G,Q,F_in_2,F_in,T_in_2] = HEX_CC(t,w,data)
%% Unpacking parameters
par = data.par;
F_in = par.F_in;            % Inlet flow rate on hot side [kmol/s]
F_in_2 = par.F_in_2;        % Inlet flow rate on cold side [kmol/s]
R = par.R;                  % Gas constant [MJ/kmol*kK]
V_T = par.V_T;              % Total volume [m^3]
P_0 = par.P_0;              % Outlet pressure [MPa]
Cp_V_1 = par.Cp_V(1);       % Vapor heat capacity component 1 [MJ/kmol*kK]
Cp_L_1 =  par.Cp_L(1);      % Liquid heat capacity component 1 [MJ/kmol*kK]
Cp_V_2 = par.Cp_V(2);       % Vapor heat capacity component 2 [MJ/kmol*kK]
Cp_L_2 =  par.Cp_L(2);      % Liquid heat capacity component 1 [MJ/kmol*kK]
h_vap_1 = par.h_vap(1);     % Heat of vaporization component 1 [MJ/kmol]
h_vap_2 = par.h_vap(2);     % Heat of vaporization component 2 [MJ/kmol]
A_1 = par.Antoine(1,1);     % Antoine Parameter component 1
B_1 = par.Antoine(2,1);     % Antoine Parameter component 1
C_1 = par.Antoine(3,1);     % Antoine Parameter component 1
A_2 = par.Antoine(1,2);     % Antoine Parameter component 2
B_2 = par.Antoine(2,2);     % Antoine Parameter component 2
C_2 = par.Antoine(3,2);     % Antoine Parameter component 2
rho_1 = par.rho(1);         % Molar density component 1 [Mmol/m^3]
rho_2 = par.rho(2);         % Molar density component 2 [Mmol/m^3]
z_1 = par.z(1);             % Molar fraction hot side (Should be 1)
```

```
z_2 = par.z(2);              % Molar fraction cold side (Should be 1)
h_in = par.h_in;             % Inlet molar enthalpy hot side [MJ/kmol]
h_in_2 = par.h_in_2;         % Inlet molar enthalpy hot side [MJ/kmol]
T_in_2 = par.T_in_2;         % Inlet temperature cold side [kK]
M = par.M;                   % Number of flash tanks in series
C0 = par.C0;                 % Compressability factor [1/MPa]
T_ref = par.T_ref;           % Reference temperature [kK]
c_V = par.c_V*sqrt(M);       % Vapor valve coefficient [kmol/bar^.5 s]
c_L = par.c_L*sqrt(M);       % Liquid valve coefficient [kmol/bar^.5 s]
UA = data.par.UA;            % Heat transfer coefficent times area [MW/kK]

%% Unpacking hot side variables
L = length(w);
w = valder(w,eye(length(w)));
M_1 = w(1:M);                % Component Molar hold-up [kmol]
U = w(1*M+1:2*M);            % Internal Energy [MJ]
F_V = w(2*M+1:3*M);          % Vapor flow rate [kmol/s]
F_L = w(3*M+1:4*M);          % Liquid flow rate [kmol/s]
M_V = w(4*M+1:5*M);          % Vapor molar hold-up [kmol]
P = w(5*M+1:6*M);            % Pressure [MPa]
T = w(6*M+1:7*M);            % Temperature [kK]
V_L = w(7*M+1:8*M);          % Liquid Volume [m^3]
x_1 = w(8*M+1:9*M);          % Liquid mole fraction
y_1 = w(9*M+1:10*M);         % Vapor mole fraction
H = w(10*M+1:11*M);          % Enthalpy [MJ]
h_L = w(11*M+1:12*M);        % Molar enthalpy in liquid phase [MJ/kmol]
h_V = w(12*M+1:13*M);        % Molar enthalpy in vapor phase [MJ/kmol]
V_V = w(13*M+1:14*M);        % Vapor volume [m^3]
rho_L = w(14*M+1:15*M);      % Liquid molar density [Mmol/kK]
M_L = w(15*M+1:16*M);        % Liquid molar hold-up [kmol]
k_1 = w(16*M+1:17*M);        % Vapor-liquid equilibrium coefficent
Ps_1 = w(17*M+1:18*M);       % Saturation pressure

%% Unpacking hot side variables
M_1_2 = w(18*M+1:19*M);      % Component Molar hold-up [kmol]
U_2 = w(19*M+1:20*M);        % Internal Energy [MJ]
F_V_2 = w(20*M+1:21*M);      % Vapor flow rate [kmol/s]
F_L_2 = w(21*M+1:22*M);      % Liquid flow rate [kmol/s]
M_V_2 = w(22*M+1:23*M);      % Vapor molar hold-up [kmol]
P_2 = w(23*M+1:24*M);        % Pressure [MPa]
T_2 = w(24*M+1:25*M);        % Temperature [kK]
V_L_2 = w(25*M+1:26*M);      % Liquid Volume [m^3]
x_1_2 = w(26*M+1:27*M);      % Liquid mole fraction
y_1_2 = w(27*M+1:28*M);      % Vapor mole fraction
H_2 = w(28*M+1:29*M);        % Enthalpy [MJ]
h_L_2 = w(29*M+1:30*M);      % Molar enthalpy in liquid phase [MJ/kmol]
h_V_2 = w(30*M+1:31*M);      % Molar enthalpy in vapor phase [MJ/kmol]
V_V_2 = w(31*M+1:32*M);      % Vapor volume [m^3]
rho_L_2 = w(32*M+1:33*M);    % Liquid molar density [Mmol/kK]
M_L_2 = w(33*M+1:34*M);      % Liquid molar hold-up [kmol]
k_1_2 = w(34*M+1:35*M);      % Vapor-liquid equilibrium coefficent
Ps_1_2 = w(35*M+1:36*M);     % Saturation pressure

%% Preallocating memory
f1 = valder(zeros(M,1),zeros(M,L));
f2 = valder(zeros(M,1),zeros(M,L));
f3 = valder(zeros(M,1),zeros(M,L));
```

```
f4 = valder(zeros(M,1),zeros(M,L));
f5 = valder(zeros(M,1),zeros(M,L));
f6 = valder(zeros(M,1),zeros(M,L));
f7 = valder(zeros(M,1),zeros(M,L));
f8 = valder(zeros(M,1),zeros(M,L));
f9 = valder(zeros(M,1),zeros(M,L));
f10 = valder(zeros(M,1),zeros(M,L));
f11 = valder(zeros(M,1),zeros(M,L));
f12 = valder(zeros(M,1),zeros(M,L));
f13 = valder(zeros(M,1),zeros(M,L));
f14 = valder(zeros(M,1),zeros(M,L));
f15 = valder(zeros(M,1),zeros(M,L));
f16 = valder(zeros(M,1),zeros(M,L));
f17 = valder(zeros(M,1),zeros(M,L));
f18 = valder(zeros(M,1),zeros(M,L));
f19 = valder(zeros(M,1),zeros(M,L));
f20 = valder(zeros(M,1),zeros(M,L));
f21 = valder(zeros(M,1),zeros(M,L));
f22 = valder(zeros(M,1),zeros(M,L));
f23 = valder(zeros(M,1),zeros(M,L));
f24 = valder(zeros(M,1),zeros(M,L));
f25 = valder(zeros(M,1),zeros(M,L));
f26 = valder(zeros(M,1),zeros(M,L));
f27 = valder(zeros(M,1),zeros(M,L));
f28 = valder(zeros(M,1),zeros(M,L));
f29 = valder(zeros(M,1),zeros(M,L));
f30 = valder(zeros(M,1),zeros(M,L));
f31 = valder(zeros(M,1),zeros(M,L));
f32 = valder(zeros(M,1),zeros(M,L));
f33 = valder(zeros(M,1),zeros(M,L));
f34 = valder(zeros(M,1),zeros(M,L));
f35 = valder(zeros(M,1),zeros(M,L));
f36 = valder(zeros(M,1),zeros(M,L));

%% Defining how some of the input will change during simulation

% Changing the temperature in the inlet on the cold side
if t < 5
    T_in_2 = T_in_2;
    h_in_2 = h_in_2;
elseif t < 70
    T_in_2 = T_in_2 - 0.0003*(t-5);
    h_in_2 = par.Cp_L(2)*(T_in_2-T_ref);
else
    T_in_2 = T_in_2 - 0.0003*65;
    h_in_2 = par.Cp_L(2)*(T_in_2-T_ref);
end

% Changing the inlet flow rate on the cold side
if t < 100
    F_in_2 = F_in_2;
elseif t < 150
    F_in_2 = F_in_2 + 0.001*(t-100);
else
    F_in_2 = 0.15;
end
```

```
%% Differential equations
f1(1) = (F_in*z_1 - (max2(0,F_L(1))*x_1(1) +  max2(0,F_V(1))*y_1(1))...
    - (min2(0,F_L(1))*x_1(2) +  min2(0,F_V(1))*y_1(2)));
f2(1) = (F_in*h_in - (max2(0,F_L(1))*h_L(1) + max2(0,F_V(1))*h_V(1))...
    - min2(0,F_L(1))*h_L(2) + min2(0,F_V(1))*h_V(2) + UA*(T_2(M)-T(1)));
f19(1) = (F_in_2*z_2 - (max2(0,F_L_2(1))*x_1_2(1)...
    +  max2(0,F_V_2(1))*y_1_2(1)) - (min2(0,F_L_2(1))*x_1_2(2)...
    +  min2(0,F_V_2(1))*y_1_2(2)));
f20(1) = (F_in_2*h_in_2 - (max2(0,F_L_2(1))*h_L_2(1)...
    + max2(0,F_V_2(1))*h_V_2(1)) - (min2(0,F_L_2(1))*h_L_2(2)...
    + min2(0,F_V_2(1))*h_V_2(2)) + UA*(T(M)-T_2(1)));
Q(1) = getVal(UA*(T_2(M)-T(1)));
for j = 2:M-1
    f1(j) = (max2(0,F_L(j-1)*x_1(j-1) + F_V(j-1)*y_1(j-1))...
        - max2(0,F_L(j)*x_1(j) + F_V(j)*y_1(j)) + min2(0,F_L(j-1)*x_1(j)...
        + F_V(j-1)*y_1(j)) - min2(0,F_L(j)*x_1(j+1) + F_V(j)*y_1(j+1)));
    f2(j) = ((max2(0,F_L(j-1))*h_L(j-1) + max2(0,F_V(j-1))*h_V(j-1))...
        - (max2(0,F_L(j))*h_L(j) + max2(0,F_V(j))*h_V(j))...
        + (min2(0,F_L(j-1))*h_L(j) + min2(0,F_V(j-1))*h_V(j))...
        - (min2(0,F_L(j))*h_L(j+1) + min2(0,F_V(j))*h_V(j+1))...
        + UA*(T_2(M-j+1)-T(j)));
    f19(j) = (max2(0,F_L_2(j-1)*x_1_2(j-1) + F_V_2(j-1)*y_1_2(j-1))...
        - max2(0,F_L_2(j)*x_1_2(j) + F_V_2(j)*y_1_2(j))...
        + min2(0,F_L_2(j-1)*x_1_2(j) + F_V_2(j-1)*y_1_2(j))...
        - min2(0,F_L_2(j)*x_1_2(j+1) + F_V_2(j)*y_1_2(j+1)));
    f20(j) = ((max2(0,F_L_2(j-1))*h_L_2(j-1)...
        + max2(0,F_V_2(j-1))*h_V_2(j-1)) - (max2(0,F_L_2(j))*h_L_2(j)...
        + max2(0,F_V_2(j))*h_V_2(j)) + (min2(0,F_L_2(j-1))*h_L_2(j)...
        + min2(0,F_V_2(j-1))*h_V_2(j)) - (min2(0,F_L_2(j))*h_L_2(j+1)...
        + min2(0,F_V_2(j))*h_V_2(j+1))  + UA*(T(M-j+1)-T_2(j)));
    Q(j) = getVal(UA*(T_2(M-j+1)-T(j)));
end
f1(M) = (max2(0,F_L(M-1)*x_1(M-1) + F_V(M-1)*y_1(M-1))...
    - max2(0,F_L(M)*x_1(M) + F_V(M)*y_1(M))...
    + min2(0,F_L(M-1)*x_1(M) + F_V(M-1)*y_1(M)));
f2(M) = ((max2(0,F_L(M-1))*h_L(M-1) + max2(0,F_V(M-1))*h_V(M-1))...
    - (max2(0,F_L(M))*h_L(M) + max2(0,F_V(M))*h_V(M))...
    + (min2(0,F_L(M-1))*h_L(M) + min2(0,F_V(M-1))*h_V(M))...
    + UA*(T_2(1)-T(M)));
f19(M) =(max2(0,F_L_2(M-1)*x_1_2(M-1) + F_V_2(M-1)*y_1_2(M-1))...
    - max2(0,F_L_2(M)*x_1_2(M) + F_V_2(M)*y_1_2(M))...
    + min2(0,F_L_2(M-1)*x_1_2(M) + F_V_2(M-1)*y_1_2(M)));
f20(M) = ((max2(0,F_L_2(M-1))*h_L_2(M-1) + max2(0,F_V_2(M-1))*h_V_2(M-1))...
    - (max2(0,F_L_2(M))*h_L_2(M) + max2(0,F_V_2(M))*h_V_2(M))...
    + (min2(0,F_L_2(M-1))*h_L_2(M) + min2(0,F_V_2(M-1))*h_V_2(M))...
    + UA*(T(1)-T_2(M)));
Q(M) = getVal(UA*(T_2(1)-T(M)));
%% Algebraic equations
for j = 1:M
    f3(j) = M_1(j) - (M_L(j)*x_1(j) + M_V(j)*y_1(j));
    f4(j) = (M_1(j)) - (M_L(j) + M_V(j));
    f5(j) = H(j) - (M_L(j)*h_L(j) + M_V(j)*h_V(j));
    f6(j) = H(j) - (U(j) + P(j)*V_T/M);
    f7(j) = y_1(j) - k_1(j)*x_1(j);
    f8(j) = midobj(M_V(j)/(M_V(j)+M_L(j)),x_1(j) - y_1(j),...
        (M_V(j)/(M_V(j)+M_L(j)))-1);
    f9(j) = V_T/M - (V_L(j) + V_V(j));
```

```
    f10(j) = P(j)*V_V(j) - M_V(j)*R*T(j);
    f11(j) = V_L(j)*(rho_L(j)*1e3) - M_L(j);
    if j < M
        f12(j) = F_V(j)*V_T/M - c_V*V_V(j)*((P(j)-P(j+1))...
            /sqrt(abs((P(j)-P(j+1)))+1e-10));
        f13(j) = F_L(j)*V_T/M - c_L*V_L(j)*((P(j)-P(j+1))...
            /sqrt(abs((P(j)-P(j+1)))+1e-10));
    else
        f12(j) = F_V(j)*V_T/M - c_V*V_V(j)*max2(0,(P(j)-P_0)...
            /sqrt(abs((P(j)-P_0))+1e-10));
        f13(j) = F_L(j)*V_T/M - c_L*V_L(j)*max2(0,((P(j)-P_0)...
            /sqrt(abs((P(j)-P_0))+1e-10)));
    end
    f14(j) = rho_L(j)*(x_1(j)) - (rho_1*(1+C0*(P(j)-0.1)));
    f15(j) = h_L(j) - (x_1(j)*Cp_L_1)*(T(j)-T_ref);
    f16(j) = h_V(j) - (y_1(j)*(h_vap_1 + Cp_V_1*(T(j)-T_ref)));
    f17(j) = k_1(j)*P(j) - Ps_1(j);
    f18(j) = Ps_1(j) - 10^(A_1 - B_1/(T(j)*1e3+C_1))/10;

    f21(j) = M_1_2(j) - (M_L_2(j)*x_1_2(j) + M_V_2(j)*y_1_2(j));
    f22(j) = (M_1_2(j)) - (M_L_2(j) + M_V_2(j));
    f23(j) = H_2(j) - (M_L_2(j)*h_L_2(j) + M_V_2(j)*h_V_2(j));
    f24(j) = H_2(j) - (U_2(j) + P_2(j)*V_T/M);
    f25(j) = y_1_2(j) - k_1_2(j)*x_1_2(j);
    f26(j) = midobj(M_V_2(j)/(M_V_2(j)+M_L_2(j)),x_1_2(j) - y_1_2(j),...
        (M_V_2(j)/(M_V_2(j)+M_L_2(j)))-1);
    f27(j) = V_T/M - (V_L_2(j) + V_V_2(j));
    f28(j) = P_2(j)*V_V_2(j) - M_V_2(j)*R*T_2(j);
    f29(j) = V_L_2(j)*(rho_L_2(j)*1e3) - M_L_2(j);
    if j < M
        f30(j) = F_V_2(j)*V_T/M - c_V*V_V_2(j)*((P_2(j)-P_2(j+1))...
            /sqrt(abs((P_2(j)-P_2(j+1)))+1e-10));
        f31(j) = F_L_2(j)*V_T/M - c_L*V_L_2(j)*((P_2(j)-P_2(j+1))...
            /sqrt(abs((P_2(j)-P_2(j+1)))+1e-10));
    else
        f30(j) = F_V_2(j)*V_T/M - c_V*V_V_2(j)*max2(0,(P_2(j)-P_0)...
            /sqrt(abs((P_2(j)-P_0))+1e-10));
        f31(j) = F_L_2(j)*V_T/M - c_L*V_L_2(j)*max2(0,((P_2(j)-P_0)...
            /sqrt(abs((P_2(j)-P_0))+1e-10)));
    end

    f32(j) = rho_L_2(j)*(x_1_2(j)) - (rho_2*(1+C0*(P_2(j)-0.1)));
    f33(j) = h_L_2(j) - (x_1_2(j)*Cp_L_2)*(T_2(j)-T_ref);
    f34(j) = h_V_2(j) - (y_1_2(j)*(h_vap_2 + Cp_V_2*(T_2(j)-T_ref)));
    f35(j) = k_1_2(j)*P_2(j) - Ps_1_2(j);
    f36(j) = Ps_1_2(j) - 10^(A_2 - B_2/(T_2(j)*1e3+C_2))/10;

end

%% Extracting the values and the derivatives from the valder objects
F = [getVal(f1);getVal(f2);getVal(f3);getVal(f4);getVal(f5);getVal(f6);...
    getVal(f7);getVal(f8);getVal(f9);getVal(f10);getVal(f11);...
    getVal(f12);getVal(f13);getVal(f14);getVal(f15);getVal(f16);...
    getVal(f17);getVal(f18);getVal(f19);getVal(f20);getVal(f21);...
    getVal(f22);getVal(f23);getVal(f24);getVal(f25);getVal(f26);...
    getVal(f27);getVal(f28);getVal(f29);getVal(f30);getVal(f31);...
    getVal(f32);getVal(f33);getVal(f34);getVal(f35);getVal(f36)];
```

```
G = [getDer(f1);getDer(f2);getDer(f3);getDer(f4);getDer(f5);getDer(f6);...
    getDer(f7);getDer(f8);getDer(f9);getDer(f10);getDer(f11);...
    getDer(f12);getDer(f13);getDer(f14);getDer(f15);getDer(f16);...
    getDer(f17);getDer(f18);getDer(f19);getDer(f20);getDer(f21);...
    getDer(f22);getDer(f23);getDer(f24);getDer(f25);getDer(f26);...
    getDer(f27);getDer(f28);getDer(f29);getDer(f30);getDer(f31);...
    getDer(f32);getDer(f33);getDer(f34);getDer(f35);getDer(f36)];
end
```

D.3.6 HEX_CC_implicit.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Counter current heat exchanger with discretized
% differential equations
% @input: t = time, w = system variables, data = parameters,
%         dynVar = dynamic variables values at previous time step
%         dt = Time Step [s]
% @output: F = residuals, G = Generalized Derivative Matrix,
%          Q = Heat transfer, F_in_2 = Inlet flow rate cold Side
%          F_in = Inlet flow rate hot side,
%          T_in_2 = Inlet temperature cold side
%*****************************************************************

function [F,G,Q,F_in_2,F_in,T_in_2] = HEX_CC_implicit(t,w,data,dynVar,dt)
%% Unpacking parameters
par = data.par;
F_in = par.F_in;            % Inlet flow rate on hot side [kmol/s]
F_in_2 = par.F_in_2;        % Inlet flow rate on cold side [kmol/s]
R = par.R;                  % Gas constant [MJ/kmol*kK]
V_T = par.V_T;              % Total volume [m^3]
P_0 = par.P_0;              % Outlet pressure [MPa]
Cp_V_1 = par.Cp_V(1);       % Vapor heat capacity component 1 [MJ/kmol*kK]
Cp_L_1 =  par.Cp_L(1);      % Liquid heat capacity component 1 [MJ/kmol*kK]
Cp_V_2 = par.Cp_V(2);       % Vapor heat capacity component 2 [MJ/kmol*kK]
Cp_L_2 =  par.Cp_L(2);      % Liquid heat capacity component 1 [MJ/kmol*kK]
h_vap_1 = par.h_vap(1);     % Heat of vaporization component 1 [MJ/kmol]
h_vap_2 = par.h_vap(2);     % Heat of vaporization component 2 [MJ/kmol]
A_1 = par.Antoine(1,1);     % Antoine Parameter component 1
B_1 = par.Antoine(2,1);     % Antoine Parameter component 1
C_1 = par.Antoine(3,1);     % Antoine Parameter component 1
A_2 = par.Antoine(1,2);     % Antoine Parameter component 2
B_2 = par.Antoine(2,2);     % Antoine Parameter component 2
C_2 = par.Antoine(3,2);     % Antoine Parameter component 2
rho_1 = par.rho(1);         % Molar density component 1 [Mmol/m^3]
rho_2 = par.rho(2);         % Molar density component 2 [Mmol/m^3]
z_1 = par.z(1);             % Molar fraction hot side (Should be 1)
z_2 = par.z(2);             % Molar fraction cold side (Should be 1)
h_in = par.h_in;            % Inlet molar enthalpy hot side [MJ/kmol]
h_in_2 = par.h_in_2;        % Inlet molar enthalpy hot side [MJ/kmol]
T_in_2 = par.T_in_2;        % Inlet temperature cold side [kK]
```

```
M = par.M;                  % Number of flash tanks in series
C0 = par.C0;                % Compressability factor [1/MPa]
T_ref = par.T_ref;          % Reference temperature [kK]
c_V = par.c_V*sqrt(M);      % Vapor valve coefficient [kmol/bar^.5 s]
c_L = par.c_L*sqrt(M);      % Liquid valve coefficient [kmol/bar^.5 s]
UA = data.par.UA;           % Heat transfer coefficent times area [MW/kK]

%% Unpacking hot side variables
L = length(w);
w = valder(w,eye(length(w)));
M_1 = w(1:M);               % Component Molar hold-up [kmol]
U = w(1*M+1:2*M);           % Internal Energy [MJ]
F_V = w(2*M+1:3*M);         % Vapor flow rate [kmol/s]
F_L = w(3*M+1:4*M);         % Liquid flow rate [kmol/s]
M_V = w(4*M+1:5*M);         % Vapor molar hold-up [kmol]
P = w(5*M+1:6*M);           % Pressure [MPa]
T = w(6*M+1:7*M);           % Temperature [kK]
V_L = w(7*M+1:8*M);         % Liquid Volume [m^3]
x_1 = w(8*M+1:9*M);         % Liquid mole fraction
y_1 = w(9*M+1:10*M);        % Vapor mole fraction
H = w(10*M+1:11*M);         % Enthalpy [MJ]
h_L = w(11*M+1:12*M);       % Molar enthalpy in liquid phase [MJ/kmol]
h_V = w(12*M+1:13*M);       % Molar enthalpy in vapor phase [MJ/kmol]
V_V = w(13*M+1:14*M);       % Vapor volume [m^3]
rho_L = w(14*M+1:15*M);     % Liquid molar density [Mmol/kK]
M_L = w(15*M+1:16*M);       % Liquid molar hold-up [kmol]
k_1 = w(16*M+1:17*M);       % Vapor-liquid equilibrium coefficent
Ps_1 = w(17*M+1:18*M);      % Saturation pressure

%% Unpacking hot side variables
M_1_2 = w(18*M+1:19*M);     % Component Molar hold-up [kmol]
U_2 = w(19*M+1:20*M);       % Internal Energy [MJ]
F_V_2 = w(20*M+1:21*M);     % Vapor flow rate [kmol/s]
F_L_2 = w(21*M+1:22*M);     % Liquid flow rate [kmol/s]
M_V_2 = w(22*M+1:23*M);     % Vapor molar hold-up [kmol]
P_2 = w(23*M+1:24*M);       % Pressure [MPa]
T_2 = w(24*M+1:25*M);       % Temperature [kK]
V_L_2 = w(25*M+1:26*M);     % Liquid Volume [m^3]
x_1_2 = w(26*M+1:27*M);     % Liquid mole fraction
y_1_2 = w(27*M+1:28*M);     % Vapor mole fraction
H_2 = w(28*M+1:29*M);       % Enthalpy [MJ]
h_L_2 = w(29*M+1:30*M);     % Molar enthalpy in liquid phase [MJ/kmol]
h_V_2 = w(30*M+1:31*M);     % Molar enthalpy in vapor phase [MJ/kmol]
V_V_2 = w(31*M+1:32*M);     % Vapor volume [m^3]
rho_L_2 = w(32*M+1:33*M);   % Liquid molar density [Mmol/kK]
M_L_2 = w(33*M+1:34*M);     % Liquid molar hold-up [kmol]
k_1_2 = w(34*M+1:35*M);     % Vapor-liquid equilibrium coefficent
Ps_1_2 = w(35*M+1:36*M);    % Saturation pressure

%% Preallocating memory
f1 = valder(zeros(M,1),zeros(M,L));
f2 = valder(zeros(M,1),zeros(M,L));
f3 = valder(zeros(M,1),zeros(M,L));
f4 = valder(zeros(M,1),zeros(M,L));
f5 = valder(zeros(M,1),zeros(M,L));
f6 = valder(zeros(M,1),zeros(M,L));
f7 = valder(zeros(M,1),zeros(M,L));
```

```
f8 = valder(zeros(M,1),zeros(M,L));
f9 = valder(zeros(M,1),zeros(M,L));
f10 = valder(zeros(M,1),zeros(M,L));
f11 = valder(zeros(M,1),zeros(M,L));
f12 = valder(zeros(M,1),zeros(M,L));
f13 = valder(zeros(M,1),zeros(M,L));
f14 = valder(zeros(M,1),zeros(M,L));
f15 = valder(zeros(M,1),zeros(M,L));
f16 = valder(zeros(M,1),zeros(M,L));
f17 = valder(zeros(M,1),zeros(M,L));
f18 = valder(zeros(M,1),zeros(M,L));
f19 = valder(zeros(M,1),zeros(M,L));
f20 = valder(zeros(M,1),zeros(M,L));
f21 = valder(zeros(M,1),zeros(M,L));
f22 = valder(zeros(M,1),zeros(M,L));
f23 = valder(zeros(M,1),zeros(M,L));
f24 = valder(zeros(M,1),zeros(M,L));
f25 = valder(zeros(M,1),zeros(M,L));
f26 = valder(zeros(M,1),zeros(M,L));
f27 = valder(zeros(M,1),zeros(M,L));
f28 = valder(zeros(M,1),zeros(M,L));
f29 = valder(zeros(M,1),zeros(M,L));
f30 = valder(zeros(M,1),zeros(M,L));
f31 = valder(zeros(M,1),zeros(M,L));
f32 = valder(zeros(M,1),zeros(M,L));
f33 = valder(zeros(M,1),zeros(M,L));
f34 = valder(zeros(M,1),zeros(M,L));
f35 = valder(zeros(M,1),zeros(M,L));
f36 = valder(zeros(M,1),zeros(M,L));

%% Defining how some of the input will change during simulation

% Changing the temperature in the inlet on the cold side
if t < 5
    T_in_2 = T_in_2;
    h_in_2 = h_in_2;
elseif t < 70
    T_in_2 = T_in_2 - 0.0003*(t-5);
    h_in_2 = par.Cp_L(2)*(T_in_2-T_ref);
else
    T_in_2 = T_in_2 - 0.0003*65;
    h_in_2 = par.Cp_L(2)*(T_in_2-T_ref);
end

% Changing the inlet flow rate on the cold side
if t < 100
    F_in_2 = F_in_2;
elseif t < 150
    F_in_2 = F_in_2 + 0.001*(t-100);
else
    F_in_2 = 0.15;
end


M_1_0 = dynVar(1:M);
U_0 = dynVar(M+1:2*M);
M_1_2_0 = dynVar(2*M+1:3*M);
```

```
U_2_0 = dynVar(3*M+1:4*M);
%% Differential equations
f1(1) = M_1(1)  - (M_1_0(1)  + (F_in*z_1 - (max2(0,F_L(1))*x_1(1)...
    +  max2(0,F_V(1))*y_1(1)) - (min2(0,F_L(1))*x_1(2)...
    +  min2(0,F_V(1))*y_1(2)))*dt);
f2(1) = U(1) - (U_0(1) + (F_in*h_in - (max2(0,F_L(1))*h_L(1)...
    + max2(0,F_V(1))*h_V(1)) - min2(0,F_L(1))*h_L(2)...
    + min2(0,F_V(1))*h_V(2) + UA*(T_2(M)-T(1)))*dt);
f19(1) = M_1_2(1)  - (M_1_2_0(1)  + (F_in_2*z_2 -...
    (max2(0,F_L_2(1))*x_1_2(1) +  max2(0,F_V_2(1))*y_1_2(1))...
    - (min2(0,F_L_2(1))*x_1_2(2) +  min2(0,F_V_2(1))*y_1_2(2)))*dt);
f20(1) = U_2(1) - (U_2_0(1) + (F_in_2*h_in_2 - (max2(0,F_L_2(1))*h_L_2(1)...
    + max2(0,F_V_2(1))*h_V_2(1)) - (min2(0,F_L_2(1))*h_L_2(2)...
    + min2(0,F_V_2(1))*h_V_2(2)) + UA*(T(M)-T_2(1)))*dt);
Q(1) = getVal(UA*(T_2(M)-T(1)));
for j = 2:M-1
    f1(j) = M_1(j) - (M_1_0(j) + (max2(0,F_L(j-1)*x_1(j-1)...
        + F_V(j-1)*y_1(j-1)) - max2(0,F_L(j)*x_1(j) + F_V(j)*y_1(j))...
        + min2(0,F_L(j-1)*x_1(j) + F_V(j-1)*y_1(j))...
        - min2(0,F_L(j)*x_1(j+1) + F_V(j)*y_1(j+1)))*dt);
    f2(j) = U(j) - (U_0(j) + ((max2(0,F_L(j-1))*h_L(j-1)...
        + max2(0,F_V(j-1))*h_V(j-1)) - (max2(0,F_L(j))*h_L(j)...
        + max2(0,F_V(j))*h_V(j)) + (min2(0,F_L(j-1))*h_L(j)...
        + min2(0,F_V(j-1))*h_V(j)) - (min2(0,F_L(j))*h_L(j+1)...
        + min2(0,F_V(j))*h_V(j+1))  + UA*(T_2(M-j+1)-T(j)))*dt);
    f19(j) = M_1_2(j) - (M_1_2_0(j) + (max2(0,F_L_2(j-1)*x_1_2(j-1)...
        + F_V_2(j-1)*y_1_2(j-1)) - max2(0,F_L_2(j)*x_1_2(j)...
        + F_V_2(j)*y_1_2(j)) + min2(0,F_L_2(j-1)*x_1_2(j)...
        + F_V_2(j-1)*y_1_2(j)) - min2(0,F_L_2(j)*x_1_2(j+1)...
        + F_V_2(j)*y_1_2(j+1)))*dt);
    f20(j) = U_2(j) - (U_2_0(j) + ((max2(0,F_L_2(j-1))*h_L_2(j-1)...
        + max2(0,F_V_2(j-1))*h_V_2(j-1)) - (max2(0,F_L_2(j))*h_L_2(j)...
        + max2(0,F_V_2(j))*h_V_2(j)) + (min2(0,F_L_2(j-1))*h_L_2(j)...
        + min2(0,F_V_2(j-1))*h_V_2(j)) - (min2(0,F_L_2(j))*h_L_2(j+1)...
        + min2(0,F_V_2(j))*h_V_2(j+1))  + UA*(T(M-j+1)-T_2(j)))*dt);
    Q(j) = getVal(UA*(T_2(M-j+1)-T(j)));
end
f1(M) = M_1(M) - (M_1_0(M) + (max2(0,F_L(M-1)*x_1(M-1)...
    + F_V(M-1)*y_1(M-1)) - max2(0,F_L(M)*x_1(M) + F_V(M)*y_1(M))...
    + min2(0,F_L(M-1)*x_1(M) + F_V(M-1)*y_1(M)))*dt);
f2(M) = U(M) - (U_0(M) + ((max2(0,F_L(M-1))*h_L(M-1)...
    + max2(0,F_V(M-1))*h_V(M-1)) - (max2(0,F_L(M))*h_L(M)...
    + max2(0,F_V(M))*h_V(M)) + (min2(0,F_L(M-1))*h_L(M)...
    + min2(0,F_V(M-1))*h_V(M))  + UA*(T_2(1)-T(M)))*dt);
f19(M) = M_1_2(M) - (M_1_2_0(M) + (max2(0,F_L_2(M-1)*x_1_2(M-1)...
    + F_V_2(M-1)*y_1_2(M-1)) - max2(0,F_L_2(M)*x_1_2(M)...
    + F_V_2(M)*y_1_2(M)) + min2(0,F_L_2(M-1)*x_1_2(M)...
    + F_V_2(M-1)*y_1_2(M)))*dt);
f20(M) = U_2(M) - (U_2_0(M) + ((max2(0,F_L_2(M-1))*h_L_2(M-1)...
    + max2(0,F_V_2(M-1))*h_V_2(M-1)) - (max2(0,F_L_2(M))*h_L_2(M)...
    + max2(0,F_V_2(M))*h_V_2(M)) + (min2(0,F_L_2(M-1))*h_L_2(M)...
    + min2(0,F_V_2(M-1))*h_V_2(M))  + UA*(T(1)-T_2(M)))*dt);
Q(M) = getVal(UA*(T_2(1)-T(M)));
%% Algebraic equations
for j = 1:M
    f3(j) = M_1(j) - (M_L(j)*x_1(j) + M_V(j)*y_1(j));
    f4(j) = (M_1(j)) - (M_L(j) + M_V(j));
```

```
    f5(j) = H(j) - (M_L(j)*h_L(j) + M_V(j)*h_V(j));
    f6(j) = H(j) - (U(j) + P(j)*V_T/M);
    f7(j) = y_1(j) - k_1(j)*x_1(j);
    f8(j) = midobj(M_V(j)/(M_V(j)+M_L(j)),x_1(j) - y_1(j),...
        (M_V(j)/(M_V(j)+M_L(j)))-1);
    f9(j) = V_T/M - (V_L(j) + V_V(j));
    f10(j) = P(j)*V_V(j) - M_V(j)*R*T(j);
    f11(j) = V_L(j)*(rho_L(j)*1e3) - M_L(j);
    if j < M
        f12(j) = F_V(j)*V_T/M - c_V*V_V(j)*((P(j)-P(j+1))...
            /sqrt(abs((P(j)-P(j+1)))+1e-10));
        f13(j) = F_L(j)*V_T/M - c_L*V_L(j)*((P(j)-P(j+1))...
            /sqrt(abs((P(j)-P(j+1)))+1e-10));
    else
        f12(j) = F_V(j)*V_T/M - c_V*V_V(j)*max2(0,(P(j)-P_0)...
            /sqrt(abs((P(j)-P_0))+1e-10));
        f13(j) = F_L(j)*V_T/M - c_L*V_L(j)*max2(0,((P(j)-P_0)...
            /sqrt(abs((P(j)-P_0))+1e-10)));
    end
    f14(j) = rho_L(j)*(x_1(j)) - (rho_1*(1+C0*(P(j)-0.1)));
    f15(j) = h_L(j) - (x_1(j)*Cp_L_1)*(T(j)-T_ref);
    f16(j) = h_V(j) - (y_1(j)*(h_vap_1 + Cp_V_1*(T(j)-T_ref)));
    f17(j) = k_1(j)*P(j) - Ps_1(j);
    f18(j) = Ps_1(j) - 10^(A_1 - B_1/(T(j)*1e3+C_1))/10;

    f21(j) = M_1_2(j) - (M_L_2(j)*x_1_2(j) + M_V_2(j)*y_1_2(j));
    f22(j) = (M_1_2(j)) - (M_L_2(j) + M_V_2(j));
    f23(j) = H_2(j) - (M_L_2(j)*h_L_2(j) + M_V_2(j)*h_V_2(j));
    f24(j) = H_2(j) - (U_2(j) + P_2(j)*V_T/M);
    f25(j) = y_1_2(j) - k_1_2(j)*x_1_2(j);
    f26(j) = midobj(M_V_2(j)/(M_V_2(j)+M_L_2(j)),x_1_2(j) - y_1_2(j),...
        (M_V_2(j)/(M_V_2(j)+M_L_2(j)))-1);
    f27(j) = V_T/M - (V_L_2(j) + V_V_2(j));
    f28(j) = P_2(j)*V_V_2(j) - M_V_2(j)*R*T_2(j);
    f29(j) = V_L_2(j)*(rho_L_2(j)*1e3) - M_L_2(j);
    if j < M
        f30(j) = F_V_2(j)*V_T/M - c_V*V_V_2(j)*((P_2(j)-P_2(j+1))...
            /sqrt(abs((P_2(j)-P_2(j+1)))+1e-10));
        f31(j) = F_L_2(j)*V_T/M - c_L*V_L_2(j)*((P_2(j)-P_2(j+1))...
            /sqrt(abs((P_2(j)-P_2(j+1)))+1e-10));
    else
        f30(j) = F_V_2(j)*V_T/M - c_V*V_V_2(j)*max2(0,(P_2(j)-P_0)...
            /sqrt(abs((P_2(j)-P_0))+1e-10));
        f31(j) = F_L_2(j)*V_T/M - c_L*V_L_2(j)*max2(0,((P_2(j)-P_0)...
            /sqrt(abs((P_2(j)-P_0))+1e-10)));
    end

    f32(j) = rho_L_2(j)*(x_1_2(j)) - (rho_2*(1+C0*(P_2(j)-0.1)));
    f33(j) = h_L_2(j) - (x_1_2(j)*Cp_L_2)*(T_2(j)-T_ref);
    f34(j) = h_V_2(j) - (y_1_2(j)*(h_vap_2 + Cp_V_2*(T_2(j)-T_ref)));
    f35(j) = k_1_2(j)*P_2(j) - Ps_1_2(j);
    f36(j) = Ps_1_2(j) - 10^(A_2 - B_2/(T_2(j)*1e3+C_2))/10;

end


%% Extracting the values and the derivatives from the valder objects
```

```
F = [getVal(f1);getVal(f2);getVal(f3);getVal(f4);getVal(f5);getVal(f6);...
    getVal(f7);getVal(f8);getVal(f9);getVal(f10);getVal(f11);...
    getVal(f12);getVal(f13);getVal(f14);getVal(f15);getVal(f16);...
    getVal(f17);getVal(f18);getVal(f19);getVal(f20);getVal(f21);...
    getVal(f22);getVal(f23);getVal(f24);getVal(f25);getVal(f26);...
    getVal(f27);getVal(f28);getVal(f29);getVal(f30);getVal(f31);...
    getVal(f32);getVal(f33);getVal(f34);getVal(f35);getVal(f36)];

G = [getDer(f1);getDer(f2);getDer(f3);getDer(f4);getDer(f5);getDer(f6);...
    getDer(f7);getDer(f8);getDer(f9);getDer(f10);getDer(f11);...
    getDer(f12);getDer(f13);getDer(f14);getDer(f15);getDer(f16);...
    getDer(f17);getDer(f18);getDer(f19);getDer(f20);getDer(f21);...
    getDer(f22);getDer(f23);getDer(f24);getDer(f25);getDer(f26);...
    getDer(f27);getDer(f28);getDer(f29);getDer(f30);getDer(f31);...
    getDer(f32);getDer(f33);getDer(f34);getDer(f35);getDer(f36)];
end
```

D.3.7 implicitSolverFull_CC.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Implicit Euler integrator for the counter current heat
% exchanger model.
% @input: w0 = initial conditions, data = parameters, dt = time step,
% tspan = [t_start, t_end]
% @output: t_vec = time vector, w = System variables matrix, Q = heat
% transfer, F_in_2 = Inlet flow rate cold side, F_in = Flow Rate hot side,
% T_in_2 = Inlet temperature cold side
%*****************************************************************

function [t_vec,w,Q,F_in_2,F_in,T_in_2] = implicitSolverFull_CC...
    (w0,data,dt,tspan)

%% Solver settings
options_fsolve = optimoptions(@fsolve,'Display','iter',...
     'MaxIterations',2e3,'MaxFunEvals',1e10,...
     'stepTolerance',1e-13,'FunctionTolerance',1e-13,...
     'specifyObjectiveGradient', true,...
     'OptimalityTolerance',1e-13, 'Algorithm','levenberg-marquardt');
%% Initial values and preallocating memory
t = tspan(1);
t_vec = tspan(1);
M = data.par.M;
w = zeros(length(w0),length(t));
w(:,1) = w0;
dynVar = [w0(1:2*M);w0(18*M+1:20*M)];
[~,~,Q(:,1),F_in_2(1),F_in(1),T_in_2(1)] =...
    HEX_CC_implicit(t,w0,data,dynVar,dt);

i = 1;
while t + dt <= tspan(2)
    i = i+1;                % Index counter
```

```
    t = t + dt;              % Next Time
    t_vec = [t_vec t];       % Adding next time in time vector
    w0 = w(:,i-1);           % Initial guess equal to previous solution
    dynVar = [w(1:2*M,i-1);w(18*M+1:20*M,i-1)]; % Dynamic variable values
    w(:,i) = fsolve(@(w) HEX_CC_implicit...
        (t,w,data,dynVar,dt),w0,options_fsolve);    % Solving model
    [~,~,Q(:,i),F_in_2(i),F_in(i),T_in_2(i)] =...
        HEX_CC_implicit(t,w(:,i),data,dynVar,dt);   % Saving values
end
end
```

D.3.8 initialGuesses_gases.m

```
%*********************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Returns inital guesses and boundaries for a Flash tank
% given the parameters and that the flash is in the vapor-only regime
% @input: par = parameters
% @output: w0 = Initial guesses, lb = lower boundaries, ub = upper
% boundaries
%*********************************************************************

function [w0,lb,ub] = initialGuesses_gases(par)

% M_1 = w(1);            % Component molar hold-up [kmol]
w0(1) = 0.5;
lb(1) = 0;
ub(1) = inf;
% U = w(2);              % Internal energy [MJ]
w0(2) = -1;
lb(2) = -inf;
ub(2) = inf;
% F_V = w(3);            % Vapor flow rate [kmol/s]
w0(3) = par.F_in;
lb(3) = par.F_in;
ub(3) = par.F_in;
% F_L = w(4);            % Liquid flow rate [kmol/s]
w0(4) = 0;
lb(4) = 0;
ub(4) = 0;
% M_V = w(5);            % Vapor molar hold-up [kmol]
w0(5) = 0.1;
lb(5) = 0;
ub(5) = inf;
% P = w(6);              % Pressure [MPa]
w0(6) = 0.11;
lb(6) = 0;
ub(6) = inf;
% T = w(7);              % Temperature [kK]
w0(7) = par.T_in;
lb(7) = par.T_in;
ub(7) = par.T_in;
```

```
% V_L = w(8);            % Liquid volume [m^3]
w0(8) = 0;
lb(8) = 0;
ub(8) = 0;
% x_1 = w(9);            % Liquid mole fraction
w0(9) = 0.5;
lb(9) = 0;
ub(9) = inf;
% y_1 = w(10);           % Vapor mole fraction
w0(10) = 1;
lb(10) = 1;
ub(10) = 1;
% H = w(11);             % Enthalpy [MJ]
w0(11) = 1;
lb(11) = -inf;
ub(11) = inf;
% h_L = w(12);           % Liquid molar enthalpy [MJ/kmol]
w0(12) = par.h_in;
lb(12) = -inf;
ub(12) = inf;
% h_V = w(13);           % Vapor molar enthalpy [MJ/kmol]
w0(13) = par.h_in;
lb(13) = par.h_in;
ub(13) = par.h_in;
% V_V = w(14);           % Vapor volume [m^3]
w0(14) = par.V_T;
lb(14) = par.V_T;
ub(14) = par.V_T;
% rho_L = w(15);         % Liquid molar density [Mmol/m^3]
w0(15) = 0.5;
lb(15) = 0;
ub(15) = inf;
% M_L = w(16);           % Liquid molar hold-up [kmol]
w0(16) = 0;
lb(16) = 0;
ub(16) = 0;
% K_1 = w(17);           % Vapor-liquid equilibrium coefficient
w0(17) = 1;
lb(17) = 0;
ub(17) = inf;
% Ps_1 = w(18);          % Saturation Pressure [MPa]
w0(18) = 0.1;
lb(18) = 0;
ub(18) = inf;
w0 = w0';
end
```

D.3.9 initialGuesses_liquid.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Returns inital guesses and boundaries for a Flash tank
```

```
% given the parameters and that the flash is in the liquid-only regime
% @input: par = parameters
% @output: w0 = Initial guesses, lb = lower boundaries, ub = upper
% boundaries
%*****************************************************************


function [w0,lb,ub] = initialGuesses_liquid(par)

% M_1 = w(1);           % Component molar hold-up [kmol]
w0(1) = 0.5;
lb(1) = 0;
ub(1) = inf;
% U = w(2);             % Internal energy [MJ]
w0(2) = -1;
lb(2) = -inf;
ub(2) = inf;
% F_V = w(3);           % Vapor flow rate [kmol/s]
w0(3) = 0;
lb(3) = 0;
ub(3) = 0;
% F_L = w(4);           % Liquid flow rate [kmol/s]
w0(4) = par.F_in;
lb(4) = par.F_in;
ub(4) = par.F_in;
% M_V = w(5);           % Vapor molar hold-up [kmol]
w0(5) = 0;
lb(5) = 0;
ub(5) = 0;
% P = w(6);             % Pressure [MPa]
w0(6) = 0.12;
lb(6) = 0.1;
ub(6) = inf;
% T = w(7);             % Temperature [kK]
w0(7) = par.T_in;
lb(7) = par.T_in;
ub(7) = par.T_in;
% V_L = w(8);           % Liquid volume [m^3]
w0(8) = par.V_T;
lb(8) = par.V_T;
ub(8) = par.V_T;
% x_1 = w(9);           % Liquid mole fraction
w0(9) = 1;
lb(9) = 1;
ub(9) = 1;
% y_1 = w(10);          % Vapor mole fraction
w0(10) = 0.5;
lb(10) = 0;
ub(10) = inf;
% H = w(11);            % Enthalpy [MJ]
w0(11) = 1;
lb(11) = -inf;
ub(11) = inf;
% h_L = w(12);          % Liquid molar enthalpy [MJ/kmol]
w0(12) = par.h_in;
lb(12) = par.h_in;
ub(12) = par.h_in;
```

```
% h_V = w(13);           % Vapor molar enthalpy [MJ/kmol]
w0(13) = par.h_in;
lb(13) = 0;
ub(13) = inf;
% V_V = w(14);           % Vapor volume [m^3]
w0(14) = 0;
lb(14) = 0;
ub(14) = 0;
% rho_L = w(15);         % Liquid molar density [Mmol/m^3]
w0(15) = 0.5;
lb(15) = 0;
ub(15) = inf;
% M_L = w(16);           % Liquid molar hold-up [kmol]
w0(16) = 1;
lb(16) = 0;
ub(16) = inf;
% K_1 = w(17);           % Vapor-liquid equilibrium coefficient
w0(17) = 1;
lb(17) = 0;
ub(17) = inf;
% Ps_1 = w(18);          % Saturation Pressure [MPa]
w0(18) = 0.1;
lb(18) = 0;
ub(18) = inf;
w0 = w0';
end
```

D.3.10 parameters_Methanol.m

```
%*************************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Parameters for HEX with methanol as inlet in vapor phase
%*************************************************************************

function par = parameters_Methanol()
%https://webbook.nist.gov/cgi/cbook.cgi?ID=C67561&Mask=4&Type=ANTOINE&Plot=on
%https://webbook.nist.gov/cgi/cbook.cgi?ID=C7732185&Mask=4&Type=ANTOINE&Plot=on
%% Parameters
% Antoine Parameters log(p_sat[bar]) = A - B/(T[K]+C)
A = 5.15853;
B = 1569.613;
C = -34.846;
par.T_in = 0.410;       % Inlet Temperature [kK]
par.F_in = 0.1;         % Inlet flow rate hot side [kmol/s]
par.Q = 0;              % Heat transfer [MW]
par.Antoine = [A;B;C];  % Antoine parameters
par.h_vap = 35.210;     % Heat of vaporization [MJ/kmol]
par.Cp_V = 44.06;       % Vapor Heat Capacity [MJ/kmol*kK]
par.Cp_L = 81.08;       % Liquid Heat Capacity [MJ/kmol*kK]
par.R = 8.314;          % Gas constant [MJ/(kK * kmol)]
par.P_0 = 0.1;          % Outlet Pressure [MPa]
par.V_T = 0.2;          % Total volume [m^3]
```

```
par.rho = 0.792/32.04;  % Liquid molar density[Mmol/m^3]
par.c_V = 1;            % Valve coefficient vapor[kmol/(bar^(-0.5) s)]
par.c_L = 5;            % Valve coefficient liquid [kmol/(bar^(-0.5) s)]
par.z = 1;              % Inlet composition
par.C0 = 3*145.0377e-6; % Liquid compressability factor [1/MPa]
par.M = 3;              % Nr of Flash Tanks in series
par.T_ref = 0.29815;    % Reference temperature [kK]
%% Calculation of inlet enthalpy [MJ/kmol]
% This calculation assumes that the inlet  is vapor
par.h_in = par.h_vap(1) + par.Cp_V(1)*(par.T_in-par.T_ref);
end
```

D.3.11 parameters_water.m

```
%**************************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Parameters for HEX with water as inlet in liquid phase
%**************************************************************************

function par = parameters_water()
%https://webbook.nist.gov/cgi/cbook.cgi?ID=C67561&Mask=4&Type=ANTOINE&Plot=on
%https://webbook.nist.gov/cgi/cbook.cgi?ID=C7732185&Mask=4&Type=ANTOINE&Plot=on
%% Parameters
% Antoine Parameters log(p_sat[bar]) = A - B/(T[K]+C)
A = 4.6543;
B = 1435.264;
C = -64.848;
par.T_in = 0.300;       % Inlet Temperature [kK]
par.F_in = 0.1;         % Inlet flow rate hot side [kmol/s]
par.Q = 0;              % Heat transfer [MW]
par.Antoine = [A;B;C];  % Antoine parameters
par.h_vap = 40.660;     % Heat of vaporization [MJ/kmol]
par.Cp_V =  35;         % Vapor Heat Capacity [MJ/kmol*kK]
par.Cp_L =  75;         % Liquid Heat Capacity [MJ/kmol*kK]
par.R = 8.314;          % Gas constant [MJ/(kK * kmol)]
par.P_0 = 0.1;          % Outlet Pressure [MPa]
par.V_T = 0.2;          % Total volume [m^3]
par.rho = 1/18.016;     % Liquid molar density[Mmol/m^3]
par.c_V = 1;            % Valve coefficient vapor[kmol/(bar^(-0.5) s)]
par.c_L = 5;            % Valve coefficient liquid [kmol/(bar^(-0.5) s)]
par.z = 1;              % Inlet composition
par.C0 = 3*145.0377e-6; % Liquid compressability factor [1/MPa]
par.M = 3;              % Nr of Flash Tanks in series
par.T_ref = 0.29815;    % Reference temperature [kK]
%% Calculation of inlet enthalpy [MJ/kmol]
% This calculation assumes that the inlet  is liquid
par.h_in = par.Cp_L*(par.T_in-par.T_ref);
end
```

D.3.12 parameters_CC.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Process Systems Engineering, NTNU
% @since: 03-12-2018
% @requires: MATLAB R2017b (not tested in other releases)
% @description: Parameters for counter current HEX
%*****************************************************************

function par = parameters_CC()
%% Literature
% https://webbook.nist.gov/cgi/cbook.cgi?ID=C67561&Mask=4&Type=ANTOINE&Plot=on
% https://webbook.nist.gov/cgi/cbook.cgi?ID=C7732185&Mask=4&Type=ANTOINE&Plot=on
% http://www.personal.utulsa.edu/~geoffrey-price/Courses/ChE7023/
% HeatCapacity-HeatOfFormation.pdf
%% Parameters
% Antoine Parameters log(p_sat[bar]) = A - B/(T[K]+C)
A = [5.15853 4.6543];
B = [1569.613 1435.264];
C = [-34.846 -64.848];

par.T_in = 0.410;       % Temperature inlet hot side [kK]
par.T_in_2 = 0.300;     % Temperature inlet cold side [kK]
par.F_in = 0.1;         % Inlet flow rate hot side [kmol/s]
par.F_in_2 = 0.1;       % Inlet flow rate cold side [kmol/s]
par.Antoine = [A;B;C];  % Antoine parameters
par.h_vap = [35.210 40.660]; % Heat of vaporization [MJ/kmol]
par.Cp_V = [44.06, 35]; % Vapor Heat Capacity [MJ/kmol*kK]
par.Cp_L = [81.08, 75]; % Liquid Heat Capacity [MJ/kmol*kK]
par.R = 8.314;          % Gas constant [MJ/(kK * kmol)]
par.P_0 = 0.1;          % Outlet Pressure [MPa]
par.V_T = 0.2;          % Total volume [m^3]
par.rho = [0.792/32.04, 1/18.016]; % Liquid molar density[Mmol/m^3]
par.c_V = 1;            % Valve coefficient vapor[kmol/(bar^(-0.5) s)]
par.c_L = 5;            % Valve coefficient liquid [kmol/(bar^(-0.5) s)]
par.z = [1 1];          % Composition inlet [Hot Cold]
par.C0 = 3*145.0377e-6; % Liquid compressability factor [1/MPa]
par.M = 3;              % Nr of Flash Tanks in series
par.T_ref = 0.29815;    % Reference temperature [kK]
par.UA = 4;             % Heat transfer coefficient times area [MW/kK]

%% Calculation of inlet enthalpy [MJ/kmol]
% This calculation assumes that the inlet on the hot side is vapor and that
% the cold side is liquid.
par.h_in = par.h_vap(1) + par.Cp_V(1)*(par.T_in-par.T_ref);
par.h_in_2 = par.Cp_L(2)*(par.T_in_2-par.T_ref);
end
```