NTNU
Norwegian University of
Science and Technology

Nonsmooth analysis of connected oil well system

Marius Reed

December 19, 2017

TKP4580 - Chemical Engineering, Specialization Project
Department of chemical engineering
Norwegian University of Science and Technology

Supervisor 1: Johannes Jschke
Supervisor 2: Marlene L. Lund

Summary

The objective of the project was to show that recent development within nonsmooth analysis, with special focus on the lexicographic derivative, make it possible to formulate a model that allow bi-directional flow without use of logical statements or smooth approximations. This was illustrated by formulating a nonsmooth model of a connected oil well system consisting of three wells and two risers. First some preliminary theory about nonsmooth analysis, including convexity, generalized derivatives and piecewise differentiable functions, was introduced. Afterwards, two solvers were suggested and explained. With the mathematical introduction given, the development of the model of the connected oil-well system was described, with focus on the nonsmooth formulations.

The system was solved using the Levenberg-Marquardt algorithm (LMA) as well as a Newton-type method. The generalized derivatives were computed by using automatic forward differentiation, exploiting the strict calculus rules that the lexicographic directional derivatives obey. The solutions showed that the proposed formulation fulfilled the objective of bi-directional flow, meaning that a nonsmooth approach is suitable to model such systems. In addition, the LMA proved to be the most robust solver of the two proposed solvers, while the Newton-type method converged faster close to the nonsmooth point.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

LD	Lexicographic directional
LMA	Levenberg-Marquardt algorithm
OOP	Object-oriented programming

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
$\mathbb{N}$	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
$\|\cdot\|$	Unspecified norm
$\partial \mathbf{f}$	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
$\subset$	Subset 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
det$\mathbf{M}$	Determinant of $\mathbf{M}$

Model nomenclature

$\hat{m}_g$	Mass flow rate of gas	$\mathrm{kg{\cdot}s^{-1}}$
$\hat{m}_o$	Mass flow rate of oil	$\mathrm{kg{\cdot}s^{-1}}$
$\hat{m}_{w,i,g}$	Mass flow rate of gas in well *i*	$\mathrm{kg{\cdot}s^{-1}}$
$\hat{m}_{w,i,o}$	Mass flow rate of oil in well *i*	$\mathrm{kg{\cdot}s^{-1}}$
$\hat{m}_{w,i,w}$	Mass flow rate of water in well *i*	$\mathrm{kg{\cdot}s^{-1}}$
$\hat{m}_w$	Mass flow rate of water	$\mathrm{kg{\cdot}s^{-1}}$
Φ	Friction	J
ρ	Density	$\mathrm{kg{\cdot}m^{-3}}$
ρ_g^{ig}	Density of ideal gas	$\mathrm{kg{\cdot}m^{-3}}$
ρ_{mix}	Density of pseudo fluid	$\mathrm{kg{\cdot}m^{-3}}$
ρ_o	Density of oil	$\mathrm{kg{\cdot}m^{-3}}$
ρ_w	Density of water	$\mathrm{kg{\cdot}m^{-3}}$
A	Area	$\mathrm{m^2}$
C_d	Valve constant	$(\,\mathrm{kg \cdot m^{-1} \cdot bar^{-1} \cdot s^{-2}}\,)^{\,-0.5}$
F^i	Flow rate in flow i	kg s$^-$1

g	Gravitational constant	$\mathrm{m{\cdot}s^{-2}}$
h	Height	m
$k_{g,i}$	Transport coefficient for gas in well *i*	$\mathrm{kg{\cdot}bar^{-4}{\cdot}s^{-1}}$
$k_{o,i}$	Transport coefficient for oil in well *i*	$\mathrm{kg{\cdot}bar^{-2}{\cdot}s^{-1}}$
$k_{w,i}$	Transport coefficient for water in well *i*	$\mathrm{kg{\cdot}bar^{-2}{\cdot}s^{-1}}$
m	mass	kg
M_g	Molar mass of gas	$\mathrm{kg{\cdot}m^{-3}}$
p	Pressure	bar
$p_{r,i}$	Reservoir pressure around well *i*	bar
$p_{wf,i}$	Pressure at well inflow in well *i*	bar
$p_{wh,i}$	Pressure at well head in well *i*	bar
R	Gas constant	$\mathrm{m^3 \cdot bar \cdot kmol^{-1} \cdot K^{-1}}$
T	Temperature	K
V	Volume	$\mathrm{m^3}$
v_j	Volumetric fraction of component *j*	-
W_s	Work	J
x_j^i	Mass fraction of component j in flow i.	-
z	Valve position	-
GOR_i	Gas-oil ratio in well *i*	-

Chapter 1

Introduction

Nonsmooth equations are equations that are not differentiable at every point. For systems of such equations it is not possible to calculate the Jacobian, which gradient based solvers rely on. However, nonsmooth equation systems can be solved by semismooth Newton methods, which are similar to the standard Newton method, but where the derivative is replaced by a element from Clarke's (generalized) Jacobian[3]. The Clarke Jacobian is a set-valued generalized derivative for Lipschitz continuous functions. Such functions are not necessarily differentiable at every point. The problem with the Clarke Jacobian is that they do not obey strict calculus rules by equality, meaning that it is not possible to compute them automatically.

During the last few years there has been a development within the theory of nonsmooth analysis, making it possible to automatically compute generalized derivatives that can be a replacement of Jacobian elements for lexicographic smooth functions. First, the lexicographic derivative was proposed by Nesterov in 2005[7], before Khan and Barton proved that these derivatives was part of the plenary hull of the Clarke generalized Jacobian[4]. This means that such derivatives are equally useful as the Clarke Jacobian in a semismooth Newton method. The main advantage of the lexicographic derivative is that it is possible to calculate it automatically through the lexicographic directional derivative.

In many physcial systems, like chemical systems, there are nonsmooth behavior. Example of such behavior is discrete events like phase shifts and, important to this project, the change in flow direction in a pipe. Instead of using hybrid models, where logical statements are used to incorporate such discrete events of the model, a nonsmooth model formulation can be used. In this project, a nonsmooth model of a connected oil well system is developed and solved by using the newly developed theory within nonsmooth analysis.

1.1 Motivation

As mentioned, there are nonsmooth behaviors in chemical systems, and one of them is when a flow changes direction. In this project the main objective is to show that the new developments within nonsmooth analysis can be used to formulate a model which allows bi-directional flow. Today, such systems are often solved by using logical statements (i.e a hybrid model) or by using a smooth approximation of the nonsmooth functions. By implementing automatic differentiation and using nonsmooth solvers, such problems are avoided. In this project a connected oil well system is modeled by the use of nonsmooth equations. The system is used as an example as it naturally has nonsmooth behavior. However, the nonsmooth equation formulations are not restricted to this system and can be applied to other systems with similar behavior. Therefore, this project should demonstrate the advantages of using this approach, as well as motivate others to consider using nonsmooth formulations when suitable.

1.2 Approach

The model of the connected oil well system was implemented in MATLAB. In addition, a class called valder was implemented in MATLAB using object-oriented programming (OOP). It is this class that performs the automatic differentiation, of both smooth and nonsmooth functions. The system of equations was solved using the built in Levenberg-Marquardt algorithm (LMA) in Matlab, and by using a Newton-type method.

Chapter 2

Theory

In this chapter mathematical theory regarding the calculation of the generalized derivatives needed to solve the connected oil well system, described in chapter 3, is presented. This chapter is important as it explains how the derivatives of the nonsmooth functions are computed, which is essential to this project. The connected oil well system is modeled by using nonsmooth functions (in the form of *piecewise differentiable* functions), meaning that some equations in the model is not differentiable at certain points. For such function, *generalized derivative* information is required as an replacement for the ordinary derivative of continuously differentiable functions. This information can be obtained from elements of the *Clarke Jacobian* or the *B-subdifferential*[3]. The problem with these subdifferentials is that they do not obey strict calculus rules, meaning that it is not possible to calculate them automatically[3]. 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[4]. These lexicographic derivatives is possible to compute through the *lexicographic-directional derivatives*, which obeys strict calculus rules. This development makes it possible to obtain a element of the Clarke Jacobian by *automatic differentiation*.

The chapter will define piecewise differentiable functions, *convex* functions, sets and hulls before the B-subdifferential and Clarke Jacobian is introduced. After this introduction, the lexicographic and lexicographic directional derivative will be defined. Further, the *automatic differentiation* with the extension of the computation of lexicographic derivative of the absolute function will be covered. Finally, solvers for nonsmooth problems is discussed.

2.1 Piecewise differentiable ($\mathcal{PC}^1$) functions and convexity

In the connected oil well system the equations for flow rates through valves and compositions in manifold includes the $\mathcal{PC}^1$ functions abs, min and max. $\mathcal{PC}^1$-functions are defined in definition 2.1.

Definition 2.1. *(From [3]) Consider an open set $X \subset \mathbb{R}^n$ and a function $\mathbf{f} : X \to \mathbb{R}^m$. As defined by Scholtes[11], $\mathbf{f}$ 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 $\mathbf{f}_{(1)},...,\mathbf{f}_{(q)} : \mathbf{N} \to \mathbb{R}^m$ such that $\mathbf{f}$ is continuous on N and such that*

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

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*

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

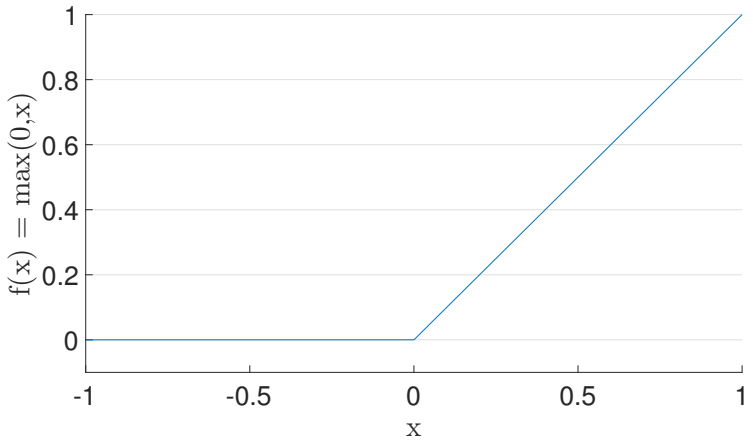


Figure 2.1: Graph of max(0,x)

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

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

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

To define the B-subdifferentialand 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 [8]) A set $S \in \mathbb{R}^n$ is a convex set, if for any two point $\mathbf{x} \in S$ and $y \in S$, 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 a example of both a convex and non-convex set is illustrated.

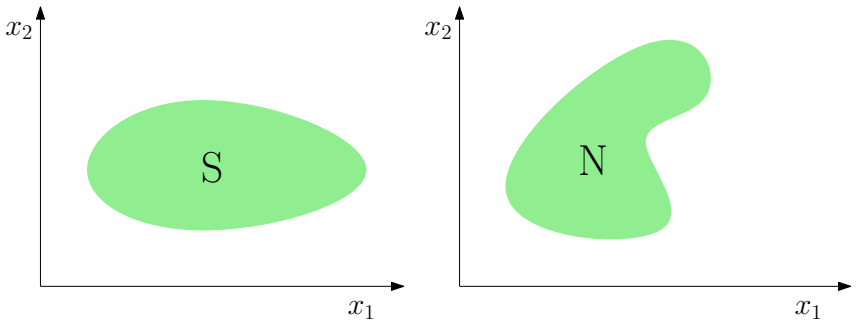


Figure 2.2: Illustration of a convex set, S, and a non-convex set N.

Definition 2.3. *(From [8]) 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 $\mathbf{y} \in$, 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 [8]) A convex combination of a finite set of vectors* $\{\mathbf{x}_1,\mathbf{x}_1,...,\mathbf{x}_m\}$ *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 $\{\mathbf{x}_1,\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ *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

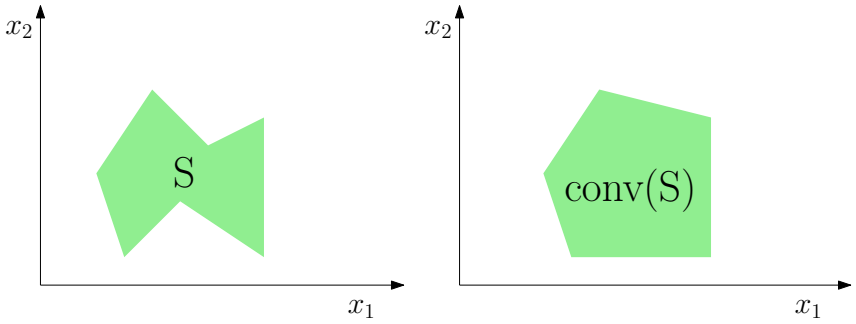


Figure 2.3: Illustration of a non-convex set (S) and its corresponding convex hull (conv(S)).

2.2 B-subdifferential & Clarke generalized Jacobian

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

Definition 2.5. *(From [8]) 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 are **not** Lipshitz continuous is f(x) = sqrt(x). This is because as $x \to 0$, $f'(x) \to \infty$.

Definition 2.6. *(From [3]) 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*

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

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

The B-subdifferential will in other words be a set which consist of Jacobian that arises when $\mathbf{x}$ is approached from every possible directions. The Clarke Jacobian is, from definition 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 jacbobian when $\mathbf{f}$ is continuously differeniable at $\mathbf{x}$.

Example 2.2. *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.*

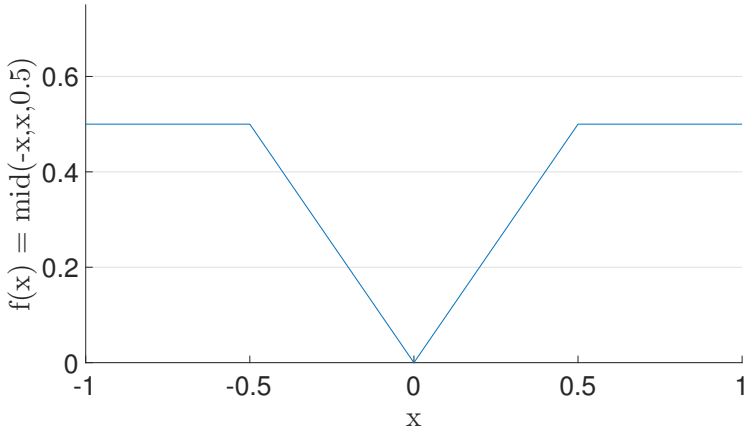


Figure 2.4: The figure shows the graph of f(x) = mid(-x,x,0.5).

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 point is a single valued set as the slope of f(x) does not change depending on from which direction the point is approached. For the nondifferentiable points on the other hand, the Jacobians 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. A disadvantage with the Clarke Jacobian is that it only

satisfies some classical calculus rules as inclusions, and not equality. One of these, the chain rule, has to be satisfied as equality to be used in automatic vector forward differentiation[3], which is used in this project. In calculus, the chain rule is the formula for calculating the derivative of the composition of two or more functions,

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

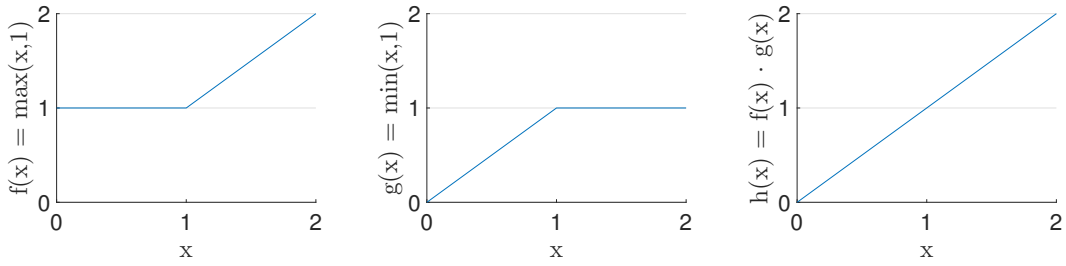
with $(\mathbf{f} \circ \mathbf{g})$ being $\mathbf{f}(\mathbf{g}(\mathbf{x}))$.

Example 2.3. *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:*

$$\partial f(1) = [0, 1]$$
$$\partial g(1) = [0, 1]$$

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 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[4], 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 [3]) The plenary Jacobian $\partial_P \mathbf{f}(\mathbf{x})$ is the plenary hull of the Clarke Jacobian, and satisfies:*

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

This means that the lexicographic derivatives can be used in non-smooth 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 [10].

2.3 Lexicographic derivatives

The lexicographic derivatives was first proposed and developed by Nesterov[7], and, as mentioned earlier, proved to be a part of the plenary hull of the Clarke generalized Jacobian by Khan and Barton[4]. For a function to have a well defined lexicographic derivative at a point $\mathbf{x}$, it must 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 2.8. *(From [8]) The directional derivative of a function* $\mathbf{f}(\mathbf{x})$*:* $\mathbb{R}^n \to \mathbb{R}^m$ *in the direction p is given by,*

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

Definition 2.9. *(From [3]) 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* $p \in \mathbb{N}$ *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[3].

Definition 2.10. *(From [3]) 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*

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

Definition 2.11. *(From [3]) 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. *Let* $\mathbf{f}(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}$ *be the* $\mathcal{PC}^1$ *function: f(*x_1*,*x_2*) = min*$\{x_1,x_2\}$*. The function is not differentiable at every 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 2.9 at (*x_1*,*x_2*) = (0,0).*

Given the direction matrix

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

which is square and nonsingular. The first order directional derivative is computed as defined in definition 2.9:

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

The second order directional derivative is further calculated by using definition 2.9:

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

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

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

Using definition 2.11, the LD-derivative can be calculated in the two presented alternatives. Equation 2.12 gives

$$f'(\mathbf{0};\mathbf{M}) = \left[f^0_{\mathbf{0},\mathbf{M}}\left(\mathbf{m}_{(1)}\right) \quad f^1_{\mathbf{0},\mathbf{M}}\left(\mathbf{m}_{(2)}\right)\right] = [min\{m_{11}, m_{21}\} \quad m_{22}] = [0 \quad 1]\,.$$

Alternative 2, equation 2.13 gives the same generalized derivative:

$$f'(\mathbf{0};\mathbf{M}) = \left[f^2_{\mathbf{0},\mathbf{M}}\left(\mathbf{m}_{(1)}\right) \quad f^2_{\mathbf{0},\mathbf{M}}\left(\mathbf{m}_{(2)}\right)\right] = [m_{21} \quad m_{22}] = [0 \quad 1]\,.$$

The graph of f(x,y) = min{x,y} is presented in figure 2.6.

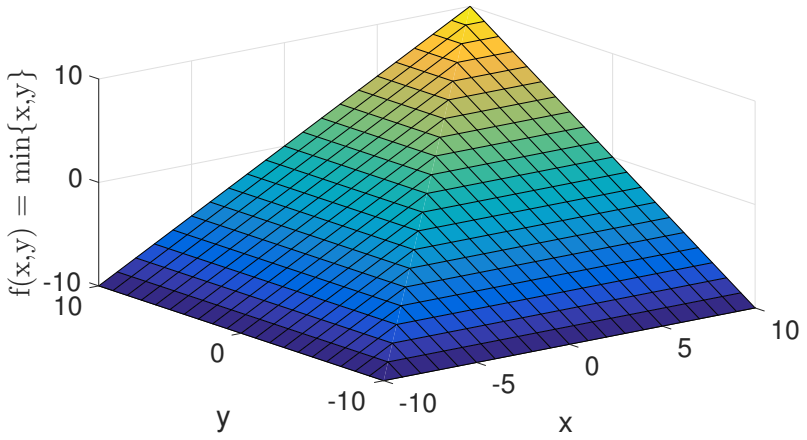


Figure 2.6: Graph of f(x,y) = min{x,y}

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[3].

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 report, 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[6]. 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 project this has been implemented using object-oriented programming (OOP) in MATLAB. The objects has both a value and a derivative. By overwriting the elemental functions, both the value and derivative is updated after each elemental operation. The class *valder* is presented in appendix C.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 2.12. *(From [3]) A factorable function* $\mathbf{f}$ *us* **L***-factorable if the elemental library* $\mathcal{L}$ *contains only lexicographically smooth functions whose LD-derivatives are known or computable.*

In definition 2.12 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** and **max** are used in the model formulation presented in chapter 3, but these are expressed as a function of the absolute function in the *valder* class. This means that it is actually the **abs**-function that is evaluated when the **min** and **max** functions are called. **min**$\{x, y\}$ and **max**$\{x, y\}$ can be expressed as a function of **abs**,

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

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

Algorithm 1: Computes the LD-derivative, $\mathbf{u}'(\mathbf{x}; \mathbf{M})$ for the absolute value function u = $|x|$ [3]

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

if x $\neq$ 0 **then**
  Set $\dot{\mathbf{V}} \leftarrow (\text{sign}x)\mathbf{M}$
else
  Set $s_1 \leftarrow 1$
  for k = 1 **to** p **do**
    if $m_{(k)} \neq$ **then**
      Set $s_1 \leftarrow \text{sign}m_k$
      Break out of for-loop
    end if
  end for
  $\dot{\mathbf{V}} \leftarrow s_1\mathbf{M}$
end if
return $\dot{\mathbf{V}}$

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 **abs**(x). With x = 0, the LD-derivative is determined by the sign of the first non-zero direction.

2.5 Solvers for nonsmooth equation systems

One type of solvers that can be used to solve nonsmooth equation systems is the nonsmooth Newton-like methods, (see equation 2.8). In this project the Newton-like method, using the Lexicographic derivatives as a replacement for the Jacobian, has proven to be quite sensitive to the initial guess (further discussed in section 4.4). As a result, the Levenberg-Marquardt algorithm (LMA) in MATLAB has been used to solve the system. The LMA is used to solve non-linear least squares problem and can be thought of as a combination of steepest descent and the Gauss-Newton method[5]. Far away from the solution the LMA will behave as the steepest descent, which is slow, but guarantees convergence[5]. Closer to the correct solution, it becomes Gauss-Newton method. The gradient descent is described in equation 2.17[8], and the search direction in the Gauss-Newton method is the solution of equation 2.18[8].

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{Jf}(\mathbf{x}^k) \tag{2.17}$$

$$\mathbf{Jf}(\mathbf{x}^k)^\mathbf{T}\mathbf{Jf}(\mathbf{x}^k)\mathbf{p}_{GN}^k = -\mathbf{Jf}(\mathbf{x}^k)^\mathbf{T} f(\mathbf{x}^k) \tag{2.18}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \left(\mathbf{Jf}(\mathbf{x}^k)^\mathbf{T}\mathbf{Jf}(\mathbf{x}^k)\right)^{-1}\mathbf{Jf}(\mathbf{x}^k)^\mathbf{T}\mathbf{f}(\mathbf{x}^k) \tag{2.19}$$

In this project, the generalized derivative $\mathbf{G}$ has been used as a substitution for Jacobian in equation 2.17 and 2.18. 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.

Chapter 3

Connected oil wells modeling

In this chapter the steady state model of connected oil wells will be described in detail. The model is describing the system presented in figure 3.1.

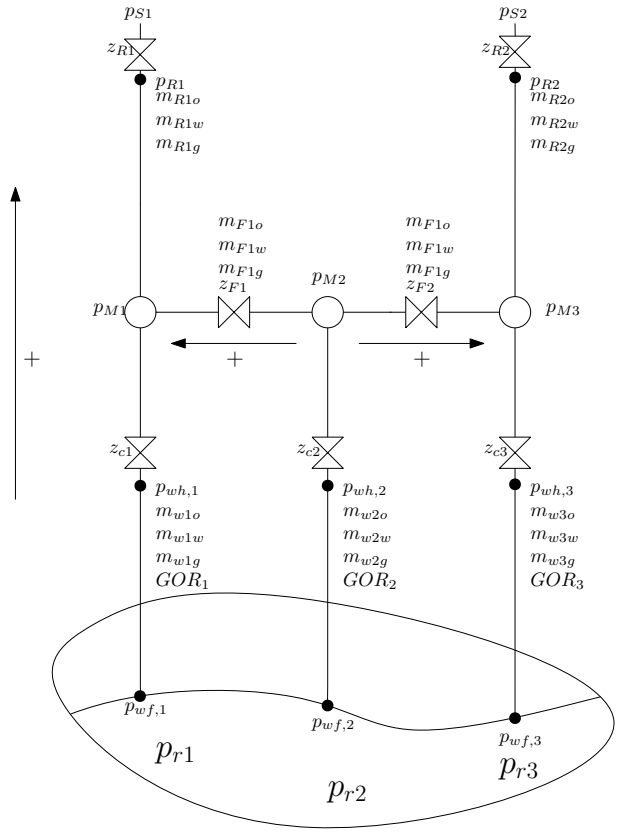


Figure 3.1: Illustration of the connected oil well system modeled in this project with the relevant nomenclature and positive flow directions indicated.

The system consists of three wells that are connected to two risers. In the model the reservoir and separator pressures are set, and it is this pressure gradients that sets the mass flows in the different pipe segments. The model is formulated using the newly developed methods within nonsmooth analysis described in chapter 2. The advancements within nonsmooth analysis has made it possible to solve systems with nonsmooth equations and is computationally tractable thanks to the vector forward mode of automatic differentiation for generalized derivative evaluation[3][9]. Formulating the problem in this manner allows the streams to flow in both directions. The main objective of modeling this system is to show that

this is possible to achieve using nonsmooth equations were the Jacobian elements at nondifferentiable points are substituted by elements from the lexicographic derivative. The nonsmoothness of the system arises in the valve equations (section 3.4) in addition to the calculation of the composition in the manifolds (section 3.5). This chapter will include the assumptions made and all the equations used in the model with a comprehensive description of how the nonsmooth equations are formulated.

3.1 Reservoir inflow model

The inflow of oil and water into the well is assumed to follow the quadratic deliverability function proposed by Fetkovich[1]:

$$\hat{m}_{w,i,o} = k_{o,i}(p_{r,i}^2 - p_{wf,i}^2), \tag{3.1}$$

$$\hat{m}_{w,i,w} = k_{w,i}(p_{r,i}^2 - p_{wf,i}^2). \tag{3.2}$$

Here $\hat{m}_{w,i,o}$ and $\hat{m}_{w,i,w}$ is the mass flow of oil and water into well i, $k_{o,i}$ and $k_{w,i}$ is the transport coefficient for oil and water from the reservoir to well i. $p_{r,i}$ is the reservoir pressure and $p_{wf,i}$ is the well inflow pressure for well i. The inflow of gas is given by the gas-oil ratio in the well (GOR$_i$):

$$\hat{m}_{w,i,g} = \text{GOR}_i \cdot \hat{m}_{w,i,o}. \tag{3.3}$$

where $\hat{m}_{w,i,g}$ is the mass flow of gas into well *i*. The GOR is assumed to follow the relation proposed by Grimholt[2]:

$$\text{GOR}_i = \frac{k_{g,i}}{k_{o,i}}(p_{r,i} - p_{wf,i})^2. \tag{3.4}$$

3.2 One-phase pseudo fluid

The well flow is a multiphase fluid consisting of liquid and gas. In this model this multiphase fluid is approximated as a one-phase fluid. By doing so, and neglecting mixing effects, the density of the pseudo fluid can be calculated using the volumetric average of the separate densities:

$$\rho_{mix} = v_g\rho_g^{ig} + v_o\rho_o + v_w\rho_w. \tag{3.5}$$

ρ_{mix} is the density of the pseudo fluid, v_g,v_o and v_w is the volumetric fractions of gas, oil and water respectively. ρ_o, ρ_w and ρ_g^{ig} is the density of oil, water and ideal gas. The flows in the model has units of mass per time, so equation 3.5 is reformulated to mass basis by the use of the following relation:

$$v_i = \frac{\hat{m}_i/\rho_i}{m_o/\rho_o + m_w/\rho_w + m_g/\rho_g^{ig}} \qquad \text{for } i = \{o, w, g\}. \tag{3.6}$$

By combining equation 3.5 and 3.6, the overall density, in terms of mass flows, becomes

$$\rho_{mix} = \frac{\hat{m_o} + \hat{m_g} + \hat{m_w}}{\hat{m_g}/\rho_g^{ig} + \hat{m_o}/\rho_o + \hat{m_w}/\rho_w}. \tag{3.7}$$

The oil and water is assumed to be incompressible, and that the gas behaves as an ideal gas. These assumption implies that the density of oil and water is constant, whilst the density of gas can be calculated from the ideal gas law,

$$\rho_g^{ig} = \frac{pM_g}{RT}, \tag{3.8}$$

where M_g is the molar mass of the gas, R is the gas constant and T is the temperature.

3.3 Pressure drop through a vertical pipe

The calculation of the pressure drop through the well (vertical pipe) is calculated from the stationary mechanical energy balance [12];

$$m\alpha\frac{v_2^2}{2} + mgz_2 + \int_{p_1}^{p_2}\frac{dp}{\rho} + \Phi = m\alpha\frac{v_1^2}{2} + mgz_1 + W_s. \tag{3.9}$$

In the mechanical energy balance the kinetic energy ($mv^2/2$), potential energy (mgz), "potential pressure energy" ($m\int\frac{\mathrm{d}p}{\rho}$), energy loss due to friction (Φ) and work (W_s) are included. By assuming no slip between the phases and neglecting kinetic energy, friction and work the equation is reduced to:

$$\int_{p_1}^{p_2}\mathrm{d}p = \int_{h_1}^{h_2}\rho_{mix}(p)g\mathrm{d}h. \tag{3.10}$$

Using the expression of ρ_{mix} obtained in 3.7 and integrating equation 3.10 between the limits, indicated in figure 3.2, gives

$$\frac{\hat{m}_g RT}{M_g}\ln(p_2/p_1) + \left(\hat{m}_o/\rho_o + \hat{m}_w/\rho_w\right)(p_2 - p_1) = \hat{m}_{tot}g\Delta h. \tag{3.11}$$

Equation 3.11 cannot be solved directly for p_2, but by performing a serial expansion of the natural logarithmic part,

$$\ln(p_2/p_1) = \ln(p_1 + \Delta p/p_1) \approx \Delta p/p_1, \tag{3.12}$$

the pressure difference through a vertical pipe can be expressed as:

$$\Delta p = p_2 - p_1 = \frac{\hat{m}_{tot}gp_1\Delta h}{\frac{\hat{m}_g RT}{M_g} + p_1\left(\hat{m}_o/\rho_o + \hat{m}_w/\rho_w\right)}. \tag{3.13}$$

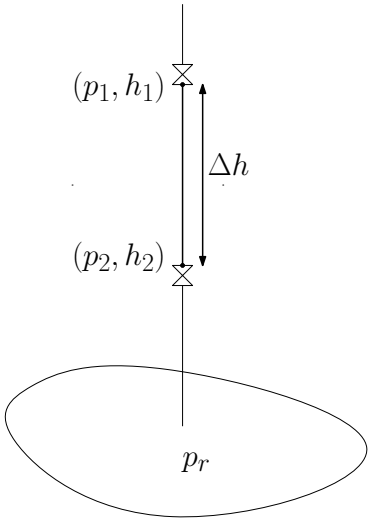


Figure 3.2: Sketch of a single oil well where the limits for calculation of pressure through the vertical pipe is indicated.

3.4 Pressure drop through a valve

The mass flow through a valve is modeled by a modification of the standard valve equation,

$$\hat{m} = f(z)C_d A\rho\sqrt{p_2 - p_1}, \tag{3.14}$$

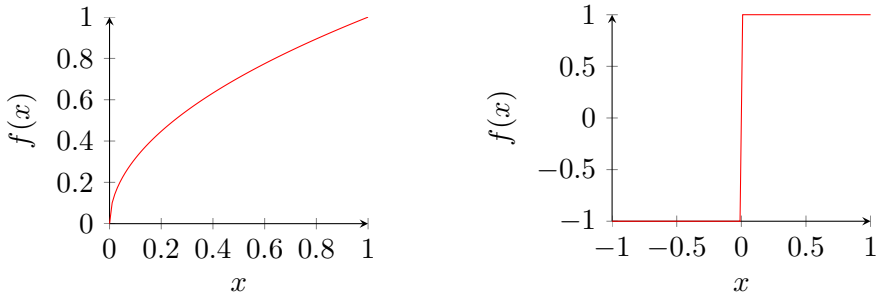
where $f(z)$ is the valve characteristic function, C_d is the valve coefficient, A is the cross-section area and p_1 and p_2 is the pressure on each side of the valve. The valves are assumed to have linear characteristics, meaning $f(z) = z$, where z is the valve position ranging from z = 0 (fully closed) to z = 1 (fully open). The standard valve equation is modified to allow mass flows in both directions. One approach to achieve this is to introduce the $sign$-function,

$$sign(x) = \begin{cases} 1 & :\text{x}> 0 \\ -1 & :\text{x}< 0 \end{cases} \tag{3.15}$$

resulting in the following equation:

$$\hat{m} = zC_d A\rho\, sign(p_2 - p_1)\sqrt{|p_2 - p_1|}. \tag{3.16}$$

This approach is not possible to use as the computation of the lexicographical directional derivatives demands that the function is Lipschitz continuous (see definition 2.11). Neither the $sqrt$-function nor the $sign$ function meets this demand. The $sqrt$-function is not Lipschitz continuous at x = 0, as the derivative is not bounded from above (it approaches infinity). The $sign$-function has a step change at x = 0, and is therefore not Lipschitz continuous.



(a) $sqrt(x)$ **(b)** $sign(x)$

Figure 3.3: Lipschitz discontinuous functions sqrt(x) and sign(x)

Another approach is to square equation 3.14, resulting in,

$$\hat{m}^2 = \begin{cases} z^2 C_d^2 A^2 \rho(p_2 - p_1) & : p_2 - p_1 > 0 \\ -z^2 C_d^2 A^2 \rho(p_2 - p_1) & : p_2 - p_1 < 0 \end{cases} \tag{3.17}$$

Equation 3.17 is not defined at x = 0. Reformulating the equation by substituting $\hat{m}^2$ with $\hat{m}|\hat{m}|$ results in equation 3.18, which is Lipschitz continous and allows the mass to flow in both directions.

$$\hat{m}|\hat{m}| = z^2 C_d^2 A^2 \rho(p_2 - p_1) \tag{3.18}$$

As for the flow in the well, the fluid is assumed to be a pseudo one-phase fluid inside the valve. The density is calculated by the average of the densities on both sides of the valve,

$$\rho = \frac{1}{2}(\rho_{mix,1} + \rho_{mix,2}) \tag{3.19}$$

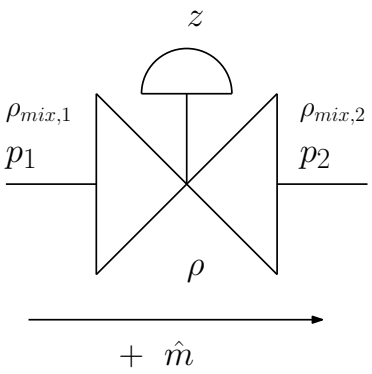


Figure 3.4: Sketch of valve with the densities, pressures and positive flow direction indicated.

3.5 Manifolds

The wells are connected through a common point of mixing. As the flow is modeled by equation 3.18, the flow is either in or out of each well, depending on the pressure difference. Since there is no accumulation in the manifolds, the sum of all flows in and out of the manifold must be zero, leading to the following relationship,

$$\sum_{i=1}^{n} F^i = 0, \tag{3.20}$$

where F^i is the flow rate in or out of well i, depending on the sign. The calculation of the composition in a mixing point/manifold is dependent on which flows that are going into it.

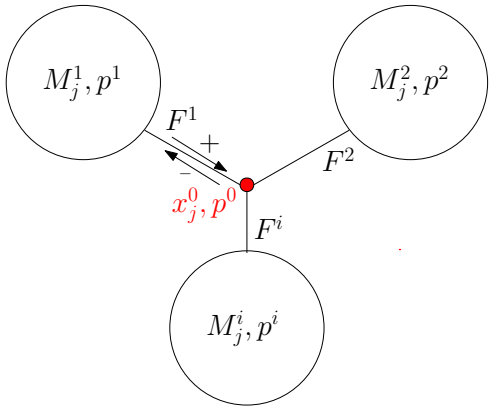


Figure 3.5: Scheme of a the manifold. The flow rate is defined as positive into the manifold. The sign of the flow rate is dependent on the pressure difference.

The standard approach for calculation of the composition in such mixing points is to apply logical statements on the pressure difference. The possibility to include nonsmooth equation in the model removes the need of these statements. A nonsmooth formulation of the components balance, which covers all cases of different flow directions, is proposed by Stechlinski 2017 [9] as,

$$\sum_{i}^{n}(\max\{F^i, 0\}x_j^i) = \left(\sum_{i}^{n}\max\{F^i, 0\}\right)x_j^0, \quad j = 1, \ldots, nc, \tag{3.21}$$

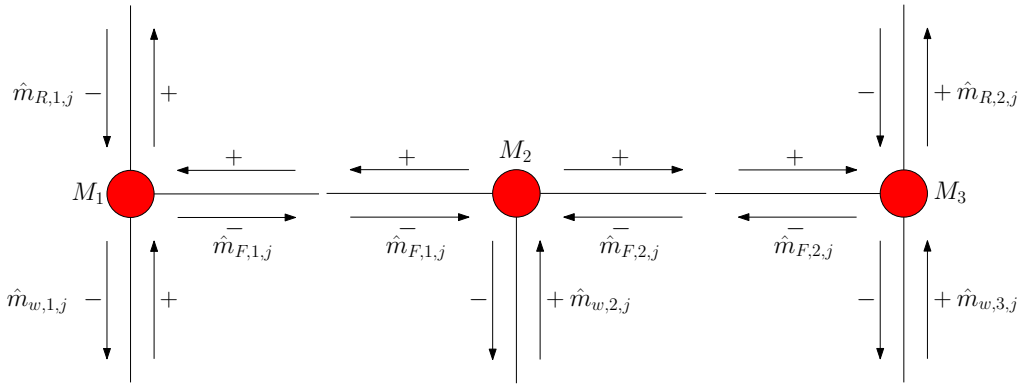
where j is the component and x_j^0 is the composition in the manifold. By using this formulation, only the positive flows are taken into account when calculating the compositions. In equation 3.21 flows from a well into the manifold is defined as positive. In the model the flows are defined as positive upwards and

from Manifold 2, M_2 to Manifold 1, M_1 and Manifold 3, M_3, as illustrated in figure 3.1 and 3.6. For this reason, the defined direction has to be accounted for, leading to the following equations for the three manifolds:

$$\begin{aligned}&(\max(\hat{m}_{w,1},0)+\max(\hat{m}_{F,1},0)+\max(-\hat{m}_{R,1},0))\,x_j^{M_1}\\&=\max(\hat{m}_{w,1,j},0)+\max(\hat{m}_{F,1,j},0)+\max(-\hat{m}_{R,1,j},0)\quad j=o,w,g,\end{aligned}\tag{3.22a}$$

$$\begin{aligned}&(\max(\hat{m}_{w,2},0)+\max(-\hat{m}_{F,1},0)+\max(-\hat{m}_{F,2},0))\,x_j^{M_2}\\&=\max(\hat{m}_{w,2,j},0)+\max(-\hat{m}_{F,1,j},0)+\max(-\hat{m}_{F,2,j},0)\quad j=o,w,g,\end{aligned}\tag{3.22b}$$

$$\begin{aligned}&(\max(\hat{m}_{w,3},0)+\max(\hat{m}_{F,2},0)+\max(-\hat{m}_{R,2},0))\,x_j^{M_3}\\&=\max(\hat{m}_{w,3,j},0)+\max(\hat{m}_{F,2,j},0)+\max(-\hat{m}_{R,2,j},0)\quad j=o,w,g\end{aligned}\tag{3.22c}$$



(a) Manifold 1 **(b)** Manifold 2 **(c)** Manifold 3

Figure 3.6: The figures shows how the positive and negative directions in and out of the manifolds are defined in the model.

Equations 3.22a, 3.22b and 3.22c is the component balances for manifold 1, manifold 2 and manifold 3 respectively. $\hat{m}_{w,i,j}$ is flow rate in well *i* of component *j*, $\hat{m}_{w,i}$ is the total flow rate in well *i*, $\hat{m}_{F,i,j}$ is the flow from manifold 2 to manifold 1 or 3 of component *j*, whilst $\hat{m}_{F,i}$ is the total flow rate in the same streams. $\hat{m}_{R,i,j}$ is the flow rate of component *j* in riser *i*, $\hat{m}_{R,i}$ is the total flow rate in riser *i* and $x_j^{M_i}$ is the molar fraction of component *j* in manifold *j*.

3.6 Calculation of flow rate in connection pipes

The calculation of the flow rates in the connection pipe from manifold 2 to manifold 1 and 3, F_1 and F_2, requires a nonsmooth formulation to be valid for both the possible flow directions. There are two physical requirements that has to be satisfied. First, all the three components (oil, water and gas) needs to have the same flow direction. Second, the composition of the flows is the same as the composition in the manifold which it flows from. Equation 3.18, together with a equation formulated in a similar manner as for the calculations of the composition in the manifolds used in section 3.5 can fulfill both requirements. The driving force for the flows is the difference in pressure, so by using the sign of the pressure difference it is possible to find the direction of the flow. Using this logic, a possible formulation for the flow rate of component *j* in stream $\hat{m}_{F,i}$ is,

$$\hat{m}_{F,i,j}=\left(\frac{\min(\Delta p_i,0)}{\Delta p_i}x_j^{M_k}+\frac{\min(-\Delta p_i,0)}{-\Delta p_i}x_j^{M_2}\right)\hat{m}_{F,i},\quad i=1,2\quad j=o,w,g\tag{3.23}$$

$$\Delta p_i=p_{M_2}-p_{M_k}\quad i=1,2\tag{3.24}$$

where subscript *k* is 1 and 3 when subscript *i* is 1 and 2 respectively, as indicated in figure 3.7. If the pressure difference is positive, the pressure in manifold 2, p_{M_2}, is higher than in the connected manifold. If this is the case, equation 3.23 makes sure that the composition in the flow is equal to the composition in M_2, as it should. If the pressure difference is negative, the composition will be equal to the composition in the connected manifold.

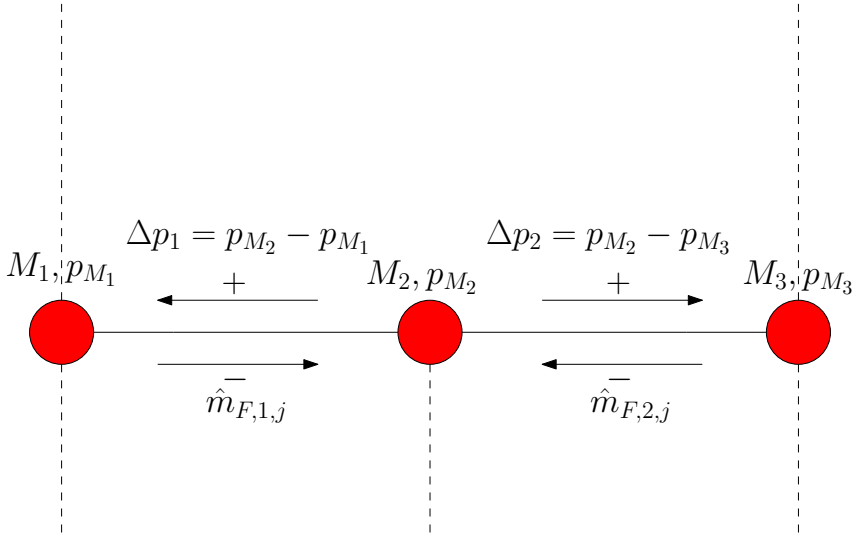


Figure 3.7: A scheme of the connected manifolds with the defined pressure difference included.

Chapter 4

Results and discussion

In this chapter the results obtained from running the model at different values for the degrees of freedom, which are the 7 valve positions and reservoir pressure, are presented. First, some results showing the behaviour of the model with respect to changes in a valve position is presented. Second, the allowance of bi-directional flow in the model is shown. Finally, a simple graphical optimization, with respect to different objectives, of the valve positions are presented. In this chapter all the parameters are set to the values presented in table A.1 if not stated otherwise. Similarly, all valves are set to be half open ($z = 0.5$) as default.

4.1 Varying the valve position in well 1

To illustrate the general behavior of the system, this section will present results obtained by solving the model for different valve positions in well 1. This is done to introduce how the system is connected before presenting the results which shows that the model handles any flow direction. In figure 4.1 the total flow rates in the well system as functions of the valve position, z_{c1}.

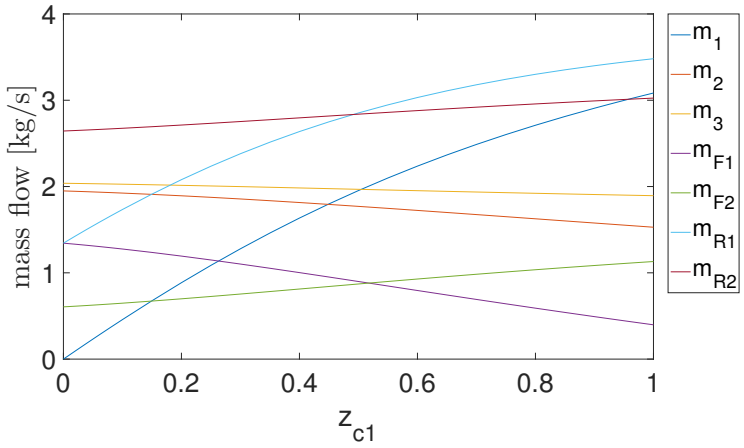


Figure 4.1: Graph of the total flow rates with varying valve position in well 1.

The results presented in figure 4.1 shows that all flows have the same sign for any valve opening in well 1. This means that the flow directions does not change. These flow directions are illustrated in figure 4.2. Overall, with increasing valve opening in well 1 the flow rate in well 1 and riser 1 increases, whilst the flow rate in well 2 and 3 is reduced. Still, the total flow rate in riser 2 is slightly increased as the fraction of the flow in well 2 that flows into manifold 3 is increased.

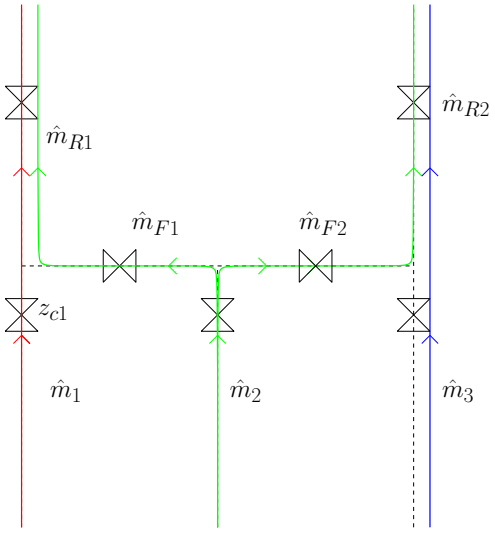
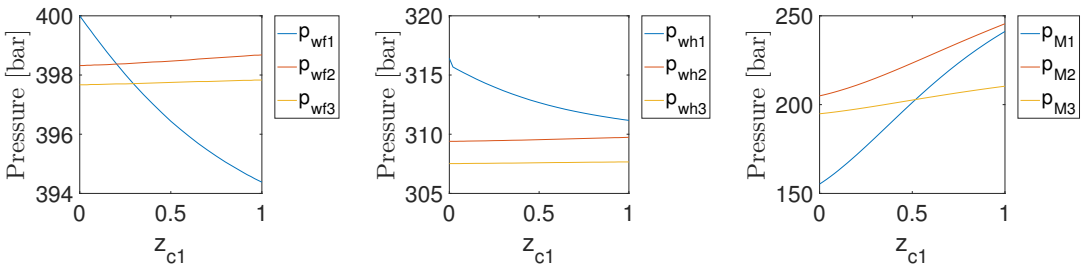


Figure 4.2: The flow directions in the well system for all valve position, z_{c1}, with all other parameters kept constant.

Further, the results in 4.1 shows that for $z_{c1} = 0$ the flow in well 1 is zero, which is what is desired. It also shows that larger parts of the flow in well 2 flows into riser 1 than riser 2 for small valve openings. This is because the pressure in manifold 1 is lower for small openings as the pressure drop through the valve in well 1 is high for small valve openings (see section 3.4). As the valve opening is increased, the flow rate in well 1 increases, and thereby also the flow rate in riser 1. The higher production in well 1 leads to an increase in the pressure in manifold 1 (see figure 4.3), further decreasing the flow rate from manifold 2 to manifold 1 due to a decrease in the pressure gradient.



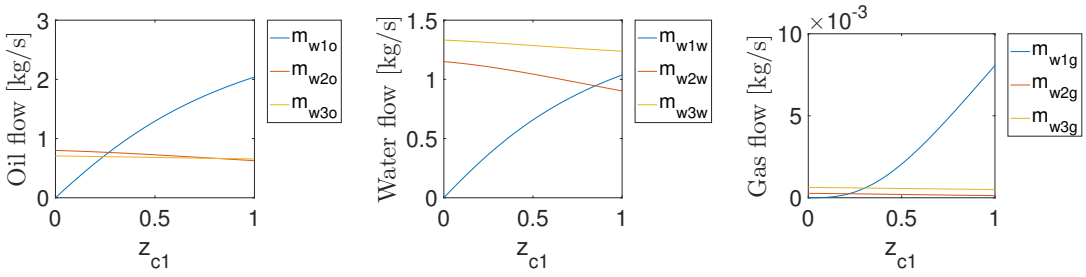
(a) Well inflow **(b)** Well heads **(c)** Manifolds

Figure 4.3: Pressures in the oil well system at different valve openings in well 1, z_{c1}

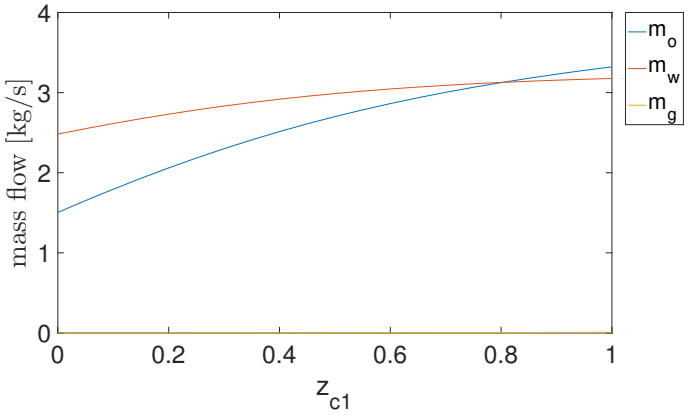
The graphs of the pressures in figure 4.3 shows the pressures in the well system plotted against the valve position in well 1. Figure 4.3a correlates with the flow rates presented in figure 4.1 as the mass flow rates increases with increasing pressure difference between the reservoir and well inflow (see section 3.1). The well head pressure decreases for well 1, while it increases for well 2. This behaviour arises from the energy balance described in section 3.3, which states that the pressure drop through a vertical pipe is dependent on the mass flow rate and the densities of the components. Thus, figure 4.3b corresponds to the flow rates.

The valve opening will also affect the component flow rates in each well due to the different dependencies of the pressure gradients (see section 3.1). In addition the different wells have different transport coefficients, meaning that the composition in each well will be different. The main objective of an oil well is to produce oil, but often there are capacity problems for gas and water. The gas production in the model has a higher order dependency on the pressure difference, meaning that the gas flow increases

relatively faster than both oil and water. Figure 4.4 shows how the different component flows change as functions of the valve opening. Notice how both oil and water flow rate starts to converge, while the gas rate seem to diverge. In figure 4.4d the total component flows are presented. Since both well 2 and 3 has higher transport coefficients for water than for oil, the flow rate of water is larger than that of oil in these wells. As the valve in well 1 is gradually opened, the oil rate increases faster compared to water. This is due to the relative high oil transport coefficient in well 1. It is important to notice that these results are dependent on the transport coefficients, which is set. By changing which well is the best oil producer (changing the transport coefficient to be relative high for oil compared water and gas), the results will be different.



(a) Oil flow **(b)** Water flow **(c)** Gas flow



(d) Total component flow

Figure 4.4: Component mass flow in the oil well system at different valve openings in well 1, z_{c1}

4.2 Response to changes in the valve position in riser 1, z_{R1}

The main purpose of modeling the system using nonsmooth functions is to allow bi-directional flow in the pipe segments. In the previous example, varying the valve position in well 1, did not show that the model is solve-able for all flow directions. In this section, the valve position in riser 1 will be adjusted. Closing the valve entirely will force the flow in well 1 to flow from manifold 1 to manifold 2, which is the opposite flow direction compared to the case with all valves at half open position. In figure 4.5 the flow rates for different valve position in riser 1 is presented.

The results in figure 4.5 shows that the flows are allowed to flow in both directions. Flow $\hat{m}_{F2}$, from manifold 2 to 1, is negative for small valve opening (below $z_{R1} = 0.165$), while it is positive for larger openings. This means that the flow direction changes in this pipe segment at $z_{R1} \approx 0.165$. These results shows that the nonsmooth formulation together with the replacement of the Jacobian elements with Lexicographic derivative elements allow the model to have a solution for all flow directions. The direction of the flows are presented in figure 4.6. In addition, the model behaves in a manner which is

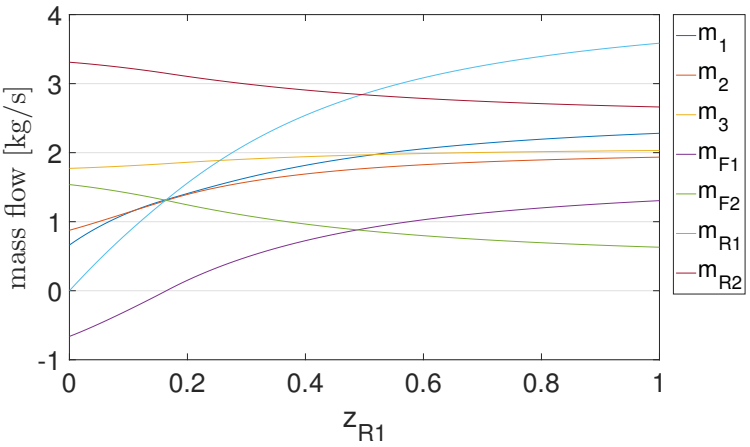
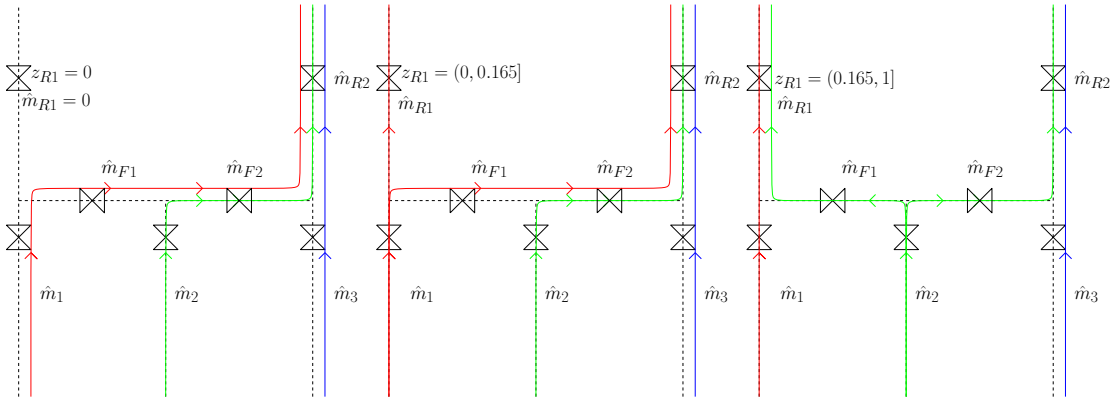


Figure 4.5: Graph of the total flow rates with varying valve position in riser 1.

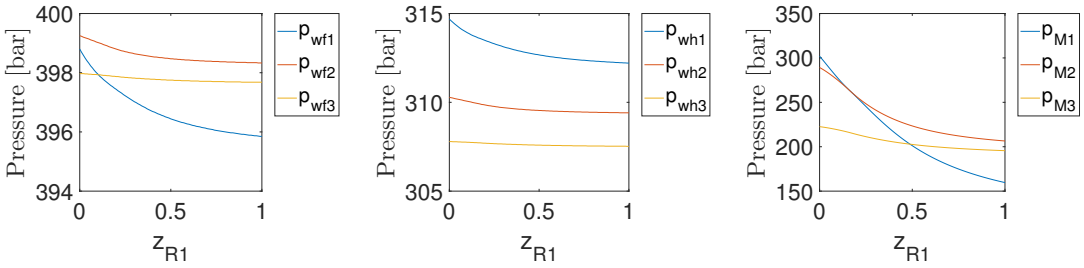
physically expected. When the valve in riser 1 is completely closed, the flow rates from well 1, 2 and 3 is produced through riser 2 as shown in figure 4.6a. By these results it shows that it will be possible to produce from well 1, even with riser 1 completely closed. When the valve in riser 1 is gradually opened, some of the flow rate from well 1 is starting to be produced through riser 1. The reason why only parts of the flow is produced through riser 1 is that the pressure in manifold 1 is higher than in manifold 2 at these valve positions, as shown in figure 4.6b. With the valve in riser 1 at small openings, the pressure drop through the valve is high, limiting the flow rate in riser 1 which is what causes the high pressure in manifold 1.



(a) $z_{R1} = 0$ **(b)** $z_{R1} = (0, 0.165]$ **(c)** $z_{R1} = (0.165, 1]$

Figure 4.6: Flow directions for different valve positions in riser 1, z_{R1}, with all other parameters kept constant.

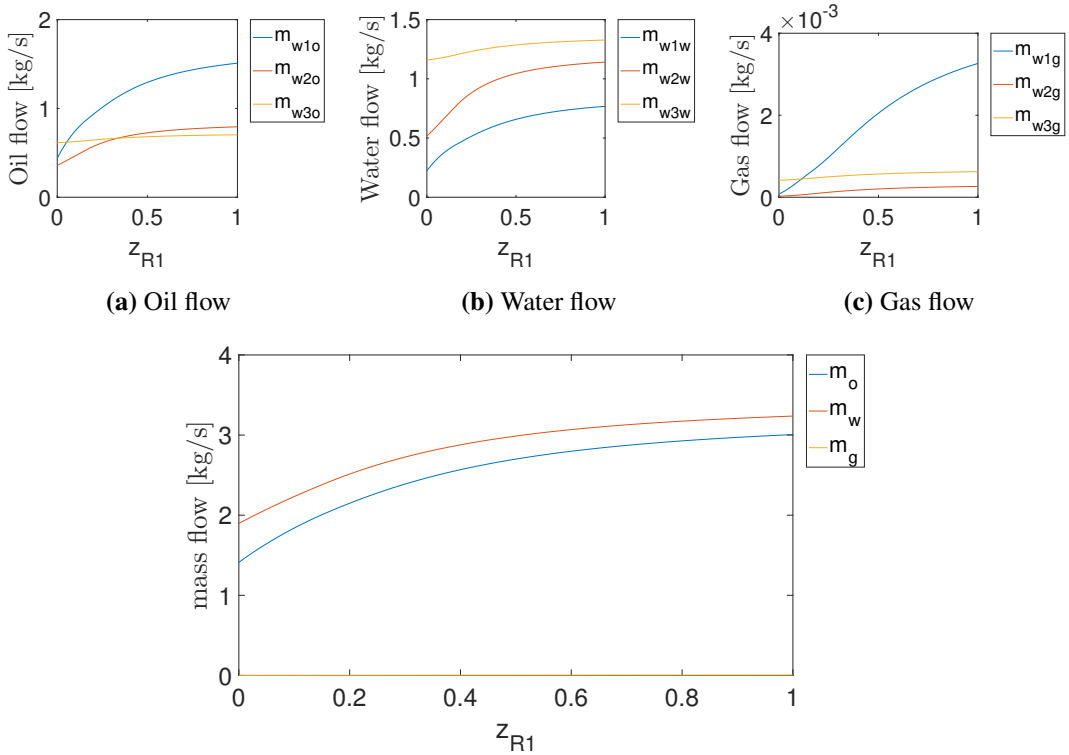
As the valve in riser 1 is further opened, the flow rate increases, and the pressure in manifold 1 decreases. When the valve is at $z_{R1} = 0.165$ the pressure in manifold 1 and 2 intersect each other in figure 4.7c. At this point the pressure gradient changes sign, and the flow changes direction. From this valve position until fully open, the flow direction is as presented in figure 4.6c, which is the same as for the case in section 4.1.



(a) Well inflow **(b)** Well heads **(c)** Manifolds

Figure 4.7: Pressures in the oil well system at different valve openings in riser 1, z_{R1}

Overall, the flow rate in the wells increase with increasing valve opening in riser 1, but as mentioned in the previous section, the component flows are more important from an economic point of view. The component flows as functions of the valve position is presented in figure 4.8. From the graph in figure 4.8a it can be seen that the ratio between the oil and flow rate is close to constant. This means that it is not possible to adjust this ratio using this valve position. But there is another ratio which is important, namely the GOR. The total GOR as well as the GOR in the three wells is shown in figure 4.9. The graphical results shows that the GOR gets approximately three times higher from a closed to fully open valve. This follows from that the pressure at the inflow decreases, see figure 4.7a. This leads to a higher increase in the gas production than for both oil and water. If there are problems with the gas capacity topside, adjusting the valve in riser 1 can reduce the GOR, at the cost of less total production.



(a) Oil flow **(b)** Water flow **(c)** Gas flow

(d) Total component flow

Figure 4.8: Component mass flow in the oil well system at different valve openings in riser 1, z_{R1}

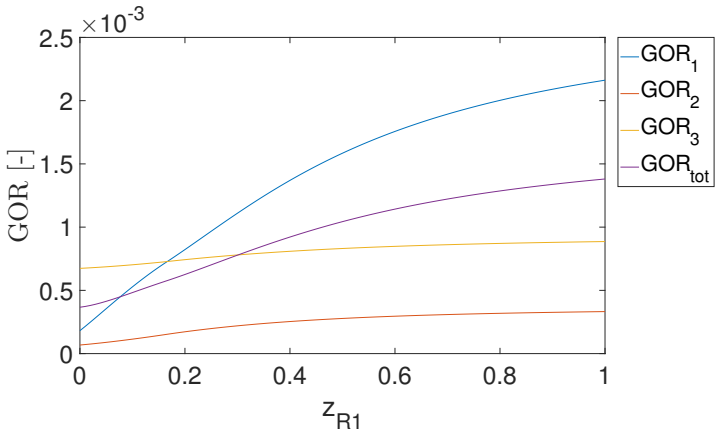


Figure 4.9: Gas-oil ratio (GOR) as a function of the valve position in riser 1.

In addition to the valves, the equation of the mass fractions in both the manifolds and the connecting streams was formulated using the nonsmooth functions min and max (see equation 3.22 and 3.23). In figure 4.10 the mass fraction of oil in manifold 1, x_{M1o}, and in the connecting flow between manifold 1 and 2, x_{F1o}, as plotted against the valve position in riser 1 is shown. The graph in figure 4.10a shows that the mass fraction of oil in manifold 1 is the same as in well 1 for valve openings below 0.165, while it is a weighted average of the mass fraction in the connecting flow 1 and well 1 for larger openings. This can be explained by relating it to the direction of the flows presented in figure 4.5 and 4.6. As discussed earlier, the flow between manifold 1 and manifold 2 changes direction at $z_{R1} = 0.165$. At valve openings less than this, the only flow into manifold 1 is the flow in well 1, thus the mass fraction should be the same as in well 1. This is also what the results in figure 4.10a say it is. With larger valve openings, both flow from well 1 and manifold 2 will flow into manifold 1. This results in that the mass fractions in manifold should be a weighted average of the fractions in these two flows, which the results in figure 4.10a agree with.

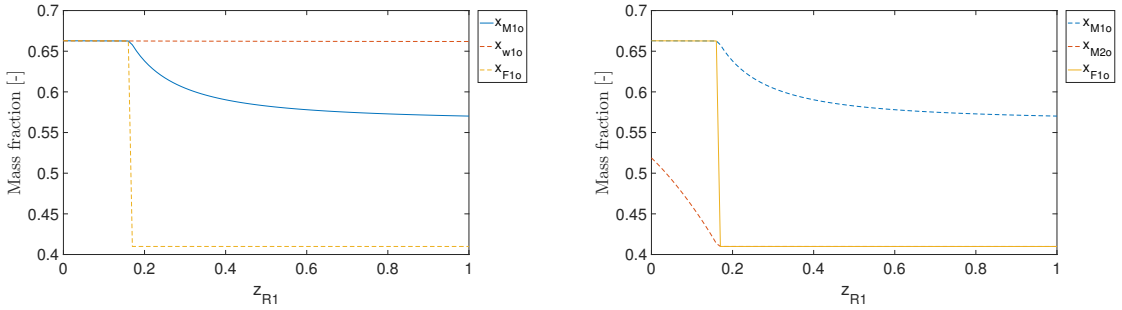
The results in figure 4.10b shows the mass fraction of oil in the connecting flow between manifold 1 and manifold 2 as a function of the valve position in riser 1. The obtained results says that the mass fraction of oil in this flow is equal to the mass fraction in manifold 1 for valve positions less than 0.165, while it is equal to the mass fraction in manifold 2 for larger openings, as it should. When the direction of the flow changes, the mass fractions should also change. This is because the mass fractions of a flow is equal to the fractions in the point it comes from, and not the point it goes into. This behavior is nonsmooth as the equation for the mass fraction "changes" when the flow direction in the connecting flow changes. The equation for the composition in manifold 1 is given in equation 3.22a. With the valve position below 0.165, this equation (for oil) is reduced to

$$\hat{m}_{w,1,o} = \hat{m}_{w,1} \cdot x_o^{M1}.$$

For valve opening above 16.5 %, this equation changes to

$$\hat{m}_{w,1,o} + \hat{m}_{F,1,o} = (\hat{m}_{w,1} + \hat{m}_{F,1}) \cdot x_o^{M1}.$$

The graphs of the mass fraction of oil is a good visualization of that the model formulation captures the nonsmooth behavior, and that it is possible to obtain a solution by using lexicographic derivatives.



(a) Mass fraction of oil in manifold 1, x_{M1o}.

(b) Mass fraction of oil in the connecting flow between manifold 1 and 2,x_{F1o}.

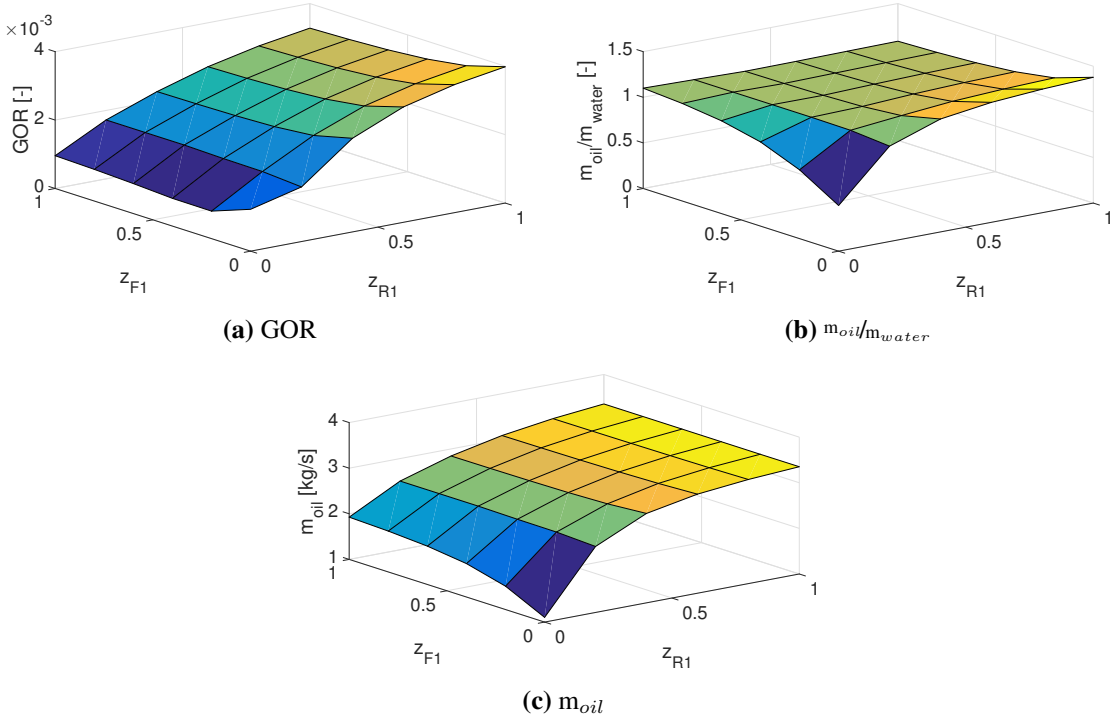
Figure 4.10: The mass fraction of oil in Manifold 1,x_{M1o}, and the connecting flow between manifold 1 and 2, x_{F1o}.

The results presented in this sections shows that the nonsmooth oil well model allows bi-directional flow in the connecting flow between manifold 1 and manifold 2. In addition, it has been shown that the nonsmooth formulation both captures and handles the discrete event where the flow changes directions. This is shown in how the mass fraction in manifold 1 and in the connecting flow changes at this event. Overall, the model behaves in the way it is supposed to, and the main objective has been achieved. The valve equation and the equations used to calculate the composition in the manifolds is not specific for this system. This means that systems with bi-directional flow and/or mixing points can use these formulations and thereby capture all states with a few equations. Further, these equations are simpler to formulate than making a logical statements for all different combination of in- and outflows in a mixing point, which would be the other option.

4.3 Surface response to changes in valve position in riser 1 and connecting flow 1

After establishing that the model and the solvers handle the nonsmoothness in the system, it is possible to optimize the valve positions in the system. Doing a full optimization is out of the scope of this project, but in this section some surface plots that can be used to perform a simple graphical optimization of two variables are presented. These two variables are the valve positions in the riser, z_{R1}, and the valve in the pipe segment between manifold 1 and 2, z_{F1}. For this section, some of the parameters for the system was changed. This was to have a larger difference in the well characteristics. The parameters that was changed is presented in table A.2. All valve positions are still set to 0.5, except the valve in the second connection flow (from manifold 2 to manifold 3) which is set to be fully open (z_{F2} = 1).

As mentioned earlier, there are three important properties that has to be considered to find the optimal valve positions in the system. These three are the amount of oil produced, the gas-to-oil ratio (GOR) and the oil-to-water ratio. The amount of oil produced is important as that is the most valueable product. The GOR should be low due to the gas capacity limitations topside, while oil-to-water ratio is desired to be low as there a cost associated with the water spills, due to oil impurities in the water.



(a) GOR

(b) m_{oil}/m_{water}

(c) m_{oil}

Figure 4.11: GOR, oil-to-water ratio and the total oil production in the oil well system at different valve openings in connection flow 1 and riser 1, z_{F1} and z_{R1}

In figure 4.11a the total GOR in the system is presented as a function of both the valves position, z_{R1} and z_{F1}. The results show that the GOR has a higher dependency of the valve position in the riser than in the connecting flow. The GOR increases as the valve in riser 1 is opened. This is explained by that the valve in the riser has an bigger impact on the production rate than the valve in the connecting flow. As the gas production has a higher order of dependency of the pressure difference between the reservoir and inflow (see section 3.1) than oil and gas, the amount of gas produced will increase relatively faster. This effect seems to affect the GOR to a greater extent than the difference in transport coefficients for gas in the different wells. The optimal conditions for the oil wells are not necessarily to minimize the GOR as it will limit the amount of oil produced. The limit of the GOR depends on the gas capacity topside, and the GOR of other wells in the same network.

The oil-to-water ratio for the system is presented in figure 4.11b. In the model, oil and gas has the same dependency of the pressure difference between the reservoir and inflow (see section 3.1). Because of this, this ratio depends on the transport coefficients for the different wells. With both the valves closed, $z_{R1} = 0$ and $z_{F1} = 0$, well 1 is not produced at all. Therefore, as the results show, the oil-to-water ratio is low at these valve positions. This is because well 1 is the best oil producer. This ratio should be maximized as long as the gas capacity constraint is not violated. From the results the valve position in riser 1 should be fully open, while the valve in the connecting flow should be fully closed. This is to fully utilize the production capacity in riser 1 to only produce well 1, and thereby maximize the oil-to-water ratio.

The last property that is considered is the total oil production. The resulting production rate of oil at the different valve positions is presented in figure 4.11c. With both valves closed the oil production is at the lowest, mainly because the total production rate is lower, but also since the best oil producer, well 1, is not produced at this valve configuration. With the valve in riser 1 completely closed, the valve position in the connecting pipe has a positive effect on the oil rate. This is because some of well 1, which is the best oil producer, is produced through this pipe. Opening the valve in the riser gradually, the flow direction in this pipe segment will change, see figure 4.12. When the direction of the flow changes the

flow, which now will have the composition as in well 2, will reduce the mass fraction of oil in manifold 1, and thereby reducing the amount of oil produced through riser 1. The optimal valve positions with respect to maximizing the total oil production is to have riser 1 fully open, whilst the connecting flow should be fully closed.

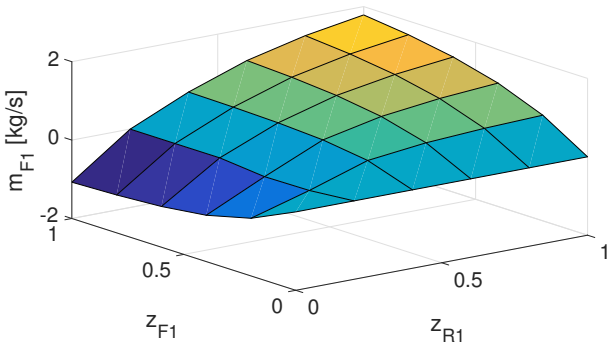


Figure 4.12: Flow rate in the connecting flow between manifold 1 and 2.

To minimize the GOR, the connecting flow should be fully open and the riser should be closed. On the other hand, to maximize the oil-to-water ratio and the oil production the connecting flow should be closed, while the riser should be fully open. By these results, the valve in the connecting flow should be closed and the valve in riser 1 should be as open as possible without exceeding the gas capacity topside. It is important to keep in mind that these results is highly dependent on the transport coefficients for each well. Changing these coefficients would change the optimal valve positions.

An alternative method to the previous approach to find the optimal valve positions is to formulate an objective function and make a surface plot of it. In this project a simple objective function has been formulated,

$$J = 60m_o - 6m_w - 10m_g. \tag{4.1}$$

These weights in the objective functions should be considered to be an illustrative example and are not based on any real values. The plot of this objective function is presented in figure 4.13 and shows, as the other approach did, that the optimal valve positions are $z_R1 = 1$ and $z_F1 = 0$. An important aspect here is that the gas capacity is not included as a constraint, but the negative weight of amount of gas is included to try to minimize the amount produced.

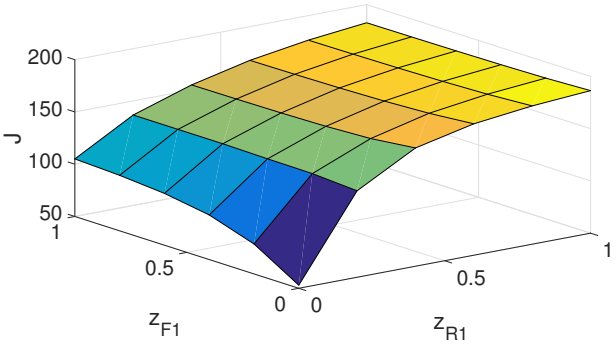


Figure 4.13: Value of the objective function, J, as at different valve openings in connection flow 1 and riser 1, z_{F1} and z_{R1}

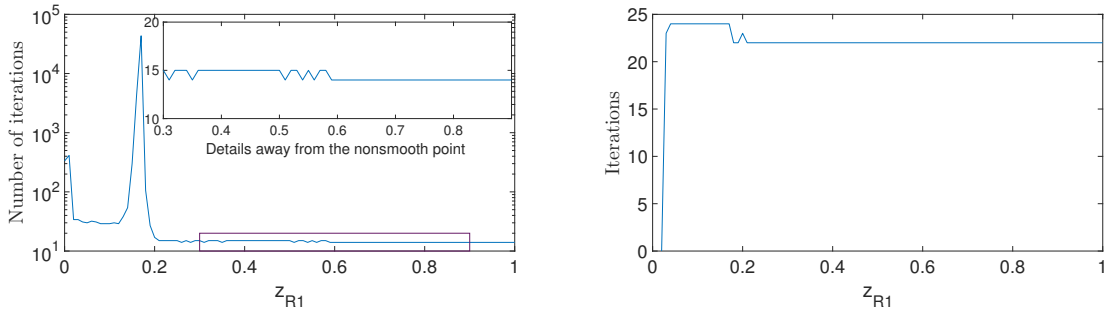
4.4 Convergence of the solvers

Solving the problem for different valve position was done using the Levenberg-Marquardt algorithm (see section 2.5). The number of iterations needed to solve the system varied from 14 to over 43 000 iterations. The number of iterations at the different valve positions is presented in figure 4.14a. The number of iterations needed at $z_{R1} = 0$ is quite high due to lack of a good initial guess. The other initial values fed to the solver is the solution at the previous valve positions. The amount of iterations close to the valve position where the flow direction between manifold 1 and 2 changes direction are significantly higher than for other points. This indicates that the solver have problems converging around the nonsmooth point.

Another method that can be used to obtain a solution is a Newton-type method. The method is very sensitive to the initial guess when trying to solve the oil-well system. When a valve is completely closed, this method has not been able to find the desired solution. The oil well system has shown to have (at least) two solutions for all valve position configurations. One solution is the values presented in the graphs in the previous sections, which is also the desired solution. The other solution is setting every flow rate to zero. The zero flow rate solution is the solution that the Newton-type solver converges to when the initial point is not good enough, while this is not a problem for the LMA. A possible explanation for this is that the desired solution is a very narrow solution, where the equations are stiff. This can lead to that, with a to large step size, the Newton method moves into another convex domain where the zero flow rate solution is located. The Newton-type method used in this project is a slightly altered version of the one presented in equation 2.8. To make it converge for all valve positions, except of completely closed valves, a damping factor of 0.7 was introduced,

$$\mathbf{x}^{(k+1)} = \mathbf{x}^k - 0.7 \cdot \mathbf{G}(\mathbf{x}^k)^{-1}\mathbf{f}(\mathbf{x}^k), \tag{4.2}$$

where the pseudo-inverse of G is used, as the generalized derivative matrix is badly scaled and therefore not easily inverted. A fix to these problems has been to let the LMA find the solution for completely closed valve positions, as well as finding a good initial guess, that can be used to find the solution of the next point were the valve position is slightly adjusted, for the Newton-type solver. As visualized in figure 4.14b, the number of iterations using the Newton-type method is much less than for the LMA close to the nonsmooth point. Further, the Newton-type method uses the same amount of iterations close to the nonsmooth point as for the other points.



(a) Number of iterations needed to solve the system at different valve position in riser 1, z_{R1}, using the Levenberg-Marquardt algorithm.

(b) Number of iterations needed to solve the system at different valve position in riser 1, z_{R1}, using the nonsmooth Newton-like method.

Figure 4.14: Number of iterations needed to solve the system at different valve position in riser 1, z_{R1}, using the nonsmooth Newton-like method and LMA.

Another attribute of a solver is the rate of convergence. By plotting the error, in this project measured by the 1-norm, versus the iterations number, the convergence rate can be visualized. This has been done and is presented in figure 4.15. For both the solvers, the same initial guess, which is close to

the correct solution, was provided to the solver. From the plot it can be seen that the Newton-type solver has quadratic convergence, whilst LMA converges linearly. However, the quadratic convergence of the Newton-type method ”kicks in” at quite small errors (approximately 1e-4), leading to the LMA to converge with fewer iterations.

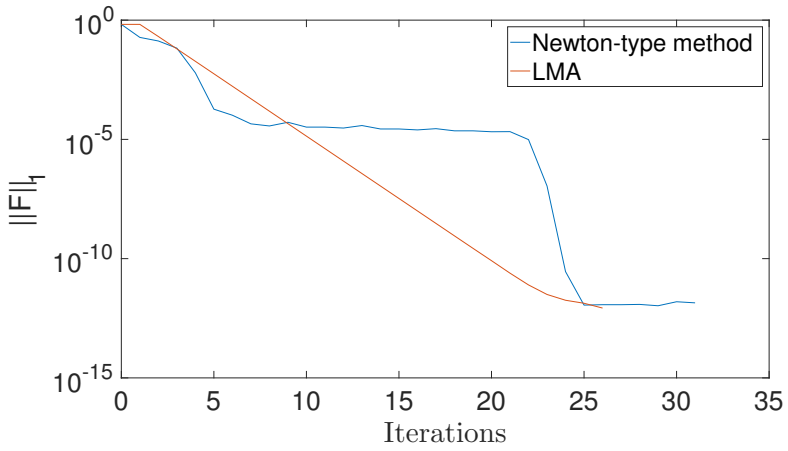


Figure 4.15: The 1-norm of the residual functions as a function of iteration number for the LMA and Newton-type solver.

These results suggests that the LMA is more robust than the Newton-type method. However, the LMA converges slowly when approaching the nonsmooth point compared to the Newton-type method. Which solver to use depends on how big changes are induced to the system. With big changes, the previous solution might not be good enough for the Newton method to find the desired solution, and the LMA has to be used. On the other hand, if the changes are small, the previous solution is good enough for the Newton-type method to find the desired solution. As this method converges faster close to the nonsmooth point, this would be the preferred solver.

4.5 Further discussion

The results presented in the previous sections shows that it is possible to solve a system of nonsmooth equation by using lexicographic derivatives as a replacement for Jacobian elements. The theory presented in chapter 2 is quite complicated mathematical theory. However, the computation of the lexicographic derivative for the absolute function reduces to algorithm 1, which is easily implemented in for example MATLAB. This makes it quite simple to use the nonsmooth theory in practice, at least for $\mathcal{PC}^1$ functions.

The nonsmooth equation, equation 3.21, describing the mass fractions in the manifolds is a good example of where nonsmooth formulations are very useful. Consider a mixing point with more than three flows going in or out. With the formulation used in this project, just another flow has to be added to the equation. The alternative is to use logical statements that checks which flows are going into the point or not. The number of such statements increases rapidly with increasing amount of flows. Therefore, the nonsmooth approach is a good alternative, where all different flow direction scenarios does not need to be considered by the user.

There are also some negative aspects with the use of nonsmooth equations in a model. The main problems are related to the solvers, which is illustrated in the number of iterations needed by the built-in Levenberg-Marquardt algorithm to solve the system at the nonsmooth point. The built-in solvers in software such as MATLAB is more computational economic than user-developed solvers in the same language. In addition, the solvers are often more robust. That such solvers are not necessarily usable for systems of nonsmooth equations can make it harder to solve the system.

Chapter 5

Conclusion and recommendations for further work

This project work has shown that a nonsmooth formulation can be used to model a system with bi-directional flows. The model handles the discrete event where the direction of the flow changes and is able to calculate the composition in the manifold for all different flow directions. The nonsmooth formulations makes it possible to include physical equations that are of nonsmooth nature without the need of smooth approximations.

The Levenberg-Marquardt algorithm was found to be the most robust solver, but has problems converging close to the nonsmooth point. The Newton-type solver on the other hand has no problems converging at the nonsmooth point, but is very sensitive to the initial guess. Taking this into account, the choice of solver should be based on how good initial guesses it is possible to provide the solver.

5.1 Recommendations for further work

There are several parts of this project that can be developed further. For the model of the oil well system, it can be made dynamically and thereby add depletion of the reservoir into the model. By doing so, the model can be simulated over time to see how the production rates changes. Afterwards it can be optimized to find the optimal valve positions to maximize the profit. Another part that can be developed further is the automatic differentiation class, valder. In this project the class is implemented in MATLAB which have quite high run times compared to low-level languages such as C++. This class can be implemented in C++, and be made to communicate with MATLAB to be able to use the existing solvers, which will make it faster. If this is to be done, it should be made in a way which makes it easy for other to use. This will make it possible for others to solve nonsmooth models without having to implement their on automatic differentiation class. This can reduce the barrier of going into nonsmooth analysis, making more people consider using this approach.

Bibliography

[1] M. J. Fetkovich. “The Isochronal Testing of Oil Wells”. In: (1973).

[2] Chriss Grimholt and Sigurd Skogestad. “Optimization of oil field production under gas coning conditions using the optimal closed-loop estimator”. In: *IFAC-PapersOnLine* 48.6 (2015), pp. 39–44.

[3] 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.

[4] 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.

[5] Manolis Lourakis. “A Brief Description of the Levenberg-Marquardt Algorithm Implemened by levmar”. In: 4 (Jan. 2005).

[6] Richard D. Neidinger. “Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming”. In: *SIAM Review* 52.3 (2010), pp. 545–563.

[7] Yu. Nesterov. “Lexicographic differentiation of nonsmooth functions”. In: *Mathematical Programming* 104.2 (2005), pp. 669–700.

[8] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2nd. Springer, 2006.

[9] Paul I. Barton Peter G. Stechlinski Michael Shoham Patrascu. “Nonsmooth Differential-Algebraic Equations in Chemical Engineering”. In: *Computers & Chemical Engineering* (2017).

[10] Liqun Qi and Jie Sun. “A nonsmooth version of Newton’s method”. In: *Mathematical Programming* 58.1 (1993), pp. 353–367.

[11] Stefan Scholtes. *Introduction to Piecewise Differentiable Equations*. Jan. 2012.

[12] Sigurd Skogestad. *Prosessteknikk - Masse- og energibalanser*. Tapir, 2003.

Appendix A

Parameters

Parameter	Value	Unit
Pressures		
p_{r1}	400	[bar]
p_{r1}	400	[bar]
p_{r1}	400	[bar]
p_{S1}	10	[bar]
p_{S2}	10	[bar]
Transport Coefficients		
$k_{o,1}$	$4.567 \cdot 10^{-4}$	[kg $\cdot$ bar^{-2} $\cdot$ s^{-1}]
$k_{o,2}$	$5.966 \cdot 10^{-4}$	[kg $\cdot$ bar^{-2} $\cdot$ s^{-1}]
$k_{o,3}$	$3.793 \cdot 10^{-4}$	[kg $\cdot$ bar^{-2} $\cdot$ s^{-1}]
$k_{w,1}$	$2.322 \cdot 10^{-4}$	[kg $\cdot$ bar^{-2} $\cdot$ s^{-1}]
$k_{w,2}$	$8.572 \cdot 10^{-4}$	[kg $\cdot$ bar^{-2} $\cdot$ s^{-1}]
$k_{w,3}$	$7.160 \cdot 10^{-4}$	[kg $\cdot$ bar^{-2} $\cdot$ s^{-1}]
$k_{g,1}$	$5.721 \cdot 10^{-8}$	[kg $\cdot$ bar^{-4} $\cdot$ s^{-1}]
$k_{g,2}$	$7.110 \cdot 10^{-8}$	[kg $\cdot$ bar^{-4} $\cdot$ s^{-1}]
$k_{g,3}$	$6.219 \cdot 10^{-8}$	[kg $\cdot$ bar^{-4} $\cdot$ s^{-1}]
Heights		
h_1	1000	[m]
h_2	1000	[m]
h_3	1000	[m]
$h_{riser,1}$	1500	[m]
$h_{riser,2}$	1500	[m]
Miscellaneous		
T	373	[K]
M_g	16.04	[kg $\cdot$ kmol^{-1}]
ρ_o	800	[kg $\cdot$ m^{-3}
ρ_w	1000	[kg $\cdot$ m^{-3}
A_{wells}	0.01267	[m^2]
A_{risers}	0.0248	[m^2]
R	$8.314 \cdot 10^{-2}$	m^3 $\cdot$ bar $\cdot$ kmol^{-1} $\cdot$ K^{-1}
C_d	1	(kg $\cdot$ m^{-1} $\cdot$ bar^{-1} $\cdot$ s^{-2}) $^{-0.5}$

Table A.1: The parameters used in the connected oil well model.

Parameter	Value	Unit
Transport Coefficients		
$k_{o,1}$	$1.157 \cdot 10^{-3}$	[kg $\cdot$ bar$^{-2} \cdot$ s^{-1}]
$k_{o,2}$	$1.157 \cdot 10^{-4}$	[kg $\cdot$ bar$^{-2} \cdot$ s^{-1}]
$k_{o,3}$	$5.787 \cdot 10^{-4}$	[kg $\cdot$ bar$^{-2} \cdot$ s^{-1}]
$k_{w,1}$	$1.165 \cdot 10^{-4}$	[kg $\cdot$ bar$^{-2} \cdot$ s^{-1}]
$k_{w,2}$	$1.158 \cdot 10^{-3}$	[kg $\cdot$ bar$^{-2} \cdot$ s^{-1}]
$k_{w,3}$	$4.729 \cdot 10^{-4}$	[kg $\cdot$ bar$^{-2} \cdot$ s^{-1}]
$k_{g,1}$	$1.042 \cdot 10^{-6}$	[kg $\cdot$ bar$^{-4} \cdot$ s^{-1}]
$k_{g,2}$	$7.110 \cdot 10^{-8}$	[kg $\cdot$ bar$^{-4} \cdot$ s^{-1}]
$k_{g,3}$	$1.589 \cdot 10^{-7}$	[kg $\cdot$ bar$^{-4} \cdot$ s^{-1}]

Table A.2: The parameters used in the connected oil well model for the surface response in section 4.3.

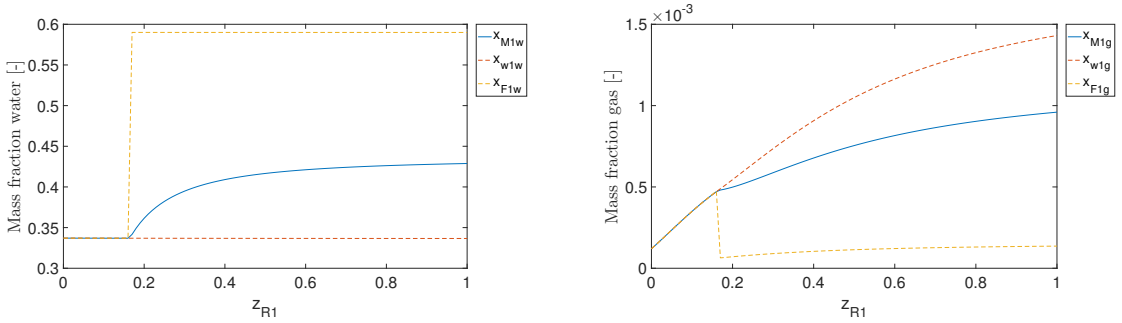
Appendix B

Additional results

In this appendix, additional results from different case studies that are not discussed in the main report are presented.

B.1 Additional composition graphs from changing the valve position in riser 1

In section 4.2 graphs of the oil mass fraction in Manifold 1 was presented. In this section, the corresponding plots for water and gas is given. As for the oil fraction, both the equation for water and gas mass fraction in the manifolds changes as the direction of the flow in the connection flows changes.



(a) Mass fraction water,$x_{M1,w}$

(b) Mass fraction gas, $x_{M1,g}$

Figure B.1: The mass fraction of water and gas plotted against the valve position in riser 1, z_{R1}.

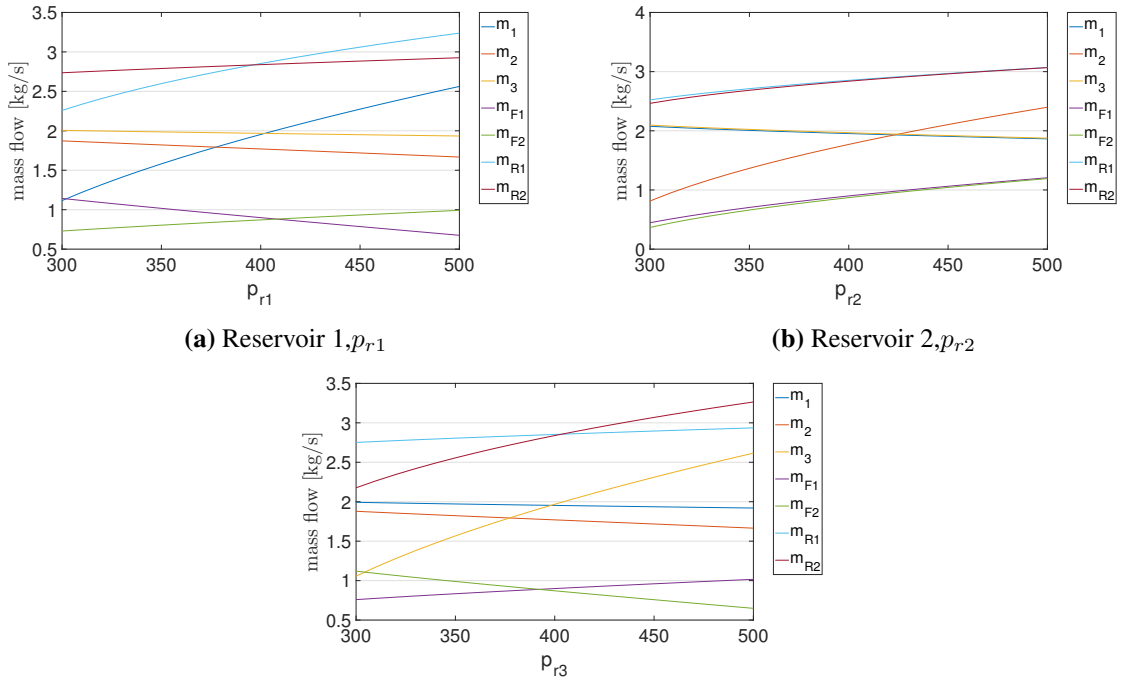
Both the water and gas mass fraction is equal to the corresponding fraction in well 1 for valve openings below 0.165. For higher openings, the mass fraction in manifold 1 is a weighted average between the mass fractions in well 1 and manifold 2.

B.2 Graphs of the total flows by changing the different valves positions and the reservoir pressure

In section 4.1 and 4.2, the results from varying the valve position in well 1 and riser 1 was presented. Here, the resulting flow rates in the different pipe segments by varying the reservoir pressures and the valves are presented. For all the plots presented in this section the valve are set to be half open ($z_i = 0$) and the parameters is equal to the once presented in table A.1.

B.2.1 Reservoir pressures

The reservoir pressures affect the production rate as the pressure difference between the separator and the reservoir set the maximum production rate. It is interesting to examine these results as in a dynamic model, the reservoir will deplete over time. This means that the reservoir pressure will gradually decrease it is produced. From the results in figure B.2 it is possible to see that the reservoir pressure mainly affects the production rate in the corresponding well. However, the production rate from the other wells will slightly increase. The total production rate (the sum of the flow rate in riser 1 and riser 2) will decrease with decreasing reservoir pressure. This is due to reduction in pressure difference between the reservoir and separator.



(a) Reservoir 1,p_{r1}

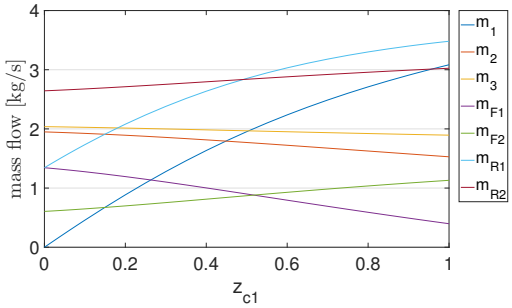
(b) Reservoir 2,p_{r2}

(c) Reservoir 3,p_{r3}

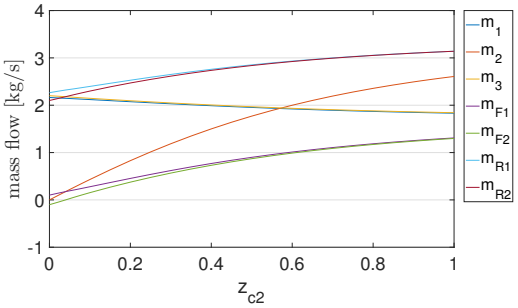
Figure B.2: The total flow rate in each section as a function of the different reservoir pressures.

B.2.2 Valve positions

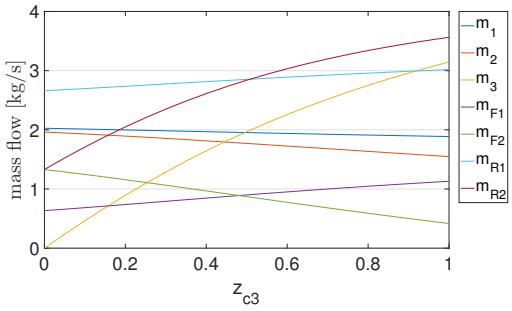
All the valve positions in the system will affect the flow rates in the different pipe segments. In figure B.3 the flow rates for different valve positions are presented. Generally for the valves in the wells, increasing valve opening will increase the production from the corresponding well, but will also slightly decrease the flow rates in the other wells. With all other valves set to half open, only the positions of the valves in the two risers and in well 2 causes a flow to change direction. For riser 1 this is the connection flow between manifold 1 and 2, while for riser 2 and well 2 this is the connection flow between manifold 2 and 3. This is additional results that shows that the model allows bi-directional flow.



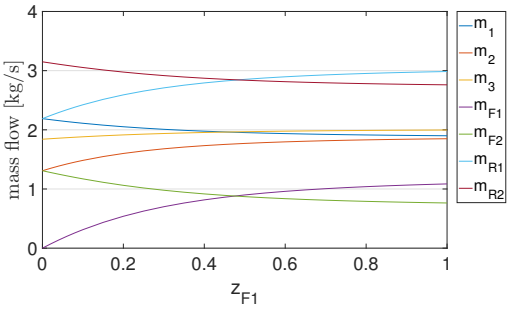
(a) Well 1,z_{c1}



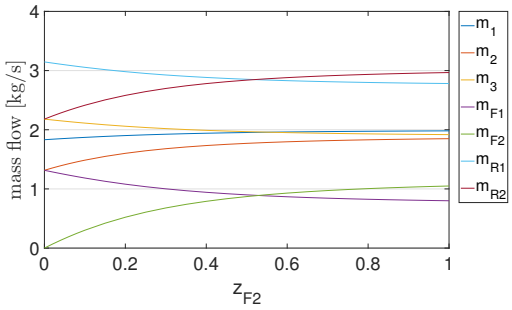
(b) Well 2,z_{c2}



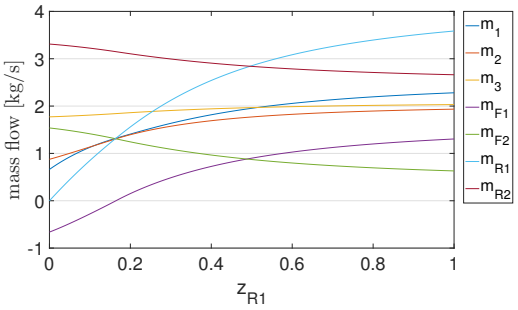
(c) Well 3,z_{c3}



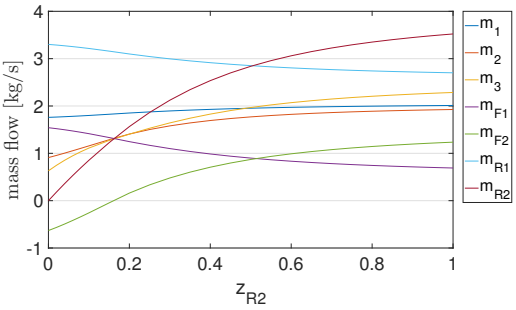
(d) Connection flow 1,z_{F1}



(e) Connection flow 2,z_{F2}



(f) Riser 1,z_{R1}



(g) Riser 2,z_{R2}

Figure B.3: The total flow rate in each section as a function of the different valve positions.

B.3 Values for all variables at standard valve positions, $z_i = 0.5$

Variable	Value	Unit	Variable	Value	Unit
Well 1			Well 2		
$\hat{m}_{w,1,o}$	1.2935	$\text{kg} \cdot \text{s}^{-1}$	$\hat{m}_{w,2,o}$	0.7256	$\text{kg} \cdot \text{s}^{-1}$
$\hat{m}_{w,1,w}$	0.6578	$\text{kg} \cdot \text{s}^{-1}$	$\hat{m}_{w,2,w}$	1.0443	$\text{kg} \cdot \text{s}^{-1}$
$\hat{m}_{w,1,g}$	0.0020	$\text{kg} \cdot \text{s}^{-1}$	$\hat{m}_{w,2,g}$	0.0002	$\text{kg} \cdot \text{s}^{-1}$
GOR_1	0.0016	-	GOR_2	0.0003	-
$p_{wf,1}$	394.4	bar	$p_{wf,2}$	398.5	bar
$p_{wh,1}$	312.7	bar	$p_{wh,2}$	309.5	bar
$\rho_{wh,1}$	854.0	$\text{kg} \cdot \text{m}^{-3}$	$\rho_{wh,2}$	906.6	$\text{kg} \cdot \text{m}^{-3}$
$\rho_{c,1}$	852.7	$\text{kg} \cdot \text{m}^{-3}$	$\rho_{c,2}$	906.4	$\text{kg} \cdot \text{m}^{-3}$
Well 3					
$\hat{m}_{w,3,o}$	0.6813	$\text{kg} \cdot \text{s}^{-1}$			
$\hat{m}_{w,3,w}$	1.2859	$\text{kg} \cdot \text{s}^{-1}$			
$\hat{m}_{w,3,g}$	0.0006	$\text{kg} \cdot \text{s}^{-1}$			
GOR_3	0.0008	-			
$p_{wf,3}$	397.7	bar			
$p_{wh,3}$	307.6	bar			
$\rho_{wh,3}$	919.1	$\text{kg} \cdot \text{m}^{-3}$			
$\rho_{c,3}$	918.7	$\text{kg} \cdot \text{m}^{-3}$			
Manifold 1			Manifold 2		
$p_{M,1}$	201.2	bar	$p_{M,2}$	223.4	bar
$\rho_{M1,w}$	851.4	$\text{kg} \cdot \text{m}^{-3}$	$\rho_{M2,w}$	906.3	$\text{kg} \cdot \text{m}^{-3}$
$\rho_{M1,R}$	867.9	$\text{kg} \cdot \text{m}^{-3}$	$\rho_{M2,1}$	906.3	$\text{kg} \cdot \text{m}^{-3}$
$\rho_{M1,2}$	906.3	$\text{kg} \cdot \text{m}^{-3}$	$\rho_{M2,3}$	906.3	$\text{kg} \cdot \text{m}^{-3}$
$x_{M1,o}$	0.5827	-	$x_{M2,o}$	0.4099	-
$x_{M1,w}$	0.4166	-	$x_{M2,w}$	0.5900	-
$x_{M1,g}$	0.0007	-	$x_{M2,g}$	0.0001	-
Manifold 3					
$p_{M,3}$	202.6	bar			
$\rho_{M3,w}$	918.3	$\text{kg} \cdot \text{m}^{-3}$			
$\rho_{M3,R}$	914.5	$\text{kg} \cdot \text{m}^{-3}$			
$\rho_{M3,2}$	906.3	$\text{kg} \cdot \text{m}^{-3}$			
$x_{M3,o}$	0.3657	-			
$x_{M3,w}$	0.6340	-			
$x_{M3,g}$	0.0003	-			
Connecting flow 1			Connecting flow 1		
$\hat{m}_{F,1,o}$	0.3686	$\text{kg} \cdot \text{s}^{-1}$	$\hat{m}_{F,2,o}$	0.3569	$\text{kg} \cdot \text{s}^{-1}$
$\hat{m}_{F,1,w}$	0.5306	$\text{kg} \cdot \text{s}^{-1}$	$\hat{m}_{F,2,w}$	0.5137	$\text{kg} \cdot \text{s}^{-1}$
$\hat{m}_{F,1,g}$	0.0001	$\text{kg} \cdot \text{s}^{-1}$	$\hat{m}_{F,2,g}$	0.0001	$\text{kg} \cdot \text{s}^{-1}$
$\rho_{F,1}$	906.3	$\text{kg} \cdot \text{m}^{-3}$	$\rho_{F,2}$	906.3	$\text{kg} \cdot \text{m}^{-3}$
Riser 1			Riser 2		
$\hat{m}_{R,1,o}$	1.6621	$\text{kg} \cdot \text{s}^{-1}$	$\hat{m}_{R,2,o}$	1.0382	$\text{kg} \cdot \text{s}^{-1}$
$\hat{m}_{R,1,w}$	1.1884	$\text{kg} \cdot \text{s}^{-1}$	$\hat{m}_{R,2,w}$	1.7997	$\text{kg} \cdot \text{s}^{-1}$
$\hat{m}_{R,1,g}$	0.0022	$\text{kg} \cdot \text{s}^{-1}$	$\hat{m}_{R,2,g}$	0.0007	$\text{kg} \cdot \text{s}^{-1}$
$p_{R,1}$	74.8	bar	$p_{R,2}$	68.5	bar
$\rho_{R,1}$	858.8	$\text{kg} \cdot \text{m}^{-3}$	$\rho_{R,2}$	910.9	$\text{kg} \cdot \text{m}^{-3}$
$\rho_{T,1}$	816.8	$\text{kg} \cdot \text{m}^{-3}$	$\rho_{T,2}$	895.4	$\text{kg} \cdot \text{m}^{-3}$
$\rho_{S,1}$	774.73	$\text{kg} \cdot \text{m}^{-3}$	$\rho_{S,2}$	879.9	$\text{kg} \cdot \text{m}^{-3}$

Table B.1: Values for all variables at standard valve positions, $z_i = 0.5$

Appendix C

MATLAB code

In this appendix, the automatic differentiation class, valder, the oil-well system, and the initialization of values and solvers are presented as MATLAB code.

C.1 valder.m

```
%*********************************************************************
% @author: Marius Reed
% @organization: Department of chemical engineering, NTNU
% @since: 09-01-2017
% @requires: MATLAB R2016a (not tested in other releases)
%*********************************************************************

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(uminus(u.val), uminus(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)
    h = mtimes(u,v^(-1));
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),u.der/u.val);
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

% Creating the midfunction 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,max2(u1,u2));
    s.subs = {2};
    u = subsasgn(u,s,max2(u1,u3));
    s.subs = {3};
    u = subsasgn(u,s,max2(u2,u3));
    h = min(u);
end

% Creating the midfunction for the three objects
function h = midobj(u1,u2,u3)
    s.type = '()';
    s.subs = {1};
    u = valder();
    u = subsasgn(u,s,max2(u1,u2));
    s.subs = {2};
```

```
    u = subsasgn(u,s,max2(u1,u3));
    s.subs = {3};
    u = subsasgn(u,s,max2(u2,u3));

    value = u.val;
    if length(find(value ~= inf)) == 3
        h = min(u);
    else
        index = find(value ~= inf);
        s.subs = {index};
        h = subsref(u,s);
    end
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
```

```
        % Overloading 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

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

C.2 wellSystem.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Department of chemical engineering, NTNU
% @since: 09-01-2017
% @requires: MATLAB R2016a (not tested in other releases)
%*****************************************************************
%% Parameters
par.p_r_1 = 400;                        % Reservoir pressure in well 1 [bar]
par.p_r_2 = 400;                        % Reservoir pressure in well 2 [bar]
par.p_r_3 = 400;                        % Reservoir pressure in well 3 [bar]
par.k_o_1 = 39.456/86400; % Transport coefficient oil in well1 [kg/bar^2*s]
par.k_o_2 = 51.456/86400; % Transport coefficient oil in well2 [kg/bar^2*s]
par.k_o_3 = 32.772/86400; % Transport coefficient oil in well3 [kg/bar^2*s]
par.k_g_1 = 4.943e-3/86400;%Transport coefficient gas in well1 [kg/bar^4*s]
par.k_g_2 = 6.143e-3/86400;%Transport coefficient gas in well2 [kg/bar^4*s]
par.k_g_3 = 5.373e-3/86400;%Transport coefficient gas in well3 [kg/bar^4*s]
par.k_w_1 = 20.064/86400;%Transport coefficient water in well1 [kg/bar^2*s]
par.k_w_2 = 74.064/86400;%Transport coefficient water in well2 [kg/bar^2*s]
par.k_w_3 = 61.86/86400; %Transport coefficient water in well3 [kg/bar^2*s]
par.h_1 = 1000;                                     % Height of well 1 [m]
par.h_2 = 1000;                                     % Height of well 2 [m]
par.h_3 = 1000;                                     % Height of well 3 [m]
par.T = 373;                                           % Temperature[K]
par.M_g = 16.04;                    % Molar mass of gas in well 1 [kg/kmol]
par.rho_o = 800;                                 % Density of oil [kg/m^3]
par.rho_w = 1000;                               % Density of water [kg/m^3]
par.R = 8.314e-2;                          % Gas constant [m^3 bar/(kmol*K)]
par.g = 9.81;                               % Gravitational constant [m/s^2]
par.Cd = 1;                         % Valve coefficient [(kg/(m bar s^2))^.5]
par.A_wells = 0.01267;                  % Cross section valve in wells [m^2]
par.h_riser_1 = 1500;                             % Height of riser 1 [m^2]
par.h_riser_2 = 1500;                             % Height of riser 2 [m^2]
par.A_riser = 0.0248;                   % Cross section valve in riser [m^2]

par.P_S1 = 10;                                  % Pressure in separator 1 [bar]
par.P_S2 = 10;                                  % Pressure in separator 2 [bar]

par.z_R1 = 0.5;                                 % Valve position in riser 1 [-]
par.z_R2 = 0.5;                                 % Valve position in riser 2 [-]
par.z_F1 = 0.5;                     % Valve position in connection pipe 1 [-]
par.z_F2 = 0.5;                     % Valve position in connection pipe 2 [-]
par.z_c1 = 0.5;                                  % Valve position in well 1 [-]
par.z_c2 = 0.5;                                  % Valve position in well 2 [-]
par.z_c3 = 0.5;                                  % Valve position in well 3 [-]
data.par = par;


%% Initial guess used to find solution at standard conditions
% Well 1
z(1) = 2;                                                      % m_w1o [kg/s]
z(2) = 2;                                                      % m_w1w [kg/s]
z(3) = 2;                                                      % m_w1g [kg/s]
z(4) = 1;                                                         % GOR_1 [-]
z(5) = 215;                                                    % P_wf1 [bar]
z(6) = 190;                                                    % P_wh1 [bar]
z(7) = 300;                                                 % rho_wh1 [kg/m^3]
z(8) = 600;                                                  % rho_c1 [kg/m^3]
```

```
% Well 2
z(9) = 2;                                                    % m_w2o [kg/s]
z(10) = 2;                                                   % m_w2w [kg/s]
z(11) = 2;                                                   % m_w2g [kg/s]
z(12) = 1;                                                      % GOR_2 [-]
z(13) = 215;                                                 % P_wf2 [bar]
z(14) = 190;                                                 % P_wh2 [bar]
z(15) = 300;                                              % rho_wh2 [kg/m^3]
z(16) = 600;                                               % rho_c2 [kg/m^3]

% Well 3
z(17) = 2;                                                   % m_w3o [kg/s]
z(18) = 2;                                                   % m_w3w [kg/s]
z(19) = 2;                                                   % m_w3g [kg/s]
z(20) = 1;                                                      % GOR_3 [-]
z(21) = 215;                                                 % P_wf3 [bar]
z(22) = 190;                                                 % P_wh3 [bar]
z(23) = 300;                                              % rho_wh3 [kg/m^3]
z(24) = 600;                                               % rho_c3 [kg/m^3]

% Manifolds
z(25) = 170;                                                    % P_M1 [bar]
z(26) = 300;                                              % rho_M1w [kg/m^3]
z(27) = 300;                                              % rho_M1R [kg/m^3]
z(28) = 300;                                              % rho_M12 [kg/m^3]

z(29) = 180;                                                    % P_M2 [bar]
z(30) = 300;                                              % rho_M2w [kg/m^3]
z(31) = 300;                                              % rho_M21 [kg/m^3]
z(32) = 300;                                              % rho_M23 [kg/m^3]

z(33) = 175;                                                    % P_M3 [bar]
z(34) = 300;                                              % rho_M3w [kg/m^3]
z(35) = 300;                                              % rho_M3R [kg/m^3]
z(36) = 300;                                              % rho_M32 [kg/m^3]

% Connecting flows
z(37) = 1;                                                   % m_F1o [kg/s]
z(38) = 1;                                                   % m_F1w [kg/s]
z(39) = 1;                                                   % m_F1g [kg/s]
z(40) = 600;                                               % rho_F1 [kg/m^3]

z(41) = 1;                                                   % m_F2o [kg/s]
z(42) = 1;                                                   % m_F2w [kg/s]
z(43) = 1;                                                   % m_F2g [kg/s]
z(44) = 600;                                               % rho_F2 [kg/m^3]

% Riser
z(45) = 2;                                                   % m_R1o [kg/s]
z(46) = 2;                                                   % m_R1w [kg/s]
z(47) = 2;                                                   % m_R1g [kg/s]
z(48) = 150;                                                  % P_R1  [bar]

z(49) = 3;                                                   % m_R2o [kg/s]
z(50) = 3;                                                   % m_R2w [kg/s]
z(51) = 3;                                                   % m_R2g [kg/s]
z(52) = 150;                                                  % P_R2  [bar]

z(53) = 200;                                            % rho_R1_top [kg/m^3]
z(54) = 200;                                            % rho_R2_top [kg/m^3]
z(55) = 200;                                               % rho_T1 [kg/m^3]
z(56) = 200;                                               % rho_T2 [kg/m^3]
```

```
z(57) = 200;                                                    % rho_S1 [kg/m^3]
z(58) = 200;                                                    % rho_S2 [kg/m^3]

% Mass fractions
z(59) = 0.33;                                                           % x_M1o
z(60) = 0.33;                                                           % x_M1w
z(61) = 0.33;                                                           % x_M1g
z(62) = 0.33;                                                           % x_M2o
z(63) = 0.33;                                                           % x_M2w
z(64) = 0.33;                                                           % x_M2g
z(65) = 0.33;                                                           % x_M3o
z(66) = 0.33;                                                           % x_M3w
z(67) = 0.33;                                                           % x_M3g
z0 = z';

%% Solving

options = optimoptions(@fsolve,'Display','iter',...
     'MaxIterations',1e5,'MaxFunEvals',1e5,...
     'specifyObjectiveGradient', true,'Algorithm','levenberg-marquardt',...
     'stepTolerance',1e-14,'FunctionTolerance',1e-8);
% Solving using the Levenberg-Marquardt algorithm
z = fsolve(@(z) myFuncs(z,data),z0,options);
```

C.3 myFuncs.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Department of chemical engineering, NTNU
% @since: 09-01-2017
% @requires: MATLAB R2016a (not tested in other releases)
%*****************************************************************
% The function calculates the residual functions F and the generalized
% derivative G.

function [F,G] = myFuncs(z,data)

%% Parameters
par = data.par;
P_r1 = par.p_r_1;                       %Reservoir pressure in well 1 [bar]
P_r2 = par.p_r_2;                       %Reservoir pressure in well 2 [bar]
P_r3 = par.p_r_3;                       %Reservoir pressure in well 3 [bar]
k_o_1 = par.k_o_1;       %Transport coefficient for oil in well 1 [s/bar^2]
k_o_2 = par.k_o_2;       %Transport coefficient for oil in well 2 [s/bar^2]
k_o_3 = par.k_o_3;       %Transport coefficient for oil in well 3 [s/bar^2]
k_g_1 = par.k_g_1;       %Transport coefficient for gas in well 1 [s/bar^4]
k_g_2 = par.k_g_2;       %Transport coefficient for gas in well 2 [s/bar^4]
k_g_3 = par.k_g_3;       %Transport coefficient for gas in well 3 [s/bar^4]
k_w_1 = par.k_w_1;     %Transport coefficient for water in well 1 [s/bar^2]
k_w_2 = par.k_w_2;     %Transport coefficient for water in well 2 [s/bar^2]
k_w_3 = par.k_w_3;     %Transport coefficient for water in well 3 [s/bar^2]
h_1 = par.h_1;                                      %Length of well 1 [m]
h_2 = par.h_2;                                    %Length of in well 2 [m]
h_3 = par.h_3;                                    %Length of in well 3 [m]
T = par.T;                                      %Temperature in well 1 [K]
M_g = par.M_g;                                       %Molar mass [kg/kmol]
rho_o = par.rho_o;                                %Density of oil [kg/m^3]
rho_w = par.rho_w;                              %Density of water [kg/m^3]
R = par.R;                                      %Gas constant [J/(kmol*K)]
g = par.g;                                 %Gravitational constant [m/s^2]
Cd = par.Cd;                          %Valve coefficient [(kg/(m bar s^2))^.5]
A_wells = par.A_wells;                  %Cross section valve in wells [m^2]
h_riser_1 = par.h_riser_1;                        %Height of riser 1 [m^2]
h_riser_2 = par.h_riser_2;                        %Height of riser 2 [m^2]
A_riser = par.A_riser;                  %Cross section valve in riser [m^2]

P_S1 = par.P_S1;                                  %Pressure separator 1 [bar]
P_S2 = par.P_S2;                                  %Pressure separator 2 [bar]
z_R1 = par.z_R1;                                       %Valve position R1 [-]
z_R2 = par.z_R2;                                       %Valve position R2 [-]
z_F1 = par.z_F1;                                       %Valve position F1 [-]
z_F2 = par.z_F2;                                       %Valve position F2 [-]
z_c1 = par.z_c1;                                       %Valve position c1 [-]
z_c2 = par.z_c2;                                       %Valve position c2 [-]
z_c3 = par.z_c3;                                       %Valve position c3 [-]

%% Variables
% Construction valder objects from the variable vector
z = valder(z, eye(length(z)));
% Well 1
m_w1o = z(1);                                     %Oil flow in well 1 [kg/s]
m_w1w = z(2);                                   %Water flow in well 1 [kg/s]
m_w1g = z(3);                                     %Gas flow in well 1 [kg/s]
GOR_1 = z(4);                                   %Gas-oil-ratio in well 1 [-]
```

```
P_wf1 = z(5);                          %Well inflow pressure in well 1 [bar]
P_wh1 = z(6);                             %Wellhead pressure in well 1 [bar]
rho_wh1 = z(7);                       %Density at wellhead in well 1 [kg/m^3]
rho_c1 = z(8);                           %Density at valve in well 1 [kg/m^3]
% Well 2
m_w2o = z(9);                                    %Oil flow in well 2 [kg/s]
m_w2w = z(10);                                 %Water flow in well 2 [kg/s]
m_w2g = z(11);                                   %Gas flow in well 2 [kg/s]
GOR_2 = z(12);                                %Gas-oil-ratio in well 2 [-]
P_wf2 = z(13);                         %Well inflow pressure in well 2 [bar]
P_wh2 = z(14);                            %Wellhead pressure in well 2 [bar]
rho_wh2 = z(15);                      %Density at wellhead in well 2 [kg/m^3]
rho_c2 = z(16);                       %Density at wellhead in well 2 [kg/m^3]
% Well 3
m_w3o = z(17);                                   %Oil flow in well 3 [kg/s]
m_w3w = z(18);                                 %Water flow in well 3 [kg/s]
m_w3g = z(19);                                   %Gas flow in well 3 [kg/s]
GOR_3 = z(20);                                %Gas-oil-ratio in well 3 [-]
P_wf3 = z(21);                         %Well inflow pressure in well 3 [bar]
P_wh3 = z(22);                            %Wellhead pressure in well 3 [bar]
rho_wh3 = z(23);                      %Density at wellhead in well 3 [kg/m^3]
rho_c3 = z(24);                       %Density at wellhead in well 3 [kg/m^3]

% Manifold 1
P_M1 = z(25);                                  %Pressure at manifold 1 [bar]
rho_M1w = z(26);      %Density of stream from well 1 to manifold 1 [kg/m^3]
rho_M1R = z(27);     %Density of stream from manifold 1 to riser 1 [kg/m^3]
rho_M12 = z(28);  %Density of stream from manifold 1 to manifold 2 [kg/m^3]
% Manifold 2
P_M2 = z(29);                                  %Pressure at manifold 2 [bar]
rho_M2w = z(30);      %Density of stream from well 2 to manifold 2 [kg/m^3]
rho_M21 = z(31);  %Density of stream from manifold 2 to manifold 1 [kg/m^3]
rho_M23 = z(32);  %Density of stream from manifold 2 to manifold 3 [kg/m^3]
% Manifold 3
P_M3 = z(33);                                  %Pressure at manifold 2 [bar]
rho_M3w = z(34);      %Density of stream from well 3 to manifold 3 [kg/m^3]
rho_M3R = z(35);     %Density of stream from manifold 3 to riser 2 [kg/m^3]
rho_M32 = z(36);  %Density of stream from manifold 3 to manifold 2 [kg/m^3]

%Connection flows
% Well 2 - Well 1
m_F1o = z(37);               %Oil flow from manifold 2 to manifold 1 [kg/s]
m_F1w = z(38);             %Water flow from manifold 2 to manifold 1 [kg/s]
m_F1g = z(39);               %Gas flow from manifold 2 to manifold 1 [kg/s]
rho_F1 = z(40);         %Density at valve between manifold 1 and 2 [kg/m^3]

% Well 2 - Well 3
m_F2o = z(41);               %Oil flow from manifold 2 to manifold 3 [kg/s]
m_F2w = z(42);             %Water flow from manifold 2 to manifold 3 [kg/s]
m_F2g = z(43);               %Gas flow from manifold 2 to manifold 3 [kg/s]
rho_F2 = z(44);         %Density at valve between manifold 2 and 3 [kg/m^3]

% Riser 1
m_R1o = z(45);                                   %Oil flow in riser 1 [kg/s]
m_R1w = z(46);                                 %Water flow in riser 1 [kg/s]
m_R1g = z(47);                                   %Gas flow in riser 1 [kg/s]
P_R1 = z(48);                           %Density at top of riser 1 [kg/m^3]
% Riser 2
m_R2o = z(49);                                   %Oil flow in riser 2 [kg/s]
m_R2w = z(50);                                 %Water flow in riser 2 [kg/s]
m_R2g = z(51);                                   %Gas flow in riser 2 [kg/s]
P_R2 = z(52);                         %Density at at top of riser 2 [kg/m^3]
```

```
rho_R1_top = z(53);                      %Density at top of riser 1 [kg/m^3]
rho_R2_top = z(54);                      %Density at top of riser 2 [kg/m^3]
rho_T1 = z(55);  %Density at valve between riser 1 and separator 1 [kg/m^3]
rho_T2 = z(56);  %Density at valve between riser 2 and separator 2 [kg/m^3]
rho_S1 = z(57);                            %Density at separator 1 [kg/m^3]
rho_S2 = z(58);                            %Density at separator 2 [kg/m^3]
% Composition in manifolds
x_M1o = z(59);                           % Mass fraction of oil in manifold 1
x_M1w = z(60);                         % Mass fraction of water in manifold 1
x_M1g = z(61);                           % Mass fraction of gas in manifold 1
x_M2o = z(62);                           % Mass fraction of oil in manifold 2
x_M2w = z(63);                         % Mass fraction of water in manifold 2
x_M2g = z(64);                           % Mass fraction of gas in manifold 2
x_M3o = z(65);                           % Mass fraction of oil in manifold 3
x_M3w = z(66);                         % Mass fraction of water in manifold 3
x_M3g = z(67);                           % Mass fraction of gas in manifold 3
%% Equations
f1 = m_w1o - k_o_1*(P_r1^2 - P_wf1^2);
f2 = m_w1w - k_w_1*(P_r1^2 - P_wf1^2);
f3 = m_w1g - GOR_1*m_w1o;
f4 = GOR_1 - k_g_1/k_o_1 * (P_r1 - P_wf1)^2;

f5 = m_w2o - k_o_2*(P_r2^2 - P_wf2^2);
f6 = m_w2w - k_w_2*(P_r2^2 - P_wf2^2);
f7 = m_w2g - GOR_2*m_w2o;
f8 = GOR_2 - k_g_2/k_o_2 * (P_r2 - P_wf2)^2;

f9 = m_w3o - k_o_3*(P_r3^2 - P_wf3^2);
f10 = m_w3w - k_w_3*(P_r3^2 - P_wf3^2);
f11 = m_w3g - GOR_3*m_w3o;
f12 = GOR_3 - k_g_3/k_o_3 * (P_r3 - P_wf3)^2;

f13 = (P_wf1-P_wh1)*((m_w1g * R * T)/M_g + P_wh1*(m_w1o/rho_o + m_w1w/rho_w))...
     - 1e-5*(((m_w1o + m_w1g + m_w1w)*g*P_wh1 * h_1));
f14 = (P_wf2-P_wh2)*((m_w2g * R * T)/M_g + P_wh2*(m_w2o/rho_o + m_w2w/rho_w))...
    - 1e-5*(((m_w2o + m_w2g + m_w2w)*g*P_wh2 * h_2));
f15 = (P_wf3-P_wh3)*((m_w3g * R * T)/M_g + P_wh3*(m_w3o/rho_o + m_w3w/rho_w))...
    - 1e-5*(((m_w3o + m_w3g + m_w3w)*g*P_wh3 * h_3));

f16 = (P_M1 - P_R1)*((m_R1g * R * T)/M_g + P_R1*(m_R1o/rho_o + m_R1w/rho_w))...
    - 1e-5*(((m_R1o + m_R1g + m_R1w)*g*P_R1 * h_riser_1));
f17 = (P_M3-P_R2)*((m_R2g * R * T)/M_g + P_R2*(m_R2o/rho_o + m_R2w/rho_w))...
    - 1e-5*(((m_R2o + m_R2g + m_R2w)*g*P_R2 * h_riser_2));

f18 = rho_R1_top*(m_R1g*R*T/(M_g)+ P_R1*m_R1o/rho_o + P_R1*m_R1w/rho_w)...
    - P_R1*(m_R1o + m_R1g + m_R1w);
f19 = rho_R2_top*(m_R2g*R*T/(M_g)+ P_R2*m_R2o/rho_o + P_R2*m_R2w/rho_w)...
    - P_R2*(m_R2o + m_R2g + m_R2w);

f20 = rho_S1*(m_R1g*R*T/(M_g)+ P_S1*m_R1o/rho_o + P_S1*m_R1w/rho_w)...
    - P_S1*(m_R1o + m_R1g + m_R1w);
f21 = rho_S2*(m_R2g*R*T/(M_g)+ P_S2*m_R2o/rho_o + P_S2*m_R2w/rho_w)...
    - P_S2*(m_R2o + m_R2g + m_R2w);
f22 = rho_T1 - 0.5*(rho_R1_top + rho_S1);
f23 = rho_T2 - 0.5*(rho_R2_top + rho_S2);

f24 = (m_R1o + m_R1g + m_R1w)*abs(m_R1o + m_R1g + m_R1w)...
    - z_R1^2*Cd^2 *A_riser^2*rho_T1*(P_R1-P_S1);
f25 = (m_R2o + m_R2g + m_R2w)*abs(m_R2o + m_R2g + m_R2w)...
    - z_R2^2*Cd^2 *A_riser^2*rho_T2*(P_R2-P_S2);
f26 = rho_M1R*((m_R1g *R *T)/(M_g)+ P_M1*m_R1o/rho_o + P_M1*m_R1w/rho_w)...
```

```
    - P_M1*(m_R1o + m_R1g + m_R1w);
f27 = rho_M3R*((m_R2g *R *T)/(M_g)+ P_M3*m_R2o/rho_o + P_M3*m_R2w/rho_w)...
    - P_M3*(m_R2o + m_R2g + m_R2w);

% Mass balance manifolds
% M1
f28 = m_w1o + m_F1o - m_R1o;
f29 = m_w1g + m_F1g - m_R1g;
f30 = m_w1w + m_F1w - m_R1w;

% M2
f31 = m_w2o - m_F2o - m_F1o;
f32 = m_w2g - m_F2g - m_F1g;
f33 = m_w2w - m_F2w - m_F1w;

% M3
f34 = m_w3o + m_F2o - m_R2o;
f35 = m_w3g + m_F2g - m_R2g;
f36 = m_w3w + m_F2w - m_R2w;

f37 = rho_M12*(m_F1g*R*T/(M_g)+ P_M1*m_F1o/rho_o + P_M1*m_F1w/rho_w)...
    - P_M1*(m_F1o + m_F1g + m_F1w);
f38 = rho_M21*(m_F1g*R*T/(M_g)+ P_M2*m_F1o/rho_o + P_M2*m_F1w/rho_w)...
    - P_M2*(m_F1o + m_F1g + m_F1w);

f39 = rho_M23*(m_F2g*R*T/(M_g)+ P_M2*m_F2o/rho_o + P_M2*m_F2w/rho_w)...
    - P_M2*(m_F2o + m_F2g + m_F2w);
f40 = rho_M32*(m_F2g*R*T/(M_g)+ P_M3*m_F2o/rho_o + P_M3*m_F2w/rho_w)...
    - P_M3*(m_F2o + m_F2g + m_F2w);

f41 = rho_F1 - 0.5*(rho_M12 + rho_M21);
f42 = rho_F2 - 0.5*(rho_M23 + rho_M32);

f43 = (m_F1o + m_F1g + m_F1w)*abs(m_F1o + m_F1g + m_F1w)...
    - z_F1^2*Cd^2*A_wells^2*rho_F1 * (P_M2 - P_M1);
f44 = (m_F2o + m_F2g + m_F2w)*abs(m_F2o + m_F2g + m_F2w)...
    - z_F2^2*Cd^2*A_wells^2*rho_F2 * (P_M2 - P_M3);

f45 = rho_wh1*(m_w1g*R*T/(M_g)+ P_wh1*m_w1o/rho_o + P_wh1*m_w1w/rho_w)...
    - P_wh1*(m_w1o+m_w1g + m_w1w);
f46 = rho_wh2*(m_w2g*R*T/(M_g)+ P_wh2*m_w2o/rho_o + P_wh2*m_w2w/rho_w)...
    - P_wh2*(m_w2o+m_w2g + m_w2w);
f47 = rho_wh3*(m_w3g*R*T/(M_g)+ P_wh3*m_w3o/rho_o + P_wh3*m_w3w/rho_w)...
    - P_wh3*(m_w3o + m_w3g + m_w3w);

f48 = rho_M1w*(m_w1g*R*T/(M_g)+ P_M1*m_w1o/rho_o + P_M1*m_w1w/rho_w)...
    - P_M1*(m_w1o+m_w1g + m_w1w);
f49 = rho_M2w*(m_w2g*R*T/(M_g)+ P_M2*m_w2o/rho_o + P_M2*m_w2w/rho_w)...
    - P_M2*(m_w2o+m_w2g + m_w2w);
f50 = rho_M3w*(m_w3g*R*T/M_g + P_M3*m_w3o/rho_o + P_M3*m_w3w/rho_w)...
    - P_M3*(m_w3o + m_w3g + m_w3w);

f51 = rho_c1 - 0.5*(rho_wh1 + rho_M1w);
f52 = rho_c2 - 0.5*(rho_wh2 + rho_M2w);
f53 = rho_c3 - 0.5*(rho_wh3 + rho_M3w);

f54 = (m_w1o + m_w1g + m_w1w)*abs(m_w1o + m_w1g + m_w1w) - ...
    z_c1^2 * Cd^2 * A_wells^2 * rho_c1 * (P_wh1 - P_M1);
f55 = (m_w2o + m_w2g + m_w2w)*abs(m_w2o + m_w2g + m_w2w) - ...
    z_c2^2 * Cd^2 * A_wells^2 * rho_c2 * (P_wh2 - P_M2);
f56 = (m_w3o + m_w3g + m_w3w)*abs(m_w3o + m_w3g + m_w3w) - ...
    z_c3^2 * Cd^2 * A_wells^2 * rho_c3 * (P_wh3 - P_M3);
```

```
m_w1 = m_w1o + m_w1w + m_w1g;
m_w2 = m_w2o + m_w2w + m_w2g;
m_w3 = m_w3o + m_w3w + m_w3g;
m_R1 = m_R1o + m_R1w + m_R1g;
m_R2 = m_R2o + m_R2w + m_R2g;
m_F1 = m_F1o + m_F1w + m_F1g;
m_F2 = m_F2o + m_F2w + m_F2g;

f57 = max2(m_w1o,0) + max2(-m_R1o,0) + max2(m_F1o,0) -...
    (max2(m_w1,0) + max2(-m_R1,0) + max2(m_F1,0))*x_M1o;
f58 = max2(m_w1w,0) + max2(-m_R1w,0) + max2(m_F1w,0) -...
    (max2(m_w1,0) + max2(-m_R1,0) + max2(m_F1,0))*x_M1w;
f59 = x_M1w +x_M1g + x_M1o - 1;

f60 = max2(m_w3o,0) + max2(-m_R2o,0) + max2(m_F2o,0) -...
    (max2(m_w3,0) + max2(-m_R2,0) + max2(m_F2,0))*x_M3o;
f61 = max2(m_w3w,0) + max2(-m_R2w,0) + max2(m_F2w,0) -...
    (max2(m_w3,0) + max2(-m_R2,0) + max2(m_F2,0))*x_M3w;
f62 = x_M3w + x_M3g + x_M3o - 1;

dP_12 = P_M2 - P_M1;
dP_32 = P_M2 - P_M3;

f63 = dP_12^2*m_F1o + (min2(dP_12,0)*-dP_12*x_M1o...
    + min2(-dP_12,0)*dP_12 * x_M2o)*m_F1;
f64 = dP_12^2*m_F1w + (min2(dP_12,0)*-dP_12*x_M1w...
    + min2(-dP_12,0)*dP_12 * x_M2w)*m_F1;
f65 = dP_12^2*m_F1g + (min2(dP_12,0)*-dP_12*x_M1g...
    + min2(-dP_12,0)*dP_12 * x_M2g)*m_F1;

f66 = dP_32^2*m_F2o + (min2(dP_32,0)*-dP_32*x_M3o...
    + min2(-dP_32,0)*dP_32 * x_M2o)*m_F2;
f67 = dP_32^2*m_F2w + (min2(dP_32,0)*-dP_32*x_M3w...
    + min2(-dP_32,0)*dP_32 * x_M2w)*m_F2;

%% 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);...
    getVal(f37);getVal(f38);getVal(f39);getVal(f40);getVal(f41);...
    getVal(f42);getVal(f43);getVal(f44);getVal(f45);getVal(f46);...
    getVal(f47);getVal(f48);getVal(f49);getVal(f50);getVal(f51);...
    getVal(f52);getVal(f53);getVal(f54);getVal(f55);getVal(f56);...
    getVal(f57);getVal(f58);getVal(f59);getVal(f60);getVal(f61);...
    getVal(f62);getVal(f63);getVal(f64);getVal(f65);getVal(f66);...
    getVal(f67)];

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);...
    getDer(f37);getDer(f38);getDer(f39);getDer(f40);getDer(f41);...
    getDer(f42);getDer(f43);getDer(f44);getDer(f45);getDer(f46);...
    getDer(f47);getDer(f48);getDer(f49);getDer(f50);getDer(f51);...
```

```
    getDer(f52);getDer(f53);getDer(f54);getDer(f55);getDer(f56);...
    getDer(f57);getDer(f58);getDer(f59);getDer(f60);getDer(f61);...
    getDer(f62);getDer(f63);getDer(f64);getDer(f65);getDer(f66);...
    getDer(f67)];

end
```

C.4 newtonMethod.m

```
%*****************************************************************
% @author: Marius Reed
% @organization: Department of chemical engineering, NTNU
% @since: 09-01-2017
% @requires: MATLAB R2016a (not tested in other releases)
%*****************************************************************
% Semi smooth newton-type method. Finding the z such that the norm of the
% residual F is below the given tolerance

function [z,iter,error] = newtonMethod(z0,data,tol)
    z = z0;
    F = myFuncs_Newton(z,data);              % Finding the initial residual
    iter = 0;                                      % Iterations set to zero
    error = norm(F,1);                            % Calulating inital error
    while norm(F,1) > tol                        % Checking for convergence
        [F,G] = myFuncs_Newton(z,data); % Computing residual and derivative
        z = z - 0.7*pinv(G)*F;                      % Performs one iteration
        iter = iter+1;
        if mod(iter,100) == 0          % If number of iterations exceeds 100
             tol = tol*10;                    % reduce convergence tolerance
        end
        error = [error norm(F,1)];               % Saving the current error
    end
end
```