Coupling between a reservoir and a surface facilities network

Elvera Boogaart

Delft University of Technology

TU Delft

Coupling between a reservoir and a surface facilities network

by

Elvera Boogaart

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday February 9, 2016 at 11:00 AM.

Student number:	4096703	
Project duration:	April, 2015 – February, 2016	
Thesis committee:	Dr. ir. J.E. Romate,	TU Delft, supervisor
	Prof. dr. ir. C. Vuik,	TU Delft
	Dr. ir. W.T. van Horssen,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This report is written to fulfill the requirements for the degree of Master of Science in Applied Mathematics at the Delft University of Technology. The first three months of the project were dedicated to a literature study and this report is a sequel of the literature study.

Motivation

The production of oil and gas from a reservoir is aided by the use of simulation models. Reservoir simulators are used to predict the oil and gas flow out of the reservoir. Oil and gas flowing out of a reservoir enter a production network. This network flow is modeled in network simulators. In reality the reservoir and the network form one flow system, and for certain studies it is desirable to simulate the reservoir and network also as one system. If a reservoir simulator and a network simulator are available already, it is required that they be coupled together. The coupling method, however, determines how then the system as a whole will behave. Therefore the coupling method is required to lead to a stable and efficient simulation model of the combined system. The question then is how to achieve this.

Main Findings

After investigating all different possible problems we could think of individually and looking into some algorithms for minimization and to solve nonlinear systems, a stable coupling method is derived.

Acknowledgements

I would like to express my gratitude towards my supervisor Johan Romate, and thank him for the opportunity to do my Master thesis about such an interesting subject. I also want to thank him for his help, support, advice and guidance during my thesis project. I have worked on this project with great pleasure, due to the interesting and sometimes challenging assignment.

Further, I would also like to express my gratitude and thanks to my family and friends for their encouragement, interest and support during the entire time of this project. This with special gratitude towards Rudi van de Velde, I could not have done it without his support and encouragement.

Delft, January 2016

Elvera Boogaart

Contents

List of Figures v

List of Tables vii

1 Introduction 1
- 1.1 Structure of the Report . . . 2

2 Nodal Analysis 3
- 2.1 Inflow Performance Relationship (IPR) . . . 6
- 2.2 Tubing Performance Relationship (TPR) . . . 6
- 2.3 Pipe Performance Relationship (PPR) . . . 6

3 Oil Flow 7
- 3.1 IPR for Single-Phase Liquid Reservoirs . . . 7
  - 3.1.1 Transient Flow . . . 7
  - 3.1.2 Steady-State Flow . . . 8
  - 3.1.3 Pseudo Steady-State Flow . . . 9
- 3.2 TPR for Single-Phase Liquid Flow . . . 9
- 3.3 PPR for Single-Phase Liquid Flow . . . 9
- 3.4 Properties of Oil . . . 10
  - 3.4.1 Density of Oil . . . 10
  - 3.4.2 Formation Volume Factor of Oil . . . 10
- 3.5 Fanning Friction Factor . . . 11

4 Gas Flow 13
- 4.1 IPR for Single-Phase Gas Reservoirs . . . 13
  - 4.1.1 Steady-State Flow . . . 13
  - 4.1.2 Pseudo Steady-State Flow . . . 14
  - 4.1.3 Transient Flow . . . 14
- 4.2 TPR for Single-Phase Gas Flow . . . 14
- 4.3 PPR for Single-Phase Gas Flow . . . 15
- 4.4 Properties of Gas . . . 15
  - 4.4.1 Gas Compressibility Factor . . . 15
  - 4.4.2 Gas Pseudo-reduced and Pseudo-critical Pressure and Temperature . . . 16
  - 4.4.3 Density of Gas . . . 16

5 Two-Phase Flow 17
- 5.1 IPR for Two-Phase Flow . . . 17
- 5.2 TPR for Two-Phase Flow . . . 18
  - 5.2.1 Homogeneous-Flow Models . . . 18
  - 5.2.2 Separated-Flow Models . . . 19
- 5.3 PPR for Two-Phase Flow . . . 21

6 Modeling 23
- 6.1 Subsurface Response Function . . . 23
- 6.2 Surface Response Function . . . 26
  - 6.2.1 Single Well . . . 26
  - 6.2.2 Multiple Wells . . . 26
- 6.3 Operating Point . . . 27
  - 6.3.1 A One-Well System . . . 27
  - 6.3.2 A Multiple-Well System . . . 28

6.4 Object Oriented Programming . . . 29
6.5 Example System Calculation . . . 29

7 Possible Problems **31**
7.1 The IPR and TPR have Two Intersections . . . 31
7.1.1 Test . . . 32
7.1.2 To the Right Intersection Point . . . 33
7.1.3 If there is No Right Intersection Point . . . 35
7.2 The IPR and TPR have No Intersections . . . 36
7.2.1 The TPR Curve is To Short . . . 36
7.2.2 The TPR Curve lies Above the IPR Curve . . . 37

8 Optimization **41**
8.1 The Tubing-Head Choke . . . 41
8.2 Optimization Formulation . . . 42
8.3 Example Case . . . 43

9 Conclusion **45**

A Correlations in SI Units **47**
A.1 Oil . . . 47
A.1.1 IPR . . . 47
A.1.2 TPR . . . 48
A.2 Gas . . . 49
A.2.1 IPR . . . 49
A.2.2 TPR . . . 50
A.2.3 PPR . . . 50

B List of Conversion Factors **51**

C Algorithms **53**
C.1 *fmincon* Algorithm . . . 53
C.2 *fminunc* Algorithm . . . 53
C.3 *fsolve* Algorithm . . . 54
C.4 *fzero* Algorithm . . . 54

D Test Data **55**
D.1 System Test . . . 55
D.2 Test SSRF 1: No Intersections . . . 56
D.2.1 Data IPR . . . 56
D.2.2 Data TPR . . . 57
D.3 Test SSRF 2: Two Intersections . . . 58
D.3.1 Data IPR . . . 58
D.3.2 Data TPR . . . 59

E MATLAB Codes **61**
E.1 Functions . . . 61
E.2 Parameter objects . . . 62
E.3 Subsurface . . . 64
E.4 Surface . . . 72
E.5 System . . . 77
E.5.1 Non-Linear System of Equations . . . 78
E.5.2 Optimization . . . 81

Bibliography **85**

List of Figures

2.1 Location of various nodes.(Beggs [2]) . . . 3
2.2 Determination of flow capacity. (Beggs [2]) . . . 4
2.3 Simple producing system. . . . 5

3.1 The Darcy-Weisbach friction factor diagram. (Guo et al. [7]) . . . 11
3.2 The different equations to describe the Darcy-Weisbach friction factor diagram. . . . 12

6.1 A single well. . . . 24
6.2 An example of an IPR and TPR curve for a certain tubing-head pressure. The SSRF will return the flow rate at the intersection point. . . . 25
6.3 The resulting SSRF curve. . . . 25
6.4 Example system for a system with a single well. . . . 26
6.5 Example system for a system with multiple wells. . . . 27
6.6 An example of an SSRF curve and an SRF curve. . . . 28

7.1 The IPR and TPR have two intersections. . . . 31
7.2 The test for determining whether or not the right intersection point is found. . . . 32
7.3 The second intersection point has to be inside the boxed area. . . . 33
7.4 This is how the diffference function f works. . . . 34
7.5 The TPR curve is to short to have a second intersection point. . . . 35
7.6 The TPR curve is to short to have an intersection point. . . . 36
7.7 The TPR curve is to short and would intersect the IPR curve at a negative flow rate. . . . 37
7.8 No intersections because the entire TPR curve lies above the IPR curve and a possible solution to this problem. . . . 37
7.9 In some situations the horizontal extension doens't seems to be the most logical solution. . . . 38
7.10 The Residual vector. . . . 39
7.11 The shortest distance solution. . . . 40

8.1 The pressure drop caused by a choke. . . . 41

List of Tables

5.1 Beggs and Brill holdup constants. (Economides et al. [4]) 21

D.1 Parameter for the system test in SI units. 55
D.2 Data IPR for Test 1: No intersections. 56
D.3 Data TPR for Test 1: No intersections. 57
D.4 Data IPR for Test 2: Two intersections. 58
D.5 Data TPR for Test 2: Two intersections. 59

1 Introduction

In this study a reservoir takes the form of a collection of very simple tanks, each one having one well. We take one or more of these as reservoir simulator. For the network we take a simple algebraic system of equations representing flow from the wells into one outlet. This is the network simulator. For the coupling we consider explicit coupling. The question is: under what conditions can it give a stable and efficient method for the combined simulation.

The coupling between a reservoir production well and the inflow point of the network is assumed to be located at the tubing head. This is a location at the surface where the well tubing coming from the reservoir below ends in the main valve on top of the well. Here the fluid flow is assumed to be expressed in terms of pressures and flow rates only. On the coupling point (the tubing head) the well flow rate is a function of the tubing-head pressure, given the bottom-hole pressure: the tubing performance curve. On the coupling point the pressure is also related to the flow rate on the network side. Given the outflow pressure of the network (typically there is only one outflow point) the inflow pressure is a function of the inflow rate, which could be called the surface response function. When coupled the flow rate and pressure at the coupling is solved from these two equations. Problems that occur and need to be addressed are the non-existence of a solution and the non-convergence of the solution process. Non-existence may occur because physically there is no solution (the curves have no intersection point) or it may occur because the numerically determined curves cannot be calculated at the required flow rates and/or pressures. A possible physical issue occurs when the reservoir pressure gets too low to sustain an upward flow in a well. This is called lift die-out. The well then needs to be closed (shut-in). An important feature of the process calculating the coupling conditions is that the two equations from which the conditions are to be determined are not given functions, but are implicitly given by the results given by the network solver and the reservoir simulator. It may be that these two solvers can give the values but not any derivatives, so that a Newton-type solution process may not be possible.

In practice it may be possible to avoid doing complete reservoir simulations to calculate values for the coupling, by approximating the reservoir model by a table or a curve for any given reservoir pressure. This is what we will do here to mimic the reservoir side.

For the surface network we assume a converging network. All well inflow points are connected to pipes which are connected in groups at the outflow side (pipe junctions). Each junction has one outflow pipe and these outflow pipes come together again in the final junction of the network. This final junction has again one outflow pipe to the outflow point of the network (which is the fixed pressure point). At the junctions mass balance laws are imposed, and for each pipe the flow rate through the pipe is related via a simple equation to the pressure drop over the pipe.

The goal now is to develop a robust solution process to calculate the coupling conditions (rate and pressure) at each well when connecting a reservoir with multiple wells to a converging surface network. The surface network model and the reservoir model up to the well head are assumed to be separate software implementations, which can give values at the coupling points, but not necessarily derivatives.

1.1. Structure of the Report

First a procedure called Nodal Analysis is explained in Chapter 2. This procedure is used to analyze the performance of a well system. Then the different kind of correlations needed for this procedure are summarized, for Oil Flow in Chapter 3, for Gas Flow in Chapter 4 and for Two-Phase Flow in Chapter 5. Next, we start assembling our entire system. This is done in Chapter 6. Then, in Chapter 7, are all possible problems we could think of solved, to obtain a robust solution process. In Chapter 8, the problem is translated into a optimization problem and a choke is added to the model to make it possible to impose an maximal flow rate out of the system. And, in Chapter 9 this report is finished with a conclusion.

2

Nodal Analysis

To analyze the performance of a well system we want to use a type of analyzing that is called Nodal analysis. Nodal analysis has been applied for this purpose for many years as we can see from Beggs [2] and Guo et al. [7]. Beggs [2] gives us a good idea of the approach which will be summarized here.

With Nodal analysis a flow system is divided in several smaller parts. These smaller parts can than individually be solved. The procedure starts with selecting a division point or node in the well and dividing the system at this point. All of the component upstream of the node cover the inflow section, while the outflow section consists of all of the components downstream of the node. Figure 2.1 from Beggs [2] shows the location of the nodes which are commonly used.

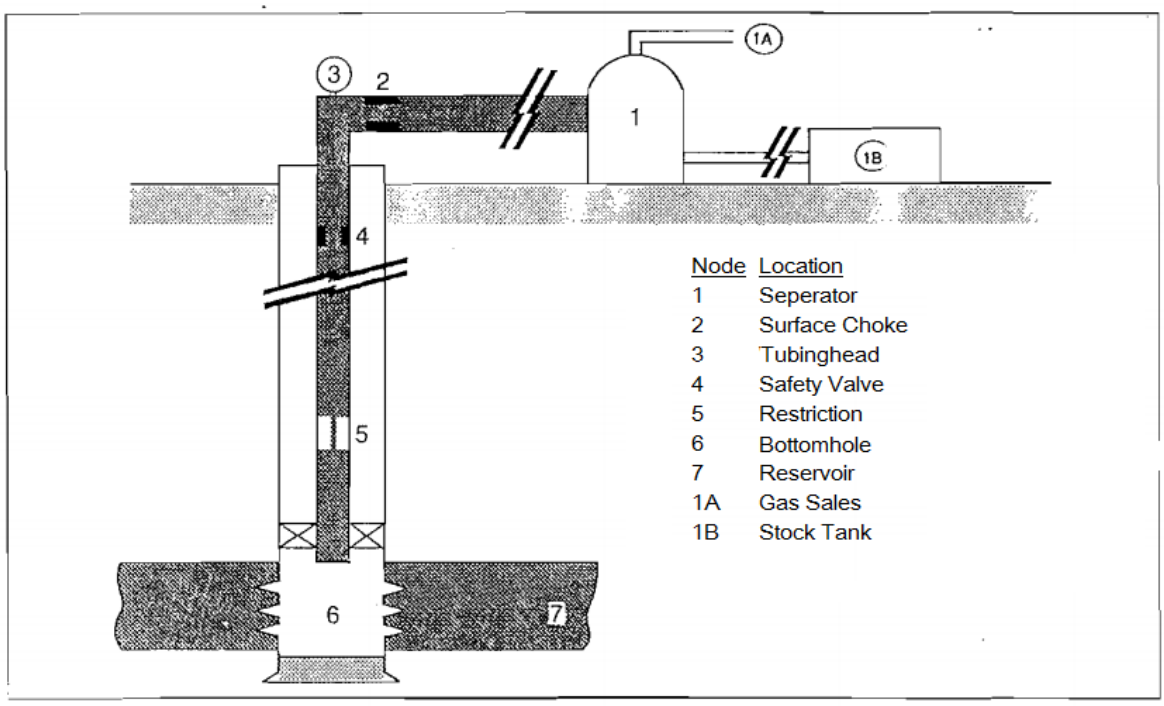


Figure 2.1: Location of various nodes.(Beggs [2])

If a relationship between flow rate and pressure drop is available for each component in the system the flow rate through the entire system can be determined. To do so the following requirements, listed from Beggs [2], must be satisfied:

1. Flow into the node equals flow out of the node

2. Only one pressure can exist at a node

Two pressures remain fixed and are not functions of flow rate. The first one is the average reservoir pressure p_{res} , and the other is the system outlet pressure p_{out}.

After selecting a node, the pressure in that node can be calculated starting with the average reservoir pressure and the system outlet pressure. From the flow into the node it follows that [2]:

$$p_{Res} - \Delta p \text{ (upstream components)} = p_{node}$$

From the flow out of the node it follows that [2]:

$$p_{out} + \Delta p \text{ (downstream components)} = p_{node}$$

The pressure drop, varies with flow rate. A plot of node pressure versus flow rate will thus produce two curves, the intersection of which will give the conditions satisfying the requirements which were stated before (see Figure 2.2 [2]).

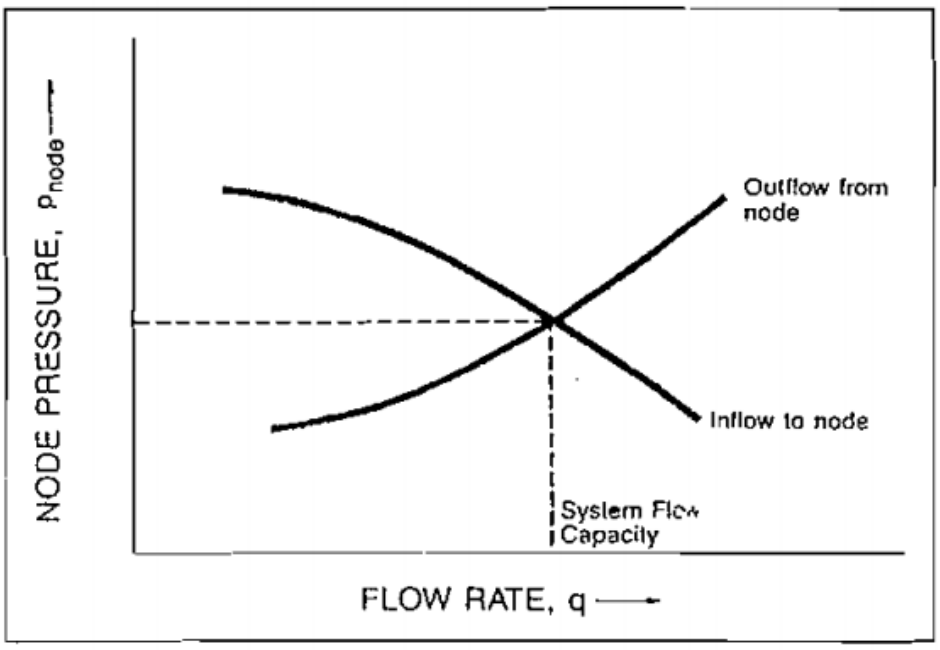


Figure 2.2: Determination of flow capacity. (Beggs [2])

The pressure-rate relation of upstream equipment is called ‘inflow performance curve’, the pressure-rate relation of downstream equipment is called ‘outflow performance curve’. The intersection of the two curves defines the operating point, that is, operating flow rate and pressure, at the specified node. Because pressure data is normally measured at either the bottom hole or the tubing head, Nodal analysis is usually conducted using the bottom hole or tubing head as the solution node.

The effect of a change in any of the components can be analyzed by recalculating the node pressure versus flow rate. If a change was made in an upstream component, the outflow curve will remain unchanged. But, if one of the curves is changed, the intersection will shift, and a new operating point will exist. The curves will also shift if either one of the fixed pressures is changed.

The procedure can be further illustrated by considering the simple system shown in Figure 2.3 and selecting the tubing head as the node.

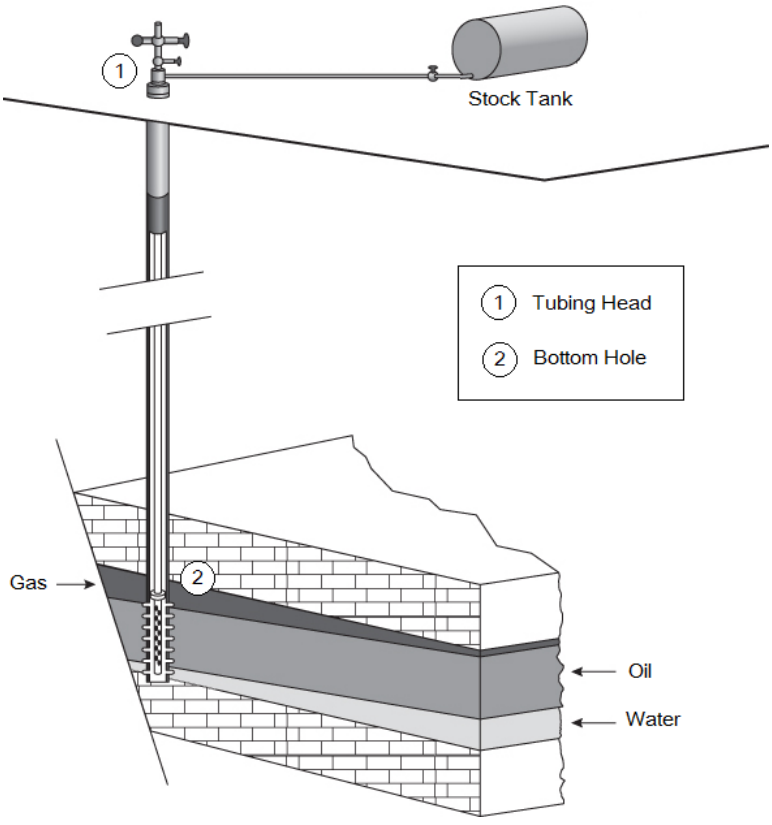


Figure 2.3: Simple producing system.

Then the node pressure can be calculated from both directions starting at the fixed pressures.

Inflow to the node:

$$p_{res} - \Delta p_{res} - \Delta p_{tubing} = p_{TH}$$

Outflow from the node:

$$p_{out} + \Delta p_{surface} = p_{TH}$$

Where

p_{res} = average reservoir pressure
p_{TH} = tubing-head pressure
p_{out} = outflow pressure at the stock tank
$\Delta p_{surface}$ = pressure drop due to horizontal flow trough the surface facilities
Δp_{res} = pressure drop due to flow through porous media in the reservoir
Δp_{tubing} = pressure drop due to vertical or inclined tubing

2.1. Inflow Performance Relationship (IPR)

To use Nodal analysis a relationship between flow rate and pressure drop must be available for each component in the system. When the bottom-hole is used as a solution node in Nodal analysis, the inflow performance is the well Inflow Performance Relationship (IPR).

IPR is used for evaluating reservoir deliverability. The IPR is a presentation of the relation between the bottom-hole pressure and flow rate depending on the average reservoir pressure. Well IPR curves are usually constructed using reservoir inflow models. Some models found in Guo et al. [7] and Economides et al. [4] will be used in the following chapters.

If the bottom-hole pressure is given, the flow rate can be calculated. However, the bottom-hole pressure is a function of the tubing-head pressure, which, in turn, can depend on a lot of other things. What a well thus actually produces depends on what the reservoir can deliver but also on what the wellbore hydraulics would allow.

2.2. Tubing Performance Relationship (TPR)

When the bottom-hole is used as a solution node in Nodal analysis, the outflow performance is the Tubing Performance Relationship (TPR). The IPR describes the reservoir deliverability but the achievable oil flow rate from a well is determined by tubing-head pressure and the flow performance of production string. The flow performance of production string depends on characteristics of the production string and properties of the fluids. Wellbore performance analysis involves establishing a relationship between tubing-head and bottom-hole pressure and fluid production rate.

2.3. Pipe Performance Relationship (PPR)

When the tubing head is used as a solution node in Nodal analysis, the inflow performance curve is the 'Subsurface Response Function' (SSRF), which is obtained by transforming the IPR to tubing head through the TPR. The outflow performance is given by a relationship which describes the flow through the surface network. Because the surface network mainly consists of pipelines a relationship is needed for the flow in pipelines. This relationship will be called the Pipe Performance Relationship (PPR).

3

Oil Flow

The fluids in oil wells can include oil, water, and gas. Correlations will be given for use in the computer model for singe-phase oil and gas flow and for two-phase flow. In this chapter the correlations for oil flow will be listed in field units, for a conversion to SI units the conversion factor from Appendix B can be used. For convenience the converted correlation for oil and gas flow in SI units are listed in Appendix A.

Because single-phase oil flow only exists when the tubing-head pressure is above bubble-point pressure this is usually not the case. But, we start with single-phase oil flow to improve our notion of the concept. For units of various quantities see Appendix B

3.1. IPR for Single-Phase Liquid Reservoirs

Mathematical models for reservoir deliverability are based on three different flow regimes; transient flow, steady-state flow and pseudo steady-state flow. "Transient flow" refers to a flow regime where pressure changes around the wellbore have not reached any boundaries of the reservoir. It is assumed that during transient flow the pressure response in the reservoir is not affected by the presence of the outer boundary. Therefore, the reservoir acts like an infinitively large reservoir. "Steady-state flow" refers to a flow regime where the pressure at every location in the reservoir remains constant. In other words, the pressure does not change with time. 'Pseudo-steady-state" flow refers to a flow regime where the pressure throughout the reservoir decreases at the same constant rate over time. For all of these flow regimes we will list a correlation found in the literature.

For use in our computer model we will also look at the productivity index. The "productivity index" (J^*) is defined as Guo et al. [7]: "the magnitude of the slope of the IPR curve", which can be written as [7]:

$$J = \frac{q}{p_{Res} - p_{BH}},$$

where

$$\begin{aligned} J^* &= \text{the productivity index} \\ q &= \text{the oil production rate} \\ p_{Res} &= \text{reservoir pressure} \\ p_{BH} &= \text{flowing bottom-hole pressure} \end{aligned}$$

3.1.1. Transient Flow

There are various analytical solutions developed for describing the transient flow behavior for single-phase oil flow. One which is frequently used is stated by [7]:

$$p_{BH} = p_{Res} - \frac{162.6 q B_0 \mu_0}{kh} \times \left(\log t + \log \frac{k}{\phi \mu_0 c_t r_w^2} - 3.23 + 0.87 S \right),$$

where

p_{BH} = flowing bottom-hole pressure, psia
p_{Res} = initial reservoir pressure, psia
q = oil production rate, stb/day
B_0 = oil formation factor
μ_0 = viscosity of oil, cp
k = effective horizontal permeability to oil, md
h = reservoir thickness, ft
t = flow time, hours
ϕ = porosity, fraction
c_t = total compressibility, psi^{-1}
r_w = wellbore radius to the sand face, ft
S = skin factor
log = 10-based logarithm $\log_{10}$

This equation defines the productivity index for bottom-hole pressures above the bubble-point pressure. For radial transient flow around a vertical well the productivity index (J^*) can be written as follows [7]:

$$\begin{aligned} J^* &= \frac{q}{p_{Res} - p_{BH}} \\ &= \frac{kh}{162.6 B_0 \mu_0 \left(\log t + \log \frac{k}{\phi \mu_0 c_t r_w^2} - 3.23 + 0.87 S \right)}. \end{aligned}$$

3.1.2. Steady-State Flow

In the case of single-phase oil flow the following relation can be derived from Darcy's law. It holds for steady-state flow with a circular constant-pressure boundary at distance r_e from the wellbore [7]:

$$q = \frac{kh(p_{Res} - p_{BH})}{141.2 B_o \mu_o \left(\ln \frac{r_e}{r_w} + S \right)},$$

where

q = oil production rate, stb/day
k = effective horizontal permeability to oil, md
h = reservoir thickness, ft
p_{Res} = pressure at the constant-pressure boundary, psia
p_{BH} = flowing bottom-hole pressure, psia
B_0 = oil formation volume factor
μ_0 = viscosity of oil, cp
r_w = wellbore radius to the sand face, ft
ln = 2.718-based logarithm $\log_e$

This equation defines the productivity index for bottom-hole pressures above the bubble-point pressure. For radial steady-state flow around a vertical well the productivity index (J^*) can be written as follows [7]:

$$\begin{aligned} J^* &= \frac{q}{p_e - p_{wf}} \\ &= \frac{kh}{141.2 B_0 \mu_0 \left(\ln \frac{r_e}{r_w} + S \right)}. \end{aligned}$$

3.1.3. Pseudo Steady-State Flow

In the case of single-phase oil flow the following theoretical relation can be derived from Darcy's law. It holds for pseudo-steady-state flow with a circular no-flow boundary at distance r_e from the wellbore [7]:

$$q = \frac{kh(p_{Res} - p_{BH})}{141.2 B_0 \mu_0 \left(\ln \frac{r_e}{r_w} - \frac{3}{4} + S\right)},$$

This equation defines the productivity index for bottom-hole pressures above the bubble-point pressure. For pseudo-steady-state flow around a vertical well the productivity index (J^*) can be written as follows [7]:

$$\begin{aligned} J^* &= \frac{q}{p_{Res} - p_{BH}} \\ &= \frac{kh}{141.2 B_0 \mu_0 \left(\ln \frac{r_e}{r_w} - \frac{3}{4} + S\right)}. \end{aligned}$$

The productivity index for bottom-hole pressures above the bubble-point pressure for transient, steady-state and pseudo steady-state flow can be written in such a way that it is flow rate independent. This means that the IPR curve for a single-phase liquid reservoir is linear (above bubble-point pressure).

3.2. TPR for Single-Phase Liquid Flow

Consider a tubing string of length l and height Δz. The pressure drop created by a fluid flowing from point 1 to point 2 can then be described by the first law of thermodynamics. This is stated in Guo et al. [7] as:

$$\Delta P = P_1 - P_2 = \frac{g}{g_c}\rho \Delta z + \frac{\rho}{2g_c}\Delta u^2 + \frac{2 f_F \rho u^2 L}{g_c D} \tag{3.1}$$

where

P = pressure drop, lb_f/ft^2
P_1 = pressure at point 1, lb_f/ft^2
P_2 = pressure at point 2, lb_f/ft^2
g = gravitational acceleration, 32.17 ft/s^2
g_c = unit conversion factor, 32.17 lb_m ft/lb_f s^2
ρ = fluid density lb_m/ft^3
Δz = elevation increase, ft
u = fluid velocity, ft/s
f_F = Fanning friction factor
L = tubing length, ft
D = tubing inner diameter, ft

Here the fist term of the equation represents the pressure drop due to changes in elevation. The second term of the equation represents the pressure drop due to changes in the kinetic energy, and the third term of the equation represents the pressure drop due to friction.

3.3. PPR for Single-Phase Liquid Flow

For a pipline for transportation of (crude) oil the expression stated in the previous section holds, see Equation (3.1). For horizontal flow the first term becomes zero, because there are no changes in elevation. Further, is the second term normally negligible for flow in transportation oil lines.

3.4. Properties of Oil

In this section some quantities used in the previous correlations are further explained. This for use in the computer model. The correlations given in this section are all given in field units.

3.4.1. Density of Oil

Density of oil is defined as the mass of oil per unit volume. In Field units this is in lb_m/ft^3.

Because of gas content, density of oil is pressure dependent. With use of API gravity we can express the density of oil at standard conditions. The "standard condition " is defined as 14.7 psia and 60 degrees Fahrenheit. Oil at standard conditions is called stock tank oil. The relationship between the density of stock tank oil and API gravity is stated in Guo et al. [7] as:

$$^\circ API = \frac{141.5}{\gamma_o} - 131.5$$

and

$$\gamma_o = \frac{\rho_{o,st}}{\rho_w},$$

where

$^\circ API = API$ gravity of stock tank oil
γ_o = specific gravity of stock tank oil, 1 for freshwater
$\rho_{o,st}$ = density of stock tank oil, lb_m/ft^3
ρ_w = density of freshwater, 62.4 lb_m/ft^3

3.4.2. Formation Volume Factor of Oil

For the definition of the formation volume factor of oil we follow Guo et al. [7]:

> "Formation volume factor of oil is defined as the volume occupied in the reservoir at the prevailing pressure and temperature by volume of oil in stock tank, plus its dissolved gas".

Which can be written as [7],

$$B_o = \frac{V_{res}}{V_{st}},$$

where

B_o = formation volume factor of oil (rb/stb)
V_{res} = oil volume in reservoir condition (rb)
V_{st} = oil volume in stock tank condition (stb)

Because oil dissolves more gas in reservoir conditions than in stock tank condition the formation volume factor oil is always greater then one. There are numerous empirical correlation available to describe the formation volume factor of oil. But oil formation volume factor remains nearly constant at pressures above bubble-point pressure. So for use in the computer model, this quantity is chosen to be constant.

3.5. Fanning Friction Factor

The Fanning friction factor (f_F) is a factor that depends on the Reynolds number of the flow and the relative roughness of the pipe. Reynolds number is defined as [7]

$$N_{Re} = \frac{Du\rho}{\mu}$$

When $N_{Re} < 2000$ we speak of laminar flow. In that case the Fanning friction factor can be described as Guo et al. [7]: $f_F = \frac{16}{N_{Re}}$. When $N_{Re} > 2100$ we speak of turbulent flow. In that case the Fanning friction can be estimated using correlation. The Colebrook-White equation was originally used for generating the friction factor chart used in practice .

Chen [3] gives a explicit correlation that approximates the Colebrook– White equation. Chen's correlation takes the following form :

$$\frac{1}{\sqrt{f_F}} = -4 \times \log\left\{\frac{\epsilon}{3.7065} - \frac{5.0452}{N_{Re}} \log\left[\frac{\epsilon^{1.1098}}{2.8257} + \left(\frac{7.149}{N_{Re}}\right)^{0.8981}\right]\right\}$$

where the relative roughness is defined as $\epsilon = \frac{\delta}{d}$, and δ is the absolute roughness of the pipe wall.

One could also determine the Fanning friction factor based on the Darcy-Weisbach friction factor diagram (see Figure 3.1 [7]). In some literature is the Darcy–Weisbach friction factor (f_D) also referred to as the Moody friction factor (f_M). The relation between the Moody and the Fanning friction factor is $f_F = \frac{f_M}{4}$.

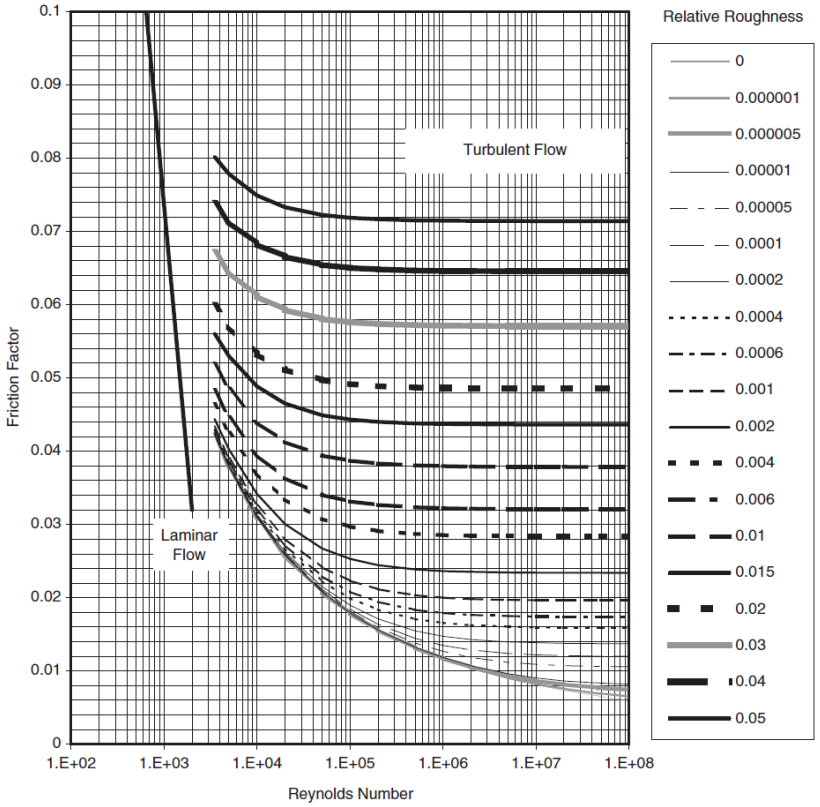


Figure 3.1: The Darcy-Weisbach friction factor diagram. (Guo et al. [7])

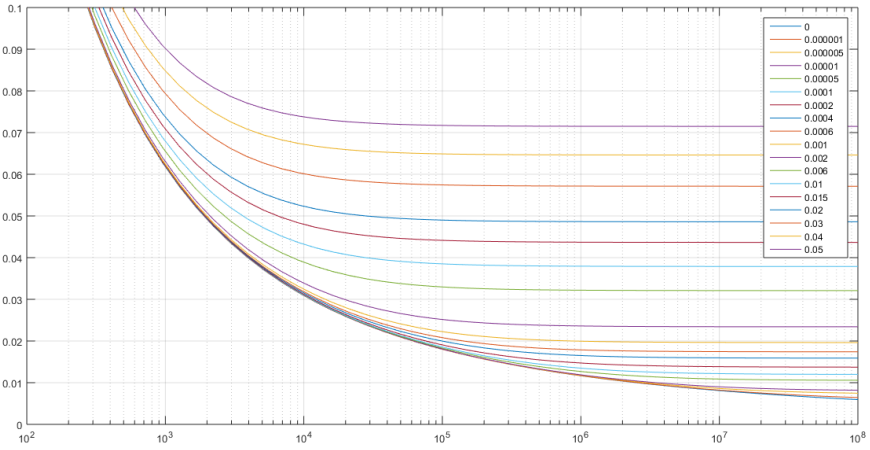
Churchill developed a formula that covers the friction factor for both laminar and turbulent flow, which were originally produced to describe the Darcy-Weisbach friction factor diagram. Because this equations is said to hold for all possible Reynolds numbers and relative roughness, this equation is more suited to be used with computer simulations.

Churchill's Equation takes the following form [3]:

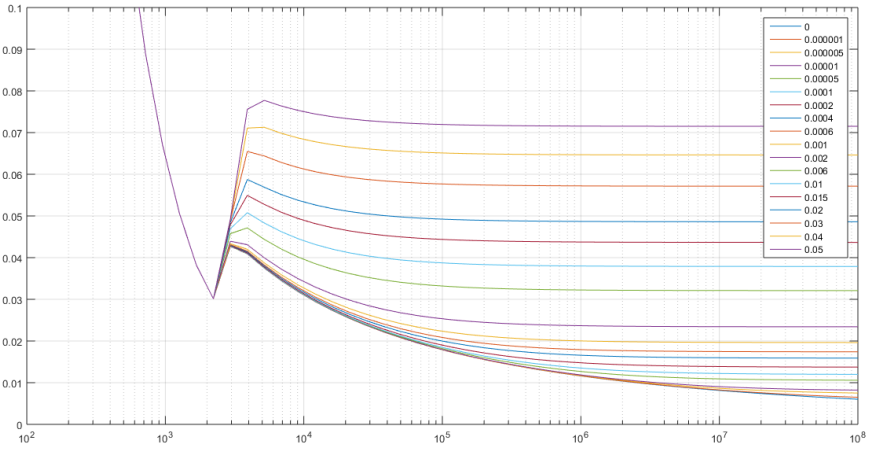
$$f_M = 8\left((8/Re)^{12} + \frac{1}{A+B}^{\frac{3}{2}}\right)^{\frac{1}{12}}$$

where

$$A = \left(2.457 \ln \frac{1}{\frac{7}{Re}^{0.9} + 0.27\epsilon}\right)^{16}$$
$$B = \left(\frac{37530}{Re}\right)^{16}$$



(a) Chen's equation to describe the Darcy-Weisbach friction factor diagram.



(b) Churchill's equation to describe the Darcy-Weisbach friction factor diagram.

Figure 3.2: The different equations to describe the Darcy-Weisbach friction factor diagram.

4

Gas Flow

In this chapter correlations will be derived for use in the computer model for single-phase gas flow. In this chapter the correlations for gas flow will be listed in field units, for a conversion to SI units the conversion factors from Appendix B can be used. For convenience the converted correlations for oil and gas flow in SI units are listed in Appendix A.

4.1. IPR for Single-Phase Gas Reservoirs

Just as with single-phase liquid flow are the mathematical models for reservoir deliverability based on three different flow regimes; transient flow, steady-state flow and pseudo steady-state flow.

4.1.1. Steady-State Flow

The steady-state relationship developed from Darcy's law for oil flow was presented in Section 3.1.2. Following Economides et al. [4] that relationship can be adjusted to hold for gas flow by converting the flow rate from STB/d to MSCF/d and using an average value of the gas formation volume factor between p_{Res} and p_{BH}.

Following Guo et al. [7]: "Gas formation volume factor is defined as the ratio of gas volume at reservoir condition to the gas volume at standard condition." Which can be written as [7]:

$$B_g = \frac{V}{V_{sc}} = \frac{p_{sc}}{p}\frac{T}{T_{sc}}\frac{z}{z_{zc}} = 0.0283\frac{zT}{p}$$

Where V_{sc}, p_{sc}, T_{sc} and z_{sc} represent volume, pressure temperature and gas compressibility factor at standard conditions. As stated before the "standard condition" is defined as 14.7 psia and 60 degrees Fahrenheit.

Therefore [4]:

$$\overline{B}_g = \frac{0.0283\overline{z}T}{(p_{Res}+P_{BH})/2},$$

and from the equation in Section 3.1.2 it follows that [4]:

$$p_{Res}-p_{BH} = \frac{141.2(1000/5.615)q(MSCF/d)(0.0283\overline{z}T\overline{\mu}}{[(p_e+p_{wf})/2]kh}\left(\ln\frac{r_e}{r_w}+S\right),$$

and after rearranging and gathering the terms this results in [4]:

$$p_{Res}^2-p_{BH}^2 = \frac{1424q\overline{\mu}\overline{z}T}{kh}\left(\ln\frac{r_e}{r_w}+S\right).$$

This equation suggests that a gas well production rate is approximately proportional to the pressure squared difference. The properties $\overline{\mu}$ and $\overline{z}$ are average properties between p_{Res} and p_{BH}.

4.1.2. Pseudo Steady-State Flow

The pseudo steady-state flow relationship developed from Darcy's law for oil flow was presented in section 3.1.3. To develop a relationship for pseudo-steady-state flow for gas flow a similar approximation as for steady-state flow can be followed. This results in the following relationship

$$p_{Res}^2 - p_{BH}^2 = \frac{1424 q \overline{\mu z} T}{kh}\left(\ln\frac{r_e}{r_w} - \frac{3}{4} + S\right).$$

4.1.3. Transient Flow

The transient flow relationship for oil flow was presented in section 3.1.1. To develop a relationship for transient flow for gas flow a similar approximation as for steady-state flow can be followed. This results in the following relationship

$$p_{Res}^2 - p_{BH}^2 = \frac{1639 q \overline{\mu z} T}{kh}\left(\log t + \log\frac{k}{\phi \mu c_t r_w^2} - 3.23 + 0.87S\right).$$

4.2. TPR for Single-Phase Gas Flow

To calculate the TPR for a well with gas flow, the compressibility of the fluid must be considered. Due to the compressibility of the fluid, the density and velocity vary along the pipe, and this has to be taken into account. One way to do so is here summarized from Economides et al. [4].

Assuming a average temperate over the segment of the pipe, an average compressibility factor can be calculated. Using these average values of Z and T, the following equation yields [4]

$$p_{out}^2 = e^s p_{in}^2 + \frac{32 f_f}{\pi^2 D^5 g_c \sin\theta}\left(\frac{\overline{ZT} q p_{sc}}{T_{sc}}\right)^2 (e^s - 1)$$

where s is defined as

$$s = \frac{-(2)(28.97)\gamma_g (g/g_c)\sin\theta L}{\overline{ZT} R}$$

In the case of horizontal flow, $\sin\theta$ and s are zero; then the following equation holds

$$p_{in}^2 - p_{out}^2 = \frac{(64)(28.97)\gamma_g f_f \overline{ZT}}{\pi^2 g_c D^5 R}\left(\frac{p_{sc} q}{T_{sc}}\right)^2 L$$

Now, the friction factor must be calculated from the Reynolds number and the pipe roughness. This results in [4]:

For vertical or inclined flow

$$p_{out}^2 = e^s p_{in}^2 + 2.685 \cdot 10^{-3} \frac{f_f (\overline{ZT} q)^2}{\sin\theta D^5}(e^s - 1)$$

where

$$s = \frac{-0.0375 \gamma_g \sin\theta L}{\overline{ZT}}.$$

For horizontal flow we get

$$p_{in}^2 - p_{out}^2 = 1.007 \cdot 10^{-4} \frac{f_f \overline{ZT} q^2 L}{D^5}$$

$$N_{Re} = 20.09 \frac{\gamma_g q}{D\mu}.$$

Where p is in psia, q is in MSCF/d, D is in in., L is in ft, μ is in cp, T is in °R, and all other variables are dimensionless.

4.3. PPR for Single-Phase Gas Flow

For steady-stead flow in a horizontal pipeline for transportation of gas can the mechanical energy equation be written as [7]

$$\frac{dp}{dL} = \frac{f_m \rho u^2}{2g_c D} = \frac{p(MW)_a}{zRT}\frac{fu^2}{2g_c D},$$

Integrating this Equation gives [7]

$$\int dp = \frac{(MW)_a f_M u^2}{2Rg_c D}\int \frac{p}{zT} dL.$$

Assuming an avarage temperature in the pipline, an average compressibility factor can be calculated. Then the integrated Equation can be evaluated over a distance L between upstream pressure, p_1, and downstream pressure, p_2 [7]:

$$p_1^2 - p_2^2 = \frac{25\gamma_g Q^2 \overline{Tz} f_M L}{d^5}$$

where

γ_g = gas gravity (air =1)
Q = gas flow rate, MMscfd (at 14.7 psia, 60 °F)
$\overline{T}$ = average temperature, °R
$\overline{z}$ = average gas deviation factor
L = pipe length, ft
d = pipe internal diameter, in.
f_M = Moody friction factor

4.4. Properties of Gas

In this section some quantities used in correlations in the previous sections are further explained. The correlations given in this section are all given in field units.

4.4.1. Gas Compressibility Factor

"Gas compressibility factor" is defined as [7]:

$$z = \frac{V_{actual}}{V_{idealgas}}.$$

Brill and Beggs [7] give a correlation which is accurate enough for many calculations. Their correlation is expressed as follow :

$$A = 1.39(T_{pr} - 0.92)^{0.5} - 0.36T_{pr} - 0.10,$$
$$B = (0.62 - 0.23T_{pr})p_{pr} + \left(\frac{0.066}{T_{pr} - 0.86} - 0.037\right)p_{pr}^2 + \frac{0.32p_{pr}^2}{10^E},$$
$$C = 0.132 - 0.32\log(T_{pr}),$$
$$D = 10^F,$$
$$E = 9(T_{pr} - 1),$$
$$F = 0.3106 - 0.49T_{pr} + 0.1824T_{pr}^2,$$

and

$$z = A + \frac{1 - A}{e^B} + Cp_{pr}^D,$$

where p_{pr} and T_{pr} are the pseudo-reduced pressure and temperature. These will be discussed next.

4.4.2. Gas Pseudo-reduced and Pseudo-critical Pressure and Temperature

The correlation to determine the z value used the pseudo-reduced pressure and temperature. Pseudo-reduced pressure and temperature are defined as actual pressure and temperature divided by the pseudo-critical pressure and temperature:

$$p_{pr} = \frac{p}{p_{pc}}$$

and

$$T_{pr} = \frac{T}{T_{pc}}$$

The pseudo-critical properties of a gas can be determined on the basis of the critical properties of compounds in the gas using the mixing rule. But there are also various correlations developed to estimate the pseudo-critical pressure and temperature in the case that the gas composition is not known.

One set of simple correlations we will use is [7]

$$p_{pc} = 709.604 - 58.718\gamma_g$$
$$T_{pc} = 170.491 + 307.344\gamma_g$$

where p is in psia and T is in degrees Rankine.

4.4.3. Density of Gas

Density of gas is defines as the mass of gas per unit volume. In Field units this is in lb_m/ft^3. Because gas is compressible, density of gas is pressure depended. But, because gas specific gravity is defined as the density of stock tank gas divided by the density of air, we can also use that correlation to estimate the density of the gas. So,

$$\gamma_g = \frac{\rho_{g,st}}{\rho_{air}}$$

where

$$\gamma_g = \text{specific gravity of stock tank gas}$$
$$\rho_{g,st} = \text{density of stock tank gas}$$
$$\rho_w = \text{density of air.}$$

5

Two-Phase Flow

In this chapter correlations will be derived for use in the computer model for two-phase flow. Two-phase flow occurs for pressure below the bubble-point pressure. In this chapter the correlations for two-phase flow will be listed in field units, for a conversion to SI units the conversion factors from Appendix B can be used.

5.1. IPR for Two-Phase Flow

For two-phase flow the IPR curve will no longer be linear. For pressures below bubble-point pressure the IPR curve will deviate from the 'original' line.

There are a lot of equations available for modeling IPR of two-phase reservoirs. All of them are empirical. According to Guo et al. [7] Vogel's equation is still widely used in the industry. It is written, independent of units, as [7]:

$$q = q_{max}\left[1 - 0.2\left(\frac{p_{TH}}{\overline{p}}\right) - 0.8\left(\frac{p_{TH}}{p_{Res}}\right)^2\right]$$

or

$$p_{TH} = 0.125 p_{Res}\left[\sqrt{81 - 80\left(\frac{q}{q_{max}}\right)} - 1\right],$$

where q_{max} is an empirical constant and its value represents the maximum possible value of reservoir deliverability.

The q_{max} can be theoretically estimated based on reservoir pressure and productivity index above the bubble-point pressure. For example, as in Economides et al. [4], which will be summarized here.

For any flow rate greater than the rate q_b, corresponding to $p_{TH} = p_b$, with p_b the bubbble-point pressure, Vogel's equation gives [4]:

$$\frac{q_0 - q_b}{q_{0(max)} - q_b} = 1 - 0.2\frac{p_{TH}}{p_b} - 0.8\left(\frac{p_{wf}}{p_b}\right)^2$$

or

$$q_0 = q_b + (q_{0(max)} - q_b)\left[1 - 0.2\frac{p_{TH}}{p_b} - 0.8\left(\frac{p_{TH}}{p_b}\right)^2\right]$$

This reciprocal slope is defined as the change in flow rate with respect to the change in p_{TH}, or

$$\frac{dq_0}{dp_{TH}} = (q_{0(max)} - q_b)\left[\frac{-0.2}{p_b} - \frac{1.6 p_{TH}}{p_b^2}\right]$$

Evaluating the reciprocal slope at $p_{TH} = p_b$ gives

$$-\frac{dq_0}{dp_{TH}} = \frac{q_{0(max)} - q_b}{p_b}(0.2 + 1.6)$$

$$-\frac{dq_0}{dp_{TH}} = \frac{1.8(q_{0(max)} - q_b)}{p_b}.$$

The productivity index is defined as the negative of the reciprocal slope, and if J^* is evaluated at any value of $p_{TH} \geq p_b$, the last equation becomes:

$$J^* = \frac{1.8(q_{0(max)} - q_b)}{p_b}$$

or

$$q_{0(max)} - q_b = \frac{J^* p_b}{1.8}$$

This equation also establishes a relationship between J^* and $q_{0(max)}$ for saturated reservoirs, that is for $p_b \geq p_{Res}$ and $q_b = 0$. In this case

$$q_{0(max)} = \frac{J^* p_{Res}}{1.8}$$

5.2. TPR for Two-Phase Flow

The TPR equation for single phase flow is not valid for multiphase oil wells. To analyze TPR of multiphase oil wells, a multiphase flow model is required. Numerous TPR models have been developed for analyzing multiphase flow in vertical pipes. There are two categories of TPR models for multiphase flow. Homogeneous flow models and seperated-flow models. Because homegeneous models treat multiphase as a homogeneous mixture these are usually less accurate. Separated-flow models are more realistic than the homogeneous-flow models.

5.2.1. Homogeneous-Flow Models

Numerous homogeneous-flow models have been developed for analyzing the TPR of multiphase wells. Poettmann and Carpenter's model is explained in Guo et al. [7]. According to them, Poettmann and Carpenter used the following equation to determine the pressure increment [7]:

$$\Delta p = \left(\overline{\rho} + \frac{\overline{k}}{\overline{\rho}}\right) \frac{\Delta h}{144}$$

where

Δp = pressure increment, psi
$\overline{\rho}$ = average mixture density (specific weight), lb/ft^3
Δh = depth increment, ft

and

$$\overline{k} = \frac{f_{2F} q_0^2 M^2}{7.4137 \cdot 10^{10} D^5}$$

where

f_{2F} = Fanning friction factor for two-phase flow
q_0 = oil production rate, stb/day
M = total mass associated with 2 stb of oil
D = tubing inner diameter, ft

The average density is given by

$$\overline{\rho} = \frac{\rho_1 + \rho_2}{2}$$

where

ρ_1 = mixture density at top of tubing segment, lb/ft^3
ρ_2 = mixture density at bottom of segment, lb/ft^3

Then at any given point the mixture density can be calculated based on mass flow and volume flow rates [7]:

$$\rho = \frac{M}{V_m}$$

where

$$M = 350.17(\gamma_0 + WOR\gamma_w) + GOR\rho_{air}\gamma_g$$

and

$$V_m = 5.615(B_0 + WORB_w) + GOR - R_s)\left(\frac{14.7}{p}\right)\left(\frac{T}{520}\right)\left(\frac{z}{1.0}\right)$$

where

γ_0 = oil specific gravity, 1 for freshwater
WPR = producing water-oil ratio, bbl/stb
γ_w = water-specific gravity, 1 for freshwater
GOR = producing gas-oil ratio, scf/stb
ρ_{air} = density of air, lb_m/ft^3
γ_g = gas-specific gravity, 1 for air
V_m = volume of mixture associated with 1 stb of oil, ft^3
B_0 = formation volume factor of oil, rb/stb
B_w = formation volume factor of water, rb/bbt
R_s = solution gas-oil ratio, scf/stb
p = in situ pressure, psia
T = in situ temperature, °R
z = gas compressibility facotr at p and T.

Where producing gas-oil ratio stands for the volume of gas produced per day divided by the volume of oil produced per day at standard conditions. (e.g. in standard cubic feet per stock tank barrel, SCF/STB). And solution gas-oil ratio stand for the volume of gas (at standard conditions) liberated from a single-phase oil volume at reservoir temperature and pressure when being brought to stock tank conditions, divided by the remaining volume of oil (at stock tank conditions). (e.g. again in SCF/STB).

For use in the computer model, solution gas-oil ratio and formation volume factor of oil can be estimated using the following correlations [7]:

$$R_s = \gamma_g \left[\frac{p}{18} \frac{10^{0.0125API}}{10^{0.00091t}} \right]^{1.2048}$$

$$B_0 = 0.9759 + 0.00012 \left[R_s \left(\frac{\gamma_g}{\gamma_0} \right)^{0.5} + 1.25t \right]^{1.2}$$

where t is in situ temperature in ° F.

Poettmann and Carpenter [11] recommended a chart from which the two-phase friction factor can be estimated. The following correlation, which we will use, was developed by Guo and Ghalambor [6] to represent the chart:

$$f_{2F} = 10^{1.444 - 2.5\log(D\rho v)}, \tag{5.1}$$

where $(D\rho v)$ is the numerator of Reynolds number representing inertial force and can be formulated as

$$(D\rho v) = \frac{2.4737 \times 10^{-5} M q_0}{D}.$$

5.2.2. Separated-Flow Models

A number of separated-flow models are available for TPR calculations. Based on comprehensive comparisons of these models, Ansari et al. [1] recommended the Hagedorn-Brown method with modifications for near-vertical flow. More general then the Hagedorn-Brown method is the Beggs and Brill method found in Economides et al. [4]. This method is maybe not as accurate as Hagedorn-Brown for vertical flow, but the Beggs and Brill method can be used for flow at all angles, from horizontal to vertical flow. Below we describe the Beggs and Brill method.

The Beggs and Brill method
Economides et al. [4] describes the Beggs and Brill method, this description will be summarized here. The difference betweeen the Beggs and Brill method and the Hagedorn and Brown method is that that the Beggs and Brill method could be applied to pipelines and flow in any direction.

The Beggs and Brill method is based on the following parameters [4]:

$$
\begin{aligned}
N_{FR} &= \frac{u_m^2}{gD} \\
\lambda_l &= \frac{u_{sl}}{u_m} \\
L_1 &= 316\lambda_l^{0.302} \\
L_2 &= 0.0009252\lambda_l^{-2.4684} \\
L_3 &= 0.10\lambda_l^{-1.4516} \\
L_4 &= 0.5\lambda_l^{-6.738}
\end{aligned}
$$

where

$$
\begin{aligned}
u_{sl} &= \frac{4Q}{\pi D} \\
u_{sg} &= \frac{4}{\pi D} qZ\left(\frac{T}{T_{sc}}\right)\left(\frac{p_{sc}}{p}\right) \\
u_m &= u_{sl} + u_{sg}
\end{aligned}
$$

and T_{sc} and p_{sc} represent the temperature and pressure at standard conditions, so in our case, 60 °F and 14.7 psia.

The horizontal flow regimes used as correlating parameters in the Beggs-Brill method are segregated, transition, intermittent, and distributed. The flow regime transitions are given by the following:

- *Segregated flow* exists if

$$\lambda_l < 0.01 \text{ and } N_{FR} < L_1 \quad \text{or} \quad \lambda_l \geq 0.01 \text{ and } N_{FR} < L_2$$

- *Transition flow* occurs when

$$\lambda_l \geq 0.01 \text{ and } L_2 < N_{FR} \leq L_3$$

- *Intemittent flow* exists when

$$0.01 \leq \lambda_l < 0.4 \text{ and } L_3 < N_{FR} \leq L_1 \quad \text{or} \quad \lambda_l \geq 0.4 \text{ and } L_3 < N_{FR} \leq L_4$$

- *Distributed flow* occurs if

$$\lambda_l < 0.4 \text{ and } N_{JFR} \geq L_1 \quad \text{or} \quad \lambda_l \geq 0.4 \text{ and } N_{FR} > L_4$$

For all these different flow regimes it hold that the liquid holdup is determined by [4]

$$
\begin{aligned}
y_l &= y_{lo}\psi \\
y_{lo} &= \frac{a\lambda_l^b}{N_{FR}^c}
\end{aligned}
$$

with the constraint that $y_{lo} \geq \lambda_l$ and

$$\psi = 1 + C[\sin(1.8\theta) - 0.333\sin^3(1.8\theta)]$$

where $C = (1-\lambda_l)\ln(d\lambda_l^e N_{vl}^f N_{FR}^g)$ where a, b, c, d, e, f, and g depend on the flow regime and are given in Table 5.1 [4]. C must be larger or equal to 0.

Table 5.1: Beggs and Brill holdup constants. (Economides et al. [4])

Flow regime	a	b	c	
Segregated	0.98	0.4846	0.0868	
Intermittent	0.845	0.5351	0.0173	
Distributed	1.065	0.5824	0.0609	
	d	e	f	g
Segregated uphill	0.011	-3.768	3.539	-1.614
Intermittent uphill	2.96	0.305	-0.4473	0.0978
Distributed uphill	No correction, $C = 0, \psi = 1$			
All regimes downhill	4.70	-0.3692	0.1244	-0.5056

In the case of transition flow, bot the segregated and intermittent equation are used to calculate the liquid holdup [4]:

$$\lambda_l = A\lambda_l(\text{segregated}) + B\lambda_l(\text{intermittent})$$

where $A = \frac{L_3 - N_{FR}}{L_3 - L2}$ and $B = 1 - A$.

The frictional pressure gradient is calculated from [4]

$$\left(\frac{dp}{dz}\right)_F = \frac{2 f_{tp} \rho_m u_m^2}{g_c D},$$

where $\rho_m = \rho_l \lambda_l + \rho_g \lambda_g$ and $f_{tp} = f_n \frac{f_{tp}}{f_n}$. The no-slip friction facotr, f_n, is based on smooth pipe ($\epsilon = 0$) and the Reynolds number,

$$N_{Re_m} = \frac{\rho_m u_m D}{\mu_m},$$

where $\mu_m = \mu_l \lambda_l + \mu_g \lambda_g$. The two-phase friction facotr, f_{tp} is then $f_{tp} = f_n e^S$ where

$$S = \frac{\ln(x)}{-0.0523 + 3.182\ln(x) - 0.8725(\ln(x))^2 + 0.01853(\ln(x))^4}$$

and $x = \frac{\lambda_l}{y_l^2}$. Since S is unbounded in the interval $1 < x < 1.2$, for this interval, $S = \ln(2.2x - 1.2)$.

The kinetic energy contribution to the pressure gradient is accounted for with a parameter E_k as follows:

$$\frac{dp}{dz} = \frac{(dp/dz)_{PE} + (dp/dz)_F}{1 - E_k},$$

where

$$E_k = \frac{u_m u_{sg} \rho_m}{g_c p},$$

and $\left(\frac{dp}{dz}\right)_{PE}$ represents the potential energy pressure gradient which is calculate from

$$\left(\frac{dp}{dz}\right)_{PE} = \frac{g}{g_c} \rho \sin\theta.$$

Here this method can again be converted to SI units by the use of the conversion factors from Appendix B.

5.3. PPR for Two-Phase Flow

Various correlation have been developed to describe the flow through a pipeline for transportation of a two-phase fluid. Economides et al. [4] covers some of them. In section 5.2 the Beggs and Brill correlation is summarized. This correlation could also be applied to horizontal flow, because it holds for flow in any direction. For horizontal flow the angle θ is equal to zero which makes ψ equal to 1 and simplifies the correlation somewhat.

6

Modeling

In this chapter all the correlations found in the previous chapters are coupled together to represent an entire system with one or more wells. First we will be looking at what happens in a well below the tubing-head point (subsurface) and then we will be looking at what happens in a well upward of the tubing-head point (surface). Next, these two parts will be coupled together to represent the entire system and to determine the final operating point, which is the point which gives the tubing-head pressure(s) and the flow in the entire system for given reservoir pressure(s) and outlet pressure.

6.1. Subsurface Response Function

The Subsurface Response Function (SSRF) is a relationship to describe everything which happens in a well below the tubing-head point. This is thus a combination of the Inflow Performance Relationship and the Tubing Performance Relationship, and determines the flow in the tubing head according to the tubing-head pressure and the reservoir pressure. The bottom-hole pressure is also determined along the way.

If we look at a single well this will look something like Figure 6.1. In this Figure you can see the underground reservoir containing oil, water and gas and the well which is drilled in the reservoir. The bottom of the well is partially open so the fluids can flow through the well up to the surface. The highest point of the well where fluids can enter the well is called the bottom-hole point.

If we assume that the only subsurface node in our system is the bottom-hole node, the flow out of the node is given by the TPR and the flow into the node is given by the IPR. So, the flow and the bottom-hole pressure following from a certain tubing-head pressure are given by the intersection point of the IPR and the TPR curve, see Figure 6.2. This follows from nodal analysis because: 1. flow into the node equals flow out of the node and, 2. only one pressure can exist at a node.

To find this intersection point we are looking for we have to remember that for a given tubing-head pressure and a given reservoir pressure the relationships we have are given in the following form:

$$Q = IPR(P_{BH}\{, P_{Res}\})$$
$$P_{BH} = TPR(Q\{, P_{TH}\})$$

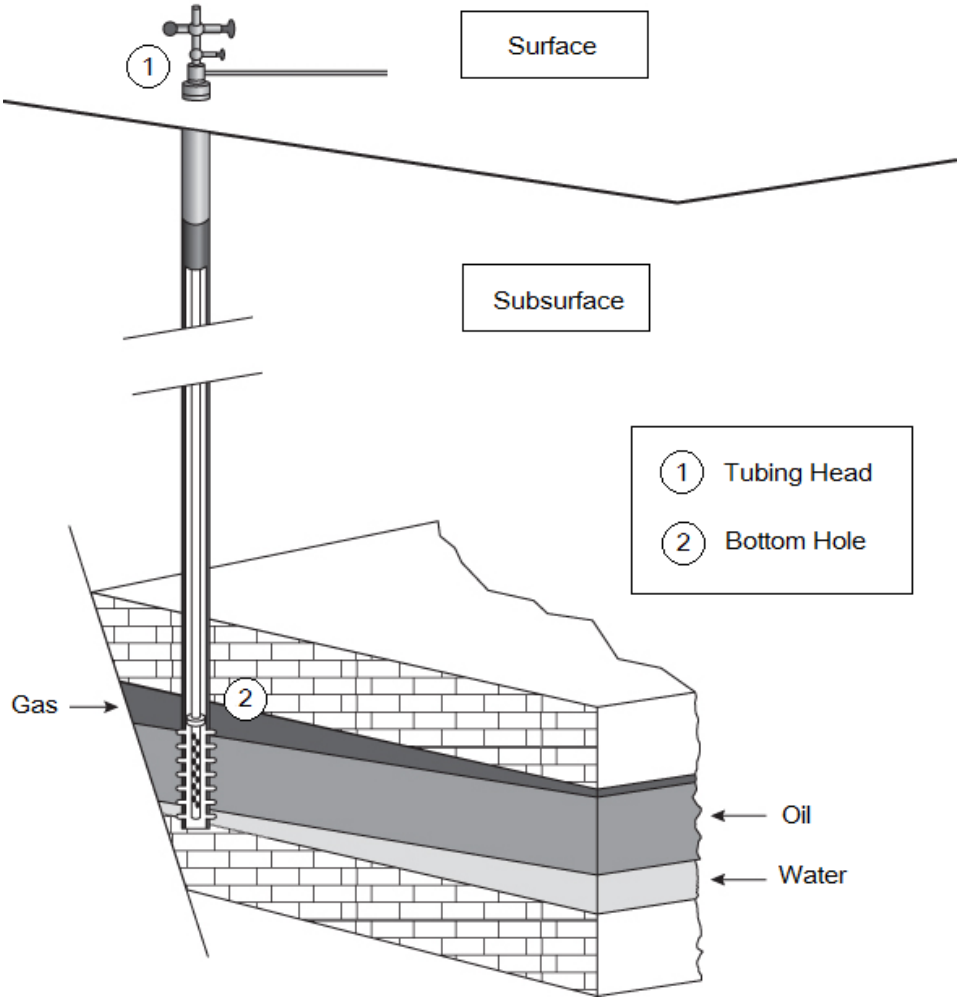


Figure 6.1: A single well.

And now we are looking for the one point (Q, P_{BH}) that satisfies both of these equations. This is the same as saying that we are looking for a point (Q, P_{BH}) which satisfies

$$\begin{aligned} Q - IPR(P_{BH}) &= 0 \\ P_{BH} - TPR(Q) &= 0 \end{aligned} \tag{6.1}$$

which can be done with the *fsolve* algorithm of MATLAB, which is further explained in Appendix C.3. This algorithm was chosen because the algorithm itself is not the main focus of this project, and the *fsolve* algorithm of MATLAB is an algorithm that is fairly robust.

The SSRF will return the flow at the intersection point. If the intersection point between the IPR and the TPR is calculated for a range of tubing-head pressures, the SSRF can also be represented by a curve. This is shown in Figure 6.3.

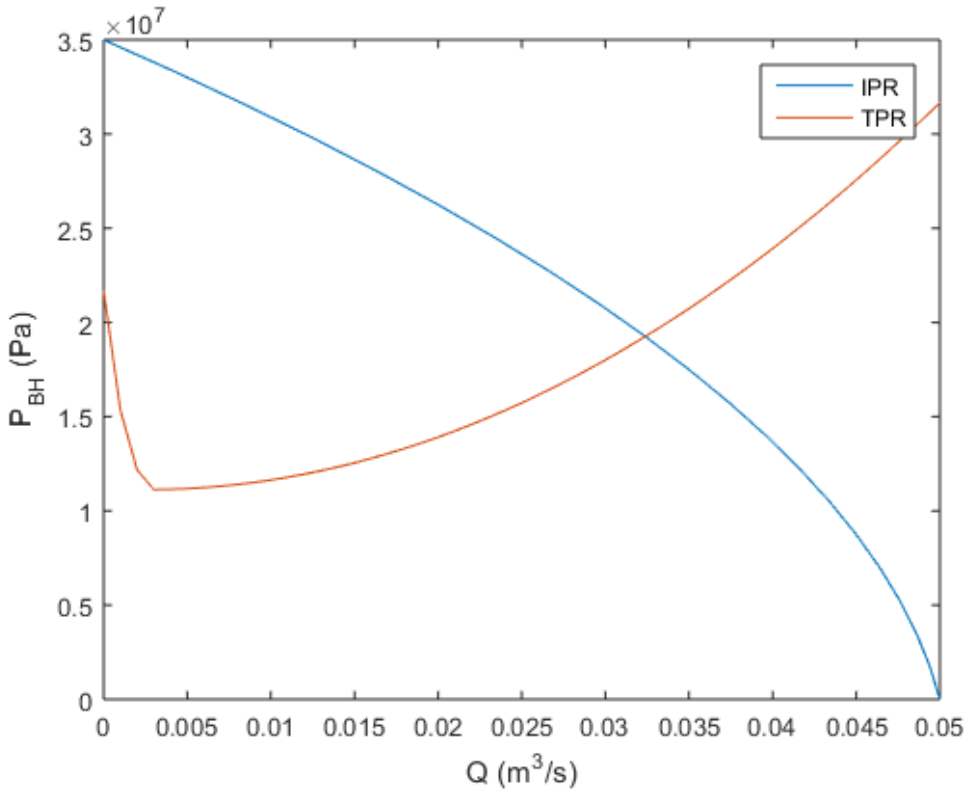


Figure 6.2: An example of an IPR and TPR curve for a certain tubing-head pressure. The SSRF will return the flow rate at the intersection point.

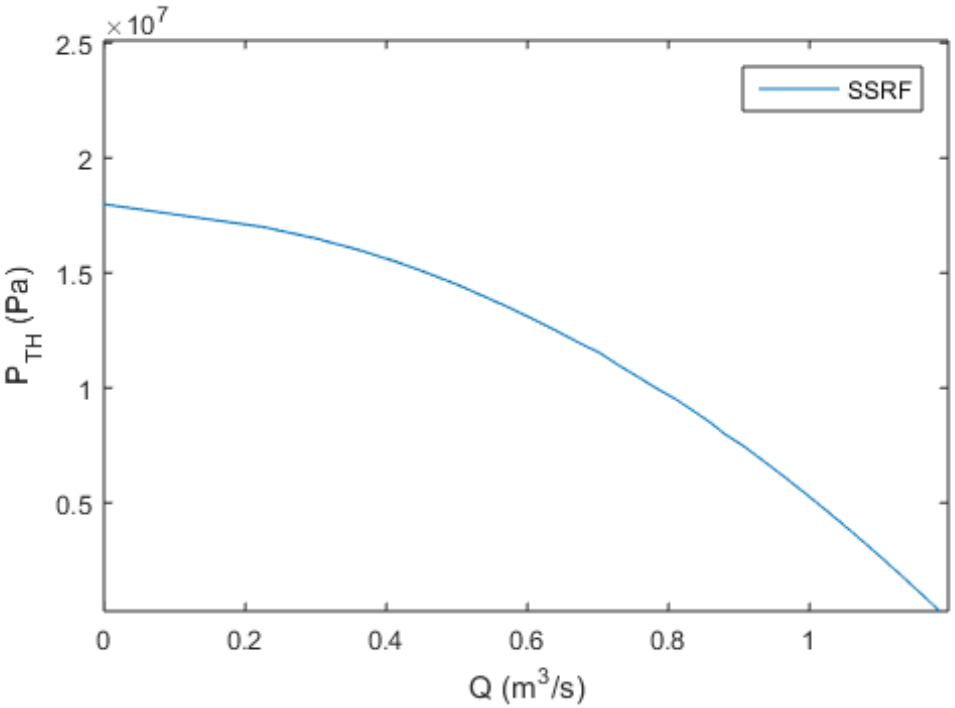


Figure 6.3: The resulting SSRF curve.

6.2. Surface Response Function

The Surface Response Function (SRF) is a function to describe everything which happens in the system upward of the tubing-head point. To determine the Surface Response Function some assumptions had to be made about the construction of the system.

6.2.1. Single Well

If the entire system consist of only one well, which is a situation that in practice almost never happens but could be useful to start modeling, the surface system is easy. It would namely consist of one or multiple pipes connecting the tubing head with a stock tank, as is the case in Figure 6.4. Here the flow in the pipes connecting the tubing head with the stock tank could be described by the Pipe Performance Relationships described in the previous chapters.

6.2.2. Multiple Wells

When there are multiple wells in the system, it is a bit more difficult. In that case, the assumption is made that first the pipes coming from the multiple wells are coupled together. This coupling can be at the outflow point, but it is also possible that the pipes are coupled before the stock tank and the coupling point is connected to the stock tank with another single pipe, see for an example Figure 6.5. Here the flow in the pipelines connecting the tubing head with the stock tank could again be described by the Pipe Performance Relationships described in the previous chapters.

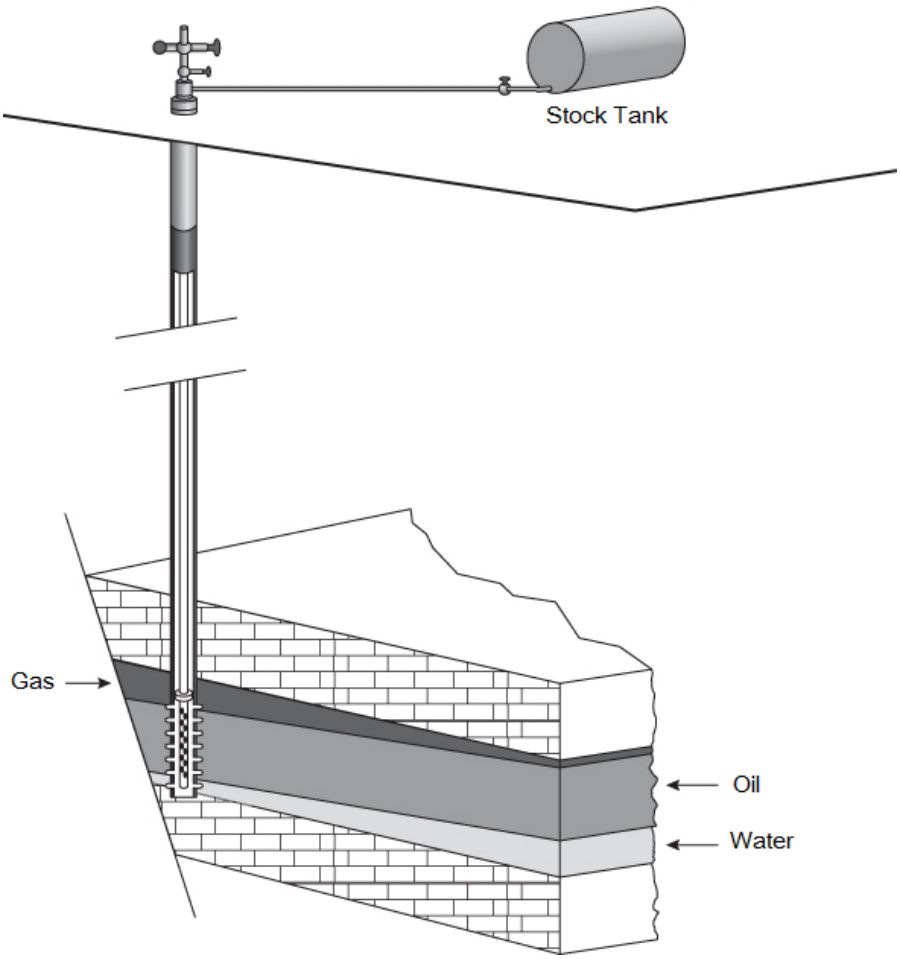


Figure 6.4: Example system for a system with a single well.

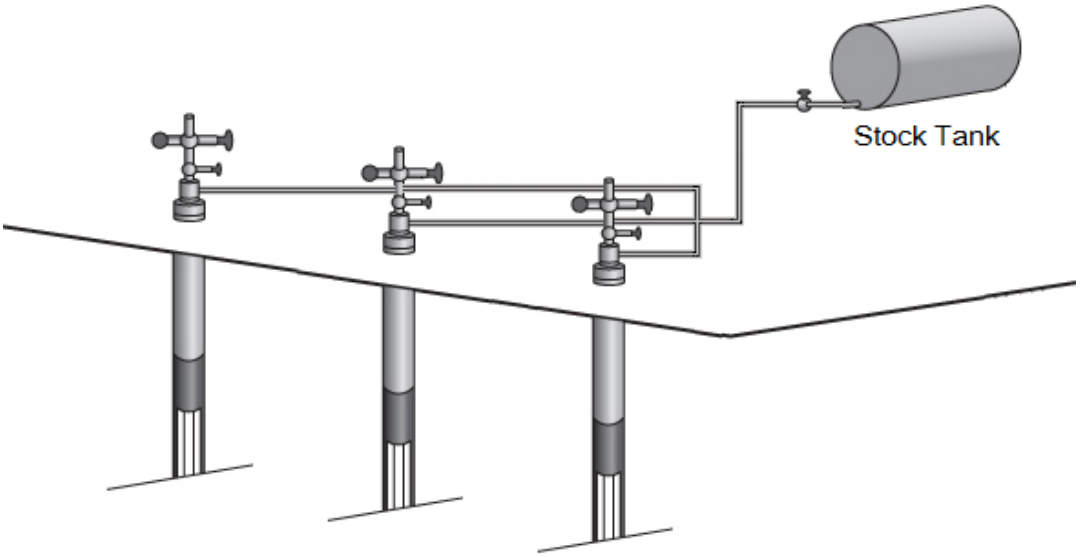


Figure 6.5: Example system for a system with multiple wells.

6.3. Operating Point

Now, the final operating point for both a one-well system and a multiple-well system can be determined. This is the point which gives the tubing-head pressure(s) and the flow in the entire system for given reservoir pressure(s) and outlet pressure.

6.3.1. A One-Well System

The final operating point we are looking for can now be found by looking for the intersection point from the SSRF with the SRF. From nodal analysis point of view this intersection point is the solution when looking at the tubing head as node. This point finally gives the tubing-head pressure and the flow in the entire system.

For a given outlet pressure and reservoir pressure the relationships we have can be described in the following form:

$$\begin{aligned} Q &= SSRF(P_{TH}\{, P_{out}\}) \\ P_{TH} &= SRF(Q\{, P_{Res}\}) \end{aligned}$$

And now we are looking for the one point (Q, P_{TH}) that satisfies both of these equations. This is the same as saying that we are looking for a point (Q, P_{TH}) which satisfies

$$\begin{aligned} Q - SSRF(P_{TH}) &= 0 \\ P_{TH} - SRF(Q) &= 0 \end{aligned} \tag{6.2}$$

which can be done with the *fsolve* algorithm of MATLAB, which is further explained in Appendix C.3.

The calculation of this final operating point is actually a Newton-type iterative process. For each possible tubing-head pressure a flow will be calculated by solving system (6.1). For this flow a new tubing-head pressure can be calculated according to the SRF following from the outlet pressure. Then system (6.1) is solved again for this new tubing-head pressure, which gives a new flow, etc.

Unfortunately, using *fsolve* can give rise to negative presuures P_{TH} during the iterative process causing failure of the solver. To avoid this *fsolve* can be replaced by a constrained minimization algorithm, where positivity of P_{TH} is set as a constraint. For this the MATLAB function *fmincon* can be used. This function is used hereafter for finding the operating point at the tubing head. The algorithm *fmincon* is further explained in Appendix C.1.

If we look at the curve representation of these two relationships then the operating point we are looking for is again the intersection of the two curves, see Figure 6.6

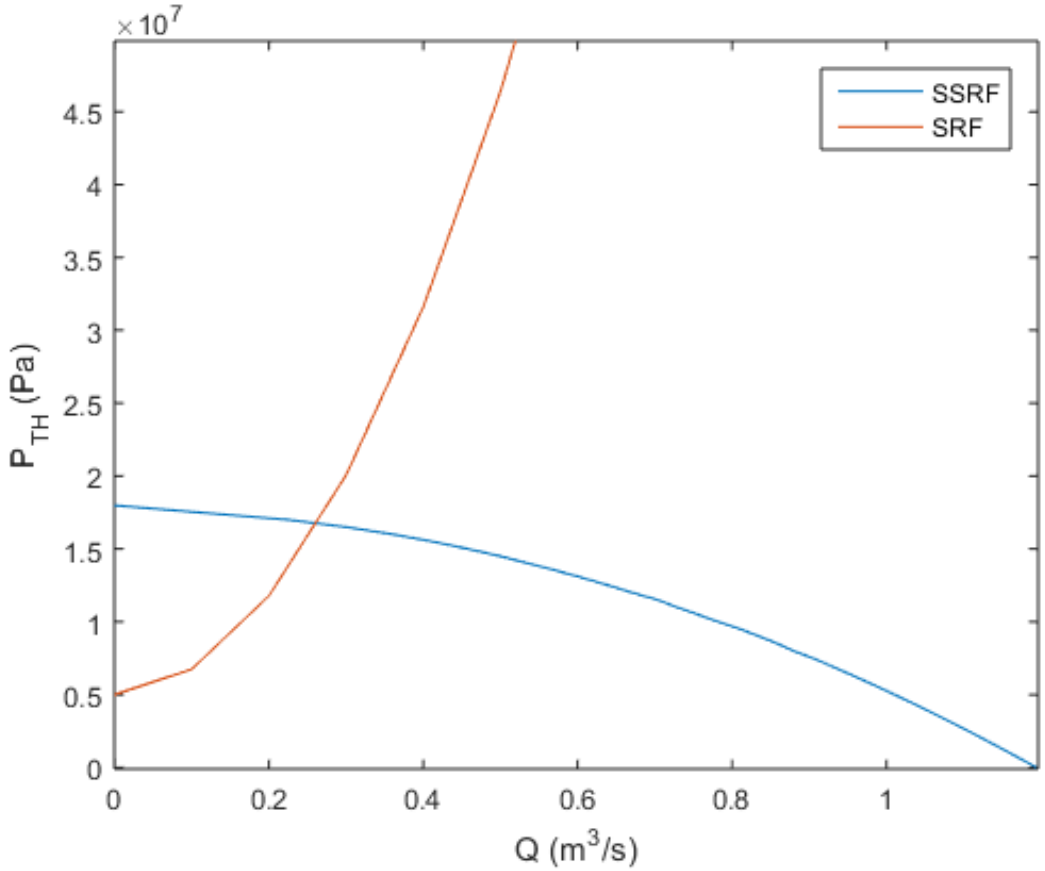


Figure 6.6: An example of an SSRF curve and an SRF curve.

6.3.2. A Multiple-Well System

For a multiple-well system there are multiple wells that have to be considered. So there are also multiple tubing-head pressures, bottom-hole pressures and flow rates which have to be taken into account. If we gather these pressures and flow rates in column vectors, the equations which have to be solved to obtain the operating point are now vector equations. So, system (6.1) becomes

$$\begin{aligned} \mathbf{Q} - \mathbf{IPR}(\mathbf{P_{BH}}) &= \mathbf{0} \\ \mathbf{P_{BH}} - \mathbf{TPR}(\mathbf{Q}) &= \mathbf{0} \end{aligned} \tag{6.3}$$

where the IPR and TPR are calculated per component.

And system (6.2) becomes

$$\begin{aligned} \mathbf{Q} - \mathbf{SSRF}(\mathbf{P_{TH}}) &= \mathbf{0} \\ \mathbf{P_{TH}} - \mathbf{SRF}(\mathbf{Q}) &= \mathbf{0} \end{aligned} \tag{6.4}$$

where the SSRF is again calculated component wise, and the SRF is defined as described in section 6.2.2.

6.4. Object Oriented Programming

For the programming of this entire system in MATLAB we started with function based programming. But to better structure the use of the large amount of functions we changed to object oriented programming. The MATLAB codes can be found in Appendix E.

Object oriented programming is a way of programming that is based on 'objects'. 'Objects' are structures that contain data, in the form of properties; and code in the form of methods. The big difference with function based programming is that the methods of an object can access and modify the properties of its object. In object oriented programming, programs are designed by making them out of objects that interact with one another.

6.5. Example System Calculation

Now we have build up our system we can test our entire model. An operating point can be found for an entire system of multiple wells.

For example a system of three wells with the parameters given in Appendix D.1, where the first well has a tubing length of 2000 meters instead of 200 and the third well has a pipe inner diameter of 0.1 meter instead of 0.2 meter. After solving the entire system this gives the following results:

$$P_{BH} = \begin{bmatrix} 187.6379 \\ 59.9187 \\ 227.9013 \end{bmatrix},$$

$$P_{TH} = \begin{bmatrix} 31.3957 \\ 44.2939 \\ 212,2773 \end{bmatrix},$$

$$Q = 1 \cdot 10^{-3} \begin{bmatrix} 1.4217 \\ 2.5400 \\ 1.0691 \end{bmatrix}$$

and

$$Q_{out} = 5.0307 \cdot 10^{-3}$$

Here the pressures are given in bar and the flow rates in m^3/s. Further are the results for the ith well represented by the number on the ith row of the array. So, the bottom-hole pressure in the first well is 187.6379 bar, the bottom-hole pressure in the second well is 59.9187 bar and so on.

7

Possible Problems in the SSRF and Their Solutions

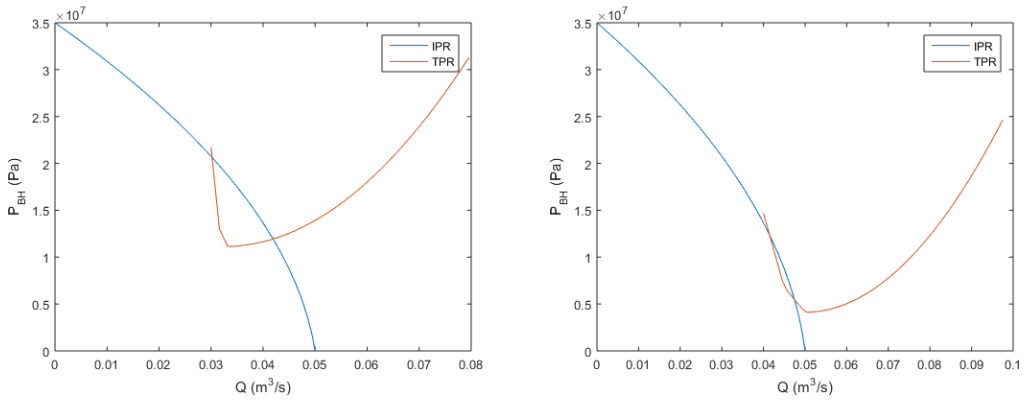
To determine the flow throughout the entire system, the IPR and TPR should have feasible solutions during the iterations in order to be able to determine the intersection of the SSRF with the SRF. Unfortunately, it could happen that that is not always the case. In this section a solution is given for a series of possible problems that could be encountered.

To come to the solutions of the different problems we used a few properties of the various curves. For example, we know that the IPR curve is monotonically decreasing, continuous and differentiable. And the TPR curve is also continuous and differentiable but this curve doesn't have to be monotonic. For some (small) value of Q the TPR may have a minimum. Furthermore, curves are often not specified functions, but rather are given in tabular form, based on measured data. Hence, curves may be incomplete.

7.1. The IPR and TPR have Two Intersections

If the IPR and the TPR have two intersections, which intersections should then be chosen? And does the solver always automatically give the right one ?

If the IPR and the TPR have two intersection there are two possibilities. Namely, that the minimum of the TPR curve lies inside the IPR curve, as can be seen in Figure 7.1a, or that the minimum of the TPR curve lies outside the IPR curve, as can be seen in Figure 7.1b.



(a) Two intersections where the minimum of the TPR curve lies inside the IPR curve.

(b) Two intersections where the minimum of the TPR curve lies outside the IPR curve.

Figure 7.1: The IPR and TPR have two intersections.

From practice it is known that the right intersection point is the desired solution. But, when trying to solve these problems the solver unfortunately doesn't always automatically pick the right intersection point. To determine if the right or the wrong solution is found by the solver some kind of test has to be constructed.

7.1.1. Test

By definition for increasing flow rate the TPR curve is going from above the IPR curve to below the IPR curve in the left intersection point, while in the right intersection point the TPR curve is going from below the IPR curve to above the IPR curve for increasing flow rate. In case of one intersection the intersection is also always found in a point where the TPR is going from below the IPR curve to above the IPR curve for increasing flow rate.

Now, this could be a good starting point for the desired test, but we have to keep in mind that the TPR gives the pressure according to the flow while the IPR gives the flow according to the pressure, so we can't just fill in a certain flow and compare the pressures.

To solve this, we are going to take a small step higher and lower in pressure then the intersection point, determine the flow following from that pressure according to the IPR and then determine the pressure following from the calculated flow according to the TPR. In that way, we can compare two pressure values at the same flow.

Assume $x^* = (Q^*, P^*)$ is the given intersection following from a solver. A small step h higher and lower in pressure will then be $P^* + h$ and $P^* - h$. The flow according to these pressure following from the IPR are then $Q_1 = IPR(P^* + h)$ and $Q_2 = IPR(P^* - h)$. For these flows the pressure following from the TPR can be calculated. These are then given by $P_1 = TPR(Q_1)$ and $P_2 = TPR(Q_2)$. And now we can compare values because we have two pressure values corresponding to Q_1 and two pressure values corresponding to Q_2. One point on the TPR curve and one point on the IPR curve, see Figure 7.2.

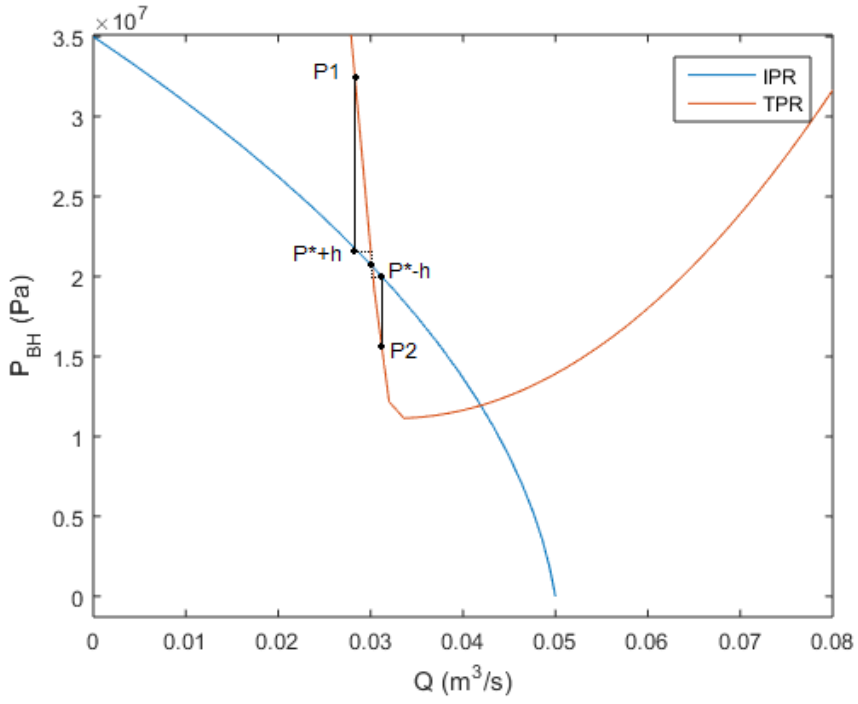


Figure 7.2: The test for determining whether or not the right intersection point is found.

Now, we want to discard our solution if the TPR curve is going from above the IPR curve to below the IPR curve for increasing flow rate. This means that we want to discard our solution if for flow Q_1 (which lies before the intersection point), P_1 (the pressure following from the TPR curve) is higher then $P^* + h$ (the pressure on the IPR curve). And if on the other side for flow Q_2 (which lies after the intersection point), P_2 (the pressure following from the TPR curve) is lower then $P^* - h$ (the pressure on the IPR curve).

So summarized; we want to discard our intersection point $x^* = (Q^*, P^*)$ if

$$P_1 > P^* + h \text{ and } P_2 < P^* - h$$

where $P_1 = TPR(Q_1)$, $P_2 = TPR(Q_2)$, $Q_1 = IPR(P^* + h)$ and $Q_2 = IPR(P^* - h)$.

And because the test both looks at a small step h higher and lower in pressure only the case where the TPR curve goes from above the IPR curve to below the IPR curve will be discarded. So when for example the only intersection point is at the minimum of the TPR curve, this will not be discarded by the test, and this will be accepted as a feasible answers (which it is).

7.1.2. To the Right Intersection Point

We now know when we want to discard an intersection point. But when this happens we want a routine that can guarantee to find the second intersection point, the right intersection point we are looking for.

To do so, we will use the first found intersection because we know a few things after a first intersection point is found. First we know that the second intersection point will be further to the right than the first intersection point, and that the second intersection point will happen before the IPR curve becomes zero. Second, the intersection point we are looking for will be lower then the first found intersection point. This because the IPR curve is monotonically decreasing. So now we have a sense of in what area the second intersection point should lie, see Figure 7.3.

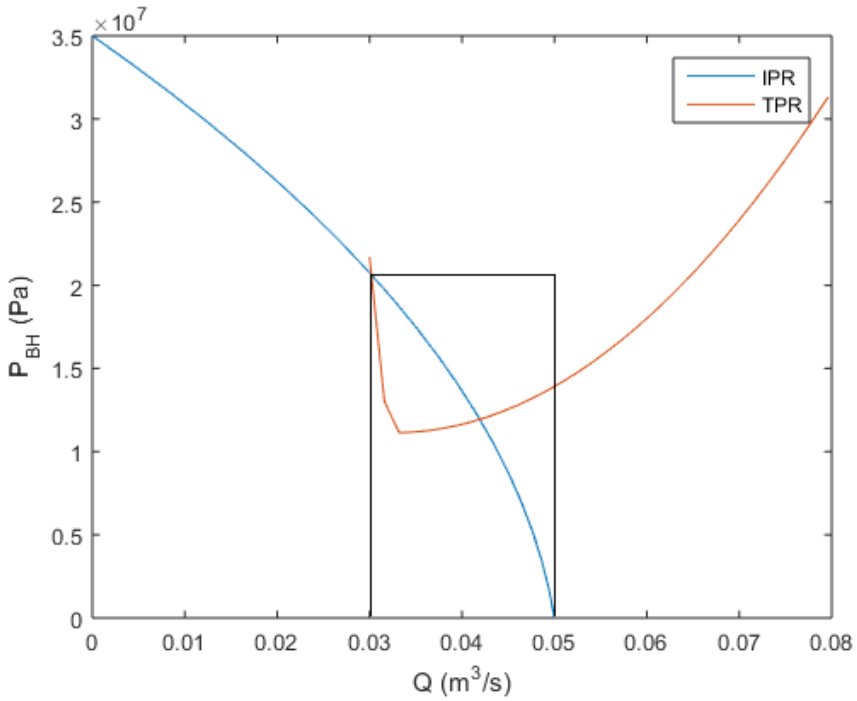


Figure 7.3: The second intersection point has to be inside the boxed area.

Further we know that the right intersection point is always found in a point where the TPR is going from below the IPR curve to above the IPR curve (as used before for the test). If we use some kind of difference function to measure the difference between the IPR curve and the TPR curve this phenomenon should always happen in a point where the difference function is zero and where it changes in sign.

Now take,

$$f(P) = P - TPR(IPR(P)).$$

This function measures for a given value of Q the difference in P. For the given P, $IPR(P)$ is the corresponding Q value according to the IPR curve, and $TPR(IPR(P)$ then gives the corresponding P value according to the TPR curve following from the same Q value. So the value of Q which is used to measure the difference in P is $Q = IPR(P)$, see Figure 7.4.

After finding this difference function f and the interval which has to contain the second intersection point, we can look for a sign change in the function f within the interval with the matlab function *fzero*, which will guaranteed give us the second (and right) intersection point. This algortihm will be explained further in Appendix C.4.

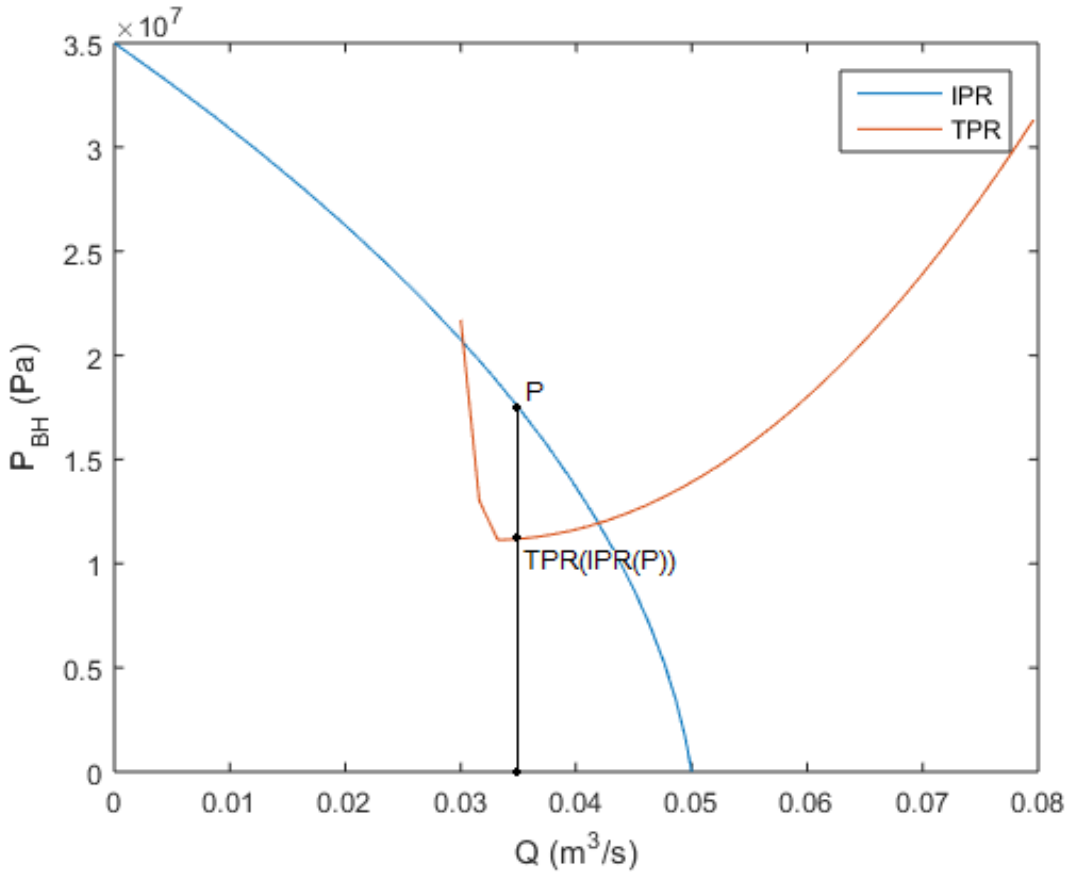


Figure 7.4: This is how the diffference function f works.

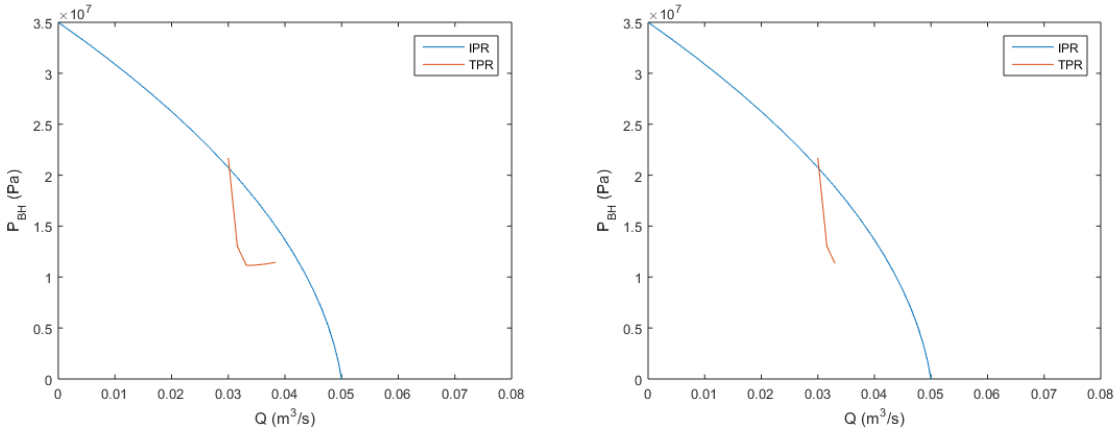
The only possible difficulty is that if we want the MATLAB function *fzero* to work that the begin point of the interval has to have another sign then the end point of the interval. And this could be a problem because the first intersection point is determined with numerical precision, so it could happen that the point returned is above the actual intersection point. Luckily, there is a point of which we know that it lies under the TPR curve so that it will have another sign then the endpoint of the interval. Namely, (Q_2, P_2) where $Q_2 = IPR(P^* - h)$ and $P_2 = TPR(Q_2)$, the second point we used of our test. This because if this point would not lie under the TPR curve, we would not have been looking for a second intersection point.

This method is implemented in MATLAB and tested with the data from Appendix D.3. The intersection of the curves resulting from this data, which are the curves depicted in Figures 7.1a, 7.2, 7.3 and 7.4 is given by the implementation in MATLAB as $Q = 0.0419$ m^3/s and $P_{BH} = 119.3770$ bar. Note that this is indeed the second intersection point.

7.1.3. If there is No Right Intersection Point

But what if there is no right intersection point? This would mean that the IPR curve and the TPR curve actually only have one intersection point but that this intersection point didn't pass the test. So, the TPR curve is going form above the IPR curve to below the IPR curve, but doesn't reach the IPR curve anymore, see Figure 7.5 .

This can happen when the TPR curve seems to short to have a second intersection point, which can happen because of lack in data. Unfortunately, the TPR curve can stop at two different points, when the TPR curve reached its minimum and is again ascending and before the TPR curve has reached its minimum and is still descending, see Figure 7.5. In the first case some kind of estimation can be made of where the second intersection point should be by extrapolation. But, in the second case there is no way of telling where the TPR curve begins ascending again, so there is no way of telling where the second intersection point approximately should be.



(a) No second intersection due to a to short TPR curve, but the TPR curve has reached its minimum.

(b) No second intersections due to a to short TPR curve, and the TPR curve hasn't reached its minimum yet.

Figure 7.5: The TPR curve is to short to have a second intersection point.

To determine which one of these possibilities we are dealing with, first the TPR curve is extrapolated using linear extrapolation. Second, the bottom-hole pressure according to this extrapolated TPR at the flow rate following from the IPR at 1 bar is determined. If this bottom-hole pressure is more then 1 bar the extrapolated TPR would give us a feasible second intersection point. If this bottom-hole pressure is less then 1 bar no feasible second intersection point can be found. Here we look at 1 bar because in practice 1 bar is the lowest use bottom-hole pressure. So summarized; no second feasible intersection point can be found if $P_{1bar} < 1 \cdot 10^5$ for $P_{1bar} = TPR(IPR(1 \cdot 10^5))$.

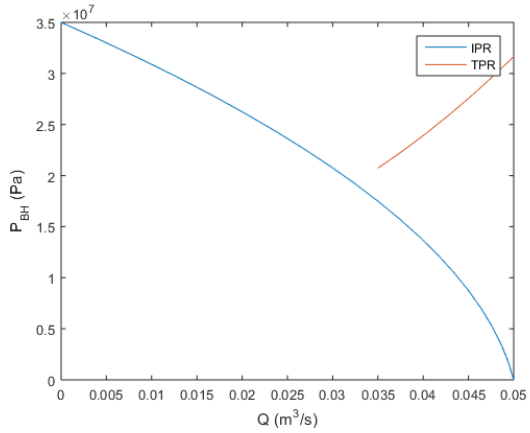
If no feasible second intersection point can be found during the iterations some kind of logical temporary solutions has to be given in order to continue the iterations. That is why in this case the solution is (temporarily) set at a pressure of 1 bar with the corresponding flow rate following from the IPR.

But when no feasible second intersection point can be found in the last iteration (when the solution process has converged) the entire well where this occurs will be closed and the entire system is solved again for the remaining wells. This because we know for sure that the first intersection point was not the desired solution for this well and the second intersection point can't be found.

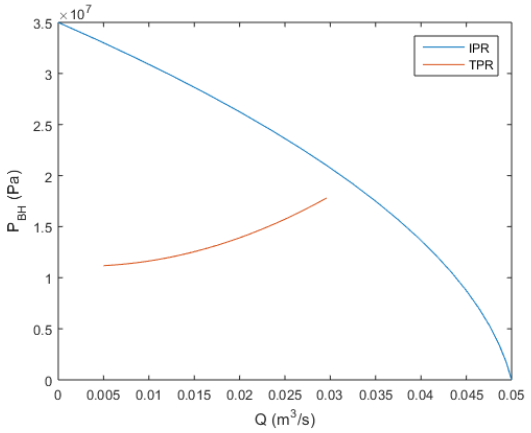
7.2. The IPR and TPR have No Intersections

If the IPR curve and the TPR curve don't have an intersection at all there are also two possibilities. Namely, that the TPR curve seems to short which can happen because of lack in data, as can be seen in Figure 7.6, or when it seems that the entire TPR curve lies above the IPR curve, as can be seen in Figure 7.8a.

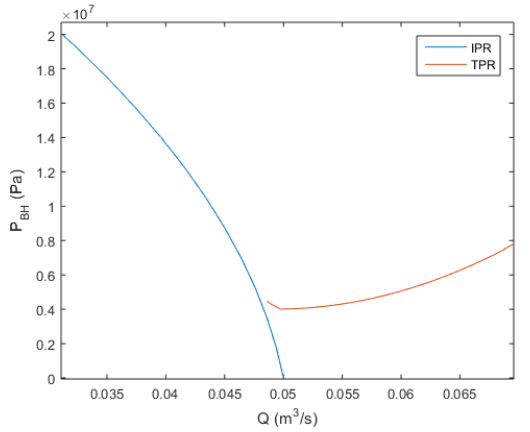
7.2.1. The TPR Curve is To Short



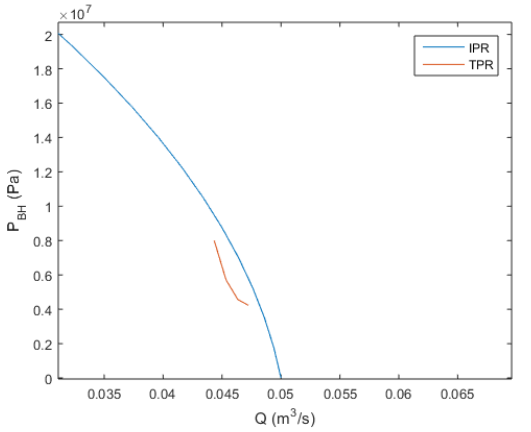
(a) No intersections due to a to short TPR curve which lies entirely above the IPR curve.



(b) No intersections due to a to short TPR curve which lies entirely under the IPR curve.



(c) No intersections due to a to short TPR curve which lies entirely above the IPR curve.



(d) No intersections due to a to short TPR curve which lies entirely under the IPR curve.

Figure 7.6: The TPR curve is to short to have an intersection point.

When it seems that the TPR curve is to short (which can happen because of lack in data) it seems intuitive to extrapolate the data to be able to determine an intersection point. To do so, linear extrapolation is used. This because linear extrapolation can't create new extrema.

But there is one thing that we have to take into account, the case in which an extrapolated TPR curve would only intersect with the IPR curve at a negative pressure or negative flow rate. When the extrapolated TPR curve would make an intersection with the IPR curve at a negative pressure we have a situation like Figure 7.5b (with or without the first intersection point) and this will be dealt with as described in the previous section.

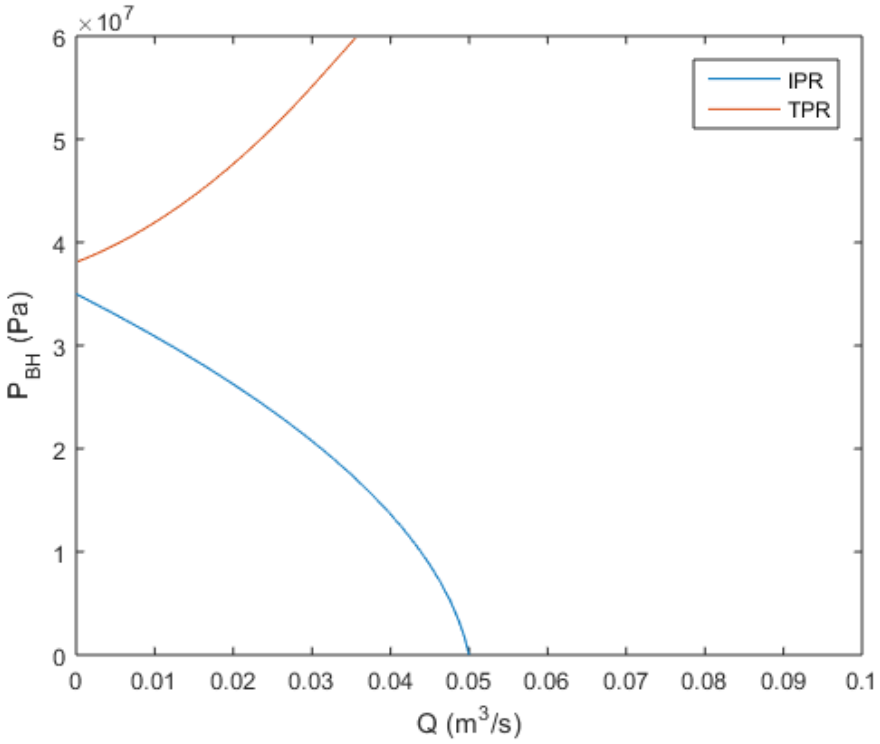
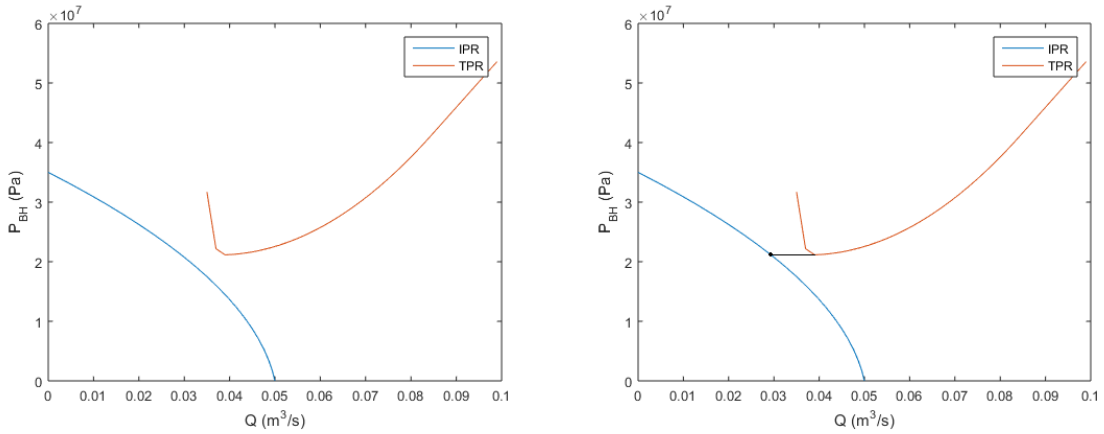


Figure 7.7: The TPR curve is to short and would intersect the IPR curve at a negative flow rate.

If the extrapolated TPR curve would make an intersection with the IPR curve at a negative flow rate, it would be some kind of situation like Figure 7.7. In this case a temporary solution is needed. To make sure the iteration process continues there is chosen to set the negative flow rate (or any flow rate lower then the minimum flow value) to a predefined minimum flow value. But when this happens in the last iteration there is obviously not actually an intersection, so any solution process where this situation occurs in the last iteration doesn't give a realistic solution of the entire system, and the well where this occurs should be closed.

7.2.2. The TPR Curve lies Above the IPR Curve



(a) No intersections because the entire TPR curve lies above the IPR curve.

(b) Possible solution when there is no intersection because the entire TPR curve lies above the IPR curve.

Figure 7.8: No intersections because the entire TPR curve lies above the IPR curve and a possible solution to this problem.

When the entire TPR curve lies above the IPR curve the solution isn't that straight forward. In practice one often makes use of a horizontal extension, starting at the minimum of the TPR, to find a temporary solution, see Figure 7.8b. But in some situations this doesn't seems to be the most logical solution, see Figure 7.9. In this Figure we see a situation where the IPR and TPR curve could have an intersection if the TPR curve shifts just a little bit (due to a changing tubing-head pressure), but this intersection is more likely to at the descending part of the TPR curve instead of at the minimum of the TPR curve. So, when we would use the horizontal extension as the solution in the iteration, there could be a jump in the solutions.

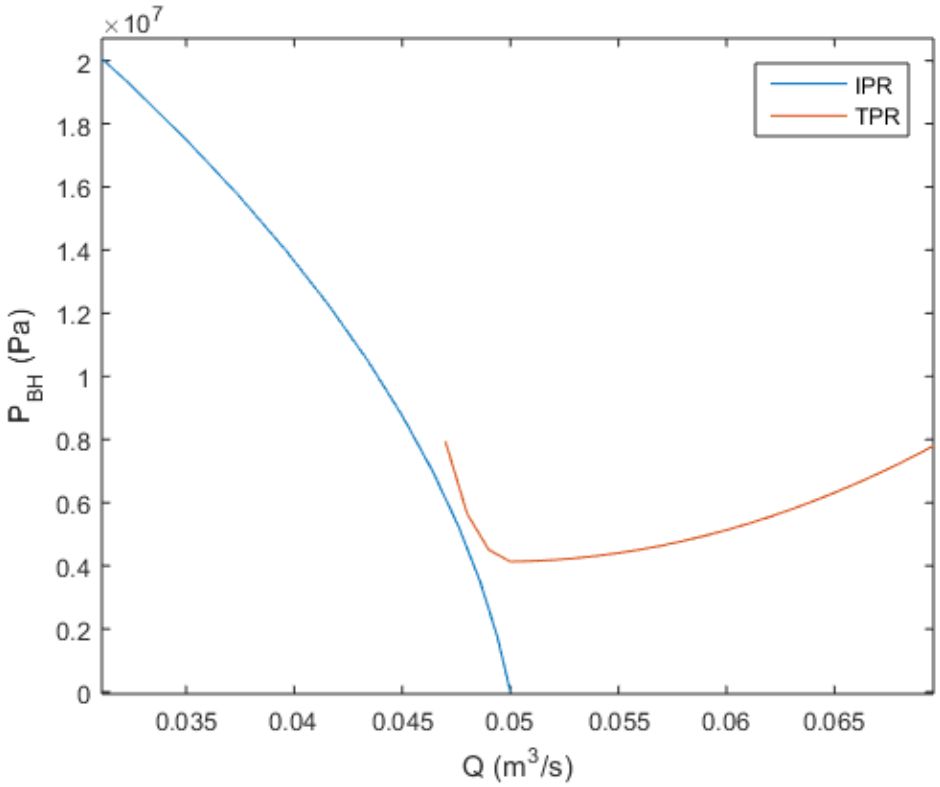


Figure 7.9: In some situations the horizontal extension doens't seems to be the most logical solution.

To solve this we looked at the place where the IPR and the TPR curve are the closest to each other, because there the two curves are the closest to an intersection. When this situation occurs during an iteration, one thus could provide a temporary solution at that point.

But one thing that should be kept in mind is that there is not actually an intersection. So, any solution process where this situation occurs in the last iteration doesn't give a realistic solution of the entire system, and the well should be closed.

To determine the point where the IPR and the TPR curve are closest to each other we could simply minimize the norm of the residual vector, where the Residual vector for each $x = (Q, P)$ is given by

$$\text{Res} = \begin{bmatrix} |Q - IPR(P)| \\ |P - TPR(Q)| \end{bmatrix},$$

and represent the distance from point x to both the IPR and the TPR curve, as can be seen in Figure 7.10. The distance from point I on the IPR curve to point T on the TPR curve is then given by

$$||\text{Res}||_2 = \sqrt{(Res_1)^2 + (Res_2)^2} = \sqrt{(|Q - IPR(P)|)^2 + (|P - TPR(Q)|)^2}.$$

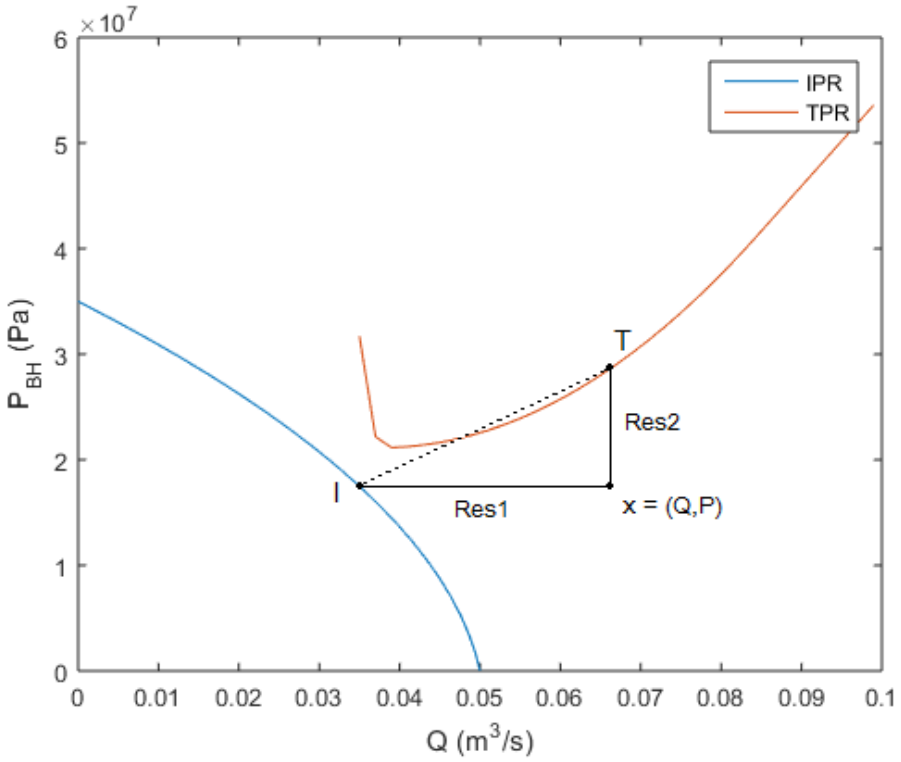


Figure 7.10: The Residual vector.

So, when minimizing this norm over all possible places for x gives us the minimum distance from a point I on the IPR curve to a point T on the TPR curve. The only decision left to make now is whether we want to choose point I as the temporary solution or point T.

Because the TPR curve changes during the iterations (due to a changing tubing-head pressure) and the IPR curve stays the same during the iterations (due to a fixed reservoir pressure) the point I on the IPR curve with the minimum distance to the TPR curve is chosen as the solution for this situation. The point I is given by $I = (IPR(P), P)$. This solution is shown in Figure 7.11. Here the curves in the Figure are made from the tabular data from Appendix D.2 and the temporary solution is given by $Q = 0.0375$ m^3/s and $P_{TH} = 157.0032$ bar.

Now, because if the two curves do make an intersection the distance between the two curves is also minimized (the minimum is zeros in this case) it is possible to replace the non linear solver by an optimization routine. So, instead of using the *fsolve* algorithm to determine the intersection between the IPR and the TPR, as described in section 6.1, this is now done by using the *fminunc* algorithm of MATLAB, which is further explained in Appendix C.3.

When there is not actually an intersection in the last iteration this doesn't give a realistic solution of the entire system. Therefore, when this happens the well where this happens is closed and the entire system is solved again for the remaining wells. If there are multiple wells where this happens the well with the largest minimum distance is closed first.

The minimum distance is calculated with a certain numerical error. In our case with a tolerance set at $\epsilon = 1 \cdot 10^{-6}$. This tolerance is a lower bound on the change in the value of the objective function (the distance between the IPR and the TPR curve) during a step. If $||Res(x_i)|_2 - |Res(x_{i+1})|_2| < \epsilon$, the iterations ends. This means that if there is an intersection and the norm of the Residual vector should be zero at this intersection point, the iteration ends with a norm smaller then epsilon. So, we have chosen to handle the case where the norm is more then epsilon as if there was no intersection point.

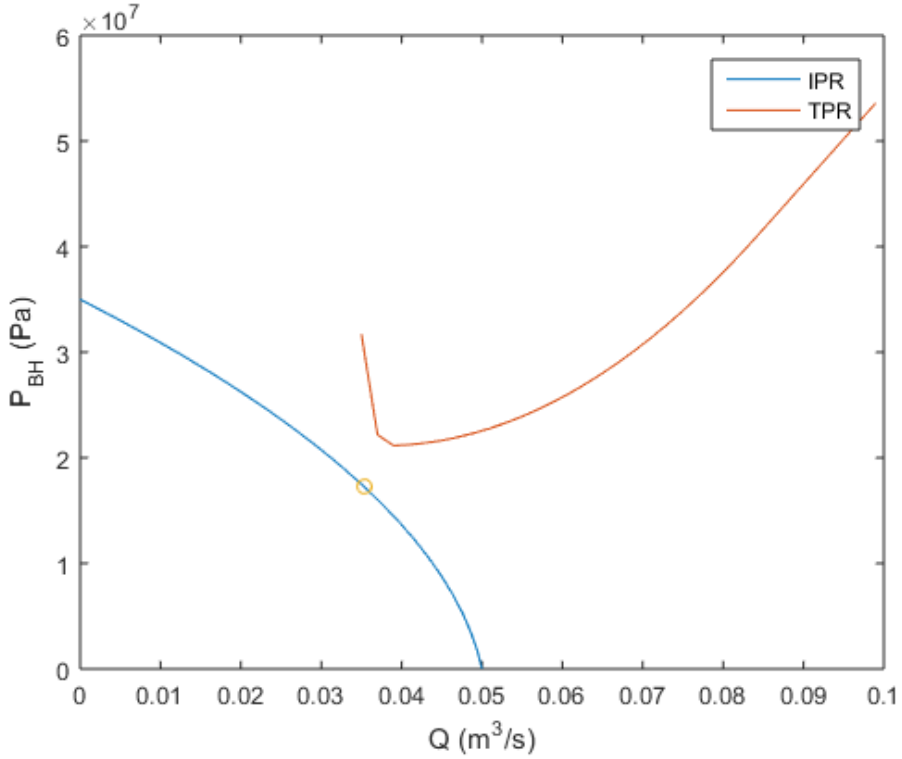


Figure 7.11: The shortest distance solution.

8

Optimization

The flow rate from almost all flowing wells is controlled with a tubing-head choke which is a device that places a restriction in the flow line. A variety of factors may make it desirable to restrict the production rate from a flowing well, including satisfying production rate limits set by regulatory authorities and meeting limitations of rate or pressure imposed by surface equipment.

In practice the total outflow typically is maximized given certain constraints. To satisfy the constraints the chokes are typically used to achieve maximum flow by closing them just enough. So instead of a solution of a system of nonlinear equations we are now considering a optimization (maximization) problem with constraints and we utilize the well chokes as controlling devices for the optimization.

8.1. The Tubing-Head Choke

At the tubing head a choke will cause a pressure drop by placing a restriction in the flow line. Without the choke the flow rate follows from the intersection of the SSRF curve with the SRF curve, and with the choke inserted the flow rate will be lower (see Figure 8.1).

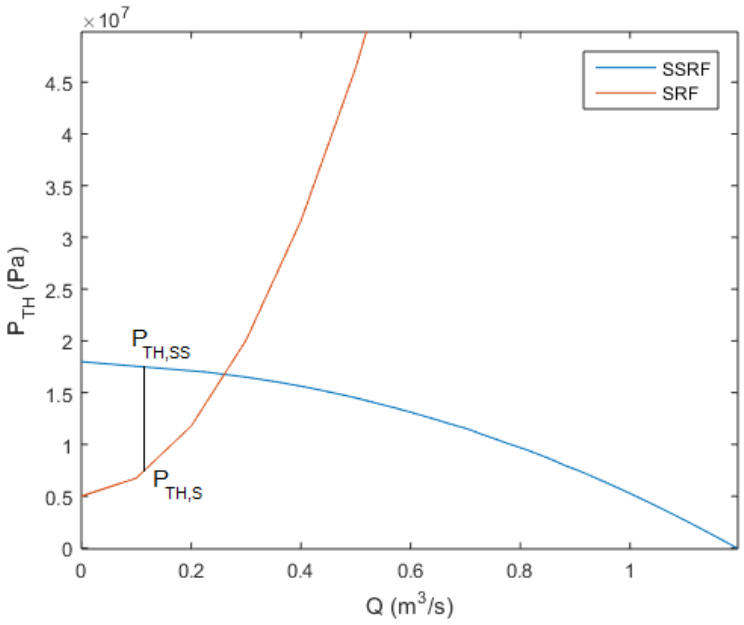


Figure 8.1: The pressure drop caused by a choke.

We introduce a Choke Performance Relationship (CPR). This CPR represents the flow through a choke for a certain setting of this choke. To demonstrate the working of the choke in this optimization formulation we choose here a simple choke model for liquid flow,

$$Q = \alpha \frac{A}{\rho} \sqrt{2\rho\Delta p}$$

where

$$\begin{aligned} Q &= \text{flow rate, } \mathrm{m}^3/\mathrm{s} \\ \alpha &= \text{choke setting, } 0 \le \alpha \le 1 \\ A &= \text{area of the connected pipe, } \mathrm{m}^2 \\ \rho &= \text{fluid density, } \mathrm{kg}/\mathrm{m}^3 \\ \Delta p &= \text{pressure drop, Pa} \end{aligned}$$

where the pressure drop is given by $P_{TH,SS} - P_{TH,S}$, where $P_{TH,SS}$ is the (tubing-head) pressure on the sub-surface side of the choke, and $P_{TH,S}$ the (tubing-head) pressure on the surface side of the choke. the choke setting $\alpha = 1$ means that the choke is fully open.

Rearranging the terms we get the following form

$$P_{TH,SS} = \frac{\rho Q^2}{2\alpha^2 A^2} + P_{TH,S}.$$

Which we write as $P_{TH,SS} = CPR(Q, P_{TH,S}, \alpha)$.

And we know that

$$P_{TH,S} = SRF(Q),$$

so $P_{TH,SS}$ is given by $P_{TH,SS} = CPR(Q, SRF(Q), \alpha)$.

So, with the choke inserted the tubing-head pressure $P_{TH,SS}$ and the flow rate Q are found from the equations

$$\begin{aligned} P_{TH,SS} &= CPR(Q, SRF(Q), \alpha) \\ Q &= SSRF(P_{TH,SS}) \end{aligned}$$

8.2. Optimization Formulation

Say that there is imposed a maximal rate Q_{max}. Then Q_{out}, the flow rate coming out of the entire system should be lower or equal to Q_{max}: $Q_{out} \le Q_{max}$. And one would want to have the resulting flow rate as large as possible. So, then we find our operating point by maximizing the possible outflow rate

$$\begin{aligned} \max_{\alpha} \quad & Q_{out} \\ \text{subject to} \quad & Q_{out} \le Q_{max} \\ & P_{TH,SS} = CPR(Q, SRF(Q), \alpha) \\ & Q = SSRF(P_{TH,SS}) \\ & 0 \le \alpha \le 1 \end{aligned}$$

For multiple wells the tubing-head equations are of course per well, and α is a vector (α_i is the choke setting for well i).

The implementation in MATLAB of this formulation of the problem can be found in Appendix E.5.2.

8.3. Example Case

The same system as in section 6.5 is solved again to show the working of a choke. First this system was solved to have a flow out of the system equal to $Q_{out} = 5.0307 \cdot 10^{-3}$ m^3/s. But what if the highest flow allowed out of the system is a flow of just $4.5 \cdot 10^{-3}$ m^3/s. While setting Q_{max} at precisely that, so $Q_{max} = 4.5 \cdot 10^{-3}$, this system is solved again. (This system is thus again with the parameters from Appendix D.1, while L_{TPR} of well 1 is set to 2000 and d of well 3 is set to 0.1.) Then the results are the following:

$$P_{BH} = \begin{bmatrix} 187.6446 \\ 116.2620 \\ 232.4286 \end{bmatrix},$$

$$P_{TH} = \begin{bmatrix} 31.4024 \\ 100.6374 \\ 216.8045 \end{bmatrix},$$

$$\Delta P_{TH} = \begin{bmatrix} 0.0073 \\ 62.0340 \\ 11.8422 \end{bmatrix},$$

$$Q = 1 \cdot 10^{-3} \begin{bmatrix} 1.4216 \\ 2.0466 \\ 1.0295 \end{bmatrix},$$

$$\alpha = \begin{bmatrix} 0.0339 \\ 0.0005 \\ 0.0024 \end{bmatrix},$$

and

$$Q_{out} = 4.4977 \cdot 10^{-3}$$

Here the pressures are given in bar and the flow rates in m^3/s. Further are the results for the ith well represented by the number on the ith row of the array. So the bottom-hole pressure in the first well is 187.6446 bar, the bottom-hole pressure in the second well is 116.2620 bar and so on.

Note that the Q_{out} now satisfies the constraint of being smaller than Q_{max} of $4.5 \cdot 10^{-3}$ and that that has happened by closing the choke more for the second and the third well (the lower the value of α the further the choke is closed) which led to a pressure drop at the tubing head.

9 Conclusion

While modeling the coupling between the reservoir and the surface facilities network it was the intention to come up with a robust coupling that could cope with the problems that sometimes occur in other coupling methods. Further, it should be an efficient simulation model of the combined system.

The coupling method derived in this report definitely meets the first requirement. For all possible problems in the Subsurface Response Function we could think of a solution was derived and implemented. This led to a coupling method that should converge to a solution in every case.

Since the implementation was done in MATLAB using default nonlinear solvers and optimizers it is difficult to judge the efficiency of the total algorithm in practice. For that a dedicated implementation would be required to be used in combination with real simulators and compared with existing solutions.

The whole implementation here was done in MATLAB with routines provided by MATLAB in an object oriented programming style. Because MATLAB normally isn't used for object oriented programming the coupling is also tested in a functional programming way, but that didn't really decrease the solving time. One other thing that possibly could be done in the future is to replace the MATLAB routines; maybe that could decrease the solving time within the MATLAB framework.

A

Correlations in SI Units

For convenience, the correlations for oil and gas flow are converted to SI units and will be listed in this chapter. For this conversions the conversion factors from Appendix B are used.

A.1. Oil

In chapter 3 the different correlations for oil flow are listed in field units. In this section these correlations will be listed again but now in SI units.

A.1.1. IPR

Transient Flow

The producitivity index (J^*) for flowing bottom-hole pressures above the buble-point pressure for radial transient flow around a vertical well is defined as follows:

$$J^* = \frac{5.4567\ kh}{B_0\mu_0\left(\log t + \log\frac{k}{\phi\mu_0 c_t r_w^2} + 0.34895 + 0.87S\right)}$$

Steady-State Flow

The productivity index (J^*) for flowing bottom-hole pressures above the bubble-point pressure for radial steady-state flow around a vertical well is defined as follows:

$$J^* = \frac{6.2837\ kh}{B_0\mu_0\left(\ln\frac{r_e}{r_w} + S\right)}$$

Pseudo Steady-State Flow

The productivity index (J^*) for flowing bottom-hole pressures above the bubble-point pressure for pseudo-steady state flow around a vertical well is defined as follows:

$$J^* = \frac{6.2837\ kh}{B_0\mu_0\left(\ln\frac{r_e}{r_w} - \frac{3}{4} + S\right)}$$

where

J^* = productivity index, m^3/s Pa
q = oil production rate, m^3/s
B_o = oil formation factor
μ_0 = viscosity of oil, Pa s
k = effective horizontal permeability to oil, m^2
h = reservoir thickness, m
t = flow time, s
ϕ = porosity, fraction
c_t = total compressibility, Pa^{-1}
r_e = distance from wellbore, m
r_w = wellbore radius to the sand face, m
S = skin factor

A.1.2. TPR

Consider a fluid flowing from point 1 to point 2 in a tubing string of length l and height Δz. The first law of thermodynamics yields the following equation for pressure drop:

$$\Delta P = P_1 - P_2 = 9.8062\rho\Delta z + \frac{\rho}{2}\Delta u^2 + \frac{2f_F\rho u^2 L}{D}$$

where

P = pressure drop, Pa
P_1 = pressure at point 1, Pa
P_2 = pressure at point 2, Pa
ρ = fluid density kg/m^3
Δz = elevation increase, m
u = fluid velocity, m/s
f_F = Fanning friction factor
L = tubing length, m
D = tubing inner diameter, m

A.2. Gas

In chapter 4 the different correlations for gas flow are listed in field units. In this section these correlation will be listed again but now in SI units.

A.2.1. IPR

Transient Flow
The transient flow relationship adjusted to a natural gas well is given by:
When converted to SI units this equation becomes:

$$p_{Res}^2 - p_{BH}^2 = \frac{128.77 q \overline{\mu z}(T+273.15)}{kh}\left(\log t + \log\frac{k}{\phi\mu c_t r_w^2} + 8.0261 + 0.87S\right),$$

Steady-State Flow
The steady-state flow relationship adjusted to a natural gas well is given by:

$$p_{Res}^2 - p_{BH}^2 = \frac{111.87 q \overline{\mu z}(T+273.15)}{kh}\left(\ln\frac{r_e}{r_w} + S\right),$$

Pseudo Steady-Sate Flow
The pseudo steady-state flow relationship adjusted to a natural gas well is given by:

$$p_{Res}^2 - p_{BH}^2 = \frac{111.87 q \overline{\mu z}(T+273.15)}{kh}\left(\ln\frac{r_e}{r_w} - \frac{3}{4} + S\right),$$

where

p_{res} = reservoir pressure, Pa
p_{BH} = flowing bottom-hole pressure, Pa
q = production rate, m^3/s
μ = viscosity of the fluid, Pa s
z = gas compressibility factor
T = temperature, °C
k = effective horizontal permeability, m^2
h = reservoir thickness, m
t = flow time, s
ϕ = porosity, fraction
c_t = total compressibility, Pa^{-1}
r_e = distance from wellbore, m
r_w = wellbore radius to the sand face, m
S = skin factor

A.2.2. TPR

For vertical or inclined flow the TPR for single-phase gas flow is given by:

$$p_{in}^2 = e^{-s} p_{out}^2 - 4.0737 \cdot 10^4 \frac{f_F(\overline{z}(\overline{T}+273.15)q)^2}{\sin\theta D^5}(1-e^{-s})$$

where

$$s = \frac{-0.06835\gamma_g \sin\theta L}{\overline{z}(\overline{T}+273.15)}.$$

For horizontal flow the TPR for single-phase gas flow is given by:

$$p_{in}^2 - p_{out}^2 = 2.7847 \cdot 10^3 \frac{f_F \overline{z}(\overline{T}+273.15)q^2 L}{D^5}$$

A.2.3. PPR

Consider steady-state flow of dry gas in a constant-diameter, horizontal pipeline. The mechanical energy equation evaluated over a distance L, is given by:

$$p_{in}^2 - p_{out}^2 = 6.9134 \cdot 10^2 \frac{\gamma_g q^2 (\overline{T}+273.15)\overline{z} f_M L}{D^5}.$$

where

p_{in} = inflow pressure, Pa
p_{out} = outflow pressure, Pa
q = production rate, m^3/s
f_F = Fanning friction factor
z = gas compressibility factor
T = temperature, °C
θ = degrees from horizontal
D = tubing inner diameter, m
γ_g = gas specific gravity
L = pipe length, m
f_M = Moody friction factor

B

List of Conversion Factors

Symbol	Quantity	Field unit	to SI	to Field	SI unit
c_t	Porosity	psi^{-1}	$0.145 \cdot 10^{-3}$	$6.8948 \cdot 10^{3}$	Pa^{-1}
d	Diameter	in	0.0254	39.370	m
h	Reservoir Thickness	ft	0.3084	3.2808	m
J^*	Productivity Index	STB/d-psi	$1.2689 \cdot 10^{-2}$	78.799	m^3/s-Pa
k	Permeability	md	$0.9862 \cdot 10^{-15}$	$1.0133 \cdot 10^{15}$	m^2
m	Mass	lb	0.4536	2.205	kg
p	Pressure	psi	$6.8948 \cdot 10^{3}$	$0.145 \cdot 10^{-3}$	Pa
		lb_f/ft^2	47.8803	$2.0885 \cdot 10^{-2}$	Pa
q	Flow Rate	stb/day	$1.8403 \cdot 10^{-6}$	$5.4344 \cdot 10^{5}$	m^3/s
		bbl/day	$1.8403 \cdot 10^{-6}$	$5.4344 \cdot 10^{5}$	m^3/s
		scf/day	$3.2765 \cdot 10^{-7}$	$3.0524 \cdot 10^{6}$	m^3/s
		ft^3/day	$3.2744 \cdot 10^{-7}$	$3.0512 \cdot 10^{6}$	m^3/s
		MMscfd	0.32765	3.0524	m^3/s
r_w	Wellbore Radius	ft	0.3084	3.2808	m
t	Time	hours	3600	$2.7778 \cdot 10^{-4}$	s
T	Temperature	°F	$(T(°\text{F}) - 32)/1.8$	$1.8T(°\text{C}) + 32$	°C
		°R	$\frac{5}{9}(T(°\text{R}) - 491.67)$	$\frac{9}{5}(T(°\text{C}) + 273.15)$	°C
u	Speed	ft/s	0.3084	3.2808	m/s
ρ	Density	lb_m/ft^3	16.019	$6.2426 \cdot 10^{-2}$	kg/m^3
μ	Viscosity	cp	0.001	1000	Pa s

C

Algorithms

In the development of the Coupling method a lot of routines from MATLAB are used. The routines used are described further in this chapter.

C.1. *fmincon* Algorithm

fmincon is a MATLAB function used for constrained minimization. It finds a minimum of a constrained nonlinear multivariable function. *fmincon* is a matlab function which contains multiple algorithms: the 'interior-point' algorithm, the 'trust-region-reflective' algorithm, the 'sqp' algorithm and the 'active-set' algorithm.

For the 'trust-region-reflective' algorithm a gradient has to be supplied, but because the gradient is not always available in our case, this algorithm will not be used. Further the 'active-set' and 'sqp' algorithms are not large-scale. This means that these methods internally create full matrices and use dense linear algebra. If a problem is sufficiently large, full matrices take up a significant amount of memory, and the dense linear algebra may require a long time to execute. But the 'interior-point' algorithm is large scale, what means that it uses linear algebra that does not need to store, nor operate on, full matrices, and it does not necessarily needs a gradient. That is why there is chosen to uses the 'interior-point' algorithm.

The 'interior-point' algorithm is described in the documentation of MATLAB [8]. For this algorithm, the optimizer chooses one of two steps at each iteration. A Newton step in which the optimizer tries to solve the KKT conditions for the barrier function of the initial objective function and a conjugate gradient step, using a trust region. The algorithm first tries to do a Newton step. If that does not work, it tries a CG step. Fur a full description of this algorithm the reader is referred to [8].

C.2. *fminunc* Algorithm

fminunc is a MATLAB function used for unconstrained minimization. It finds a minimum of a unconstrained nonlinear multivariable function. *fminunc* is a matlab function which contains multiple algorithms: the 'trust-region' algorithm and the 'quasi-newton' algorithm. The 'trust-region' algorithm requires that the gradient is supplied, but since this is not always available in our case, there is chosen to use the 'quasi-newton' algorithm.

The 'quasi-newton' algorithm is described in the documentation of MATLAB [9]. It is an algorithm that uses the BFGS Quasi-Newton method with a cubic line search procedure. For a full description of this algorithm the reader is referred to [9].

C.3. *fsolve* Algorithm

fsolve is a MATLAB function for solving a linear system of equations. *fsolve* attempts to solve systems of equations by minimizing the sum of squares of the components. If the sum of squares is zero, the system of equations is solved. *fsolve* has three algorithms: the 'trust-region-reflective' algorithm, the 'trust-region dogleg' algorithm and the 'Levenberg-Marquardt' algorithm. There is chosen to use the 'trust-region-dogleg' algorithm here.

The 'trust-region-dogleg' algorithm is described in the documentation of MATLAB [10]. For this algorithm the Powell dogleg procedure is used to determine the search direction. This is a convex combination of a Cauchy step and a Gauss-Newton step. For a full description of this algorithm the reader is referred to [10]

C.4. *fzero* Algorithm

fzero is a MATLAB function to determine the root of a nonlinear function. *fzero* attempts to find the root of a scalar function f of a scalar variable x. *fzero* looks for an interval around an initial point such that $f(x)$ changes sign. If you give an initial interval instead of an initial point, *fzero* checks to make sure $f(x)$ has different signs at the endpoints of the interval. The initial interval must be finite; it cannot contain $\pm$inf.

fzero uses a method which is known as Brent's method. Brent's method uses a combination of interval bisection, linear interpolation, and inverse quadratic interpolation in order to locate a root of $f(x)$ and according to Forsythe, Malcolm, and Moler [5] it is one of the best computer algorithms that we have for finding a real zero of a single function. "It combines the certainty of bisection with the ultimate speed of the secand and inverse quadratic interpolation methods for smooth functions." (Forsythe et al. [5]) For a full description of this algorithm the reader is referred to [5].

D

Test Data

D.1. System Test

Parameter	Value	Description
ang	15	Degrees from Vertical
API	40	API Value
B_o	1.1	Fluid Formation Volume Factor of Oil
ϵ	0.001	Relative Roughness
c_t	$9 \cdot 10^{-8}$	Total Compressibility
d	0.20	Pipe Ineer Diameter
D	0.20	Tubing Inner Diameter
γ_o	0.8251	Oil Specific Gravity
h	30	Pay Zone Thickness
k	$8 \cdot 10^{-15}$	Effective Horizontal Permeability
L_{SRF}	30000	Pipe Length
L_{TPR}	200	Tubing Length
μ_o	$1.7 \cdot 10^{-3}$	Oil Viscosity
ϕ	0.2	Porosity
R_e	1000	Radius of Drainage
R_w	0.10	Wellbore radius
ρ_o	824.7346	Density of Oil
T	25	Average Temperature

Table D.1: Parameter for the system test in SI units.

D.2. Test SSRF 1: No Intersections

D.2.1. Data IPR

Bottom-Hole Pressure (Pa)	Flow Rate (m^3/s)
0	0.05
875000	0.049725
1750000	0.0494
2625000	0.049025
3500000	0.0486
4375000	0.048125
5250000	0.0476
6125000	0.047025
7000000	0.0464
7875000	0.045725
8750000	0.045
9625000	0.044225
10500000	0.0434
11375000	0.042525
12250000	0.0416
13125000	0.040625
14000000	0.0396
14875000	0.038525
15750000	0.0374
16625000	0.036225
17500000	0.035
18375000	0.033725
19250000	0.0324
20125000	0.031025
21000000	0.0296
21875000	0.028125
22750000	0.0266
23625000	0.025025
24500000	0.0234
25375000	0.021725
26250000	0.02
27125000	0.018225
28000000	0.0164
28875000	0.014525
29750000	0.0126
30625000	0.010625
31500000	0.0086
32375000	0.006525
33250000	0.0044
34125000	0.002225
35000000	0

Table D.2: Data IPR for Test 1: No intersections.

D.2.2. Data TPR

Flow Rate (m^3/s)	Bottom-Hole Pressure (Pa)
0.035	31700000
0.03625	23768750
0.0375	21125000
0.03875	21139217.5
0.04	21181870
0.04125	21252957.5
0.0425	21352480
0.04375	21480437.5
0.045	21636830
0.04625	21821657.5
0.0475	22034920
0.04875	22276617.5
0.05	22546750
0.05125	22845317.5
0.0525	23172320
0.05375	23527757.5
0.055	23911630
0.05625	24323937.5
0.0575	24764680
0.05875	25233857.5
0.06	25731470
0.06125	26257517.5
0.0625	26812000
0.06375	27394917.5
0.065	28006270
0.06625	28646057.5
0.0675	29314280
0.06875	30010937.5
0.07	30736030
0.07125	31489557.5
0.0725	32271520
0.07375	33081917.5
0.075	33920750
0.07625	34788017.5
0.0775	35683720
0.07875	36607857.5
0.08	37560430
0.08125	38541437.5
0.0825	39550880
0.08375	40588757.5
0.085	41655070

Table D.3: Data TPR for Test 1: No intersections.

D.3. Test SSRF 2: Two Intersections

D.3.1. Data IPR

Bottom-Hole Pressure (Pa)	Flow Rate (m^3/s)
0	0.05
875000	0.049725
1750000	0.0494
2625000	0.049025
3500000	0.0486
4375000	0.048125
5250000	0.0476
6125000	0.047025
7000000	0.0464
7875000	0.045725
8750000	0.045
9625000	0.044225
10500000	0.0434
11375000	0.042525
12250000	0.0416
13125000	0.040625
14000000	0.0396
14875000	0.038525
15750000	0.0374
16625000	0.036225
17500000	0.035
18375000	0.033725
19250000	0.0324
20125000	0.031025
21000000	0.0296
21875000	0.028125
22750000	0.0266
23625000	0.025025
24500000	0.0234
25375000	0.021725
26250000	0.02
27125000	0.018225
28000000	0.0164
28875000	0.014525
29750000	0.0126
30625000	0.010625
31500000	0.0086
32375000	0.006525
33250000	0.0044
34125000	0.002225
35000000	0

Table D.4: Data IPR for Test 2: Two intersections.

D.3.2. Data TPR

Flow Rate (m^3/s)	Bottom-Hole Pressure (Pa)
0.03	21700000
0.03125	13768750
0.0325	11125000
0.03375	11139217.5
0.035	11181870
0.03625	11252957.5
0.0375	11352480
0.03875	11480437.5
0.04	11636830
0.04125	11821657.5
0.0425	12034920
0.04375	12276617.5
0.045	12546750
0.04625	12845317.5
0.0475	13172320
0.04875	13527757.5
0.05	13911630
0.05125	14323937.5
0.0525	14764680
0.05375	15233857.5
0.055	15731470
0.05625	16257517.5
0.0575	16812000
0.05875	17394917.5
0.06	18006270
0.06125	18646057.5
0.0625	19314280
0.06375	20010937.5
0.065	20736030
0.06625	21489557.5
0.0675	22271520
0.06875	23081917.5
0.07	23920750
0.07125	24788017.5
0.0725	25683720
0.07375	26607857.5
0.075	27560430
0.07625	28541437.5
0.0775	29550880
0.07875	30588757.5
0.08	31655070

Table D.5: Data TPR for Test 2: Two intersections.

E

MATLAB Codes

E.1. Functions

```
function [ z ] = zfactor( T, p, gamma_g )
%zfactor: Brills and Begg's  z-factor correlation
%   Brills and Begg's z-factor correlation determines the Gas
%   Compressibility Factor depending on the Temperature and the pressure
%   (Guo: Section 2.3.4) (T in degrees C and p in Pa)

%Pseudo-critical pressure an temperature
p_pc = 709.604 - 58.718*gamma_g;
T_pc = 170.491 + 307.344*gamma_g;

%Pseudo-reduced pressure and temperature
p_pr = (0.145e-3*p)/p_pc;
T_pr = ((T*1.8+32)+459.67)/T_pc;

%Brill and Beggs correlation
A = 1.39*(T_pr-0.92)^0.5 - 0.36*T_pr-0.10;
C = 0.132 - 0.32*log10(T_pr);
E = 9*(T_pr-1);
F = 0.3106 - 0.49*T_pr+0.1824*T_pr^2;

B = (0.62-0.23*T_pr)*p_pr+(0.066/(T_pr-0.86)-0.037)*p_pr^2+0.32*p_pr^6/10^E;
D = 10^F;

z = A + (1-A)/exp(B) + C*p_pr^D;

end
```

```
function [ f ] = friction( Re, epsilon )
%Moody's Friction factor according to Churchill
%   Churchill's correlation determines the Fanning Friction factor
%   depending on the Reynolds number and the relative roughness

 A = (-2.457*log((7/Re)^0.9+0.27*epsilon))^16;
 B = (37530/Re)^16;
 f = 8*((8/Re)^12+(A+B)^(-1.5))^(1/12);

end
```

E.2. Parameter objects

```
classdef Par_Fluid < handle

    properties
        %Here the initial values of the different properties are given.
        State   = 1;                %1 for Oil flow, 2 for Gas flow,
                                    %3 multiphase flow and 4 for Tabular

        Flow    = 2;                %1 Transient, 2 Steady-state,
                                    %3 Pseudo-Steady-State

        API     = 40;               %API
        Bo      = 1.1;              %Fluid formation Volume Factor of Oil
        epsilon = 0.001;            %Relative roughness
        gamma_g = 0.76;             %Gas specific gravity
        gamma_o                     %Oil-specific gravity
        mu_g    = 1e-5;             %Gas viscosity (Pa s)
        mu_o    = 1.7e-3;           %Oil viscosity (Pa s)
        p_b     = 3.5e5             %Bubble-Point pressure (Pa)
        rho_g;                      %Density of gas (kg/ m^3)
        rho_o;                      %Density of oil (kg/ m^3)
        T       = 25;               %Average temprature (degrees C)
    end

    methods
        function obj = Par_Fluid
            %Function for creating the object Par_TPR
            %   While creating the object Par_Fluid the oil specific
            %   gravity, the density of gas and the density of oil are
            %   calculated according to the initial values of the gas
            %   specific gravity and the API value.
            obj.gamma_o = 141.5/(obj.API+131.5);
            obj.rho_g = 1.225*obj.gamma_g;
            obj.rho_o = 999.59*obj.gamma_o;
        end

        function Rs = GOR(obj, p)
            % Empirical correlation for solution GOR (Guo: Section 2.2.1)
            Rs = obj.gamma_g*(0.145e-3*p/18 * (10^(0.0125*obj.API))/...
                (10^(0.00091*(obj.T*1.8+32))))^1.2048;
        end

    end

end
```

```
classdef Par_IPR < handle

    properties
        %Here the initial values of the different properties are given.
        ct      = 9e-8;             %Total compressibility (Pa^-1)
        h       = 30;               %Pay zone thickness (m)
        k       = 8e-15;            %Effective horizontal permeability(m^2)
        phi     = 0.2;              %Porosity
        re      = 1000;             %Radius of drainage area (m)
        rw      = 0.10;             %Wellbore radius (m)
        t       = 30*24*3600;       %Flow time (s)
    end

end
```

```
classdef Par_TPR < handle

    properties
        %Here the initial values of the different properties are given.
        ang     = 15;              %Degrees from vertical
        D       = 0.20;            %Tubing inner diameter (m)
        Du      = 0;               %Increase in Fluid Velocity
        L_TPR   = 200;             %Tubing length (m)
        Dz                         %Elevation increase
        WOR     = 0;               %Water Oil Ratio
        gamma_w = 1;               %Water specific gravity (WOR =0)
        Bw      = 1;               %Fluid formation volume factor (WOR = 0)
        GOR     = 141.8;           %Gas Oil Ratio
        rho_air = 1.2230;          %Density of air (kg / m^3)
        theta                      %Degrees from horizontal
    end

    methods
        function obj = Par_TPR
            %Function for creating the object Par_TPR
            %   While creating the object Par_TPR the elevation increase
            %   and the degrees from horizontal are caculated according to
            %   the initial values of the degrees from vertical and the
            %   tubing length.
            obj.Dz      = cos(obj.ang/180*pi)*obj.L_TPR;
            obj.theta   = 90 - obj.ang;
        end

        function obj = set_length(obj, length)
            %set_length changes the tubing length
            %   Because the length changes also the elevation increase
            %   changes and that should thus also be calculated again.
            obj.L_TPR = length;
            obj.Dz = cos(obj.ang/180*pi)*obj.L_TPR;
        end

    end

end
```

```
classdef Par_PPR < handle

    properties
        %Here the initial values of the different properties are given.
        d       = 0.2;             %Pipe inner diameter (m)
        L_SRF   = 30000;           %Pipe length (m)
        theta   = 0;               %Dip angle from horizontal (degrees)
    end

end
```

E.3. Subsurface

```
classdef WellInflow < handle
    %Well inflow class

    properties
        ID;
        parF;
        parIPR
        q;
        p_bh;
        p_res;
    end

    properties (Access = private)
        F
    end

    methods
        function obj = WellInflow(parFluid, parWellInflow, id)
            obj.parF = parFluid;
            if obj.parF.State == 4
                table = readtable('Flow_IPR.xls', 'ReadVariableNames', false);
                T= table2array(table);
                Q = T(:,2);
                P = T(:,1);

                obj.F = griddedInterpolant(P, Q);
            end
            obj.parIPR = parWellInflow;
            obj.ID = id;
        end
        function q_bh = IPR(obj)
            %The Inflow Performance Relationship.
            %Explicit expression for q_bh as function of p_bh and p_res.
            if obj.parF.State == 1
                % Oil Flow (Guo: Section 3.3.1)
                %Transient
                if obj.parF.Flow == 1
                    J = (5.4567*obj.parIPR.k*obj.parIPR.h)/...
                        (obj.parF.Bo((obj.p_res+obj.p_bh)/2)*...
                        obj.parF.mu_o((obj.p_bh+obj.p_res)/2)*...
                        (log10(obj.parIPR.t)+log10(obj.parIPR.k/...
                        (obj.parIPR.phi*obj.parF.mu_o((obj.p_bh+...
                        obj.p_res)/2)*obj.parIPR.ct*obj.parIPR.rw^2))+...
                        0.34895));
                % Steady-State
                elseif obj.parF.Flow == 2
                    J = (6.2837*obj.parIPR.k*obj.parIPR.h)/...
                        (obj.parF.Bo*obj.parF.mu_o*log(obj.parIPR.re/...
                        obj.parIPR.rw));
                % Pseudo-Steady-State
                elseif obj.parF.Flow == 3
                    J = (6.2837*obj.parIPR.k*obj.parIPR.h)/(obj.parF.Bo*...
                        obj.parF.mu_o*(log(obj.parIPR.re/obj.parIPR.rw)-3/4));
                end

                q_bh = (obj.p_res - obj.p_bh)*J;

            elseif obj.parF.State == 2
                % Gas Flow (Economides: Section 4.3)
                % Transient
                if obj.parF.Flow == 1
                    z = zfactor(obj.parF.T, (obj.p_bh + obj.p_res)/2,...
                        obj.parF.gamma_g);
                    q_bh = (7.7657e-3*obj.parIPR.k* obj.parIPR.h*...
                        ((obj.p_res)^2-(obj.p_bh)^2)/(obj.parF.mu_g*z*...
                        (obj.parF.T+273.15)*(log10(obj.parIPR.t)+...
                        log10(obj.parIPR.k/(obj.parIPR.phi*obj.parF.mu_g*...
```

```
                        obj.parIPR.ct*obj.parIPR.rw^2))+8.0261)));
                % Steady-State
                elseif obj.parF.Flow == 2
                    z = zfactor(obj.parF.T,(obj.p_bh + obj.p_res)/2,...
                        obj.parF.gamma_g);
                    q_bh = (8.9382e-3*obj.parIPR.k*obj.parIPR.h*...
                        ((obj.p_res)^2-(obj.p_bh)^2)/(obj.parF.mu_g*z*...
                        (obj.parF.T+273.15)*log(obj.parIPR.re/obj.parIPR.rw)));
                % Pseudo-Steady-Sate
                elseif obj.parF.Flow == 3
                    z = zfactor(obj.parF.T,(obj.p_bh + obj.p_res)/2,...
                        obj.parF.gamma_g);
                    q_bh = (8.9382e-3*obj.parIPR.k*obj.parIPR.h*...
                        ((obj.p_res)^2-(obj.p_bh)^2)/(obj.parF.mu_g*z*...
                        (obj.parF.T+273.15)*(log(obj.parIPR.re/...
                        obj.parIPR.rw)-3/4)));
                end

            elseif obj.parF.State == 3
                % Multiphase Flow (according to Vogel's)
                % (Guo: Section 3.3.2 )
                %Transient
                if obj.parF.Flow == 1
                    J = 5.4567*obj.parIPR.k*obj.parIPR.h/(obj.parF.Bo*...
                        obj.parF.mu_o*(log10(obj.parIPR.t)+...
                        log10(obj.parIPR.k/(obj.parIPR.phi*obj.parF.mu_o*...
                        obj.parIPR.ct*obj.parIPR.rw^2))+8.0261));
                    Q_max = J*obj.p_res/1.8;
                %Steady-State
                elseif obj.parF.Flow == 2
                    J = 6.2837*obj.parIPR.k*obj.parIPR.h/(obj.parF.Bo*...
                        obj.parF.mu_o*log(obj.parIPR.re/obj.parIPR.rw));
                    Q_max = J*obj.p_res/1.8;
                %Pseudo-Steady-State
                elseif obj.parF.Flow == 3
                    J = 6.2837*obj.parIPR.k*obj.parIPR.h/(obj.parF.Bo*...
                        obj.parF.mu_o*(log(obj.parIPR.re/obj.parIPR.rw)-3/4));
                    Q_max = J*obj.p_res/1.8;
                end
                q_bh = Q_max*(1-0.2*(obj.p_bh/obj.p_res)-0.8*(obj.p_bh/...
                    obj.p_res)^2);

            elseif obj.parF.State == 4
                q_bh = F(obj.p_bh);

            end
        end
    end

end
```

```
classdef Wellbore < handle
    %Wellbore class

    properties
        parF;
        parTPR;
        ID;
        q;
        p_th;
        p_bh;
    end

    properties (Access = private)
        T
        F
    end

    methods
        function obj = Wellbore(parFluid, parWellbore, id)
            obj.parF = parFluid;
            if obj.parF.State == 4
                % If the state is tabular the values form the table are
                % read only once during the making of the object.
                table = readtable('Flow_TPR.xls', 'ReadVariableNames', false);
                obj.T = table2array(table);

                Q       = obj.T(:,1);
                P_TH    = obj.T(1,:);
                Q(1)    = [];
                P_TH(1) = [];
                P_BH    = obj.T;
                P_BH(:,1) = [];
                P_BH(1,:) = [];

                [Q, P_TH] = ndgrid(Q, P_TH);
                obj.F = griddedInterpolant(Q, P_TH, P_BH);
            end

            obj.parTPR = parWellbore;
            obj.ID = id;
        end

        function p_bh = TPR(obj)
            %The Tubing Performance Relationship.
            %Explicit expression for p_bh as function of q and p_th.
            u  = 4*obj.q/(pi*obj.parTPR.D^2);
            Re = obj.parF.rho_o*u*obj.parTPR.D/obj.parF.mu_o;
            fF = friction(Re,obj.parF.epsilon)/4;
            if obj.parF.State == 1
                % Oil Flow (Guo: Section 4.2)
                p_bh = (obj.p_th + 9.8062*obj.parF.rho_o*obj.parTPR.Dz +...
                    obj.parF.rho_o*obj.parTPR.Du^2/2 + 2*fF*...
                    obj.parF.rho_o*u^2*obj.parTPR.L_TPR/obj.parTPR.D);

            elseif obj.parF.State == 2
                % Gas Flow (Economides: Section 7.3)
                % z-factor taken at p_th!
                z    = zfactor(obj.parF.T, obj.p_th, obj.parF.gamma_g);
                s    = -0.06835*obj.parF.gamma_g*sin(obj.parTPR.ang)*...
                    obj.parTPR.L_TPR/((obj.parF.T+273.15)*z);
                p_bh2 = ( exp(-s)* (obj.p_th)^2-...
                    4.0737e4*fF*(z*(obj.parF.T+273.15)*obj.q)^2/...
                    (sin(obj.parTPR.ang)*obj.parTPR.D^5)*(1-exp(-s)) );
                p_bh = sqrt(p_bh2);

            elseif obj.parF.State == 3
                % Multiphase Flow (according to Poettmann and Carpenter)
                % (Guo: Section 4.3.3.1)
                % z-factor taken at p_th!
                z    = zfactor(obj.parF.T, obj.p_th, obj.parF.gamma_g);
```

```
71                  M   = 158.84*(obj.parF.gamma_o + obj.parTPR.WOR*...
72                      obj.parTPR.gamma_w) + 2.8316e-2*obj.parTPR.GOR*...
73                      obj.parTPR.rho_air*obj.parF.gamma_g;
74                  Dpv = 23.943*M*abs(obj.q)/obj.parTPR.D;
75                  f_2F = 10^(3.0645-2.5*log10(Dpv));
76                  Rs  = obj.parF.gamma_g*(0.145e-3*obj.p_th/18 *...
77                      (10^(0.0125*obj.parF.API))/...
78                      (10^(0.00091*(obj.parF.T*1.8+32))))^1.2048;
79                  V   = 0.15900*(obj.parF.Bo + obj.parTPR.WOR*...
80                      obj.parTPR.Bw)+9.9372*(obj.parTPR.GOR - Rs)*...
81                      ((obj.parF.T+273.15)*z/(obj.p_th));
82                  rho = M/V;
83                  k = 5.0955e-2*f_2F*obj.q^2*M^2/(obj.parTPR.D^5);
84                  p_bh = obj.p_th + (6.2426e-2*rho+k/(6.2426e-2*rho))*...
85                      157.09*obj.parTPR.L_TPR;
86
87
88              elseif obj.parF.State == 4
89                  p_bh = obj.F(obj.q, obj.p_th);
90
91              end
92
93          end
94
95          function [Q_min, Q_max] = minmax(obj)
96              %This function determines the lowest and highest flow rate
97              %for which the table provides values.
98
99              %Only use this function if State == 4 (Tabular)
100             Q = obj.T(:,1);
101
102             Q(1)=[];
103             Q_min = Q(1);
104             Q_max = Q(length(Q));
105
106         end
107
108     end
109
110 end
```

```
classdef Well < handle
    %Well class

    properties
        ID;
        wellbore;
        wellInflow;
        p_scale;
        q_scale;
        feasible;
        n = 0;
    end

    methods
        function obj = Well(parFluid, parWellInflow, parWellbore, id)
            obj.wellbore = Wellbore(parFluid, parWellbore, id);
            obj.wellInflow = WellInflow(parFluid, parWellInflow, id);
            obj.ID = id;
        end
        function q_th = WPR(obj, p_th, p_scale, q_scale)
            %The Well Performance Relationship. It solves the well flow.
            %It returns the tubing-head volume flow rate, given the tubing-head
            %pressure.
            obj.wellbore.p_th = p_th;
            obj.p_scale = p_scale;
            obj.q_scale = q_scale;



            %Initial values for p_bh and q
            p_bh0  = (obj.wellbore.p_th + obj.wellInflow.p_res) / 2.0;
            q0      = 0.03;
            y = zeros(2,1);
            y(1) = p_bh0 / obj.p_scale;
            y(2) = q0 / obj.q_scale;


            x = obj.Solve( y );

            %The actual solution is scaled back
            p_bh = x(1) * obj.p_scale;
            q = x(2) * obj.q_scale;
            q_th = q;

            obj.n = obj.n+1;

        end
        function x = Solve(obj, x0)
            %This function actually solves the bottomholesystem.
            Func = @obj.BottomHoleSystem;

            %Check if extrapolation should be on to determine a solution
            obj.Check;

            %Scaled epsilon
            epsilon = 1e-6;

            % Using quasi-newton algorithm because for default
            % (trust-region) a gradient must be provided.
            options = optimoptions('fminunc', 'Algorithm',...
                'quasi-newton', 'TolFun', epsilon, 'TolX', epsilon*1e-3);


            %Non Scaled epsilon
            epsilon_p = epsilon*obj.p_scale;
            epsilon_q = epsilon*1e-3*obj.q_scale;

            x   = x0;

            %Scaled solution
```

```
71              x = fminunc(Func, x, options);
72
73              %If the minimum isn't reached in an epsilon around zero the two
74              %curves have no intersection and the point on the IPR curve
75              %with the least distance to the TPR curve will be chosen
76              %instead.
77              if obj.BottomHoleSystem(x) > epsilon
78                  obj.wellInflow.p_bh = x(1)*obj.p_scale;
79                  x(2) = obj.wellInflow.IPR()/obj.q_scale;
80                  obj.feasible = 0;
81              else
82                  obj.feasible = 1;
83
84                  %Test if a 'wrong' intersection point is found
85
86                  %Scaled step which takes inaccuracy into account
87                  %(outside epsilon range)
88                  h = 2*epsilon;
89
90                  x1 = x+h;
91                  x2 = x-h;
92
93                  %Non Scaled values of P a (scaled) step h higher and lower
94                  P_plus = x1(1)*obj.p_scale;
95                  P_min = x2(1)*obj.p_scale;
96
97                  %Non Scaled values of Q following from the IPR curve in point
98                  %P_min and P_plus
99                  obj.wellInflow.p_bh = P_plus;
100                 Q1 = obj.wellInflow.IPR();
101
102                 obj.wellInflow.p_bh = P_min;
103                 Q2 = obj.wellInflow.IPR();
104
105                 %Non Scaled values of P following from the TPR curve in point
106                 %Q1 and Q2
107                 obj.wellbore.q = Q1;
108                 P1 = obj.wellbore.TPR();
109
110                 obj.wellbore.q = Q2;
111                 P2 = obj.wellbore.TPR();
112
113                 %Comparison of the P values according to the IPR and TPR in the
114                 %points Q1 and Q2
115                 if P2 < P_min && P1 > P_plus
116                     % The 'wrong' intersection point is found
117
118                     % Check if the function has a second intersection point
119
120                     % The IPR ends for with a BHP of 1 bar.
121                     obj.wellInflow.p_bh = 1e5;
122                     Q0 = obj.wellInflow.IPR();
123
124                     obj.wellbore.q = Q0;
125                     P0 = obj.wellbore.TPR();
126
127                     % There has to be a second intersection point if P0>1e5.
128                     if P0 > 1e5
129                         % Finding the second intersection point within a
130                         % interval with a guaranteed sign change.
131                         Func2 = @obj.Difference;
132
133                         ub = P2/obj.p_scale; %Scaled
134                         lb = 0;
135                         x0 = [lb ub];
136
137                         P = fzero(Func2, x0); %Scaled
138
139                         x(1) = P;
140                         obj.wellInflow.p_bh = P*obj.p_scale;
141                         obj.wellbore.p_bh = P*obj.p_scale;
```

```
                        Q = obj.wellInflow.IPR();
                        x(2) = Q/obj.q_scale;
                    else
                        % If there is not a second intersection point and
                        % the first intersection point is infeasible the
                        % lowest possible pressure will be selected (1 bar)

                        x(1) = 1e5/obj.p_scale;
                        obj.wellInflow.p_bh = 1e5;
                        Q = obj.wellInflow.IPR();
                        x(2) = Q/obj.q_scale;

                        % If this happens in the last iteration the well
                        % will be closed.
                        obj.feasible = 0;

                    end

                end
            end

            % Q can't be smaller then Qmin, and is set to equal Qmin
            % otherwise.
            Q_min = 1e-4;
            if x(2)*obj.q_scale < Q_min
                x(2) = Q_min/obj.q_scale;
                obj.feasible = 0;
            end

        end

    end

    methods (Access = private)
        function F = BottomHoleSystem(obj,x)
            %BottomHoleSystem determines the Residuals from the TPR
            %and the IPR following from the given Bottom Hole Pressure and
            %Flow (in x) and returns these in one vector F.

            p_bh = x(1) * obj.p_scale;
            q = x(2) * obj.q_scale;
            obj.wellbore.p_bh = p_bh;
            obj.wellbore.q = q;
            obj.wellInflow.p_bh = p_bh;
            obj.wellInflow.q = q;

            F = zeros(2,1);
            F(1) = (p_bh - obj.wellbore.TPR()) / obj.p_scale;
            F(2) = (q - obj.wellInflow.IPR()) / obj.q_scale;
            F = norm(F);
        end

        function G = Difference(obj, P)
            obj.wellInflow.p_bh = P*obj.p_scale;
            obj.wellbore.q = obj.wellInflow.IPR();

            G = P - obj.wellbore.TPR()/obj.p_scale;
        end
    end

end
```

```
classdef Subsurface < handle
    %Subsurface system class

    properties
        well = Well.empty(); % creates an empty array of Well objects.
        p_res;
        n_wells;
```

```
8       end
9
10      methods
11          function obj = Subsurface(parFluid, parWellInflow, parWellbore,...
12                  n_wells, p_res)
13              % Builds a subsurface system with n_wells wells.
14              % For the time being all wells are the same.
15              obj.p_res = p_res;
16              obj.n_wells = n_wells;
17              for i = 1:n_wells
18                  well_new = Well(parFluid, parWellInflow(i),...
19                      parWellbore(i), i);
20                  obj.well = [obj.well, well_new]; % add well to the list
21              end
22              for i = 1:obj.n_wells
23                  obj.well(i).wellInflow.p_res = obj.p_res;
24              end
25          end
26
27          function q_th = SSRF(obj, p_th, p_scale, q_scale)
28              %The Subsurface Response Function. It solves the well flows.
29              %It returns the tubing-head volume flow rates, given the
30              %tubing-head pressures.
31              q_th = zeros(obj.n_wells,1);
32              % Make sure the current reservoir pressure is used.
33              for i = 1:obj.n_wells
34                  obj.well(i).wellInflow.p_res = obj.p_res;
35              end
36              for i = 1:obj.n_wells
37                  q_th(i) = obj.well(i).WPR(p_th(i), p_scale, q_scale);
38              end
39          end
40      end
41
42  end
```

E.4. Surface

```
classdef Choke < handle
    %Choke class

    properties
        parF;
        parPPR;
        ID;
        q;
        p_th;
        p_c;
        alpha;
    end

    methods
        function obj = Choke(parFluid, parPipe, id)
            obj.parF = parFluid;
            obj.parPPR = parPipe;
            obj.ID = id;
        end

        function p_th = CPR(obj, q)
            % The Choke Performance Relationship
            A = 1/4*pi*obj.parPPR.d^2;

            if obj.parF.State == 1
                rho = obj.parF.rho_o;
            elseif obj.parF.State == 2
                rho = obj.parF.rho_g;
            elseif obj.parF.State == 3
                 z   = zfactor(obj.parF.T, abs(obj.p_c), obj.parF.gamma_g);
                 u_l = 4*q/(pi*obj.parPPR.d^2);
                 u_g = 4*q/(pi*obj.parPPR.d^2)*z*((obj.parF.T+...
                     273.15)/(15.556+273.15))*(1.0135e5/obj.p_c);
                 u_m = u_l + u_g;
                 lambda_l = u_l / u_m;
                 lambda_g = u_g / u_m;
                 if u_m == 0;
                     lambda_l = 0;
                     lambda_g = 0;
                 end
                 rho_l = obj.parF.rho_o;
                 rho_g = obj.parF.rho_g;
                 rho = rho_l*lambda_l + rho_g*lambda_g;
            elseif obj.parF.State == 4
                rho = obj.parF.rho_o;
            end

            p_th = rho*q^2/(2*obj.alpha^2*A^2) + obj.p_c;

        end

    end

end
```

```
classdef Pipe < handle
    %Pipe class

    properties
        ID;
        parF;
        parPPR;
        q;
        p_in;
        p_out;
    end

    methods
        function obj = Pipe(parFluid, parPipe, id)
            obj.parF = parFluid;
            obj.parPPR = parPipe;
            obj.ID = id;
        end
        function PPR(obj)

            if obj.parF.State == 1
                % Oil Flow (Guo: Section 11.4.1.1)
                u_l = 4*obj.q/(pi*obj.parPPR.d^2);
                Re = obj.parF.rho_g*abs(u_l)*obj.parPPR.d/obj.parF.mu_o;
                f_M = friction(Re,obj.parF.epsilon);
                obj.p_in = (9.7947e3* obj.parF.gamma_o*obj.parPPR.L_SRF*...
                    sin(obj.parPPR.theta)+8.1222e2*f_M*obj.parF.gamma_o*...
                    obj.q^2*obj.parPPR.L_SRF/obj.parPPR.d^5 + obj.p_out);
            elseif obj.parF.State == 2
                 % Gas Flow (Guo: Section 11.4.1.2) Horizontal pipeline!!
                 % z-factor taken at p_out!
                 z   = zfactor(obj.parF.T, obj.p_out, obj.parF.gamma_g);
                 u_g = 4*obj.q/(pi*obj.parPPR.d^2)*z*((obj.parF.T+...
                     273.15)/(15.556+273.15))*(1.0135e5/obj.p_out);
                 Re = obj.parF.rho_g*abs(u_g)*obj.parPPR.d/...
                     obj.parF.mu_g;
                 f_M = friction(Re,obj.parF.epsilon);

                 p_in2 = ((6.9134e2*obj.parF.gamma_g*(obj.parF.T+273.15)*...
                     z*obj.q^2*f_M*obj.parPPR.L_SRF/obj.parPPR.d^5) +...
                     (obj.p_out)^2);
                 obj.p_in = sqrt(p_in2);
            % Multiphase Flow (Economides: Section 7.4.3)
            % Horizontal pipeline !!!
            elseif obj.parF.State == 3 || obj.parF.State == 4
                z   = zfactor(obj.parF.T, (obj.p_out), obj.parF.gamma_g);
                u_l = 4*obj.q/(pi*obj.parPPR.d^2);
                u_g = 4*obj.q/(pi*obj.parPPR.d^2)*z*((obj.parF.T+...
                    273.15)/(15.556+273.15))*(1.0135e5/obj.p_out);
                u_m = u_l + u_g;
                lambda_l = u_l / u_m;
                lambda_g = u_g / u_m;

                N_FR = 0.10198*u_m^2 /obj.parPPR.d;
                L1   = 316*lambda_l^(0.302);
                L2   = 0.000925*lambda_l^(-2.4684);
                L3   = 0.10*lambda_l^(-1.4516);
                L4   = 0.5*lambda_l^(-6.738);

                % Segregated Flow
                if ((lambda_l < 0.01) && (N_FR < L1)) || ...
                        ((lambda_l >= 0.01) && (N_FR < L2))
                    FlowRegime = 1;
                    a = 0.98;
                    b = 0.4846;
                    c = 0.0868;
                    % Transition Flow
                elseif  (lambda_l >= 0.01) && ...
                        ((L2 < N_FR) && (N_FR <= L3))
                    FlowRegime = 2;
```

```
71                      as = 0.98;
72                      bs = 0.4846;
73                      cs = 0.0868;
74                      ai = 0.845;
75                      bi = 0.5351;
76                      ci = 0.0173;
77                      % Intermittent Flow
78                  elseif (((0.01 <=lambda_l)&&(lambda_l<0.4))&&...
79                          ((L3<N_FR)&&(N_FR <=L1))) || ...
80                          ((lambda_l >= 0.4)&&((L3<N_FR)&&(N_FR<=L4)))
81                      FlowRegime = 3;
82                      a = 0.845;
83                      b = 0.5351;
84                      c = 0.0173;
85                      % Distributed Flow
86                  elseif ((lambda_l < 0.4)&& (N_FR >= L1)) || ...
87                          ((lambda_l >=0.4) && (N_FR > L4))
88                      FlowRegime = 4;
89                      a = 1.065;
90                      b = 0.5824;
91                      c = 0.0609;
92                  end
93
94                  if FlowRegime == 2
95                      A = (L3 - N_FR)/(L3-L2);
96                      B = 1-A;
97                      y_ls = as*lambda_l^bs/(N_FR^cs);
98                      y_li = ai*lambda_l^bi/(N_FR^ci);
99                      y_l = A*y_ls+B*y_li;
100                 else
101                     y_l = a*lambda_l^b/(N_FR^c);
102                 end
103
104                 %WOR = 0
105                 rho_l = obj.parF.rho_o;
106                 rho_g = obj.parF.rho_g;
107                 rho_m = rho_l*lambda_l + rho_g*lambda_g;
108
109                 mu_l = obj.parF.mu_o;
110                 mu_m = mu_l*lambda_l + obj.parF.mu_g*lambda_g;
111
112                 Re = rho_m*abs(u_m)*obj.parPPR.d/mu_m;
113                 fn = friction(Re,0)/4;
114
115                 x = lambda_l / y_l^2;
116                 if (1<x) && (x<1.2)
117                     S = log(2.2*x-1.2);
118                 else
119                     S = log(x)/(-0.0523+3.182*log(x)-0.8725*...
120                         (log(x))^2+0.01853*(log(x))^4);
121                 end
122
123                 ftp = fn*exp(S);
124                 dpdz = (86.766*ftp*rho_m*u_m)/(obj.parPPR.d);
125                 obj.p_in = dpdz*obj.parPPR.L_SRF + obj.p_out;
126
127             end
128
129         end
130
131     end
132
133 end
```

```
classdef Surface < handle
    %Surface system class

    properties
        inletPipe = Pipe.empty();  % creates an empty array of Pipe objects
        outletPipe = Pipe.empty(); % creates an empty array of Pipe objects
        choke = Choke.empty();     % creates an empty array of Choke objects
        p_out;
        q_out;
        n_inlets;
        n_deep;
    end

    methods
        function obj = Surface(parFluid, parPipe, n_inlets, n_deep, p_out)
            % Builds a converging network with n_inlets inlet pipes,
            % ending in 1 outlet.
            % If n_deep=1 all inlet pipes connect to the same outlet
            % pressure point.
            % If n_deep=2 all inlet pipes connect to 1 outlet pipe which
            % connects to the outlet pressure point.
            obj.p_out = p_out;
            if n_deep == 1
                % All inlet pipes converge to the same pressure point, and
                % there is no separate outlet pipe.
                for i = 1:n_inlets
                    pipe = Pipe(parFluid, parPipe(i), i);
                    obj.inletPipe = [obj.inletPipe, pipe]; % add pipe to the list
                end

                for i = 1:obj.n_inlets
                    obj.inletPipe(i).p_out = obj.p_out;
                end

            elseif n_deep == 2
                % All inlet pipes converge to an outlet pipe.
                for i = 1:n_inlets
                    pipe = Pipe(parFluid, parPipe(i), i);
                    obj.inletPipe = [obj.inletPipe, pipe]; % add pipe to the list
                end

                pipe = Pipe(parFluid, parPipe(n_inlets+1), n_inlets + 1);
                obj.outletPipe = [obj.outletPipe, pipe]; % add pipe to the list
                obj.outletPipe(1).p_out = obj.p_out;
            end

            % Each inlet pipe is also connected to a choke. For the
            % nonlinear system of equations version this must be commented.
            for i = 1 : n_inlets
                newChoke = Choke(parFluid, parPipe(i), i);
                obj.choke = [obj.choke, newChoke]; % add choke to the list
            end

            obj.n_inlets = n_inlets;
            obj.n_deep = n_deep;
        end

        function p_in = SRF(obj, q_in)
            %The Surface Response Function. It solves the network flow.
            %It returns the inflow pressures, given the inflow volume flow
            %rates.
            p_in = zeros(obj.n_inlets,1);
            if obj.n_deep == 1
                % All inlet pipes converge to the same pressure point, and
                % there is no separate outlet pipe.
                % Make sure the current outlet pressure is used.
                for i = 1:obj.n_inlets
                    obj.inletPipe(i).p_out = obj.p_out;
                end

```

```
71                  % Calculate q's.
72                  q_total = 0.0;
73                  for i = 1:obj.n_inlets
74                      obj.inletPipe(i).q = q_in(i);
75                      q_total = q_total + q_in(i);
76                  end
77
78                  obj.q_out = q_total;
79                  % Calculate p's.
80                  for i = 1:obj.n_inlets
81                      obj.inletPipe(i).PPR
82                  end
83
84                  for i = 1:obj.n_inlets
85                      p_in(i) = obj.inletPipe(i).p_in;
86                  end
87
88              elseif obj.n_deep == 2
89                  % All inlet pipes converge to 1 outlet pipe.
90                  % Make sure the current outlet pressure is used.
91                  obj.outletPipe(1).p_out = obj.p_out;
92                  % Calculate q's.
93                  q_total = 0.0;
94                  for i = 1:obj.n_inlets
95                      obj.inletPipe(i).q = q_in(i);
96                      q_total = q_total + q_in(i);
97                  end
98
99                  obj.outletPipe(1).q = q_total;
100                 obj.q_out = q_total;
101                 % Calculate p's.
102                 obj.outletPipe(1).PPR
103                 for i = 1:obj.n_inlets
104                     obj.inletPipe(i).p_out = obj.outletPipe(1).p_in;
105                 end
106
107                 for i = 1:obj.n_inlets
108                     obj.inletPipe(i).PPR
109                 end
110
111                 for i = 1:obj.n_inlets
112                     p_in(i) = obj.inletPipe(i).p_in;
113                 end
114
115             end
116
117         end
118
119     end
120
121 end
```

E.5. System

```
% This is the main program for the entire coupling.

% First everything is cleared.
clear all;
clc;

% A Fluid Parameter object is created.
parFluid = Par_Fluid;

parWellInflow = Par_IPR.empty(); % creates an empty array of IPR Parameter objects
parWellbore = Par_TPR.empty();   % creates an empty array of TPR Parameter objects
parPipe = Par_PPR.empty();       % creates an empty array of PPR Parameter objects

n_wells = 3;
n_deep  = 1;

% For each well there is added a IPR-, a TPR- and a PPR parameter object.
for i = 1 : n_wells
    parWellInflow = [parWellInflow, Par_IPR]; % add IPR Parameter object to the list.
    parWellbore = [parWellbore, Par_TPR];     % add TPR Parameter object to the list.
    parPipe = [parPipe, Par_PPR];             % add PPR Parameter object to the list.
end

% Changing the individual parameters of the differnt wells.
parWellbore(1).set_length(2000);
parPipe(3).d = 0.1;

% Adding an extra Pipe Parameter object for the pipe that is needen in the
% case that n_deep = 2.
if n_deep ==2
    parPipe = [parPipe, Par_PPR];
end

% This creates the system object with the different parameter object,
% n_wells and n_deep and with a reservoir pressure of 350 bar and an outlet
% pressure of 15 bar.
sys = System(parFluid, parWellInflow, parWellbore, parPipe, n_wells,...
    n_deep, 350e5, 15e5);

% This solves the entire system.
sys.Solve

% These are the results
p_bh = sys.p_bh_sol .* 1e-5
p_th = sys.p_th_sol .* 1e-5
p_out = sys.surface.inletPipe(1).p_out .* 1e-5
if n_deep == 2
    p_middel = sys.surface.inletPipe(1).p_out .* 1e-5
    p_out = sys.surface.outletPipe(1).p_out * 1e-5
end
q = sys.q_sol.*1e3
q_out = sys.surface.q_out.*1e3
Delta_p_th = sys.Delta_p_th_sol.*1e-5
alpha = sys.alpha_sol
```

E.5.1. Non-Linear System of Equations

```
classdef System < handle
    %Production system class.
    %The assumption is that the surface system has exactly the number of
    %inlet pipes as there are wells in the subsurface system.
    %Furthermore, the first pipe is connected to the first well, the second
    %pipe to the second well, etc.

    properties
        surface;
        subsurface;
        n_wells;
        q_sol;
        p_th_sol;
        p_bh_sol;
        p_scale = 1.e7; % scaled pressure is p/p_scaled, p in [Pa].
        q_scale = 1.e-3; % scaled volume rate is q/q_scaled, q in [m3/s].

    end

    methods
        function obj = System(parFluid, parWellInflow, parWellbore, parPipe,...
                n_wells, n_deep, p_res, p_outlet)
            % Input pressures to be given in [Pa].
            obj.n_wells = n_wells;
            obj.subsurface = Subsurface(parFluid, parWellInflow,...
                parWellbore, n_wells, p_res);
            obj.surface = Surface(parFluid, parPipe, n_wells, n_deep,...
                p_outlet);
        end


        function x = Solve(obj)
            %Solve is the function which actually performs the coupling
            %between the Surface and the Subsurface object. This function
            %determines the Tubing Head Pressures and the Flows.

           %Initial values for p_th and q
            p_th0 = (obj.surface.p_out + obj.subsurface.p_res) / 2.0;
            q0      = 1.0e-5;
            y   = zeros(2*obj.n_wells,1);
            y(1:obj.n_wells,1)= p_th0 / obj.p_scale;
            y(obj.n_wells+1:2*obj.n_wells,1)= q0 / obj.q_scale;

            H = @obj.Coupling;

            options = optimset('MaxFunEvals', 800, 'TolFun', 1e-4, 'TolX', 1e-6);

            % The tubing-head pressure should be positive, so a lower
            % bound is set.
            lb(1:obj.n_wells,1) = 0;
            lb(obj.n_wells+1:2*obj.n_wells) = -Inf;

            % If a lower bound is supplied there should also be an upper
            % bound.
            ub = zeros(2*obj.n_wells,1);
            ub(1:2*obj.n_wells) = Inf;

            x = fmincon(H, y, [], [], [], [], lb, ub, [], options);

            obj.p_th_sol = x(1:obj.n_wells,1) * obj.p_scale;
            obj.q_sol = x(obj.n_wells+1:2*obj.n_wells,1) * obj.q_scale;
            obj.p_bh_sol = zeros(obj.n_wells,1);
            for i = 1:obj.n_wells
                obj.p_bh_sol(i) = obj.subsurface.well(i).wellbore.p_bh;
            end

            % n keeps track of the number of wells where there is no
            % feasible intersection between the IPR and the TPR
```

```
            % in the final iteration
            n = 0;
            for i = 1: obj.n_wells
                if obj.subsurface.well(i).feasible == 0
                    n = n+1;
                end

            end

            % If there are wells whithout a feasible intersection between
            % the IPR and the TPR in the final iteration these will be
            % closed and the resulting system is solved again.
            while n > 0
                D = zeros(n,2);
                k=1;
                for i = 1 : obj.n_wells
                    if  obj.subsurface.well(i).feasible == 0
                        % First we want to close the well with the largest
                        % distance between the IPR and the TPR curve

                        % Distance between the IPR and the TPR curve for
                        % well i
                        x  = [obj.p_bh_sol(i); obj.q_sol(i)];
                        d = obj.subsurface.well(i).BottomHoleSystem(x);

                        D(k,:) = [d,i];
                        k=k+1;
                    end

                end

                % Sort the distances from the different wells in descending
                % order
                D = sortrows(D,-1);

                % c is the number of the well with the biggest distance
                c = D(1,2);

                % Closing a well
                for j = c : obj.n_wells -1
                    obj.subsurface.well(j) = obj.subsurface.well(j+1);
                    obj.surface.inletPipe(j) = obj.surface.inletPipe(j+1);
                end
                obj.n_wells = obj.n_wells -1;
                obj.subsurface.well(obj.subsurface.n_wells)=[];
                obj.subsurface.n_wells = obj.subsurface.n_wells -1;
                obj.surface.n_inlets = obj.surface.n_inlets-1;

                if obj.n_wells == 0
                    error('Error. Number of Wells is returned to zero.')
                end

                %After a well is closed the system is solved again

                %Initial values for p_th and q
                p_th0 = (obj.surface.p_out + obj.subsurface.p_res) / 2.0;
                q0      = 1.0e-5;
                y   = zeros(2*obj.n_wells,1);
                y(1:obj.n_wells,1)= p_th0 / obj.p_scale;
                y(obj.n_wells+1:2*obj.n_wells,1)= q0 / obj.q_scale;

                H = @obj.Coupling;

                options = optimset('MaxFunEvals', 800, 'TolFun', 1e-4, 'TolX', 1e-6);

                % The tubing-head pressure should be positive, so a lower
                % bound is set.
                lb(1:obj.n_wells,1) = 0;
                lb(obj.n_wells+1:2*obj.n_wells) = -Inf;

                % If a lower bound is supplied there should also be an upper
```

```
140                 % bound.
141                 ub = zeros(2*obj.n_wells,1);
142                 ub(1:2*obj.n_wells) = Inf;
143 
144                 x = fmincon(H, y, [], [], [], [], lb, ub, [], options);
145 
146                 obj.p_th_sol = x(1:obj.n_wells,1) * obj.p_scale;
147                 obj.q_sol = x(obj.n_wells+1:2*obj.n_wells,1) * obj.q_scale;
148                 obj.p_bh_sol = zeros(obj.n_wells,1);
149                 for i = 1:obj.n_wells
150                     obj.p_bh_sol(i) = obj.subsurface.well(i).wellbore.p_bh;
151                 end
152 
153                 % n is calculated again to keep track of the number of
154                 % wells where there is no feasible intersection between
155                 % the IPR and the TPR in the final iteration
156                 n = 0;
157                 for i = 1: obj.n_wells
158                     if obj.subsurface.well(i).feasible == 0
159                         n = n+1;
160                     end
161 
162                 end
163 
164             end
165 
166         end
167 
168     end
169 
170     methods (Access = private)
171         function F = Coupling(obj,x)
172             %Coupling determine the Residuals from the SSRF and
173             %the SRF for the given Tubing Head Pressure and Flow
174             %(in x) and returns these in one vector F.
175             %The input vector x consists of n_wells tubing head pressures
176             %and n_wells tubing head volume flows.
177             %The residual vector F consists of n_wells subsurface response
178             %function (SSRF) residuals and n_wells surface response
179             %function (SRF) residuals. alpha = x(1:obj.n_wells);
180             p_th = x(1:obj.n_wells,1) * obj.p_scale;
181             q_th = x(obj.n_wells+1:2*obj.n_wells,1) * obj.q_scale;
182 
183             Fssrf = (q_th - obj.subsurface.SSRF(p_th, obj.p_scale, obj.q_scale)) /...
184                 obj.q_scale;
185             Fsrf  = (p_th - obj.surface.SRF(q_th)) / obj.p_scale;
186             F = [Fssrf; Fsrf];
187             F = 1/2*norm(F,2)^2;
188         end
189 
190     end
191 
192 end
```

E.5.2. Optimization

```
classdef System < handle
    %Production system class.
    %The assumption is that the surface system has exactly the number of
    %inlet pipes as there are wells in the subsurface system.
    %Furthermore, the first pipe is connected to the first well, the second
    %pipe to the second well, etc.

    properties
        surface;
        subsurface;
        n_wells;
        q_sol;
        p_th_sol;
        Delta_p_th_sol;
        alpha_sol;
        p_bh_sol;
        p_scale = 1.e7; % scaled pressure is p/p_scaled, p in [Pa].
        q_scale = 1.e-3; % scaled volume rate is q/q_scaled, q in [m3/s].

    end

    methods
        function obj = System(parFluid, parWellInflow, parWellbore, parPipe, n_wells,...
                n_deep, p_res, p_outlet)
            % Input pressures to be given in [Pa].
            obj.n_wells = n_wells;
            obj.subsurface = Subsurface(parFluid, parWellInflow,...
                parWellbore, n_wells, p_res);
            obj.surface = Surface(parFluid, parPipe, n_wells, n_deep,...
                p_outlet);
        end


        function x = Solve(obj)
            %Solve is the function which actually performs the coupling
            %between the Surface and the Subsurface object. This function
            %determines the Tubing Head Pressures and the Flows.

            %Initial values for alpha.
            alpha = 1e-3;
            y(1:obj.n_wells,1) = alpha;

            H = @obj.Coupling;
            C = @obj.Boundaries;

            options = optimset('MaxFunEvals', 800, 'TolFun', 1e-4, 'TolX', 1e-6);

            % Alpha  has to be between 0 and 1, so a lower and an upper
            % bound is set.
            lb(1:obj.n_wells,1) = 0;
            ub(1:obj.n_wells,1) = 1;


            x = fmincon(H, y, [], [], [], [], lb, ub, C, options);


            obj.alpha_sol = x(1:obj.n_wells);

            [q, p_th] = obj.Conditions(x(1:obj.n_wells));
            obj.q_sol = q;
            obj.p_th_sol = p_th;
            obj.Delta_p_th_sol = p_th - obj.surface.SRF(q);
            obj.p_bh_sol = zeros(obj.n_wells,1);
            for i = 1:obj.n_wells
                obj.p_bh_sol(i) = obj.subsurface.well(i).wellbore.p_bh;
            end

            % n keeps track of the number of wells where there is no
```

```
69              % feasible intersection between the IPR and the TPR
70              % in the final iteration
71              n = 0;
72              for i = 1: obj.n_wells
73                  if obj.subsurface.well(i).feasible == 0
74                      n = n+1;
75                  end
76
77              end
78
79              % If there are wells whithout a feasible intersection between
80              % the IPR and the TPR in the final iteration these will be
81              % closed and the resulting system is solved again.
82              while n > 0
83                  D = zeros(n,2);
84                  k=1;
85                  for i = 1 : obj.n_wells
86                      if  obj.subsurface.well(i).feasible == 0
87                          % First we want to close the well with the largest
88                          % distance between the IPR and the TPR curve
89
90                          % Distance between the IPR and the TPR curve for
91                          % well i
92                          x  = [obj.p_bh_sol(i); obj.q_sol(i)];
93                          d = obj.subsurface.well(i).BottomHoleSystem(x);
94
95                          D(k,:) = [d,i];
96                          k=k+1;
97                      end
98
99                  end
100
101                 % Sort the distances from the different wells in descending
102                 % order
103                 D = sortrows(D,-1);
104
105                 % c is the number of the well with the biggest distance
106                 c = D(1,2);
107
108                 % Closing a well
109                 for j = c : obj.n_wells -1
110                     obj.subsurface.well(j) = obj.subsurface.well(j+1);
111                     obj.surface.inletPipe(j) = obj.surface.inletPipe(j+1);
112                 end
113                 obj.n_wells = obj.n_wells -1;
114                 obj.subsurface.well(obj.subsurface.n_wells)=[];
115                 obj.subsurface.n_wells = obj.subsurface.n_wells -1;
116                 obj.surface.n_inlets = obj.surface.n_inlets-1;
117
118                 if obj.n_wells == 0
119                     error('Error. Number of Wells is returned to zero.')
120                 end
121
122                 %After a well is closed the system is solved again
123
124                 %Initial values for alpha.
125                 alpha = 1e-3;
126                 y(1:obj.n_wells,1) = alpha;
127
128                 H = @obj.Coupling;
129                 C = @obj.Boundaries;
130
131                 options = optimset('MaxFunEvals', 800, 'TolFun', 1e-4, 'TolX', 1e-6);
132
133                 % Alpha  has to be between 0 and 1, so a lower and an upper
134                 % bound is set.
135                 lb(1:obj.n_wells,1) = 0;
136                 ub(1:obj.n_wells,1) = 1;
137
138
139                 x = fmincon(H, y, [], [], [], [], lb, ub, C, options);
```

```
140
141
142                 obj.alpha_sol = x(1:obj.n_wells);
143
144                 [q, p_th] = obj.Conditions(x(1:obj.n_wells));
145                 obj.q_sol = q;
146                 obj.p_th_sol = p_th;
147                 obj.Delta_p_th_sol = p_th - obj.surface.SRF(q);
148                 obj.p_bh_sol = zeros(obj.n_wells,1);
149                 for i = 1:obj.n_wells
150                     obj.p_bh_sol(i) = obj.subsurface.well(i).wellbore.p_bh;
151                 end
152
153                 % n is calculated again to keep track of the number of
154                 % wells where there is no feasible intersection between
155                 % the IPR and the TPR in the final iteration
156                 n = 0;
157                 for i = 1: obj.n_wells
158                     if obj.subsurface.well(i).feasible == 0
159                         n = n+1;
160                     end
161
162                 end
163
164             end
165
166         end
167
168     end
169
170     methods (Access = private)
171         function F = Coupling(obj,x)
172             % Coupling is the function which determines the flow rate of
173             % the current iteration.
174             alpha = x(1:obj.n_wells);
175             q = obj.Conditions(alpha);
176
177             obj.surface.SRF(q);
178             F = -obj.surface.q_out/obj.q_scale;
179         end
180
181         function [C, Ceq] = Boundaries(obj,x)
182             % Boundaries gives the value of the constraints which have
183             % to be satisfied. C gives the value of the constraints which
184             % have to be lower or equal then zero and Ceq gives the value
185             % of the constraints which have to equal to zero, which we do
186             % not have in this case.
187             alpha = x(1:obj.n_wells);
188             q = obj.Conditions(alpha);
189
190             % Qmax is the maximamum set flow rate value for which should
191             % hold that Qout <= Qmax.
192             Qmax = 0.0045;
193
194             obj.surface.SRF(q);
195             C = (obj.surface.q_out-Qmax)/obj.q_scale;
196             Ceq = 0;
197         end
198
199         function [Q,P_th] = Conditions(obj, alpha)
200             % Conditions gives the flow rate and tubing-head pressure
201             % according to the current alpha value.
202
203             % For each choke the different alpha values are set.
204             for i = 1: obj.n_wells
205                 obj.surface.choke(i).alpha = alpha(i);
206             end
207
208             F = @obj.fun;
209
210             % Initial values for the tubing-head pressure and the flow rate
```

```
211             p_th0 =(obj.subsurface.p_res-obj.surface.p_out)/2;
212             q0    =1e-5;
213
214             z(1:obj.n_wells,1) = p_th0/obj.p_scale;
215             z(obj.n_wells+1:2*obj.n_wells,1) = q0/obj.q_scale;
216
217
218             options = optimset('MaxFunEvals', 800,'TolFun', 1e-6, 'TolX', 1e-6);
219
220             x = fsolve(F,z,options);
221
222             % The values of the tubing-head pressure and the flow rate
223             % according to the current value of alpha
224             P_th = x(1:obj.n_wells,1)*obj.p_scale;
225             Q = x(obj.n_wells+1:2*obj.n_wells,1)*obj.q_scale;
226
227         end
228
229         function F = fun(obj, x)
230             % This function gives the residu of the system of equations
231             % that should hold for alpha, the tubing-head pressure and
232             % the flow rate.
233
234             p_th = x(1:obj.n_wells,1)*obj.p_scale;
235             q = x(obj.n_wells + 1: 2*obj.n_wells,1)*obj.q_scale;
236
237             Fsrf = zeros(obj.n_wells,1);
238             p_c = obj.surface.SRF(q);
239
240             for i = 1: obj.n_wells
241                 obj.surface.choke(i).p_c = p_c(i);
242                 Fsrf(i) = (p_th(i) - obj.surface.choke(i).CPR(q(i)))/...
243                     obj.p_scale;
244             end
245                 Fssrf = (q - obj.subsurface.SSRF(p_th, obj.p_scale,...
246                     obj.q_scale))/obj.q_scale;
247
248             F = [Fssrf; Fsrf];
249         end
250
251     end
252
253 end
```

Bibliography

[1] A.M. Ansari, N.D. Sylvester, C. Sarica, O. Shoham, and J.P. Brill. A Comprehensive Mechanistic Model for Upward Two-Phase Floww in Wellbores. 1994.

[2] H.D. Beggs. *Production Optimization Using NODAL analysis*. Tulsa: OGCI and Petroskilss Publications, 2003.

[3] N.H. Chen. An Explicit Equation for Friction Factor in Pipe. 1979.

[4] M.J. Economides, A.D. Hill, and C. Ehlig-Economides. *Petroleum Production Systems*. Prentice Hall, 1994.

[5] G.E. Forsythe, M.A. Malcolm, and C.B. Moler. *Computer Methods For Mathematical Computations*. 1977.

[6] B. Guo and A. Ghalambor. Gas Volume Requrements for Underbalanced Drillling Deviated Holes. 2002.

[7] B. Guo, W.C. Lyons, and A. Ghalambor. *Petroleum Production Engineering*. Gulf Professional Pub., 2007.

[8] MathWorks. Constrained nonlinear optimization algorithms, . URL `http://nl.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html#brnpd5f`.

[9] MathWorks. Unconstrained nonlinear optimization algorithms, . URL `http://nl.mathworks.com/help/optim/ug/unconstrained-nonlinear-optimization-algorithms.html#brnpcye`.

[10] MathWorks. Equation solving algorithms, . URL `http://nl.mathworks.com/help/optim/ug/equation-solving-algorithms.html?refresh=true`.

[11] F.H. Poettmann and P.G. Carpenter. The multiphase flow of gas, oil, and water through vertical strings. 1952.