Master's thesis

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

Sogol Bandekian

Model-Based Gas Lift Optimization for Oil Well Networks Using Python and PROSPER Well Models

Master's thesis in Petroleum Engineering
Supervisor: Milan Stanko
Co-supervisor: Knut Vannes, Lars Imsland
July 2023

NTNU
Norwegian University of Science and Technology

AkerBP

Sogol Bandekian

Model-Based Gas Lift Optimization for Oil Well Networks Using Python and PROSPER Well Models

Master’s thesis in Petroleum Engineering
Supervisor: Milan Stanko
Co-supervisor: Knut Vannes, Lars Imsland
July 2023

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

Acknowledgment

The completion of my Master's thesis marked the final stage of my academic journey at the Norwegian University of Science and Technology (NTNU), which I found to be an excellent experience. This place has enabled the acquisition of abundant new skills and knowledge. I thank NTNU and Norway for endowing me with numerous valuable gifts.

I am deeply grateful to my supervisor, Professor Milan Stanko, for his assistance despite being busy. I want to acknowledge his continuous support, enthusiasm, and helpful comments. Without his motivation, my work would not have advanced, and this thesis would not be the same. The encouragement that he provided helped me boost my confidence while facing technical issues. It was his trust that gave me the freedom to try my bold ideas and finally achieve successful results. I have appreciated his highly competent guidance over the past year.

Expressing my sincere gratitude to my co-supervisors for their help, advice, and stimulating discussions is important to me. I am grateful to Knut Vannes for his time, attention, and guidance during meaningful discussions from day one. Also, I am incredibly thankful for the opportunity to collaborate with Aker BP for my Master's thesis. I would like to express my gratitude to Lars Imsland for providing guidance and assistance throughout my thesis and for always being available to help with any Python programming-related issues.

Lastly, I would like to convey my profound appreciation for the constant love and support I have received from my family, particularly my parents. The love and belief they have in me have been fundamental in keeping my spirits and motivation high throughout my two-year master's degree program.

(This page is left intentionally blank)

Abstract

In the oil industry, continuous flow gas lift is a frequently employed artificial lift technique to boost oil recovery by decreasing bottom hole pressure. By injecting high-pressure gas into the tubing, the oil column can be gasified, resulting in a more efficient production process. Although each oil well can obtain a significant amount of gas to achieve optimal production, the available injection gas is often inadequate because of constraints. Excessive gas injection in a field with a limited gas supply can be costly due to high gas prices and compressing costs, leading to a reduction in revenue. Thus, it becomes essential to distribute the injection gas efficiently among all wells to attain the highest oil production rate for the field. Mathematical optimization can be used to formulate and solve the problem of determining gas lift allocation per well for the purpose of maximizing oil production. By maximizing the nonlinear function that models the total oil production rate for a group of wells, the gas allocation can be optimized. The aim of this project is to optimize the oil production system and gas lift rate within the well networks.

GAP, a software for multiphase network modeling and optimization, utilizes model-based numerical optimization to determine the optimal allocation of gas lift injection for each well in the network. Gas-lift injection rates per well are determined by applying numerical optimization to a production system model, which helps identify the most efficient operational settings. The GAP model comprised six gas-lifted wells, two manifolds, and one separator, all spanning from the subsea to the surface.

Besides GAP, a Python-based tool is developed for performing model-based optimization for gas lift injection. For the proper functioning of this tool, two essential components, namely the model and the optimizer, were required. This tool operates through the following process. PROSPER, software for well modeling and nodal analysis, generated the well-model that portrays the behavior of six wells in a network during gas lift injection. The well models comprised free variables such as gas lift injection, liquid rate, water cut, wellhead pressure, and gas-oil ratio. Additionally, the models contained calculated variables that relied on the free variables, including bottom-hole pressure and temperature. The free variables remained consistent across all six wells in the network, whereas the calculated variables varied.

The optimization for well networks was performed through the utilization of $scipy.optimize.minimize$ and some special algorithms in Python. To perform optimization using well-model files, the initial step involved importing the data from the files into Python. The JSON format was utilized to read and store the necessary data. The code implemented for the purpose of reading well models can effectively handle a diverse range of free and calculated variables. Afterward, a class was defined for performing multidimensional interpolation of data points with a cubic method. The unit conversion methods in this class aligned with Norwegian standards. Before proceeding with gas lift optimization, the function for determining the equilibrium flow rates of each well in the network must be established. To maximize the oil production for each well, the last method used in this tool was finding the optimal values of gas lift injection. By implementing a nonlinear optimization technique that took into account constraints and an objective function, the optimal gas injection rates were determined.

By simulating the tool in Python, the gas lift injection was allocated to maximize oil production while meeting technical and operational constraints. The major features of this tool are its ability to function with multiple well numbers and adjust well inputs without impacting its performance. The results indicate that both Python and GAP optimization methods exhibited similar performance, with negligible deviations between the two models for the majority of the process. Additionally, the findings demonstrate that conducting sensitivity analysis in both GAP and Python can provide a more comprehensive understanding of how to optimize oil production in the field, accounting for factors such as restricted water production and choke configuration.

The presence of numerous wells, coupled with complex operational and technical constraints and the need for choke setting optimization, can result in GAP experiencing prolonged search times or a potential failure to arrive at a solution. Additionally, obtaining access to the GAP software proves challenging, as one must purchase a license to operate it. The Python solver enables gas lift optimization with simple input variable management for users and has the potential for optimization using real well models.

(This page is left intentionally blank)

Table of Contents

List of Figures

List of Tables

Abbreviations

BHP	=	Bottom Hole Pressure
WC	=	Water Cut
IPM	=	Integrated Production Modeling
AOF	=	Absolute Open Flow
PI	=	Productivity Index
GAP	=	General Allocation Package
QGL	=	Gas Lift Injection Rate
QLIQ	=	Liquid Rate
GOR	=	Gas-Oil Ratio
WHP	=	Well Head Pressure
CSV	=	Comma Separated Values
WHT	=	Well Head Temperature
IPR	=	Inflow Performance Relationship
JSON	=	JavaScript Object Notation
SQP	=	Sequential quadratic programming
ESP	=	Electrical Submersible Pump
VLP	=	Vertical Lift Performance
TPR	=	Tubing Performance Relationship
VFP	=	Vertical Flow Performance
WOR	=	Water Oil Ratio
TPR	=	Tubing Performance Relationship
GA	=	Genetic Algorithm
QP	=	Quadratic Programming
SLSQP	=	Sequential Least-Squares Quadratic Programming
LP	=	Linear Programming
NLP	=	Nonlinear Programming
MILP	=	Mixed Integer Linear Programming
MINLP	=	Mixed Integer Nonlinear Programming
CG	=	Conjugate Gradient
LC	=	Lift Curves

Chapter 1

Introduction

An introduction to the master's thesis is presented with background information on production optimization, gas lift method, and gas lift assessment. Subsequently, an explanation of the problem formulation, research objectives, and scope of work will be provided, followed by an outline of the thesis structure.

1.1 Background

The optimization of gas lift is a vital aspect of the oil and gas industry as it assists companies in reaching optimal production efficiency and economic benefits. The optimization problem related to gas lift is a well-known issue, specifically for wells with low or no production. A considerable amount of work has been conducted in this area, including an analysis of single wells, solutions based on networks, and solutions that take both networks and reservoirs into account [49].

The motivation for this thesis is rooted in the concern of AkerBP in Norway, which is deeply interested in gas lift optimization for their fields. AkerBP requires the allocation of gas-lift gas for their system of gas-lifted wells, determining the flow rates of gas to be injected into each well. The aim is to allocate resources in a manner that maximizes oil production while still complying with technical and operational constraints, such as limited gas availability.

To perform this task, model-based numerical optimization is typically employed. This involves utilizing numerical optimization techniques on a production system model to determine the most optimal operational settings. Production optimization involves two distinct components, namely the optimizer and the model, although they may occasionally be combined. Commercial software, such as GAP, is commonly utilized by oil and gas companies for this function. In systems with several complicated operational and technical constraints, numerous wells, and the desire to optimize choke settings, GAP frequently fails to find a solution or requires a significant amount of time to do so.

One option for model-based optimization is to define it in a programming language like Python instead of using commercial software. Building the model requires the use of performance tables produced by the black box simulator, which should be treated as the model. Once the well-modeled data is imported from PROSPER to Python, interpolation between existing data points becomes essential.

It is required to provide some underlying knowledge regarding the principal topic. In order to facilitate comprehension of the thesis topic, relevant backgrounds such as production optimization, gas lift method, and gas lift method assessment are presented prior to a detailed problem formulation.

1.1.1 Production Optimization

Production optimization is a critical element in certain petroleum fields, as it seeks to enhance the efficiency and profitability of production operations. A conventional oil field typically comprises a gathering system, a fluid distribution network, and an injection network. The collection and transportation of fluids from production wells to separation units are conducted by the gathering system. This system is made up of a flowline network and processing facilities that facilitate the transfer of oil and gas from wells to a temporary storage facility. Figure 1.1 illustrates the fluid flow in a subsea petroleum production plant, which follows a path starting from wells and ending at the topside separator via manifolds and flow lines. The separated fluids are then distributed to different locations for storage, sale, disposal, injection, or further processing. The primary function of the injection network is to inject fluids into the reservoir, intending to improve oil recovery projects or dispose of or store fluids. In petroleum fields, production optimization is a term used to describe the various activities that aim to improve productivity through measurement, analysis, modeling, prioritization, and implementation. In production optimization, the main principle is to introduce small, cost-effective changes to improve the production system [2].

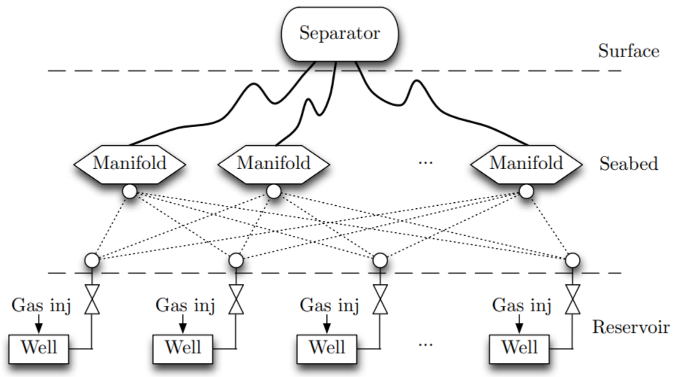


Figure 1.1: Cluster Topology of a Well-Flowline Network [47]

Dynamic production optimization involves optimizing operational settings to achieve specific objectives while considering all constraints, like maximizing daily oil rates or minimizing production costs. To maximize production efficiency by properly allocating the capacity of the field, it is necessary to implement strategies such as developing new wells, utilizing well intervention techniques, modifying the completion process, or implementing artificial lift methods that enable optimal well conditions.

Gas lift injection is the only artificial lift method considered in this study, despite other methods such as Sucker Rod pumps, Plunger lifts, Progressive Cavity pumps, and Hydraulic pumps. Using gas lift is a common practice in production optimization, which involves injecting gas into a well to reduce fluid density and hydrostatic pressure, thus making it easier for fluid to flow toward the surface (Figure 1.2). To achieve reliable production levels in mature fields and wells with less-than-ideal design, gas lift is employed while considering multiple constraints associated with the system handling capacity of the field.

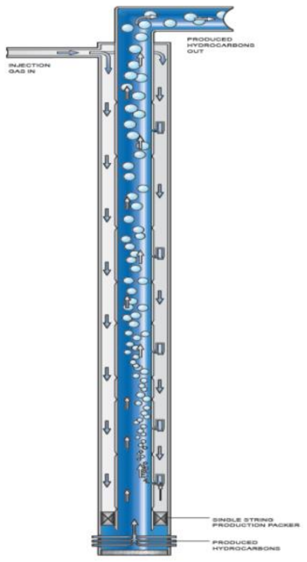


Figure 1.2: Continuous Gas Lift; Gas is Pumped Down the Annulus and into the Tubing [48]

The oil and gas industry must cope with complicated operations due to the exhaustion of easily accessible oil reserves. This involves extracting oil from challenging reservoirs with heavy and viscous oil, using advanced well completions, and implementing subsea processing and tie-backs. Consequently, decision-making and planning have become increasingly challenging. The optimization of decision-making processes in the oil industry through the integration of methodologies and models from diverse disciplines is known as integrated production optimization. To address this matter, industry professionals employ software tools such as PROSPER and GAP during upstream oil and gas operations, which can address non-linear optimization challenges.

1.1.2 Gas Lift Method

By utilizing the gas lift technique, production rates can be considerably increased and the lifespan of a well can be prolonged. High-pressure gas is injected continuously or intermittently at designated points during the gas lift process. Continuous gas lift is a production technique utilized for depleted and mature reservoirs that are no longer capable of producing using their natural energy. In a continuous gas lift operation, gas is injected into the tubing-casing annulus to reduce hydrostatic pressure in the producing fluid column. The density of crude oil is reduced when gas bubbles are infused, leading to a boost in drawdown and oil production. Efficient gas lift optimization is crucial to ensuring high oil production, as excessive gas injection can lead to a reduction in production rates and an increase in operational expenses.

As part of the gas lift procedure, it is necessary to treat, compress, and distribute the gas that comes from the production separator to the assigned production wells. Gas is supplied by the compressor and enters the production tubing through valves, where it is combined with oil. Normally, to apply the gas lift technique, the lift gas is typically sourced from other producing wells, separated from the oil, compressed using a gas compressor, and then pumped at high pressure into the annulus, as illustrated in Figure 1.3. The injection of gas can be performed in the downhole or riser by using gas lift valves that are affixed to mandrels connected to the tubing string [11]. Within the group of gas-lifted wells, hydraulic interdependency is present. This implies that alterations in the operational conditions of one well can have an impact on the pressure and production rates of the others based on network and piping layout conditions.

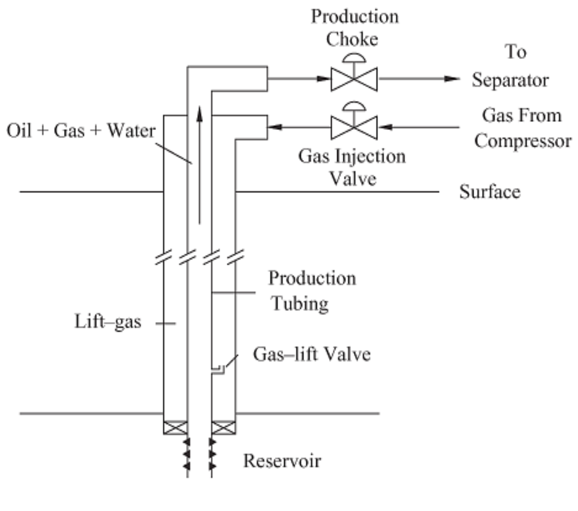


Figure 1.3: Schematic of Oil Well Operating Via Continuous Gas Lift [26]

Gas lift systems are extensively employed in the industry to optimize production. The success of gas lift in optimizing production relies on a range of factors, such as reservoir pressure, well configuration, and operational conditions. In optimizing production via the gas lift technique, appropriate gas distribution to the oil-well network and a finite gas supply are also significant considerations. Gas lift performance curves are utilized for the allocation of lift gases to wells and to enhance production conditions. Figure 1.4 displays the expected operational conditions for each well. Moreover, it demonstrates the response of the well to increased lift gas volumes and the regions where gas is injected in relation to oil production rates. Optimum oil rates can be achieved with a range of gas injection rates. Theoretical ideal gas injection yields zero derivative of reservoir oil production with respect to gas lift rate ($\frac{dq_o}{dq_g} = 0$), while gas injection may cause unstable conditions in another specific zone.

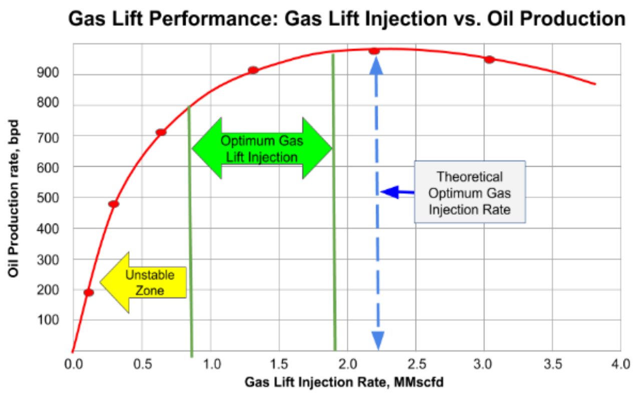


Figure 1.4: Gas Lift Performance Relationship [12]

This curve, Figure 1.4, plays a fundamental role in the design of a gas lift system and its operation. The curve shows how increased lift gas improves productivity at lower rates but can also lead to "over injecting" and reduced productivity.

1.1.3 Gas Lift Assessment

In order to assess the gas lift method as one of the most popular artificial lift methods, it is necessary to describe some of its benefits and challenges.

Gas lift flexibility in production rates and depth of lift surpasses other artificial lift methods, as long as there is sufficient injection-gas pressure and volume. The gas lift is a highly versatile method that can effectively operate under a wide range of production conditions. Gas lift stands out as the most effective artificial lift method for handling sand or solid materials. Sand production is still evident in multiple wells despite sand control implementation. The gas lift valve is barely impacted by the sand produced, whereas most pumping methods are severely affected by even a small amount of sand. Additionally, the gas lift system is convenient to maintain, as there is no requirement for extra downhole equipment to ensure effective maintenance. Moreover, as the device has only a few moving components, repairing it is an affordable and quick process, and replacing the gas lift valve is an affordable action. Lastly, a central gas lift system can efficiently service multiple wells or even the entire field. Centralization can simplify the management and testing of wells, potentially reducing the overall cost of capital.

Despite the numerous benefits, certain challenges arise during the implementation and optimization of gas lift. Certain challenges are universal to gas lift operations, while others are particular to specific fields based on their reservoir characteristics or facility design. The gas lift optimization probably faces several challenges, including optimizing dual gas lift, retrieving tight dummy valves at a specific platform, lifting production from a thin oil column, and managing difficult-to-lift emulsions and viscous crude. The challenges of gas lift in a mature field go beyond production. Surface equipment and sub-surface well completion are also at risk for deterioration, such as leaks, holes, and inaccurate metering. Gas compressor availability and efficiency can also become unstable. To optimize gas lift design, it is essential to have accurate data and precise control over gas distribution. Without reliable data, gas lift systems are likely to remain inefficient. Gas lift distribution and injection downhole have not been effective as a result of these cases.

1.2 Problem Formulation

Gas lift optimization in a production network is a complex and demanding challenge. The gas-lift method presents a challenge in optimizing the oil production, as each well has a unique optimum injection limit. The injection of large quantities of gas results in decreased oil production. This occurs when the friction pressure loss reaches a certain threshold, causing the gas phase to move more quickly than the liquid phase, resulting in the fluid from the reservoir being left behind.

Figure 1.5 shows the gas lift performance relationship curve that was explained in Section 1.1.2. This illustrates the variation in oil production resulting from different gas lift injection rates (QGL) in a single well. The red color in this figure denotes the maximum oil production at the point where the derivative of the oil production rate from the reservoir with respect to the gas lift rate is zero. The injection gas lift has a constraint, by exceeding its optimum point, it generates adverse results, diminishing the net oil production rate.

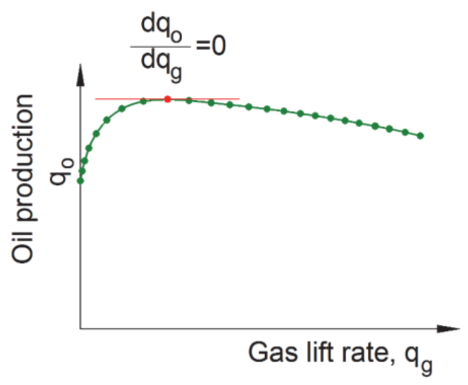


Figure 1.5: Gas Lift Performance Relationship [12]

This thesis primarily focuses on optimizing production in oil gathering systems or well networks with gas-lifted wells. This master's thesis aims to maximize the oil production of an actual well network that includes n wells. Gas lift is employed for this process, which is subjected to several limitations, such as pressure, gas lift injection rate, and other technical and operational constraints. The purpose of this thesis is to propose a model-based numerical optimization technique that utilizes numerical optimization on a production system model to allocate gas-lift gas to well networks, leading to the highest possible oil production. The objective of this thesis is to employ and develop Python scripts to optimize gas lift for the GAP synthetic case using models generated from well performance tables in GAP. A crucial objective is to evaluate and contrast the efficacy of the Python optimizer and optimization in GAP for the simulated GAP model of a synthetic field case. The current thesis work is an extension of the previous specialization project [41], which gave the opportunity to compare different solvers and optimization algorithms in the $SciPy$ library to find the best solution for gas lift optimization in the field.

The well models utilized in this study, which depict the performance of wells when artificially lifted with gas, are constructed in PROSPER and produce TPD files that will be imported into Python in the next stages. PROSPER-derived files comprise a table that demonstrates the numerical values of calculated variables such as Flowing Bottom Hole Pressure (BHP), Flowing Wellhead Temperature (WHT), and Injection Depth. For this work, the only calculated variable that matters is BHP. The calculated variables are the result of combining several independent variables, including gas lift gas injection rate (QGL), liquid rate (QLIQ), water cut (WC), gas-oil ratio (GOR), and boundary pressure (WHP). The number of independent variables or free variables in this study is five, although this may differ in various well-models.

To start this task, the model files need to be imported into Python, which is the desired programming language. The efficiency of the encoding process is crucial due to the potential size of the tables and data points. The model files are in TPD format; however, they require conversion to.txt and subsequent transformation into JSON to be utilized in Python. Python reads the data containing free variables, corresponding values, and the first column of calculated values named BHP, which is then used for processing. The code for importing the model data can be utilized for various free and calculated variables.

Since the imported table is composed of discrete points, it is essential to develop a code that can interpolate these points in the next step. It is crucial to construct this code in a way that enables its adaptability to other situations. Due to numerous free variables, the interpolation is multidimensional. The most significant aspect of interpolation is determining the correlation between the combination and its corresponding value. Interpolation Python objects are defined for each well to be tested during the interpolation step and utilized to interpolate any free variables. The subsequent step involves determining the equilibrium flow rate in each well through the application of the Python solver, under the condition that the inflow and outflow pressures in the well are equal.

When this task is completed, the final action is to execute gas lift optimization on the well network. The process involves modifying the gas lift injection to determine the optimal value for each well using established Python algorithms in $SciPy.Optimize$. The maximum oil rate (q_o) is ensured for each well by keeping the other free variables constant, ultimately resulting in the highest field oil rate $(q_{o-field})$. Achieving the highest oil production and optimal allocation of gas lift injection for each well requires a well-designed optimization process.

Finally, an assessment is conducted via sensitivity analysis to compare the speed and accuracy of the GAP optimizer and the Python gas lift optimizer in diverse scenarios. Examples of such cases include constraints on gas availability for gas lift, limitations in water production due to water processing, and the need to optimize both gas lift rates and choke opening.

1.3 Research Objectives

Section 1.2 presented the overall problem description, and the following objectives are summarized for the thesis. These objectives need to be addressed and demonstrated in the results chapter. The goal of this thesis is to use gas lift injection in both GAP and Python programming to maximize oil production in a well network while adhering to constraints on oil production and gas injection magnitude, as proposed by AkerBP. The main objective is achieved by dividing the thesis into several tasks.

The initial objective is to generate a simulation model in GAP for a synthetic field case. Evaluate the GAP optimizer by performing gas-lift optimization under various conditions.

The second objective is to employ Python to perform model-based optimization of gas lift for the GAP synthetic case, which necessitates the creation of well performance tables derived from GAP.

The third objective is to import the model files, TPD files, into Python and organize and store the information in a suitable structure. The import code needs to be versatile for varying numbers of free and calculated values.

The fourth objective is to interpolate the data points from the model files. Converting units is required due to the use of field units in PROSPER values and metric units in Python calculations.

The fifth objective is to define the interpolation Python "object" for all six wells to perform the interpolation at any given value.

The sixth objective requires creating a function and employing a solver to ascertain the liquid equilibrium rate when inflow and outflow pressures are the same. This information will be used to determine the oil rate (q_o), water rate (q_w), and gas rate (q_g) for each well separately.

The last objective is to optimize the oil rate in the field by reaching the optimal gas lift injection value while taking into account the gas lift injection constraint. Following that, compute the water and gas rates for the whole field.

1.4 Thesis Structure

This present thesis work is structured into five chapters, organized as follows. The first chapter serves as an introduction to the thesis, briefly introducing the topic along with some background information, such as production optimization, the gas lift method, and the assessment of the gas lift method. The problem formulation clarifies the thesis problem as well as the objectives of this study and the techniques and strategies employed to address it. Additionally, this chapter provides an overview of the primary and specific objectives of this research.

Chapter two entails a comprehensive literature and research review on relevant topics for this project. This chapter is designed to facilitate an understanding of the fundamental concepts, such as the performance of production systems, interpolation techniques, optimization methods, and gas lift optimization in well networks, to achieve the aim of the thesis. Additionally, Integrated Production Modeling (IPM) is being presented to enhance familiarity with the model files and synthetic GAP case procedures.

In the third chapter, the planning and execution of the project are discussed, along with the utilization of the GAP model in the thesis. The techniques employed for importing necessary data from TPD files into Python and performing interpolation are detailed. The approaches and procedures for determining the liquid equilibrium rate and calculating q_o, q_w, and q_g for each well are clearly explained. The leading methodology for addressing the optimization problem in this thesis is discussed at the end of this chapter. Through a flowchart, the entire optimization process for the thesis is visually represented.

In chapter four, the outcomes and results achieved in this project are presented in a coherent and structured manner, building on the context and objectives discussed earlier. Tables and charts are employed to illustrate the data and assess its performance. Diverse cases and scenarios have been executed to contrast the outcomes obtained from the GAP optimizer and Python optimization. The findings and results have been interpreted and analyzed.

The final chapter summarized the main points and findings of PROSPER and Python. A list of recommendations for further work is included as well.

Chapter 2

Theory

The theories covered in this chapter, including integrated production modeling software, well performance in production systems, flow equilibrium in production networks, interpolation, and optimization, will serve as support for this thesis. The section on well performance in production systems encompasses well inflow performance relationships, vertical lift performance, and well deliverability. Three relevant techniques of interpolation theory, including linear, polynomial, and spline, are shown. Next, several optimization algorithms will be detailed and their contrast shown. This chapter ends with theories about optimization in GAP and gas lift optimization for well networks.

2.1 Integrated Production Modeling (IPM)

Integrated Production Modeling Toolkit (IPM) is a software program that simulates the entire oil or gas production system, encompassing the reservoirs, wells, and surface network developed by the Petroleum Expert (Petex). The IPM package contains a range of tools, such as GAP, GAP TRANSIENT, PROSPER, MBAL, PVTP, REVEAL, MOVE, and RESOLVE, which can be smoothly integrated and utilized together by engineers to construct comprehensive field models (Figure 2.1). IPM has the capability to simultaneously model and optimize the water or gas injection system along with the production system. Also, all types of commonly occurring naturally flowing well setups, including those with multiple laterals as well as those utilizing artificial lift, can be modeled and optimized in conjunction with each other [1].

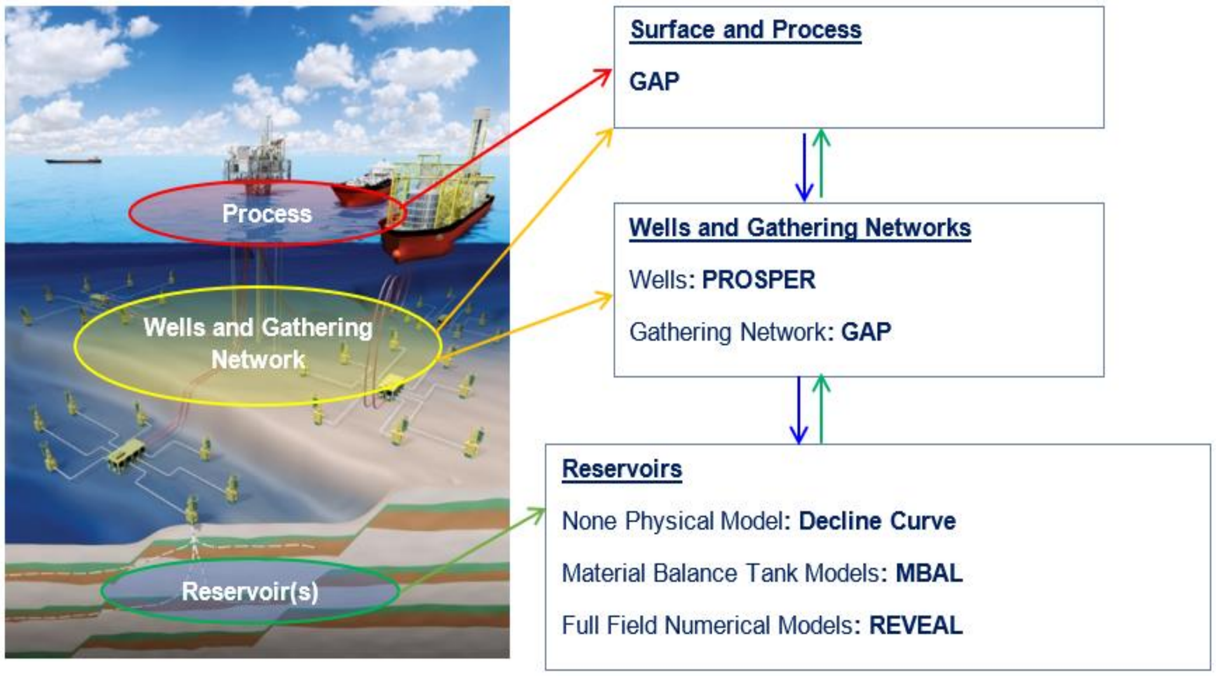

Process
Wells and Gathering Network
Reservoir(s)
Surface and Process
GAP
Wells and Gathering Networks
Wells: PROSPER
Gathering Network: GAP
Reservoirs
None Physical Model: Decline Curve
Material Balance Tank Models: MBAL
Full Field Numerical Models: REVEAL

Figure 2.1: Network Modeling from Subsurface to Surface and Utilization of IPM

The integrated production modeling approach involves collaboration between surface and subsurface teams to identify potential opportunities to improve performance and results while considering the limitations and constraints that exist within the field. For instance, IPM techniques are used to integrate the subsea infrastructure and topside processing facilities in deepwater subsea systems. And it is important for the components of the model to be precise enough to accurately depict the performance of the system, specifically regarding pressure and flow rate. Two components of IPM are described below: PROSPER and GAP.

2.1.1 PROSPER

PROSPER is a **PRO**duction and **S**ystem **PER**formance analysis software utilized for the purpose of well performance, analysis, design, and optimization. Using this tool allows production or reservoir engineers to predict, with accuracy and speed, the hydraulics and temperature of tubing and pipelines.

With the help of PROSPER, it is possible to construct well models that can account for all factors, including well configuration, fluid characteristics (PVT), multiphase VLP correlations, and different IPR models. One crucial aspect of analyzing the well model is to identify the wells with the potential to produce at a higher rate than the current one. Another critical aspect of well analysis involves demonstrating the present correlation between well flow and performance, known as the inflow performance relationship (IPR). The appropriate well-test data and IPR model must be selected for the analysis to be effective.

The concept of integrated production modeling (IPM) is employed when utilizing Prosper for well modeling. PROSPER serves as the bridge between the reservoir model and the surface model and links the two. This includes providing insight into the functioning of the well as well as the associated reservoir and vertical rise.

There are various ways to utilize PROSPER applications. The primary use of this software is to compute VLP utilizing multiphase flow correlations and variables. By altering the outflow variables, more analysis can be conducted. For instance, altering the injected gas lift rate in a gas lift system simulation results in an increase in the amount of produced liquid.

2.1.2 GAP

GAP is a **G**eneral **A**llocation **P**rogram. This software is an effective tool provided in petroleum engineering for fulfilling many crucial jobs, such as complete surface production, injection network modeling, production optimization, lift gas allocation, and production forecasting.

A full field model can be created in GAP. As it considers the multiphase network response of various wells, each with a unique PVT, producing into one production system where the response of one well can affect the production of another, like a back pressure response.

In GAP, the production system consists of producing elements like wells or sources that are linked through shared manifolds and pipelines to a static pressure system called the separator. The separator need not be the physical separator that is present in the field. Instead, it operates as a reference point for stable pressure within the network.

GAP, one of the most rapid and efficient optimization programs available, is built upon nonlinear optimization strategies, including the sequential quadratic programming (SQP) methodology. The GAP optimizer provides the engineer with the option to maximize a certain objective function, such as oil output, while also abiding by any constraints and limitations placed on the system. Full-field gas lift optimization for distributing gas among wells is one example of how GAP optimization is used.

2.2 Well Performance in Production Systems

The oil and gas industry adopts techniques that optimize processes, including characterization, measurement, and monitoring of fields. One essential aspect of field optimization in the oil and gas industry is well performance management. This involves monitoring the performance of individual wells in an oil field to ensure that they are producing at optimal levels and contributing positively to the overall production. Well performance management involves various techniques to measure various parameters, including flow rate, pressure, temperature, and production water rate. These parameters are usually monitored using a combination of physical measurements and downhole sensors, which transmit data to surface control systems. By effectively monitoring well performance, it is possible to identify and resolve problems that may arise in the oil production process.

Well performance management is important for optimizing the productivity of oil fields, as it enables operators to ensure efficient and cost-effective operations. Additionally, it allows for the early identification of potential problems in individual wells or across the entire production system. Efficient performance management strategies are essential in recognizing prospects for amplifying production rates and prolonging the economic longevity of oil fields. Well performance and productivity can be optimized using a range of methods. These include infill drilling, artificial lift techniques such as gas lift or ESP (electric submersible pump) systems, and hydraulic fracturing. In this thesis, the preferred method of artificial lifting is the gas lift technique.

To optimize production in a gas lift well system, understanding the inflow and outflow performance of the wells is crucial for assessing their behavior under particular conditions and features. Important theoretical concepts that are crucial to the thesis work are presented in the following sections.

2.2.1 Well Inflow Performance Relationship (IPR)

Fluids move from the reservoir through the wellbore to reach the wellhead in subsurface hydrocarbon production. The diagram in Figure 2.2 illustrates the division of this fluid movement into two parts: inflow and outflow. Inflow refers to the movement of hydrocarbons from the reservoir rock to the wellbore. The inflow performance of a well is an indicator of its fluid production behavior, which is based on its flowing pressure and production rate. Well performance varies significantly from one well to another in heterogeneous reservoirs. The Inflow Performance Relationship (IPR) expresses the connection between the flow rate of a well (q), reservoir pressure (P_R), and the flowing pressure of the well (P_{wf}).

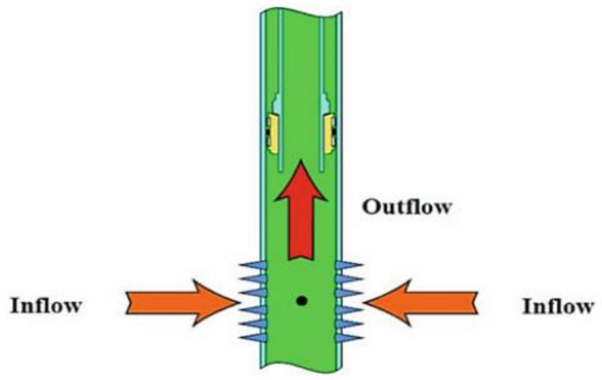


Figure 2.2: Subsurface Production [29]

The Inflow Performance Relationship (IPR) is a fundamental concept in the oil and gas industry, especially in production systems analysis. The IPR is frequently essential for well capacity estimation, well completion design, tubing string design, well production optimization, nodal analysis calculations, and artificial lift design. A pressure-rate correlation is used to forecast the production of oil and gas wells. The relationship between oil production rate and bottom hole hydraulic pressure (BHP) is also plotted to generate the IPR curve. The IPR curve is one of the standard methods for estimating oil well production.

IPR shape depends on multiple factors, such as pressure drop, viscosity, formation volume factor, skin, and relative permeability throughout the reservoir. The most well-known IPR correlations can be categorized as empirically derived or analytically derived. Vogel (1968), Fetkovich (1973), Kilns and Majcher (1992), Wiggins (1993), and Sukarno et al. (1995) are some of the most widely recognized empirically derived correlations. Wiggins et al. (1991, 1992) and Del Castillo, Yanil, et al. (2003) are well-known examples of analytically derived correlations [30]. Empirical correlations, including the straight line model, Vogel, and Composite, are explained in this work.

The type of reservoir in which it is present has a significant impact on the shape of the IPR. This can vary based on whether the reservoir is undersaturated with oil or saturated with both oil and gas, as shown in Figure 2.3. The IPR shape for a saturated reservoir is typically hyperbolic, while for an undersaturated reservoir, it begins as hyperbolic and transforms into a linear shape at the bubble point pressure.

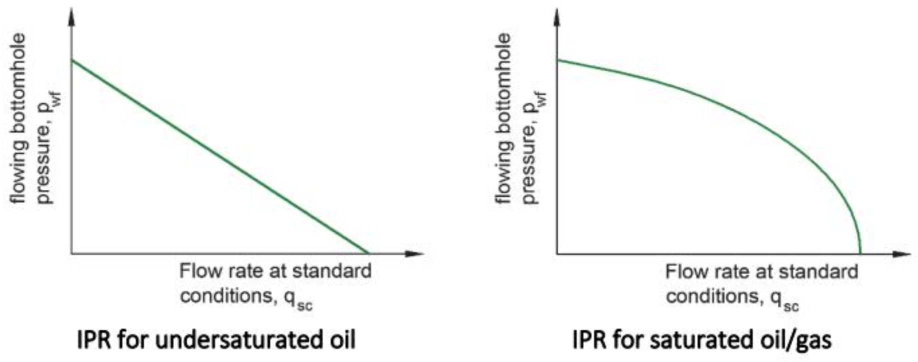


Figure 2.3: IPR Curves [2]

The pressure-temperature (PT) phase diagram, Figure 2.4, can be used to determine the condition of the reservoir by comparing the values of reservoir pressure (P_R), bubble point pressure (P_B), and bottom hole pressure (P_{wfi}) of the formation. The range of pressures and temperatures that an oil reservoir experiences throughout its existence is known as the pressure and temperature envelope.

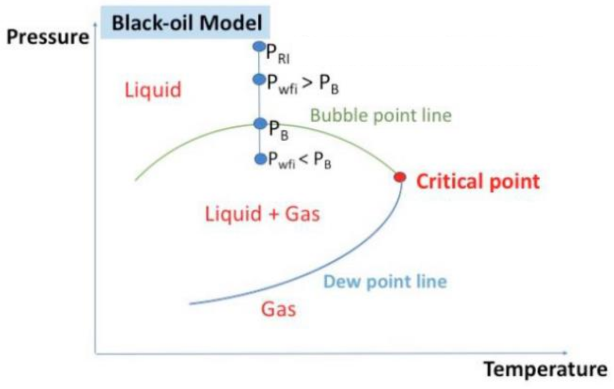


Figure 2.4: P x T Curve for Multi-Layer Reservoirs [39]

2.2.1.1 Linear Inflow Performance Relationship

The productivity index is commonly used to indicate the production capability of the reservoir to transport fluids to the wellbore. The ratio of the total liquid flow rate to the pressure drawdown is what defines the productivity index, which is represented by the symbol J.

The Productivity Index enables the estimation and prediction of productivity and production efficiency of the well. Obtaining the J value is usually done by conducting a flow test in a well to get an initial stabilized flow (q_{oi}) for wellbore pressure(P_{wfi}).The productivity index is essential in evaluating the inflow performance of the well, which is influenced by the reservoir and fluid properties. The calculation of J is possible through the Equation of Darcy shown by Equations 2.1 and 2.2.

$$q_o = \frac{0.00708 k_o h (P_R - P_{wf})}{\mu_o B_o [\ln\left(\frac{r_e}{r_w}\right) - 0.75 + S]} \tag{2.1}$$

$$J = \frac{q}{(P_R - P_{wf})} = \frac{0.00708 k_o h}{\mu_o B_o [\ln\left(\frac{r_e}{r_w}\right) - 0.75 + S]} \tag{2.2}$$

where k_o is effective permeability of the oil $(md), h$ is pay thickness$(ft), B_o$ is oil formation volume factor$\left(\frac{bbl}{STB}\right)$, μ_o is oil viscosity(cp), P_R is reservoir pressure(psi) , P_{wf} is flowing bottom hole pressure(psi) , r_e is radius of external boundary $(ft), r_w$ radius of well $(ft), S$ is skin factor ,and q_o is oil rate $\left(\frac{STB}{day}\right)$, and J is productivity index $STB/day/psi$.

Equation 2.3 simplifies calculating the expected inflow rate at a given flowing well pressure if J is known.

$$q = J(P_R - P_{wf}) \tag{2.3}$$

Equation 2.3 indicates that liquid inflow into a wellbore is directly proportional to pressure drawdown. The straight-line IPR equation is the most common and basic one, indicating the rate is directly proportional to pressure drawdown in the undersaturated oil reservoir or is slightly compressible (as shown in Figure 2.5). Absolute open flow (AOF) is the term for the maximum flow rate, $q_{o,max}$,when the bottom hole flowing pressure is zero, as shown in Figure 2.5.

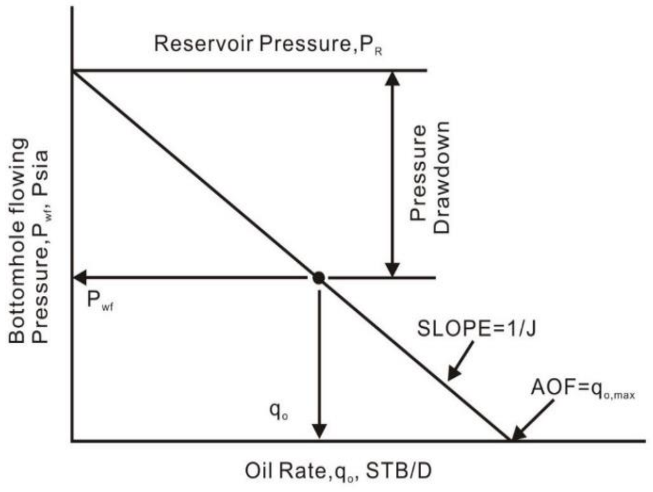


Figure 2.5: The Linear IPR Curve [3]

The Figure 2.5 illustrates various significant characteristics of the straight-line IPR, for instance

- The wellbore flowing pressure is usually the independent variable that is plotted on the y axis, while the dependent variable rate is plotted on the x axis as per convention.

- The flow rate in the wellbore will be zero if the wellbore flowing pressure is equal to the average reservoir pressure (known as static pressure) and there is no pressure drawdown.
- At a wellbore pressure of zero, the maximum flow rate ($q_{o,max}$) or absolute open flow (AOF) is achieved. Despite not being applicable in all cases, this definition is beneficial and widely utilized in the petroleum industry to measure the performance of various wells in a field.
- The inverse of the productivity index is equal to the slope of the straight line (slope $= 1/J$).

2.2.1.2 Vogel Inflow Performance Relationship

A technique for forecasting the inflow performance of the well in solution gas drive (two-phase flow) conditions where $P_{wf} \leq P_b$ was devised by Vogel. Highly compressible gas and two-phase flow affect IPR, making the linear IPR unsuitable for saturated oil and gas wells. Vogel proposed an IPR for forecasting saturated oil reservoirs, which was widely embraced by the industry due to its accuracy and simplicity [30]. If the produced fluid is pure oil and the flowing bottom hole pressure is less than the bubble point pressure, then Vogel might be applicable. The Vogel equation is depicted by Equations 2.4 and 2.5.

$$\frac{q_o}{q_{o,max}} = 1 - 0.2\left[\frac{P_{wf}}{\bar{P}_r}\right] - 0.8\left[\frac{P_{wf}}{\bar{P}_r}\right]^2 \tag{2.4}$$

This equation may be solved directly for P_{wf} as follows

$$P_{wf} = 0.125P_r\left[-1 + \sqrt{81 - 80(q_o/q_{o,max})}\right] \tag{2.5}$$

where q_o is oil rate (STB/day) at P_{wf}, $q_{o,max}$ is maximum oil flow rate (AOF) at zero wellbore pressure, $\bar{P}_r$ is current average reservoir pressure(psig), and P_{wf} is wellbore pressure(psig)[28].

A significant curvature can be observed in the IPR of the Vogel, as illustrated in Figure 2.6. The shape of the curve is influenced by the behavior of fluid phases during flow and the composition of the reservoir fluid. Fetkovich (1973) introduced an equation that explains the relationships of two-phase inflow performance, besides the Vogel correlation. But the equation of Vogel is a best-fit approximation based on various simulated well performance calculations [9].

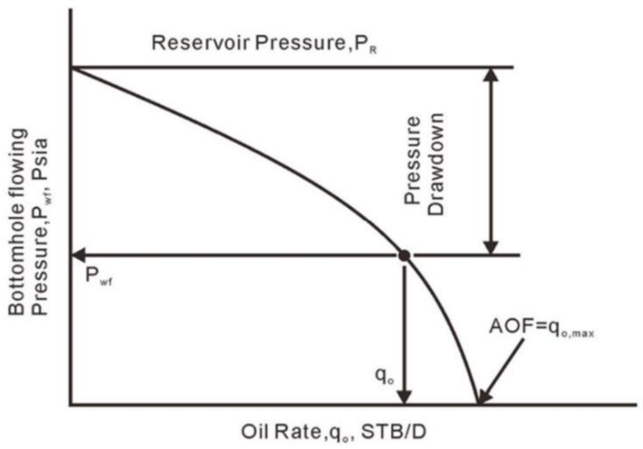

Reservoir Pressure, P_R
Pressure Drawdown
Bottomhole flowing Pressure, P_{wf}, Psia
P_{wf}
AOF=$q_{o,max}$
q_o
Oil Rate, q_o, STB/D

Figure 2.6: The Vogel's IPR Curve for the Saturated Oil and Gas Wells [32]

2.2.1.3 Composite Inflow Performance Relationship

The Composite Inflow Performance Relationship was established by merging the PI model with the inflow performance relationship of Vogel. As previously stated, the PI model applies to situations where the well is producing a single-phase flow with no gas in the solution $(P_{wf} > P_b)$. If the produced fluid is pure oil and the flowing bottom hole pressure (P_{wf}) is less than the bubble point pressure (P_b) , $(P_{wf} < P_b)$ the IPR of Vogel can be used.

For mixed oil, water, and gas produced fluids, if the P_{wf} is less than P_b , the Composite IPR can be used to describe the inflow performance of the well. IPR curves can be found that fall between those that apply to pure oil (Vogel Model) and those that apply to a PI Model $(P_{wf} > P_b$) . The position of the Composite IPR curve presented in Figure 2.7 is expected to be placed between the Vogel curve and the Darcy straight line.

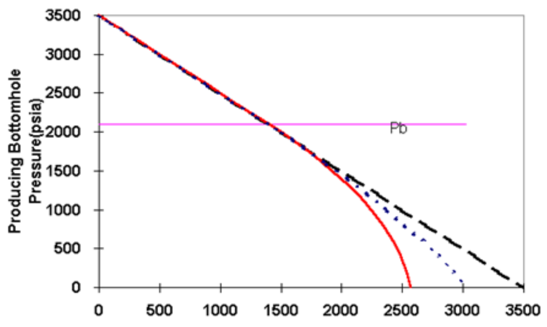


Figure 2.7: IPR Curves for Three Different Phases [4]

A schematic plot, as depicted in Figure 2.8, can aid in comprehending the behavior of the IPR function. The plot illustrates how the function behaves above and below the bubble point pressure. By combining single-phase liquid and two-phase flow, the productivity index (PI) of the well is determined, and its production behavior and productivity are analyzed.

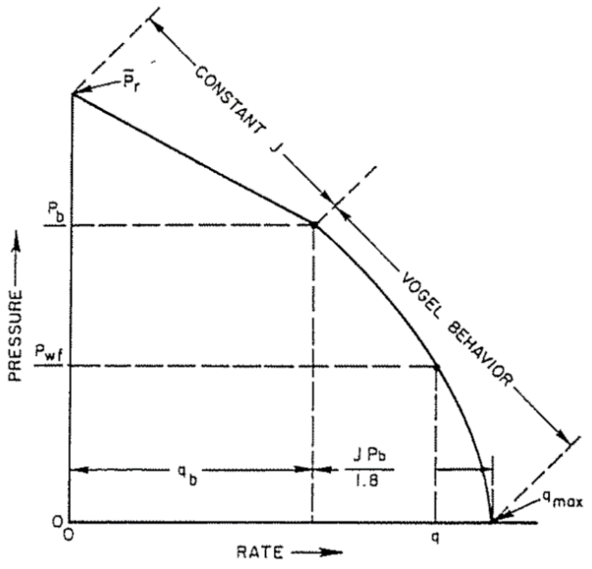


Figure 2.8: Combination Constant PI and Vogel Behavior Case [28]

A total flow rate q, which encompasses the flow from P_r to P_b and the flow from P_b to P_{wf}, can be obtained using a set of special equations that have been formulated below. By using Equation 2.6a, it is possible to calculate the total rate of the well which needs to calculate q_{omax} and q_b from Equations 2.6b and 2.6c , respectively. Furthermore, Equation 2.7a can be utilized to determine the bottom hole pressure(BHP) at different points of the curve.

$$q = q_b + (q_{max} - q_b) * [1 - 0.2\left(\frac{P_{wf}}{P_b}\right) - 0.8\left(\frac{P_{wf}}{P_b}\right)^2] \quad (2.6a)$$

where

$$q_{omax} = q_b + \frac{JP_b}{1.8} \quad (2.6b)$$

$$q_b = J(P_r - P_b) \quad (2.6c)$$

and based on P_{wf} it can be as below

$$P_{wf} = 0.125P_b\left[-1 + \sqrt{81 - 80\left\{\frac{q_t - q_b}{q_{omax} - q_b}\right\}}\right] \qquad (2.7a)$$

where like before, q_{omax} and q_b can be calculated from Equations 2.7b and 2.7c.

$$q_{omax} = q_b + \frac{JP_b}{1.8} \qquad (2.7b)$$

$$q_b = J(P_r - P_b) \qquad (2.7c)$$

2.2.2 Vertical Lift Performance (VLP)

Efficient performance of the vertical lift system is necessary for the oil and gas industry to lift fluids from deep underground reservoirs to the surface. The relationship between flow rate and pressure losses is represented by the vertical lift performance curve shown in Figure 2.9, in which Q_L is operating flow rate. The VLP curve illustrates the amount of pressure needed to raise a specific volume of fluid through the tubular to the surface at a specific well head pressure.

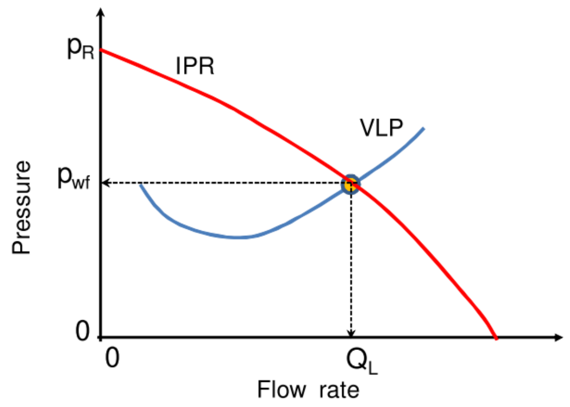


Figure 2.9: Vertical Lift Performance Curve [31]

Sufficient bottomhole pressure is required to overcome flow resistance in the tubing and surface choke because of friction and to support the hydrostatic head caused by the weight of the compressible mixture in the tubing due to gravity. Further, the decrease in pressure because of acceleration is also considered, which occurs as fluids expand when pressure reduces [6]. Equations 2.8a and 2.8b illustrate the pressure drop that occurs during the lifting of reservoir fluids to the surface as a key factor affecting well deliverability [8].

$$\frac{dp}{dL} = \left(\frac{dp}{dL}\right)_{Hydrostatic} + \left(\frac{dp}{dL}\right)_{Friction} + \left(\frac{dp}{dL}\right)_{Kinetic} \tag{2.8a}$$

$$\frac{dp}{dL} = \rho g sin\theta + f\frac{\rho\, v^2}{2d} + \rho v\frac{dv}{dL} \tag{2.8b}$$

where

$\rho =$ density of fluid, kg/m^3
$g = acceleration\ of$ gravity,ft/s^2
$v = flow$ velocity,m/s
f =friction factor,

$d =$ pipe inside diameter,$inchL$ =vertical depth,ft

$\theta =$ pipe inclination angle, measured from horizontal

In Equation 2.8b,the first term $\rho g sin\theta$ is known as hydrostatic or gravity(g) losses, which is a result of the density(ρ)of fluid column. A frictional loss is then calculated by $f\frac{v^2}{2d}$ due to the fluid friction on the pipe inner wall and viscous drag(v). The last term $\rho v\frac{dv}{dL}$ for kinetic losses comes from the expansion of fluid and the change in cross-sectional area.

The VLP is affected by various factors, such as well characteristics, reservoir properties, fluid properties, and operating conditions. In other words, the VLP curve, similar to the IPR, is influenced by multiple variables, including production rate, choke, gas-oil ratio (GOR), tubing diameter, and water-oil ratio (WOR) [5]. These factors can affect the selection of an appropriate artificial lift method, such as gas-based lift or pump-based lift processes, to achieve optimal performance. Gas-based lift technology has been identified as an efficient and cost-effective method to improve well performance in underperforming oil or oil and gas wells. Gas lift technology relies on the injection of gas into the wellbore to decrease the hydrostatic pressure in the tubing and increase fluid flow rates [7].

2.2.2.1 Effect of Gas Lift Injection on VLP Curves

In the absence of gas injection, the point of flow equilibrium can be determined through the intersection of the VLP and IPR curves, where the bottom-hole pressure is plotted against the oil flow rate. However, upon injecting the gas through the valve, the gas-oil ratio (GOR) of the tubing and pipeline undergoes modification, leading to an alteration in the VLP curve and intersection point. An increase in gas lift injection typically causes a shift in the VLP curves to the right and upward, resulting in the increased natural oil flow rate or operating point shown in Figure 2.10.

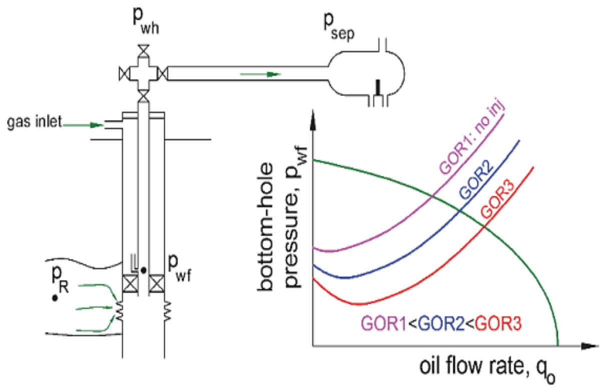


Figure 2.10: Natural Equilibrium Points for Well with No Gas Lift Injection and with Gas Injection [2]

Until it reaches its optimal point and produces the highest $q_{o,max}$, this trend continues, after which it begins to reverse. When the optimal gas lift injection point is attained, the more gas injected, the less natural flow can be observed. Varying the gas injection rate has diverse effects on friction and hydrostatic factors that account for the pressure drop in tubing. By increasing gas lift injection, the fluid density is lightened, leading to a reduction in pressure loss from hydrostatic pressure. Nevertheless, greater amounts of gas injection lead to increased pressure losses because of friction. Figure 2.11 illustrates the combined impact of gas injection on the total pressure drop in the tubing. When the gas-oil ratio is at its highest level, GOR4, there is a decrease in oil flow production due to pressure losses.

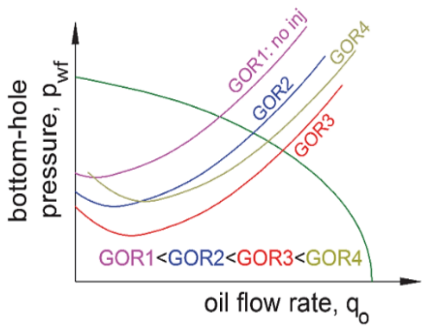


Figure 2.11: Natural Equilibrium Points Calculated for Different Amounts of Gas Lift Injected [2]

2.2.3 Well Deliverability

Well deliverability, or the ability to produce fluids from a reservoir, plays a vital role in optimizing the economic value of an oil or gas field. The point where the Inflow Performance Relationship (IPR) intersects with the Vertical Lift Performance (VLP) represents the deliverability of the well identified by the operating point. At this point, the well has achieved its optimum liquid production, showing its actual production capability under specific operating conditions (Figure 2.12). The IPR and the VLP both establish a connection between the flowing pressure within the well bore and the production rate at the surface. The IPR measures the potential output from the reservoir to the bottom of the well, and the VLP indicates the potential output from the well to the surface.

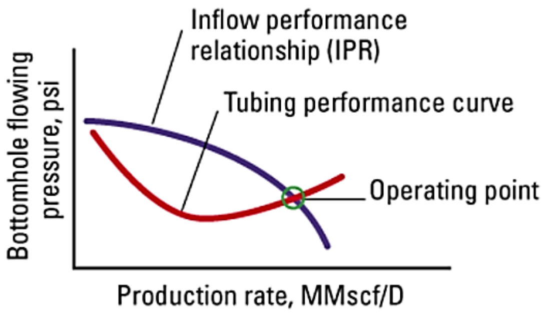


Figure 2.12: IPR and VLP Curves and Operating Point [9]

Alteration in VLP, in Figure 2.13, indicates that reducing the flowing bottom-hole pressure can be achieved by minimizing pressure losses between the bottom-hole and the separation facility through actions such as removing unnecessary restrictions, optimizing tubing size, or improving artificial lift procedures. A significant responsibility of a production engineer is to enhance the productivity of a well by optimizing the flow system that runs from the bottom of the well to the surface production facility. Boosting productivity in the oil field requires optimizing specific portions of the flow system. Possible corrective actions can vary from well-stimulation techniques like hydraulic fracturing that increase flow in the reservoir to resizing surface flow lines or skin removal to improve productivity and improve IPR and VLP curves to reach $q_{improved}$.

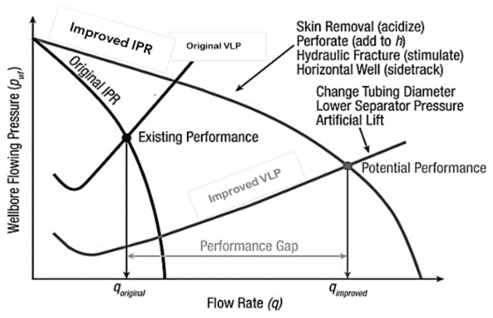


Figure 2.13: Well Deliverability Gap between Original Well Performance and Optimized Well Performance [10]

2.3 Wellhead Choke Performance

When describing and estimating the behavior of a reservoir, flow rate is the most important parameter. Wellhead chokes are responsible for regulating and maintaining stable flow rates in single or multiple phases during production. During the early phases of production, it is common to use a choke to limit the flow of oil or gas from a well. However, as the pressure in the reservoir decreases over time, the choke may be adjusted or even removed altogether. Chokes serve several purposes, including ensuring safety, staying within production limits, preventing sand from entering the well, optimizing the rate of production, and stopping the formation of gas or water coning. Installing a choke at the wellhead results in the wellhead pressure being set, which in turn determines the bottom-hole pressure and production rate.

The function of a choke is to limit the flow of fluid by providing a narrow pathway. It is used in oil drilling to create back pressure on a flowing well, which results in an increase in the bottom hole flowing pressure. This increased pressure helps to reduce the pressure drop from the reservoir to the wellbore, also known as pressure drawdown. However, by increasing the backpressure in the wellbore, the flow rate from the reservoir is reduced.

Further, by enlarging the choke opening to reduce wellhead pressure, the tubing performance relationship (TPR) curve usually moves downward, resulting in a decrease in intake pressure. When the wellhead pressure is minimized, the well will produce at its maximum potential. Conversely, narrowing the choke opening to raise well head pressure will shift the TPR curve upward, causing a decrease in rate. However, if the wellhead pressure exceeds a certain threshold, the well will no longer produce [3], Figure 2.14.

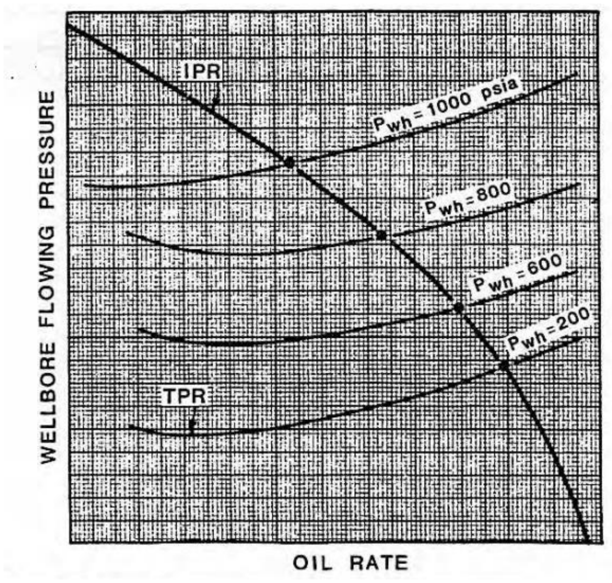


Figure 2.14: Effect of Well Head Pressure on the Natural Flow [3]

The percentage of the wellhead choke opening, which is in charge of controlling and managing the fluid flow from the reservoir to the surface production equipment, has a significant impact on the level of oil production in the production system. Given that the fluid flow rate through the wellhead choke is directly linked to its aperture diameter, any variation in the opening percentage of the choke will have an impact on the oil or gas production rate. Besides, the percentage of the wellhead choke opening affects significant factors, like the velocity of production fluids and the pressure difference between the reservoir and surface equipment. The data presented in Figure 2.15 illustrates that the oil production rate is higher when the wellhead choke is fully open compared to 75% opening, where P_2 is outlet pressure.

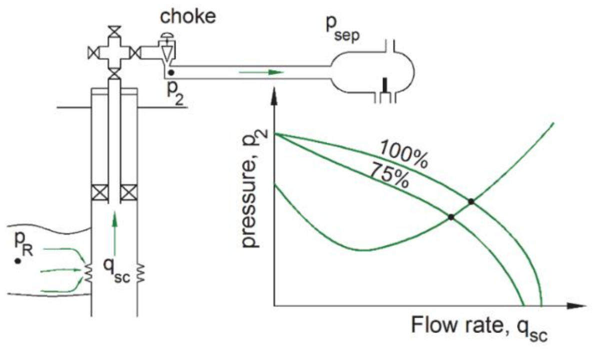


Figure 2.15: Equilibrium Flow Rate of the System for Fully Open Choke and 75% Open Choke [2]

2.4 Flow Equilibrium Rate in Production Networks

Typically, the production system is delimited by two fixed pressure boundaries: the reservoir pressure (P_R) and the separator pressure denoted as P_{sep}. The pressure curves in Figure 2.16 demonstrate that the available pressure accounts for the losses in the reservoir and wellbore, while the required pressure encompasses the pressure losses in the pipeline that ensure the separator pressure remains constant. As the reservoir depletes, the pressure curves change, and the necessary pressure difference required for producing the specified rate varies over time. The terms available and required pressure curves are often used interchangeably with inflow and outflow performance curves, respectively. When the inflow and outflow performances are equivalent, this is referred to as the operating (equilibrium) point of the integrated production system, as illustrated by q_{sc} in Figure 2.16. The intersection point of the inflow performance and vertical lift curves of the producing wells determines the equilibrium liquid rate. The reliable point in producing wells is referred to as equilibrium, where there is no deficit or surplus of a specific quantity.

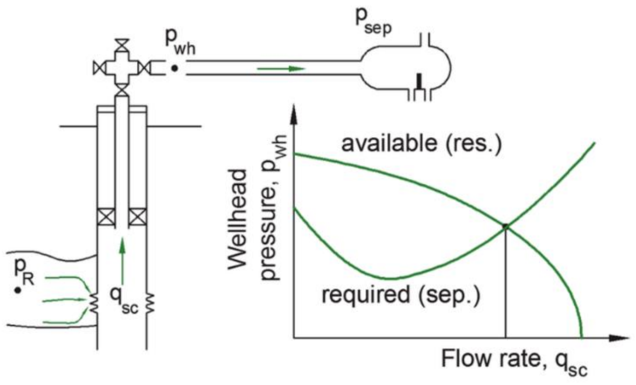
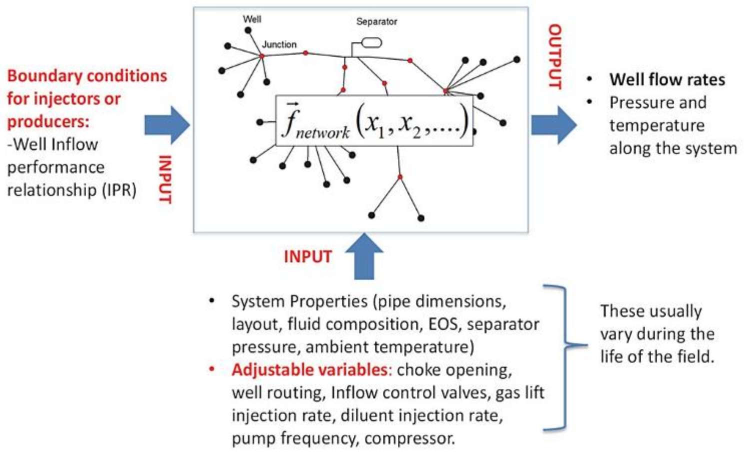


Figure 2.16: Equilibrium Flow Rate of the System Calculated by Intersecting the Available Pressure Curve Calculated from the Reservoir and the Required Pressure Curve from the Separator [2]

The hydraulic performance of a well in a production network is influenced by the operating conditions of other wells. Hence, all hydraulic interactions must be considered when computing its performance. The graphical intersection technique is typically used in single- well pressure curves, but it is uncommon to employ this method for explaining the equilibrium calculations of a production network. The analysis of flow performance in production networks is primarily done through computerized routines and software. Figure 2.17 provides a visual representation of the mathematical function that accurately describes the production network and can be more practically applied. To generate well rates, the network model as a function requires inputs of system properties and adjustable elements, such as choke opening, inflow control valves, gas lift injection rate, and compressor settings. The properties that constitute the production system are pipe dimensions, layout, fluid composition, separator pressure, and ambient temperature. It is important to note that system properties and adjustable variables generally undergo alterations during the lifespan of the field.



Well
Junction
Separator
$\vec{f}_{network}(x_1, x_2, ...)$
OUTPUT
Well flow rates
Pressure and temperature along the system
Boundary conditions for injectors or producers:
-Well Inflow performance relationship (IPR)
INPUT
INPUT
System Properties (pipe dimensions, layout, fluid composition, EOS, separator pressure, ambient temperature)
Adjustable variables: choke opening, well routing, Inflow control valves, gas lift injection rate, diluent injection rate, pump frequency, compressor.
These usually vary during the life of the field.

Figure 2.17: Depiction of the Production Network Model as a Mathematical Function [2]

The presence of adjustable equipment in the production system, such as chokes, pumps, and gas lift injection, means that the system can produce various feasible equilibrium rates. The point of intersection of the two curves is affected by adjustable equipment, which has an impact on the hydraulic performance of the system. For instance, in a system that includes a choked well, the flow rate of the well may vary depending on the degree of openness of the choke, be it fully open, fully closed, or somewhere in between. Another instance is altering the rotational speed of the electric submersible pump (ESP), which can lead to a diverse range of operational rates. This study investigates the hydraulic equilibrium modifications caused by gas lift injection and choke.

Once the production network does not have any adjustable elements or has fixed settings, there will be a unique solution for the production network. But if there are adjustable elements in the production network, the solution will vary based on the settings of these elements. There are typically multiple operational conditions that can be achieved. To illustrate, when each well is equipped with a gas lift injection, the network model function solution becomes contingent on the injected gas lift. Gas lift injection has the potential to enhance well performance and achieve maximum flow rates under specific operating conditions, as depicted in Figure 2.18. The figure demonstrates that increasing the injection gas flow rate shifts the equilibrium point downward and towards the right, thus enabling maximum liquid flow rate. Gas lift injection has a direct impact on vertical lift performance (VLP), and increasing the gas lift injection results in a shift in VLP to achieve the maximum liquid flow rate for individual wells.

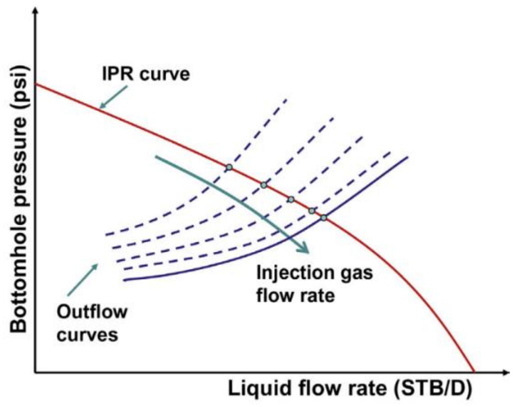


Figure 2.18: Total System Analysis for Different Injection Gas Flow Rates [50]

2.5 Interpolation

Interpolation is a mathematical technique that approximates an unknown value using two or more certain known values. Interpolation is extensively employed in diverse disciplines, including mathematics, physics, and computer science, for approximating functions based on a limited set of data points and constructing new data points within the range of known data. In many real-world applications, such as weather forecasting, financial analysis, and stock market predictions, where data is not available at every point and therefore needs to be estimated for improved analysis or decision-making, this technique plays a crucial role.

Interpolation is also employed in numerical analysis to solve differential equations and optimization problems with unknown or missing values. Various interpolation techniques, including linear and polynomial interpolation and spline functions, can be used to approximate unknown values. These multiple forms of interpolation, each of which employs unique equations, may be applicable to datasets with distinct distributions.

Interpolation is the process of determining a function $y = f(x)$ whose graph passes through the $n + 1$ given points (x_i, y_i) for $i = 0, 1, \dots, n$, given the values $x_0, \dots, x_n$ and their corresponding values $y_0, \dots, y_n$. There are a limitless number of functions of this kind. Figure 2.19 depicts an example of Polynomial interpolation that represents the values of an unknown function at six specific points, namely $(x_1, y_1), (x_2, y_2)$, and so forth.

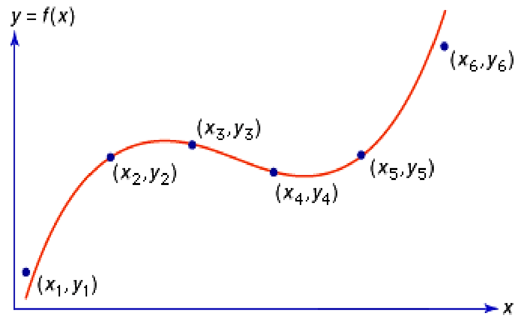


Figure 2.19: Polynomial Interpolation of the Six Points [42]

A comprehensive explanation of linear, polynomial, and spline interpolations can be found in the following sections.

2.5.1 Linear Interpolation

Linear interpolation is the simplest mathematical technique that involves the generation of new values based on an existing set of values. This allows to estimate an unknown value for a variable by constructing a straight line between two known values and using this line to make predictions about the unknown value. Suppose that the x-data points are arranged in ascending order; that is $x_i < x_{i+1}$ and $x_i < x < x_{i+1}$ The linear interpolation at x or straight line is given by Equation 2.9 [13].

$$y(x) = y_i + \frac{(y_{i+1} - y_i)(x - x_i)}{(x_{i+1} - x_i)} \tag{2.9}$$

where x is the point performing interpolation, and y is the interpolated value in Equation 2.9. Furthermore, the formula for linear interpolation is a technique that is beneficial for curve fitting with linear polynomials. Essentially, the interpolation method is employed to determine novel values for any function by utilizing a series of established values.

The accuracy of the estimation depends on how well the relationship between the data points can be approximated by a straight line. Linear interpolation is a quick and easy method for estimating values within a given range of data points, but it may not always provide the most accurate results, especially when the data points do not exhibit a linear relationship. Linear interpolation tends to be more precise when numerous data points are available, such as the eleven points shown in Figure 2.20.

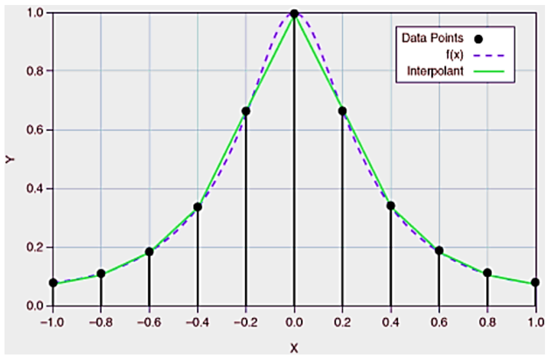


Figure 2.20: Linear Interpolation with Available Data Points [14]

2.5.2 Polynomial Interpolation

Polynomial interpolation is a fundamental problem of computational mathematics that dates back centuries to the classic work of Newton, Waring, and Lagrange. In the field of numerical analysis, the problem of interpolating a set of $n + 1$ discrete given points $(x_i, y_i), i = 0, \dots, n$, by a polynomial of the lowest possible degree that satisfies the interpolation condition $p(x_i) = y_i$ for $i = 0, \dots, n$ is a common and important task. The solution to this problem is known as the interpolation polynomial, denoted by $p(x)$, which approximates the original function passing through the given data points.

In mathematical terms, the interpolation polynomial $p(x)$ is a polynomial function of degree at most n that passes through all $n + 1$ given points. The values x_i are referred to as nodes, while the points (x_i, y_i) are called interpolation points. The goal of polynomial interpolation is to find a polynomial function that provides a reasonable approximation of the original function over the interval of interest based on the available data points. The polynomial function of degree n for $n + 1$ data points is denoted by p_n and can be written by Equation 2.10.

$$p_n(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots . + c_1 x + c_0 = \sum_{i=0}^{n} c_i x^i \qquad (2.10)$$

where $c_i \in \mathbb{R}$, $i = 0, .., n$ are real coefficients [15]. The interpolation polynomial can be obtained through a variety of methods, such as the Lagrange interpolation formula, the Newton divided difference formula, or the Vandermonde matrix method. These methods involve solving a system of linear equations or performing mathematical operations on the given data points to derive the coefficients of the polynomial. Figure 2.21 displays the polynomial interpolation on the function $f(x)$.

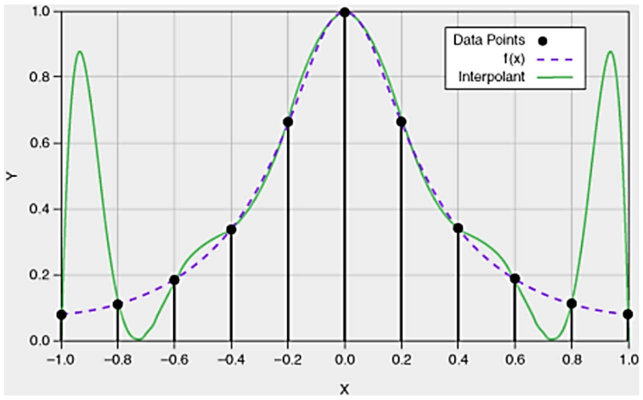


Figure 2.21: Polynomial Interpolation with Available Data Points in the Interval [-1, 1] [14]

2.5.3 Spline Interpolation

Spline interpolation is a technique used for interpolation that results in a smoother curve than linear interpolation. The term "spline" originated from a "flexible" strip used to draw smooth curves through data points.

The spline interpolation technique uses a specific type of piecewise polynomial known as a spline. A piecewise polynomial function smoothly connects a set of given data points, Figure2.22. Rather than attempting to fit a single, high-degree polynomial into all data values, spline interpolation fits low-degree polynomials to small subsets of the data. Subintervals are taken as $[x_i, x_{i+1}]$, $i = 0,1,..n$. For example, it may use nine cubic polynomials to fit between each of the pairs of ten points instead of using a single degree-ten polynomial for all of them.

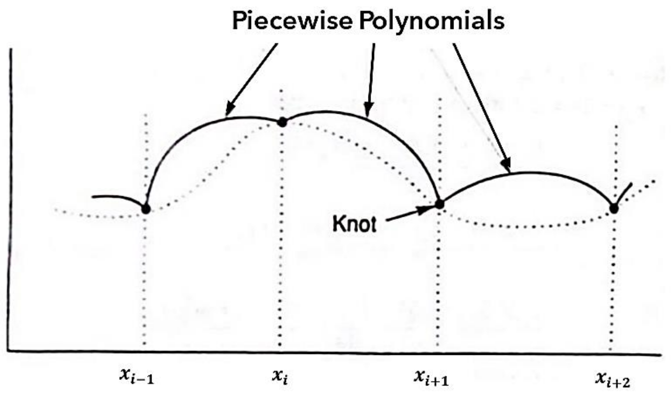


Figure 2.22: Piecewise Polynomial Interpolation [17]

Piecewise polynomials in Figure 2.22 have breaks at the interpolating points. Splines, on the other hand, prevent such discontinuities, appearing smooth at the knots (connecting points) where polynomial pieces are tied together, as shown in Figure 2.23. In general, a spline function $S(x)$ with a degree m needs to fulfill certain criteria. Firstly, $S(x)$ is a polynomial of degree at most m in each subinterval$[x_i, x_{i+1}]$, $i = 0,1,2,..n$. Then, $S(x)$ and its derivatives of order $1,2,\ldots,m-1$ are continuous in the range $[x_0, x_n]$.

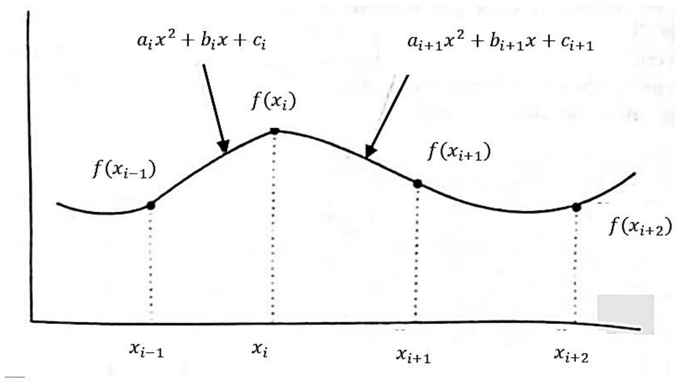


Figure 2.23: Second Degree Spline Polynomials [17]

Spline interpolation is often preferred over polynomial interpolation since its higher accuracy with less computational effort even when using low-degree polynomials. Furthermore, it has the potential to prevent the oscillation problem between large datasets, also known as the Runge phenomenon, which can manifest when interpolating with high-degree polynomials [16]. Spline interpolation can be categorized based on the continuity of the function at data points and the degree of polynomial function employed. Linear, cubic, Akima, B-spline, and natural spline interpolation are among the most frequently used spline interpolation categories. The interpolation of the spline interpolation closely resembles function $f(x)$ when additional known data points are available, as illustrated in Figure 2.24.

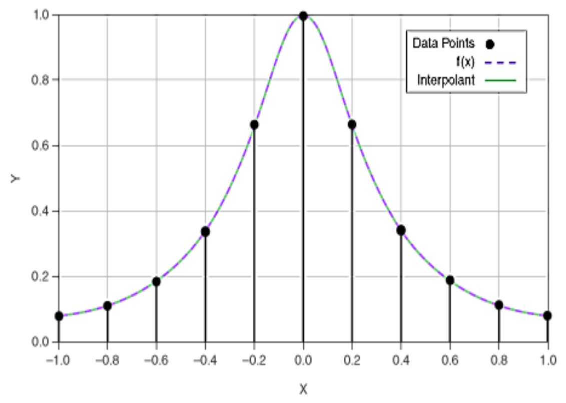


Figure 2.24: Spline Interpolation with more Data Points, Cubic Type [51]

2.5.3.1 Cubic Spline Interpolation

A cubic spline was originally established and developed by James Ferguson more than a century ago. Compared to other interpolating polynomials such as the Lagrange polynomial and the Newton polynomial, this method produces a smoother polynomial with less error. Cubic splines are splines that are composed of polynomials with a degree of less than 3 on each subinterval and have two continuous derivatives across the entire spline.

Given a set of n data point, it is possible to find an infinite number of curves that pass through all the points in ascending order, as illustrated in Figure 2.25a. However, a single, hypothetical smooth curve through the points can be followed by the human eye, particularly when they are arranged in a recognizable pattern. Having an algorithm that can accomplish the same thing is beneficial. In order to create the desired curve, it is imperative to adopt an interactive approach, as computers are not capable of identifying familiar patterns like humans. This can be achieved through the use of the cubic spline method algorithm. By using n data points, it generates a curve that is smooth and passes through the points. The curve is formed by the smooth connection of $n-1$ individual segments at $n-2$ interior points, which is simple to calculate and exhibit. At every interior point, the segments must have matching tangent vectors, first derivative, in order to intersect. Cubic splines have the added feature of having the same second derivatives at interior points. Interactivity is a feature of the cubic spline method.

Considering the set of n data points P_1 through P_n, the objective is to find a set of $n-1$ parametric cubic $P_1(t)$, $P_2(t)$, .., $P_{n-1}(t)$, where $P_k(t)$ represents the polynomial segment between P_k and P_{k+1}, as illustrated in Figure 2.25b. The first derivatives of parametric cubic at every interior point, including $P_2, P_3, \ldots, P_{n-1}$, must match to achieve a smooth connection. For a spline to be defined, it is required that their second derivatives match. For example, To achieve a smooth connection between segments $P_k(t)$ and $P_{k+1}(t)$ at point P_{k+1}, it is necessary that the end tangent of $P_k(t)$ be equivalent to the start tangent of $P_{k+1}(t)$. For a total of n unknowns, there can only be one tangent vector per point.

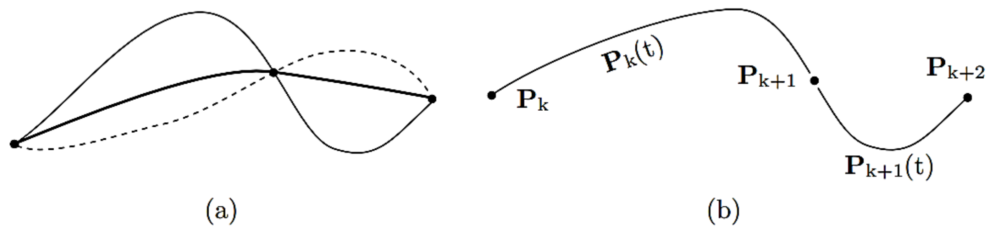


Figure 2.25: (a) Three Different Curves. **(b)**Two Segments [52]

Interpolation on all data points will be executed by the piecewise function $S_i(x)$. The use of this notation is favored over $P_k(t)$ for representing the cubic spline function and is defined by Equation 2.11[55].

$$S(x) = \begin{cases} S_1(x) & if \quad x_1 \leq x \leq x_2 \\ S_2(x) & if \quad x_2 \leq x \leq x_3 \\ \quad . & \\ \quad . & \\ S_{n-1}(x) & if \quad x_{n-1} \leq x \leq x_n \end{cases} \tag{2.11}$$

where $S(x)$ is a third degree polynomial defined by

$$S_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i)^1 + d_i \quad \text{for } \; i = 1,2,\ldots,n-1 \tag{2.12}$$

The first and second derivatives of these $n - 1$ equations play a fundamental role in this process (Equations 2.13 and 2.14).

$$S_i'(x) = 3a_i(x - x_i)^2 + 2b_i(x - x_i)^1 + c_i \tag{2.13}$$

$$S_i''(x) = 6a_i(x - x_i)^1 + 2b_i \quad \text{for } \; i = 1,2,\ldots,n-1 \tag{2.14}$$

To satisfy the interpolation conditions, it is required that the piecewise function $S(x)$ initially interpolate all data points,Equation2.15. Additionally, $S(x)$ must be continuous on the interval $[x_1, x_n]$, and both $S'(x)$ and $S''(x)$ must also be continuous on the same interval shown by Equations 2.16 and 2.17.

$$S_{i-1}(x_i) = S_i(x_i) \qquad 2 \le i \le n-1 \tag{2.15}$$

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \qquad 0 \le i \le n-2 \tag{2.16}$$

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \qquad 0 \le i \le n-2 \tag{2.17}$$

2.6 Regular Grid Interpolator

The process of regular grid interpolation involves calculating the values of a function at points that are evenly distributed within a grid. The interpolation method of a regular or rectilinear grid can be used in any number of dimensions. A rectilinear grid is required to define the data. More precisely, a rectangular grid displays either regular or irregular spacing. Linear interpolation, nearest-neighbor interpolation, and spline interpolation are all supported by this system.

Once the interpolator object has been configured, the user can determine which interpolation method to use for each evaluation. There are various methods that can be utilized in the interpolation process, including linear, nearest, slinear, cubic, quintic, and pchip. The implementation of multivariate interpolation for data on a grid in Python using $scipy.interpolate$ is demonstrated by the following syntax.

$$scipy.interpolate.RegularGridInterpolator(points,\ values,\ method,\ bounds\_error,\ fill\_value) \tag{2.18}$$

The first parameter assigned to this class is typically referred to as points. A set of points represents the regular grid in n dimensions. The points must be structured as a tuple of ndarrays of floats. It is required that all elements of the points tuple follow a strictly ascending or descending pattern with no deviation [22]. Moreover, values are array-like and are defined as data on a regular grid in n dimensions. The process involves the estimation of the values of a continuous function at various points within a grid that is regular in nature. The estimation is based on values that have been previously identified at specific grid points.

2.7 Optimization

Optimization, also known as mathematical programing, employs mathematical principles and techniques to solve problems that involve numerical data in various fields, including physics, biology, engineering, economics, and business [25]. In recent years, optimization techniques have rapidly progressed, resulting in significant advancements. Concurrently, digital computers experience an increase in speed, flexibility, and efficiency. As a result, the potential to tackle complex optimization problems that were thought to be unachievable just a few years ago is now attainable. The best solution or optimal value can be obtained by means of the optimization process. The optimization process entails three essential stages: comprehending the system, measuring system efficiency, and utilizing a suitable optimization algorithm to obtain the solution while analyzing the degrees of freedom. The optimization problem in general can be formulated as follows.

$$minimize\ f(x) \tag{2.19a}$$

$$x = [x_1, x_2, \dots, x_n]^T \in \mathbb{R}^n \tag{2.19b}$$

subject to the constraints:

$$h_j(x) = 0, \qquad j = 1,2,\dots,r \tag{2.19c}$$

$$g_j(x) \leq 0, \qquad j = 1,2,\dots,m \tag{2.19d}$$

The primary objective of an optimization problem is to determine the decision variables x that optimize the objective function $f\,(x)$ while ensuring that the model functions within the fixed limits imposed by the equality constraints $h_j(x)$ (Equation 2.19c) and inequality constraints $g_j(x)$ (Equation 2.19d).

The employed numerical optimization technique is schematically illustrated in Figure 2.26, which depicts the iterative procedure. Figure 2.26 demonstrates how the optimizer starts with initial values and passes a set of decision variable values represented by x to invoke the model. The model replicates the phenomenon and computes both the objective function and constraints. A new set of decision variables is computed by the optimizer using this information. The iterative sequence continues until the optimization criteria associated with the optimization algorithm are fulfilled and the optimal design is found [24].

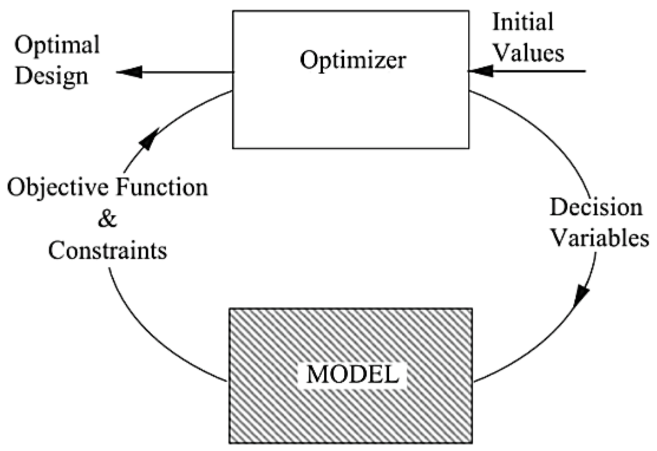


Figure 2.26: Pictorial Representation of the Numerical Optimization Framework [24]

Optimization problems can be categorized into different types based on decision variables, objective functions, and constraints. These categories include Linear Programming (LP), Nonlinear Programming (NLP), Mixed Integer Linear Programming (MILP), Mixed Integer Nonlinear Programming (MINLP), stochastic optimization, and Multi-Objective Optimization. Also, Constrained optimization and Unconstrained optimization are two essential classifications in mathematical optimization. In constrained optimization, the goal is to optimize the objective function while ensuring that feasible solutions comply with a set of constraints. Unconstrained optimization is the term used for problems without constraints.

Various optimization algorithms belonging to different categories of optimization problems are presented below to illustrate how optimization problems can be addressed. The presented techniques for solving optimization problems comprise BFGS, Nelder-Mead, Newton-CG, and Sequential Quadratic Programming (SQP).

2.7.1 BFGS Method

The BFGS method (Broyden, Fletcher, Goldfarb, and Shanno) is an iterative method used to solve unconstrained nonlinear optimization problems. The purpose of this algorithm is to optimize local searches for convex problems that contain a singular optimum.

By employing the second-order derivative of an objective function, this algorithm is classified as a second-order optimization algorithm. It pertains to a category of algorithms commonly known as Quasi-Newton methods, which provide an estimate for the second derivative (known as the Hessian) in cases where it is infeasible to calculate it for optimization problems. Quasi-Newton methods approximate the inverse of the Hessian matrix using the gradient, meaning that the Hessian and its inverse do not need to be available or calculated precisely for each step. The BFGS algorithm offers a specific approach to updating the computation of the inverse Hessian as opposed to re-computing it every iteration. BFGS and its extension, L-BFGS or LM-BFGS, are considered among the most widely used Quasi-Newton or second-order optimization algorithms in numerical optimization. Similar to the first-order method, the BFGS algorithm eases the determination of the movement direction while additionally requiring a line search in the designated direction to determine the step size.

Among quasi-Newton methods for unconstrained optimization, the BFGS algorithm is widely acknowledged as one of the most efficient. Typically, the formulation of the BFGS algorithm is as follows [33].

$$min\, f(x),\ \ x \in R^n \tag{2.20a}$$

$$x_1 \in R^n \tag{2.20b}$$

$$B_1 \in\ R^{n\times n} \tag{2.20c}$$

The mathematical formulation is defined as positive definite entities; Equation 2.20a, in which x_1 represents the starting point and B_1 serves as an approximation to the Hessian. It is requested to compute the gradient of the function by utilizing Equation 2.21. In the event that g_1 equals zero, the process must be terminated.

$$g_1 = \nabla \mathrm{f}(x_1) \tag{2.21}$$

The searching direction of BFGS can be computed using Equation 2.22, where ${B_k}^{-1}$represents the inverse of the Hessian matrix and g_k denotes the gradient of the function.

$$d_k = {B_k}^{-1} g_k \tag{2.22}$$

The iteration points are addressed by means of the following iteration formula,Equation 2.23a. This involves a line search along d_k, where α_k represents the step length or step size and x_k denotes the k-th iterative point.

$$x_{k+1} = x_k\ + \alpha_k d_k \tag{2.23a}$$

$$\alpha_k > 0 \tag{2.23b}$$

Next, calculate the gradient of the following point using Equation 2.24. If g_{k+1} equals zero, the process will terminate, and g_{k+1} will be considered the estimated optimal solution. If not, the following step is executed.

$$g_{k+1} = \nabla f\ (x_{k+1}) \tag{2.24}$$

Equation 2.25a defines the update of the Hessian matrix, B_k ,as follows.

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{s_k^T y_k} \tag{2.25a}$$

where

$$s_k = \alpha_k d_k \tag{2.25b}$$

$$y_k = g_{k+1} - g_k \tag{2.25c}$$

The process must begin again by assigning the new value of d_k, once $k\ = k + 1$.

2.7.2 Nelder - Mead

The Nelder-Mead technique dates back to 1965, when it was introduced by John Nelder and Roger Mead. The Nelder and Mead (NM) methodology has developed the simplex search algorithm for identifying a local minimum of a function with several variables. Through the use of simplex, a generalized N-dimensional triangle, the algorithm is capable of determining the minimum of a function with N variables. The working principle of the algorithm is based on the application of a shape structure referred to as a simplex, which encompasses n + 1 points (vertices). In this case, n refers to the count of dimensions that are present in the input function. As an illustration, when dealing with a two-dimensional problem that can be graphed as a surface, the shape structure would consist of a triangle represented by three points. In contrast, a three-dimensional simplex is a tetrahedron. The methodology uses a heuristic that compares the value of the function at different points, commonly known as the "simplex", and gradually approaches improvements.

The Nelder-Mead algorithm was created for the purpose of resolving the classical unconstrained optimization problem, in which a nonlinear function must be minimized. This direct search method, which relies on function comparison, is frequently employed in solving nonlinear optimization problems where derivatives are unknown. Given that Nelder-Mead is an optimization algorithm based on pattern search, it is well-suited for problems with non-smooth functions since it does not require derivatives or gradient information. Additionally, it has the potential to be utilized for discontinuous functions, which are commonly observed in the fields of statistics and experimental mathematics.

Through an iterative implementation of the Reflection, Expansion, Outside Contraction, Inside Contraction, and Shrinkage operations, the NM method employs the objective function values associated with each vertex of the simplex, which are the fundamental transformations of the simplex. The five operations are presented in two-dimensional space in Figure 2.27. In this figure, y^0 , y^1 , and y^2 are the vertices of the simplex before the operations, such that $f(y^0) < f(y^1) < f(y^2)$ [34].

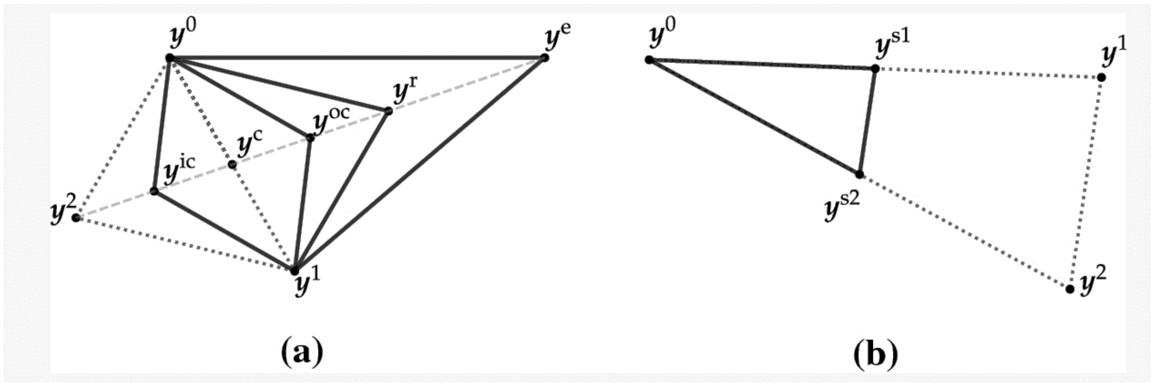


Figure 2.27: Simplex Transformations by the NM Method: **(a)** Reflection (y^r), Expansion (y^e), Outside Contraction (y^{oc}), Inside Contraction (y^{ic}), **(b)** Shrinkage (y^{s1} and y^{s2})[34]

2.7.3 Newton-CG

The abbreviation Newton-CG represents Newton-conjugate gradient, which refers to a set of algorithmic procedures created in the 1980s. Refinements have been made to the procedures, allowing for worst-case complexity outcomes to be established in achieving convergence to points that meet approximate optimality criteria of the first and second order. To minimize a smooth non-convex objective function, the Newton-conjugate gradient algorithm combines an iterative approach based on both Newton's method and the linear conjugate gradient algorithm. This approach involves the explicit identification and application of negative curvature directions for the Hessian of the objective function.

Non-convex smooth function minimization without constraints, involving numerous variables, has been extensively studied in the optimization field. Approaches such as Limited memory BFGS, nonlinear conjugate gradient, and Newton-CG are commonly used to handle high dimensions, like n-dimension. The Newton-CG method employed the conjugate gradient (CG) method, with linearity, to apply the second-order Taylor-series approximation of f around the present iterate x_k. Every CG iteration necessitates computing a Hessian-vector product resembling $\nabla^2 f(x_k)v$ [35].

A Newton-CG variant that employs trust regions terminates CG iterations upon achieving adequate accuracy in minimizing the quadratic approximation, when a CG step exits the trust region, or when encountering a negative curvature in $\nabla^2 f(x_k)$ [36]. A line-search variant is employed in the applied CG algorithm until either negative curvature is identified or a convergence criterion is met. The search direction is reversed to the negative gradient in the case of negative curvature.

2.7.4 Sequential Least-Squares Quadratic Programming

Constrained optimization problems are resolved using the numerical optimization algorithm known as Sequential Least Squares Quadratic Programming (SLSQP). The SLSQP optimization subroutine, which was devised by Kraft in [37], is the basis for SLSQP, a gradient-based approach used to solve nonlinear optimization problems with constraints. SLSQP is capable of function minimization of multiple variables, subject to different types of constraints, including bounds, equalities, and inequalities. This algorithm is particularly advantageous for instances where nonlinear objective functions and constraints are present. Through iterative variable updates, SLSQP aims to minimize the objective function while satisfying the given constraints and achieving the optimal solution.

The algorithm integrates the concepts of the sequential quadratic programming (SQP) method and the least squares method. The problem is addressed as a sequence of constrained least-squares problems, which can be reformulated as a quadratic programming (QP) problem, providing the ability to control the iteration direction. It approximates the optimization problem with constraints through a series of quadratic subproblems that are resolved by minimizing the least squares. The approximation of the objective function is carried out through the utilization of a linear model and least squares minimization. It utilizes a linear approximation of the objective function and seeks to minimize the sum of squares of the residuals. The algorithm begins by making an initial approximation for the solution and then proceeds iteratively until convergence is achieved.

The form of a constrained nonlinear optimization problem can be described as follows [38]:

$$\text{minimize} \quad f(x), \qquad over\, x \in \mathcal{R}^n \tag{2.26a}$$

$$\text{subject to} \quad g_i(x) = 0, \qquad j \in E, \tag{2.26b}$$

$$g_u(x) \geq 0, \qquad u \in I, \tag{2.26c}$$

$$x_{iL} \leq x_i \leq x_{iu}, \quad i = 1, \dots, n \tag{2.26d}$$

The objective is to minimize the target scalar function (Equation 2.26a). When $x = (x_i)_{i=1}^{n}$ is an n-tuple of input variables x_i. The functions denoted in Equations 2.26b and 2.26c correspond to equality and inequality constraints, respectively. The sets of functions for each constraint type are represented by E and I. Each input variable x_i has an upper bound shown by x_{iU} and a lower bound labeled by x_{iL}, which are represented by Equation 2.26d.

SLSQP starts with an initial guess tuple x^0 and iteratively solves the optimization problem to find a local minimum. The $(l+1)$-th tuple variable of $x^{\ell+1}$ is obtained by the following process, Equation 2.27, on x^{ℓ}.

$$x^{\ell+1} = x^{\ell} + \alpha^{\ell} d^{\ell} \tag{2.27}$$

where d^{ℓ} is the search direction and α^{ℓ} is the iteration length within the ℓ-th iteration.

2.8 Optimization in GAP

The GAP optimization tool empowers the engineer to optimize a specific objective function, such as oil production, revenue, or lift gas allocation for gas-lifted wells, while following any system-defined constraints. The GAP optimizer has a dual function: guaranteeing that all constraints are satisfied and improving an objective function, like the oil rate. GAP can optimize oil production by modifying wellhead chokes on natural wells and allocating lift gas efficiently on gas-lifted wells. GAP constraints guide the optimizer algorithm. There are constraints in the GAP production system, which are as follows:

- Pressure design for flow pipe lines
- The total gas export capacity for compressors
- Control of differential pressure (DP) at the choke point of the well

The optimizer in GAP uses numerical schemes to find a solution. In order to implement these schemes, the numerical derivatives (rates of change) for each element must be calculated. To evaluate convergence, the GAP solver sets multiple tolerance criteria, such as tolerance Min and tolerance X. Optimization in GAP is powerful because it uses a non-linear optimization algorithm (NLP), such as SQP. An accurate assessment of the interactions between gas lift injection and well head pressures in the network is necessary. With GAP non-linear optimization, mixed constraints located throughout the network are handled with ease, while linear optimizers, including sequential linear optimizers, cannot find the optimum, particularly in gas lift optimization. In GAP optimization, automatic control of a dynamic wellhead choke set is an advantageous benefit that satisfies constraints [28].

Gross revenue, oil and gas production, gross heating value, and water production are the potential factors for GAP optimization. There are many ways to apply the features of GAP optimization, including:

- Full field optimization with mixed systems like ESP, gas-lifted, and naturally flowing wells
- Full field gas lift optimization; gas lift gas allocation amongst wells
- Determining control settings to achieve field management objectives, such as wellhead choke and compressor speeds
- Multi-phase looped network optimization
- Water injection system optimization
- Gas lift injection system optimization; optimizing the gas lift gas network that provides the gas to the artificially lifted wells

The primary objective of this study is to implement full-field gas lift optimization to distribute gas lift across wells.

2.9 Gas Lift Optimization in Well Network

The gas lift method is used for wells that lack adequate reservoir pressure to produce oil at desired flow rates. The process involves injecting gas as deeply as possible into the tubing to reduce the weight of the fluid hydrostatic column and decrease the backpressure on the formation. Several factors play a critical role in the effectiveness of a gas lift operation, and defining these parameters is necessary to enhance production and maximize the net present value of the operation. Among these factors, the optimum injection rate is the most significant parameter in gas lift operations. Identifying the optimal injection rate is pivotal since injecting excessive gas not only fails to improve production but can also decrease it by increasing slippage between liquid and gas phases.

Allocating the appropriate amount of gas to each well in a gas lift operation is challenging because the amount of gas injected does not have a direct relationship with the amount of oil produced. Various factors, including fluid characteristics, well completion, and surface network, influence the amount of gas required for each well. Production optimization is one of the most intricate and interdisciplinary tasks in the oil and gas industry, requiring a continuous improvement process that involves managing and optimizing production scenarios with a more frequent timeframe. Gas lift optimization involves determining the ideal distribution of gas throughout a network of wells and pipelines. Various established methods exist for addressing this non-linear problem, including sequential quadratic programming (SQP), augmented Lagrangian models (AIM), and stochastic solvers like genetic algorithms (GA) [53]. The mathematical expression for gas lift optimization is demonstrated by Equation 2.28, in which total oil production (Q_{oT}) from a set of n wells, represented by the sum of individual oil production rates (q_{oi}), can be expressed as a function of individual gas injection rates (q_{gi}).

$$Q_{oT} = \sum_{i=1}^{n} q_{oi} = f(Q_g) = f(q_{g1}, q_{g2}, \dots, q_{gn}) \qquad (2.28)$$

Total Gas injection rate , Q_g , for operation is displayed by Equation 2.29 where the notation T represents transposition.

$$Q_g = (q_{g1}, q_{g2}, \dots, q_{gn})^T \qquad (2.29)$$

Expressing the objective of identifying the optimal gas injection rates that maximize total oil production can be formulated as

$$Max Q_{oT} = Max f(Q_g) \qquad (2.30a)$$

Subject to the following constraints

$$\sum_{i=1}^{n} q_{gi} \leq Q_{gTotal} \tag{2.30b}$$

$$q_{gi} \geq 0 \; for \; i = 1,2, \dots, n \tag{2.30c}$$

$$q_{gi} \leq q_{gi\,max} \tag{2.30d}$$

Equation 2.30b imposes a constraint that mandates the summation of individual gas injection rates to be less than or equal to the total injection gas volume that is available to the system (Q_{gTotal}). The inequality Equation 2.30c provides the constraint that each gas injection rate must be greater than or equal to zero. As a result, during computation, the gas injection rates must comply with the constraints. Thus, it is necessary to ensure that the gas injection rates meet the requirements specified in Equations 2.30b and 2.30d during the optimization process. The optimization technique could potentially include more constraints, such as water cut, minimum gas lift injection rates.

Continuous-flow gas lift is an extensively applied artificial methodology for oil recovery in several regions. The primary objective is to inject an adequate amount of gas into each well to maximize revenue. However, in reality, there is often a limited supply of gas available, which makes it difficult to achieve the maximum output from every well in the field. Therefore, it becomes determined to distribute the gas in a restricted manner among the wells to optimize oil production from the entire field.

Four key components must be analyzed to achieve the most effective gas lift system, including flow through a porous medium, flow through a production string, flow through a flowline and trunk line, and the total injection gas volume available for the system. Due to the limited injection gas volume, precise distribution of injection gas is an important element in the design of a gas lift system. The primary aim of gas allocation optimization in this study is to maximize the total oil production rate from the gas lift system, given the available gas volume to support the system. This objective can be seen as a nonlinear function that seeks to maximize the total oil production rate. The gas injection rates for each well are the variables or unknowns in this function, subject to physical constraints [54].

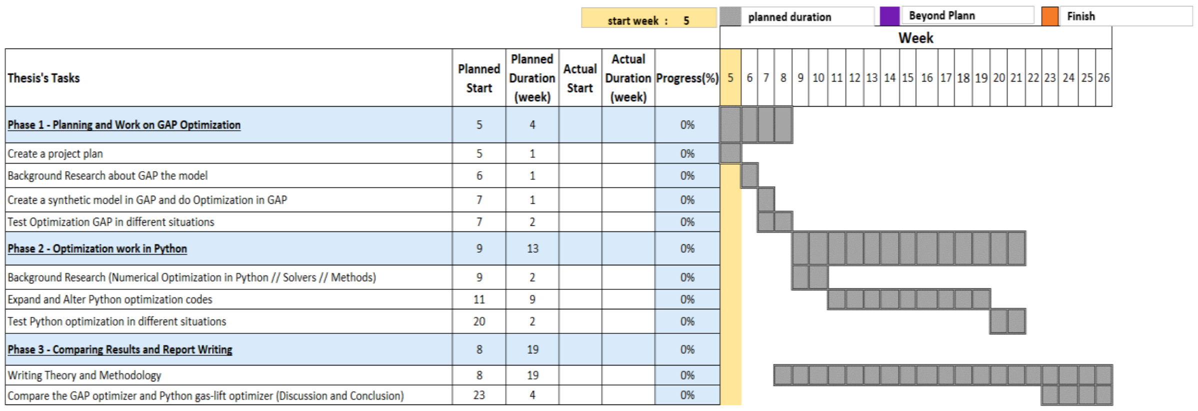
Chapter 3

Methodology

The methodology chapter provides a comprehensive overview of the working system, outlining the process of code generation and development, and giving a brief explanation of the working process in GAP and PROSPER. The first part of this chapter presents an overview of the planning and execution of the thesis, followed by a description of how the GAP and PROSPER well-models are employed in the study. In addition, the code implemented for importing data from the six well-models into Python is described, together with relevant theoretical explanations that support the understanding of the process. Prior to describing the implementation of interpolation code and unit conversion, a distinct figure illustrates the mathematical formulation of interpolation. An interpolation object is assigned to each well to perform interpolation at any desired point. Also, a function is defined to calculate the equilibrium rate for each well, which is then used to determine the water, oil, and gas rates per well. Afterward, the intention and mechanism of the optimization problem are defined, resulting in the determination of the optimal gas lift injection and oil rate values for all wells as well as the entire field. At the end of this chapter, a flowchart diagram provides a visual representation of the entire design of the system.

3.1 Planning and Execution

In order to create a schedule for the master's thesis, a Gantt chart was created as part of the planning stage. The graph below provides a visual representation of the project timeline via bar charts that demonstrate the start and completion of each part [21]. The primary schematic of the Gantt chart is represented in Figure 3.1. Throughout the work, there are three phases, each containing multiple tasks. It outlines the number of weeks allocated to each task and, at the end, shows the progress status of each task.



Thesis's Tasks	Planned Start	Planned Duration (week)	Actual Start	Actual Duration (week)	Progress(%)
Phase 1 - Planning and Work on GAP Optimization	5	4			0%
Create a project plan	5	1			0%
Background Research about GAP the model	6	1			0%
Create a synthetic model in GAP and do Optimization in GAP	7	1			0%
Test Optimization GAP in different situations	7	2			0%
Phase 2 - Optimization work in Python	9	13			0%
Background Research (Numerical Optimization in Python // Solvers // Methods)	9	2			0%
Expand and Alter Python optimization codes	11	9			0%
Test Python optimization in different situations	20	2			0%
Phase 3 - Comparing Results and Report Writing	8	19			0%
Writing Theory and Methodology	8	19			0%
Compare the GAP optimizer and Python gas-lift optimizer (Discussion and Conclusion)	23	4			0%

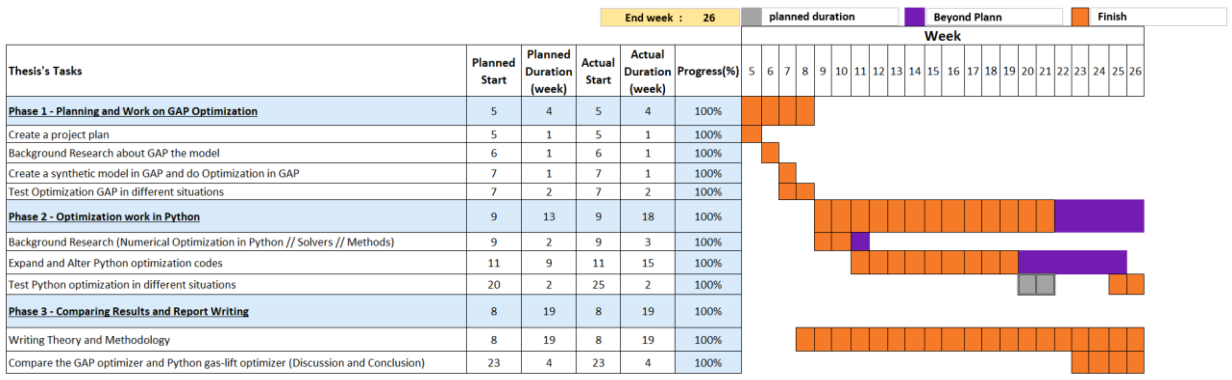
Figure 3.1: The Initial Representation of the Gantt Chart Utilized in the Planning Stage of a Master's Thesis

3.1.1 Project Execution

During the project timeline, persistent communication was sustained with the supervisors via biweekly virtual meetings on Teams and supplementary emails to address any complications or uncertainties encountered. The scheduled sessions had a methodical agenda, which included reviewing accomplished milestones since the last meeting and addressing inquiries regarding upcoming tasks. Progress was kept on track during meetings by regularly updating the Gantt chart, as shown in Figure 3.1. Positive discussions and collaborations were made possible by using the Gantt chart as a central reference point to organize and direct research activities in this thesis.

As expected, during any project such as this one, certain elements of the original plan experienced modifications along the way. The revised Gantt chart, shown in Figure 3.2, shows how closely the tasks were completed to their planned deadlines. The modified chart emphasizes any variations from the original plan and acts as a visual depiction of how the project is evolving. By comparing the planned tasks in Figure 3.1 with the revised Gantt chart in Figure 3.2, it is possible to identify modifications and assess how closely the project followed the deadline.

The graphical illustration provided by Figure 3.2 displays how the initial four weeks of the project were dedicated to phase one. This stage consisted of fundamental activities such as planning, conducting comprehensive research on the GAP model, constructing a synthetic model using GAP framework principles, and rigorously testing GAP optimization across diverse scenarios. Impressively enough, it was managed to complete these tasks within designated timeline. The subsequent stage - phase two - revolved around extending or altering existing codes to incorporate a GAP synthetic case, well-network, in Python. This phase entailed tasks such as importing six well-model files into Python, carrying out table interpolation activities and executing and evaluating optimization process under varying conditions. It is important to note that this stage ended up taking longer than initially expected.



End week : 26 | ▨ planned duration | 🟪 Beyond Plann | 🟧 Finish

Thesis's Tasks	Planned Start	Planned Duration (week)	Actual Start	Actual Duration (week)	Progress(%)	Week																					
						5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Phase 1 - Planning and Work on GAP Optimization	5	4	5	4	100%	🟧	🟧	🟧	🟧																		
Create a project plan	5	1	5	1	100%	🟧																					
Background Research about GAP the model	6	1	6	1	100%		🟧																				
Create a synthetic model in GAP and do Optimization in GAP	7	1	7	1	100%			🟧																			
Test Optimization GAP in different situations	7	2	7	2	100%			🟧	🟧																		
Phase 2 - Optimization work in Python	9	13	9	18	100%					🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟪	🟪	🟪	🟪	🟪
Background Research (Numerical Optimization in Python // Solvers // Methods)	9	2	9	3	100%					🟧	🟧	🟪															
Expand and Alter Python optimization codes	11	9	11	15	100%							🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟪	🟪	🟪	🟪	🟪	🟪	
Test Python optimization in different situations	20	2	25	2	100%																▨	▨				🟧	🟧
Phase 3 - Comparing Results and Report Writing	8	19	8	19	100%																						
Writing Theory and Methodology	8	19	8	19	100%				🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
Compare the GAP optimizer and Python gas-lift optimizer (Discussion and Conclusion)	23	4	23	4	100%																			🟧	🟧	🟧	🟧

Figure 3.2: The Final Representation of the Gantt Chart Utilized in the Planning Stage of a Master's Thesis

3.1.2 Development of Codes

The process of optimizing the well network for maximum oil production rates in a field through optimal gas lift injection rates, while considering several constraints, required a multi-step adjustment to the previous code [40]. This procedure involved several essential components, which are briefly explained in the following list.

1. Import the data from well-models, PROSPER TPD files, into Python and appropriately store the required data in a suitable data structure.
2. Develop an interpolator employing multi-linear interpolation techniques to interpolate the imported data.
3. Construct an interpolation object for each well, enabling the evaluation of interpolated values at specific input parameters for testing purposes.
4. Construct a function (or class) for every well with the purpose of computing the liquid equilibrium rate and subsequently determining the oil production rate (q_o), water production rate (q_w), and gas production rate (q_g).
5. Create a function (or class) to do optimization on oil production rates $(q_o - field)$ when gas-lift rate is considered an independent variable, allowing for the identification of optimal gas-lift rates that maximize oil production while considering various constraints. Then, based on that calculate the water production rates $(q_w - field)$, and gas production rates $(q_g - field)$.

3.2 GAP Network Model

A surface network model was developed for a field that includes six oil producer-gas lifted well types with the aid of GAP software. This model included wells, wellheads, flow lines that connect the wellhead to the production manifold, and the production line leading to the low-pressure separator, Figure 3.3. First, the wells were connected to the manifolds from the wellhead points, and then to the separator from the manifolds.

The goal of constructing a network model for gas-lifted wells was to efficiently manage the gas lift of the entire field. This was accomplished by distributing gas to each well in the best possible manner, which led to maximum oil production while meeting various technical and operational restrictions. Optimization of the gas lift required careful consideration of the capacity of the separator in the field. To evaluate the efficacy of the GAP optimizer, gas-lift optimization was conducted for various situations, including gas-lift limitations, water production constraints, and the optimization of both gas lift rates and choke opening.

A simulation model of a synthetic field case was prepared through modifications made to Example 7 in the GAP manual [27]. The alterations comprised the elimination of reservoir tanks because the optimization approach employed in the thesis work pertains to a production model rather than a prediction model, and hence, no reservoirs were integrated into the system. After running the network solver, it was determined that the wells without gas lift were unnecessary and were removed. Moreover, placing the pipes on Bypass could eliminate the pipe segments. The reason was that the pipeline itself caused a pressure drop and decreased production; hence, no extra pressure loss from the pipes should be considered. Well scheduling and separator scheduling were excluded from the thesis, as it concentrated on a specific point without considering prediction or future performance, which is noteworthy.

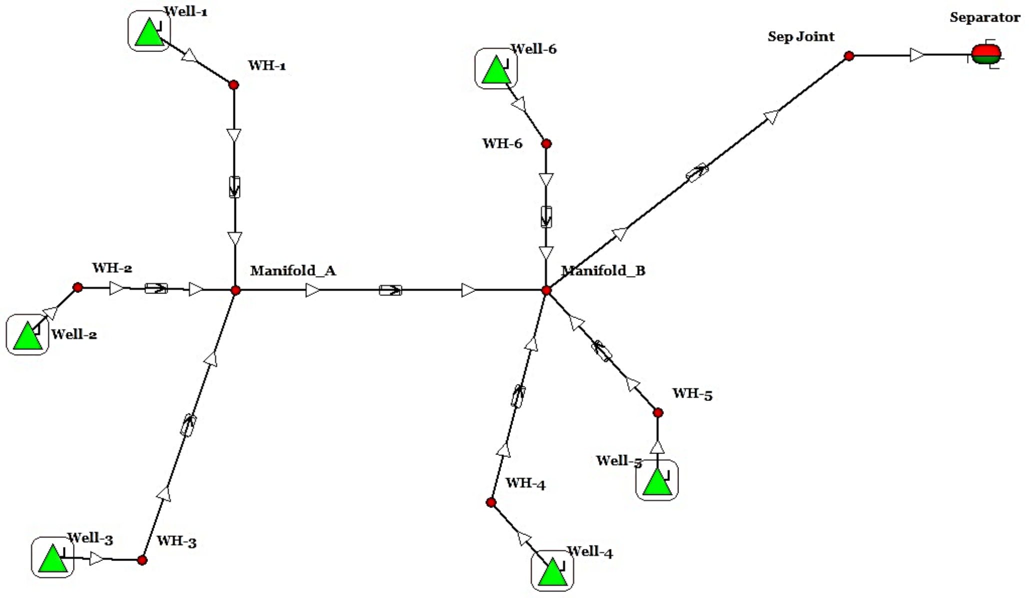


Figure 3.3: Synthetic Field Case from GAP

Within the GAP program, the network solver presented four calculation options.

- No Optimization
- Rule Based
- Optimize with all constraints
- Optimize potential constraints only

The aim of this thesis work was fulfilled by executing the network solver calculation with optimization that included all constraints. Limited total gas lift injection, compressor restrictions, and water and liquid production rate capacity defined the constraints for the field.

3.2.1 Well Models in PROSPER

The major targets of optimizing daily production consist of maximizing the production of oil and gas while minimizing related expenses. Production planning and decision-making for oil wells in complicated reservoirs is a challenging task. PROSPER and GAP have commonly employed software tools for this purpose. The software tools are meant to resolve non-linear optimization problems by employing steady-state well models. In order to qualify the model for production optimization, it is necessary to generate Vertical Lift Performance (VLP) tables of the wells by using the black box simulator. The term Vertical Lift Performance (VLP) is used interchangeably with other names, such as Lift Curves (LC), Tubing Performance Relationship (TPR), and Vertical Flow Performance (VFP). Prosper was utilized for well modeling, providing users with artificial lift design and modeling capabilities. Nodal analysis was used in this thesis for wells with gas lift injection [23]. The VLP curve can be influenced by various factors, such as fluid PVT properties, production rate, well depth, tubing size and diameter, surface pressure, water cut, and Gas-oil ratio (GOR). VLP curves typically characterize the connection between the pressure of the top node and the pressure of the bottom hole for several flow rates. Through the generation of VLP curves with varying ranges of parameters and exporting them to various formats, they can be employed in combination with other software.

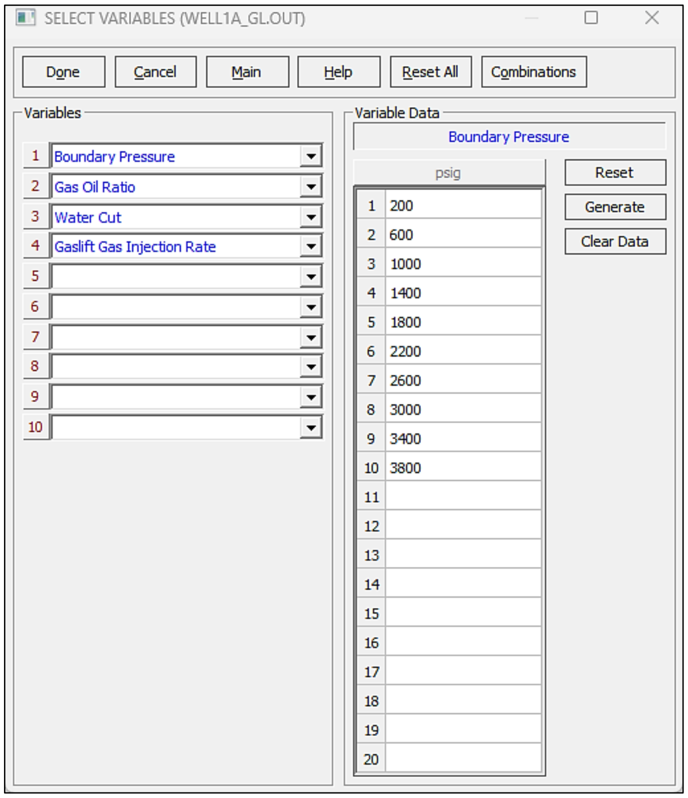
The process to produce VLP tables was begun by selecting the calculation menu from the main toolbar of PROSPER and then choosing the VLP (Tubing Curves) option. Establishing the sensitivity parameters was an essential requirement for producing VLP tables by using the Sensitivity Cases screen. These variables elevated the precision of the VLP curves, which were later used in other programs, for instance, Python, in this thesis. The sensitivity variables generated should cover the complete spectrum of potential well operating conditions. It was wise to prepare a table of VLP cases that covers all variables, as it eliminated the need to regenerate the table if conditions change. The VLP cases of six gas lifted wells were generated by considering various factors, such as Boundary Pressure (Top Node Pressure), Gas-Oil Ratio, Water Cut, Gas lift Injection Rate, and Liquid Rate. Table 3.1 displays the sensitivity variable ranges used in this study to generate the lift curves.

Table 3.1: The Range of Sensitivity Variables and the Corresponding Count of each Variable in Oilfield Unit

Sensitivity Variables	Units	First Value	Last Value	Number
QLIQ	(STB/day)	400	20000	10
QGL	(MMscf/day)	1	50	10
WC	(%)	0	95	10
GOR	(scf/STB)	100	2000	10
WHP	(psig)	200	3800	10

The variables presented in Table 3.1 include Liquid Rate Values (QLIQ), Gas Lift Injection Rate (QGL), Water Cut (WC), Gas-Oil Ratio (GOR), and Well Head Pressure (WHP), which shows the boundary pressure. In addition, Table 3.1 exhibits the identification of the initial and final values and the number of values that were determined for all variables.

Figure 3.4 demonstrates the positioning of four sensitivity variables in PROSPER, namely Boundary Pressure, Gas-Oil Ratio, Water Cut, and Gaslift Gas Injection Rate on the left-hand side, while the right-hand side depicts the ten-point volume for the Boundary Pressure. Their generation primarily employed Linear Spacing, apart from the use of Geometric Spacing for Gaslift Gas Injection Rate.



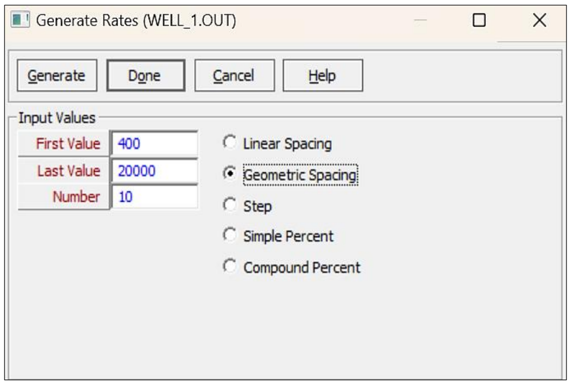
	Variables
1	Boundary Pressure
2	Gas Oil Ratio
3	Water Cut
4	Gaslift Gas Injection Rate
5	
6	
7	
8	
9	
10	

Boundary Pressure

	psig
1	200
2	600
3	1000
4	1400
5	1800
6	2200
7	2600
8	3000
9	3400
10	3800
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	

Figure 3.4: Sensitivity Case Values for WELL_1 in Sensitivity Screen

Subsequently, the liquid rate (QLIQ) was generated using the "Generate" option present in the VLP toolbar. The liquid rate (QLIQ) presented in Figure 3.5 was produced through the selection of initial and final values, the amount of values, and the application of the Geometric Spacing distribution method.


Generate Rates (WELL_1.OUT)

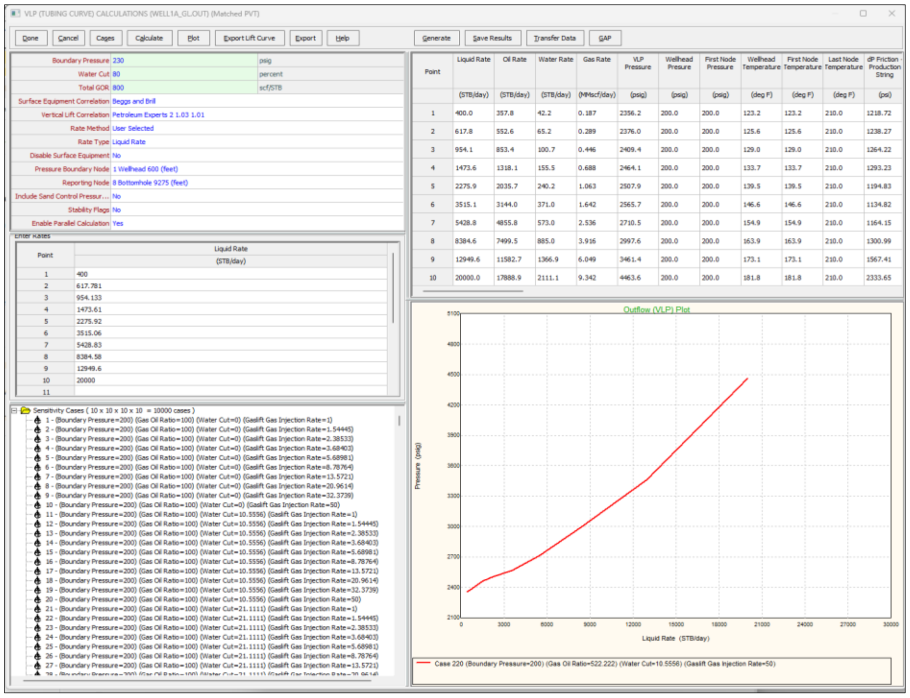
Input Values

First Value	400
Last Value	20000
Number	10

Linear Spacing [✓ Geometric Spacing] Step Simple Percent Compound Percent

Figure 3.5: Generating Rates for Calculation Process Well Performance Model, WELL_1

After inputting the sensitivity variables and liquid rate, the generation of lift curves was initiated by selecting the "Calculate" option on the VLP top toolbar (Figure 3.6). The lift curves that were produced can be exported to the appropriate format by selecting the "Export Lift Curve" button. In this work, "Petroleum Expert-GAP/MBAL" was utilized as a VLP format.


VLP (TUBING CURVE) CALCULATIONS (WELL1A_GL.OUT) (Matched PVT)

Boundary Pressure	230	psig
Water Cut	80	percent
Total GOR	800	scf/STB
Surface Equipment Correlation	Beggs and Brill	
Vertical Lift Correlation	Petroleum Experts 2 1.03 1.01	
Rate Method	User Selected	
Rate Type	Liquid Rate	
Disable Surface Equipment	No	
Pressure Boundary Node	1 Wellhead 600 (feet)	
Reporting Node	8 Bottomhole 9275 (feet)	
Include Sand Control Pressur...	No	
Stability Flags	No	
Enable Parallel Calculation	Yes	

Enter Rates

Point	Liquid Rate
	(STB/day)
1	400
2	617.781
3	954.133
4	1473.61
5	2275.92
6	3515.06
7	5428.83
8	8384.58
9	12949.6
10	20000
11	

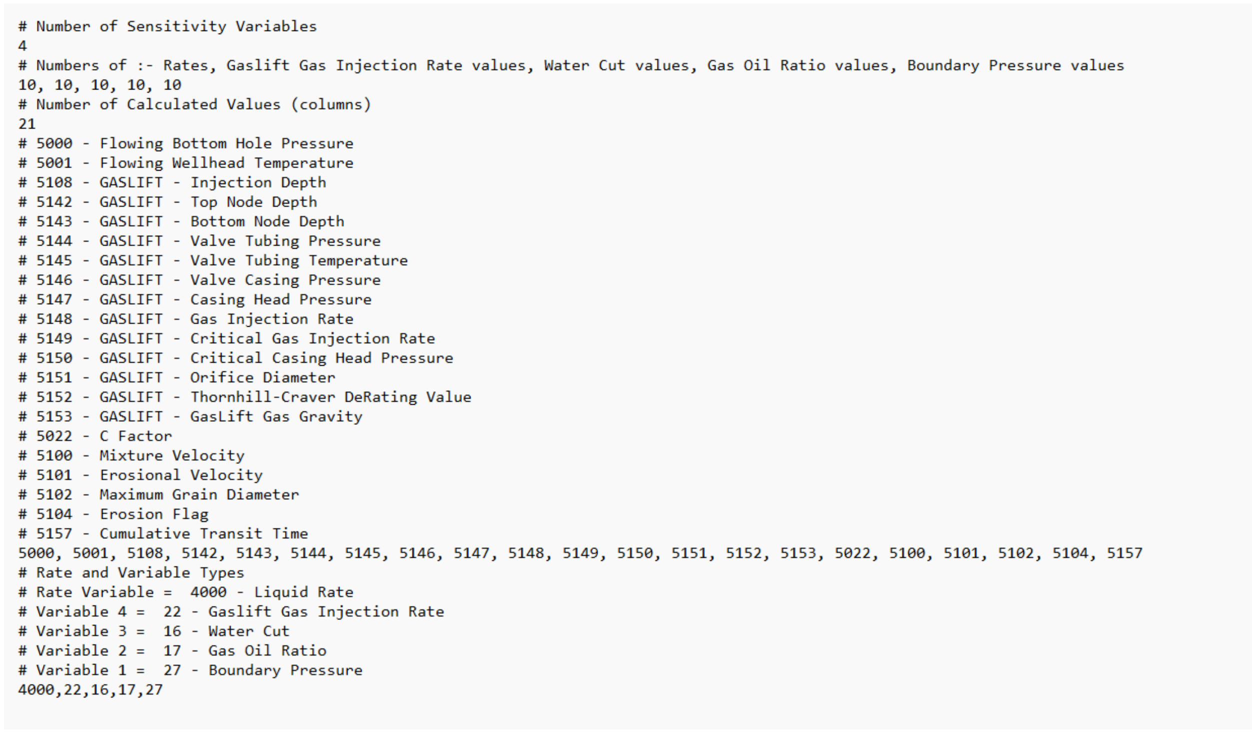
Sensitivity Cases (10 x 10 x 10 x 10 = 10000 cases)
1 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=0) (Gaslift Gas Injection Rate=1)
2 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=0) (Gaslift Gas Injection Rate=1.54445)
3 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=0) (Gaslift Gas Injection Rate=2.38533)
4 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=0) (Gaslift Gas Injection Rate=3.68403)
5 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=0) (Gaslift Gas Injection Rate=5.68981)
6 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=0) (Gaslift Gas Injection Rate=8.78764)
7 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=0) (Gaslift Gas Injection Rate=13.5721)
8 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=0) (Gaslift Gas Injection Rate=20.9614)
9 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=0) (Gaslift Gas Injection Rate=32.3739)
10 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=0) (Gaslift Gas Injection Rate=50)
11 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=10.5556) (Gaslift Gas Injection Rate=1)
12 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=10.5556) (Gaslift Gas Injection Rate=1.54445)
13 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=10.5556) (Gaslift Gas Injection Rate=2.38533)
14 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=10.5556) (Gaslift Gas Injection Rate=3.68403)
15 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=10.5556) (Gaslift Gas Injection Rate=5.68981)
16 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=10.5556) (Gaslift Gas Injection Rate=8.78764)
17 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=10.5556) (Gaslift Gas Injection Rate=13.5721)
18 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=10.5556) (Gaslift Gas Injection Rate=20.9614)
19 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=10.5556) (Gaslift Gas Injection Rate=32.3739)
20 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=10.5556) (Gaslift Gas Injection Rate=50)
21 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=21.1111) (Gaslift Gas Injection Rate=1)
22 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=21.1111) (Gaslift Gas Injection Rate=1.54445)
23 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=21.1111) (Gaslift Gas Injection Rate=2.38533)
24 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=21.1111) (Gaslift Gas Injection Rate=3.68403)
25 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=21.1111) (Gaslift Gas Injection Rate=5.68981)
26 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=21.1111) (Gaslift Gas Injection Rate=8.78764)
27 - (Boundary Pressure=200) (Gas Oil Ratio=100) (Water Cut=21.1111) (Gaslift Gas Injection Rate=13.5721)

Point	Liquid Rate	Oil Rate	Water Rate	Gas Rate	VLP Pressure	Wellhead Presure	First Node Pressure	Wellhead Temperature	First Node Temperature	Last Node Temperature	dP Friction - Production String
	(STB/day)	(STB/day)	(STB/day)	(MMscf/day)	(psig)	(psig)	(psig)	(deg F)	(deg F)	(deg F)	(psi)
1	400.0	357.8	42.2	0.187	2356.2	200.0	200.0	123.2	123.2	210.0	1218.72
2	617.8	552.6	65.2	0.289	2376.0	200.0	200.0	125.6	125.6	210.0	1238.27
3	954.1	853.4	100.7	0.446	2409.4	200.0	200.0	129.0	129.0	210.0	1264.22
4	1473.6	1318.1	155.5	0.688	2464.1	200.0	200.0	133.7	133.7	210.0	1293.23
5	2275.9	2035.7	240.2	1.063	2507.9	200.0	200.0	139.5	139.5	210.0	1194.83
6	3515.1	3144.0	371.0	1.642	2565.7	200.0	200.0	146.6	146.6	210.0	1134.82
7	5428.8	4855.8	573.0	2.536	2710.5	200.0	200.0	154.9	154.9	210.0	1164.15
8	8384.6	7499.5	885.0	3.916	2997.6	200.0	200.0	163.9	163.9	210.0	1300.99
9	12949.6	11582.7	1366.9	6.049	3461.4	200.0	200.0	173.1	173.1	210.0	1567.41
10	20000.0	17888.9	2111.1	9.342	4463.6	200.0	200.0	181.8	181.8	210.0	2333.65

Case 220 (Boundary Pressure=200) (Gas Oil Ratio=522.222) (Water Cut=10.5556) (Gaslift Gas Injection Rate=50)

Figure 3.6: VLP Screen after Entering the Sensitivity Cases and Liquid Rate

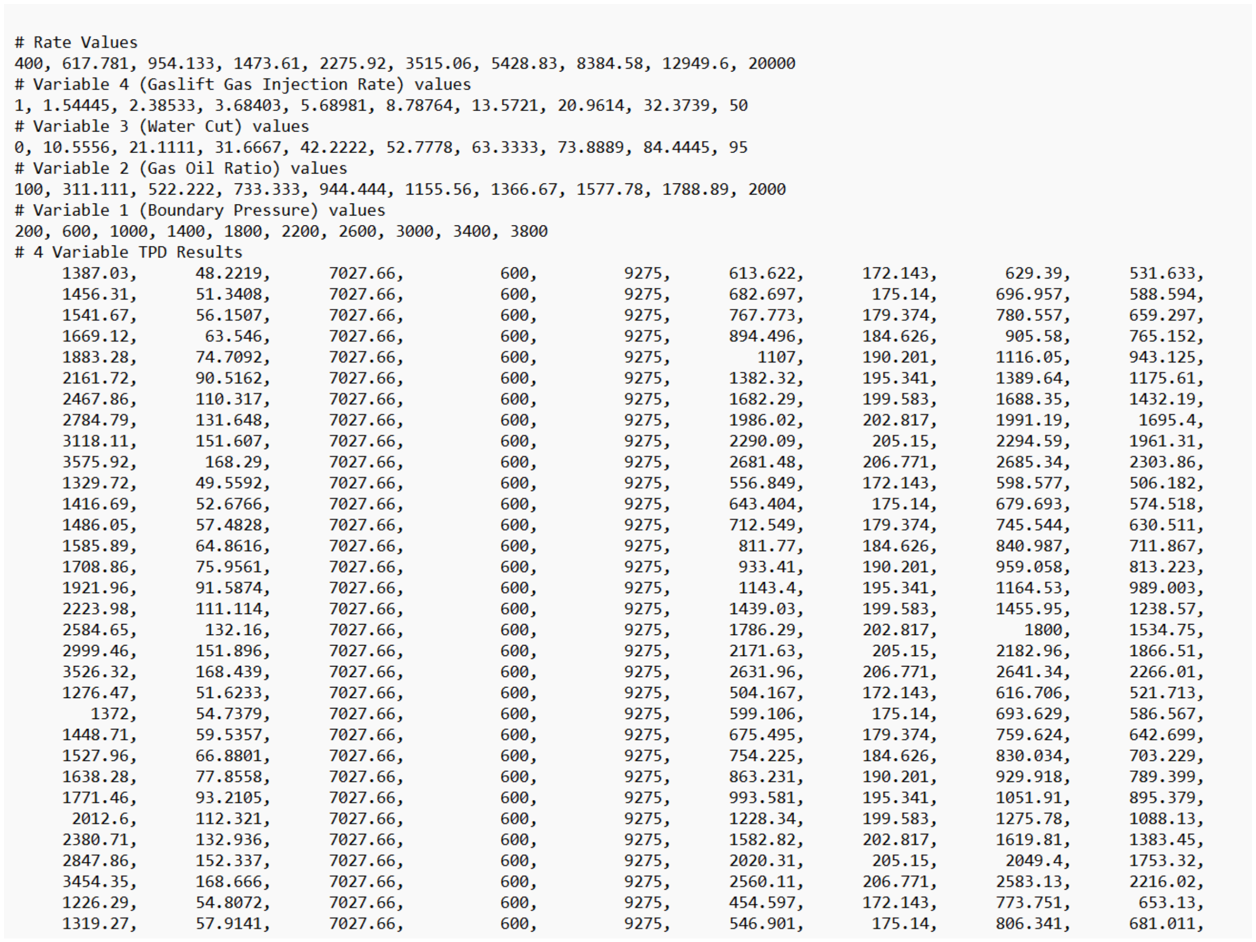
In Figure 3.7, the generated well model is represented by a TPD file containing four sensitivity variables and 21 calculated values (columns) for each well. The VLP curves reported a set of calculated variables. The primary focus of this thesis was on calculating bottom hole pressure for various rates based on a specific set of conditions, including various WHP, GOR, water cut, and gas lift injection rates.



```
# Number of Sensitivity Variables
4
# Numbers of :- Rates, Gaslift Gas Injection Rate values, Water Cut values, Gas Oil Ratio values, Boundary Pressure values
10, 10, 10, 10, 10
# Number of Calculated Values (columns)
21
# 5000 - Flowing Bottom Hole Pressure
# 5001 - Flowing Wellhead Temperature
# 5108 - GASLIFT - Injection Depth
# 5142 - GASLIFT - Top Node Depth
# 5143 - GASLIFT - Bottom Node Depth
# 5144 - GASLIFT - Valve Tubing Pressure
# 5145 - GASLIFT - Valve Tubing Temperature
# 5146 - GASLIFT - Valve Casing Pressure
# 5147 - GASLIFT - Casing Head Pressure
# 5148 - GASLIFT - Gas Injection Rate
# 5149 - GASLIFT - Critical Gas Injection Rate
# 5150 - GASLIFT - Critical Casing Head Pressure
# 5151 - GASLIFT - Orifice Diameter
# 5152 - GASLIFT - Thornhill-Craver DeRating Value
# 5153 - GASLIFT - GasLift Gas Gravity
# 5022 - C Factor
# 5100 - Mixture Velocity
# 5101 - Erosional Velocity
# 5102 - Maximum Grain Diameter
# 5104 - Erosion Flag
# 5157 - Cumulative Transit Time
5000, 5001, 5108, 5142, 5143, 5144, 5145, 5146, 5147, 5148, 5149, 5150, 5151, 5152, 5153, 5022, 5100, 5101, 5102, 5104, 5157
# Rate and Variable Types
# Rate Variable =  4000 - Liquid Rate
# Variable 4 =  22 - Gaslift Gas Injection Rate
# Variable 3 =  16 - Water Cut
# Variable 2 =  17 - Gas Oil Ratio
# Variable 1 =  27 - Boundary Pressure
4000,22,16,17,27
```

Figure 3.7: The Initial Part of the TPD File of Well-1 from PROSPER, the Names of the Calculated Values are Clearly Labeled, among Other Important Details.

The variables that had an impact on calculated values are displayed in Figure 3.8, starting with Rate Values (Liquid Rate) and ending with Variable 1 (Boundary Pressure). It should be noted that despite the presence of 21 columns for calculated variables below the line “# 4 Variable TPD Results”, only the first 9 columns are presented in Figure 3.8 owing to space constraints. The Flowing Bottom Hole Pressure (BHP) is the single most significant column in this master's thesis. As there were 10 × 10 × 10 × 10 × 10 variables, this process generated 10,000 rows of values for the calculated variables.



```
# Rate Values
400, 617.781, 954.133, 1473.61, 2275.92, 3515.06, 5428.83, 8384.58, 12949.6, 20000
# Variable 4 (Gaslift Gas Injection Rate) values
1, 1.54445, 2.38533, 3.68403, 5.68981, 8.78764, 13.5721, 20.9614, 32.3739, 50
# Variable 3 (Water Cut) values
0, 10.5556, 21.1111, 31.6667, 42.2222, 52.7778, 63.3333, 73.8889, 84.4445, 95
# Variable 2 (Gas Oil Ratio) values
100, 311.111, 522.222, 733.333, 944.444, 1155.56, 1366.67, 1577.78, 1788.89, 2000
# Variable 1 (Boundary Pressure) values
200, 600, 1000, 1400, 1800, 2200, 2600, 3000, 3400, 3800
# 4 Variable TPD Results
     1387.03,      48.2219,      7027.66,          600,         9275,      613.622,      172.143,       629.39,      531.633,
     1456.31,      51.3408,      7027.66,          600,         9275,      682.697,       175.14,      696.957,      588.594,
     1541.67,      56.1507,      7027.66,          600,         9275,      767.773,      179.374,      780.557,      659.297,
     1669.12,       63.546,      7027.66,          600,         9275,      894.496,      184.626,       905.58,      765.152,
     1883.28,      74.7092,      7027.66,          600,         9275,         1107,      190.201,      1116.05,      943.125,
     2161.72,      90.5162,      7027.66,          600,         9275,      1382.32,      195.341,      1389.64,      1175.61,
     2467.86,      110.317,      7027.66,          600,         9275,      1682.29,      199.583,      1688.35,      1432.19,
     2784.79,      131.648,      7027.66,          600,         9275,      1986.02,      202.817,      1991.19,       1695.4,
     3118.11,      151.607,      7027.66,          600,         9275,      2290.09,       205.15,      2294.59,      1961.31,
     3575.92,       168.29,      7027.66,          600,         9275,      2681.48,      206.771,      2685.34,      2303.86,
     1329.72,      49.5592,      7027.66,          600,         9275,      556.849,      172.143,      598.577,      506.182,
     1416.69,      52.6766,      7027.66,          600,         9275,      643.404,       175.14,      679.693,      574.518,
     1486.05,      57.4828,      7027.66,          600,         9275,      712.549,      179.374,      745.544,      630.511,
     1585.89,      64.8616,      7027.66,          600,         9275,       811.77,      184.626,      840.987,      711.867,
     1708.86,      75.9561,      7027.66,          600,         9275,       933.41,      190.201,      959.058,      813.223,
     1921.96,      91.5874,      7027.66,          600,         9275,       1143.4,      195.341,      1164.53,      989.003,
     2223.98,      111.114,      7027.66,          600,         9275,      1439.03,      199.583,      1455.95,      1238.57,
     2584.65,       132.16,      7027.66,          600,         9275,      1786.29,      202.817,         1800,      1534.75,
     2999.46,      151.896,      7027.66,          600,         9275,      2171.63,       205.15,      2182.96,      1866.51,
     3526.32,      168.439,      7027.66,          600,         9275,      2631.96,      206.771,      2641.34,      2266.01,
     1276.47,      51.6233,      7027.66,          600,         9275,      504.167,      172.143,      616.706,      521.713,
        1372,      54.7379,      7027.66,          600,         9275,      599.106,       175.14,      693.629,      586.567,
     1448.71,      59.5357,      7027.66,          600,         9275,      675.495,      179.374,      759.624,      642.699,
     1527.96,      66.8801,      7027.66,          600,         9275,      754.225,      184.626,      830.034,      703.229,
     1638.28,      77.8558,      7027.66,          600,         9275,      863.231,      190.201,      929.918,      789.399,
     1771.46,      93.2105,      7027.66,          600,         9275,      993.581,      195.341,      1051.91,      895.379,
      2012.6,      112.321,      7027.66,          600,         9275,      1228.34,      199.583,      1275.78,      1088.13,
     2380.71,      132.936,      7027.66,          600,         9275,      1582.82,      202.817,      1619.81,      1383.45,
     2847.86,      152.337,      7027.66,          600,         9275,      2020.31,       205.15,       2049.4,      1753.32,
     3454.35,      168.666,      7027.66,          600,         9275,      2560.11,      206.771,      2583.13,      2216.02,
     1226.29,      54.8072,      7027.66,          600,         9275,      454.597,      172.143,      773.751,       653.13,
     1319.27,      57.9141,      7027.66,          600,         9275,      546.901,       175.14,      806.341,      681.011,
```

Figure 3.8: The Second Part of the TPD File of Well1 from PROSPER Illustrates the Name and Values of the Selected Variables, as well as the Values of the Calculated Variables.

All the previously mentioned steps were implemented for all six wells to generate the models. The six well models within the network showcased an identical range of four free variables and corresponding rate values. Nevertheless, the divergences between them were due to the calculated values, specifically the Bottom Hole Pressure.

3.3 Read Data to Python

This section begins by covering some important theoretical concepts, which include diverse data structures, Pandas, and the structure of JSON. This is done before demonstrating how to read data into Python.

3.3.1 NumPy Array, List, Tuple, and Dictionaries

Python employs data structures that are vital to its programming framework to arrange and store various items. Knowledge of diverse data structures and their distinctions is of the utmost priority. NumPy arrays, lists, tuples, and dictionaries are the most applicable data types for this project.

Scientific computation in Python heavily relies on NumPy arrays. The NumPy array is superior to the Core Python array in terms of flexibility, efficiency, and benefits. Before utilizing the array module, it must be imported. NumPy arrays are used to store numerical data lists and to depict vectors, matrices, and tensors. An array is a mutable sequence of objects that have the same type. The term mutable implies that an array can be modified by adding or removing elements, updating existing elements, and so forth. A sequence is an indication that the elements are arranged in a particular order [44].

A list is a built-in data type that facilitates the storage of multiple values or elements in a single variable. These are sequences of mutable objects enclosed by square brackets []. The capacity of a list to store values for multiple categories is one of its defining characteristics. It is feasible to store string and integer values together in a single list [43].

The Tuple data type is another built-in feature of Python that is used to store collections of data and objects separated by commas. A tuple in Python has certain similarities to a Python list, such as indexing, nested objects, and repetition. Nonetheless, the key distinction between them is that a Python tuple is immutable, whereas a Python list is mutable.

Besides the tuple, another powerful built-in data type in Python is the dictionary. Using dictionaries to store information tables with a unique identifier for each record is beneficial. Dictionary entries provide an association between a set of keys and a set of values. A combination of keys and values is commonly referred to as an "item" [45]. A key represents the name of the value, and the value is any Python object that must be stored, including lists, functions, integers, etc. Curly parentheses {} enclose a dictionary, which is an unordered collection of key-value pairs. Unordered objects cannot be indexed because they cannot be sorted.

3.3.2 Pandas Package

Pandas is a Python library that is open-source and offers a variety of rich data structures and tools that are commonly used in areas such as finance, statistics, and other related disciplines for working with structured data sets. Pandas supplies numerous tools for data processing and manipulation. Pandas was created to address the need for a comprehensive data analysis library that provides all the essential tools for data extraction, processing, and manipulation in the most straightforward manner possible.

Based on the NumPy library, this Python package was developed. The success and rapid propagation of pandas were fundamentally dependent on this development. By doing so, this decision not only makes this library compatible with most other modules but also uses the high quality of the NumPy module.

The fundamental aspect of Pandas comprises two primary data structures, $Series$ and $DataFrames$, which act as the central point for all transactions involved in the analysis of data.

The $Series$ is an object within the Pandas library that is intended to represent one-dimensional data structures, which are similar to arrays but possess additional features. Comprising two arrays that are associated with each other, the internal structure is straightforward. The primary array contains data (of any NumPy type) that corresponds to a label, which is stored in a separate array known as the index. To produce the series, the $Series()$ constructor must be called, and an array containing the values to be included should be passed as an argument, as shown in Equation 3.1.

$$Frame = pandas.Series([data],\ \ index) \tag{3.1}$$

The $DataFrame$ is a tabular data structure that closely resembles a spreadsheet. The purpose of this data structure is to expand a series into multiple dimensions. The $DataFrame$ is constructed of a series of ordered columns, Equation 3.2, each having the ability to hold a value of a particular type, including numeric, string, and Boolean [46].

$$Frame = \ pandas.DataFrame(data, index, columns) \tag{3.2}$$

3.3.3 OS Module

Python employs the OS module for filesystem interaction when working with files. The Python OS module provides capabilities for interacting with the operating system. Standard Python utility modules include the operating system. This module enables the use of functionality that is dependent on the operating system and transportable across systems. Numerous functions for interacting with the file system are provided by the modules *os* and *os.path*.

The OS package in Python has an essential feature in the form of the $os.listdir()$ function. The $os.listdir()$ method in Python is used to get a list of all files and directories in the specified directory, folder, or path (Equation 3.3). In the event that a directory is not specified, the current working directory will generate a list of files and directories.

$$os.listdir(pathSpecified) \tag{3.3}$$

3.3.4 JavaScript Object Notation Structure

JavaScript Object Notation (JSON) is a format for exchanging data that is both compact and efficient. JSON is simple for computers to process and generate while also being highly readable and writable by humans. JSON is a text-based format that is not bound to a specific programming language, rendering it language-independent. However, it adheres to conventions that are common among C-based programming languages, such as Java, JavaScript, C++, and Python, among others. This characteristic makes JSON an outstanding option for exchanging data between systems [18].

The JSON architecture comprises two components. The first is a collection of name-value pairs, which can be represented in various computer languages as an object, record, dictionary, keyed list, or associative array. A linear series of values is the second, which can be expressed as an array, vector, list, or sequence in most computer languages.

The structure of JSON closely resembles that of Python dictionaries, with key-value pairs arranged in the format $``key": < value >$, Equation 3.4b. In this format, the key is always a string (Equation 3.4c), while the $value$ can be a variety of data types, including a string, number, Boolean, array, object, or null (Equation 3.4d). Python is capable of handling multiple JSON data types. The first type is an object, which can be represented as an unsorted Python dictionary or as a pair of curly braces containing key/value pairs. The second type is an array, which is comprised of a list or tuple of elements that are separated by commas and enclosed in square brackets. A JSON document named D, Equation 3.4a, is defined as below [20].

$$D ::= \{Object[, Object, \dots]\} \tag{3.4a}$$

where

$$Object ::= \{Key: \{Value\}[\, , Key: \{Value\}, \dots]\} \tag{3.4b}$$

$$Key ::= String \tag{3.4c}$$

$$Values ::= String|Integer|Number|Array|Null|Object \tag{3.4d}$$

3.3.5 Implementation of Read Model

As Python is the favored programming language for resolving the optimization problem in this thesis, it was necessary to read and extract the data from the well-models into Python. To access and store data from PROSPER well models, TPD files, into a suitable data structure, a series of steps were taken, which resulted in the development of a script named "txt_files_to_json.py" and a module titled "read_module.py". It is suggested that the codes in Appendix B be reviewed concurrently with the implementation explanations in this section up to Section 3.9.

The code utilized NumPy, JSON, and OS libraries, all of which were imported at the beginning. The TPD format files were converted to .txt to enable easy opening and working. After running the code with "txt_files_directory" and "json_files_directory" as inputs, the format of the .txt files was changed to JSON. In JSON, key-value pairs are mapped in a manner similar to Python dictionaries. The "data_dict" variable was formed specifically to store the "free variables" and "tpd results" taken from well model files for every single well. A dictionary was nested within "free variables" consisting of keys such as "Rate values", "QGL", "WC", "GOR", and "Pressure", with blank initial values.

In the for loop, the os. listdir(path) method utilized to generate a list containing all the items in the "txt_files_directory". Following the opening of each file, the readline() method was employed to read through all lines until the index containing #Rate Values was located, as shown in Figure 3.8. A single variable named "free_var" was used to collect all data related to free variables from this index to ten lines later. The list of "tpd results" was collected from the 11th row following the reference index, marked as "# 4 Variable TPD Results" and continued until the end of the file.

The values of free variable saved in JSON format by storing each value separately in a list and using the split() method to remove any spaces between numbers in the .txt file.Only the first column of calculated variables was considered in this work, which is bottom hole pressure. The Numpy array was used to store and set values for "tpd results" in the variable "data_dict".

Then, the directory named "json_files_directory" was opened, and the "data_dict" was saved with all its keys and values. The JSON package possesses a "dump" function that allows for the writing of the dictionary directly to a file in JSON format. To obtain suitable JSON formats, this method was employed to convert Python dictionaries.

The "read_module.py" module contained a single function, "read_json_data", which took the "file_name" argument to specify the directory of the file and name of the file for reading data from JSON files. JSON and Pandas libraries were imported for this code. "free_vars" and "tpd_res" were returned for each well after the JSON files were loaded and read, with the former in dictionary form and the latter in Pandas Series.

3.4 Interpolation and Unit Convert

The data used in this study, as explained earlier, comprises discrete data points involving both free variables and TPD results. Interpolation is a necessary tool when dealing with combinations of free variables. By interpolating the free variable, the resulting bottom hole pressure is obtained, as shown in Figure 3.9.

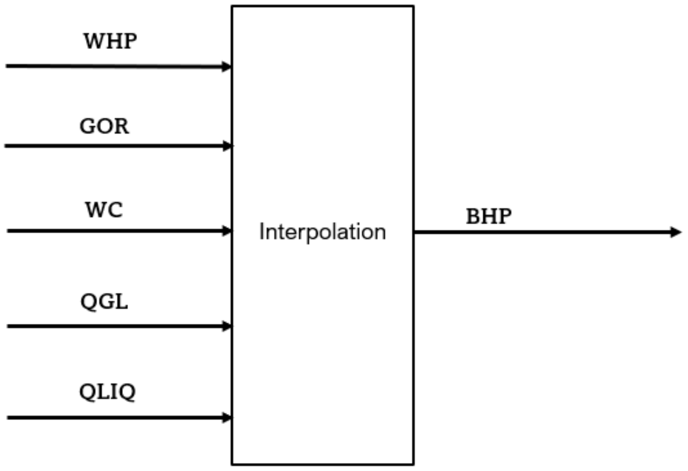


Figure 3.9: Interpolation Process of Five Free Variables and the Resulting BHP

The interpolation method for a case involving five free variables can be mathematically described as follows:

$$\theta = (QLIQ, QGL, WC, GOR, WHP) \tag{3.5a}$$

$$y = g(x, \theta) \tag{3.5b}$$

$$y = BHP \tag{3.5c}$$

The variable θ is defined as the inputs to the interpolation function y.

In TPD files, both free variables and TPD results are generated in field units, while Python calculations are performed in the metric unit. Then, it is needed to convert units from OilField units to Norwegian SI once interpolating data points on the grid. Based on the definition of unit conversion in PROSPER

$$User\ unit\ value = (Base\ unit\ value + Shift) * Multiplier \tag{3.6}$$

Yet to make it easier, α is considered for the Shift and β is replaced with the Multiplier, like below.

$$value = (x + \alpha) * \beta \qquad (3.7)$$

where the values for Shift and Multiplier are different for each free variable as:

- Rate Values: STB/day to Sm^3/day; $\alpha = 0,\ \beta = 0.159$
- Gas Lift Rate: $Mscf/day$ to MSm^3/day; $\alpha = 0,\ \beta = 28.174$
- WC: Percent to Percent; No unit convert.
- GOR: scf/STB to Sm^3/Sm^3 ; $\alpha = 0,\ \beta = 0.1772$
- Boundary Pressure(WHP): $psig$ to $BARa$; $\alpha = 14.696,\ \beta = 0.06894$

In TPD Result the first column only is interested in this master's thesis and its unit is converted as below

- Flowing Bottom Hole Pressure (BHP) : $psig$ to $BARa$; $\alpha = 14.696,\ \beta = 0.06894$

3.4.1 Implementation of Interpolation Process

Within the file named "interpolate_unit_convert.py", the defined class for interpolation is responsible for both the interpolation of the imported well-model data and the conversion of their units. To optimize the data, this step is essential. The methods "convert_points", "get_points", "get_data", and "config_interpolation" are part of the class.

Following an attentive reading of the well models presented earlier, the variables "free_vars" and "tpd_res" were assigned values. At this stage, it is necessary to alter their units and configure their shapes in order to perform interpolation. Two libraries, SciPy and NumPy, were imported by the code. The function "convert_points" was regarded as a technique to modify the units of free variables by incorporating distinct values specified by α and β for every free variable via addition and multiplication, as elaborated in Section 3.4. The variable "res_2" defined the converted free variable lists that were subsequently appended to the "list_converted" variable. NumPy arrays speed up calculations, so the list was converted into an array in the next step.

To convert units of free variables, the "convert_point" method was employed in conjunction with the "get_points" method. The free variable data was acquired by calling their keys in the "data_dict" dictionary and indicating the α and β values for each one to accomplish this. A one-dimensional array was created using the $\mathrm{flatten()}$ operation on a NumPy array. Arranging free variables in the correct order is crucial for interpolation. The tuple of free variables was created by "pnts" in the correct order, which comprised pressure, gas-oil ratio (GOR), water cut (WC), gaslift injection rate (QGL), and liquid rate (QLIQ).

A method called "get_data" was created to convert units and reshape the TPD results. To simplify unit conversion and organize the converted values, the Pandas Series was transformed into a NumPy array, which was then stored in the variable "tpd_res_unit_converted". By using the np.reshape() function, an array was reshaped without any alterations to its data. The reshaping process was carried out by taking into account the length, number of items, and converted values of the free variables, resulting in the formation of five dimensions that were then stored in the variable "reshaped_tpd_res".

"config_interpolation" was the last method in this class, employing "get_points" and "get_data" as free variables and TPD results. Then, interpolation was carried out using a new shape and cubic method through the RegularGridInterpolator. To evaluate alternative interpolation methods, such as linear and pchip, a single file named "config" was employed.

A RegularGridInterpolator callable object was obtained by creating "interpolate_obj" from an interpolation class and calling the "config_interpolation" method. BHP can be obtained by calling "self.interpolate" while taking the free variables as inputs. The CSV file named "Data_for_Interpolation.csv" contains ten random numbers each for WHP, GOR, WC, QGL, and QLIQ, which serve as test points for the interpolation object. The result will be presented in Chapter 4. The read_csv() function from Pandas must be used to obtain data in the form of a DataFrame from the CSV file.

3.5 Well Production-Implementation

In the file "well_production.py", a function was created and named "well_production" which enables the computation of the equilibrium flow rate(Q), q_o , q_w, and q_g per well through the use of specified inputs. The objective of this function was to calculate the equilibrium rate for each well, or operating point, since it is necessary in determining the rates of oil, water, and gas. "Interpolate_obj", "fixed_free_vars", "q_max", and "well_number" were the inputs that the function employed.

Achieving the equilibrium rate was possible by utilizing the "difference" nested function, which effectively reduced the discrepancy between IPR and VLP pressures, resulting in the operational or intersection point, where both pressures are equal. The calculation of both pressures was executed, and the objective was to minimize the differences between them. "BHP_VLP" defined the pressure of VLP, and its computation was based on the callable attributes "self.interpolate" and "interpolate_obj" using free variables from well model data. The free variables were defined in a CSV file named "Data_for_well_production.csv". The variables WHP, GOR, QGL, and WC were constant for each well. In order to determine the optimal value, only the QLIQ variable was subject to change and was replaced with its previous value using the insert() method.

Furthermore, the IPR pressure was determined using "BHP_IPR" by applying one of the two equations, Vogel or Composition, as explained in Section 2.1. By specifying the Productivity index (J), reservoir pressure (P_R), and bubble point pressure (P_B) in the "config" file, these equations were customized for each well.

Then, by utilizing the "minimize" function from SciPy.optimize and employing one solver, the point of minimal difference between BHP_VLP and BHP_IPR could be found. This point contains the liquid rate (Q) and pressure (BHP), and then the rates for each well can be computed by the below equations.

$$q_w = Q * \frac{WC}{100} \tag{3.8}$$

$$q_o = Q - q_w \tag{3.9}$$

$$q_g = \frac{(\, q_o * GOR + QGL)}{1000} \tag{3.10}$$

The equilibrium rate in each well is denoted as Q, whereas the gas-oil ratio is represented by GOR. QGL represents gas lift injection, q_w represents water rate, q_o represents oil rate, and q_g represents gas rate. For each well, the plot displays a visual representation of the intersection point of VLP and IPR, as well as BHP, with further clarification provided in Section 3.8.

3.6 Implementation of Optimization Problem

The script for field optimization was composed in the file named “field_optimization.py”. The import of Numpy and Scipy libraries, along with the “well_production” function outlined in Section 3.5, was required. This code seeks to discover the QGL that generates the highest possible amount of oil production per well.

The optimization of oil rate (q_o), water rate (q_w), gas rate (q_g), gas lift injection rate (QGL), and liquid rate (QLIQ) was performed by considering the function “field_optimization” as the primary function, which takes in “interpolate_obj”, “fixed_free_vars”, “qgl_ma”, and “well_number” as inputs. By inputting QGL, the inner function “optimize_qo” was intended to determine the QGL with the highest oil production and yield the corresponding qo value. Within the context of this function, the $\text{insert}\,()$ method placed new QGL values into the “fixed_free_vars” list at a specified index. Simultaneously, the $\text{delete}\,()$ method removed previous items from the same list. In order to compute the liquid rates (QLIQ) with a new QGL each time, the function “well_production” was employed, and the outcomes were recorded on a list. Equation 3.11 can be employed to determine q_o, with the first step being to extract the water cut (WC) from the free parameters of every well.

$$q_o = Qliq * (1 - \frac{WC}{100}) \tag{3.11}$$

To optimize the function “optimize_qo”, the “minimize” function was used, which demanded the specification of the initial point, QGL bound, and a chosen method like “Nelder_Mead” that can be modified via “config.py”. Once the optimal value for q_o is determined, Equations 3.8 and 3.10 can be utilized to calculate q_w and q_g. Section 3.8 will provide a lifting curve plot for each well.

3.7 Main and Config Files

The “main.py” script was written to be executed directly and produced the results of interpolation test points, “well_production.py”, and “field_optimization.py” files. In order to test the interpolation code, the file “Data_for_Interpolation.csv” was read using Pandas to create the DataFrame structure. Afterwards, the for loop iterated over the JSON files and procured the essential values for each well, particularly “free_vars” and “tpd_res”, via implementation of the “read_json_data” function. In order to construct the interpolation object, these values were required, and “config_interpolation” from the interpolation class was also called. The BHP values were stored in the “list_of_BHPs” variable and printed.

In order to run the “well_production.py” file, a series of actions were performed within the “main.py” script. In the initial stage, the values of the free variable were gained for each well from “Data_for_well_production” using the $\text{read.csv}()$ method and put together into a list. The "well_production" function utilized inputs such as "interpolate_obj", arrays of "fixed_free_vars", and "Q_MAX[i]", which is a list of the maximum liquid rates of each well defined in "config.py", and the well number determined by i.

The "field_optimization" function underwent a similar process, taking inputs such as "interpolate_obj", an array of "fixed_free_vars", "qgl_max", and i-defined well numbers, leading to the generation of q_o, q_w, q_g, QGL, $q_{o-field}$, $q_{w-field}$, $q_{g-field}$, and QGL_{field}.

In the "config" file, there were certain items that might require updating in the future. Like file paths, and some of the constraints that were defined included the productivity index (J), reservoir pressure (P_R), and bubble point (P_B),. Additionally, this file allows for experimentation and testing of different methods for interpolation, well production, and field optimization.

3.8 Plotting

The results can be easily monitored through two separate codes in "well_production.py" and "field_optimization.py" files, which were built specifically for plotting. Initially, the function "plot_ipr_vlp" was examined to create plots that exhibit intersection points between VLP and IPR. These points included liquid rate (QLIQ) and flowing bottom hole pressure (BHP). "Interpolate_obj", "fixed_free_vars", and i as a well number were among the inputs of the function. One array called "Q_list" was defined to have a range of liquid rates to make the "x" axis in plots. Following this, pressure points for VLP plots along the "y" axis could be generated using the interpolation object. Besides, the pressure points for IPR curves were calculated from equations by Vogel or Composite. Each well is represented by a plot to display the intersection point in this research.

Within the "field_optimization.py" file, the "plot_qo" function was employed to create the gas lift curve plots. An array called "QGL_LIST" that had a range for gas lift injection could be used to create the "x" axis. QGL implemented a for loop to iterate over the array while modifying the amount to discover the optimal value for each well, as detailed in Section 3.6.1. The "y" axis was created by utilizing QLIQ values derived from the "well_production" function and applying Equation 3.11 to determine the amount of oil production. Prior to running the plot code, it is vital to create a file named "images" in the directory where other Python codes are stored.

3.9 Implementation of Case studies

In this thesis, three sensitivity analyzes were tested and their codes were written in files named "wc_test_case.py", "whp_test_case.py", and "qgl_test_case.py".

The intention behind the "wc_test_case.py" file was to restrict water production in wells and examine its repercussion on production rates. Once the JSON files were read, interpolation objects were created and variables for "fixed_free_vars" were established, a for loop was used to test various WC for wells defined in the "wc_test_cases" dictionary. Well numbers were assigned as keys and test cases as values in the dictionary. The CSV file "Data_for_well_production" was used to replace the new values in the test cases. Additionally, QGLs had unique values for test cases recorded in the "qgl_for_wc" dictionary. With fresh values for WC and QGL, the "well_production" function was triggered, which subsequently yielded Q, q_w, q_o, q_g, and BHP.

In file "whp_test_case.py", a range of WHP values was examined to evaluate the influence of wellhead choke. Overall, this thesis operated with an open choke. Despite the water cut values, the examination of wells in the WHP cases was limited to only two constant values, namely 25 and 47. Each value used the special QGL in code defined in the dictionary "qgl_for_whp". Following the reading of the files and the construction of object interpolation, a for loop was employed to replace the original values in the CSV file, "Data_for_well_production" , with the tested values. Lastly, the function "well_production" was employed to return the values of Q, q_w, q_o, q_g, and BHP.

Limited gas lift injection was tested for each well in the last case by "qgl_test_case.py" file. Within the dictionary "qgl_max_dict", the values intended for testing each well were stored. After reading the JSON files, an interpolation object was created and the CSV files were read. The values for each well were determined by a for loop, and the "field_optimization" function produced the requested results, which included q_o, q_w, q_g, QGL, and QLIQ.

3.10 Structure of the System

The structure of the previously introduced system will be explained in this part of the thesis with the aid of a flowchart diagram. The purpose of this is to provide an overview of the system, making it easier to comprehend the various connections.

3.10.1 Flowchart Diagram

A flowchart is a diagrammatic representation of the sequence of steps and decisions that must be made to perform a process. The shape of the diagram indicates each step in the sequence. The steps are interconnected by lines and arrows showing direction. This permits universal access to the flowchart, enabling a logical progression through the process from start to finish. Flowchart comprises diverse geometric figures such as ovals, squares, diamonds, rectangles, and several other forms that represent symbols. Each step of the process has its own distinct symbol, including start and end points, decision-making points, input/output points, and process points. The optimization problem-solving codes in this thesis are presented in a flowchart diagram shown in figure 3.10.

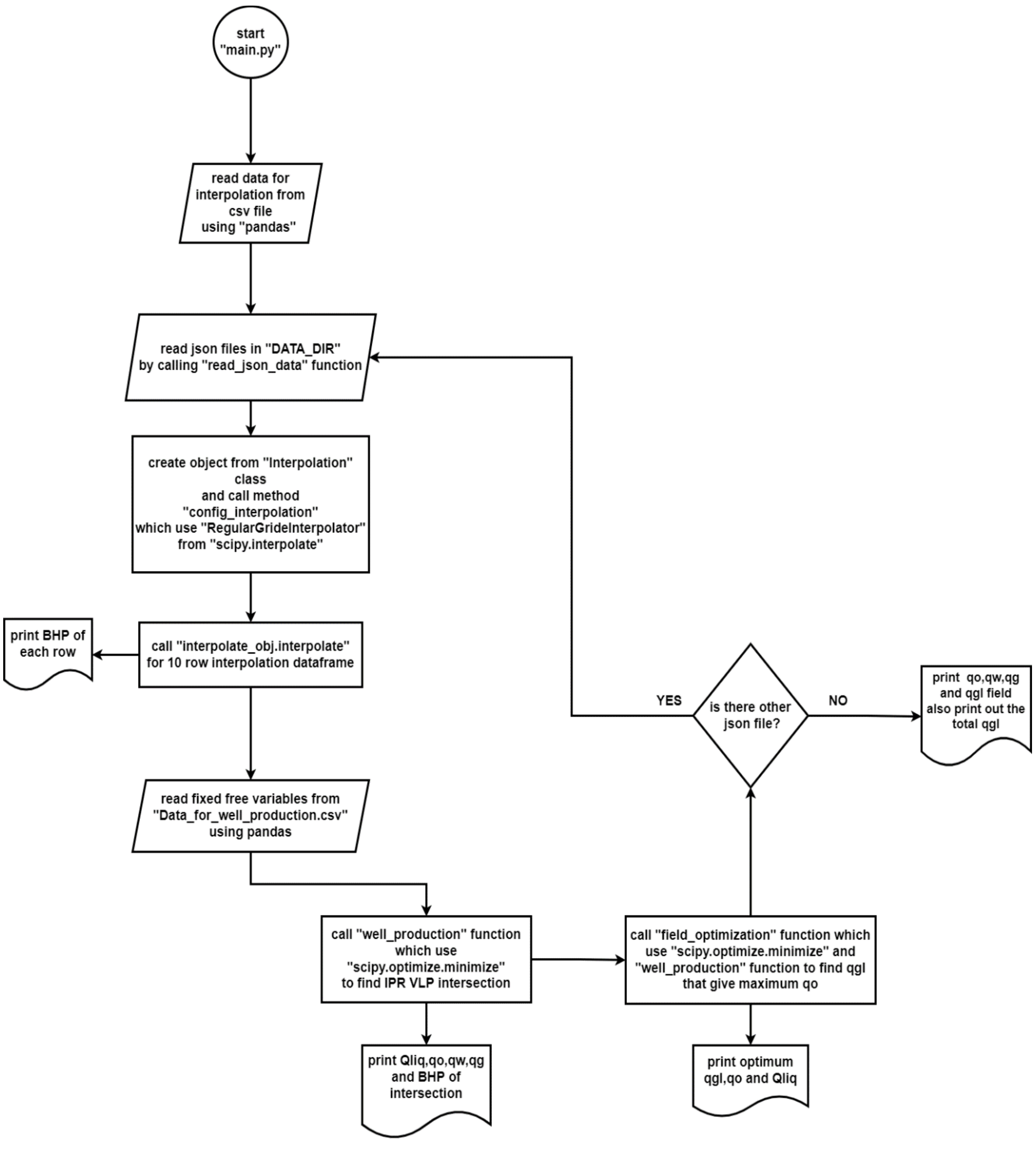

start "main.py"
read data for interpolation from csv file using "pandas"
read json files in "DATA_DIR" by calling "read_json_data" function
create object from "Interpolation" class and call method "config_interpolation" which use "RegularGrideInterpolator" from "scipy.interpolate"
call "interpolate_obj.interpolate" for 10 row interpolation dataframe
print BHP of each row
read fixed free variables from "Data_for_well_production.csv" using pandas
call "well_production" function which use "scipy.optimize.minimize" to find IPR VLP intersection
print Qliq,qo,qw,qg and BHP of intersection
call "field_optimization" function which use "scipy.optimize.minimize" and "well_production" function to find qgl that give maximum qo
print optimum qgl,qo and Qliq
is there other json file?
YES
NO
print qo,qw,qg and qgl field also print out the total qgl

Figure 3.10: Flowchart Diagram of the Optimization Problem Showing the Different Activities

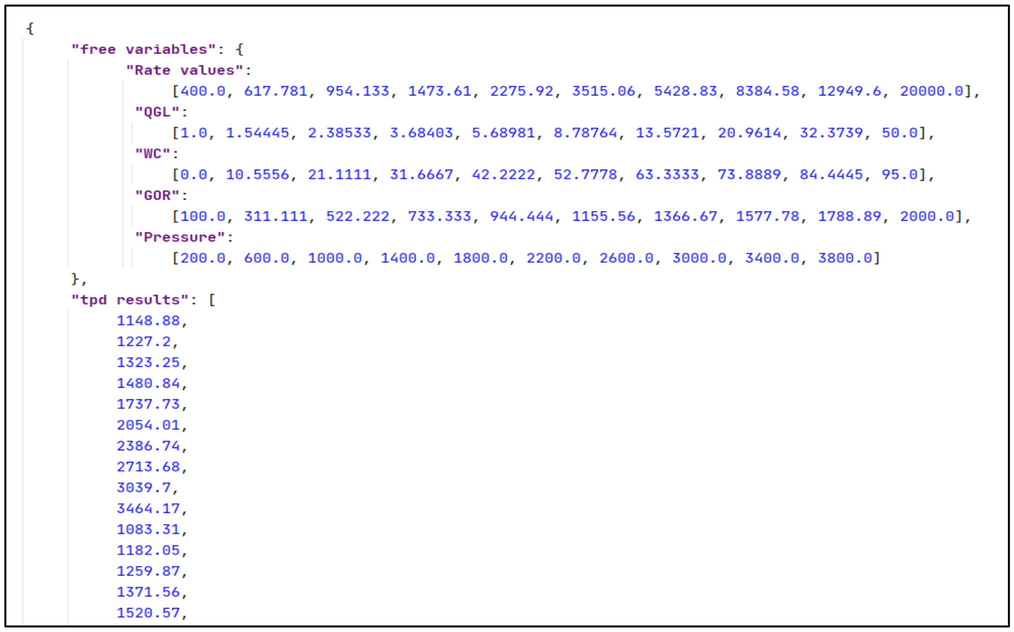
Chapter 4

Results and Discussion

This particular section of the thesis serves to present the outcomes that address the problem formulation stated in Section 1.2 and the research objectives outlined in Section 1.3. First, the findings regarding the import and JSON-based storage of data are exhibited. Afterwards, the findings of the interpolation object for ten testing points are demonstrated in both PROSPER and Python. The next section deals with the production rates of all wells within the network, involving the computation of equilibrium flow rates on plots of IPR and VLP. In the subsequent section, all outcomes concerning gas lift optimization for each well are described and compared using both GAP and Python. The final section of this chapter introduces three sensitivity analyses for the optimization problem in this work, which encompass restricted gas lift injection, constrained water production, and the impact of the wellhead choke.

4.1 Read Data of TPD Files in JSON Format

The process of reading data from well-models into Python is operating as intended. The import of data into JSON format handles many free and calculated variables, resulting in improved data processing efficiency and performance due to its compactness. Within the calculated variable columns, this thesis only considered the first column for analysis. The complexity of working with txt files, as shown in Figure 3.8, highlights the need for transformation into JSON format, which renders the well models easily accessible and straightforward to work with, as illustrated in Figure 4.1. This figure is limited to displaying only specific values from the TPD results column.



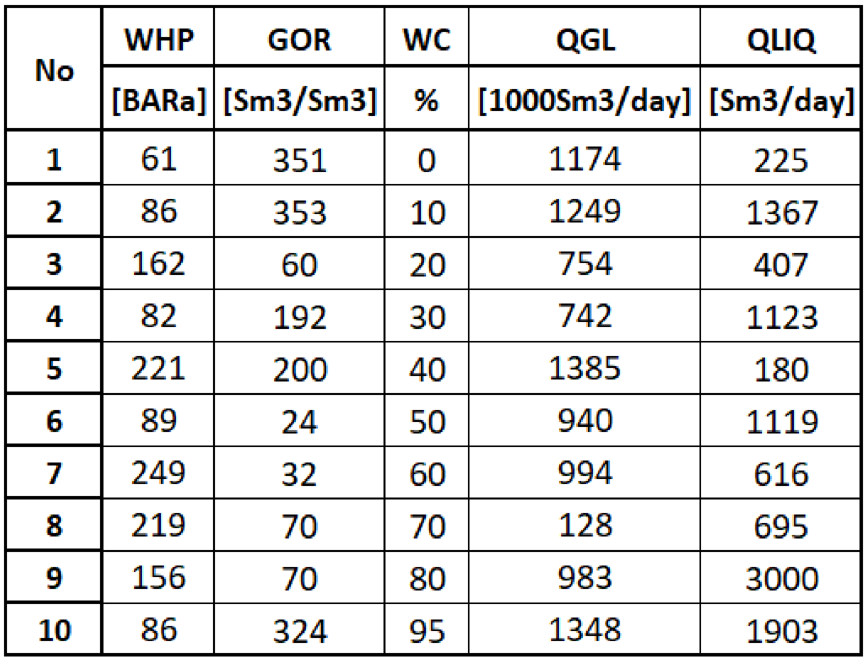
```
{
    "free variables": {
        "Rate values":
            [400.0, 617.781, 954.133, 1473.61, 2275.92, 3515.06, 5428.83, 8384.58, 12949.6, 20000.0],
         "QGL":
            [1.0, 1.54445, 2.38533, 3.68403, 5.68981, 8.78764, 13.5721, 20.9614, 32.3739, 50.0],
         "WC":
            [0.0, 10.5556, 21.1111, 31.6667, 42.2222, 52.7778, 63.3333, 73.8889, 84.4445, 95.0],
         "GOR":
            [100.0, 311.111, 522.222, 733.333, 944.444, 1155.56, 1366.67, 1577.78, 1788.89, 2000.0],
         "Pressure":
            [200.0, 600.0, 1000.0, 1400.0, 1800.0, 2200.0, 2600.0, 3000.0, 3400.0, 3800.0]
    },
    "tpd results": [
        1148.88,
        1227.2,
        1323.25,
        1480.84,
        1737.73,
        2054.01,
        2386.74,
        2713.68,
        3039.7,
        3464.17,
        1083.31,
        1182.05,
        1259.87,
        1371.56,
        1520.57,
```

Figure 4.1: JSON Format for Well_2 Containing Lists of Free Variables and TPD Results

4.2 Interpolation Object Test Points

Discrete data points are provided for imported data, and it is frequently necessary to estimate data points between these discrete data points. Multivariate interpolation is employed as a dependable method of estimation to construct these new data points. The outcome of the interpolation of free variables would yield the flowing bottom hole pressure (BHP). The performance of the interpolation class for data points is satisfactory. Verification is supported by testing with the following points in Table 4.1. For quality control of the interpolation class and object, values are generated randomly within the range of free variables, Table 3.1, besides comparing the results of PROSPER and Python. The interpolation process is an essential requirement for this work, as it serves as the fundamental basis for this project.

Table 4.1: Test Points within the Range of Free Variables



No	WHP	GOR	WC	QGL	QLIQ
	[BARa]	[Sm3/Sm3]	%	[1000Sm3/day]	[Sm3/day]
1	61	351	0	1174	225
2	86	353	10	1249	1367
3	162	60	20	754	407
4	82	192	30	742	1123
5	221	200	40	1385	180
6	89	24	50	940	1119
7	249	32	60	994	616
8	219	70	70	128	695
9	156	70	80	983	3000
10	86	324	95	1348	1903

To facilitate a comparison between the results obtained from Python and PROSPER, it is imperative to employ the relative error as computed by Equation 4.1. The measure of relative error determines how close the measured value is to the real value. The percent error is the percentage value of the relative error.

$$Relative\ Error = \frac{|measured - real|}{real} \tag{4.1}$$

$$Percent\ Error = \frac{|measured - real|}{real} \times 100\% \tag{4.2}$$

The values obtained from PROSPER and GAP are recognized as real, while those from Python are regarded as measured values.

4.2.1 Interpolation with Cubic Method

The results of interpolation for ten random points in each well are presented in Tables 4.2, 4.3, and 4.4 for both PROSPER and Python. To compute the flowing bottom hole pressure (BHP), the first column of calculated variables was used in an interpolation process. In contrast to linear outcomes, it exhibited significantly lower relative errors. It is of significance to mention that the Gradient (Traverse) option in PROSPER can be utilized to interpolate test points, and the output can be retrieved from the VLP Results segment. Each well containing the lowest relative error serves as proof that the interpolation class and object are functioning correctly.

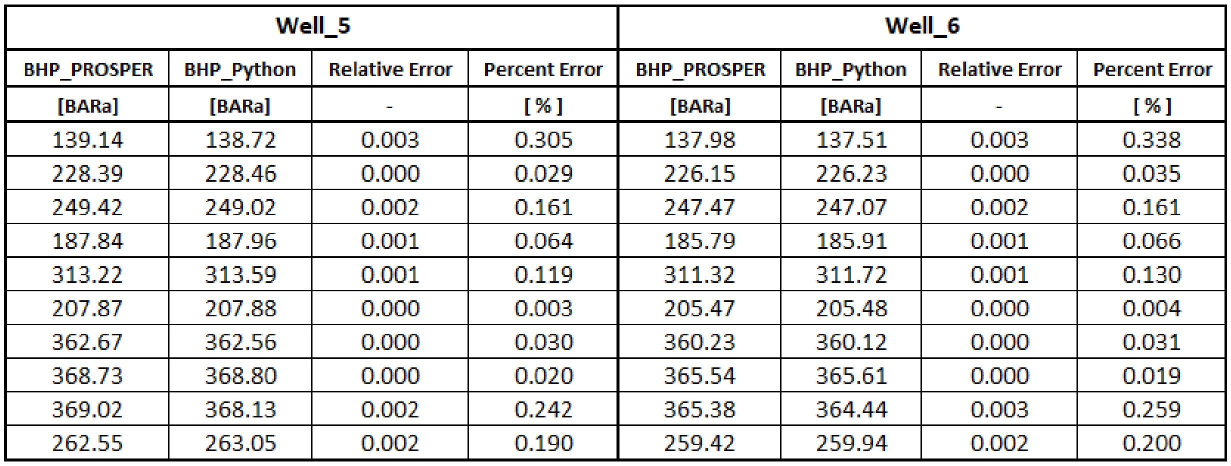
Table 4.2: Interpolation Results for Ten Random Points from PROSPER and Python for Well_1 and Well_2

Well_1				Well_2			
BHP_PROSPER	BHP_Python	Relative Error	Percent Error	BHP_PROSPER	BHP_Python	Relative Error	Percent Error
[BARa]	[BARa]	-	[%]	[BARa]	[BARa]	-	[%]
155.98	154.92	0.007	0.680	147.54	146.66	0.006	0.599
252.87	252.94	0.000	0.028	241.85	241.90	0.000	0.020
285.26	284.95	0.001	0.109	271.22	270.90	0.001	0.119
217.41	217.50	0.000	0.041	205.56	205.70	0.001	0.069
346.00	345.56	0.001	0.127	332.17	331.72	0.001	0.137
243.80	243.83	0.000	0.013	227.18	227.24	0.000	0.027
400.38	400.43	0.000	0.012	385.43	385.52	0.000	0.023
408.58	408.76	0.000	0.044	400.88	401.08	0.000	0.049
409.14	407.49	0.004	0.403	390.59	388.89	0.004	0.436
298.65	299.12	0.002	0.157	279.95	280.42	0.002	0.167

Table 4.3: Interpolation Results for Ten Random Points from PROSPER and Python for Well_3 and Well_4

Well_3				Well_4			
BHP_PROSPER	BHP_Python	Relative Error	Percent Error	BHP_PROSPER	BHP_Python	Relative Error	Percent Error
[BARa]	[BARa]	-	[%]	[BARa]	[BARa]	-	[%]
152.81	151.71	0.007	0.723	140.49	140.05	0.003	0.317
253.07	253.13	0.000	0.024	230.63	230.72	0.000	0.039
261.96	261.65	0.001	0.118	253.28	252.88	0.002	0.160
204.18	204.25	0.000	0.032	190.87	191.00	0.001	0.071
326.22	325.76	0.001	0.140	316.76	317.04	0.001	0.088
223.82	223.86	0.000	0.017	212.15	212.15	0.000	0.001
374.73	374.75	0.000	0.005	367.11	367.00	0.000	0.031
376.3	376.50	0.001	0.052	373.04	373.11	0.000	0.020
401.45	399.87	0.004	0.393	374	372.76	0.003	0.333
285.22	285.67	0.002	0.158	267.1	267.60	0.002	0.188

Table 4.4: Interpolation Results for Ten Random Points from PROSPER and Python for Well_5 and Well_6



Well_5				Well_6			
BHP_PROSPER	BHP_Python	Relative Error	Percent Error	BHP_PROSPER	BHP_Python	Relative Error	Percent Error
[BARa]	[BARa]	-	[%]	[BARa]	[BARa]	-	[%]
139.14	138.72	0.003	0.305	137.98	137.51	0.003	0.338
228.39	228.46	0.000	0.029	226.15	226.23	0.000	0.035
249.42	249.02	0.002	0.161	247.47	247.07	0.002	0.161
187.84	187.96	0.001	0.064	185.79	185.91	0.001	0.066
313.22	313.59	0.001	0.119	311.32	311.72	0.001	0.130
207.87	207.88	0.000	0.003	205.47	205.48	0.000	0.004
362.67	362.56	0.000	0.030	360.23	360.12	0.000	0.031
368.73	368.80	0.000	0.020	365.54	365.61	0.000	0.019
369.02	368.13	0.002	0.242	365.38	364.44	0.003	0.259
262.55	263.05	0.002	0.190	259.42	259.94	0.002	0.200

BHP computation for random points from both Python and PROSPER reveals a deviation below 1%. It appears that the interpolation code is working correctly based on the resulting values.

4.3 Well Production

The objective of the well production function was to determine the point of intersection between the VLP and IPR curves, which is known as the operating point, as it signifies the effectiveness of the well or production system. This function was implemented in Python, and its steps will be explained below. Additionally, the step-by-step process of determining the equilibrium flow rate in GAP will be outlined, followed by a comparison of findings from GAP and Python.

The Run Network Solver in GAP produced results for the equilibrium rate (Q), bottomhole pressure (BHP), oil rate (q_o), water rate (q_w), and gas rate (q_g) for wells using a base case of 281.73 (1000 Sm3/day) for QGL and a separator pressure of 14.80 (BARa). These results were documented in Tables 4.7, 4.8, and 4.9.

Python employed a similar method for computing equilibrium flow rate by utilizing free variables detailed in Table 4.5 to determine the bottomhole pressure for vertical lift performance (VLP). The values in Table 4.5 were what GAP used for the Network Solver.

Table 4.5: Free Variables for Six Wells Collecting from the GAP Model

Wells	Free Variables of Six Wells			
	WHP	GOR	WC	QGL
	[BARa]	[Sm3/Sm3]	%	[1000Sm3/day]
Well_1	14.8	141.7	80	35.649
Well_2	14.8	141.7	40	101.320
Well_3	14.8	141.7	40	46.726
Well_4	14.8	88.6	12	35.358
Well_5	14.8	88.6	12	33.002
Well_6	14.8	88.6	12	29.680

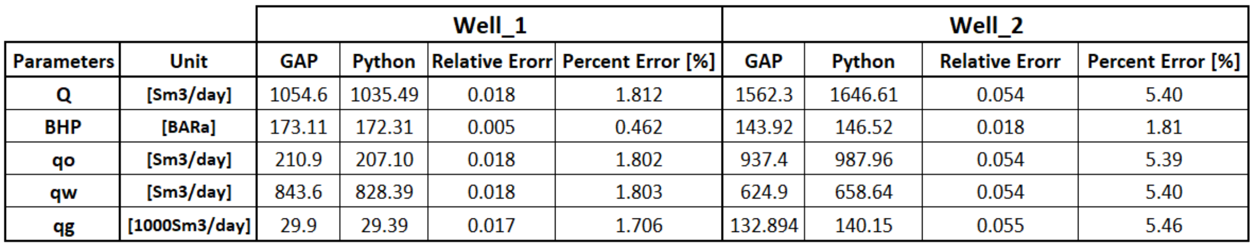
To obtain the bottomhole pressure in the inflow performance curve (IPR), Equations 2.5 and 2.7a were employed with designated inputs for individual wells, as illustrated in Table 4.6. The IPR data for each well collected from GAP is displayed in this table. Reservoir pressure (P_R), bubble point pressure (P_B), productivity index (J), and maximum liquid rate (Q_{max}) comprise these values. A comparison of the reservoir pressure and bubble point pressure can reveal the type of IPR calculation, according to the explanation in Section 2.2.1. Upon minimizing the difference between BHP from IPR and VLP, the equilibrium flow rate was computed, and BHP, q_o, q_w, and q_g were determined for each well.

Table 4.6: IPR Data for All Six Wells in the Network, and Well Status for IPR Calculation

Wells	PR	Pb	J	Qmax-GAP	Well Status	Equation-Name
	[BARa]	[BARa]	[Sm3/day/bar]	[Sm3/day]		
Well_1	238.882	242.33	19.77	2356.41	Pb>PR	Vogel
Well_2	256.119	242.33	19.73	2615.34	Pb<PR	Composite
Well_3	256.119	242.33	19.73	2615.34	Pb<PR	Composite
Well_4	209.649	152.698	3.53	503.27	Pb<PR	Composite
Well_5	209.649	152.698	3.53	503.27	Pb<PR	Composite
Well_6	209.649	152.698	3.53	503.27	Pb<PR	Composite

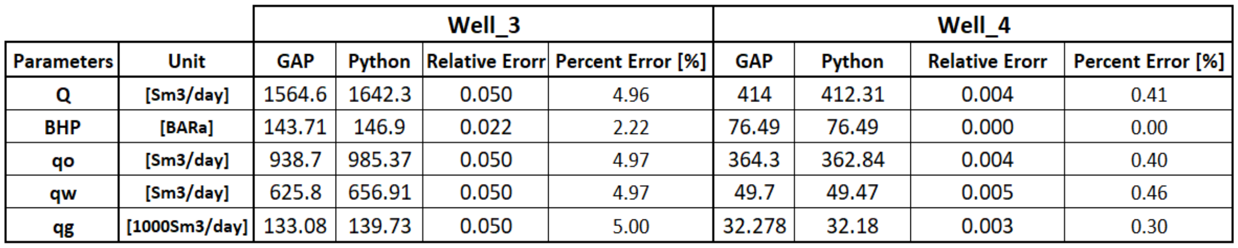
A table was used to bring the results of GAP and Python close together for performance comparison. Well_2 and Well_3 exhibit a percent error of less than 6%, while the other wells demonstrate a deviation lower than 2%.

Table 4.7: Equilibrium Rate, Pressure, Oil Rate, Gas Rate, and Water Rate from GAP and Python For Well_1 and Well_2



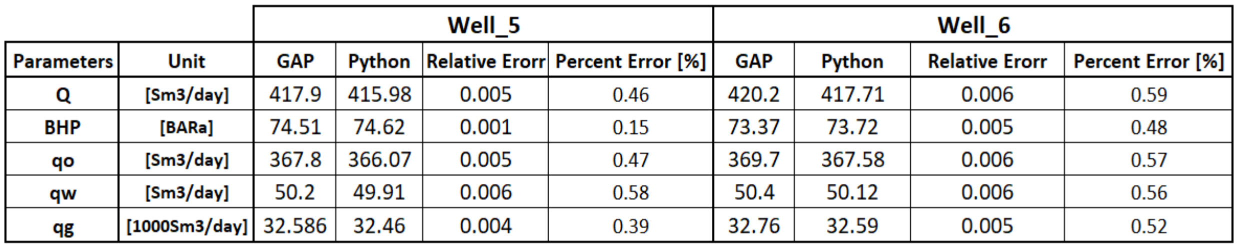
		Well_1				Well_2			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	1054.6	1035.49	0.018	1.812	1562.3	1646.61	0.054	5.40
BHP	[BARa]	173.11	172.31	0.005	0.462	143.92	146.52	0.018	1.81
qo	[Sm3/day]	210.9	207.10	0.018	1.802	937.4	987.96	0.054	5.39
qw	[Sm3/day]	843.6	828.39	0.018	1.803	624.9	658.64	0.054	5.40
qg	[1000Sm3/day]	29.9	29.39	0.017	1.706	132.894	140.15	0.055	5.46

Table 4.8: Equilibrium Rate, Pressure, Oil Rate , Gas Rate, and Water Rate from GAP and Python For Well_3 and Well_4



		Well_3				Well_4			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	1564.6	1642.3	0.050	4.96	414	412.31	0.004	0.41
BHP	[BARa]	143.71	146.9	0.022	2.22	76.49	76.49	0.000	0.00
qo	[Sm3/day]	938.7	985.37	0.050	4.97	364.3	362.84	0.004	0.40
qw	[Sm3/day]	625.8	656.91	0.050	4.97	49.7	49.47	0.005	0.46
qg	[1000Sm3/day]	133.08	139.73	0.050	5.00	32.278	32.18	0.003	0.30

Table 4.9: Equilibrium Rate, Pressure, Oil Rate ,Gas Rate, and Water Rate from GAP and Python For Well_5 and Well_6



		Well_5				Well_6			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	417.9	415.98	0.005	0.46	420.2	417.71	0.006	0.59
BHP	[BARa]	74.51	74.62	0.001	0.15	73.37	73.72	0.005	0.48
qo	[Sm3/day]	367.8	366.07	0.005	0.47	369.7	367.58	0.006	0.57
qw	[Sm3/day]	50.2	49.91	0.006	0.58	50.4	50.12	0.006	0.56
qg	[1000Sm3/day]	32.586	32.46	0.004	0.39	32.76	32.59	0.005	0.52

4.3.1 IPR and VLP Intersection Plots

In every well, the intersection points, or operating points, are the points where the pressure and flow rates are the same for both IPR and VLP curves. VLP and IPR curves can intersect at one or two points or not intersect at all, as a general rule.

Figure 4.2 displays the IPR and VLP curves plotted on the pressure versus liquid rate for each well in Python. The intersection point represents the rate at which each well and the entire production system can potentially produce. Alterations to the system, such as modifications to the choke or pipe size, can affect the intersection point. According to the data presented in Figure 4.2, the largest liquid rate (Q) and highest q_o were obtained from Well_2 within the network, while the lowest q_o derived from Well_4.

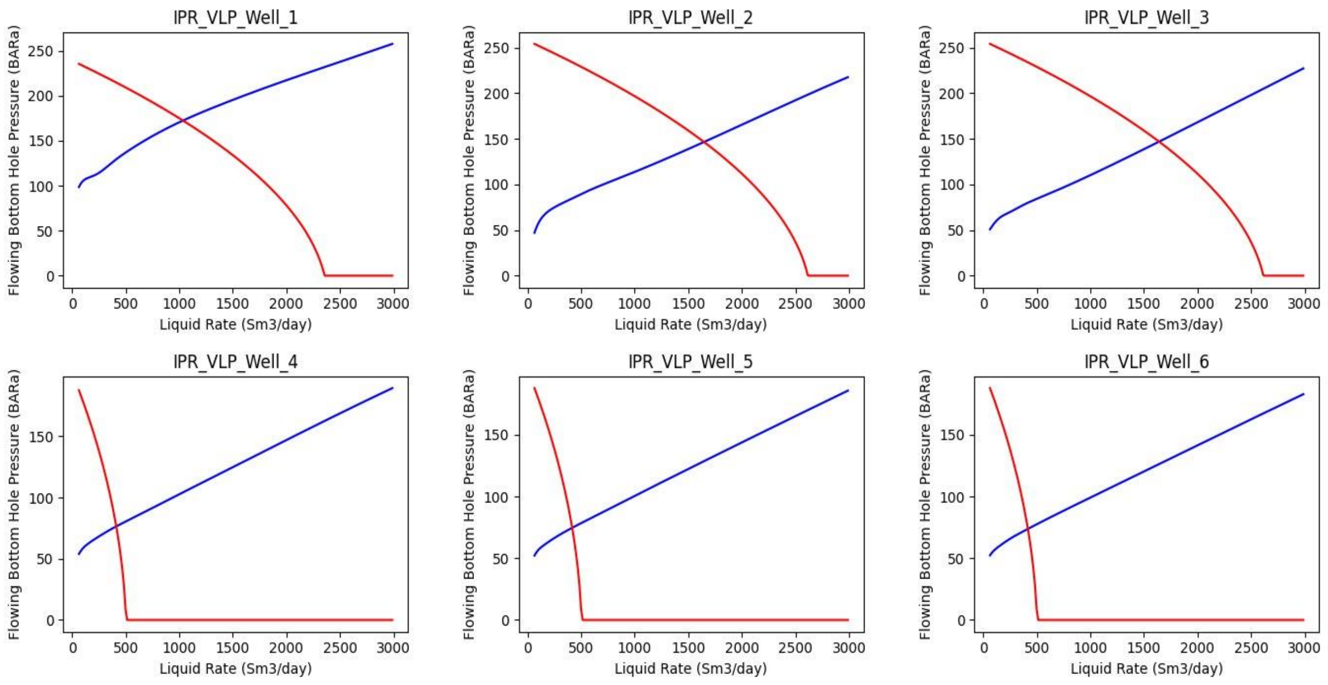


Figure 4.2: Operating Points for Six Wells Defined by the Intersection VLP and IPR Plots in Python

4.4 Field Optimization

The Python function employed in this section had the objective of determining the gas lift rate that resulted in the maximum oil rate production for every well. Except for QGL, all free variables (WC, WHP, and GOR) remained constant in this function, as shown in Table 4.5. Following this, the maximum liquid rate (QLIQ) for wells was identified by the optimal value of QGL. The oil production was then calculated using Equation 3.11, taking into account the water cut in each well. Tables 4.10 and 4.11 reveal that Well_2 and Well_3 have the highest oil production rates, although their QGL is the lowest compared to other wells.

Network Solver was utilized to optimize the field in GAP, and in this particular scenario, the gas lift available for the whole field was taken to be 1408.69 (1000 Sm3/day). Well_3 achieved the highest oil production rate in GAP. Also, despite having the highest gas lift requirement, the oil production of Well_5 and Well_6 in GAP is the lowest.

Table 4.10: Optimum Results QGL, q_o, and $QLIQ$ in Both GAP and Python for Well_1 and Well_2

		Well_1				Well_2			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
QGL	[1000Sm3/day]	181.482	200.78	0.106	10.634	216.854	177.34	0.182	18.22
qo	[Sm3/day]	256.9	253.86	0.012	1.183	945.1	997.98	0.056	5.60
QLIQ	[Sm3/day]	1284.5	1269.31	0.012	1.183	1575.1	1663.3	0.056	5.60

Table 4.11: Optimum Results QGL, q_o, and $QLIQ$ in Both GAP and Python for Well_3 and Well_4

		Well_3				Well_4			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
QGL	[1000Sm3/day]	112.316	124.84	0.112	11.151	186.827	184.68	0.011	1.15
qo	[Sm3/day]	950.8	999.36	0.051	5.107	377.3	374.82	0.007	0.66
QLIQ	[Sm3/day]	1584.7	1665.60	0.051	5.105	428.7	425.941	0.006	0.64

Table 4.12: Optimum Results QGL, q_o, and $QLIQ$ in Both GAP and Python for Well_5 and Well_6

		Well_5				Well_6			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
QGL	[1000Sm3/day]	229.26	185.00	0.193	19.306	229.456	183.75	0.199	19.92
qo	[Sm3/day]	380.8	378.33	0.006	0.649	382.7	380.25	0.006	0.64
QLIQ	[Sm3/day]	432.8	429.90	0.007	0.670	434.9	432.11	0.006	0.64

The performance of GAP and Python exhibits a significant discrepancy when it comes to optimizing and determining the optimal gas lift injection per well. The Well_4 is considered a reliable well with a marginal error rate of 1.15%.

After identifying the maximum oil production rate for each well, the subsequent step involved computing the field oil rate using Equation 4.3. Equations 4.4 and 4.5 were employed for the estimation of field water production and field gas production, respectively.

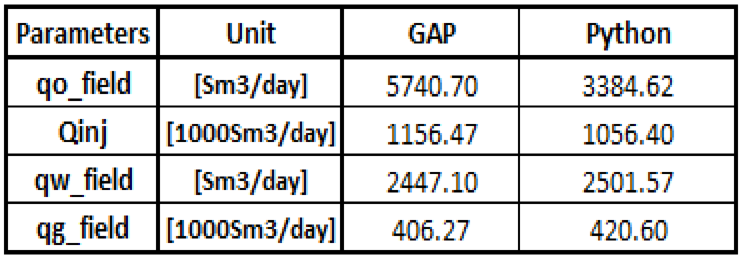
$$q_{o-field} = \sum_{i=1}^{6} q_{oi} \tag{4.3}$$

$$q_{w-field} = \sum_{i=1}^{6} q_{wi} \tag{4.4}$$

$$q_{g-field} = \sum_{i=1}^{6} q_{gi} \tag{4.5}$$

According to Table 4.13, the GAP network did not employ the full gas lift injection available, utilizing only 1156.57 (1000 Sm3/day), while the assigned amount for the field was roughly 1408.69 (1000 Sm3/day). In contrast to Python, the calculated oil rate for the GAP field is higher, while the water and gas rates remain similar. The high percentage errors in comparing GAP and Python performance necessitate the exploration of alternative methods for field optimization.

Table 4.13: GAP and Python Results for Field in Optimization Problem



Parameters	Unit	GAP	Python
qo_field	[Sm3/day]	5740.70	3384.62
Qinj	[1000Sm3/day]	1156.47	1056.40
qw_field	[Sm3/day]	2447.10	2501.57
qg_field	[1000Sm3/day]	406.27	420.60

4.4.1 Gas Lift Performance Curves

The gas lift performance curve can be obtained by plotting the normalized production rate against the normalized gas injection rate for each well. Gas lift performance curves illustrate the correlation between the production efficiency of a gas lift system and its various operational parameters. By utilizing these curves, operators can optimize gas injection rates and other relevant parameters to achieve maximum production output with minimized expenses. Figure 4.3 plots the gas lift performance curves of all six wells generated from Python results for an easy comparison. By searching for discrepancies in the position of the ideal point, the slope of the curves, and any divergences from the normal gas lift curve configuration. This illustration demonstrates that Well_3 has a greater capability to produce oil; in contrast, Well_1 exhibited the least amount.

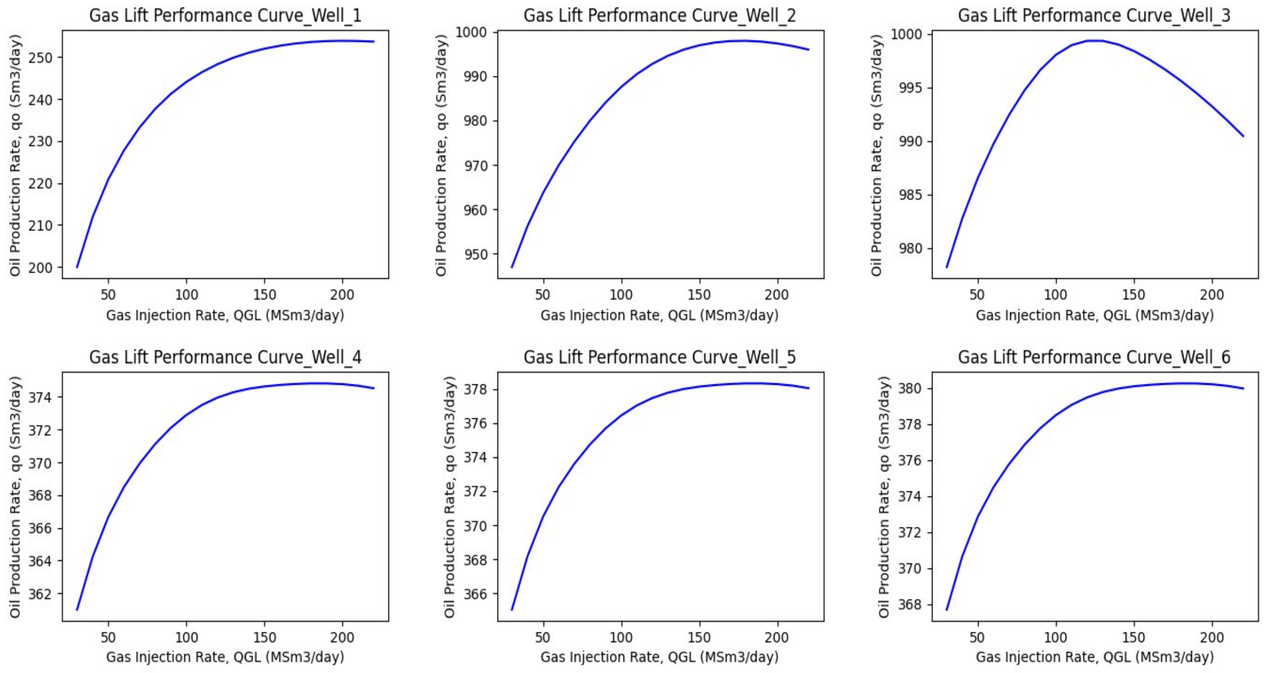


Figure4.3: Gas Lift Performance Curves for Six Wells

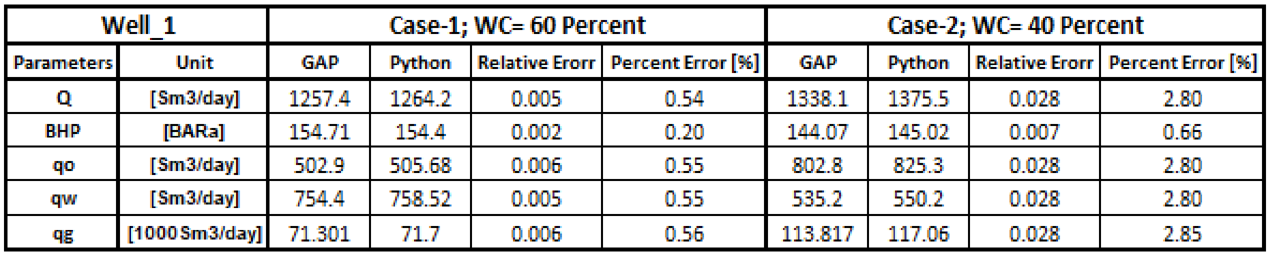
4.5 Sensitivity Analysis

The sensitivity analysis section included three distinct test cases to examine the variables that influence the escalation or reduction of oil rates in wells and to contrast the outcomes from both GAP and Python. Testing involved limiting water production, limiting gas lift availability, and examining the effect of wellhead choke. The findings of only two wells are displayed for each case, while the remaining ones are listed in Appendix A.

4.5.1 Constraints on Water Production

With consideration of the water cut (WC) values in Table 4.5, lower values were examined to determine the impact of restricted water production on the rate of oil production per well. In Well_1, the base case exhibited a water cut of approximately 80%. However, Table 4.14 illustrates a reduction of 60 and 40 percent, respectively. Evidently, these cases resulted in an increase in oil production. A reduced water cut implies that a greater proportion of the obtained fluid is oil. This can result in elevated oil recovery rates as a more significant portion of the fluid being produced constitutes a valuable hydrocarbon yield. The linear interpolation method was preferred for WC cases as it showed a lower percentage error when compared to the cubic method. The Python results in the second scenario (WC = 40) exhibit a deviation of less than 3%.

Table 4.14: Two Constraints for Water Cut in Well_1 and Results from GAP and PROSPER



Well_1		Case-1; WC= 60 Percent				Case-2; WC= 40 Percent			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	1257.4	1264.2	0.005	0.54	1338.1	1375.5	0.028	2.80
BHP	[BARa]	154.71	154.4	0.002	0.20	144.07	145.02	0.007	0.66
qo	[Sm3/day]	502.9	505.68	0.006	0.55	802.8	825.3	0.028	2.80
qw	[Sm3/day]	754.4	758.52	0.005	0.55	535.2	550.2	0.028	2.80
qg	[1000 Sm3/day]	71.301	71.7	0.006	0.56	113.817	117.06	0.028	2.85

By lowering the initial water cut value of 12% in Well_4, the production of oil values increased, Table 4.15. Furthermore, the Python outcome corresponds to the first situation, but there is a 7% difference in the following one.

Table 4.15: Two Constraints for Water Cut in Well_4 and Results from GAP and PROSPER

Well_4		Case-1; WC= 9 Percent				Case-2; WC= 7 Percent			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	414.6	413.51	0.003	0.26	385.9	414.69	0.075	7.46
BHP	[BARa]	75.64	75.88	0.003	0.32	75.07	75.28	0.003	0.28
qo	[Sm3/day]	377.3	376.29	0.003	0.27	385.9	385.66	0.001	0.06
qw	[Sm3/day]	37.3	37.21	0.002	0.24	29	29.02	0.001	0.07
qg	[1000 Sm3/day]	33.42	33.37	0.001	0.15	34.19	34.2	0.000	0.03

Figure 4.4 displays the VLP curves, which show the impact of restricting water production. The plots below show that the operating points were raised due to the increase in the limitation of the water cut.

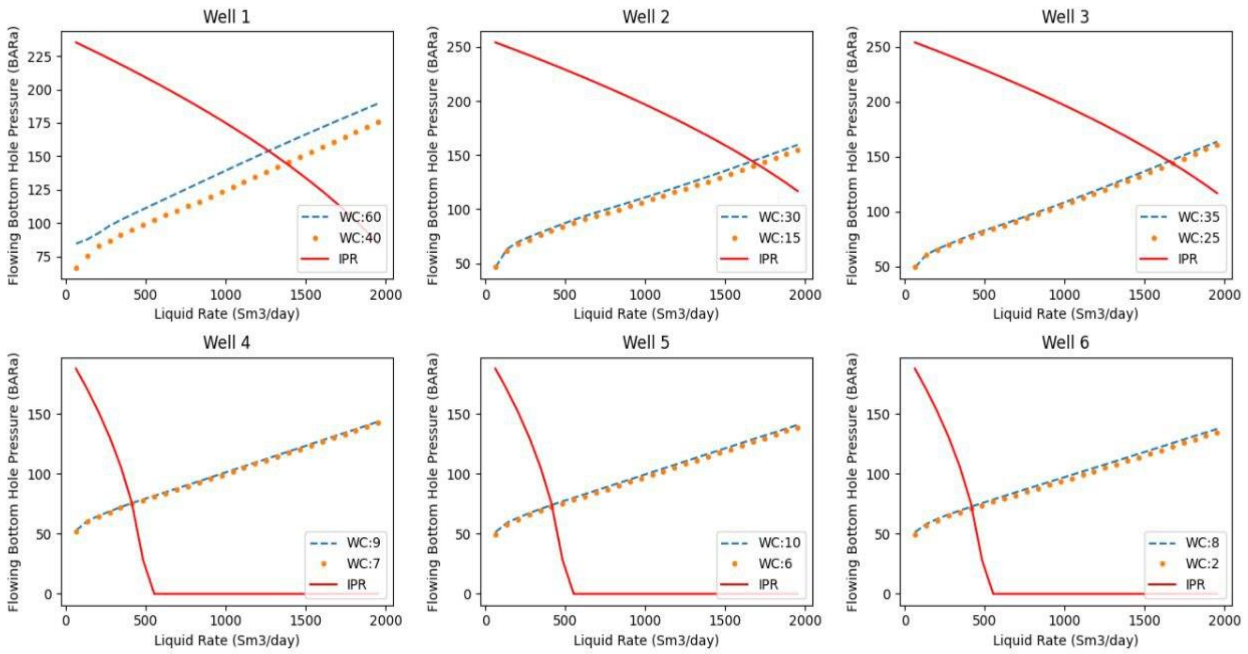


Figure4.4: Impact of Various amount of Water Cut on the Performance of the Wells

4.5.2 Constraints on Gas Lift Injection

According to Table 4.10, the initial gas lift injection rate for Well_1 was approximately 181.482 (1000 Sm3/day). However, in Table 4.16, there is a restriction of 160 and 120 (1000 Sm3/day) for this quantity. It is evident that there is a decline in oil production in both instances. Gas lift injection facilitates the transportation of oil to the surface, especially in wells where the natural reservoir energy is inadequate. Reduced production rates may result from the restriction of gas lift injection. There is a resemblance between the results obtained from Python in this well and those of GAP, with a slight difference of 0.5%.

Table 4.16: Production Rates for Well_1 with Two Different Gas Lift Injection Rates (QGL) in Both GAP and Python

Well_1		Case -1; QGL=160 MSm3/day				Case -2; QGL=120 MSm3/day			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	1270.50	1263.45	0.01	0.55	1242.70	1241.40	0.00	0.10
qo	[Sm3/day]	254.10	252.69	0.01	0.55	248.50	248.28	0.00	0.09
qw	[Sm3/day]	1016.40	1010.76	0.01	0.55	994.10	993.13	0.00	0.10
qg	[1000Sm3/day]	36.02	35.98	0.00	0.12	35.23	35.31	0.00	0.23

In Well_4, modifying the gas lift injection from 186.822 (1000 Sm3/day), as indicated in Table 4.11, to 160 and 100 (1000 Sm3/day), resulted in a decrease in the oil production rates, as illustrated in Table 4.17. Similar outcomes are observed in Python and GAP in this particular case.

Table 4.17: Production Rates for Well_4 with Two Different gas lift injection Rates (QGL) in Both GAP and Python

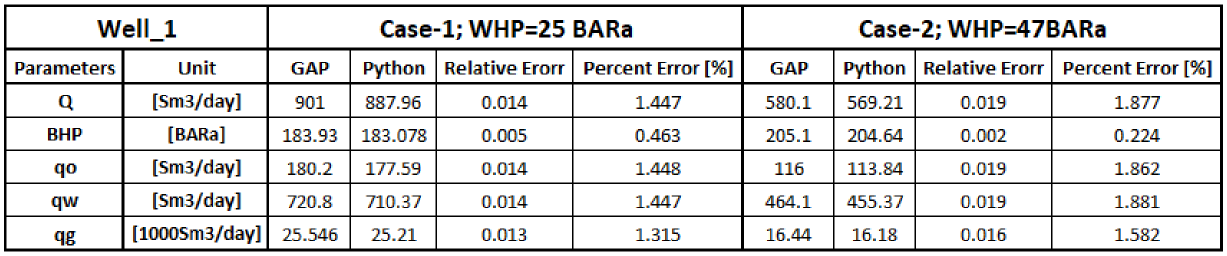
Well_4		Case -1; QGL=160 MSm3/day				Case -2; QGL=100 MSm3/day			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	428.6	425.82	0.006	0.649	425.5	423.73	0.004	0.416
qo	[Sm3/day]	377.1	374.72	0.006	0.631	374.4	372.89	0.004	0.403
qw	[Sm3/day]	51.4	51.09	0.006	0.603	51.1	50.84	0.005	0.509
qg	[1000Sm3/day]	33.41	33.36	0.001	0.150	33.178	33.138	0.001	0.121

4.5.3 Optimization with both Gas Lifting and Choke Opening

The impact of constrictions on gas-lifted wells is a crucial factor in oil and gas production, particularly when employing gas lift as an artificial lift technique. In the production system, the choke valve is of utmost importance as it governs the fluid flow from the well to the surface. The choke valve is primarily responsible for regulating the flow rate of fluids, comprising oil, gas, and water, from the well to the surface facilities. Modifying the choke opening directly influences the flow rates of these components.

The wellhead pressure remained uniform across all wells throughout this thesis, measuring at around 14.8 BARa according to Table 4.5. The effect of wellhead choke opening is visible when the wellhead values are increased in this case. Table 4.18 demonstrates that, for Well_1, raising the WHP led to diminished oil production in both instances with wellhead pressures of 26 and 47 BARa. Moreover, the magnitude of the water and gas yield dropped. Table 4.18 shows that the results of Python were similar to those of GAP.

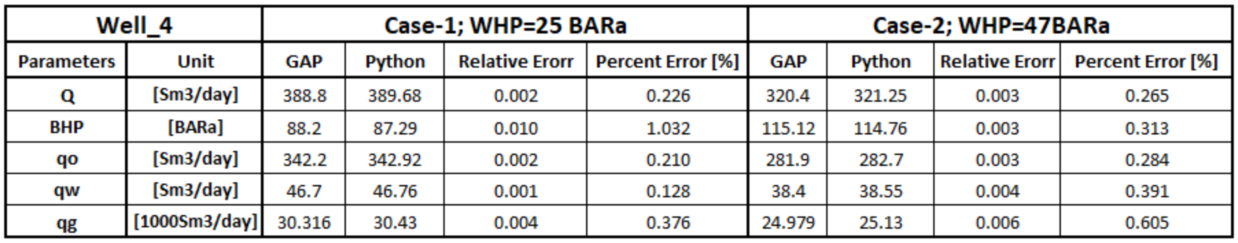
Table 4.18: Choke Partially Closing Test Cases for Well_1 in Both GAP and Python



Well_1		Case-1; WHP=25 BARa				Case-2; WHP=47BARa			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	901	887.96	0.014	1.447	580.1	569.21	0.019	1.877
BHP	[BARa]	183.93	183.078	0.005	0.463	205.1	204.64	0.002	0.224
qo	[Sm3/day]	180.2	177.59	0.014	1.448	116	113.84	0.019	1.862
qw	[Sm3/day]	720.8	710.37	0.014	1.447	464.1	455.37	0.019	1.881
qg	[1000Sm3/day]	25.546	25.21	0.013	1.315	16.44	16.18	0.016	1.582

In both Python and GAP analyses, the oil production rates for Well_4 diminish with the escalation of WHP.

Table 4.19: Choke Partially Closing Test Cases for Well_4 in Both GAP and Python



Well_4		Case-1; WHP=25 BARa				Case-2; WHP=47BARa			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	388.8	389.68	0.002	0.226	320.4	321.25	0.003	0.265
BHP	[BARa]	88.2	87.29	0.010	1.032	115.12	114.76	0.003	0.313
qo	[Sm3/day]	342.2	342.92	0.002	0.210	281.9	282.7	0.003	0.284
qw	[Sm3/day]	46.7	46.76	0.001	0.128	38.4	38.55	0.004	0.391
qg	[1000Sm3/day]	30.316	30.43	0.004	0.376	24.979	25.13	0.006	0.605

Chapter 5

Conclusions and Further Work

The concluding section of this thesis consists of two main parts, namely, a summary and conclusions, and recommendations for further research. The first part describes the primary findings of the study along with a discussion of them, while the second part provides suggestions for future research. To differentiate between the diverse results, discussion points, and future tasks, both the conclusion and further work are presented primarily as itemized lists.

5.1 Summary and Conclusions

Several points are outlined in this master's thesis to summarize the work accomplished:

- The optimization of gas lift in the network carried out through a production system with six gas lifted-wells in GAP.
- Python code was created to simulate gas lift optimization performance using the GAP package. To increase the capabilities of gas lift optimization in networks with many wells and challenging conditions. The required well models were generated from PROSPER for this solver.
- Python was able to efficiently import the well-models generated by PROSPER, and all necessary data, such as free variables and calculated variables, was saved in JSON format. Any number of free and calculated variables were handled well by the code.
- The interpolation class proved to be highly precise for both well model and random test point data sets. Due to the fewer errors, the cubic method was used for multidimensional interpolation instead of the linear.
- The well equilibrium rates function performed effectively, and its results for flow equilibrium, oil, water, and gas rates were similar to those obtained through the GAP. Six wells were plotted in Python to show the intersection points of the IPR and VLP curves.

- This solver considered the function created to optimize gas lift and maximize oil production in well networks. However, the results exhibited a significant deviation from the GAP findings. The divergence could be due to the varied optimization algorithms implemented in GAP and Python. The plotting of gas lift performance curves was done with Python.

- Three distinct case studies were conducted in both GAP and Python to determine the variables influencing oil production in the well network. The cases encompassed water production limitations, restricted gas lift availability, and optimization involving both gas lifting and choke opening. Among the test cases, limiting water production and choke opening were found to have a significant impact on optimizing oil production in the field.

5.2 Recommendation for Further Work

Several proposals for future work to enhance the scope of model-based gas lift optimization in well networks are presented. Generally, these suggestions consist of code enhancements and recommendations for optimizing gas lift.

- Utilize equivalent or possibly superior techniques and algorithms as those used by GAP for conducting field optimization in Python in order to achieve identical outcomes to those of GAP.

- Develop a Python solver with a user-friendly GUI and integrate advanced features from GAP, such as a dynamic adjustment of gas lift, unconstrained optimization, and potential constraint optimization, to streamline the optimization process for users.

- Include more variables that are positively associated with lifting performance in the free variables.

- Employing the Genetic Algorithm (GA) methodology for optimizing the allocation of the continuous gas-lift injection rate in a network system.

- The optimization of gas lift through the application of an artificial neural network and integrated production modeling in GAP.

Reference

[1] Petroleum Experts (Petex) Organization.(2023). *IPM Suite*. [Web Page], URL: https://www.petex.com/products/ipm-suite/

[2] Stanko, M. (2020). *Petroleum Production Systems*. [Compendium], Trondheim: The Norwegian University of Science and a Technology.

[3] Golan, M., & Whitson, C. H. (1991). *Well performance* (2nd ed.) [Book], Prentice-Hall.

[4] Fetoui, I. (2017, June 26). *Inflow Performance Relationship*. [Web Page], Production Technology. URL: https://production-technology.org/tag/inflow-performance-relationship/

[5] Vieira, C. R. G. (2015). *Model-based optimization of production systems.* [Master Thesis, NTNU]. URL: http://hdl.handle.net/11250/2351010

[6] Dharma, S. (2012). *Analyzing vertical lift performance in a complex gas lift well geometry*.(Master's thesis, University of Stavanger, Norway). URL: http://hdl.handle.net/11250/183618

[7] Shaikh.S. (2019). *Factors effecting vertical lift performance*. [Student presentation at Muet Khairpur], URL: https://www.slideshare.net/JALEEL48/factors-effecting-vertical-lift-performance

[8] Tuzovskiy, M. (2023.01.03). *Vertical Lift Performance*.

[Web Page], URL: https://wiki.pengtools.com/index.php?title=VLP

[9] Ibrahim, A. T. M. (2007). *Optimization of gas lift system in Varg field* [Master Thesis, University of Stavanger, Norway]. URL:http://hdl.handle.net/11250/183233

[10] Economides, M. J., Hill, A. D., Ehlig-Economides, C., & Zhu, D. (2012). *Petroleum Production Systems.* [Book], Pearson Education.
URL: https://books.google.no/books?id=qURhngEACAAJ

[11] Cooper, *J. Morgan, J. (Oct. 1,* 2001). *Artificial lift and pressure boosting options for production enhancement*.[Web Page], URL: https://www.offshore-mag.com/field-development/article/16758855/artificial-lift-and-pressure-boosting-options-for-production-enhancement

[12] Garcia, J. (2020, October 31). *Well Production Optimization: Going Back to Basics*. [Web Page], URL: https://www.linkedin.com/pulse/well-production-optimization-going-back-basics-jairo-b-garcia

[13] Siauw, T., & Bayen, A. (2014). *An introduction to MATLAB® programming and numerical methods for engineers.* [Book], Academic Press.

[14] NI Organization. (2023-02-21). *Interpolation*.[Web Page],
URL: https://www.ni.com/docs/en-US/bundle/labwindowscvi/page/advancedanalysisconcepts/interpolation_curve_fitting.html

[15] Bergmann, R. (January 12, 2022). *Polynomial interpolation Methods.* [PowerPoint slides], URL: https://www.math.ntnu.no/emner/TMA4125/2022v/lecture-notes/02-Polynomial-interpolation-methods.pdf

[16] Wikipedia. (2023, April 25). *Spline interpolation*.[Web Page],
URL: https://en.wikipedia.org/wiki/Spline_interpolation

[17] E. Balagurusamy.(1999). *Numerical Methods*. [Book], Tata McGraw-Hill Publishing Company Limited, New Delhi.

[18] Jadon, A. (May 6th, 2022). *Understanding JSON Modeling Simplified*. [Web Page],
URL：https://hevodata.com/learn/jsonmodeling/#:~:text=JSON%20Modeling%20has%20a%20data,the%20attributes%20of%20an%20entity.

[19] Friske, M. W., Buriol, L. S., & Camponogara, E. (2015). *A column generation approach for a compressor scheduling problem.* [Conference Article], Anais do XLVIII SBPO–Simpósio Brasileiro de Pesquisa Operacional, Porto de Galinas.

[20] Lv, T., Yan, P., & He, W. (2018). *Survey on JSON Data Modeling*. Journal of Physics: Conference Series, 1069, 012101. URL: https://doi.org/10.1088/1742-6596/1069/1/012101

[21] Wilson, J. M. (2003). *Gantt charts: A centenary appreciation*. European Journal of Operational Research, 149 (2), 430-437.
URL: https://doi.org/https://doi.org/10.1016/S0377-2217(02)00769-5

[22] The SciPy Community. (2023). *scipy.interpolate.RegularGridInterpolator.* [Web Page],URL:https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.RegularGridInterpolator.html

[23] Camargo, E., Aguilar, J., Ríos, A., Rivas, F., & Aguilar-Martín, J. (2008). *Nodal analysis-based design for improving gas lift wells production.* [Journal Article], WSEAS Transactions on Information Science & Applications, 5 (5), 706-715.

[24] Chong, E. K., & Żak, S. H. (2013). *An introduction to optimization (Vol. 75).* [Book], John Wiley & Sons.

[25] Wright, S. J. (2023, February 11). *Optimization mathematics*. Encyclopedia Britannica.[Web Page], URL: https://www.britannica.com/science/optimization

[26] Nakashima, P., & Camponogara, E. (2006). *Solving a gas-lift optimization problem by dynamic programming.* [Journal Article], European Journal of Operational Research 174 (2006) 1220–1246(Part A), 407-414.
URL: https://doi.org/doi:10.1016/j.ejor.2005.03.004

[27] Petroleum Experts. (March 2021). *User manual IPM GAP (Version 13.5).* Edinburgh, Scotland: Petroleum Experts, LTD,Pages 1028 – 1056.

[28] Brown, K. E. (1980). *The Technology of Artificial Lift Methods.* [Book], PennWell Corporation. URL: https://books.google.no/books?id=CNpwmwEACAAJ

[29] Okotie, S., & Ikporo, B. (2019). *Inflow Performance Relationship: Fundamentals and Applications*. [Book of Reservoir Engineering] (pp. 339-354). URL: https://doi.org/10.1007/978-3-030-02393-5_9

[30] Elias, M., El-Banbi, H. A., Fattah, K., & El-Tayeb, E.-S. A. M. (2009). *New Inflow Performance Relationship for Solution-Gas Drive Oil Reservoirs.* SPE Annual Technical Conference and Exhibition held in New Orleans, Louisiana, USA, URL: DOI:10.2118/124041-MS,

[31] MARCU, M. (2022). *Aspects Regarding The Calibration Of The Vertical Lift Performance Curves.* [Journal Article]. Acta Technica Napocensis-series: Applied Mathematics, Mechanics, and Engineering, 65 (2).

[32] Yang, Y. (2016). *A Reduced Order Model for Fast Production Prediction from an Oil Reservoir with a Gas Cap.* [Master Thesis, University of Stavanger, Norway]. URL: http://hdl.handle.net/11250/2414801

[33] Dai, Y.-H. (2002). *Convergence Properties of the BFGS Algorithm*. [Journal Article], SIAM Journal on Optimization, 13(3), 693-701. URL: https://doi.org/10.1137/s1052623401383455

[34] Takenaga, S., Ozaki, Y., & Onishi, M. (2023). *Practical initialization of the Nelder–Mead method for computationally expensive optimization problems*. [Journal Article], Optimization Letters, 17(2), 283–297. URL: https://doi.org/10.1007/s11590-022-01953-y

[35] Royer, C. W., O'Neill, M., & Wright, S. J. (2020). *A Newton-CG algorithm with complexity guarantees for smooth unconstrained optimization*. [Journal Article], Mathematical Programming, 180 (1-2), 451–488. URL: https://doi.org/10.1007/s10107-019-01362-7

[36] Steihaug, T. (1983). *The conjugate gradient method and trust regions in large-scale optimization*. SIAM Journal on Numerical Analysis, 20 (3), 626-637. URL: https://doi.org/https://doi.org/10.1137/0720042

[37] Kraft, D. (1988). *A Software Package for Sequential Quadratic Programming.*[Book], Wiss. Berichtswesen d. DFVLR.
URL: https://books.google.no/books?id=4rKaGwAACAAJ

[38] Marques, J. P. P. G., Cunha, D. C., Harada, L. M. F., Silva, L. N., & Silva, I. D. (2021). *A cost-effective trilateration-based radio localization algorithm using machine learning and sequential least-square programming optimization*. Computer communications, 177,Pages 1-9.
URL: https://doi.org/10.1016/j.comcom.2021.06.005

[39] Mascarenhas, E., & Pessoa, O. A. (2018). *Software for Evaluating IPR Composite in Grouped Subsaturated Reservoirs.* [Journal Article], IFAC-PapersOnline, 51 (8),108-112. URL: https://doi.org/DOI:10.1016/j.ifacol.2018.06.363

[40] Hulløen, B. N. (2022). *Gas Lift Optimization Using Python and PROSPER Well Models* [Master Thesis, NTNU].
URL: https://hdl.handle.net/11250/3021935

[41] Bandekian, S. (2022). *Gas Lift Optimization Using Python and PROSPER Well Models*, Specialization Project NTNU.

[42] Britannica, T. Editors of Encyclopaedia (2016, October 16). *Interpolation mathematics*. Encyclopedia Britannica.
[Web Page], URL: https://www.britannica.com/science/interpolation

[43] Wolfe, M. (2021). *Arrays vs List vs Dictionaries in Python*. [Web Page]
URL: https://python.plainenglish.io/arrays-vs-list-vs-dictionaries-47058fa19d4e

[44] Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., & Smith, N. J. (2020*). Array programming with NumPy*. [Journal Article], Nature, 585 (7825), 357-362.
URL: https://doi.org/https://doi.org/10.1038/s41586-020-2649-2

[45] Tateosian, L. (2015). *Python For ArcGIS*, 335-355. North Carolina State University [Book].
URL: https://doi.org/DOI:10.1007/978-3-319-18398-5

[46] Nelli, F. (2018). *Python data analytics with Pandas, NumPy, and Matplotlib* [Book, Chapter 2.].

[47] Nalum, K. (2013). *Modeling and Dynamic Optimization in Oil Production* [Master Thesis, NTNU,Institutt for teknisk kybernetikk].
URL: http://hdl.handle.net/11250/260881

[48] Stanghelle, K. U. (2009). *Evaluation of artificial lift methods on the Gyda field*. Master's thesis, University of Stavanger, Norway.
URL: http://hdl.handle.net/11250/183243

[49] Rashid, K., Bailey, W., & Couët, B. (2012). *A survey of methods for gas-lift optimization*. [Journal Article], Modelling and Simulation in Engineering, *2012*, 24-24.
URL: https://doi.org/https://doi.org/10.1155/2012/516807

[50] Hernandez, A. (2016). *Fundamentals of gas lift engineering: Well design and troubleshooting*. Gulf Professional Publishing. [Book], Chapter5, Pages151-209
URL:https://doi.org/10.1016/B978-0-12-804133-8.00005-1

[51] LabVIEWFundamentals, (2023,July17).*Spline Interpolation*. [Web Page],
URL:https://www.ni.com/docs/en-US/bundle/labview/page/spline-interpolation.html

[52] Salomon, D. (2007). *Curves and surfaces for computer graphics*. Springer Science & Business Media.[Book],Chapter5,Pages 141-146.
URL: https://link.springer.com/content/pdf/10.1007/0-387-28452-4.pdf

[53] Yakoot, M. S., Shedid, S. A., & Arafa, M. I. (2014). *A simulation approach for optimization of gas lift performance and multi-well networking in an Egyptian oil field*. SPE Reservoir Characterization and Simulation Conference and Exhibition held in Abu Dhabi.URL: DOI:10.4043/24703-MS

[54] Nishikiori, N., Redner, R. A., Doty, D. R., & Schmidt, Z. (1989). *An Improved Method for Gas Lift Allocation Optimization*. SPE Annual Technical Conference and Exhibition.URL: https://doi.org/10.2118/19711-MS,

[55] McKinley, S., & Levine, M. (1998). *Cubic spline interpolation*. [Journal Article], College of the Redwoods, 45(1), 1049-1060.
URL: https://mse.redwoods.edu/darnold/math45/laproj/Fall98/SkyMeg/Proj.PDF

Appendix A

A.1 Tables for Chapter Four

A.1.1 Constraints on Water Production

Table A.1: Two Constraints for Water Cut in Well_2 and Results from GAP and PROSPER

Well_2		Case-1; WC= 30 Percent				Case-2; WC= 15 Percent			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	1569.5	1664.83	0.061	6.07	1577.6	1696.09	0.075	7.51
BHP	[BARa]	141.1	144.9	0.027	2.69	137.06	142.08	0.037	3.66
qo	[Sm3/day]	1098.7	1165.38	0.061	6.07	1340.9	1441.67	0.075	7.52
qw	[Sm3/day]	470.9	499.44	0.061	6.06	236.6	254.41	0.075	7.53
qg	[1000 Sm3/day]	155.75	165.29	0.061	6.13	190.102	204.45	0.075	7.55

Table A.2: Two Constraints for Water Cut in Well_3 and Results from GAP and PROSPER

Well_3		Case-1; WC= 35 Percent				Case-2; WC= 25 Percent			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	1569.4	1650.027	0.051	5.14	1572.7	1666.87	0.060	5.99
BHP	[BARa]	142.19	146.22	0.028	2.83	139.72	144.71	0.036	3.57
qo	[Sm3/day]	1020.1	1072.51	0.051	5.14	1179.5	1250.15	0.060	5.99
qw	[Sm3/day]	549.3	577.5	0.051	5.13	393.2	416.71	0.060	5.98
qg	[1000 Sm3/day]	144.615	152.08	0.052	5.16	167.22	177.25	0.060	6.00

Table A.3: Two Constraints for Water Cut in Well_5 and Results from GAP and PROSPER

Well_5		Case-1; WC= 10 Percent				Case-2; WC= 6 Percent			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	418.2	416.47	0.004	0.41	418.8	418.71	0.000	0.02
BHP	[BARa]	73.98	74.36	0.005	0.51	72.91	73.2	0.004	0.40
qo	[Sm3/day]	376.4	374.83	0.004	0.42	393.7	393.59	0.000	0.03
qw	[Sm3/day]	41.8	41.64	0.004	0.38	25.1	25.12	0.001	0.08
qg	[1000 Sm3/day]	33.35	33.24	0.003	0.33	34.88	34.9	0.001	0.06

Table A.4: Two Constraints for Water Cut in Well_6 and Results from GAP and PROSPER

Well_6		Case-1; WC= 8 Percent				Case-2; WC= 2 Percent			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	420.4	418.95	0.003	0.34	420.7	421.59	0.002	0.21
BHP	[BARa]	72.45	73.08	0.009	0.87	71.11	71.68	0.008	0.80
qo	[Sm3/day]	386.8	385.43	0.004	0.35	412.3	413.16	0.002	0.21
qw	[Sm3/day]	33.6	33.51	0.003	0.27	8.4	8.43	0.004	0.36
qg	[1000 Sm3/day]	34.27	34.17	0.003	0.29	36.52	36.63	0.003	0.30

A.1.2 Constraint on Gas Lift Injection

Table A.5: Production Rates for Well_2 with Two Different gas lift injection Rates (QGL) in Both GAP and Python

Well_2		Case -1; QGL=150 MSm3/day				Case -2; QGL=130 MSm3/day			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	1570.9	1661.58	0.058	5.772	1569.6	1657.66	0.056	5.61
qo	[Sm3/day]	942.5	996.94	0.058	5.776	941.7	994.59	0.056	5.62
qw	[Sm3/day]	628.3	664.63	0.058	5.782	627.8	663.066	0.056	5.62
qg	[1000Sm3/day]	133.618	141.47	0.059	5.876	133.507	141.12	0.057	5.70

Table A.6: Production Rates for Well_3 with Two Different gas lift injection Rates (QGL) in Both GAP and Python

Well_3		Case -1; QGL=100 MSm3/day				Case -2; QGL=80 MSm3/day			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	1581.6	1663.36	0.052	5.169	1576.6	1657.87	0.0515	5.155
qo	[Sm3/day]	949	998.019	0.052	5.165	946	994.72	0.0515	5.150
qw	[Sm3/day]	632.7	665.34	0.052	5.159	630.7	663.15	0.0515	5.145
qg	[1000Sm3/day]	134.53	141.57	0.052	5.233	134.109	141.09	0.0521	5.205

Table A.7: Production Rates for Well_5 with Two Different gas lift injection Rates (QGL) in Both GAP and Python

Well_5		Case -1; QGL=170 MSm3/day				Case -2; QGL=70 MSm3/day			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	432.5	429.87	0.006	0.608	425.6	424.54	0.002	0.249
qo	[Sm3/day]	380.6	378.29	0.006	0.607	374.5	373.6	0.002	0.240
qw	[Sm3/day]	51.9	51.58	0.006	0.617	51.1	50.94	0.003	0.313
qg	[1000Sm3/day]	33.72	33.68	0.001	0.119	33.18	33.17	0.000	0.030

Table A.8: Production Rates for Well_6 with Two Different gas lift injection Rates (QGL) in Both GAP and Python

Well_6		Case -1; QGL=100 MSm3/day				Case -2; QGL=40 MSm3/day			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	431.9	430.11	0.004	0.414	422.6	421.21	0.003	0.329
qo	[Sm3/day]	380.1	378.49	0.004	0.424	371.8	370.66	0.003	0.307
qw	[Sm3/day]	51.8	51.61	0.004	0.367	50.7	50.54	0.003	0.316
qg	[1000Sm3/day]	33.67	33.63	0.001	0.119	32.94	32.88	0.002	0.182

A.1.3 Optimization with both Gas Lifting and Choke Opening

Table A.9: Choke Partially Closing Test Cases for Well_2 in Both GAP and Python

Well_2		Case-1; WHP=25 BARa				Case-2; WHP=47BARa			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	1461	1538.15	0.053	5.281	1244.8	1313.23	0.055	5.497
BHP	[BARa]	153.36	155.88	0.016	1.643	172.23	173.93	0.010	0.987
qo	[Sm3/day]	876.6	922.89	0.053	5.281	746.9	787.9	0.055	5.489
qw	[Sm3/day]	584.4	615.26	0.053	5.281	497.9	525.29	0.055	5.501
qg	[1000Sm3/day]	124.275	130.93	0.054	5.355	105.881	111.84	0.056	5.628

Table A.10: Choke Partially Closing Test Cases for Well_3 in Both GAP and Python

Well_3		Case-1; WHP=25 BARa				Case-2; WHP=47BARa			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	1480.9	1559.41	0.053	5.302	1274.4	1332.02	0.045	4.521
BHP	[BARa]	151.54	154.08	0.017	1.676	169.74	172.48	0.016	1.614
qo	[Sm3/day]	888.6	935.64	0.053	5.294	764.6	799.21	0.045	4.527
qw	[Sm3/day]	592.4	623.76	0.053	5.294	509.8	532.81	0.045	4.514
qg	[1000Sm3/day]	125.968	132.7	0.053	5.344	108.4	113.38	0.046	4.594

Table A.11: Choke Partially Closing Test Cases for Well_5 in Both GAP and Python

Well_5		Case-1; WHP=25 BARa				Case-2; WHP=47BARa			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	394.2	395.23	0.003	0.261	330.9	330.66	0.001	0.073
BHP	[BARa]	85.8	84.75	0.012	1.224	111.32	111.33	0.000	0.009
qo	[Sm3/day]	346.9	347.8	0.003	0.259	291.2	290.9	0.001	0.103
qw	[Sm3/day]	47.3	47.42	0.003	0.254	39.7	39.67	0.001	0.076
qg	[1000Sm3/day]	30.736	30.87	0.004	0.436	25.801	25.87	0.003	0.267

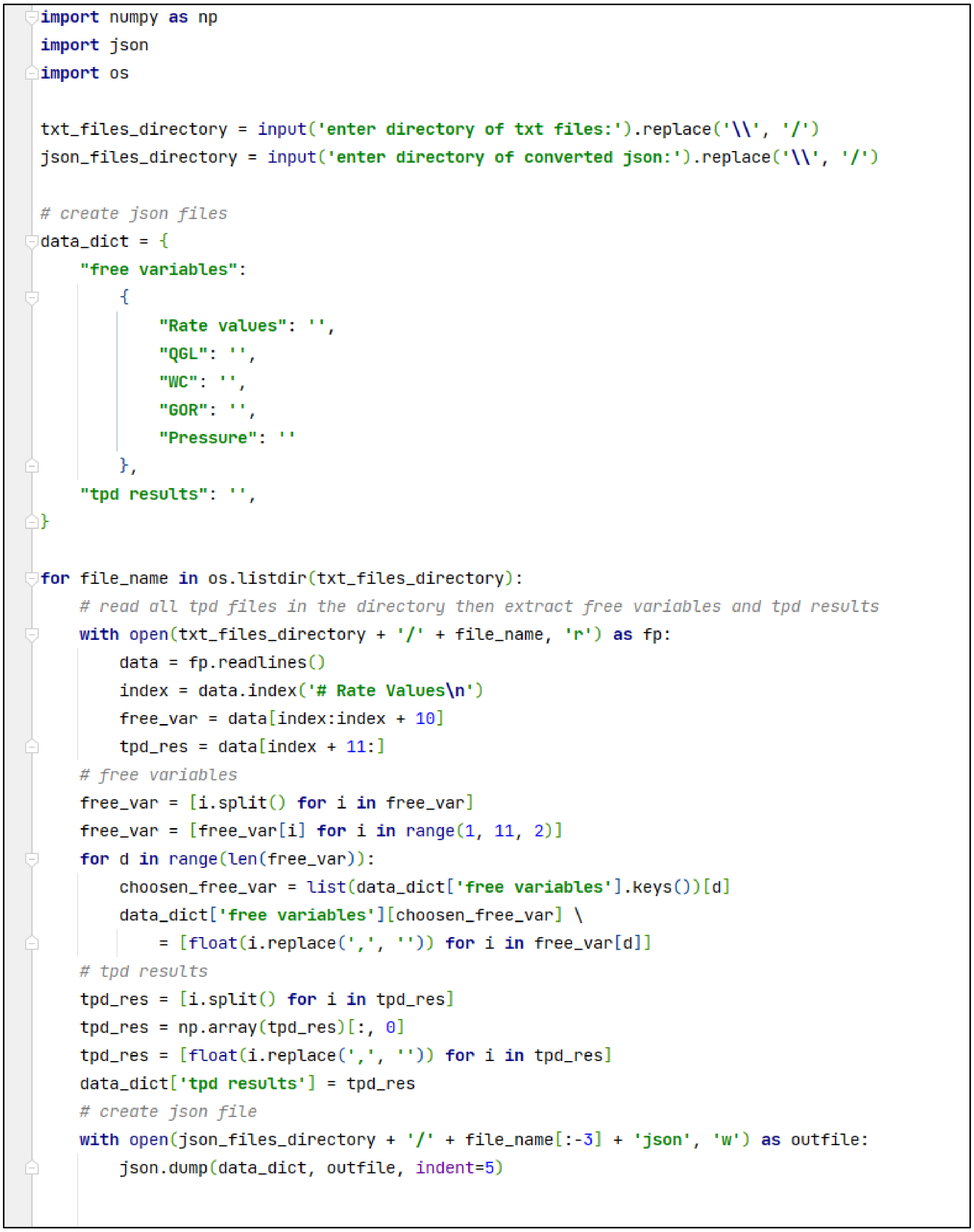
Table A.12: Choke Partially Closing Test Cases for Well_6 in Both GAP and Python

Well_6		Case-1; WHP=25 BARa				Case-2; WHP=47BARa			
Parameters	Unit	GAP	Python	Relative Erorr	Percent Error [%]	GAP	Python	Relative Erorr	Percent Error [%]
Q	[Sm3/day]	397.7	398.54	0.002	0.211	335.3	334.92	0.001	0.113
BHP	[BARa]	84.22	83.2	0.012	1.211	109.71	109.74	0.000	0.027
qo	[Sm3/day]	350	350.72	0.002	0.206	295	294.73	0.001	0.092
qw	[Sm3/day]	47.7	47.82	0.003	0.252	40.2	40.19	0.000	0.025
qg	[1000Sm3/day]	31.008	31.12	0.004	0.361	26.143	26.2	0.002	0.218

Appendix B

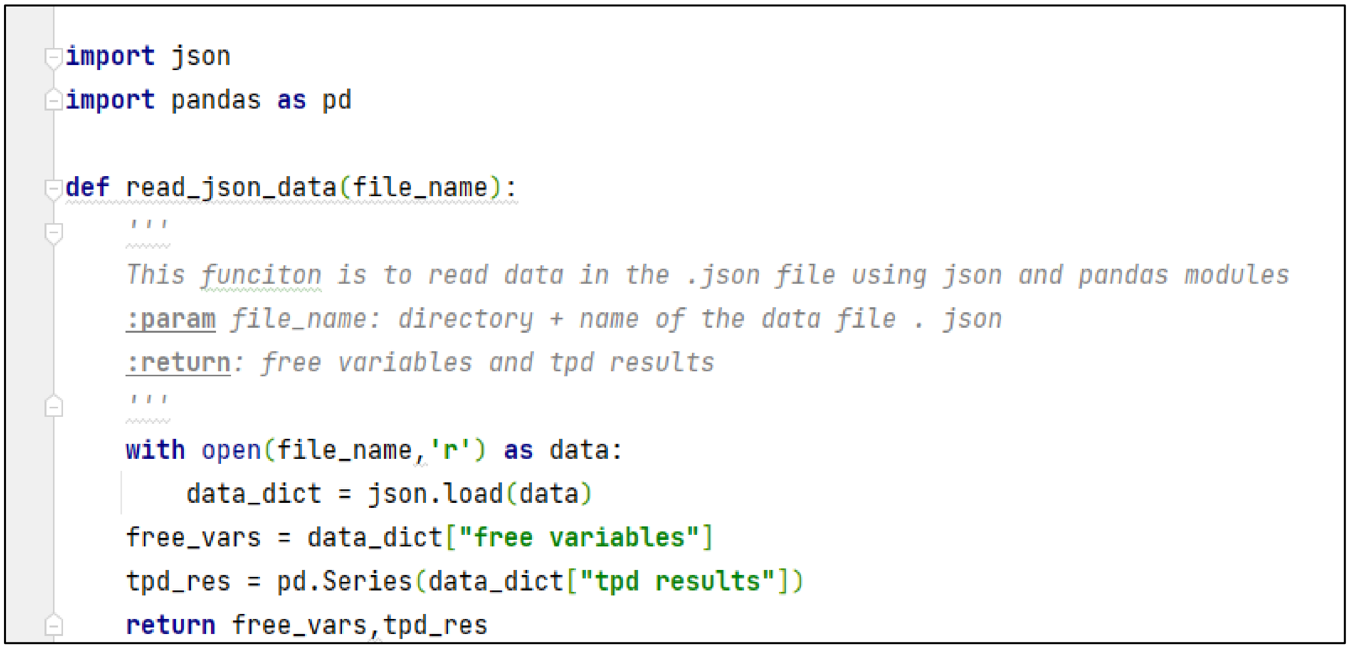
Python Code

B.1 Read Model



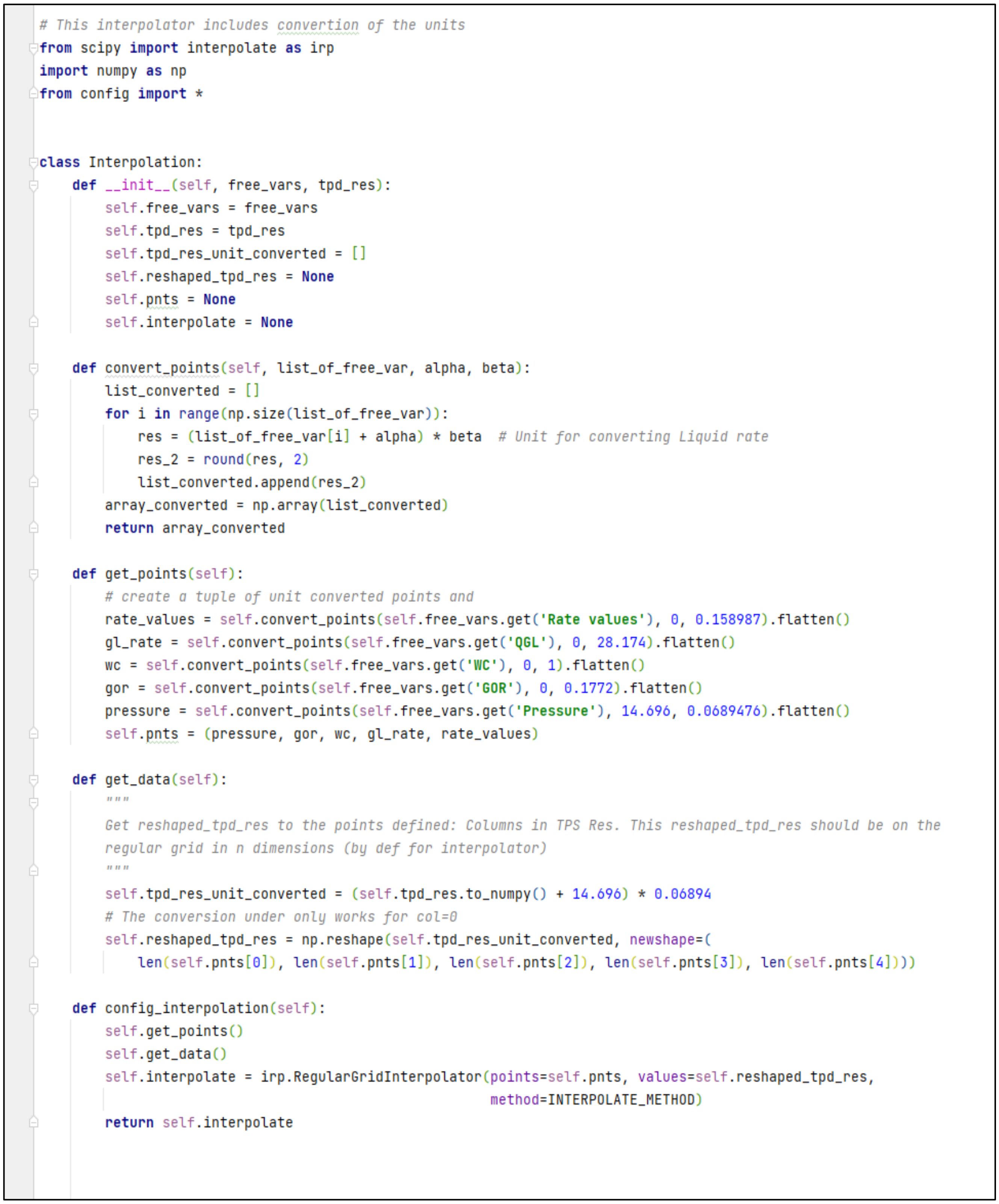
```python
import numpy as np
import json
import os

txt_files_directory = input('enter directory of txt files:').replace('\\', '/')
json_files_directory = input('enter directory of converted json:').replace('\\', '/')

# create json files
data_dict = {
    "free variables":
        {
            "Rate values": '',
            "QGL": '',
            "WC": '',
            "GOR": '',
            "Pressure": ''
        },
    "tpd results": '',
}

for file_name in os.listdir(txt_files_directory):
    # read all tpd files in the directory then extract free variables and tpd results
    with open(txt_files_directory + '/' + file_name, 'r') as fp:
        data = fp.readlines()
        index = data.index('# Rate Values\n')
        free_var = data[index:index + 10]
        tpd_res = data[index + 11:]
    # free variables
    free_var = [i.split() for i in free_var]
    free_var = [free_var[i] for i in range(1, 11, 2)]
    for d in range(len(free_var)):
        choosen_free_var = list(data_dict['free variables'].keys())[d]
        data_dict['free variables'][choosen_free_var] \
            = [float(i.replace(',', '')) for i in free_var[d]]
    # tpd results
    tpd_res = [i.split() for i in tpd_res]
    tpd_res = np.array(tpd_res)[:, 0]
    tpd_res = [float(i.replace(',', '')) for i in tpd_res]
    data_dict['tpd results'] = tpd_res
    # create json file
    with open(json_files_directory + '/' + file_name[:-3] + 'json', 'w') as outfile:
        json.dump(data_dict, outfile, indent=5)
```

B.2 JSON Format



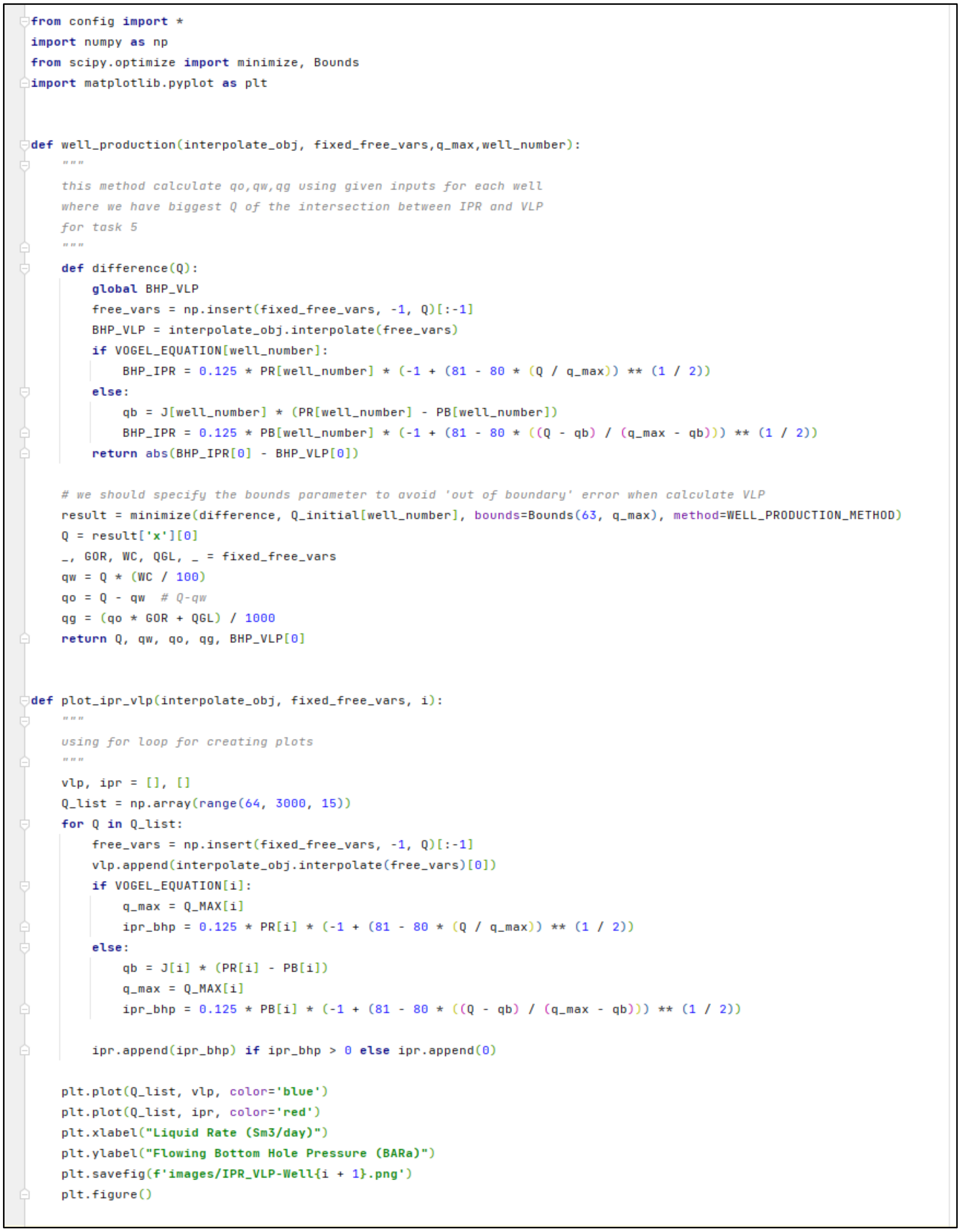
```python
import json
import pandas as pd

def read_json_data(file_name):
    '''
    This funciton is to read data in the .json file using json and pandas modules
    :param file_name: directory + name of the data file . json
    :return: free variables and tpd results
    '''
    with open(file_name,'r') as data:
        data_dict = json.load(data)
    free_vars = data_dict["free variables"]
    tpd_res = pd.Series(data_dict["tpd results"])
    return free_vars,tpd_res
```

B.3 Interpolation



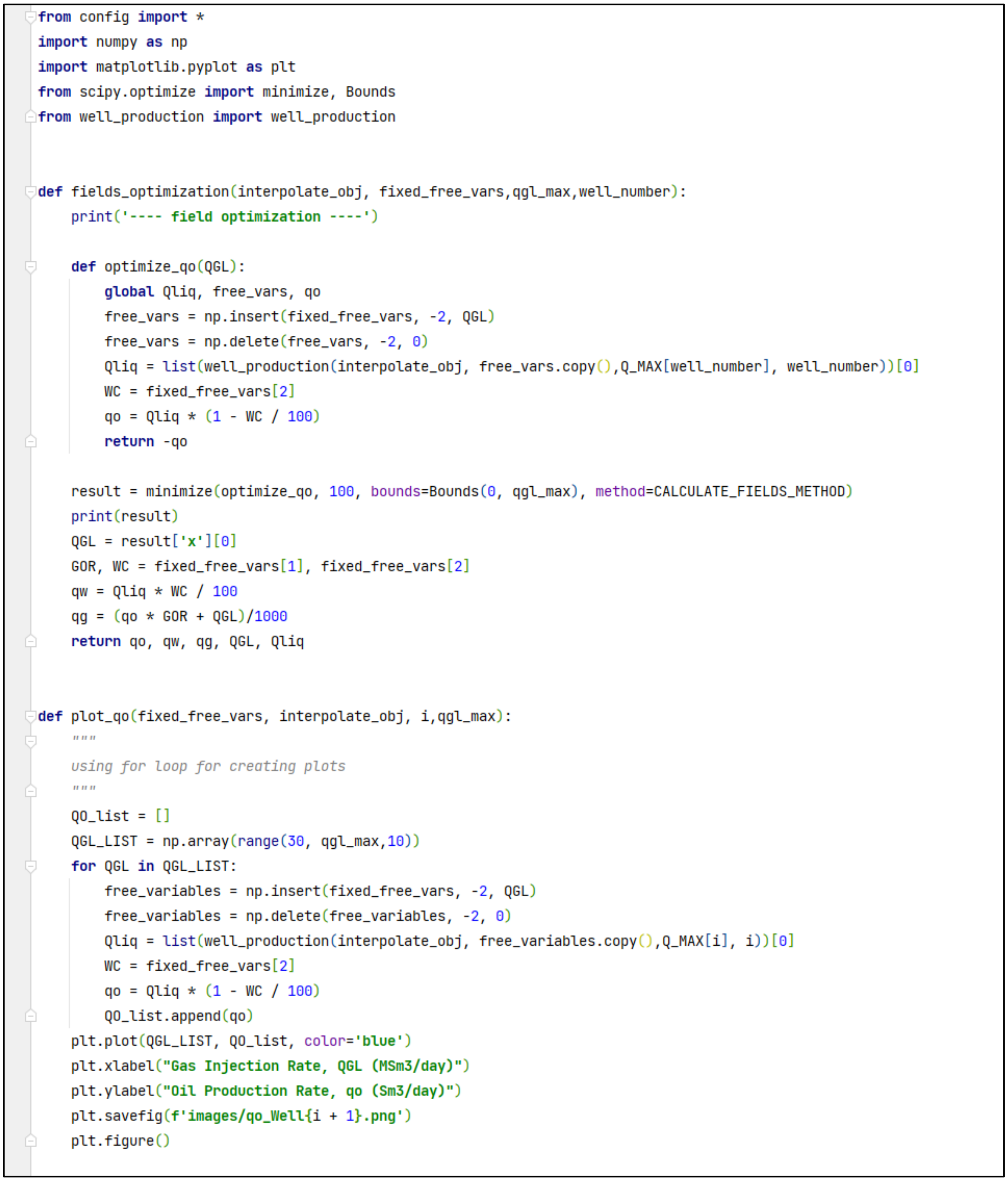
```python
# This interpolator includes convertion of the units
from scipy import interpolate as irp
import numpy as np
from config import *


class Interpolation:
    def __init__(self, free_vars, tpd_res):
        self.free_vars = free_vars
        self.tpd_res = tpd_res
        self.tpd_res_unit_converted = []
        self.reshaped_tpd_res = None
        self.pnts = None
        self.interpolate = None

    def convert_points(self, list_of_free_var, alpha, beta):
        list_converted = []
        for i in range(np.size(list_of_free_var)):
            res = (list_of_free_var[i] + alpha) * beta  # Unit for converting Liquid rate
            res_2 = round(res, 2)
            list_converted.append(res_2)
        array_converted = np.array(list_converted)
        return array_converted

    def get_points(self):
        # create a tuple of unit converted points and
        rate_values = self.convert_points(self.free_vars.get('Rate values'), 0, 0.158987).flatten()
        gl_rate = self.convert_points(self.free_vars.get('QGL'), 0, 28.174).flatten()
        wc = self.convert_points(self.free_vars.get('WC'), 0, 1).flatten()
        gor = self.convert_points(self.free_vars.get('GOR'), 0, 0.1772).flatten()
        pressure = self.convert_points(self.free_vars.get('Pressure'), 14.696, 0.0689476).flatten()
        self.pnts = (pressure, gor, wc, gl_rate, rate_values)

    def get_data(self):
        """
        Get reshaped_tpd_res to the points defined: Columns in TPS Res. This reshaped_tpd_res should be on the
        regular grid in n dimensions (by def for interpolator)
        """
        self.tpd_res_unit_converted = (self.tpd_res.to_numpy() + 14.696) * 0.06894
        # The conversion under only works for col=0
        self.reshaped_tpd_res = np.reshape(self.tpd_res_unit_converted, newshape=(
            len(self.pnts[0]), len(self.pnts[1]), len(self.pnts[2]), len(self.pnts[3]), len(self.pnts[4])))

    def config_interpolation(self):
        self.get_points()
        self.get_data()
        self.interpolate = irp.RegularGridInterpolator(points=self.pnts, values=self.reshaped_tpd_res,
                                                       method=INTERPOLATE_METHOD)
        return self.interpolate
```

B.4 Well Production



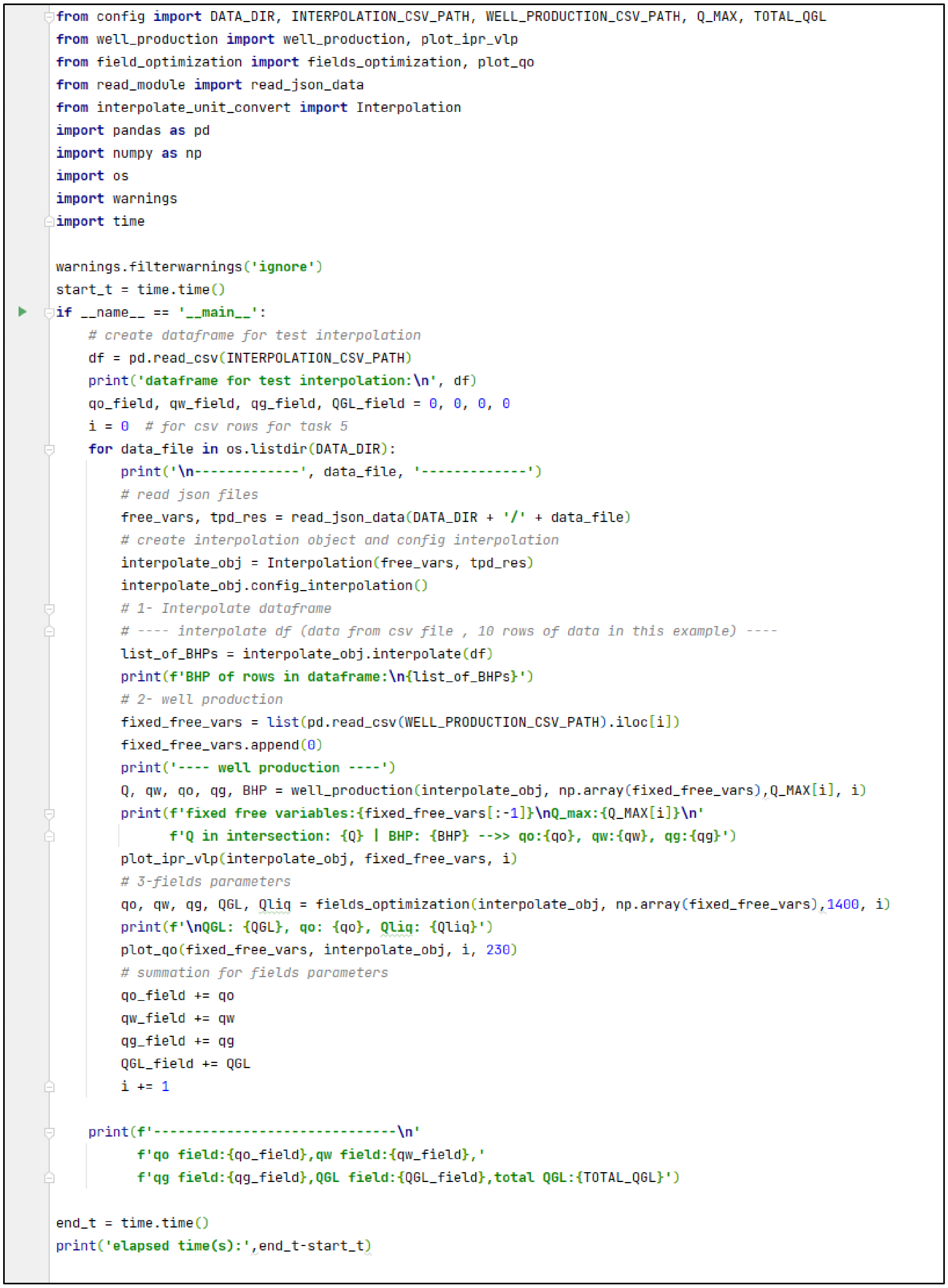
```python
from config import *
import numpy as np
from scipy.optimize import minimize, Bounds
import matplotlib.pyplot as plt


def well_production(interpolate_obj, fixed_free_vars,q_max,well_number):
    """
    this method calculate qo,qw,qg using given inputs for each well
    where we have biggest Q of the intersection between IPR and VLP
    for task 5
    """
    def difference(Q):
        global BHP_VLP
        free_vars = np.insert(fixed_free_vars, -1, Q)[:-1]
        BHP_VLP = interpolate_obj.interpolate(free_vars)
        if VOGEL_EQUATION[well_number]:
            BHP_IPR = 0.125 * PR[well_number] * (-1 + (81 - 80 * (Q / q_max)) ** (1 / 2))
        else:
            qb = J[well_number] * (PR[well_number] - PB[well_number])
            BHP_IPR = 0.125 * PB[well_number] * (-1 + (81 - 80 * ((Q - qb) / (q_max - qb))) ** (1 / 2))
        return abs(BHP_IPR[0] - BHP_VLP[0])

    # we should specify the bounds parameter to avoid 'out of boundary' error when calculate VLP
    result = minimize(difference, Q_initial[well_number], bounds=Bounds(63, q_max), method=WELL_PRODUCTION_METHOD)
    Q = result['x'][0]
    _, GOR, WC, QGL, _ = fixed_free_vars
    qw = Q * (WC / 100)
    qo = Q - qw  # Q-qw
    qg = (qo * GOR + QGL) / 1000
    return Q, qw, qo, qg, BHP_VLP[0]


def plot_ipr_vlp(interpolate_obj, fixed_free_vars, i):
    """
    using for loop for creating plots
    """
    vlp, ipr = [], []
    Q_list = np.array(range(64, 3000, 15))
    for Q in Q_list:
        free_vars = np.insert(fixed_free_vars, -1, Q)[:-1]
        vlp.append(interpolate_obj.interpolate(free_vars)[0])
        if VOGEL_EQUATION[i]:
            q_max = Q_MAX[i]
            ipr_bhp = 0.125 * PR[i] * (-1 + (81 - 80 * (Q / q_max)) ** (1 / 2))
        else:
            qb = J[i] * (PR[i] - PB[i])
            q_max = Q_MAX[i]
            ipr_bhp = 0.125 * PB[i] * (-1 + (81 - 80 * ((Q - qb) / (q_max - qb))) ** (1 / 2))

        ipr.append(ipr_bhp) if ipr_bhp > 0 else ipr.append(0)

    plt.plot(Q_list, vlp, color='blue')
    plt.plot(Q_list, ipr, color='red')
    plt.xlabel("Liquid Rate (Sm3/day)")
    plt.ylabel("Flowing Bottom Hole Pressure (BARa)")
    plt.savefig(f'images/IPR_VLP-Well{i + 1}.png')
    plt.figure()
```

B.5 Field Optimization



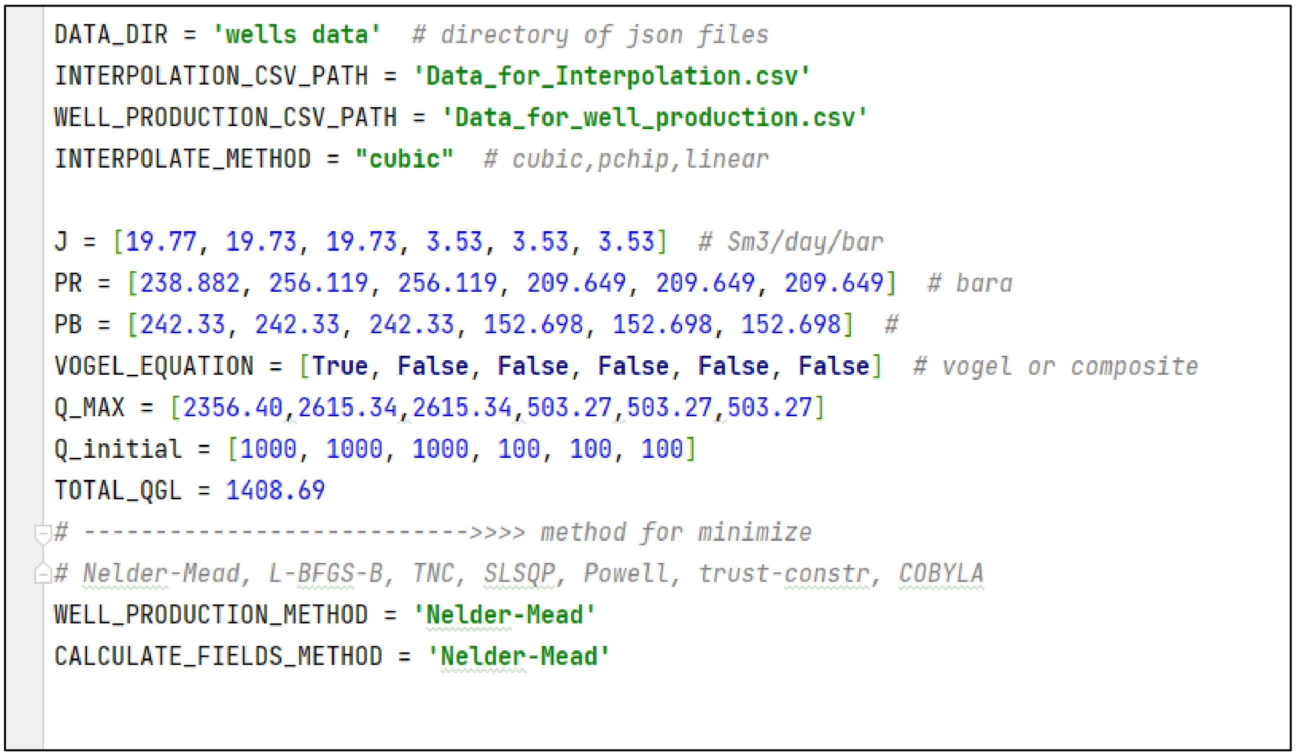
```python
from config import *
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize, Bounds
from well_production import well_production


def fields_optimization(interpolate_obj, fixed_free_vars,qgl_max,well_number):
    print('---- field optimization ----')

    def optimize_qo(QGL):
        global Qliq, free_vars, qo
        free_vars = np.insert(fixed_free_vars, -2, QGL)
        free_vars = np.delete(free_vars, -2, 0)
        Qliq = list(well_production(interpolate_obj, free_vars.copy(),Q_MAX[well_number], well_number))[0]
        WC = fixed_free_vars[2]
        qo = Qliq * (1 - WC / 100)
        return -qo

    result = minimize(optimize_qo, 100, bounds=Bounds(0, qgl_max), method=CALCULATE_FIELDS_METHOD)
    print(result)
    QGL = result['x'][0]
    GOR, WC = fixed_free_vars[1], fixed_free_vars[2]
    qw = Qliq * WC / 100
    qg = (qo * GOR + QGL)/1000
    return qo, qw, qg, QGL, Qliq


def plot_qo(fixed_free_vars, interpolate_obj, i,qgl_max):
    """
    using for loop for creating plots
    """
    QO_list = []
    QGL_LIST = np.array(range(30, qgl_max,10))
    for QGL in QGL_LIST:
        free_variables = np.insert(fixed_free_vars, -2, QGL)
        free_variables = np.delete(free_variables, -2, 0)
        Qliq = list(well_production(interpolate_obj, free_variables.copy(),Q_MAX[i], i))[0]
        WC = fixed_free_vars[2]
        qo = Qliq * (1 - WC / 100)
        QO_list.append(qo)
    plt.plot(QGL_LIST, QO_list, color='blue')
    plt.xlabel("Gas Injection Rate, QGL (MSm3/day)")
    plt.ylabel("Oil Production Rate, qo (Sm3/day)")
    plt.savefig(f'images/qo_Well{i + 1}.png')
    plt.figure()
```

B.6 Main



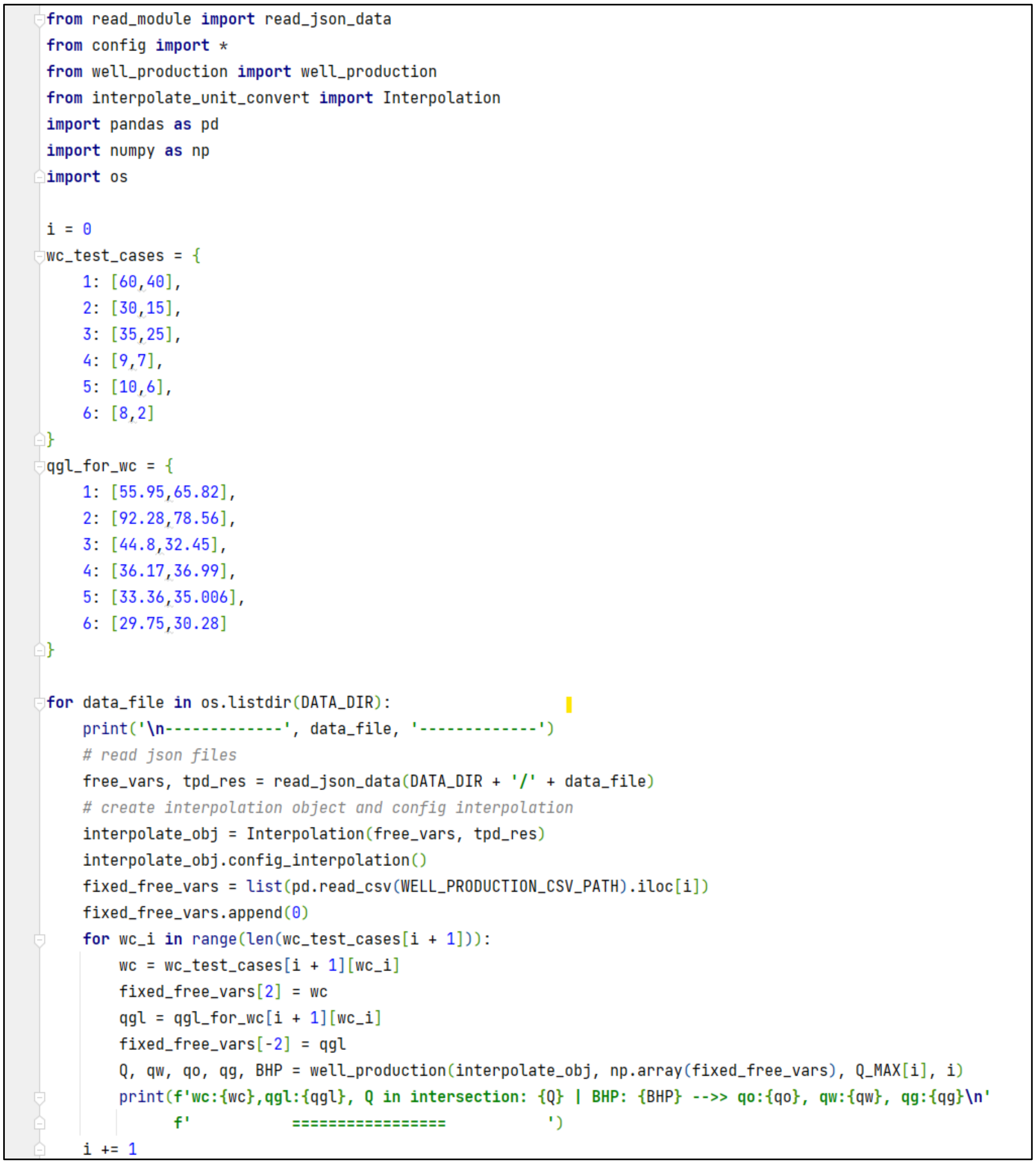
```python
from config import DATA_DIR, INTERPOLATION_CSV_PATH, WELL_PRODUCTION_CSV_PATH, Q_MAX, TOTAL_QGL
from well_production import well_production, plot_ipr_vlp
from field_optimization import fields_optimization, plot_qo
from read_module import read_json_data
from interpolate_unit_convert import Interpolation
import pandas as pd
import numpy as np
import os
import warnings
import time

warnings.filterwarnings('ignore')
start_t = time.time()
if __name__ == '__main__':
    # create dataframe for test interpolation
    df = pd.read_csv(INTERPOLATION_CSV_PATH)
    print('dataframe for test interpolation:\n', df)
    qo_field, qw_field, qg_field, QGL_field = 0, 0, 0, 0
    i = 0  # for csv rows for task 5
    for data_file in os.listdir(DATA_DIR):
        print('\n-------------', data_file, '-------------')
        # read json files
        free_vars, tpd_res = read_json_data(DATA_DIR + '/' + data_file)
        # create interpolation object and config interpolation
        interpolate_obj = Interpolation(free_vars, tpd_res)
        interpolate_obj.config_interpolation()
        # 1- Interpolate dataframe
        # ---- interpolate df (data from csv file , 10 rows of data in this example) ----
        list_of_BHPs = interpolate_obj.interpolate(df)
        print(f'BHP of rows in dataframe:\n{list_of_BHPs}')
        # 2- well production
        fixed_free_vars = list(pd.read_csv(WELL_PRODUCTION_CSV_PATH).iloc[i])
        fixed_free_vars.append(0)
        print('---- well production ----')
        Q, qw, qo, qg, BHP = well_production(interpolate_obj, np.array(fixed_free_vars),Q_MAX[i], i)
        print(f'fixed free variables:{fixed_free_vars[:-1]}\nQ_max:{Q_MAX[i]}\n'
              f'Q in intersection: {Q} | BHP: {BHP} -->> qo:{qo}, qw:{qw}, qg:{qg}')
        plot_ipr_vlp(interpolate_obj, fixed_free_vars, i)
        # 3-fields parameters
        qo, qw, qg, QGL, Qliq = fields_optimization(interpolate_obj, np.array(fixed_free_vars),1400, i)
        print(f'\nQGL: {QGL}, qo: {qo}, Qliq: {Qliq}')
        plot_qo(fixed_free_vars, interpolate_obj, i, 230)
        # summation for fields parameters
        qo_field += qo
        qw_field += qw
        qg_field += qg
        QGL_field += QGL
        i += 1

    print(f'-----------------------------\n'
          f'qo field:{qo_field},qw field:{qw_field},'
          f'qg field:{qg_field},QGL field:{QGL_field},total QGL:{TOTAL_QGL}')

end_t = time.time()
print('elapsed time(s):',end_t-start_t)
```

B.7 Config



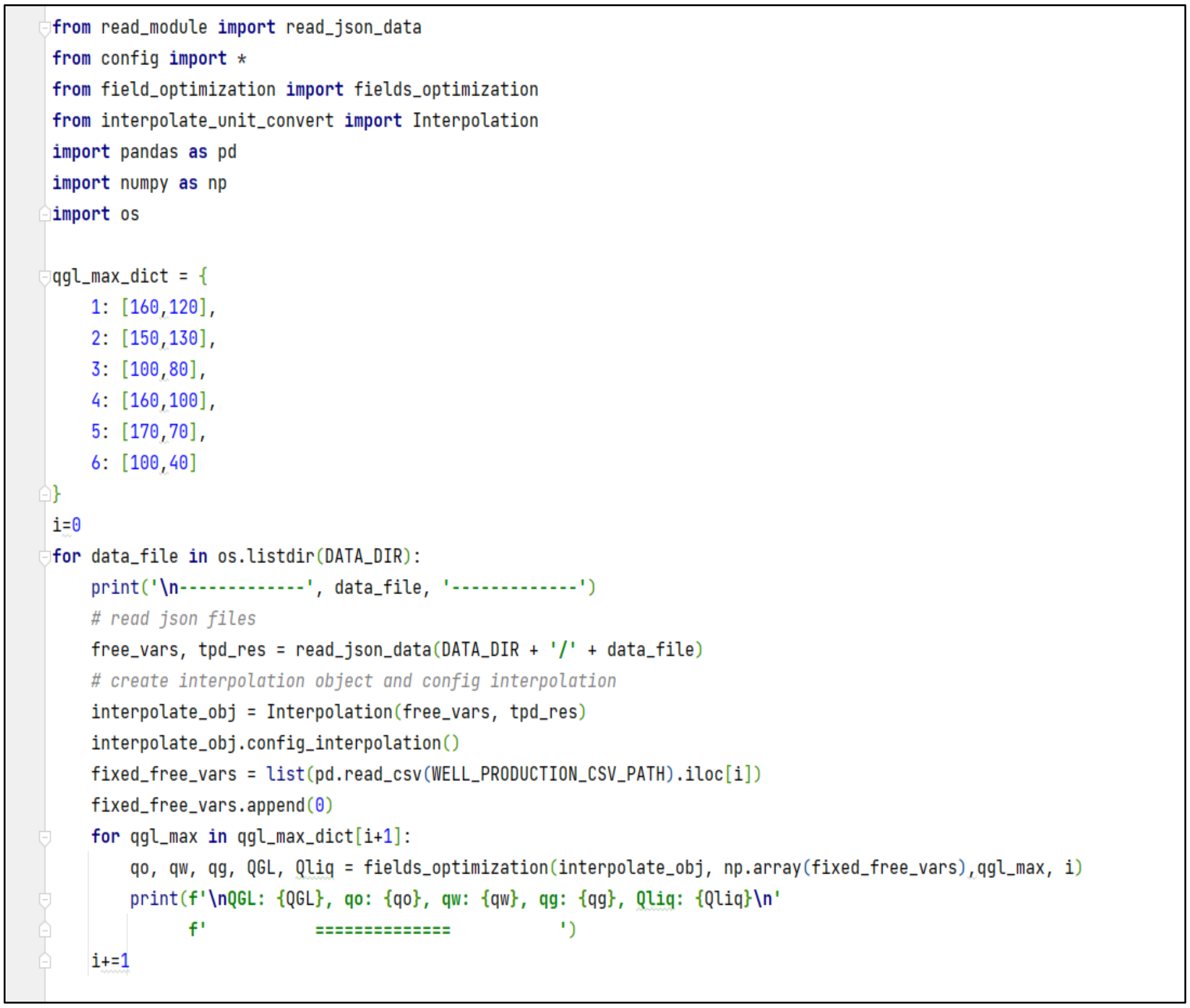
```
DATA_DIR = 'wells data'  # directory of json files
INTERPOLATION_CSV_PATH = 'Data_for_Interpolation.csv'
WELL_PRODUCTION_CSV_PATH = 'Data_for_well_production.csv'
INTERPOLATE_METHOD = "cubic"  # cubic,pchip,linear

J = [19.77, 19.73, 19.73, 3.53, 3.53, 3.53]  # Sm3/day/bar
PR = [238.882, 256.119, 256.119, 209.649, 209.649, 209.649]  # bara
PB = [242.33, 242.33, 242.33, 152.698, 152.698, 152.698]  #
VOGEL_EQUATION = [True, False, False, False, False, False]  # vogel or composite
Q_MAX = [2356.40,2615.34,2615.34,503.27,503.27,503.27]
Q_initial = [1000, 1000, 1000, 100, 100, 100]
TOTAL_QGL = 1408.69
# --------------------------->>>> method for minimize
# Nelder-Mead, L-BFGS-B, TNC, SLSQP, Powell, trust-constr, COBYLA
WELL_PRODUCTION_METHOD = 'Nelder-Mead'
CALCULATE_FIELDS_METHOD = 'Nelder-Mead'
```

B.8 WC Test Case



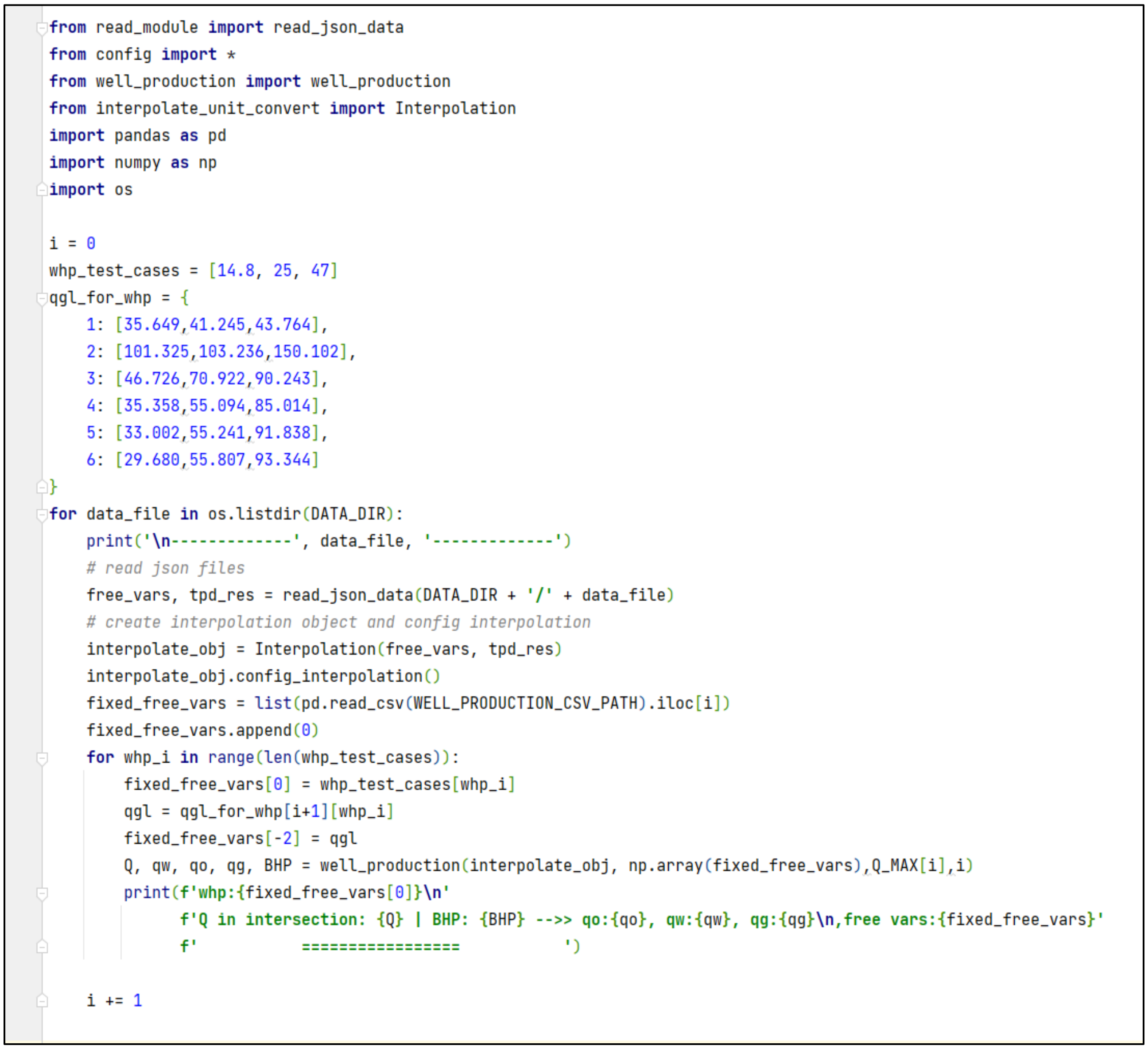
```
from read_module import read_json_data
from config import *
from well_production import well_production
from interpolate_unit_convert import Interpolation
import pandas as pd
import numpy as np
import os

i = 0
wc_test_cases = {
    1: [60,40],
    2: [30,15],
    3: [35,25],
    4: [9,7],
    5: [10,6],
    6: [8,2]
}
qgl_for_wc = {
    1: [55.95,65.82],
    2: [92.28,78.56],
    3: [44.8,32.45],
    4: [36.17,36.99],
    5: [33.36,35.006],
    6: [29.75,30.28]
}

for data_file in os.listdir(DATA_DIR):
    print('\n-------------', data_file, '-------------')
    # read json files
    free_vars, tpd_res = read_json_data(DATA_DIR + '/' + data_file)
    # create interpolation object and config interpolation
    interpolate_obj = Interpolation(free_vars, tpd_res)
    interpolate_obj.config_interpolation()
    fixed_free_vars = list(pd.read_csv(WELL_PRODUCTION_CSV_PATH).iloc[i])
    fixed_free_vars.append(0)
    for wc_i in range(len(wc_test_cases[i + 1])):
        wc = wc_test_cases[i + 1][wc_i]
        fixed_free_vars[2] = wc
        qgl = qgl_for_wc[i + 1][wc_i]
        fixed_free_vars[-2] = qgl
        Q, qw, qo, qg, BHP = well_production(interpolate_obj, np.array(fixed_free_vars), Q_MAX[i], i)
        print(f'wc:{wc},qgl:{qgl}, Q in intersection: {Q} | BHP: {BHP} -->> qo:{qo}, qw:{qw}, qg:{qg}\n'
              f'           =================           ')
    i += 1
```

B.9 QGL Test Case



```python
from read_module import read_json_data
from config import *
from field_optimization import fields_optimization
from interpolate_unit_convert import Interpolation
import pandas as pd
import numpy as np
import os

qgl_max_dict = {
    1: [160,120],
    2: [150,130],
    3: [100,80],
    4: [160,100],
    5: [170,70],
    6: [100,40]
}
i=0
for data_file in os.listdir(DATA_DIR):
    print('\n-------------', data_file, '-------------')
    # read json files
    free_vars, tpd_res = read_json_data(DATA_DIR + '/' + data_file)
    # create interpolation object and config interpolation
    interpolate_obj = Interpolation(free_vars, tpd_res)
    interpolate_obj.config_interpolation()
    fixed_free_vars = list(pd.read_csv(WELL_PRODUCTION_CSV_PATH).iloc[i])
    fixed_free_vars.append(0)
    for qgl_max in qgl_max_dict[i+1]:
        qo, qw, qg, QGL, Qliq = fields_optimization(interpolate_obj, np.array(fixed_free_vars),qgl_max, i)
        print(f'\nQGL: {QGL}, qo: {qo}, qw: {qw}, qg: {qg}, Qliq: {Qliq}\n'
              f'           ==============           ')
    i+=1
```

B.10 WHP Test Case



```python
from read_module import read_json_data
from config import *
from well_production import well_production
from interpolate_unit_convert import Interpolation
import pandas as pd
import numpy as np
import os

i = 0
whp_test_cases = [14.8, 25, 47]
qgl_for_whp = {
    1: [35.649,41.245,43.764],
    2: [101.325,103.236,150.102],
    3: [46.726,70.922,90.243],
    4: [35.358,55.094,85.014],
    5: [33.002,55.241,91.838],
    6: [29.680,55.807,93.344]
}
for data_file in os.listdir(DATA_DIR):
    print('\n-------------', data_file, '-------------')
    # read json files
    free_vars, tpd_res = read_json_data(DATA_DIR + '/' + data_file)
    # create interpolation object and config interpolation
    interpolate_obj = Interpolation(free_vars, tpd_res)
    interpolate_obj.config_interpolation()
    fixed_free_vars = list(pd.read_csv(WELL_PRODUCTION_CSV_PATH).iloc[i])
    fixed_free_vars.append(0)
    for whp_i in range(len(whp_test_cases)):
        fixed_free_vars[0] = whp_test_cases[whp_i]
        qgl = qgl_for_whp[i+1][whp_i]
        fixed_free_vars[-2] = qgl
        Q, qw, qo, qg, BHP = well_production(interpolate_obj, np.array(fixed_free_vars),Q_MAX[i],i)
        print(f'whp:{fixed_free_vars[0]}\n'
              f'Q in intersection: {Q} | BHP: {BHP} -->> qo:{qo}, qw:{qw}, qg:{qg}\n,free vars:{fixed_free_vars}'
              f'          =================           ')

    i += 1
```