Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Geoscience and Petroleum

Madelene Skintveit

Recreating Locke-Sawyer Curves with Numerical Reservoir Simulation

Master's thesis in Petroluem Engineering, MTPETR

Supervisor: Curtis Hays Whitson

June 2020

NTNU
Norwegian University of
Science and Technology

Madelene Skintveit

Recreating Locke-Sawyer Curves with Numerical Reservoir Simulation

Master's thesis in Petroluem Engineering, MTPETR
Supervisor: Curtis Hays Whitson
June 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Geoscience and Petroleum

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis is representing the work done in the course ”TPG4920 Petroleum Engineering, Master’s Thesis”, and is a continuation of the course ”TPG4560 Petroleum Engineering, Specialization Project”. The thesis is a requirement for the study program MTPETR at the Norwegian University of Science and Technology (NTNU), to qualify the author for an M.Sc in petroleum engineering.

A widely used tool for reservoir forecasting, reserve estimation, and behavior prediction of the flow of fluids through a porous media, is reservoir simulation. With numerical analysis, the aim is to model the type curves developed by Locke-Sawyer for a well producing single-phase oil with a vertical hydraulic infinite conductivity fracture. Due to the original data used by Locke and Sawyer was not accounted for in their paper, the missing data is generated by Kappa Engineering. Therefore, the purpose of this research is to remake the Locke-Sawyer curves with numerical analysis and compare them to the curves generated by Kappa.

To create a numerical model for the presented problem, it is necessary to know the dimensions of the grid. This has led to challenges as the simulator was limited by a maximum number of total grid blocks and the optimum combination of grids providing the best-estimated results were unknown. Nevertheless, this challenge led to the development of a new approach to grid refinement.

Presented herein is a probabilistic approach to grid refinement. The procedure involves assigning weight factors to the estimated results by examining the value of their derivatives, which makes it possible to separate the good results from the poor ones. The presented methodology is fully automated for estimating rates and cumulative volumes (the results) for a number of time steps, and to determine the optimum gird block combination.

The methodology is tested by comparing the estimated results from the resulting grid block combination with the most accurate numerical solution using the maximum number of grid blocks available. The result from this comparison was an exact match for the simplest fracture penetration ratio of 1, and an adequate match after the 10 first days for the ratio of 1.5. However, this confirms the potential of the presented methodology.

The modeled Locke-Sawyer curves for the fracture ratio of 1 and 1.5 presents a better set of curves than the curves developed by Kappa Engineering. Due to errors in the late rate-time data from Kappa, the part of the curves in the boundary-dominated flow region are unreliable. Consequently, a new set of modeled Locke-Sawyer curves that is perhaps more accurate for unconventional performance forecasting has been developed by the presented methodology.

Sammendrag

Denne masteroppgaven representerer arbeidet gjort i kurset "TPG4920 Petroleumsteknologi, Masteroppgave", og er fortsettelsen på kurset "TPG4560 Petroleumsteknologi, Fordypningsporsjekt". Masteroppgaven er et krav for studieprogrammet MTPETR ved Norges Teknisk Naturvitenskapelige Universitet (NTNU) for å kunne kvalifisere forfatteren med en mastergrad i petroleumsteknologi.

Reservoarsimulering er et mye brukt verktøy for å forstå reservoar prognoser, estimere gjenværende reserver og forutse strømningen av fluider gjennom et porøst media. Målet med oppgaven er å gjenskape kurvene utviklet av Locke and Sawyer for en brønn som produserer enfaset olje ved hydraulisk fakturering, der bruddet har uendelig konduktivitet. Locke og Sawyer utelatte de originale dataene som ble brukt for å skape kurvene. Data som kunne representere kurvene ble derfor generert av Kappa Engineering. Meningen med dette arbeidet er derfor å gjenskape kurvene til Locke-Sawyer med numerisk analyse og sammenligne dem med kurvene fra Kappa.

Det er nødvendig å vite dimensjonene på gitteret for å kunne lage en numerisk modell. Dette har ført til utfordringer ettersom simulatoren er begrenset av et maksimalt antall gitterblokker og den optimale gitter-kombinasjonen som gir det beste estimerte resultatet ikke har vært kjent. Dette har derimot bidra til en ny måte å gjennomføre gitter oppdeling på.

Presentert i oppgaven er en sannsynlig tilnærming til gitter oppdeling. Denne metoden innebærer å tildele vekt-faktorer til de estimerte resultatene ved å undersøke den deriverte for alle mulige gitter kombinasjoner, noe som gjør det mulig å skille de gode og dårlige resultatene fra hverandre. Metoden er fullstendig automatisert, og kan estimere rater og kumulative volumer for et gitt antall tids steg, i tillegg til å bestemme den gitter kombinasjonen som gir det best estimert resultat.

Metoden er testet ved å sammenligne de estimerte resultatene fra den optimale gitter kombinasjonen med den mest nøyaktige numeriske løsningen gitt ved bruk av maksimalt antall tilgjengelige gitterblokker. Resultatene fra denne sammenligningen er en eksakt overlapping for et bruddpenetreringsforhold lik 1, og en tilstrekkelig overlapping etter de 10 første dagene når bruddpenetreringsforholdet er 1.5. Dette resultatet bekrefter derimot potensialet til den presenterte metoden.

De nye modellerte Locke-Sawyer kurvene er angivelig mer nøyaktig en kurvene generert fra Kappa Engineering. Dette skyldes funn av upålitelige data for lave produksjonsrater i datasettet fra Kappa. Med andre ord så er de nye kurvene utviklet fra den presenterte metoden mer nøyaktig for produksjonsprognoser for et ukonvensjonelt reservoar med en vertikal bruddgeometri.

Acknowledgements

I would like to thank Dr. Curtis H. Whitson for his mentorship throughout this spring. I am thankful that I have spent my final year at NTNU working with such a recognized professor who is truly remarkable in his field.

To my co-student Markus Hays Nielsen, I would like to thank you for being my savior countless times, especially with Python. With your help, my programming skills went from non-existing to doable. I am also beyond thankful for the guidance that you have provided me, even though you have your own projects and a thesis to write. I am truly impressed by what you know and what you manage to do.

Thanks to Stian Mydland for helping me to get started with Sensor and to make sure that I understand the tasks at hand. Your technical capabilities within reservoir simulation have been highly appreciated, and thank you for taking the time to run necessary simulations.

Related to my work, I would also like to thank Coates Engineering Inc. for providing me with the license for Sensor6k. The work done in this thesis could not have been done without it.

Special thanks to my family, for their love and support throughout my years of studying Petroleum Engineering at NTNU. To my twin sister Karoline, who also finishes her master's degree this spring, thank you for always listening and providing me the mental support needed.

Last but not least, I want to thank all of my wonderful girlfriends here at NTNU - especially Kristin and Vilde, for sharing my interest in petroleum engineering, and also Hanna, for always brightening up my day at PTS. I am so grateful for looking back at 4 fantastic years here in Trondheim thanks to *Gjengen Allianse* and all of the guys.

Madelene Skintveit
Trondheim,
June 2020

Software

Sensor

Sensor- System for Efficient Numerical Simulation of Oil Recovery is developed by Coats Engineering Inc. and is a pre-processor simulator. Sensor processes the input data given and produces output data for Excel. Sensor has been a simple and powerful asset for providing the information needed.

TecPlot RS

Techplot RS is developed by Tecplot Inc, which provides visualization and analysis software for plotting purposes. Techplot RS helps to manage and analyze large amounts of reservoir simulation data and to understand the reservoir model behavior. The software has been necessary to analyze the output fort61. files from Sensor.

Python

Python is a high level, interpreted, and general-purpose programming language. The use of Python is necessary to create communication between Excel and Sensor in a fast and simple way. It is important to highlight the amount of time saved by introducing Python for data handling and analysis than transferring data between Excel and Sensor manually.

Microsoft Excel

Microsoft Excel is a powerful data visualization and analysis tool. Excel has been particularly valuable for data retrieval using Python and to visualize the generated output files from Python. In addition, many of the tables and figures presented in this work has been created using the software.

Table of Contents

Abstract **i**

Sammendrag **ii**

Acknowledgements **iii**

Software **v**

Table of Contents **viii**

List of Tables **ix**

List of Figures **xii**

1 Introduction **1**
1.1 Scope of Study 1
1.2 Available Data 2

2 Background: Locke-Sawyer Type Curves **3**

3 Basic Theory **7**
3.1 Unconventional Reservoirs 7
3.2 Infinite Conductivity Hydraulic Fracture 7
3.3 Finite-Difference Simulator 8
3.4 Grid Refinement Study 9
3.5 Statistical Distributions 9

4 Grid Description **11**
4.1 The Reservoir Model 11
4.2 1D Grid Description 12
4.2.1 Well Positioned in One Grid Block Along Fracture 12
4.2.2 Well Positioned in All Grid Blocks Along Fracture 13

4.3 2D Grid Description 14

5 Data Initialization **17**
5.1 Condition for Grid Case Generation 17
5.2 Required Input Parameters 17
5.3 Description of Python Processes 18
5.4 Input Datafile to Sensor 18
5.5 Python Code Constraints 20

6 Analysis of 1D Grid Refinement Study **21**
6.1 Determination of Minimum N_y 21

7 Probabilistic Approach To 2D Grid Refinement **25**
7.1 Representation of A Distribution Through Histograms 25
7.2 The Methodology of Weighting Factors 26
7.2.1 Manual Procedure 27
7.2.2 Automated Procedure 28
7.3 Normalization of Derivatives 34
7.4 Threshold Sensitivity Analysis 35
7.5 Weighted Histograms for $x_e/x_f = 1$ 37
7.5.1 Weighted Histogram on Day 1 37
7.5.2 Weighted Histogram on Day 10 39
7.5.3 Weighted Histogram on Day 100 40

8 Application of Weight Factor Methodology to $x_e/x_f = 1.5$ **43**

9 Recreation of Locke-Sawyer Curves with Numerical Reservoir Simulation **47**
9.1 Recreation of $x_e/x_f = 1$ 49
9.2 Recreation of $x_e/x_f = 1.5$ 50

10 Discussion of Results **53**

11 Final Comments and Conclusion **55**

Nomenclature **57**

Bibliography **59**

Appendices **61**
A Figures 63
B Python Codes 78
C Templates 97

List of Tables

4.1 The fracture penetration ratios and the respective fracture lengths 11

5.1 The value of porosity and permeability in the fracture and matrix 19

6.1 Estimated oil rate and number of N_y for relative change in rate of 1, 0.1 and 0.001 % 22

7.1 General frequency and relative frequency of weighted and un-weighted data when using method 1 30

9.1 Variables used in Locke-Sawyer dimensionless equations 48
9.2 Undersaturated black oil table and oil compressibility calculation 49

List of Figures

2.1 Decline curves for well producing single-phase oil with vertical infinite conductivity fracture developed by Locke-Sawyer 5

4.1 The reservoir model consisting of 4 symmetric squares 12
4.2 Hydraulic fracture with a centered well in only one grid block along the fracture 13
4.3 Well connected to all grid blocks along fracture 14
4.4 Example design of a hydraulic fracture for 2D grid 14

6.1 Estimated oil rate vs. N_y on day 1 22
6.2 Logarithmic plot of relative change in estimated rate vs. N_y 23

7.1 Original oil rate histogram for $N_{max} = 2000$ on day 10, when $x_e/x_f = 1$. 26
7.2 Possible combinations of N_x and N_x that do not exceed $N_{max} = 20$ 27
7.3 Original oil rate histogram for $N_{max} = 20$ on day 10 27
7.4 The estimated oil rates given for all possible cases of $N_{max} = 20$ 28
7.5 The value of the derivatives ($\Delta d_{i,j}$), after utilizing Eq.7.1 on the estimated rates 29
7.6 The continuous weight factors calculated by Eq.7.2 29
7.7 Weighted oil rate histogram for $N_{max} = 20$ with method 1 on day 10 . . . 31
7.8 The binary weight factors assigned to the estimated rates when $\varepsilon = 0.332$ 32
7.9 Weighted oil rate histogram for $N_{max} = 20$ with method 2 on day 10 . . . 33
7.10 Estimated oil rate vs. time for different threshold values 35
7.11 Estimated oil rate and number of cases used to get the estimated rate for different threshold values on day 1 36
7.12 Original oil rate histogram for $N_{max} = 2000$ on day 38
7.13 Weighted oil rate histogram for $N_{max} = 2000$ on day 1 with $\varepsilon = 10^{-6}$ . . . 38
7.14 Weighted oil rate histogram for $N_{max} = 2000$ on day 10 with $\varepsilon = 10^{-6}$ . . 39
7.15 Original oil rate histogram for $N_{max} = 2000$ on day 100 40
7.16 Weighted oil rate histogram for $N_{max} = 2000$ on day 100 with $\varepsilon = 10^{-6}$. 41

8.1 Estimated oil rate vs. time from different combinations of N_x and N_y that sums up to 6000 grid blocks 44
8.2 The accurate numerical solution with 6000 grid cells and best estimated result for N_{max} = 1000 with N_x = 9 and N_x = 46 45

9.1 The x_e/x_f = 1 curve generated by Kappa and model 50
9.2 The x_e/x_f = 1.5 curve generated by Kappa and model 51

A.1 Flowchart of Python code process in main.py 63
A.2 Flowcharts of codes used in main.py 64
A.3 Flowchart of Python code process in main_extracter.py 65
A.4 Flowcharts of codes used in main_extract.py 65
A.5 Flowchart of Python code process in main_extracter.py 66
A.6 Flowchart of codes used in main_extracter.py 67
A.7 Estimated oil rate and cumulative oil volume for PI=1 STB/d/psi 68
A.8 Estimated oil rate and cumulative oil volume for PI=10 STB/d/psi 69
A.9 Estimated oil rate and cumulative oil volume for PI=100 STB/d/psi 70
A.10 Estimated oil rate and cumulative oil volume from 1D grid refinement study 71
A.11 Estimated oil rate vs. N_y for N_x = 1 and N_y = 6000 72
A.12 Estimated cumulative oil volume vs. N_y for N_x = 1 and N_y = 6000 73
A.13 Estimated oil rate vs. time for different threshold for the first 10 days . . . 74
A.14 Estimated oil rate vs. time for different threshold values from day 70 to 100 75
A.15 Estimated oil rate and number of cases used to get the estimated rate for different threshold values on day 10 76
A.16 Estimated oil rate and number of cases used to get the estimated rate for different threshold values on day 100 77

Chapter 1

Introduction

1.1 Scope of Study

A reservoir engineering task of significant importance is production performance forecasting or well performance analysis. To be able to forecast future production of a well or a field is crucial throughout its lifetime, and the reasons are many. Production forecasting is the basis for any development decision and helps to decide among several development concepts. It is a valuable tool for determining the stage of a field's life, defining the best recovery mechanism when the production needs to be enhanced, estimating remaining reserves and estimate ultimate recovery (EUR). Well performance analysis can roughly be diagnosed from 3 angles; decline curve analysis (DCA), analytical modeling/ rate transient analysis (RTA) and numerical reservoir simulation (NRS).

In the specialization project, the aim was to recreate the Locke-Sawyer type curves for a vertical infinite-conductivity fracture without including the transition period from infinite-acting to boundary dominated flow. To achieve this goal, DCA/RTA was utilized by applying the methodology of Fetkovich type curves (see description in Chapter 2). Despite being fast and simple to use, the concepts of DCA and RTA are not as robust as numerical analysis. Therefore, the main objective of this study is to remake the Locke-Sawyer curves with numerical reservoir simulation. Even though NRS is a far more time-consuming and requires significantly more computational work, it is a general method that is going to provide the best solution when there is no exact analytical solution to compare with.

The main challenge when implementing this problem to a reservoir simulator is to determine the appropriate grid. A probabilistic approach to grid refinement is presented, which is a methodology for estimating a "true" solution for a reservoir model with an unknown behavior using a limited number of grid blocks. The underlying idea is based on the statistical behavior of all possible combinations of grid blocks that do not exceed the maximum number of total grid blocks. The ideal solution is an automated procedure for estimating rates and cumulative volumes over the production lifetime. This is done by applying a

weight factor for the results, that is either continuous between 0 and 1, or a binary of 0 and 1. Both of these methods are described in Chapter 7 and the best choice of method for further analysis is explained also. These weighting factors should isolate the "true" results, however, seeing as this information is unknown, a method is proposed to estimate these values by using the derivatives of the possible results. This derivative is expressing the change in rate with respect to the x-and y-direction for a specific grid block, and is an important term that plays a significant role when estimating the rates needed to recreate the Locke-Sawyer curves.

Two analysis is going to take place in this study, which is one-dimensional (1D) and two-dimensional (2D) grid refinement. What determines if the grid refinement is 1D or 2D, is if the hydraulic fracture is extending to the reservoir boundary or not. If the fracture does extend to the reservoir boundary, the main flow direction towards the fracture is in the y-direction, or in other words, not along the fracture in the x-direction. Hence, from 1D grid refinement, the minimum number of grid blocks away from the fracture is to be determined. This determines the necessary number of grid blocks needed for the 2D grid refinement analysis, where the method of weighting factor is applied. To do so, a threshold value that narrows down the possible grid block combinations is introduced, which consequently requires a threshold sensitivity analysis to decide the optimum value. The optimum threshold value is then going to be used to provide the best-estimated results.

After acquiring the estimated rates given by the probabilistic grid refinement study, it is necessary to transform the rates to dimensionless rates (q_D) and time to dimensionless time (t_D), to be able to compare with the Locke-Sawyer curves developed by Kappa Engineering.

The work presented is divided into 4 main parts. The first part consists of an introduction to the problem by including the background of the specialization project and other essential theory concepts. The second part is about describing the grid for different fracture penetration ratios and a description of the data initialization by combining Sensor, Python, and Excel. The third part provides the introduction and the result of applying the methodology of weighted factors, and the recreation of the Locke-Sawyer curves. Finally, the last and fourth part contains the key results and discussion of these findings.

1.2 Available Data

The Locke-Sawyer curves used in this work are generated by Leif Larsen in Kappa engineering [1]. The reason for this is because the original data used to create the curves was not presented in the 1975 paper by C.D Locke and W.K Sawyer, and the data has not been found in other research work or presented papers.

Chapter 2

Background: Locke-Sawyer Type Curves

The information provided in this chapter is taken directly from TPG4560 specialization project [2]. The reason for including this information is to provide a quick summary of the project, and to introduce the concepts discussed herein.

Decline curve analysis (DCA) is an old, yet fast and simple technique for predicting future performance of a well or a field. The reason for being one of the most preferred methods is due to requiring only the production data, which is the most available characteristic of a well during depletion. The future production is forecasted by simply extrapolating the production rates with time, a concept introduced by J.J Arps [3]. He recognized that two main fluid flow regimes could occur during the depletion process of a reservoir. The first regime is occurring immediately after the start of production when no outer boundary has yet been felt by the pressure distribution. In this period, the reservoir seems infinite in size, and the period is therefore called *infinite acting* (IA). This period is characterized by very high decline rates. Once the pressure transient reaches one of the outer boundaries, the flow regime changes into *boundary dominated* (BD) flow, and the reservoir pressure starts to decline.

Arps came up with three types of decline curves; hyperbolic, exponential, and harmonic:

$$\text{Hyperbolic}: \quad q(t) = \frac{q_i}{\left[1 + bD_i t\right]^{\frac{1}{b}}} \quad , 0 < b < 1 \tag{2.1}$$

$$\text{Exponential}: \quad q(t) = \frac{q_i}{e^{D_i t}} \quad , b = 0 \tag{2.2}$$

$$\text{Harmonic}: \quad q(t) = \frac{q_i}{\left[1 + D_i t\right]} \quad , b = 1 \tag{2.3}$$

where q_i is the initial rate, D is the decline constant, and b is the hyperbolic decline constant. These empirical equations are only valid for constant flowing bottom hole pressure, i.e when the reservoir boundaries have been felt. The type of decline is dependent on the value of the hyperbolic decline exponent, which indicates the recovery efficiency of the field. Exponential decline, denoted by $b \rightarrow 0$, indicates a low recovery and fluid expansion as the recovery mechanism. This means that either single-phase gas flow at high pressures or single-phase oil flow is represented when $b = 0$. It is the latter case that applies to the work done in the specialization project. Further details about the other possible values of b are described in the specialization project.

The challenge with low-permeability reservoirs (unconventionals) is that the period from reaching one boundary to all of the outer boundaries, so-called the *transition* period, could last for months to decades! This result is a b value much greater than 1 when the Arps method is applied, which consequently leads to significant overestimation of the estimated ultimate recovery (EUR) and of the remaining reserves in the reservoir. In other words, the major limitation of Arps empirical equations is that they only take into account production during constant bottom hole flowing pressure, which is a highly unrealistic operating condition for unconventional reservoirs.

M.J Fetkovich introduced a significant improvement in the field of DCA. Fetkovich combined the analytical solutions for the IA period with Arps empirical equations for BD flow, which made it possible to describe the production performance over the lifetime of the well [4]. The new curves were plotted with dimensionless decline rate and time on the axes, q_{Dd}, and t_{Dd} respectively. By introducing these new dimensionless terms, Fetkovich gave Arps' constants b, D, and q_i physical meaning in terms of reservoir properties and depletion performance. Regarding the challenge of maintaining constant flowing pressure conditions for low-permeability reservoirs, the Fetkovich type curves do indeed provide a solution. Changes in operating conditions have been taking into account by utilizing the concepts of rate normalization and the principle of superposition. To be able to deal with varying pressures and rates marks the start of modernizing the traditional DCA towards rate transient analysis (RTA). In other words, Fetkovich type curves are applicable for the whole depletion process for a low-permeability reservoir.

C.D Locke and W.K.Sawyer presented decline type curves for a well producing at constant bottomhole pressure; single-phase oil with a vertical infinite conductivity hydraulic fracture [5]. How these curves were created is provided in the specialization project, and the original curves are shown below in **Fig.2.1**.

The length of the distance from the well to the external boundary (x_e) compared to the length of the hydraulic fracture (x_f), is expressed in the fracture penetration ratio x_e/x_f. The line denoted with a ∞ - symbol is representing an infinitely large reservoir compared to its hydraulic fracture. All the other lines deviate from the IA line, and it happens at earlier times for smaller fracture penetration ratios. The reason for this is because the pressure transient uses a shorter amount of time to reach the outer boundaries for a smaller reservoir. In Fig.2.1 it is noticeable that the curves change shape sometime after the deviation,

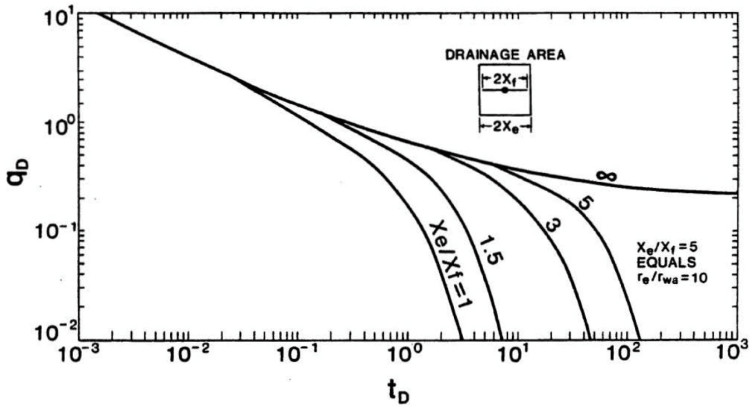


Figure 2.1: Decline curves for well producing single-phase oil with vertical infinite conductivity fracture developed by Locke-Sawyer

from a straight line to a concave downward shape. The latter is the shape of an exponential decline which indicates that the outer boundaries have been reached by the pressure transient. However, how long the transition period lasts before an exponential decline occurs is not possible to determine by the eye. As mentioned in the specialization project, it is the transition period that is of the highest interest as the period has many uncertainties related to it, especially when estimating EUR and remaining reserves of a field. In the paper by Locke and Sawyer, the original data was not included, which makes the accuracy of the curves doubtful.

In the specialization project, by applying the methodology of Fetkovich dimensionless $q_D(t_D)$ plot, the uncertain transition period was removed from the Locke-Sawyer type curve. Consequently, the Locke-Sawyer curve was transformed into a type curve for a fractured well geometry that could be more accurate for shale performance forecasting. In this study, however, the idea is to recreate the original curves in Fig.2.1, which was not possible to do in the specialization project without a reservoir simulator.

Chapter 3

Basic Theory

Description of theory and/or analytical concepts in this work. Some of the information in Section 3.1 and 3.2 is discussed in the specialization project

3.1 Unconventional Reservoirs

The term ”unconventional reservoir” is a general term for reservoirs characterized by very low permeability (10^{-5} – 10^{-2} mD) and porosity($< 10\%$), as tight-and shale reservoirs, [6]. As a consequence of these reservoir properties, achieving economical production is not possible without special recovery operations. In other words, production from unconventionals requires more specialized technologies that are more complex and far more expansive than utilizing common methods for conventional reservoirs. The most popular recovery technique for shale oil, shale gas, and tight gas is hydraulic fracturing which creates cracks in the reservoir that the reservoir fluid can flow through. Another challenging aspect with unconventional reservoirs is to predict the well performance, which is significantly more difficult because of the reservoir properties already mentioned and because of the fluid flow behavior described in Chapter 2. Therefore, to reduce the uncertainty in production forecasting by a reservoir simulator, it is necessary to create a numerical model with an accurate hydraulic fracture.

3.2 Infinite Conductivity Hydraulic Fracture

The determinants of the productivity of a low-permeability reservoir are the properties of the hydraulic fractures [7]. Especially the conductivity of the hydraulic fracture has a significant impact on productivity. In this work, infinite conductivity fracture (ICF) is the fracture type being used. Infinite conductivity means that there exists no pressure drop in the fracture during production, and the fluid flow into the fracture is the same everywhere along the fracture. To achieve this in real life is unrealistic, however, it is possible to satisfy this condition in reservoir simulation, which is explained in Chapter 4.

3.3 Finite-Difference Simulator

The fundamental equation in well testing is the hydraulic diffusivity equation, which determines how fast pressure signals moves through the reservoir [8]. The starting point of deriving the diffusivity equation is with the continuity equation for single phase flow:

$$\frac{\partial p}{\partial t} + \nabla(\rho q) = 0 \tag{3.1}$$

The continuity equation is an expression for conservation of mass in a volume element. By combining the continuity equation with Darcy's law which describes the flow of fluids through a porous media:

$$q = -\frac{k}{\mu}\nabla p \tag{3.2}$$

then the outcome is the diffusivity equation, assuming one dimensional and incompressible flow:

$$\frac{k}{\mu\phi c_t}\nabla^2 p = \frac{\partial p}{\partial t} \tag{3.3}$$

The quantity $\frac{k}{\mu\phi c_t} = \eta$ is called the hydraulic diffusivity. This equation is a partial differential equation (PDE), and PDEs are used to describe a variety of phenomena by formulating problems that involve functions of several variables. These equations cannot be solved analytically, they must be solved with numerical analysis. The main approach of solving PDEs in numerical reservoir simulation is by the use of finite-difference methods, which are a set of algebraic schemes that one can derive to approximate the PDE. It is necessary to convert the system of PDE into a system of algebraic equations, which is done by applying the Taylor series expansion for approximating the derivatives. The discretization error that occurs by using Taylor series expansion is the error that occurs when a continuous function is discretized, moreover, it is the difference between the real solution of the PDE and the solution of the discrete problem. This discretization error is proportional to the grid block size, which means that a smaller the grid block size results in a smaller error!

When performing the analysis of the finite-difference method for the numerical solution of the PDE, Lax-Richtmyer Equivalence Theorem is often used. This theorem is often called the fundamental theorem of numerical analysis, and links together the terms consistency, stability, and convergence of a finite-difference approximation:

$$\text{Convergence} \iff \text{Consistency} + \text{Stability}$$

"A finite-difference approximation that is consistent and stable is also convergent"[9]. *Stability* of a finite-difference approximation is achieved when inherent and round-off errors are not amplified for larger time steps. This is ensured in the numerical model by using the fully implicit scheme. This scheme requires significantly more computational work (larger CPU time and storage requirement) as an iterative procedure but enables a larger time step. A finite-difference approximation is *consistent* if by reducing the time step and the derivative in x-and y-direction ($\Delta d_{i,j}$), the discretization error approaches zero. Therefore, if a finite-difference approximation for a properly posed PDE is both consistent and stable, then it satisfies the necessary conditions for convergence.

3.4 Grid Refinement Study

”The aim of gridding in reservoir simulation is to turn the geological model of the field into a discrete system on which the fluid flow equation can be solved”, [10]. These discrete systems are referred to as grid blocks. To determine the minimum number of grid blocks needed in a reservoir model is an important task for a reservoir engineer because the wrong number of grid blocks might lead to inaccurate results or longer CPU time,[11]. A smaller grid block size should result in a smaller discretization error for a consistent method. However, this does not mean that multiple grid blocks should be used, as CPU time increases drastically with an increasing number of grid blocks!

In other words, deciding the grid for a reservoir simulation model is not a straightforward procedure. There exist two related problems when deciding the grid for a reservoir simulation model; first is that the accuracy of the numerical simulation is directly related to the number of grid blocks used, and second is the discretization error. The finer the grid (reducing the size of the grid blocks), the better accuracy and smaller is the discretization error! A grid refinement study must be conducted to determine the minimum number of grid blocks required to achieve a converged solution in both x-and y-direction (N_x, N_y). To reach a converged solution means that little change is seen in reservoir model performance when increasing the number of grid blocks. For Sensor6k, the maximum number of grid cells available is 6000, and to use all grid cells would result in the most accurate numerical solution. However, the aim is to decrease the number of grid blocks to a number that provides an estimated solution that is within line-thickness of the true solution with 6000 grid cells.

3.5 Statistical Distributions

The distribution of a statistical data set is a function showing all the values or intervals of data and how often they occur. When organizing a numerical data set, they are often ordered from smallest to largest and divided into appropriate sized groups before being put into charts and graphs to examine, for example, the center, shape and the variability in the distribution. Histograms are a way of graphically representing a distribution, and histograms are used extensively throughout this study. The distributions can have several different shapes other than a normal distribution. In this study, most of the distributions are skewed and not unimodal, which means that the distribution is non-symmetric. The challenge with a skewed distribution is when deciding the ”typical” value of distribution, or identifying the central location in the data set. There are especially two metrics used for when measuring the central tendency, which is the mean and the median.

In asymmetric distribution, these terms would be identical, but for skewed distributions, they could be fairly different. Moreover, it is important to point out that both the mean and the median are valid measures of central tendencies, but they are appropriate under different conditions. The mean is probably the most common method of central tendency, where the sum of all the data in the dataset is divided by the number of values in the data set. However, the mean has two disadvantages; it is highly susceptible to the influence

of outliers, values being particularly small or large compared to the rest of the data set. The second disadvantage is that it loses its ability to measure the central location when the data is skewed. In this case, the median could be a better measure, as it is less affected by skewed data and outliers. The median is defined as the number in the center of a given data set ranged from the lowest to the highest value. Therefore, the median is used when estimating the typical value of a distribution.

Chapter 4

Grid Description

4.1 The Reservoir Model

The reservoir model is a square, which is the same representation of the reservoir used by Locke-Sawyer. The square, ($x_e = y_e$), has been divided into 4 equal parts as seen in **Fig.4.1**. Only 1 part is modeled in the simulator for simplicity, however, it is important to scale up the rates by 4 when recreating the Locke-Sawyer curves in Chapter 9. The reservoir half-length (x_e) is set to be 400 ft to keep the model dimensions constant, while the fracture length (x_f) is changed accordingly to the x_e/x_f - ratio, see **Table 4.1**.

Table 4.1: The fracture penetration ratios and the respective fracture lengths

x_e/x_f	x_f
1	400
1.5	266.67
3	133.33
5	80

The model consists of a fracture cell, where the hydraulic fracture is applied, and matrix cells. This means that 2 is the minimum number of grid cells required in the y-direction. The fracture cell width, w_f, is set to be 0.1 ft, and consequently, the reservoir length in the y-direction is 401 ft. **Fig. 4.2**, **Fig.4.3** and **Fig.4.4** displayed in this chapter includes the fracture cell displayed in blue color.

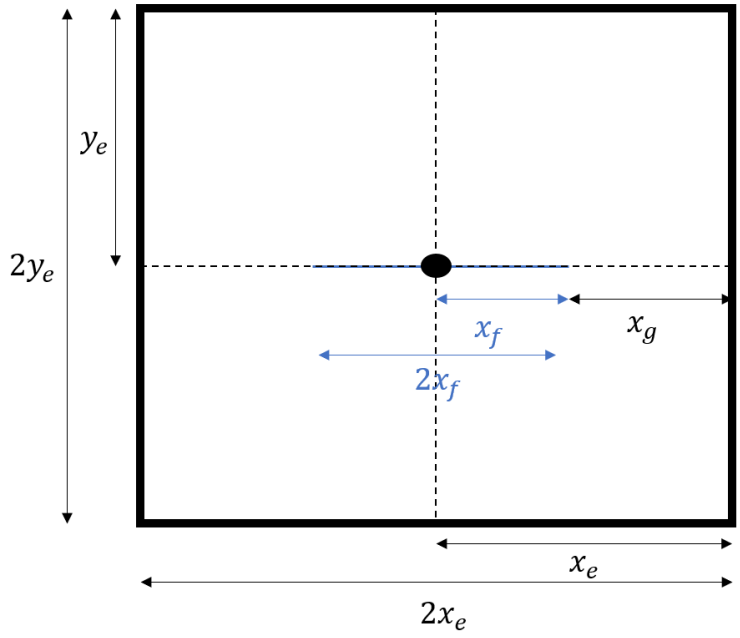


Figure 4.1: The reservoir model consisting of 4 symmetric squares

4.2 1D Grid Description

4.2.1 Well Positioned in One Grid Block Along Fracture

The simplest case is when the hydraulic fracture fully penetrates the reservoir, i.e when $x_e/x_f = 1$. When the hydraulic fracture reaches the reservoir boundary, the flow into the fracture is one-dimensional (1D), hence flow is only from the y-direction. This means that only the number of grid blocks in the y-direction has an impact on the result, as the same amount of fluid is flowing into the fracture regardless of 1 or 100 grid blocks used in the x-direction.

The following has been decided for the one-dimensional grid case:

- The reservoir half-length in the x-direction (x_e) is constant throughout the study. This value is set to be 400 ft.
- The fracture half-length (x_f) is equal to x_e

Sensor is compatible with any grid type or combination of grid types. For simplicity, regarding the numerical dispersion that occurs in reservoir simulation, uniform cartesian gridding is used. This means that all grid blocks have the same size. In addition, Sensor operates with block-centered grids, which means the grid points are in the center of their grid blocks. Therefore, when examining only one square, the well in Fig.4.1 is not placed

in the bottom-left corner of the upper right square, but as seen in Fig.4.2, where the grey circle illustrates the well. In Fig 4.2, the grid consists of $N_x = 3$ and $N_y = 5$.

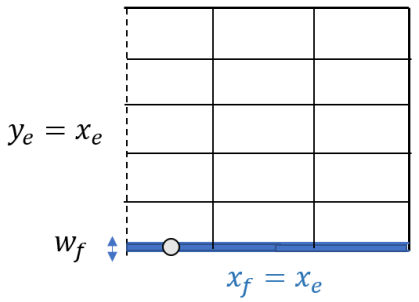


Figure 4.2: Hydraulic fracture with a centered well in only one grid block along the fracture

The hydraulic fracture modeled in this work is an infinite-conductivity fracture (ICF). However, as long as the well is present in only one of the grid blocks along the fracture, there exists a pressure drop inside the fracture. To avoid this pressure drop and keep the pressure constant throughout the fracture, which is the case for an ICF, the well is connected to all grid blocks along the fracture in the x-direction. This is further described in the next section.

The formula for uniform gridding, or the size of each grid block, in x and y-direction is:

$$\Delta x = \frac{x_f}{N_x} = \frac{x_e}{N_x} \tag{4.1}$$

$$\Delta y = \frac{y_e}{N_y} \tag{4.2}$$

and the total amount of grid blocks is given by:

$$N_{tot} = N_x \cdot N_y \tag{4.3}$$

4.2.2 Well Positioned in All Grid Blocks Along Fracture

By using the ”connected well argument”, Fig.4.2 is changed to Fig.4.3. Consequently, the same amount of fluid is flowing into the fracture grid blocks. The major simplification of applying the connected well argument is that the permeability is left unchanged and there is no need to start tweaking the permeability in order to assure zero pressure drop in the hydraulic fracture!

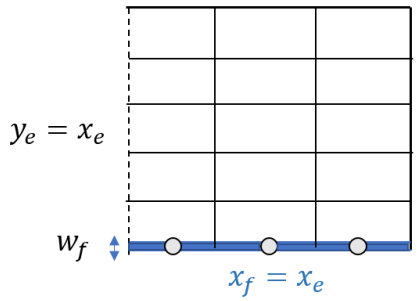


Figure 4.3: Well connected to all grid blocks along fracture

4.3 2D Grid Description

For the case of $x_e > x_f$, the following is taken into consideration:

- The fracture half-length (x_f) is calculated from the relation of $x_e/x_f = 1.5$, since $x_f = f(x_e/x_f)$
- Connected well argument is used for all grid blocks along the fracture.

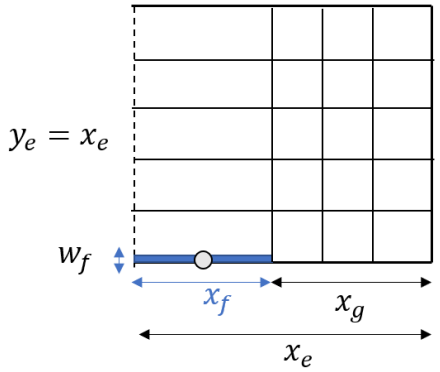


Figure 4.4: Example design of a hydraulic fracture for 2D grid

When the hydraulic fracture does not extend all the way to the reservoir boundary, the flow into the fracture is two-dimensional (2D) with a flow from the end of fracture in the x-direction, in addition to the flow from the y-direction. In Fig.4.4, cartesian gridding is used for the grid blocks along the hydraulic fracture (x_f) and for the remaining length until reservoir boundary in x-direction (x_g). In Fig. 4.4, $N_{xf} = 1$ and $N_{xg} = 3$.

The following relations must be taken into account:

$$x_e = x_f + x_g \tag{4.4}$$

$$N_x = N_{xf} + N_{xg} \tag{4.5}$$

$$N_{tot} = (N_{xf} + N_{xg}) \cdot N_y \tag{4.6}$$

The goal is to determine the minimum number of grid blocks required (N_x and N_y) for $x_e/x_f = 1$ and 1.5. However, to determine the grid block combination for the fracture penetration ratio of 1.5 is more complex compared to the fracture penetration ratio of 1, since it is necessary to determine the optimum number of N_{xf} and N_{xg}, in addition to N_y. The following is considered:

- N_y is the most important variable as flow from the y-direction into the fracture is always present.
- N_{xg} is the second most important variable as it becomes stronger with increasing x_e/x_f- ratio (contributes with more flow into fracture).
- N_{xf} is the least important variable as the fracture length becomes smaller with increasing x_e/x_f- ratio.

In other words, it is considered that the most accurate numerical solution is achieved when $N_y > N_{xf} > N_{xg}$.

With Sensor6k, the maximum amount of grid blocks is 6000. Consequently, endless combinations of grid blocks are possible! However, to run all these combinations cannot be done manually by changing the grid properties in the Sensor input data file. To automate procedures and to be time-efficient are key factors for any engineer, and therefore, Python is utilized to generate the possible grid case combinations, to write the corresponding input data files, and to save required information to Excel. This is elaborated further in the next chapter.

Chapter 5

Data Initialization

5.1 Condition for Grid Case Generation

Before conducting the 1D and 2D grid refinement study, it is necessary to generate the grid cases the refinement study will be applied to. As a part of the problem description, there exists a fixed limit of the maximum number of grid blocks in the model, N_{max}. Assuming a box-model reservoir, the total number of grid blocks is given by:

$$N_{tot} = N_x \cdot N_y \tag{5.1}$$

where

$$N_y \leq N_{tot} \tag{5.2}$$

and

$$N_x \leq \left| \frac{N_{tot}}{N_y} \right| \tag{5.3}$$

Only the combinations of N_x and N_y that do not exceed the condition: $N_{tot} \leq N_{max}$ is considered.

5.2 Required Input Parameters

The code *Read_Parameters* in Appendix B.2, is used to extract the values of the following parameters, from the excel sheet ”Parameters”:

1. N_{max} - maximum number of grid blocks in model (6000 grids is the absolute maximum with Sensor6k license)
2. x_e - reservoir length in x-direction, constant to 400 ft
3. x_f - fracture length, dependent on:
4. x_e/x_f - fracture penetration ratio

5. y_e - reservoir length in y-direction, constant to 400 ft

6. n_{max} - maximum number of combinations of N_{xf} and N_{xg} for a given value of N_x, when $x_e/x_f \neq 1$.

When changing the fracture penetration ratio, the fracture length changes automatically. Furthermore, it is only necessary to determine the value of N_{max}, x_e/x_f and n_{max}, as x_e and y_e are constant. These are the only input parameters that are required for further work in this study, the rest of the variables can be generated from those above. The meaning of the variable n_{max} is to avoid over-representation of some larger N_x-values that could have many combinations of N_{xf} and N_{xg}. For example, if $N_x = 10$ and $n_{max} = 2$, then only 2 combinations of N_{xf} and N_{xg} that gives $N_x = 10$ is given.

5.3 Description of Python Processes

All codes are presented in Appendix B, while flowcharts of how the codes correlate to each other and the process overview are displayed in Appendix A.1. There are main 3 processes, *main.py* given in Appendix B.1, *main_extract.py* in Appendix B.6 and *main_functions.py* in Appendix B.9. Following is a short description of these processes:

- main.py - the process of generating the possible cases that do not exceed the condition: $N_{tot} = N_x \cdot N_y \leq N_{max}$. For each possible case, include files containing the dimension vectors, which give the step length of each grid block in x-and y-direction, and an input data file is created. This makes it possible to run simulations in Sensor for all possible cases and to extract information about rates, cumulative rates, and pressures at different time steps as a result of that specific combination of N_x and N_y.

- main_exctracter.py - the process of extracting required information from the simulation runs mentioned above and to save the information to Excel files for further analyzing. The Python code for applying weighting factors to $x_e/x_f = 1$ is displayed in Appendix B.8, and this methodology is further explained in Section 7.2.2.

- main_functions.py - the process of creating 3D tensor containing the possible combinations of N_{xf}, N_{xg} and N_y, and to apply the weighting factors methodology to $x_e/x_f = 1.5$ presented in Appendix B.11.

5.4 Input Datafile to Sensor

To run simulations in Sensor, it is necessary to generate the input datafile, which is given the name *run_simulation.dat*. The information provided in this datafile can be divided into two parts:

1. Information that is constant regardless of case number (grid combination)

2. Information that is dependent on the case number

The constant information is saved as templates, which are presented in Appendix C. To ensure single phase flow, the bottom hole pressure (BHP) must be higher than the saturation pressure, which is 1984.96 psia. Therefore, BHP = 2500 psia in **Template C.5**. The target rate is set to be 100 000 STB/d to reassure that the well produce under constant bottomhole pressure condition, the same condition used by Locke-Sawyer. The time range is set to be 100 days, where day 1, 10 and 100 are used frequently when analysing and comparing results. When running simulations for the whole production period, the time must be significantly larger than 100 days. This is further described in Chapter 9.

The information that changes with grid combination is coded manually in Python, see Appendix B.4. This applies to:

- Grid properties - combination of N_x and N_x.
- Model properties - the include files DELX.inc and DELY.inc containing the grid block dimensions in x- and y-direction.
- Reservoir properties like porosity and permeability
- Well properties like well placement and well productivity index (PI)

The MOD keyword in the code *Write_Datafile* in Appendix B.4 is used to regionally alter the properties mentioned above. These properties are fracture dependent, which means that the value given for these properties are only applicable in the fracture grid blocks. The six integers I1 - K2, define a portion of the grid that can be altered, where I symbolizes the x-direction, J the y-direction, and K the z-direction, and index 1 means start and 2 means end. The only integer that is to be changed is I2, which is the number of N_{xf}, as the properties should only be applicable for fracture grid blocks. All the other integers should be 1, since the fracture is located in the first grid block in the y-direction (J1, J2), and the model consists of 1 layer in the z-direction (K1, K2). The matrix and fracture properties are summed up in **Table 5.1**:

Table 5.1: The value of porosity and permeability in the fracture and matrix

Property	Matrix	Fracture
Permeability (mD)	200E-06	100
Porosity (-)	0.05	0.1

In the presented table, the matrix properties and especially permeability is significantly smaller than for the fracture. This is because the matrix is representing shale, with properties explained in Section 3.1.

Another property that needs to be determined is the well productivity index, which expresses the well's ability to produce fluid from the reservoir. When running simulations for Nmax = 1000 for $x_e/x_f = 1$ at day 10, where both N_x and N_y vary to 1000 grid blocks, huge oscillations were present in the oil rate and cumulative oil volume for PI = 100 STB/d/psi. Therefore, a PI-sensitivity analysis was performed, where the PI was lowered from 100 to 10 and 1 STB/d/psi. The PI is defined as:

$$PI = \frac{q}{\Delta P} \tag{5.4}$$

where q is the oil rate in STB/d and the pressure difference in psi is between the grid cells and the wellbore: $\Delta P = PGRID - PBH$. The behavior of the oil rate and the cumulative oil volume when changing N_x and N_y, and decreasing the PI is presented in Appendix A.2. It is observed that oscillations occur in the oil rate and cumulative oil volume when the productivity index is high and this results in instability issues in the model. Therefore, the PI is set equal to 1 STB/d/psi in further work to avoid this.

5.5 Python Code Constraints

Some constraints are applied to the Python codes in Appendix B:

- If $x_e/x_f \neq 1$, then $N_x \neq 1$ since $N_x = N_{xf} + N_{xg}$
- If $x_e/x_f = 1$, then $N_{xg} = 0$ since the fracture reaches the reservoir boundary
- If $x_e/x_f = 1$, then $N_{xg} = 0$ and consequently the step length outside fracture is zero, $d_{xg} = 0$

Chapter 6

Analysis of 1D Grid Refinement Study

6.1 Determination of Minimum N_y

For x_e/x_f = 1, the number of grid blocks in x-direction should have no impact on a converged solution with a sufficient number of grid blocks in the y-direction, as mentioned in Section 4.2.1. When the hydraulic fracture is reaching the reservoir boundary, to use 1 or 100 grid blocks for N_x is insignificant. Therefore, to decide the optimum number of N_y is to be determined. The following has been decided:

- Set N_{max} = 6000
- Keep N_x = 1
- Let N_y vary between 2 and 6000 grid blocks.

The most accurate numerical solution or the ”true” solution to the 1D grid refinement study is when the maximum number of available grid blocks are used in the y-direction, i.e when N_y = 6000. The behavior of the oil rate and the cumulative oil with time for N_x = 1 and N_y = 6000 is presented in **Fig.A.10** in Appendix A.3. It is desired to determine the minimum N_y that provides an estimated solution within line-thickness of the true solution in **Fig.A.10(a)**

Fig.A.11 and **Fig.A.12** in Appendix A.3, display the oil rate and the cumulative oil volume on day 10 and 100 for varying N_y. It is observed that both the oil rate and the cumulative oil volume stabilize at some value of N_y and further increasing of N_y has no effect. The peak in the figures is occurring for an unknown reason, however, the oil rate stabilizes at around 10.74 STB/d on day 10 and for 3.29 STB/d on day 100. An important aspect to point out is the stabilization happens earlier for later times, which is represented by a more narrow peak on day 100. In other words, this means that the number of grid blocks

needed to achieve a stabilized rate is lower on day 100 then on day 10. This behavior sets the restriction that day 1 determines the minimum number of N_y required in the numerical model.

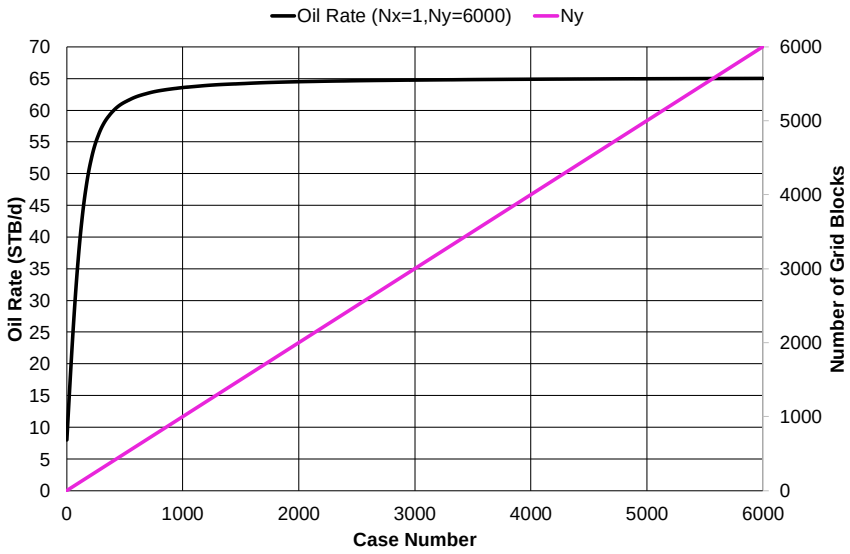


Figure 6.1: Estimated oil rate vs. N_y on day 1

In **Fig.6.1**, the rate increases significantly until N_y = 500, then the rate increases slightly until stabilizing at N_y = 5998 with the rate of 65.0147 STB/d. Besides, the true rate from Fig.A.10(a) is 65.0147 STB/d on day 1. To further examine the change in estimated rate when increasing the number of N_y, a logarithmic plot of the relative change in rate is made, as displayed in **Fig.6.2**. The relative change in rate is between the estimated rate given for a specific N_y and the reference rate (q_{ref}) given by N_y = 6000, to determine how close the estimated rate is from the true rate on day 1 of production:

$$\text{Relative change} = \left| \frac{q - q_{ref}}{q_{ref}} \right| \cdot 100\ (\%) \tag{6.1}$$

Table 6.1: Estimated oil rate and number of N_y for relative change in rate of 1, 0.1 and 0.001 %

Rel.change (%)	Ny	q (STB/d)
< 1	1728	64.3644
< 0.1	4767	64.94
< 0.01	5859	65.005

In **Table 6.1**, the estimated oil rate and the number of N_y are presented for some relative changes in rate. The relative change in rate is less than 1% from N_y = 1728, less than 0.1% from N_y = 4767 and less than 0.01% from N_y = 5859. Consequently, by reducing the number N_y from 6000 to 1728, which is 4272 fewer grid blocks, the rate is only 1% from the true rate of 65.0147 STB/d! This significant grid block reduction results in lower CPU time and determines the requirement for the maximum number of grid blocks needed in the 2D grid refinement study.

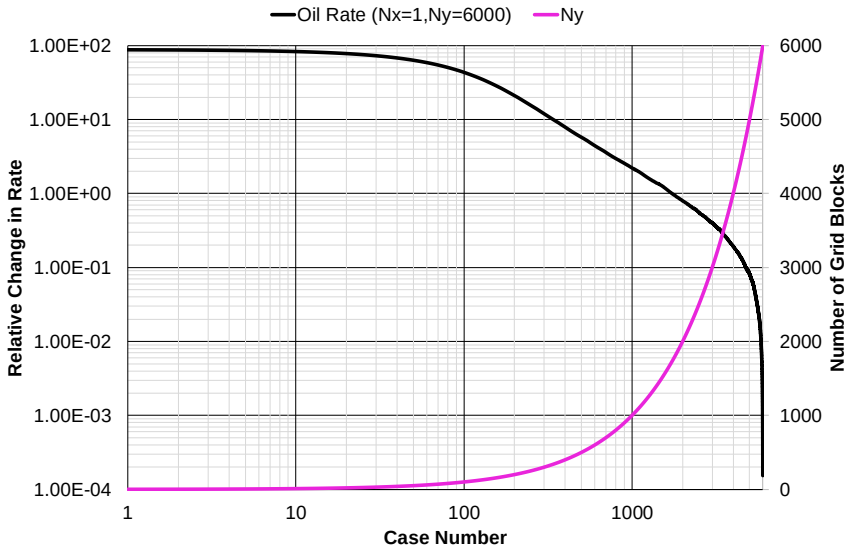


Figure 6.2: Logarithmic plot of relative change in estimated rate vs. N_y

Chapter 7

Probabilistic Approach To 2D Grid Refinement

7.1 Representation of A Distribution Through Histograms

The rates and cumulative volumes are generated for all possible cases at every time step for 100 days. To graphically represent the distribution of data (rates, cumulative volumes) at a specific time step, histograms are utilized. The advantage of a histogram is the simplicity of observing where the majority of the data falls into the measurement scale, and to observe how much variation there is. To construct a histogram, the data must be divided into a series of class intervals called ”bins” or ”buckets”, which covers the range of data from minimum to the maximum value. Then for each bin, the number of values in the data set that falls into each bin is counted. Frequency is the count of each bin, which is on the y-axis in the histogram. **Fig.7.1** represents the estimated oil rate on day 10 for possible combinations of $N_x \cdot N_y \leq N_{max} = 2000$ for $x_e/x_f = 1$. The bins are the estimated oil rates (STB/d), and the data range has been divided into 22 equal-sized bins.

In Fig.7.1, the most prominent column is bin 10.89 STB/d. Consequently, it is reasonable to assume that the best estimated rate for $x_e/x_f = 1$ on day 10 is given by this bin. However, a modification to the data must be made to automatically extract the correct bin and the best estimated rate within. It is highly inefficient for the engineer to make histograms for all time steps, where the number of time steps can vary significantly depending on the model, and pick out manually the bin with the highest frequency! Therefore, a method of doing this automatically is presented. The idea is to weight the “correct” results given by a grid block combination with a weight factor, which is either a continuous or a binary weight factor. The approach is elaborated in Section 7.2.2, but first, a manual procedure based on analyzing the estimated rates is described in Section 7.2.1.

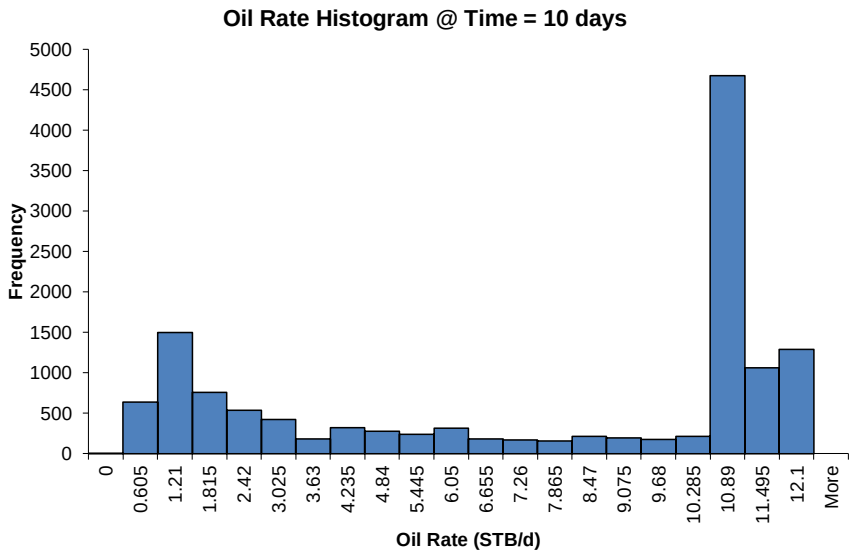


Figure 7.1: Original oil rate histogram for N_{max} = 2000 on day 10, when x_e/x_f = 1

7.2 The Methodology of Weighting Factors

The explanation of the methodology of weighting factors is conducted on a simpler example of N_{max} = 20 for x_e/x_f = 1. It is important to note that **both** N_x and N_y can vary to 20, because x_e/x_f = 1 is treated like a 2D problem. The reason for applying the methodology to x_e/x_f = 1 is because the true solution to the 1D problem is known! Furthermore, if the results after applying the proposed methodology is within line-thickness of the true solution determined in Chapter 6, then the methodology is considered successful for x_e/x_f = 1, and the methodology is most likely applicable to x_e/x_f = 1.5.

In **Fig.7.2** below, the combinations of N_x and N_y that do not exceed N_{max} is given in green, while the rest are in red. It is only 46 combinations that provide a green color that is further analyzed. Row N_y = 1 does not contain any values because this is the fracture cell, as mentioned in Section 4.1. In **Fig.7.3**, a histogram of the estimated oil rates on day 10 is presented. The frequencies of the different bins are displayed, and counting the frequency of each bin, it adds up to 46. The highest frequency is given for the bin 0.88 STB/d.

GRID BLOCKS IN Y-DIRECTION (Ny)	GRID BLOCKS IN X-DIRECTION (Nx)																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	17	18	19	20
1																			
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	34	36	38	40
3	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	51	54	57	60
4	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	68	72	76	80
5	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	85	90	95	100
6	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	102	108	114	120
7	7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	119	126	133	140
8	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	136	144	152	160
9	9	18	27	36	45	54	63	72	81	90	99	108	117	126	135	153	162	171	180
10	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	170	180	190	200
11	11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	187	198	209	220
12	12	24	36	48	60	72	84	96	108	120	132	144	156	168	180	204	216	228	240
13	13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	221	234	247	260
14	14	28	42	56	70	84	98	112	126	140	154	168	182	196	210	238	252	266	280
15	15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	255	270	285	300
16	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	272	288	304	320
17	17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	289	306	323	340
18	18	36	54	72	90	108	126	144	162	180	198	216	234	252	270	306	324	342	360
19	19	38	57	76	95	114	133	152	171	190	209	228	247	266	285	323	342	361	380
20	20	40	60	80	100	120	140	160	180	200	220	240	260	280	300	340	360	380	400

Figure 7.2: Possible combinations of N_x and N_x that do not exceed $N_{max} = 20$

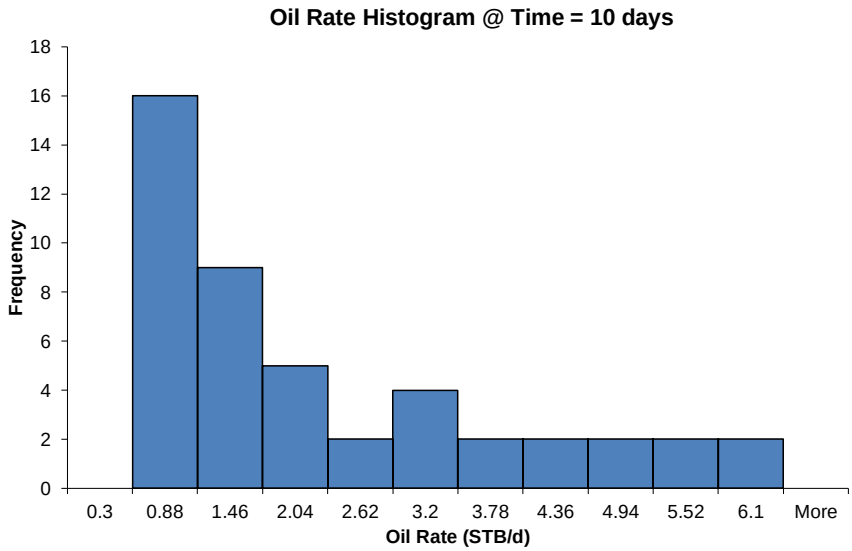


Figure 7.3: Original oil rate histogram for $N_{max} = 20$ on day 10

7.2.1 Manual Procedure

The green combinations in Fig.7.2 is replaced with the respective estimated rates in **Fig.7.4**. By analyzing the change in estimated rate in x- and y-direction, it is observed that the rates are not changing when increasing the number of N_x for a specific N_y. This indicates that only increasing the grid blocks in the y-direction affects the value of the estimated rate, not the number of grid blocks in the x-direction. In Fig.7.3, the bin 0.88 STB/d has the highest frequency of 16 and this bin is given by $N_y = 2-3$. The rate and the corresponding number

of N_y increases to the right in the histogram, and the highest estimated rate is presented by the bin 6.1 STB/d given by $N_y = 20$. In Section 3.4 about grid refinement study, it was mentioned that the more grid blocks used in a numerical model, the more accurate is the numerical solution. This suggests that the best numerical solution for $x_e/x_f = 1$ is in the direction of increased N_y, even though the histogram gives the highest frequencies for lower N_y!

GRID BLOCKS IN Y-DIRECTION (Ny)	GRID BLOCKS IN X-DIRECTION (Nx)																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1																				
2	0.337	0.337	0.337	0.337	0.337	0.337	0.337	0.337	0.337	0.337										
3	0.675	0.675	0.675	0.675	0.675	0.675														
4	1.011	1.011	1.011	1.011	1.011															
5	1.347	1.347	1.347	1.347																
6	1.681	1.681	1.681																	
7	2.013	2.013																		
8	2.344	2.344																		
9	2.672	2.672																		
10	2.998	2.998																		
11	3.321																			
12	3.640																			
13	3.956																			
14	4.269																			
15	4.577																			
16	4.880																			
17	5.179																			
18	5.472																			
19	5.760																			
20	6.042																			

Figure 7.4: The estimated oil rates given for all possible cases of $N_{max} = 20$

7.2.2 Automated Procedure

To isolate the ”true” results (estimated rates in this example) in a fully automated procedure, weight factors are assigned to the results. However, seeing as this information is unknown, a method is proposed to estimate these values by using the derivatives of the possible results. **Eq.7.1** is the forward-difference approximation that is applied to each rate in Fig.7.4. The index ”i” symbolizes the x-direction and ”j” the y-direction, which means $\Delta d_{i,j}$ is the derivative with respect to the change in rate in both the x- and y-direction. The value of the derivatives for each rate is presented in **Fig.7.5**. An important thing to note is when using the numeric derivatives, the data is reduced along all directions where i + 1 or j + 1 is not defined! However, as the total number of grid blocks is increased, the number of data points removed is far less than the total number of data points!

Nevertheless, this limitation has a greater impact on this ”small” example of $N_{max} = 20$. It is noticed after applying the forward difference approximation, the maximum number of N_y is reduced from 20 to 10 grid blocks. Consequently, the estimated rates given by $N_y = 11 – 20$ are not included when determining the correct bin.

$$\Delta d_{i,j} = |d_{i+1,j} - d_{i,j}| + |d_{i,j+1} - d_{i,j}| \tag{7.1}$$

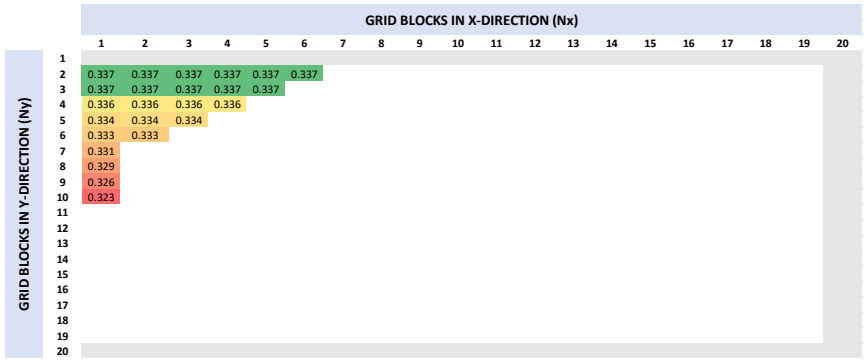


Figure 7.5: The value of the derivatives ($\Delta d_{i,j}$), after utilizing Eq.7.1 on the estimated rates

Method 1: Continuous Weight Factor

In method 1, a continuous weight factor is determined by using the weight factor equation given in **Eq.7.2**, and the weight factors are presented in **Fig.7.6**. The equation yields a weight factor which tends to 1 when the derivative in both the x- and y-direction tends to 0 and tends to 0 as the derivative term tends to the largest value. In Fig.7.6, the largest derivative has a value of 0.337 for $N_y = 2 - 3$ which results in a weight factor of 0, while the smallest derivative of 0.323 for $N_y = 10$ results in a weight factor of 0.041.

$$w_i = 1 - \frac{\Delta d_{i,j}}{\max(\Delta d_{i,j})} \tag{7.2}$$

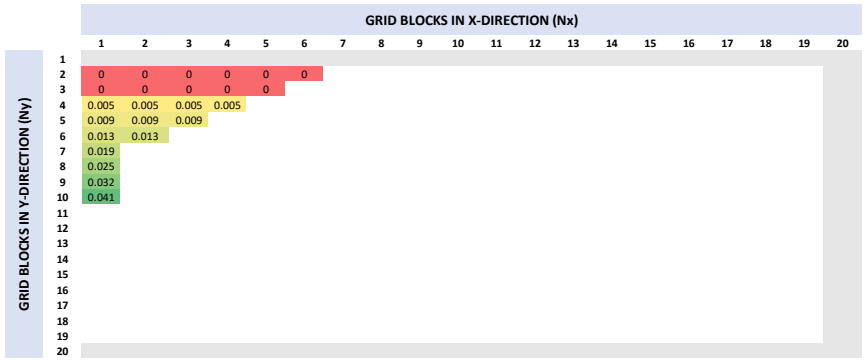


Figure 7.6: The continuous weight factors calculated by Eq.7.2

In **Table 7.1**, there are 3 columns present; where the first column contains the bins in STB/d. The second column is the frequency of the weighted rates calculated with method 1 and the original un-weighted rates. The last column is the relative frequency of the weighted and un-weighted rates, which makes up the y-axis in **Fig.7.7**. In the presented histogram, the height of all columns for weighted and un-weighted rates adds up to 1, and it expresses the probability of each bin occurring. The effect of normalizing the y-axis has a greater impact as the number of cases could be thousands when increasing N_{max}, and the could be a considerable difference in frequency between weighted and un-weighted data. Therefore, by adjusting the values measured on different scales to a common scale through normalization, it is easier to interpret the histogram. To get relative frequency, the frequency of a bin is divided by the total number of observations (sum of all frequencies for all bins).

Table 7.1: General frequency and relative frequency of weighted and un-weighted data when using method 1

	Frequency		Relative Frequency	
Bin	Weighted	Un-Weighted	Weighted	Un-Weighted
0.3	0	0	0.000	0.000
0.88	0	16	0.000	0.348
1.46	0.047	9	0.247	0.196
2.04	0.045	5	0.237	0.109
2.62	0.025	2	0.132	0.043
3.2	0.073	4	0.384	0.087
3.78	0	2	0.000	0.043
4.36	0	2	0.000	0.043
4.94	0	2	0.000	0.043
5.52	0	2	0.000	0.043
6.1	0	2	0.000	0.043

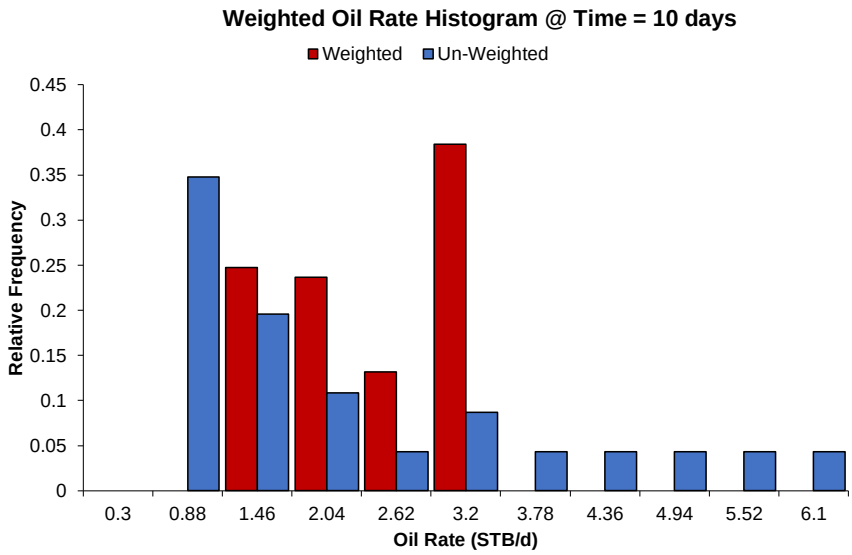


Figure 7.7: Weighted oil rate histogram for N_{max} = 20 with method 1 on day 10

The main observation in Fig.7.7 is the effect of using forward difference approximation on a small N_{max}, as the oil rates given by $N_y \geq 11$ have not been weighted. However, method 1 assigns the highest relative frequency to the bin 3.2 STB/d given by N_y = 10. Besides, the bin 0.88 STB/d given by $N_y \leq 3$ is assigned a weight factor of 0, which means the method neglects the results in the leftmost bin that are known to be "incorrect". Despite wanting the 6.1 STB/d bin to have the highest probability of occurring, this method manages to select the bin given by the highest number of N_y included in the weighting procedure.

However, there exists one major drawback regarding the continuous weight factor method. The method is heavily influenced by a large $\Delta d_{i,j}$, which is caused by a great change in the estimated rate in the x-and y-direction. A large $\Delta d_{i,j}$, which is the value of $\max(\Delta d_{i,j})$ in Eq.7.2, is going to weight the other estimated rates with a lower derivative closer to 1. The estimated rates are then assigned a too high weight factor, and consequently, too low weight factors are assigned to the estimated rates with derivatives closer to $\max(\Delta d_{i,j})$. In the presented example for N_{max} = 20, this is not a problem because of little change in rate in the y-direction, hence, a small change in the derivatives. However, this is not granted when $N_{max} \gg 20$, which makes it necessary for a method that is not heavily influenced by large changes in rate.

Method 2: Binary Weight Factor

To solve the problem with method 1, the continuous weight factor is replaced by a binary weight factor of either 0 or 1. A threshold value, ε, is introduced, which determines for what values the derivative is set equal to zero:

$$|\Delta d_{i,j}| \leq \varepsilon \tag{7.3}$$

which consequently assigns a weight factor of 1 to the respective estimated rates. The idea with the threshold value is to remove the estimated rates given by a low N_y, which are represented by larger derivatives for $x_e/x_f = 1$. By continuing with the example of $N_{max} = 20$ with method 2, the threshold value is set to be, for example, 0.332. All estimated rates with a derivative larger than 0.332 are removed from the weighting process, in other words, assigned a weight factor of 0. Fig.7.6 is then transformed into **Fig.7.8**, where only the estimated rates given by $N_y = 7 - 10$ are assigned a weight factor of 1. This results in the new histogram presented in **Fig.7.9**.

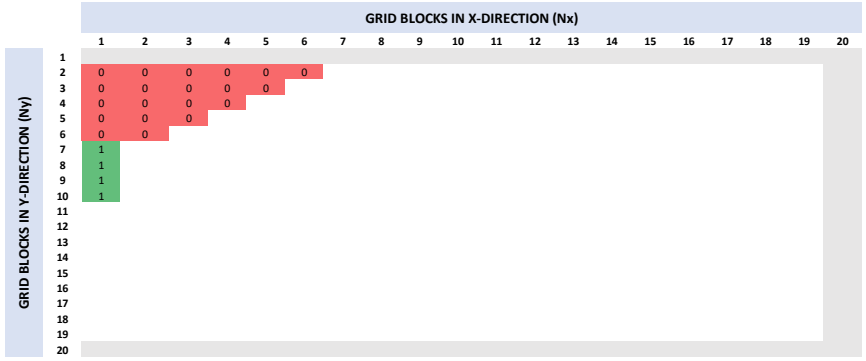


Figure 7.8: The binary weight factors assigned to the estimated rates when $\varepsilon = 0.332$

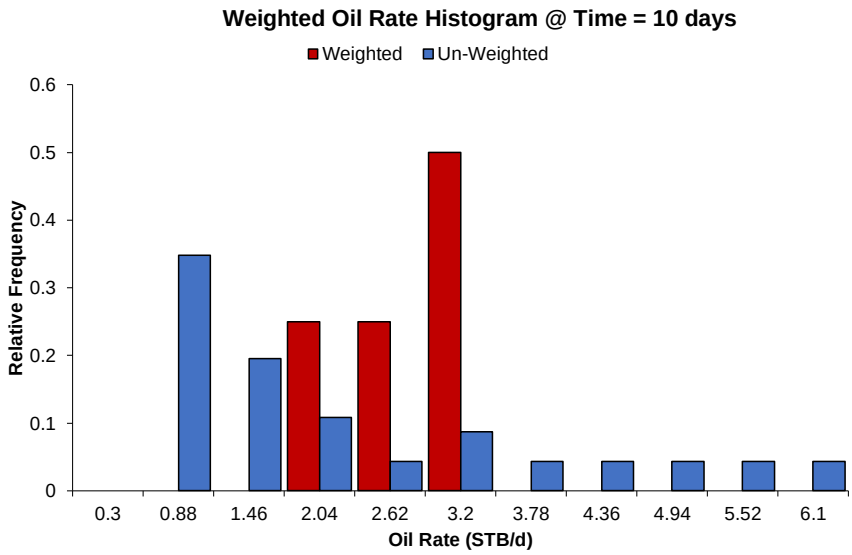


Figure 7.9: Weighted oil rate histogram for N_{max} = 20 with method 2 on day 10

For the weighted histogram made by method 2, there are several common features with the weighted histogram made by method 1. Both methods appoint the bin of 3.2 STB/d the highest relative frequency. In other words, method 2 manages to select the bin given by N_y = 10, which is the largest number of N_y the forward difference approximation for N_{max} = 20 is successfully applied to. Besides, method 2 removes at least one set of "non-correct" results, which are the rates given by $N_y \leq 6$ for the bins 0.88 STB/d and 1.46 STB/d. This is a valuable result, since it is known that the best-estimated solution is in the direction of increased N_y! It is important to remember when N_{max} increases, the number of data points removed by Eq.7.1 is far less than the total number of data points! Hence, it is easier to select the bin given by the largest number of N_y when more data points are included.

In further work, method 2 is used due to the following reasons:

- Method is not heavily influenced by large changes in rate
- Method only weights the results that fulfills the threshold requirement $\Delta d_{i,j} \leq \varepsilon$ - assures that the possible cases given by a low N_y are neglected.
- It requires less computational work to assign a binary weight factor than a continuous weight factor as N_{max} and the number of possible cases generated increases.

7.3 Normalization of Derivatives

A problem that arises when N_{max} is increasing and thousands of possible cases are generated, is the order of magnitude of the rates may vary greatly, which consequently affects the derivatives. Therefore, it is necessary to scale the derivatives to make sure that the order of magnitude of the rate has no impact on the weighting procedure. Following is the approach to normalizing the derivatives:

1. Create a square that is N_{max}-long in both x- and y-direction and determine possible cases from the condition $N_{tot} \leq N_{max}$
2. Calculate the derivatives (Δd_{ij}) for each possible case with the forward difference equation (Eq.7.1).
3. Find the combination of (N_x, N_y) that gives the lowest Δd_{ij}
4. With this combination it is possible to find the corresponding rate that gives the lowest Δd_{ij} - this is the reference rate (q_{ref}) that all derivatives is to be scaled by
5. Scale original derivatives (Δd_{ij}) by the reference rate to get the normalized derivatives:
$$\Delta \hat{d}_{ij} = \frac{\Delta d_{ij}}{q_{ref}} \tag{7.4}$$
6. Apply the restriction: $\Delta \hat{d}_{ij} \leq \varepsilon$.

The next step is to determine the optimum threshold value that is providing the best estimated rate on each time step. To determine this, a threshold sensitivity sensitivity analysis is performed in the next section.

7.4 Threshold Sensitivity Analysis

As a result of the 1D grid refinement study, to use $N_y = 1728$, the estimated rate for day 1 was only 1% from the true rate. Therefore, it is sufficient to set $N_{max} = 2000$ for the following threshold sensitivity analysis, which results in 13 516 possible combinations of N_x and N_x.

Fig.7.10 presents the estimated oil rates determined for different threshold values compared to the true solution given by $N_x = 1$ and $N_y = 6000$. The threshold value varies from $\varepsilon = 10^{-6} - 1$, and all the estimated rates are within line thickness of the true rate on day 10 to 100. However, it is noticeable that not all threshold values are providing the wanted result before day 10. **Fig.A.13(a)** in Appendix A.4 is providing a closer view on the estimated rates for the 10 first days, and it is observed that $\varepsilon = 1, 0.1, 0.01$ (underneath 0.1) and 0.05 do not match the true solution. Therefore, estimated rates given by these thresholds are removed, and only the most accurate thresholds are presented in **Fig.A.13(b)**. These are $\varepsilon = 10^{-6} - 10^{-3}$, however, only the estimated rate for $\varepsilon = 10^{-3}$ is noticed because the rest are underneath. By examining for later times in **Fig.A.14**, the match between the best threshold values and the true solution is exact! Consequently, one of these threshold values can be selected as the optimum.

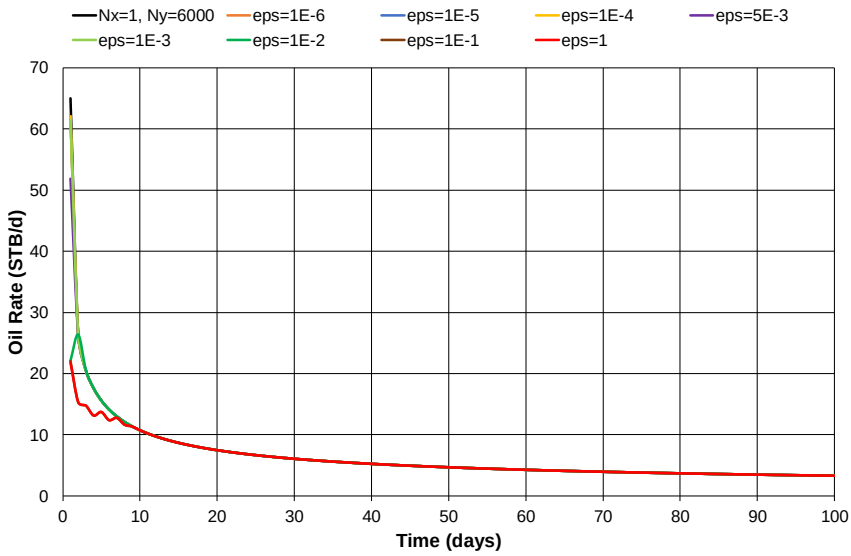


Figure 7.10: Estimated oil rate vs. time for different threshold values

It is valuable to know how many cases were used to provide the estimated rate at a specific time step when deciding the optimum threshold value. **Fig.7.11** presents for day 1 how the estimated oil rate and the number of cases that fulfilled the threshold requirement changes with threshold value. When the threshold value is 1, which is the maximum value, all 13 516 cases are included when estimating the rate on day 1. This is indeed the case for the

threshold values of 0.1 and 0.01, because all the derivatives are smaller than these thresholds. When the threshold value is too high, this affects the estimation of the rate badly, as the center of the data set shifts towards a lower value! As mentioned in Section 3.5 about statistical distributions, the median is the measure of the central location and is used for estimating the rate of the weighted data on each day. Therefore, the fewer cases included, the closer the estimated rate is to the true rate. From the 1D grid refinement study, the true rate on day 1 is 65.0147 STB/d.

With a threshold value in the range 10^{-4} to 10^{-6}, the estimated rate is 62.058 STB/d. More specifically, this is the best estimated rate determined by taking the median of all the rates given by 2912 cases. **Fig.A.15** and **Fig.A.16** in Appendix A.5 present the same plot for day 10 and 100. The true rate on day 10 is 10.7457 STB/d and the estimated rate with a threshold value in the range 10^{-4} to 10^{-6} is 10.7346 STB/d. This estimated rate was calculated from 2952 cases for the lowest threshold value compared to 4327 cases when the threshold value is 10^{-4}. On day 100, the true rate is 3.2971 STB/d and the estimated rate for threshold value of 10^{-5} and 10^{-6} is 3.2964 STB/d. 4493 cases were used when estimating this rate for both threshold values. In both figures, a supplementary figure with larger scale on the primary vertical axis is included to illustrate the small variation in rate when reducing the threshold value on day 10 and 100.

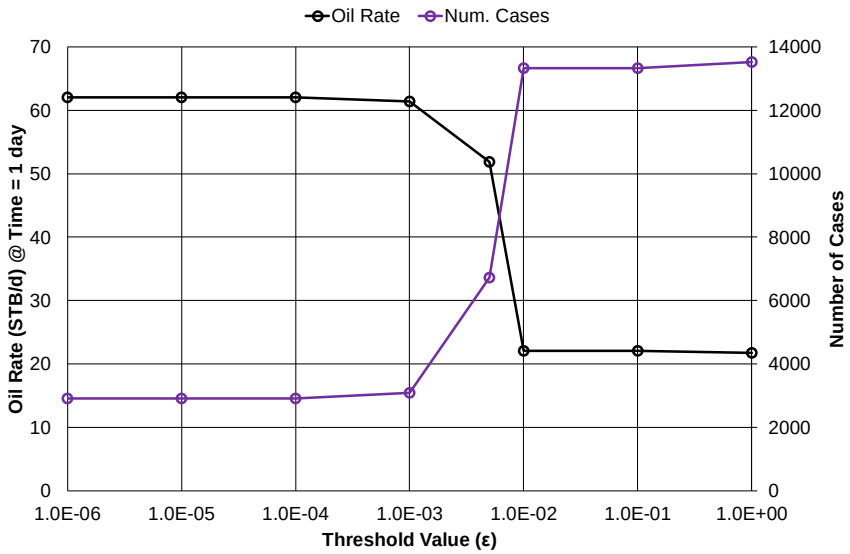


Figure 7.11: Estimated oil rate and number of cases used to get the estimated rate for different threshold values on day 1

Even if the threshold value is lowered to 0, the number of cases does not become zero and the estimated rate does not increase. The reason for this is because the maximum limit of significant digits has been reached for the value of Δd_{ij}. By examining the figures mentioned in this section, the choice of the threshold value for $x_e/x_f = 1$ is $\varepsilon = 10^{-6}$

because of the following reasons:

- Satisfying match with the true rate at early times
- Exact match with the true at late times
- The estimated rates on day 1 to 100 for this threshold value are determined from the lowest number of possible cases.

7.5 Weighted Histograms for $x_e/x_f = 1$

In this section, weighted oil histograms on day 1, 10 and 100 for $x_e/x_f = 1$ and $N_{max} = 2000$ is created. This is included to demonstrate that the presented methodology automatically selects the correct bin on these days. As emphasized earlier, both N_x and N_y can vary up to N_{max}, since the fracture penetration ratio is illustrating a 2D problem.

7.5.1 Weighted Histogram on Day 1

In the 1D grid refinement study, it was decided that day 1 determines the minimum number of N_y in the numerical model. The original oil rate histogram (with un-weighted rates) on day 1 is presented in **Fig.7.12**. Two prominent peaks are noticed at each end of the histogram. The leftmost peak is bin 3.225 STB/d given by $N_y \leq 10$. The rightmost peak is bin 64.5 STB/d given by $N_y \geq 500$. The desired outcome of applying the methodology of weighting factors is that the rightmost bin is appointed the highest relative frequency - meaning that these rates have the highest probability of occurring on day 1.

The weighted oil histogram for day 1 is displayed in **Fig.7.13** and it is noticeable that the relative frequency of the rightmost bin is 0.53, while for the leftmost bin the relative frequency is only 0.16! In Section 7.4, the estimated rate on day 1 is 62.058 STB/d. This estimated rate is 4.54% from the true rate of 65.0147 STB/d. By examining Fig.7.13, the true rate is not included in the range of the rightmost bin because the highest rate provided by all 13 516 cases is 64.4964 STB/d. However, this is most likely a consequence of N_y varying to 2000, and not 6000 as it did in the 1D grid refinement study, which affects the accuracy of the numerical solution on day 1.

To determine the optimum grid for $x_e/x_f = 1$, all the rates from the 2912 possible cases that fulfilled the threshold requirement is evaluated. The objective is to find grid combination that provides the closest estimated rate to 62.058 STB/d. Furthermore, the optimum combination for $x_e/x_f = 1$ is given by $N_x = 1$ and $N_y = 590$.

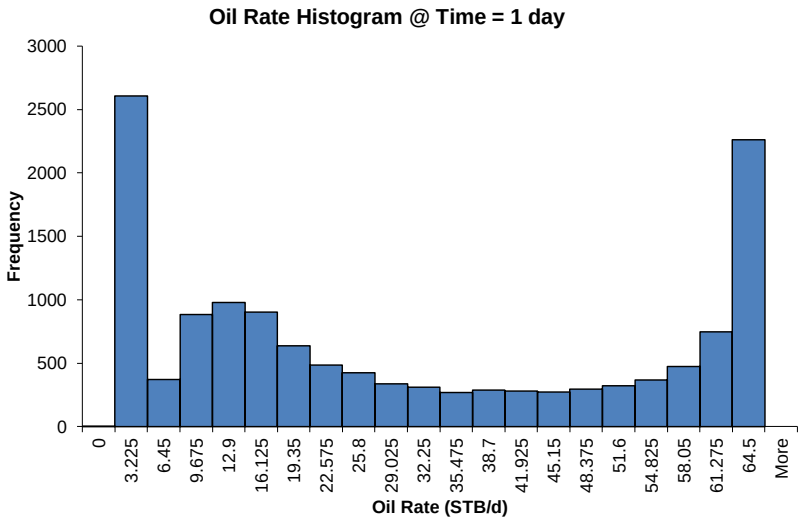


Figure 7.12: Original oil rate histogram for N_{max} = 2000 on day

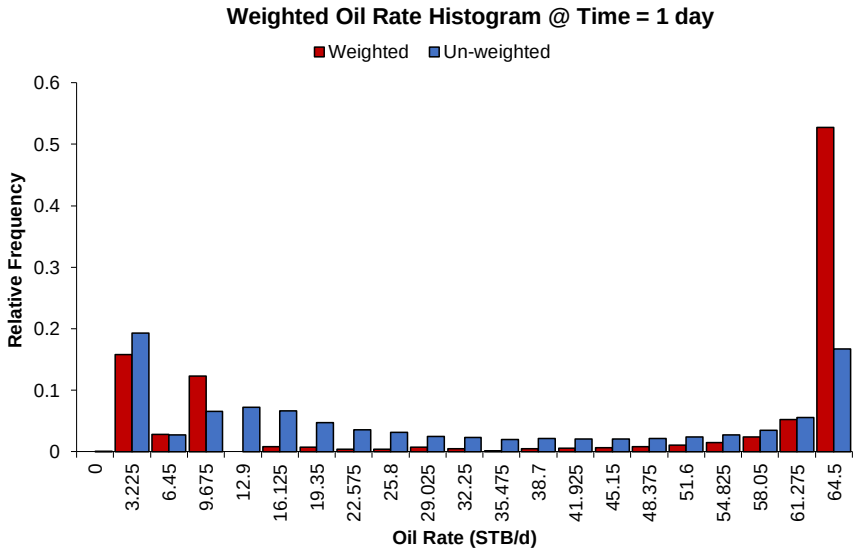


Figure 7.13: Weighted oil rate histogram for N_{max} = 2000 on day 1 with $\varepsilon = 10^{-6}$

7.5.2 Weighted Histogram on Day 10

The original oil rate histogram on day 10 was presented in Section 7.1 when describing the representation of a distribution through histograms, see Fig.7.1. By applying the proposed methodology, Fig.7.1 is transformed into **Fig.7.14**. The bin appointed the highest frequency in Fig.7.1 is also appointed the highest relative frequency in the weighted histogram. From the 1D grid refinement study, the true rate on day 10 is 10.7457 STB/d. This expresses that the true rate is in the range of the bin with the highest relative frequency! In other words, the proposed methodology managed to select the correct bin and provide an estimated rate of 10.7346 STB/d that is only 0.1 % from the true rate.

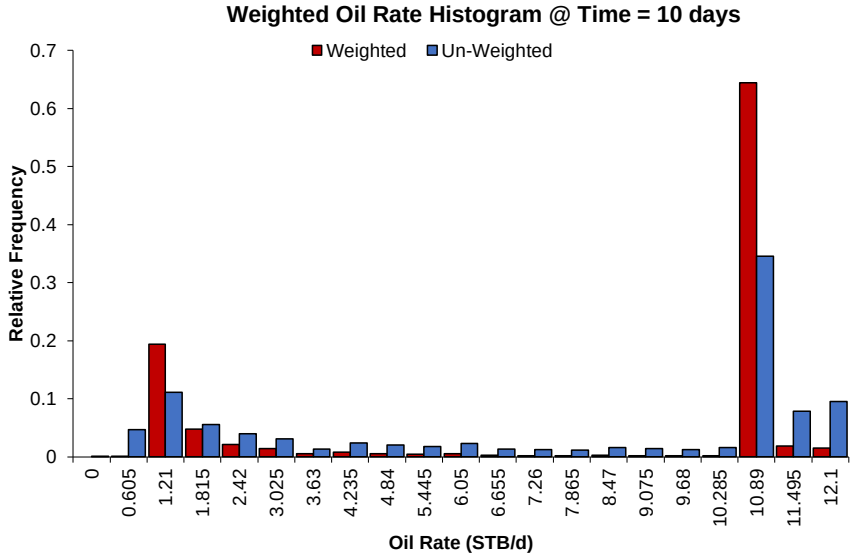


Figure 7.14: Weighted oil rate histogram for N_{max} = 2000 on day 10 with $\varepsilon = 10^{-6}$

7.5.3 Weighted Histogram on Day 100

On day 100, the original oil rate histogram is presented in **Fig.7.15**. There is only one prominent peak for bin 3.42 STB/d. When comparing with the weighted oil rate histogram in **Fig.7.14**, the same bin is appointed the highest relative frequency. Moreover, the correct bin was again selected by the methodology of weighting factors! Besides, the estimated rate on day 100 is 3.2964 STB/d that is only 0.02% from the true rate of 3.2971 STB/d. By comparing the results for day 1, 10 and 100, it is observed that the difference between the estimated rate and the true rate decreases for later times.

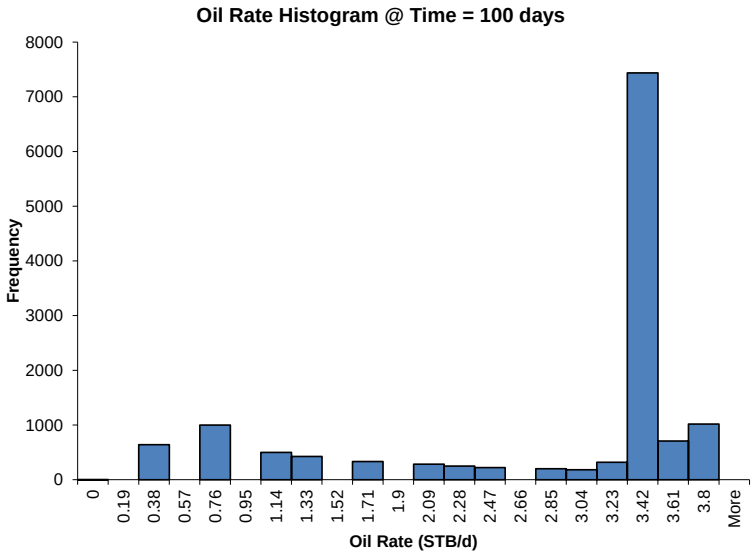


Figure 7.15: Original oil rate histogram for N_{max} = 2000 on day 100

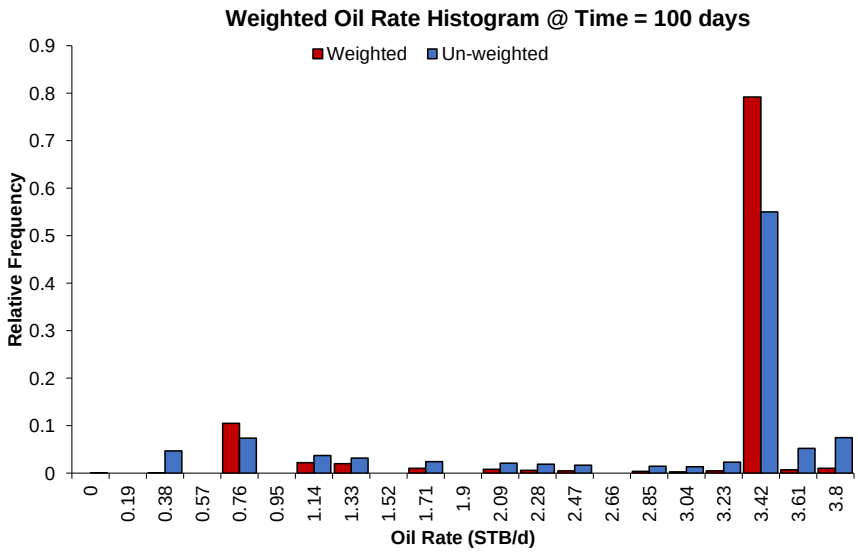


Figure 7.16: Weighted oil rate histogram for N_{max} = 2000 on day 100 with $\varepsilon = 10^{-6}$

Chapter 8

Application of Weight Factor Methodology to $x_e/x_f = 1.5$

Previously, it was expected that the 1D and 2D grid refinement study would provide the same results for $x_e/x_f = 1$ as the number of N_x has no impact on the result. This was confirmed when the methodology selected the correct bin containing the true rate and provided an estimated rate significantly close to the true rate on day 1, 10 and 100.

Regarding $x_e/x_f = 1.5$, there exists no true solution to compare with. The most accurate solution for $x_e/x_f = 1$ was given by $N_x = 1$ and $N_y = 6000$ in the 1D grid refinement study. However, for $x_e/x_f = 1.5$, the combination of N_{xf}, N_{xg} and N_y that provides the most accurate solution is unknown. Therefore, the estimated rates determined by the proposed methodology must be compared to an assumed "best case" of 6000 grid cells.

In **Fig.8.1**, simulations have been run for several combinations of N_x and N_y that provide a total of 6000 grid blocks. The rates given by $N_x = 2, 4, 6$ and 10 overlap and the resulting estimated rate is around 42 STB/d on day 1, which is a major increase in rate compared to the rates given by $N_x = 50$ and $N_x = 100$. However, this indicates that the most accurate solution for early times is provided when N_x is small and N_y is large. Since the rates given by $N_x = 2, 4, 6$ and 10 are almost identical, one of these grid block combinations are suitable for representing the best case. Therefore, chosen randomly, the combination of $N_x = 6$ and $N_y = 1000$ is used in further work.

It is important to point out that the combination of N_{xf} and N_{xg} for a specific N_x has no significant impact on the estimated rates. For example, the rate given by $N_x = 100$, to use $N_{xf} = 20$ and $N_{xg} = 80$, $N_{xf} = 80$ and $N_{xg} = 20$, or $N_{xf} = 50$ and $N_{xg} = 50$ is irrelevant as nearly the exact same rate is provided by all three combinations! This observation contradicts the statement that N_{xg} is more important than N_{xf} for providing a better numerical solution, as mentioned in Section 4.3. However, for the chosen combination of $N_x = 6$ and $N_y = 1000$, $N_{xf} = 1$ and $N_{xg} = 5$.

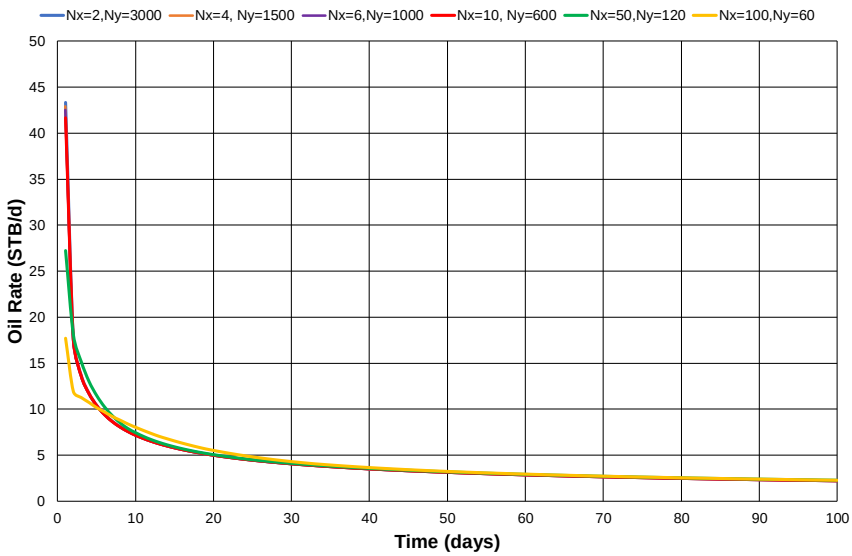


Figure 8.1: Estimated oil rate vs. time from different combinations of N_x and N_y that sums up to 6000 grid blocks

To apply the methodology of weighting factors to $x_e/x_f = 1.5$, a necessary requirement is that $n_{max} = N_{max}$. As defined in Section 5.2, n_{max} is the maximum number of combinations of N_{xf} and N_{xg} for a given value of N_x, and the combinations are randomly selected. Because of using the forward difference approximation on the estimated rates, it is necessary to know the value of $N_{xf} + 1$ and $N_{xg} + 1$ (and also $N_y + 1$) to be able to calculate the derivative of the rate given by (N_{xf}, N_{xg}, N_y). This means that all possible combinations of N_{xf} and N_{xg} for a given value of N_x must be known.

For $x_e/x_f = 1$, both N_x and N_y could vary up to the value of $N_{max} = 2000$. However, taking into consideration the requirement above, the amount of data generated for all possible combinations of N_{xf}, N_{xg} and N_y is going to be significantly large. Due to limitations in data storage, some restrictions must be set. The following has been decided for $x_e/x_f = 1.5$:

- $N_{max} = n_{max} = 1000$
- N_x can vary up 20
- N_y can vary up to N_{max}

which results in 15 190 possible grid block combination.

After applying the methodology of weighting factors and conducting a threshold sensitivity analysis, the best-estimated rates compared to the rates given by $N_x = 6$ and $N_y = 1000$

is presented in **Fig.8.2**. The match between the curves from day 10 to 100 is highly acceptable, however, the same cannot be said before day 10. The estimated rate on day 1 is only 14.9209 STB/d, which is a lot less compared to the accurate numerical solution. However, this is a direct consequence of having a limited number of maximum grid blocks. N_y cannot be greater than 500 since the minimum number of N_x is 2 grid blocks ($N_x = N_{xf}+N_{xg}$). Therefore, the lack of accuracy in the estimated rates before day 10 is a result of not using enough grid blocks in the y-direction.

As discussed in the previous chapters, day 1 determines the minimum number of grid blocks needed in the model. Furthermore, to determine the grid for $x_e/x_f = 1.5$, it is necessary to find the combination of (N_{xf}, N_{xg}, N_y) that provides the estimated rate closest to 14.9209 STB/d. The result is the combination of $N_{xf} = 6, N_{xg} = 3$ ($N_x = 9$) and $N_y = 46$, and this is the grid combination used when recreating the Locke-Sawyer curve of $x_e/x_f = 1.5$ in the next chapter.

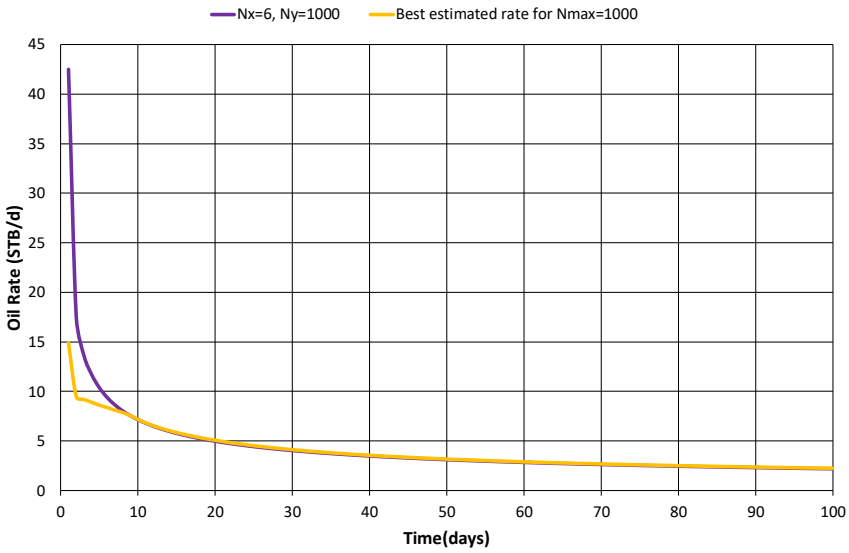


Figure 8.2: The accurate numerical solution with 6000 grid cells and best estimated result for $N_{max} = 1000$ with $N_x = 9$ and $N_x = 46$

Chapter 9

Recreation of Locke-Sawyer Curves with Numerical Reservoir Simulation

In the original Locke-Sawyer figure presented in Fig.2.1, the axis are expressed in dimensionless variables. Consequently, to recreate the curves for x_e/x_f = 1 and 1.5, the rate and elapsed time must be transformed into dimensionless variables, where the dimensionless rate is expressed by the equation:

$$q_D = \frac{141.2\mu B_0}{khΔP} \cdot q \tag{9.1}$$

and dimensionless time by:

$$t_D = \frac{0.000264k_m}{\phi_m \mu c_t x_f^2} \cdot t \tag{9.2}$$

The variables in the above equations are listed in **Table 9.1**. Reservoir thickness, matrix permeability, porosity, and drawdown pressure have been used as input data to Sensor. Initial drawdown pressure is the pressure difference between the initial reservoir pressure (7500 psi) and flowing bottom-hole pressure (2500 psi). The value of x_f, the fracture length, is 400 ft for x_e/x_f = 1 and 266.67 ft for x_e/x_f = 1.5.

Oil formation volume factor (B_o) and oil viscosity (μ_o) were determined from the generated Black Oil table seen in **Table 9.2**. Since the modeled flow is single-phase and saturation pressure for the oil is 1984.6 psi, only the undersaturated part of the BO-table belonging to PSAT = 1985 psi is needed. In the presented equations, only one value of μ_o and B_o is required. However, in reality, these are pressure functions that change with time! The oil compressibility (c_o) calculation is added into Table 9.2, and the formula used for determining this property is:

$$c_o = -\frac{1}{B_o} \cdot \left(\frac{dB_o}{dP} \right) \tag{9.3}$$

The total compressibility is given by the rock (matrix) and the oil compressibility added together, as there is no water or gas present;

$$c_t = c_r + c_o \tag{9.4}$$

where the value is presented in Table 9.1. The value of B_o, μ_o and c_o (and consequently the value of c_t) that provide the closets modeled curves to the Locke-Sawyer curves generated by Kappa, are the values marked in a light blue color and bold font in Table 9.2. It is important to take into account the following;

1. The data from Kappa express elapsed time in hours, which means that elapsed time in days used in the model must be changed to hours when determining t_D.
2. The estimated production rates must be scaled up by 4 to represent the reservoir, as mentioned in Section 4.1.
3. The model must be run long enough to get several orders of magnitude drop in the q_D value to include the BDF period. For practical purposes, 2 orders of magnitude drop are enough.

Table 9.1: Variables used in Locke-Sawyer dimensionless equations

Definition	Variable	Value	Unit
Reservoir thickness	h	150	ft
Matrix permeability	k	2.00E-04	md
Oil viscosity	μ_o	0.746	cp
Matrix porosity	ϕ	0.05	
Oil formation volume factor	B_o	1.1742	
Total compressibility	c_t	1.74E-05	1/psi
Initial drawdown pressure	ΔP	5000	psi

Table 9.2: Undersaturated black oil table and oil compressibility calculation

						Oil compressibility calc.	
PSAT	P	BO	VISO	BG	VISG	dBo/dP	co
psia	psia	rb/stb	cp	rb/scf	cp	1/psi	1/psi
1985	1985	1.2503	0.421	0.001728	0.0162		
	2500.3	1.2365	0.462	0.00138	0.0174	-2.52E-05	2.04E-05
	3000.3	1.2247	0.502	0.001164	0.0187	-2.23E-05	1.82E-05
	3500.3	1.2142	0.543	0.001015	0.0201	-2.01E-05	1.66E-05
	4000.3	1.2046	0.583	0.000907	0.0215	-1.82E-05	1.51E-05
	4500.1	1.196	0.624	0.000826	0.023	-1.65E-05	1.38E-05
	5000.1	1.1881	0.664	0.000762	0.0244	-1.51E-05	1.27E-05
	5500.5	1.1809	0.705	0.000712	0.0259	-1.39E-05	1.18E-05
	6000.4	**1.1742**	**0.746**	0.000671	0.0273	-1.29E-05	**1.10E-05**
	6500.1	1.168	0.786	0.000637	0.0286	-1.19E-05	1.02E-05
	7000.1	1.1623	0.826	0.000608	0.0299	-1.10E-05	9.46E-06
	7500.2	1.157	0.866	0.000584	0.0312	-1.03E-05	8.90E-06
	8000.1	1.152	0.906	0.000562	0.0324		

9.1 Recreation of $x_e/x_f = 1$

In Section 7.5.1 it was decided that the grid block combination of $N_x = 1$ and $N_y = 590$ was sufficient for providing the best estimated rate for $x_e/x_f = 1$ on day 1. In other words, this grid combination should be acceptable for modeling the whole production life. To get a 2 order magnitude drop in q_D, the simulation must be run for 1.5 million days.

In **Fig.9.1**, two modeled curves are present in addition to the Kappa curve. The curve denoted by $N_x = 1$ and $N_y = 6000$ is the most accurate numerical solution, which is added to demonstrate the accuracy of the curve modeled with the grid combination of $N_x = 1$ and $N_y = 590$. By examining the figure, it is observed that the curves match perfectly, which consequently means that the same accuracy of the numerical solution is achieved when using solely 590 grid blocks!

By comparing the curves to the Kappa curve, it is noticeable that the match is highly acceptable during the infinite acting flow period. The end of the IA period ($t_{D_{eia}}$) for all x_e/x_f-curves generated by Kappa was determined in the specialization project, and for $x_e/x_f = 1$ it is at $t_{D_{eia}} = 1.19E-01$. From this time onward, the rate is in the boundary dominated region and the curves separate.

In the specialization project, there was detected a strange behavior in the Kappa data at late-time rates. Validation of the data in the boundary dominated region had to be conducted. The following paragraph explains the reason for unreliable data from Kappa and is quoted directly from the project:

"Kappa uses Laplace transform to transform ordinary differential equations (ODEs) to algebraic equations. The purpose of this is to make it easier to solve ODEs. Laplace inverse transform, however, is performing the opposite function to do numerical modeling. The

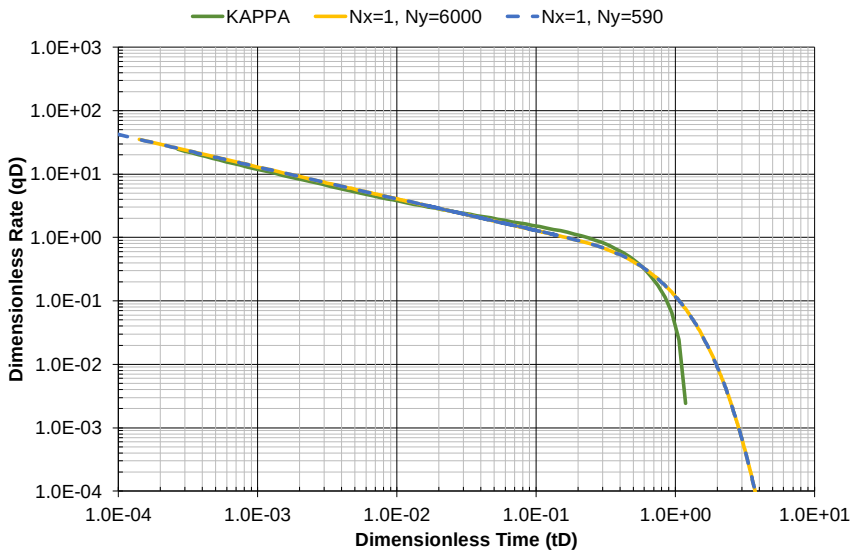


Figure 9.1: The $x_e/x_f = 1$ curve generated by Kappa and model

Kappa software utilizes the Gaver-Stehfest inversion algorithm. This is a popular inversion method in reservoir engineering due to being highly accurate and stable for most pressure solutions. However, the errors in the data from Kappa is grounded in the limitations of this inversion method. It is difficult to obtain accurate data from late-time exponential decline data [12]. The reason is, according to Leif Larsen, due to standard round-off issues that are inherent in the Gaver-Stehfest inversion method. When this inversion method is used with Laplace transform for small rates, q_D ends up oscillating around 0 and therefore data from this method becomes unreliable at late-time rates." [2].

In other words, this means that the lack of matching between the modeled curves and the Kappa curve in the BDF region is not a result of the errors in the numerical model, but a result of the errors in the data from Kappa!

9.2 Recreation of $x_e/x_f = 1.5$

In Chapter 8, the grid block combination for $x_e/x_f = 1.5$ was determined to be $N_x = 9$ and $N_y = 46$. However, this is not the "optimum" grid combination since the estimated rate was badly affected by the low number of N_y used when generating the possible cases. To include the BDF period, the simulation was run for 554 000 days.

In **Fig.9.2**, the Kappa curve, and two modeled curves are presented. It is observed that the match between the two modeled curves is quite sufficient except at early t_D, or more specifically, before $t_D = 1.3E-04$. Before this time the ($N_x = 9$,$N_y = 46$) curve deviates slightly to a lower value of q_D. However, this was expected because of the low number of

grid blocks used in the numerical model. Nevertheless, the match as t_D increases towards $t_D = 1.0E+01$ is much better than anticipated! This observation confirms that a low grid block combination mostly affect the accuracy of the numerical solution in the early times of the IA period compared to the rest of the production life.

The lack of matching between the Kappa curve and the modeled curves from the start of the BDF region at $t_{D_{eia}} = 3.76E-01$ is due to the mentioned errors with the Kappa data in the previous section. However, for the IA flow period, all three curves provide acceptable matching.

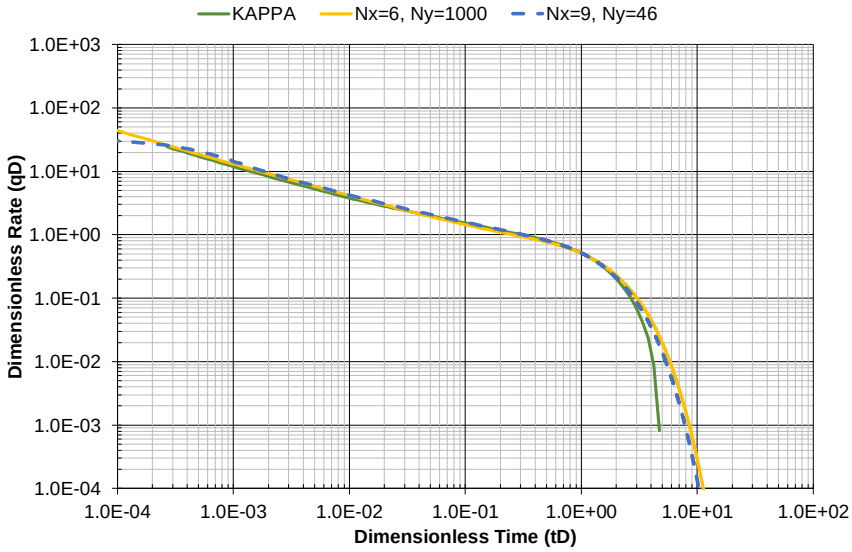


Figure 9.2: The $x_e/x_f = 1.5$ curve generated by Kappa and model

Chapter 10

Discussion of Results

The presented study describes the recreation of the Locke-Sawyer curves of x_e/x_f = 1 and 1.5 by implementing a probabilistic approach to grid refinement. If presented with a limited maximum number of grids to use in the reservoir simulator and without knowing the optimum grid combination to model the reservoir flow accurately, the methodology of weight factors presents a fully automated procedure to determine the optimum grid block combination. The results from the 2D grid refinement study conducted on the simplest fracture penetration ratio of 1, indicates that the presented methodology provides the same results as the 1D grid refinement study. This confirms the applicability of the methodology for the simplest case, and the result is a Locke-Sawyer curve that completely matches the curve made by 6000 grid cells.

The results presented for x_e/x_f = 1.5 demonstrates that the methodology is applicable for determining a grid block combination that provides an adequate recreation of the Locke-Sawyer curve. However, the study suggests that a curve with better accuracy in the early IA period could have been created if the analysis was not limited by a too-small N_{max} value, and consequently limiting the optimum grid block combination. Moreover, the study expresses that not using enough grid cells, especially in the y-direction, affects the accuracy of simulating the well's early time production.

Kappa Engineering provided the missing data to represent the original Locke-Sawyer curves. By crating numerical models with the grid block combinations acquired from the 2D grid refinement study, the objective was to analyze how much these curves deviate from the Kappa curves. The result from this comparison is that Kappa and the modeled curves provide a sufficient match for the IA period, however, the underlying errors in the Kappa data are revealed in the BDF period. This suggests that the modeled curves are better suited than the ones from Kappa when performing production forecasting for a well with a vertical hydraulic ICF, which was the overall aim of this research.

The probabilistic approach to grid refinement made it possible to decide the optimum combination of N_x and N_y for the fracture penetration ratio of 1, and the combination of N_{xf}, N_{xg} and N_y for fracture penetration ratio of 1.5, simply and systematically, an in a automated fashion. At the beginning of the study, it was considered that more grid blocks were needed in the x-direction outside the fracture than along the fracture to achieve the best numerical solution. However, this statement was contradicted, and the study emphasizes that to use enough grid blocks in the y-direction is of highest importance when trying to achieve the best numerical solution and to remake the Locke-Sawyer curves with sufficient accuracy.

The advantage of the probabilistic approach to grid refinement compared to traditional grid refinement study is the manual labor saved by running many simulations to check for the best grid block combination. If this was to be conducted for the fracture penetration ratio of 1.5, where the three variables; N_{xf}, N_{xg} and N_y had to be changed until little change was observed the reservoir model performance, this could potentially have been an extensive process. With the presented methodology that is fully automated, minimal input parameters are required to initialize the process of determining the optimum grid block combination, a process that is highly time efficient.

Due to the limitation in data storage and the requirement of $N_{max} = n_{max}$ for $x_e/x_f = 1.5$, this resulted in a N_y that was not large enough to provide the best estimated rate for the first 10 days. Consequently, this affected the determination of the optimum grid block combination, as this is a result of the grid block combination that gives the closest rate to the estimated rate on day 1. Besides, when recreating the $x_e/x_f = 1.5$ curve, the consequence of using a small number of N_y (because of too-small N_{max}) in the model manifests itself in the early time of the IA period. However, this does not indicate that the methodology does not work for fracture penetration ratio of 1.5, it just emphasizes the requirement about having enough data storage.

Chapter 11

Final Comments and Conclusion

Throughout this work, a methodology has been developed that estimates rates for a given number of time steps in an automated procedure. This procedure consists of using the derivatives of the rates from possible grid block combinations that do not exceed the maximum number of total grid blocks and assigning the estimated rates that satisfy the threshold requirement a binary weight factor of 1. This makes it possible to separate the grid block combinations that provide an accurate estimated rate from grid combinations that contribute to inaccurate rates.

To test the methodology, it was demonstrated on the simplest fracture penetration ratio of 1, because the true solution was known. Histograms were used to display the distribution of rates for all possible cases at a given time step, and the methodology managed to select the bin containing the true rate. Besides, the optimum grid block combination consisting of solely 590 total grid blocks provided an excellent match with the most accurate numerical solution obtained from 6000 grid cells.

As there was no true solution presented for the fracture penetration ratio of 1.5, the author had to select the best grid block combination given by 6000 grid cells. The requirement of generating all the possible combinations of N_{xf} and N_{xg} for a given number of N_x to be able to use the finite difference approximation, lead to restrictions in the choice of N_{max} because of limited data storage. Consequently, this resulted in a N_y that was not large enough to provide accurate estimated rates for the 10 first days. Nevertheless, for the remaining 90 days, the match was satisfactory despite methodology not selecting the optimum grid block combination.

The Locke-Sawyer curves for x_e/x_f = 1 and 1.5 were modeled using the grid combination determined from the methodology, and both modeled curves provided an exact match with the Kappa curve in the IA region, except for x_e/x_f = 1.5 in the earliest moments. As the late rate-time data in Kappa data was unreliable, this study has presented a set of curves that is more accurate to unconventional performance forecasting for a well with a vertical

ICF for the whole production period.

For future work, it is possible to conduct the methodology on other geometries, i.e rectangles, by adjusting the length of the reservoir in the y-direction.

For the remaining fracture penetration ratios of 3 and 5, it is just to follow the same procedure conducted for fracture penetration ratio of 1.5. However, unless the user has more data storage, it is important to take into account that the resulting grid block combination is most likely not large enough to model the respective Locke-Sawyer curves accurately in the early time of the IA period.

The Python codes presented herein are written in the best possible way by the author with limited programming skills, therefore, some flaws may be present. However, it is possible to optimize the codes by writing them in a more simple and efficient way.

Nomenclature

Symbol	=	*Definition*
q_i	=	Initial rate (STB/d)
D	=	Decline constant
b	=	Hyperbolic decline constant
x_e	=	Reservoir half-length in x-direction (ft)
x_f	=	Fracture half-length (ft)
x_e/x_f	=	Fracture penetration ratio
y_e	=	reservoir half-height in y-direction (ft)
N_x	=	Grid blocks along x-direction
N_y	=	Grid blocks along y-direction
N_z	=	Grid blocks along z-direction
N_{xf}	=	Grid blocks along fracture in x-direction
N_{xg}	=	Grid blocks along the end of fracture in x-direction
DELX	=	Grid block dimension in x-direction (ft)
DELX	=	Grid block dimension in y-direction (ft)
dxf	=	fracture step length in x-direction (ft)
dxg	=	step length outside fracture in x-direction (ft)
dy	=	step length in y-direction (ft)
q_{oil}	=	Oil rate (STB/d)
q_{gas}	=	Gas rate (MSCF/d)
PI	=	Productivity index (STB/d/psi)
PGRID	=	Pressure in the grid cells (psi)
PBH	=	Bottom hole pressure (psi)
ϕ	=	Porosity
μ	=	Viscosity (cp)
B_o	=	Oil formation volume factor (rb/stb)
c_t	=	Total compressibility (1/psi)
c_o	=	Oil compressibility (1/psi)
c_r	=	Rock compressibility (1/psi)
t_D	=	Dimensionless time
q_D	=	Dimensionless rate
$t_{D_{eia}}$	=	End of infinite acting period

Bibliography

[1] KAPPA Engineering. “Curriculum Vitae Leif Larsen”. In: (2019). URL: https://www.kappaeng.com/KAPPA/CV/Leif\%20Larsen.pdf1.

[2] M. Skintveit. “Describing Hydraulic Fractured Well Geometry by Fetkovich Methodology”. In: *TPG4560* (2019).

[3] J. Arps. “Analysis of Decline Curves”. In: *SPE* 160.SPE-945228-G (01 1944). DOI: https://doi.org/10.2118/945228-G.

[4] M. Fetkovich. “Decline Curve Analysis Using Type Curves”. In: *SPE* 32.SPE-4629-PA (06 1980). DOI: https://doi.org/10.2118/4629-PA.

[5] C. Locke and W. Sawyer. “Constant Pressure Injection Test in a Fractured Reservoir-History Match Using Numerical Simulation and Type Curve Analysis”. In: *SPE* SPE-5594-MS (1975). DOI: https://doi.org/10.2118/5594-MS.

[6] M. H.Kazemi I.Eker and B.Kurtoglu. “Fundamentals of Gas Shale Reservoirs”. In: (2015), pp. 277–294.

[7] Z. W. T. Hinkley R. Gu and D. Camilleri. “Multi-Porosity Simulation of Unconventional Reservoirs”. In: *SPE* SPE-167146-MS (2013). DOI: https://doi.org/10.2118/167146-MS.

[8] P. A. Slotte. “Lecture Notes in Well-Testing”. In: *TPG4115 Reservoir Determination by Core Analysis and Well Testing* (2017).

[9] D. N. Arnold. “Stability, consistency, and convergence of numerical discretizations”. In: *School of Mathematics, University of Minnesota* (2015).

[10] PetroWiki. “Gridding in Reservoir Simulation”. In: (2019). URL: https://petrowiki.org/Gridding_in_reservoir_simulation#Grid_refinement.

[11] I. Yusra. “Gridding in Reservoir Simulation”. In: (2019). URL: https://whitson.com/2019/11/10/gridding-in-reservoir-simulation/.

[12] B Josso and L Larsen. “Laplace Transform Numerical Inversion”. In: (2012). URL: https://www.kappaeng.com/PDF/Laplace_transform_numerical_inversion.pdf.

Appendices

A Figures

A.1 Flowcharts

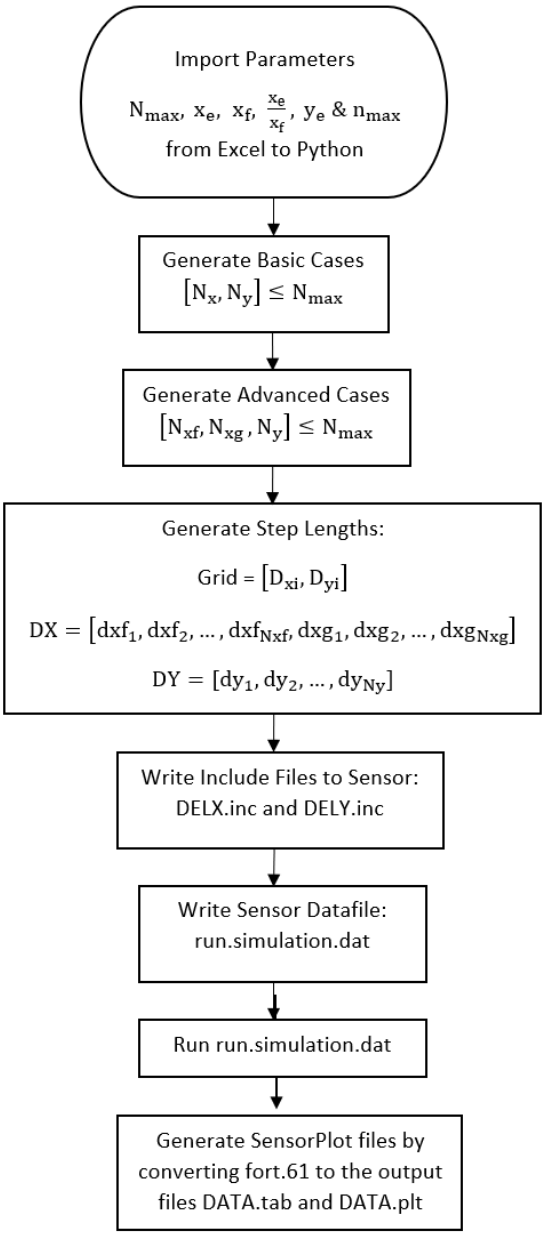
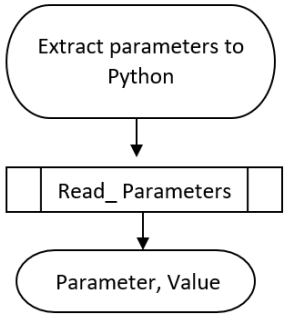
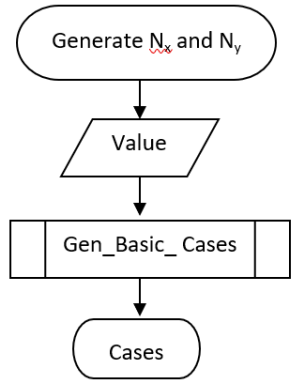


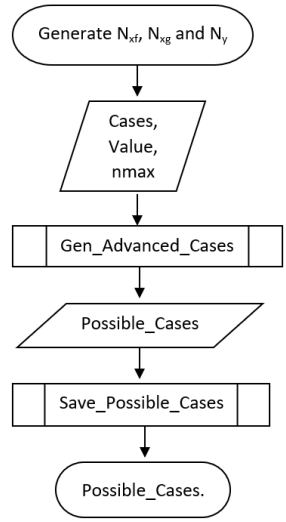
Figure A.1: Flowchart of Python code process in main.py



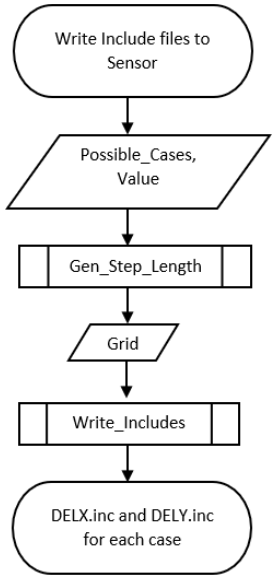
(a) Extract required parameters from Excel



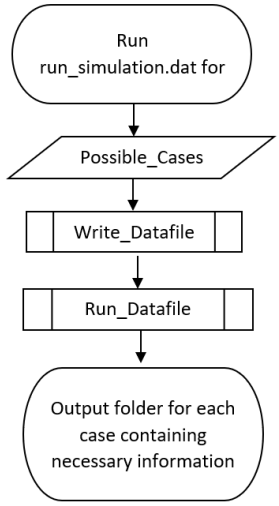
(b) Generate basic cases



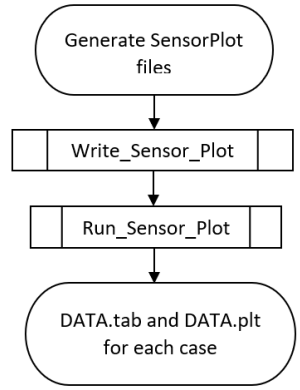
(c) Generate advanced cases



(d) Write include files



(e) Run simulations



(f) Generate files for SensorPlot

Figure A.2: Flowcharts of codes used in main.py

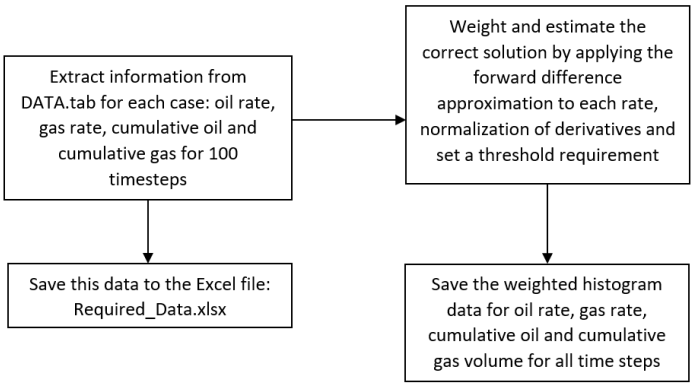
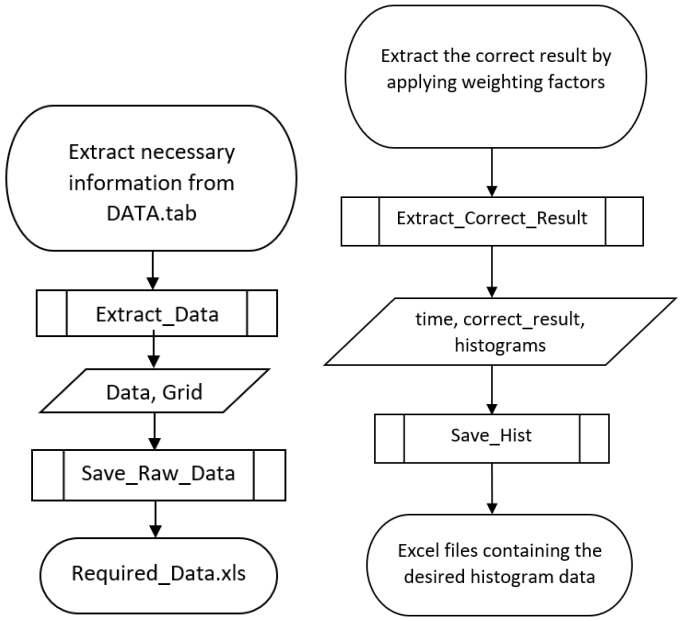


Figure A.3: Flowchart of Python code process in main_extracter.py



(a) Save required data (b) Save histogram data

Figure A.4: Flowcharts of codes used in main_extract.py

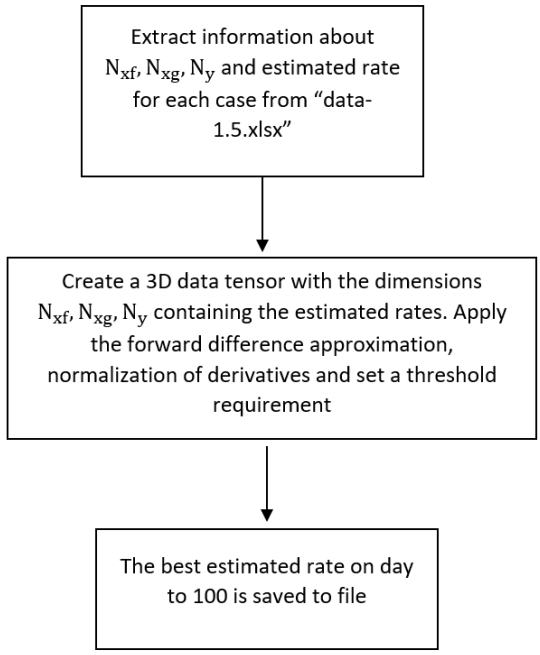


Figure A.5: Flowchart of Python code process in main_extracter.py

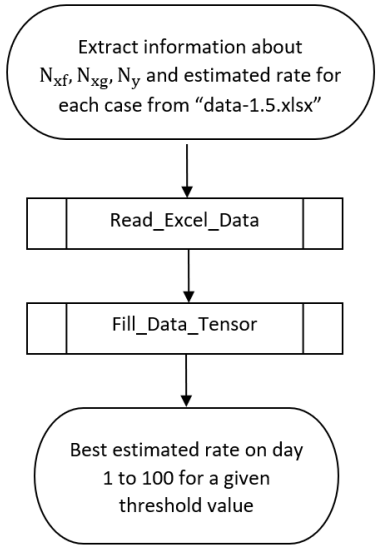
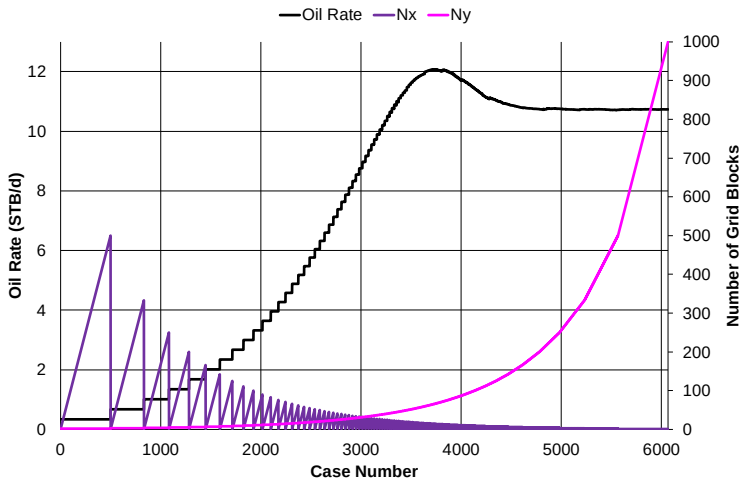
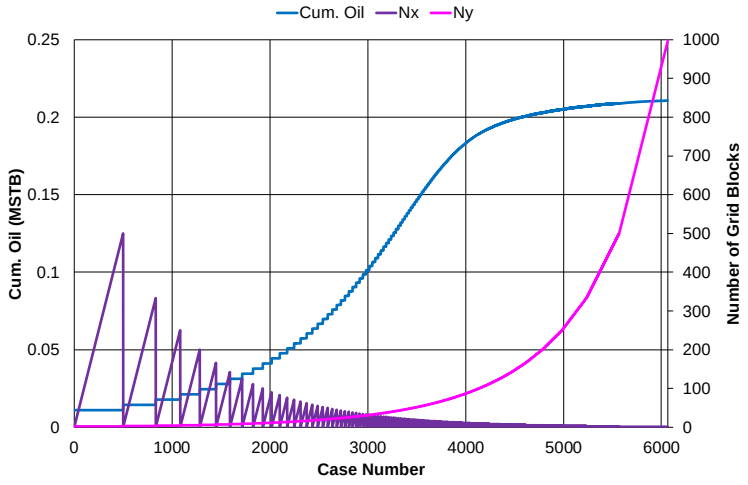


Figure A.6: Flowchart of codes used in main_extracter.py

A.2 PI Sensitivity Analysis for $N_{max} = 1000$ and $x_e/x_f = 1$

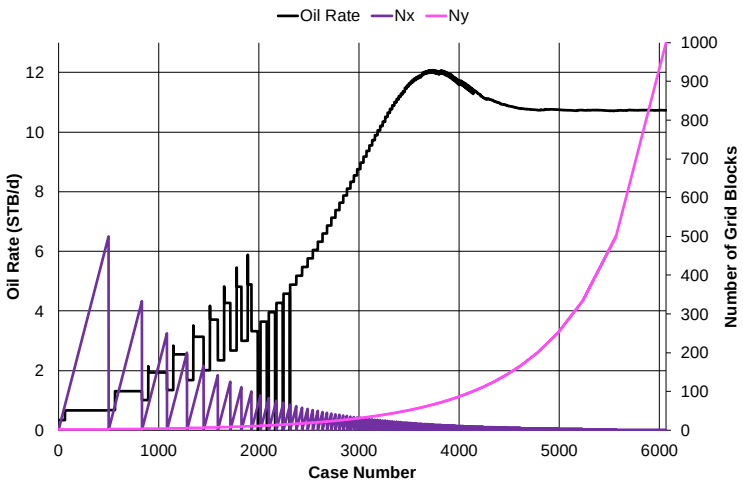


(a) Estimated oil rate vs. N_x and N_x on day 10 for PI=1 STB/d/psi

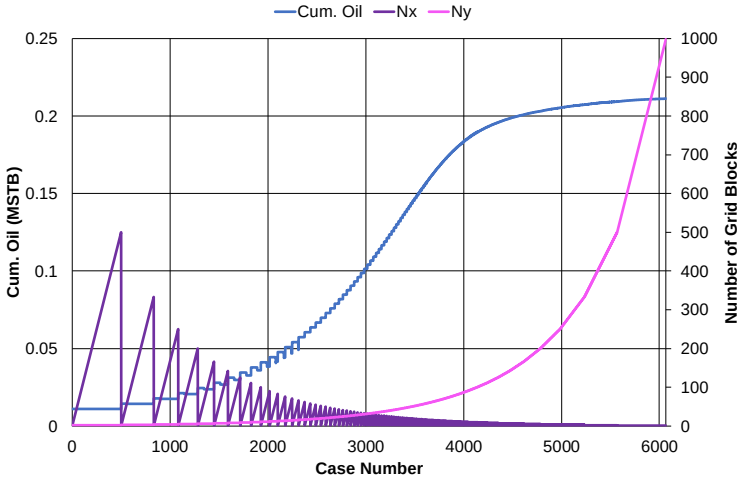


(b) Cumulative oil volume vs. N_x and N_x on day 10 for PI=1 STB/d/psi

Figure A.7: Estimated oil rate and cumulative oil volume for PI=1 STB/d/psi

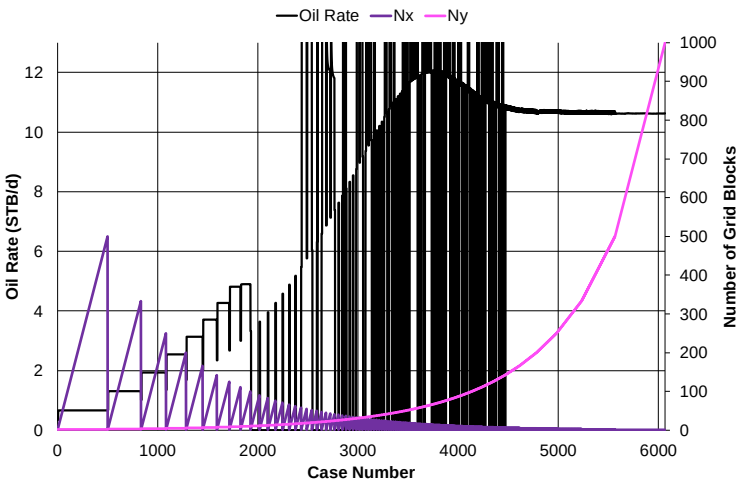


(a) Estimated oil rate vs. N_x and N_x on day 10 for PI=10 STB/d/psi

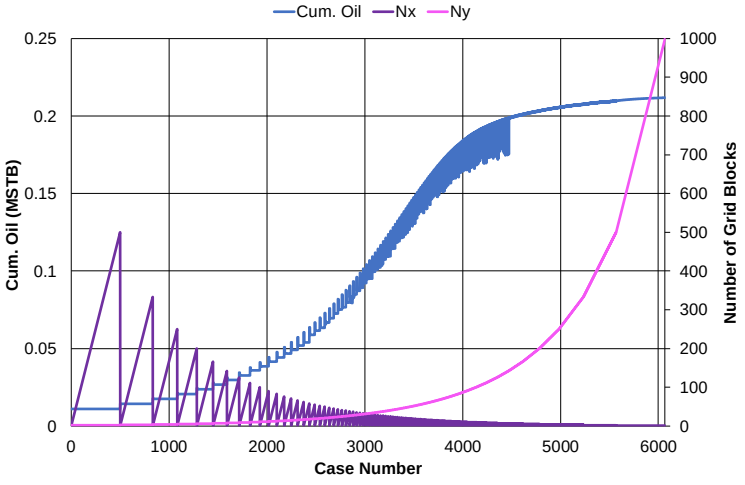


(b) Cumulative oil volume vs. N_x and N_x on day 10 for PI=10 STB/d/psi

Figure A.8: Estimated oil rate and cumulative oil volume for PI=10 STB/d/psi



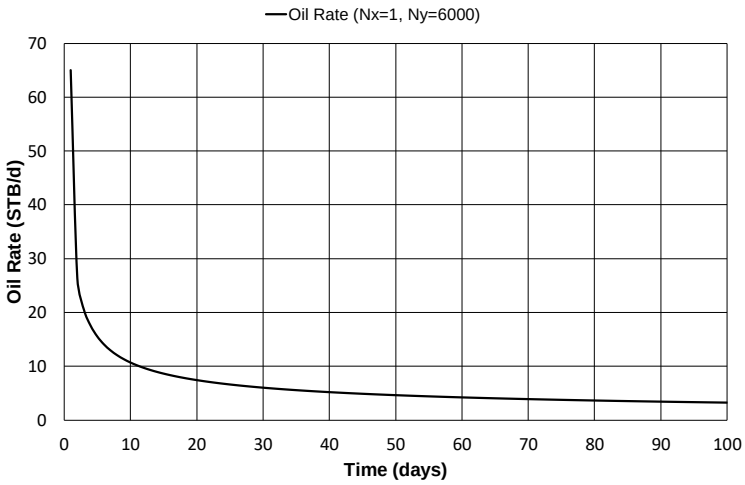
(a) Estimated oil rate vs. N_x and N_x on day 10 for PI=100 STB/d/psi



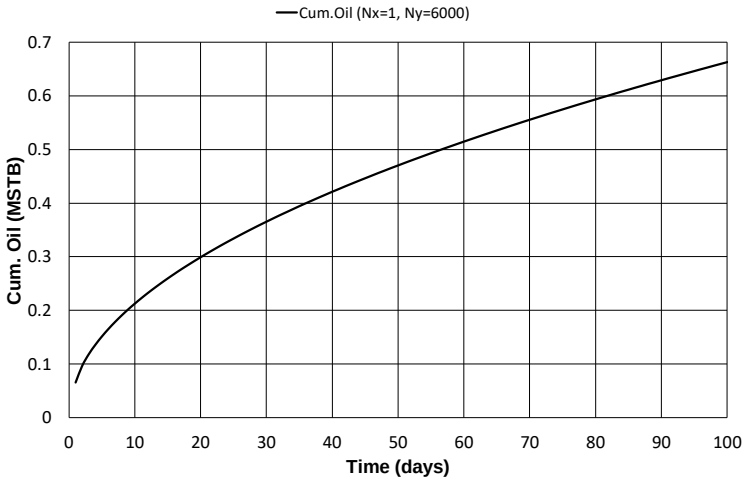
(b) Cumulative oil volume vs. N_x and N_x on day 10 for PI=100 STB/d/psi

Figure A.9: Estimated oil rate and cumulative oil volume for PI=100 STB/d/psi

A.3 Results from 1D Grid Refinement Study

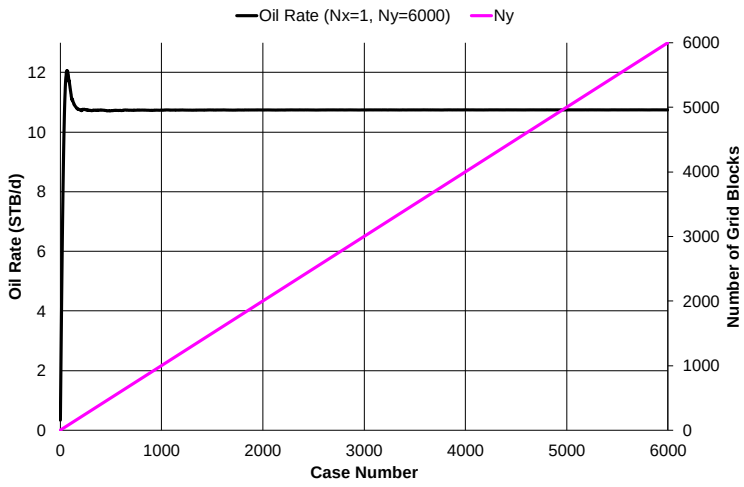


(a) Estimated oil rate vs. time for $N_x = 1$ and $N_y = 6000$

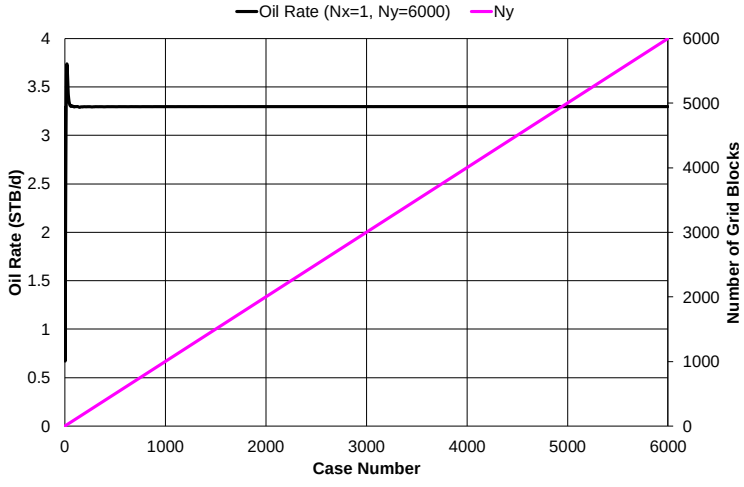


(b) Estimated cumulative oil volume vs. time for $N_x = 1$ and $N_y = 6000$

Figure A.10: Estimated oil rate and cumulative oil volume from 1D grid refinement study

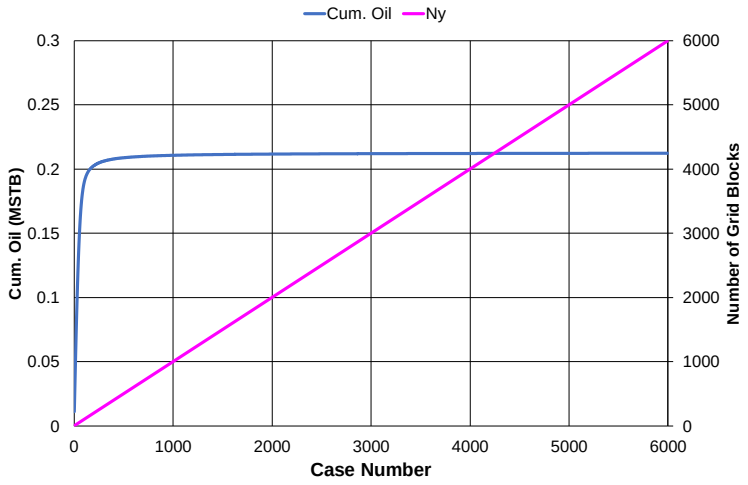


(a) On day 10

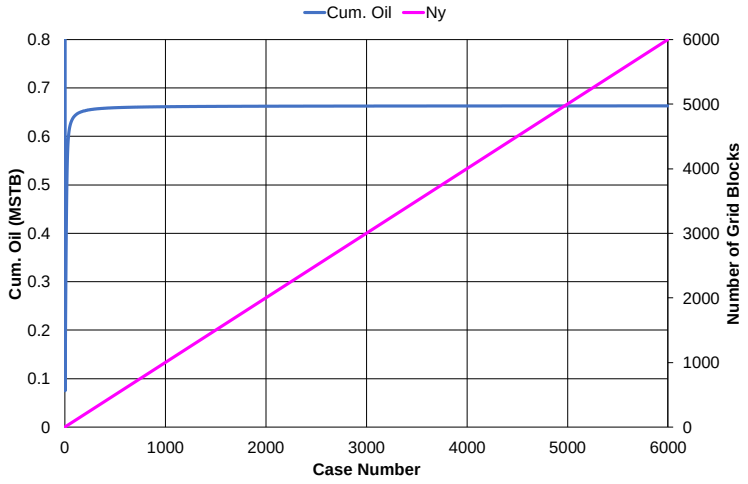


(b) On day 100

Figure A.11: Estimated oil rate vs. N_y for $N_x = 1$ and $N_y = 6000$



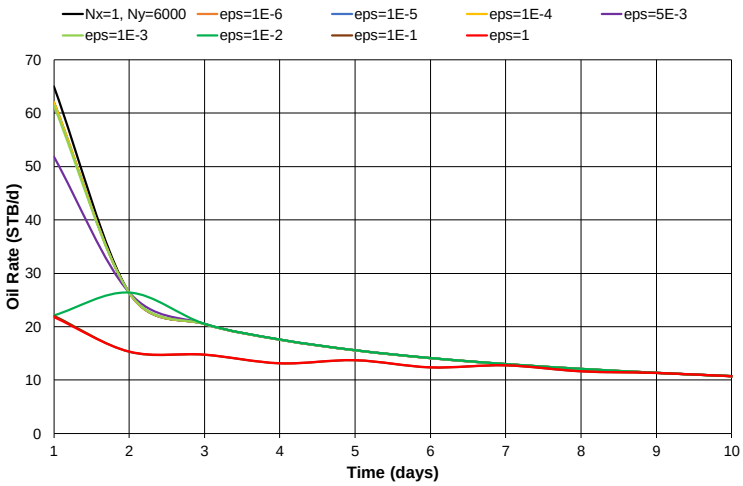
(a) On day 10



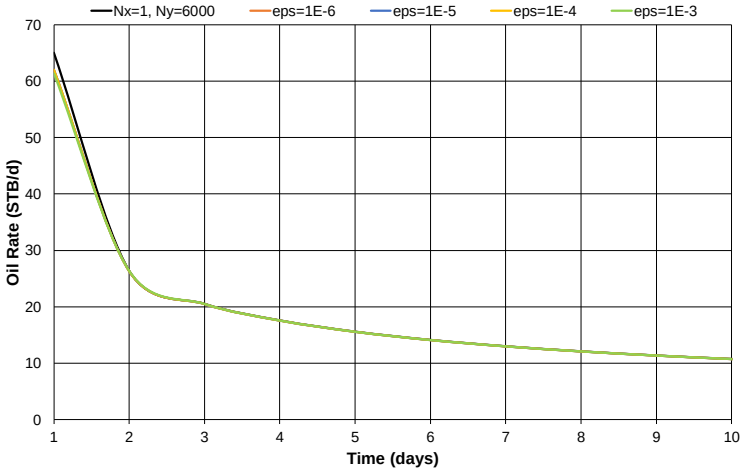
(b) On day 100

Figure A.12: Estimated cumulative oil volume vs. N_y for $N_x = 1$ and $N_y = 6000$

A.4 Estimated Oil Rate vs. Time for Different Threshold Values



(a) Estimated oil rate vs. time for all threshold values



(b) Estimated oil rate vs. time for the most accurate threshold values

Figure A.13: Estimated oil rate vs. time for different threshold for the first 10 days

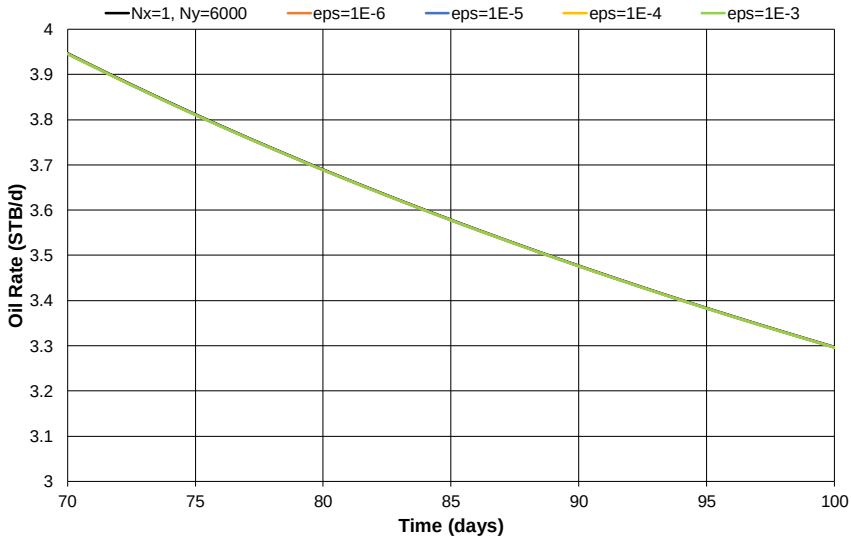
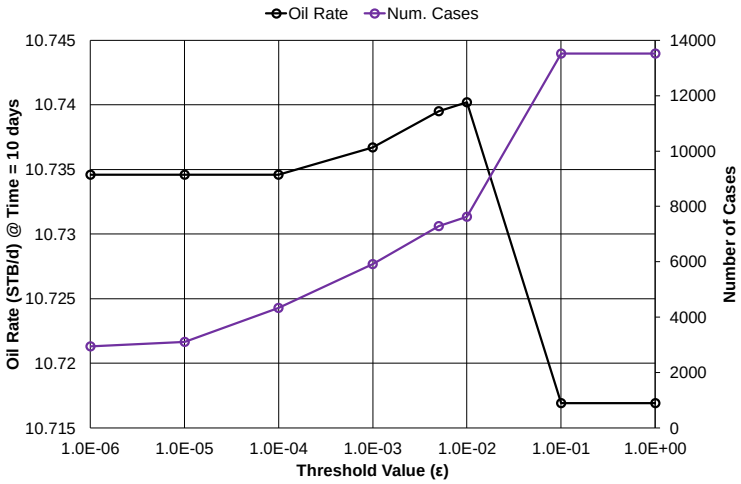
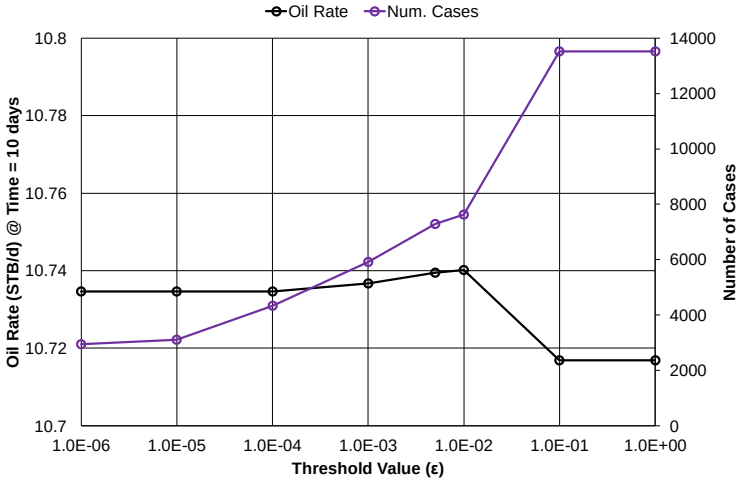


Figure A.14: Estimated oil rate vs. time for different threshold values from day 70 to 100

A.5 Oil Rate and Number of Cases vs. Threshold Value

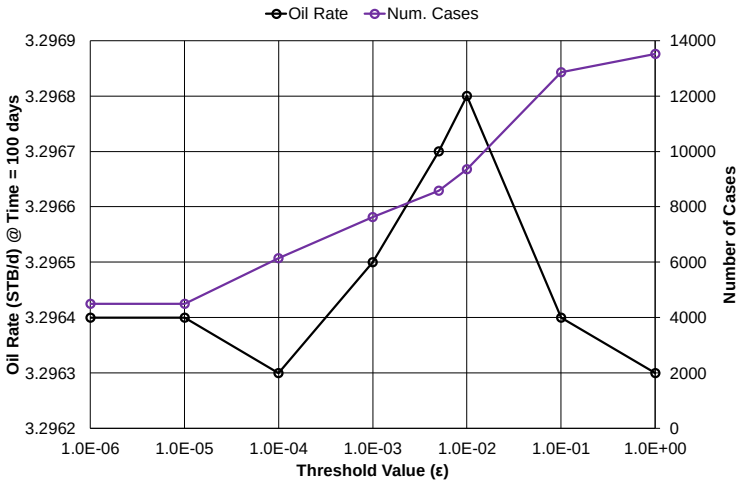


(a) Time = 10 Days

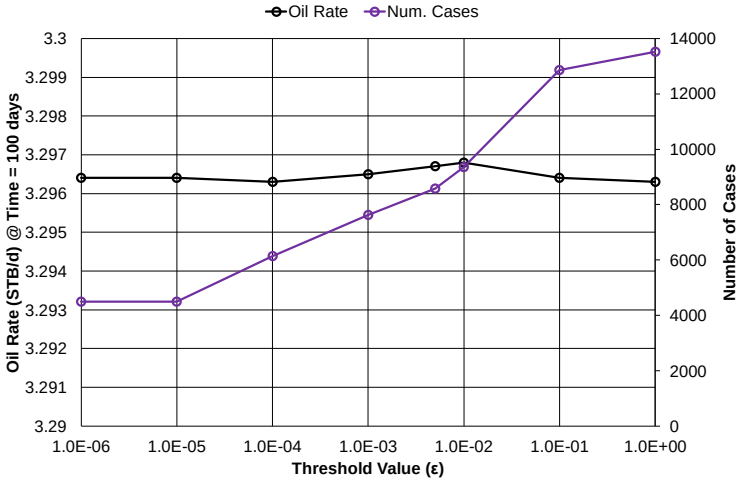


(b) Time = 10 Days, larger scale on primary vertical axis

Figure A.15: Estimated oil rate and number of cases used to get the estimated rate for different threshold values on day 10



(a) Time = 100 days



(b) Time = 100 Days, larger scale on primary vertical axis

Figure A.16: Estimated oil rate and number of cases used to get the estimated rate for different threshold values on day 100

B Python Codes

B.1 main.py

```
# ============ Import FUNCTIONS ==================
from read import Read_Parameters
from Generation import Gen_Basic_Cases
from Generation import Gen_Advanced_Cases
from Generation import Gen_Step_Length
from write import Write_Includes
from runner import Run_Datafile, Run_Sensor_Plot
from saver import Save_possible_Cases
# ================================================

# ==== Extract Parameters ==============
[parameter, value] = Read_Parameters()  # Nmax, xe, xf, xexf, ye, nmax
nmax = int(value[5])

# ===== STEP 1: Generate Basic Cases (Nx,Ny) ================
cases = Gen_Basic_Cases(value)

# ===== STEP 2: Generate Advanced Cases (Nxf,Nxg,Ny) =======
possible_cases = Gen_Advanced_Cases(cases, value, nmax)

# Save possible cases in Excel
Save_possible_Cases(possible_cases)

# ===== STEP 3: Generate dimension vectors (DX, DY) from step lengths
    ======
# grid = [case_1, case_2,....]
# case_i = [DXi,DYi]
# DXi = [dx1(i),dx2(i),...]
# DXi = [dxf1(i),dfx2(i),...,dxf_Nxf(i),dxg1(i),dxg2(i),...,dxg_Nxg(i)]
# DYi = [dy1(i),dy2(i),...]
grid = Gen_Step_Length(possible_cases, value)

# ==== STEP 4: write include files to Sensor (inc.)
    ========================
# DELX.inc and DELY.inc
Write_Includes(grid)

# === STEP 5: write Sensor.dat files for each case (.dat)
    =================
# run_simulation.dat
Run_Datafile(possible_cases)

#  == STEP 6: read fort.61 and convert to (.tab) and (.plt) files for
    plotting purposes
Run_Sensor_Plot(possible_cases)
```

B.2 read.py

```
# ============ IMPORT MODULES ===========
import xlrd
import os
import numpy as np
import pandas as pd
# ======================================

def Read_Parameters():

    loc_file = 'InputExcel/Parameters.xlsx'  # Nmax, xe, xf, xexf, ye

    #  Finding the right workbook and worksheet
    wb = xlrd.open_workbook(loc_file)
    ws = wb.sheet_by_name('Info')
    # for extracting multiple rows at a time
    parameter = [0]*6
    value = [0]*6
    for i in range(6):
        parameter[i] = ws.cell(0, 0+i).value
        value[i] = ws.cell(1, 0+i).value

    return [parameter, value]

# ===== Read data from DATA.tab ==================
def Extract_Data():

    cases = []

    for d in os.listdir("Sensor-files/Output"):
        check = 0
        time = []
        qoil = []
        qgas = []
        cumoil = []
        cumgas = []
        pbh = []
        with open("Sensor-files/Output/" + d + "/DATA.tab", "r") as file:
            for line in file.readlines():
                if line.find("NAMES") >= 0 and line.find("TIME") >= 0:
                    sec = line.replace("\n", "").split("\t")
                    cnt = 0
                    for header in sec:
                        if header == "TIME":
                            line_nr_time = cnt
                        elif header == "QOIL":
                            line_nr_qoil = cnt
                        elif header == "QGAS":
                            line_nr_qgas = cnt
                        elif header == "CUMOIL":
                            line_nr_cumoil = cnt
                        elif header == "CUMGAS":
                            line_nr_cumgas = cnt
                        elif header == "PBH":
                            line_nr_pbh = cnt
                        cnt += 1
```

```
                    check = 3
                if check == 1:
                    if line.find(".") >= 0:
                        sec = line.replace("\n", "").split("\t")
                        time.append(float(sec[line_nr_time]))
                        qoil.append(float(sec[line_nr_qoil]))
                        qgas.append(float(sec[line_nr_qgas]))
                        cumoil.append(float(sec[line_nr_cumoil]))
                        cumgas.append(float(sec[line_nr_cumgas]))
                        pbh.append(float(sec[line_nr_pbh]))
                    else:
                        break
                elif check > 1:
                    check -= 1
            data = np.zeros((len(time), 6))
            for i in range(len(time)):
                data[i, 0] = time[i]
                data[i, 1] = qoil[i]
                data[i, 2] = qgas[i]
                data[i, 3] = cumoil[i]
                data[i, 4] = cumgas[i]
                data[i, 5] = pbh[i]
            cols = ["TIME", "QOIL", "QGAS", "CUMOIL", "CUMGAS", "PBH"]
            data = pd.DataFrame(data=data, columns=cols)
            cases.append(pd.DataFrame.copy(data))

    # extract grid combination
    grid = pd.read_excel("Sensor-runs-figures/Possible_Cases.xlsx")
    grid = grid.iloc[:,1:]


    return cases, grid
```

B.3 generation.py

```
# ============ IMPORT MODULES ===========
import random
import numpy as np
# ======================================

# ==== step 1: Generate (Nx,Ny) cases ================
def Gen_Basic_Cases(value):
    # Find xexf
    xexf = value[3]

    Nmax = int(value[0])
    cases = []
    # create a square that is Nmax*Nmax containing only zeros
    square = np.zeros((Nmax, Nmax))
    for i in range(1, 11, 1):
        for j in range(2, 2000+1, 1):
            if i * j <= Nmax:
                square[i-1, j - 1] = 1
                if xexf != 1 and i != 1:
                    cases.append([i, j])
                elif xexf == 1:
                    cases.append([i, j])

    return cases

# ==== Step 2: Generate (Nxf, Nxg ,Ny) cases ========
def Gen_Advanced_Cases(cases, value, nmax):

    xexf = value[3]

    # x = cases[vector][element in vector]
    possible_cases = []

    length_array = len(cases)
    for i in range(0, length_array):
        if xexf == 1:
            Nxf = cases[i][0]
            Nxg = 0
            Ny = cases[i][1]
            possible_cases.append([Nxf, Nxg, Ny])
        else:
            Nx = cases[i][0]
            Ny = cases[i][1]
            possible_cases = \
                Find_Possible_Cases(Nx, Ny, nmax, possible_cases)

    return possible_cases

# ==== Find possible cases of Nxf and Nxg if xexef != 1 ====
def Find_Possible_Cases(Nx, Ny, nmax, possible_cases):

    vec = [0]*(Nx-1)
    for ii in range(Nx-1):
        Nxf = ii + 1
        Nxg = Nx - Nxf
```

```
        vec[ii] = [Nxf, Nxg, Ny]

    if len(vec) > nmax:
        limit_reached = 1
    else:
        limit_reached = 0

    if limit_reached == 1:
        # Pick nmax random numbers given that nmax limit is reached
        r = random.sample(range(1, len(vec)+1), nmax)
        for ii in range(nmax):
            possible_cases.append([r[ii], Nx-r[ii], Ny])
    else:
        for ii in range(len(vec)):
            possible_cases.append(vec[ii])

    return possible_cases

# ===== Step 3: Generate DX and DY ==============
def Gen_Step_Length(possible_cases, value):

    xe = value[1]
    xf = value[2]
    xg = xe-xf
    xexf = value[3]
    ye = value[4]
    length_possible_cases = len(possible_cases)
    grid = [0]*length_possible_cases

    for i in range(length_possible_cases):
        # pre allocation of DX and DY for case i
        DX = [0]*(possible_cases[i][0]+possible_cases[i][1])
        DY = [0]*possible_cases[i][2]

        dxf = xf/possible_cases[i][0]
        if possible_cases[i][1] == 0:
            dxg = 0
        else:
            dxg = xg/possible_cases[i][1]

        dy = ye/(possible_cases[i][2]-1)

        # fill DX vector
        for j in range(possible_cases[i][0]):
            DX[j] = dxf
        if dxg != 0:
            for k in range(j+1, j+1 + possible_cases[i][1]):
                DX[k] = dxg

        # fill DY vector
        for j in range(possible_cases[i][2]):
            if j == 0:
                DY[j] = 0.1
            else:
                DY[j] = dy
        case = [DX, DY]
        grid[i] = case
```

```
    return grid
```

B.4 write.py

```

# Wite DELX.inc and DELY.inc
def Write_Includes(grid):

    N = len(grid)  # number of cases in grid
    for i in range(N):
        # write DELX.inc file
        sx = ""
        sx += "DELX XVAR \n"
        NX = len(grid[i][0])
        for j in range(NX):
            sx += str(grid[i][0][j]) + " \n"

        num = Get_Num_Str(i,6)
        filename = num + "-DELX.inc"
        with open("Sensor-files/Includes-DELX/" + filename, "w") as file:
            file.write(sx)

        # write DELY.inc file
        sy = ""
        sy += "DELY YVAR \n"
        NY = len(grid[i][1])
        for j in range(NY):
            sy += str(grid[i][1][j]) + " \n"

        filename = num + "-DELY.inc"
        with open("Sensor-files/Includes-DELY/" + filename, "w") as file:
            file.write(sy)

def Get_Num_Str(i,n):
    # i - case number
    # n - number of digits
    # example: i = 145, n = 5 --> "00145"
    if i >= 10**n:
        print("case nr contains more digits than specified")
        out = str(i)
    else:
        s_a = ["0"]*n
        s_b = list(str(i))
        for i in range(len(s_b)):
            s_a[-1-i] = s_b[-1-i]
        out = "".join(s_a)

    return out

def Write_Title(i):
    sg = ""
    sg += "TITLE \n"
    sg += "Case number" + " " + str(i) + "\n"
    sg += "ENDTITLE \n \n"

    return sg
```

```
53
54 # Write input datafile to Sensor
55 def Write_Datafile(possible_cases, case_nr):
56
57     case_nr_txt = Get_Num_Str(case_nr, 6)
58
59     # 1. Title and end-title
60     sg = Write_Title(case_nr)
61
62     # Initial data
63     with open("Templates/temp_00.txt", "r") as file:
64         sg += file.read()
65
66     # 2.  Grid properties
67     sg += "\n \n"
68     sg += "C ***************************** \n"
69     sg += "C Gridding and keyword printout \n"
70     sg += "C ***************************** \n"
71     sg += "\n"
72     sg += "C       NX        NY       NZ \n"
73     sg += "C   --------   -------  -------- \n"
74     sg += "GRID    "
75     sg += str(possible_cases[0]+possible_cases[1]) + "\t" \
76           + str(possible_cases[2]) + "\t" + str(1) + "\n"
77     sg += "C   --------   -------  -------- \n"
78     sg += "\n"
79     sg += "KEYWORD \n"
80     sg += "\n"
81
82     # 3. Equation formulation and solver
83     with open("Templates/temp_01.txt", "r") as file:
84         sg += file.read()
85
86     # 4. Model properties
87     sg += "\n \n"
88     sg += "C ***************** \n"  # C stands for comments
89     sg += "C Model properties \n"
90     sg += "C ***************** \n"
91     sg += "\n"
92     sg += "C Delta x (ft) \n"
93     sg += "INCLUDE \n"
94     sg += "Includes-DELX\{}-DELX.inc \n".format(case_nr_txt)
95     sg += "\n"
96     sg += "C Delta y (ft) \n"
97     sg += "INCLUDE \n"
98     sg += "Includes-DELY\{}-DELY.inc \n".format(case_nr_txt)
99     sg += "\n"
100    sg += "C Layer thickness \n"
101    sg += "THICKNESS CON \n"
102    sg += "150 \n"
103    sg += "\n"
104
105    # 5. Reservoir properties
106    sg += "C ******************** \n"
107    sg += "C Reservoir properties \n"
108    sg += "C ******************** \n"
109    sg += "\n"
```

```
    sg += "C Porosity (-) \n"
    sg += "POROS CON \n"
    sg += "0.05 \n"
    sg += "MOD \n"
    sg += "C I1 I2 J1 J2 K1 K2 \n"  # where I2 is Nxf
    I1 = (possible_cases[0])
    sg += "1" + " " + str(I1) + " " + "1 1 1 1 = 0.1 \n"
    sg += "\n"

    sg += "C Permeability (md) \n"
    sg += "KX CON \n"
    sg += "200E-06 \n"
    sg += "MOD \n"
    I1 = (possible_cases[0])
    sg += "1" + " " + str(I1) + " " + "1 1 1 1 = 100 \n"
    sg += "\n"
    sg += "KY EQUALS KX \n"
    sg += "KZ EQUALS KX \n"
    sg += "\n"
    sg += "C Depth to middle of reservoir (ft) \n"
    sg += "DEPTH CON CENTER \n"
    sg += "9000 \n"
    sg += "\n"

    # 6.  Water and rock properties
    with open("Templates/temp_02.txt", "r") as file:
        sg += file.read()

    # Rock type
    sg += "\n"
    sg += "ROCKTYPE CON \n"
    sg += "1 \n"
    sg += "MOD \n"
    I1 = (possible_cases[0])
    sg += "1" + " " + str(I1) + " " + "1 1 1 1 = 2 \n"
    sg += "\n"

    # 7. Relative permeability curves, fluid model,
    # model initialization & printout sequence
    with open("Templates/temp_03.txt", "r") as file:
        sg += file.read()

    # 8. Well definitions
    sg += "C ***************** \n"
    sg += "C Well definitions \n"
    sg += "C ***************** \n"
    sg += "\n"
    sg += "C WELL PLACEMENT \n"
    sg += "WELL \n"
    sg += "I1 I2 J1 J2 K1 K2 PI \n"
    sg += "PRODUCER \n"
    I1 = (possible_cases[0])
    sg += "1" + " " + str(I1) + " " + "1 1 1 1 1 \n"
    sg += "\n"
    sg += "C Well type \n"
    sg += "WELLTYPE \n"
    sg += "PRODUCER STBOIL \n"
```

```
    sg += "\n"

    # 9. Time step control, well scheduling and END
    with open("Templates/temp_04.txt", "r") as file:
        sg += file.read()

    filename = "run_simulation.dat"
    with open("Sensor-files/" + filename, "w") as file:
        file.write(sg)


def Write_Sensor_Plot(path):
    #  DATA is the output name for the file fort.61
    # "path" goes into the Output folder (where all the case folders are)

    abs_path = "Sensor-files"

    file = open("Sensor-files/data_reader.dat", "w")
    file.write("TITLE \n")
    file.write("read output from Sensor \n")
    file.write("ENDTITLE \n \n")

    file.write("FILE \n")
    file.write("{}\{} run \n \n".format(path, "fort.61"))
    file.write("OUTPUTNAME \n")
    file.write("{}\{} \n \n".format(path, "DATA"))

    file.write("WELL PRODUCER \n")
    file.write("QOIL PBH \n")
    file.write("END")
```

B.5 runner.py

```
1
2  # ====== Import FUNCTIONS ====================================
3  from write import Write_Datafile, Get_Num_Str, Write_Sensor_Plot
4  # ============================================================
5
6  # ========= Import MODULES # ============================
7  import subprocess
8  from os import system, remove
9  # ============================================================
10
11
12 def Run_Datafile(possible_cases):
13
14     for i in range(len(possible_cases)):
15         # write and save .dat file
16         Write_Datafile(possible_cases[i], i)
17
18         # Run run_simulation.dat file and create a "Output" folder
       containing folders for each case
19         abs_path = "Sensor-files"
20         filename = "run_simulation.dat"
21         output_filename = "output.out"
22
23         system("cd {} & sensor {} {}".format(abs_path, filename,
       output_filename))
24         # cd - call directory and md - make new directory in the current
       directory = Output
25         system("cd {} & cd Output & md {}".format(abs_path, Get_Num_Str(i,
        6)))
26
27         system("cd {} & copy fort.61 Output\{}".format(abs_path,
       Get_Num_Str(i, 6)))
28
29         print("============================================================"
       )
30         print("Case number: " + str(i) + " of " + str(len(possible_cases))
       )
31         print("============================================================"
       )
32
33 def Run_Sensor_Plot(possible_cases):
34
35     for i in range(len(possible_cases)):
36         path = "Sensor-files\Output\{}".format(Get_Num_Str(i, 6))
37         abs_path = "Sensor-files\data_reader.dat\n"
38         Write_Sensor_Plot(path)
39         subprocess.run(["sensorplot"], stdout=subprocess.PIPE, text=True,
       input=abs_path)
40         remove("Sensor-files\Output\{}\{}".format(Get_Num_Str(i, 6), "fort
       .61"))
```

B.6 main_extract.py

```
1
2  # === Import MODULES ==========================
3  from read import Extract_Data
4  from saver import Save_Raw_Data
5  from extracter import Extract_Correct_Result
6  from saver import Save_Hist
7  # =============================================
8
9  # Extract data from DATA.tab for all cases and return as list of pandas
       dataframes
10
11 data, grid = Extract_Data()
12
13
14 # Save sorted data
15 qoil, qgas, cumoil, cumgas = Save_Raw_Data(data)
16
17 # Weigh and estimate "correct" solution
18 Nmax = grid.iloc[:, -1].max()
19 time = data[0].iloc[:, 0]
20 max_time = time.max()
21 time, correct_results, histograms = Extract_Correct_Result(qoil, Nmax,
       time, max_time)
22
23 filename = "qoil-Nmax-" + str(int(Nmax)) + ".xlsx"
24 # Save histogram data and "correct" results
25 Save_Hist(filename, time, correct_results, histograms)
```

B.7 saver.py

```python
# ==== Import MODULES =================
import pandas as pd
import numpy as np
import openpyxl
# =====================================

# Save QOIL, QGAS, CUMOIL, CUMGAS and PBH to Required_Data.xlsx
def Save_Raw_Data(data):
    num_times = len(data[0]["TIME"])
    num_cases = len(data)
    qoil_data = np.zeros((num_times, num_cases + 1))
    qgas_data = np.zeros((num_times, num_cases + 1))
    cumoil_data = np.zeros((num_times, num_cases + 1))
    cumgas_data = np.zeros((num_times, num_cases + 1))

    headers = [0] * (num_cases + 1)
    headers[0] = "TIME"

    i = 0
    qoil_data[:, 0] = data[0]["TIME"].to_numpy()
    qgas_data[:, 0] = data[0]["TIME"].to_numpy()
    cumoil_data[:, 0] = data[0]["TIME"].to_numpy()
    cumgas_data[:, 0] = data[0]["TIME"].to_numpy()

    for case in data:
        i += 1
        headers[i] = "case-" + str(i - 1)
        qoil_data[:, i] = case["QOIL"].to_numpy()
        qgas_data[:, i] = case["QGAS"].to_numpy()
        cumoil_data[:, i] = case["CUMOIL"].to_numpy()
        cumgas_data[:, i] = case["CUMGAS"].to_numpy()


    # make pandas dataframes
    qoil = pd.DataFrame(data=qoil_data, columns=headers)
    qgas = pd.DataFrame(data=qgas_data, columns=headers)
    cumoil = pd.DataFrame(data=cumoil_data, columns=headers)
    cumgas = pd.DataFrame(data=cumgas_data, columns=headers)


    #  create a separate spreadsheet for each output variable
    writer = pd.ExcelWriter('OutputExcel/Required_Data.xlsx')
    qoil.to_excel(writer, sheet_name="QOIL", index=False)
    qgas.to_excel(writer, sheet_name="QGAS",index=False)
    cumoil.to_excel(writer, sheet_name="CUMOIL",index=False)
    cumgas.to_excel(writer, sheet_name="CUMGAS",index=False)

    writer.save()
    print('DataFrame is written successfully to Excel File.')

    return qoil,qgas,cumoil,cumgas

def Save_possible_Cases(possible_cases):

    # length in x-direction: Nxf, Nxg, Ny
```

```
    num_element = len(possible_cases[0])
    # length in y-direction: the amount of cases
    num_cases = len(possible_cases)
    possible_cases_data = np.zeros((num_cases, num_element+1))
    headers = [0]*(num_element+1)
    headers[0] = "Case"
    headers[1] = "Nxf"
    headers[2] = "Nxg"
    headers[3] = "Ny"

    for i in range(num_cases):
        possible_cases_data[i][0] = str(i)
        possible_cases_data[i][1] = possible_cases[i][0]
        possible_cases_data[i][2] = possible_cases[i][1]
        possible_cases_data[i][3] = possible_cases[i][2]

    possible_cases_excel = pd.DataFrame(data=possible_cases_data, columns=
    headers)
    writer = pd.ExcelWriter("Sensor-runs-figures/Possible_Cases.xlsx")
    possible_cases_excel.to_excel(writer, sheet_name="Possible-cases",
    index=False)
    writer.save()

def Save_Hist(filename, time, correct_results, histograms):
    num_time = len(time)

    dx = 2
    dy = 2
    wb = openpyxl.Workbook()
    sheet = wb.create_sheet("data")

    for i in range(num_time):
        sheet.cell(dy+i+1, dx, time[i])
        sheet.cell(dy+i+1, dx+1, correct_results[i])
        if len(histograms[i]) != 0:
            for j in range(len(histograms[i])):
                sheet.cell(dy+i+1, dx+2+j, histograms[i][j])

    sheet.cell(dy, dx, "Time")
    sheet.cell(dy, dx+1, "c_data")
    sheet.cell(dy, dx+2, "hist_data")

    wb.save(filename)
```

B.8 extracter.py

```
1
2  # ===== Import MODULES ===============
3  import numpy as np
4  import matplotlib.pyplot as plt
5  # ====================================
6
7  def Extract_Correct_Result(raw_results, Nmax, time, max_time):
8      result_matrix = np.zeros((Nmax, Nmax))
9      correct_rate = [0] * len(time)
10     time = []
11     q = []
12     histograms = []
13
14     # 1. Create a square (Nmax, Nmax) with 1's for possible results when
       Ntot <= Nmax
15     for t in range(1, len(correct_rate)+1):
16         print("
       ================================================================================
       ")
17         print("Time-step Number: " + str(t))
18         print("
       ================================================================================
       ")
19         possible_cases_matrix = np.zeros((Nmax, Nmax))
20         cnt = 0
21         for i in range(Nmax):
22             for j in range(1, Nmax):
23                 if (i + 1) * (j + 1) <= Nmax:
24                     possible_cases_matrix[j, i] = 1
25                     result_matrix[j, i] = raw_results.iloc[t - 1, cnt + 1]
26                     if raw_results.iloc[t - 1, cnt + 1] == 0:
27                         possible_cases_matrix[j, i] = 0
28                     cnt += 1
29
30         # 2. Creating the matrix with the derivatives, dij
31         dij = np.zeros((Nmax, Nmax))
32         check = 1
33         for i in range(Nmax):
34             for j in range(Nmax):
35                 # Must remove Ny = 1 row, and the points where i+1 and j+1
        is not defined (equal to zero)
36                 if possible_cases_matrix[j, i] == 1 and i != Nmax-1 and j
       != Nmax-1:
37                     if possible_cases_matrix[j, i+1] != 0 and
       possible_cases_matrix[j + 1, i] != 0:
38                         # Apply forward difference on the dij matrix:
39                         dij[j, i] = abs(result_matrix[j + 1, i] -
       result_matrix[j, i]) + abs(
40                             result_matrix[j, i + 1] - result_matrix[j, i])
41
42         # The lowest dij value ( of those who are NOT zero)
43         length_dij = len(dij)
44         dij_min = 1000
45
46         # 3. The position (Nx,Ny) that gives dij_min
```

```
        for row in range(length_dij):
            for col in range(length_dij):
                if dij[row, col] < dij_min and dij[row, col] != 0:
                    dij_min = np.copy(dij[row, col])
                    ref_rate = np.copy(result_matrix[row, col])

        # 5. Create weight_matrix that is scaled by multiplying dij with
    the reference rate
        norm_dij = np.zeros((Nmax, Nmax))
        if ref_rate != 0:
            norm_dij = dij*(1 / ref_rate)
        else:
            norm_dij = dij
            print("Reference rate found was equal to zero")

        # 6. Logic if normalised dij is < eps, then WF = 1, if not 0
        weight_matrix = np.zeros((Nmax, Nmax))

        combinations = []
        for i in range(length_dij):
            for j in range(length_dij):
                if possible_cases_matrix[j, i] == 1:
                    if norm_dij[j, i] <= 1E-6:
                        weight_matrix[j, i] = 1
                        combinations.append([i, j])

        num_cases = len(combinations)
        print("Number of datapoints: " + str(num_cases))
        output = np.zeros((num_cases, 2))
        X = [0] * num_cases
        cnt = 0
        for cmb in combinations:
            output[cnt, 0] = result_matrix[cmb[1], cmb[0]]
            output[cnt, 1] = weight_matrix[cmb[1], cmb[0]]
            X[cnt] = cnt + 1
            cnt += 1
        # Estimate "correct" oil rate
        if not np.isnan(np.average(output[:, 0])):
            time.append(t)
            # q.append(np.average(output[:, 0]))
            q.append(np.median(output[:, 0]))
            histograms.append(output[:, 0])
            correct_rate[t - 1] = np.median(output[:, 0])
            test =1
        # plt.scatter(time,correct_rate)
    # plt.plot(time, q)
    plt.hist(output[:, 0])
    plt.xlabel("Result Value")
    plt.ylabel("Frequency")
    plt.show()
    correct_results = q

    return time, correct_results, histograms
```

B.9 main_functions.py

```
# This program reads grid data from Excel and estimates most probable "
    correct" solution

from reader_data import Read_Excel_Data
from functions_tensor import Fill_Data_Tensor
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# (1) Extract data from "data-1.5.xlsx"
data, Nxf, Nxg, Ny, time, Nt = Read_Excel_Data()

# (2) Pre-allocate grid tensor
data_tensor = np.zeros([Nxf, Nxg, Ny])

# (3) Fill data_tensor In(Grid_Structure,Case_Data,Allocated_Data_Tensor)
    => Out(Filled_Data_Tensor)

# TODO: Change derivative threshold here!!
err = 0.1
correct_results = np.zeros(Nt)
for i in range(Nt):
    print("
    ======================================================================")
    print("Time-step number: "+str(i))
    print("
    ======================================================================")
    weighted_results = Fill_Data_Tensor(Nxf, Nxg, Ny, data.iloc[:, 0:3],
    data.iloc[:, 3+i], data_tensor, err,time)
    if len(weighted_results) == 0:
        print("None of the data met the criteria for the derivative at
    time-step number: "+str(time[i]))
    else:
        correct_results[i] = weighted_results.iloc[:, 3].median()
        plt.hist(weighted_results.iloc[:, 3])
        plt.show()

s = ""
for c in correct_results:
    s += str(c) + " \n"
with open("err-1-data.txt","w") as file:
    file.write(s)

test = 1
```

B.10 reader_data.py

```
import pandas as pd
import numpy as np

def Read_Excel_Data():

    data = pd.read_excel("data-1.5.xlsx", sheet_name="Possible-cases")
    data = data.iloc[:, 1:]

    Nxf = max(data.iloc[:, 0])
    Nxg = max(data.iloc[:, 1])
    Ny = max(data.iloc[:, 2])

    time = np.asarray(list(data.columns)[3:])
    Nt = len(time)

    return data, Nxf, Nxg, Ny, time, Nt
```

B.11 functions_tensor.py

```
import numpy as np
import pandas as pd

def Fill_Data_Tensor(Nxf,Nxg,Ny,grid_structure,case_data,data_tensor,err,
    time):
    weighted_results = []
    possible_cases = np.copy(data_tensor)
    diff_tensor = np.copy(data_tensor)


    # Fill matrix
    cnt = 0
    for case in case_data:
        i = grid_structure.iloc[cnt, 0] - 1  # Position in Nxf
        j = grid_structure.iloc[cnt, 1] - 1  # Position in Nxg
        k = grid_structure.iloc[cnt, 2] - 1  # Position in Ny

        data_tensor[i, j, k] = case
        if case != 0:
            possible_cases[i, j, k] = 1

        cnt += 1

    # Calculate weights and extract weighted results
    cnt = 0
    for case in case_data:
        i = grid_structure.iloc[cnt, 0] - 1  # Position in Nxf
        j = grid_structure.iloc[cnt, 1] - 1  # Position in Nxg
        k = grid_structure.iloc[cnt, 2] - 1  # Position in Ny

        if i < Nxf-1 and j < Nxg-1 and k < Ny-1:
            test1 = possible_cases[i, j, k] != 0        # Is a possible
    case
            test2 = possible_cases[i + 1, j, k] != 0    # Next Nxf is a
    possible case
```

```
            test3 = possible_cases[i, j + 1, k] != 0    # Next Nxg is a
    possible case
            test4 = possible_cases[i, j, k + 1] != 0    # Next Ny is a
    possible case

            if test1 and test2 and test3 and test4:
                # forward difference in i, j and k- direction (Nxf, Nxg
    and Ny)
                diff1 = abs(data_tensor[i + 1, j, k] - data_tensor[i, j, k
    ])
                diff2 = abs(data_tensor[i, j + 1, k] - data_tensor[i, j, k
    ])
                diff3 = abs(data_tensor[i, j, k + 1] - data_tensor[i, j, k
    ])

                # The derivative in a point (Nxf, Nxg, Ny)
                diff = diff1+diff2+diff3

                diff_tensor[i, j, k] = diff

        cnt += 1
    diff_min = 1000
    ref_rate = 1000

    # 3. The position of  (Nxf, Nxg, Ny) that gives diff_min
    for row in range(Nxf):
        for col in range(Nxg):
            for elem in range(Ny):
                if diff_tensor[row, col, elem] < diff_min and data_tensor[
    row, col, elem] != 0:
                    diff_min = np.copy(diff_tensor[row, col, elem])
                    ref_rate = np.copy(data_tensor[row, col, elem])

    norm_diff = np.zeros([Nxf, Nxg, Ny])
    if ref_rate != 0:
        norm_diff = diff_tensor * (1 / ref_rate)
    else:
        norm_diff = diff_tensor
        print("Reference rate found was equal to zero")

    cnt = 0
    for case in case_data:
        i = grid_structure.iloc[cnt, 0] - 1  # Position in Nxf
        j = grid_structure.iloc[cnt, 1] - 1  # Position in Nxg
        k = grid_structure.iloc[cnt, 2] - 1  # Position in Ny

        if i < Nxf - 1 and j < Nxg - 1 and k < Ny - 1:
            test1 = possible_cases[i, j, k] != 0  # Is a possible case
            test2 = possible_cases[i + 1, j, k] != 0  # Next Nxf is a
    possible case
            test3 = possible_cases[i, j + 1, k] != 0  # Next Nxg is a
    possible case
            test4 = possible_cases[i, j, k + 1] != 0  # Next Ny is a
    possible case

            if test1 and test2 and test3 and test4:
                if norm_diff[i, j, k] <= err:
```

```
                    weighted_results.append([i+1, j+1, k+1, case])

    data = np.zeros((len(weighted_results), 4))
    for i in range(len(weighted_results)):
        data[i, 0] = weighted_results[i][0]
        data[i, 1] = weighted_results[i][1]
        data[i, 2] = weighted_results[i][2]
        data[i, 3] = weighted_results[i][3]

    output = pd.DataFrame(data=data, columns=["Nxf", "Nxg", "Ny", "Result"
    ])

    return output
```

C Templates

C.1 Template 00 - Initial data frame

```
C ========================================
C                 INITIAL DATA
C ========================================
```

C.2 Template 01 - Equation and formulation solver

```

C *********************************
C Equation formulation and solver
C *********************************

IMPLICIT


C ************************
C Output printing
C ************************

C Saturation maps in fort.71 file
MAPSFILE P SO SG SW KX KY POROS PV VISO VISG

C Saturations maps in .out file
MAPSPRINT OFF

C Rel-perm. tables in .out file
PRINTKR
```

C.3 Template 02 - Fluid and rock properties

```

C ***************************
C Water- and Rock Properties
C ***************************

C       B_w         c_w       rho_w      mu_w       c_r        p_ref
C       (-)       (1/psi)   (lbm/ft3)    (cp)     (1/psi)     (psia)
C     --------    --------  --------   --------  --------   --------
MISC    1.0         3E-6       62.40      0.5        4E-6       6000
C     --------    --------  --------   --------  --------   --------
```

C.4 Template 03 - Rel.perm and capllary curves, EOS to BO table conversion

```
1
2
3  KRANALYTICAL 1
4  C   Swc      Sorw     Sorg     Sgc
5  C -------  -------  -------  -------
6      0.0      0.0      0.4      0.1
7  C -------  -------  -------  -------
8
9  C  krwro    krgro    krocw
10 C -------  -------  -------
11     1.0      1.0      1.0
12 C -------  -------  -------
13
14 C   nw       now      ng       nog
15 C -------  -------  -------  -------
16     2.5      2.5      2.5      2.5
17 C -------  -------  -------  -------
18
19
20 KRANALYTICAL 2
21 C   Swc      Sorw     Sorg     Sgc
22 C -------  -------  -------  -------
23     0.0      0.0      0.0      0.0
24 C -------  -------  -------  -------
25
26 C  krwro    krgro    krocw
27 C -------  -------  -------
28     1.0      1.0      1.0
29 C -------  -------  -------
30
31 C   nw       now      ng       nog
32 C -------  -------  -------  -------
33     1.0      1.0      1.0      1.0
34 C -------  -------  -------  -------
35
36
37
38 C ******************************
39 C Fluid Model and Compositions
40 C ******************************
41
42 C EOS to Black-Oil Conversion
43 BLACKOIL 1 20 20 SRK
44 PRESSURES 100 200 400 600 800 1000 1500 2000 2500 3000 3500 4000 4500 5000
       5500 6000 6500 7000 7500 8000
45 RESERVOIR FLUID
46   0.00340000 0.00020000 0.34620000 0.04110000 0.01010000 0.00760000
       0.00490000 0.00430000 0.00210000 0.01610000
47   0.04513067 0.04700263 0.03810575 0.03312375 0.02940223 0.02641002
       0.02390351 0.02174942 0.01986943 0.01821009
48   0.01673359 0.01541062 0.01421763 0.01314129 0.01216546 0.01127701
       0.01046764 0.00972848 0.00905271 0.14889807
49 INJECTION GAS EQUILIBRIUM
50 SEPARATOR
```

```
14.69 60
ENDBLACKOIL

PVTEOS SRK
  250
         CPT          MW           TC           PC           ZCRIT       SHIFT
           AC           OMEGA        OMEGB
         N2           28.014       227.16       492.84      0.29178
    -0.00090     0.03700      0.42748      0.08664
         CO2          44.010       547.42      1069.51      0.27433
    0.21749     0.22500      0.42748      0.08664
         C1           16.043       343.01       667.03      0.28620
    -0.00247     0.01100      0.42748      0.08664
         C2           30.070       549.58       706.62      0.27924
    0.05894     0.09900      0.42748      0.08664
         C3           44.097       665.69       616.12      0.27630
    0.09075     0.15200      0.42748      0.08664
         I-C4         58.123       734.13       527.94      0.28199
    0.10952     0.18600      0.42748      0.08664
         N-C4         58.123       765.22       550.56      0.27385
    0.11028     0.20000      0.42748      0.08664
         I-C5         72.150       828.70       490.37      0.27231
    0.09773     0.22900      0.42748      0.08664
         N-C5         72.150       845.46       488.78      0.26837
    0.11947     0.25200      0.42748      0.08664
         C6           83.567       921.77       475.01      0.26852
    0.13427     0.25065      0.42748      0.08664
         C7           96.713       983.88       439.72      0.26422
    0.14520     0.28587      0.42748      0.08664
         C8          110.382      1038.69       404.83      0.25982
    0.15684     0.32558      0.42748      0.08664
         C9          123.823      1087.90       372.49      0.25562
    0.17443     0.36754      0.42748      0.08664
         C10         137.092      1131.67       344.59      0.25181
    0.19128     0.40884      0.42748      0.08664
         C11         150.296      1171.36       320.28      0.24825
    0.20739     0.44963      0.42748      0.08664
         C12         163.425      1207.59       299.00      0.24488
    0.22266     0.48979      0.42748      0.08664
         C13         176.472      1240.85       280.28      0.24166
    0.23703     0.52928      0.42748      0.08664
         C14         189.431      1271.55       263.75      0.23854
    0.25048     0.56809      0.42748      0.08664
         C15         202.298      1300.00       249.07      0.23552
    0.26300     0.60619      0.42748      0.08664
         C16         215.068      1326.47       235.99      0.23258
    0.27462     0.64359      0.42748      0.08664
         C17         227.740      1351.20       224.29      0.22971
    0.28537     0.68030      0.42748      0.08664
         C18         240.310      1374.37       213.77      0.22691
    0.29530     0.71630      0.42748      0.08664
         C19         252.777      1396.15       204.28      0.22418
    0.30445     0.75162      0.42748      0.08664
         C20         265.139      1416.67       195.70      0.22152
    0.31287     0.78627      0.42748      0.08664
         C21         277.397      1436.07       187.91      0.21892
    0.32061     0.82025      0.42748      0.08664
```

```
          C22          289.551     1454.44      180.82     0.21640
    0.32771    0.85358     0.42748     0.08664
          C23          301.600     1471.87      174.34     0.21394
    0.33423    0.88627     0.42748     0.08664
          C24          313.546     1488.46      168.41     0.21155
    0.34020    0.91834     0.42748     0.08664
          C25          325.389     1504.26      162.96     0.20922
    0.34568    0.94980     0.42748     0.08664
          C26+         381.515     1571.82      141.85     0.19885
    0.36630    1.09433     0.42748     0.08664
BIN
     0.00000  0.02000  0.06000  0.08000  0.08000  0.08000  0.08000
    0.08000  0.08000  0.08000  0.08000  0.08000  0.08000  0.08000  0.08000
      0.08000  0.08000  0.08000  0.08000  0.08000  0.08000  0.08000
    0.08000  0.08000  0.08000  0.08000  0.08000  0.08000  0.08000
     0.12000  0.12000  0.12000  0.12000  0.12000  0.12000  0.12000
    0.12000  0.10000  0.10000  0.10000  0.10000  0.10000  0.10000  0.10000
      0.10000  0.10000  0.10000  0.10000  0.10000  0.10000  0.10000
    0.10000  0.10000  0.10000  0.10000  0.10000  0.10000  0.10000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.01490  0.01732  0.01971  0.02194  0.02403  0.02598  0.02781  0.02950
      0.03108  0.03254  0.03390  0.03517  0.03635  0.03744  0.03846
    0.03941  0.04030  0.04113  0.04190  0.04504
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      0.00000  0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      0.00000  0.00000
```

```
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
      0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
    0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000  0.00000
     0.00000  0.00000  0.00000
     0.00000  0.00000
     0.00000



C **********************
C Model Initialization
C **********************

C Reference depth, saturation pressure, and in-situ composition
INITIAL

C Reference depth (ft) and initial pressure (psia)
ZINIT 9000
PINIT 7500

    DEPTH  PSATBP
C     (ft)  (psia)
C  ------  -------
    9000   1984.96
C  ------  -------

ENDINIT

C ========================================
C RECURRENT DATA
C ========================================

C ********************
C Printout frequence
C ********************

C Summary printing frequency to fort.61 file
```

```
SUMFREQ 1

C Map printing frequency to .out file
MAPSFREQ 1

C Map printing frequency to fort.71 file
MAPSFILEFREQ 1
```

C.5 Template 04 - Time step control and well schedule

```

C ********************
C Time step control
C ********************

C Maximum delta t (D)
DTMAX 1

C CFL 1


C *****************
C Well Scheduling
C *****************

C Bottomhole pressure constraint (psia)
BHP
PRODUCER 2500

C Target rate (STB/D)
RATE
PRODUCER 100000

C Time card (Total time | Delta t) (day)
TIME 100 1

END
```

Recreating Locke-Sawyer Curves with Numerical Reservoir Simulation

NTNU
Norwegian University of
Science and Technology