TECHNICAL UNIVERSITY OF CRETE
MINERAL RESOURCES ENGINEERING

**Diploma Thesis**

# Numerical study of optimal oil recovery conditions from black oil reservoirs using the MRST reservoir simulator

**Konstantina Kolipetsa & Lirois - Leonidas Tzumakaris**

SUPERVISOR: ANDREAS GIOTIS

EXAMINATIONS BOARD MENBERS:

VAROUCHAKIS EMMANOUIL

KARATZAS GEORGIOS

CHANIA, 2025

# Acknowlegments

We would like to express our sincere gratitude to our families for their unwavering support and encouragement throughout the duration of our academic journey. Our deepest thanks go to our supervisor, Professor Andreas Giotis, for his valuable guidance, insightful feedback, and excellent collaboration during the development of our thesis. We would also like to acknowledge Dr. Andreas Pavlidis for his support and direction in the early stages of this work, which played a significant role in shaping its foundation and orientation. Furthermore, we extend our heartfelt appreciation to certified engineer Ms. Sofia Nerantzaki for her generous contribution and the considerable time she devoted to assisting us. Lastly, we are grateful to our friends and to everyone who, in their own unique way, contributed to the successful completion of this thesis.

# Abstract

This thesis explores the field of hydrocarbon reservoir mechanics, focusing on oil recovery techniques through reservoir simulation. The main objective is to investigate the capabilities of the MATLAB Reservoir Simulation Toolbox (MRST) in modeling and analyzing oil production scenarios.

The study begins with an examination of the SPE1 benchmark problem, which serves as a standard reference for validating reservoir simulation models. In this context, different production parameters were modified to observe their influence on production performance. Subsequently, new simulation codes were developed specifically for the numerical investigation of the Water Alternating Gas (WAG) method, enabling a more customized study of multiphase flow behavior, pressure and saturation dynamics.

Furthermore, the thesis compares the efficiency of various secondary oil recovery methods, including water injection, gas injection and the WAG technique. Simulation results indicated that the WAG method offers the highest oil recovery rates, combining the benefits of water and gas injection while reducing inefficiencies such as trapped oil, early gas or water breakthrough and uneven displacement of oil.

In conclusion, our study demonstrates the versatility and efficiency of MRST in reservoir modeling and simulation. It highlights the WAG method as a particularly promising strategy for maximizing hydrocarbon recovery, provided it is carefully calibrated based on reservoir-specific geological conditions.

*NOTE: The code snippets presented in this thesis can be integrated to form a complete code, which can be executed by any interested reader by simply assembling them within the MRST software. This provides the ability to run simulations and reproduce the study's results in an easy and direct manner.*

# *Contents*

# *Table of figures*

# Chapter 1: Reservoir Engineering Basics

## 1.1 Hydrocarbons

### 1.1.1 Definition of Hydrocarbons

Hydrocarbons are chemical compounds that contain carbon (C) and hydrogen (H) atoms. These compounds have the ability to oxidize rapidly, meaning they can burn and produce heat. Their formation is attributed to a process of transformation and decomposition of animal and plant organic materials that are trapped in sedimentary rocks. The smallest compound observed is methane ($CH_4$), which consists of one carbon and four hydrogen atoms. Hydrocarbons can consist of hundreds or thousands of atoms bonded together in various ways. For this reason, different types of hydrocarbons are formed, each with distinct physical properties.

In general, lighter hydrocarbons are more volatile and combustible, while heavier hydrocarbons are less volatile and have different properties. These characteristics influence their application in various fields, from fuels to industrial raw materials. Specifically, small-sized hydrocarbons, exist in a gaseous form, while those of medium size are in a liquid form under standard conditions of pressure and temperature. In contrast, large-sized hydrocarbons are semi-solid or solid. Unsaturated hydrocarbons are more likely to be solid compared to their saturated counterparts, such as cyclic hydrocarbons. It can be concluded that hydrocarbons are classified as gaseous, liquid or solid depending on the complexity of their molecules.

Liquid hydrocarbons, like crude oil, naturally consist of various combinations of carbon and hydrogen, with more than six carbon atoms. These are conventional fuels derived from plant and animal organisms. In the category of solid hydrocarbons, oil shale and bituminous sands play a significant role. These materials can be processed to yield products similar to those obtained during the distillation of crude oil. Regarding gaseous hydrocarbons, natural gas stands out, as it can be used without any prior processing. It is a conventional energy source mainly composed of methane. In addition to methane, natural gas also contains small amounts of other liquid and gaseous hydrocarbons. In some cases, after undergoing reforming processing, it is distributed through urban networks, replacing town gas (e.g., coal gas), which was typically produced from coal, lignite peat, or firewood.

### 1.1.2 Types of Hydrocarbon Reservoirs

In general, a reservoir refers to a natural concentration of valuable minerals, which, due to its size and composition, either has or may have economic potential for exploitation and is assessed in terms of its extent and content. Conversely, if a natural concentration cannot be economically exploited, either due to its size or composition, it is characterized as a mere occurrence.

Hydrocarbon reservoirs are primarily categorized into dry gas reservoirs, wet gas reservoirs, gas condensate reservoirs, volatile oil reservoirs, and black oil reservoirs. Dry gas reservoirs are mixtures of light hydrocarbons that do not produce liquid condensate (oil) during production, either in the reservoir or at the surface, unlike wet gas reservoirs where liquid condensate is collected at the surface. In gas condensate reservoirs, during the initial discovery of the formation and the early stages of production, only the gaseous phase generally flows in the reservoir pores. However, when the gaseous phase undergoes retrograde condensation

during production, and as pressure drops, it produces a highly volatile liquid condensate. Volatile oil reservoirs consist of liquid mixtures characterized by a significant content of light and intermediate weight hydrocarbons that can easily evaporate. Finally, black oil reservoirs are poor in volatile components and, until recently, represented the majority of oil reservoirs due to their occurrence at shallower depths. (Hu, 2017)

## 1.2 Oil Reservoirs

### 1.2.1 Oil

Petroleum is a complex mixture primarily composed of hydrocarbons and other components that, in addition to carbon and hydrogen, contain nitrogen, oxygen, sulfur, and metal atoms, leading to significant variations in its composition depending on its origin. Similarly, natural gas, while simpler in composition as it consists mainly of methane and ethane, also possesses its own complexity. Due to the vast diversity observed in the composition and, consequently, in the physical and chemical properties of hydrocarbons, reservoirs are classified into various types according to the characteristics of the fluids that fill the pores of their respective reservoirs.

### 1.2.2 Oil System Formation

As stated by Leslie B. Magoon & Edward A. Beaumont:

*A petroleum system encompasses a pod of active source rock and all genetically related oil and gas accumulations. It includes all the geologic elements and processes that are essential if an oil and gas accumulation is to exist.* (Magoon & Beaumont, 1994)

The five necessary elements, without which a sedimentary basin cannot be characterized as a 'petroleum system' are the source rock, reservoir rock, cap rock (seal), trap structure, and the timing of petroleum migration.

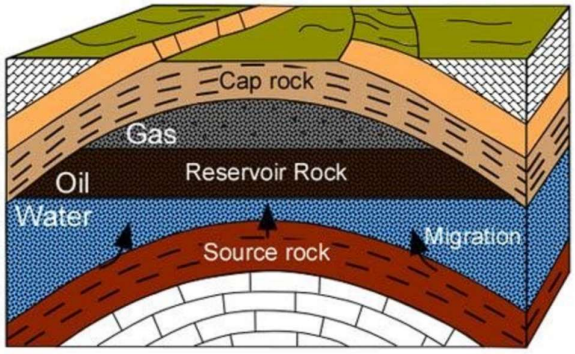


**Figure 1: Hydrocarbon Reservoir System. (https://petgeo.weebly.com/petrophysics.html)**

### 1.2.2.1 Source rock

A source rock is a sedimentary rock that contains sufficient organic matter, which, during sedimentation, as it is buried in deeper layers and undergoes heating, will lead to the formation of petroleum. There are certain environments that form very good source rocks; specifically, areas where sediments with high productivity in organic matter accumulate, and stagnant water shows high concentrations of organic matter. Environments that meet these conditions include nutrient-rich coastal areas, shallow waters, lakes, deltaic environments mainly in warm zones, and continental margins. However, high productivity in organic matter must be combined with low oxygen content or a lack of oxygen in the sediment deposition area. These low or zero-oxygen conditions can be created either in environments where water circulation leads to stagnation or by the overproduction of organic matter. (Bett, 2018)

The different nature of organic matter also leads to the formation of different types of petroleum. The two main components of organic matter are kerogen and bitumen. Kerogen represents the insoluble part of the organic matter in sedimentary rocks and is essentially a mixture of high molecular weight chemical compounds. In contrast, bitumen constitutes the soluble part of the organic matter. It contains nitrogen (N), sulfur (S), and oxygen (O) compounds, indigenous aromatic and aliphatic elements, as well as some hydrocarbons that migrated from elsewhere. Most of the organic matter is found in the form of kerogen (85–90% in shales), which consists of different types. Heating it to the appropriate temperature will release either petroleum (60–160°C), known as the oil window, or natural gas (150–200°C), known as the gas window. Since kerogen is composed of a mixture of organic materials and its composition varies in different samples, it is inferred that the breakdown of kerogen will yield different types of hydrocarbons. Based on this parameter, heavy hydrocarbons (heavy oils) result from the breakdown of unstable kerogen, while light hydrocarbons, such as natural gas, result from the breakdown of stable kerogen. (Awang, 2017)

### 1.2.2.2 Reservoir Rock

Reservoir rock is a porous, permeable, sedimentary rock in which hydrocarbons are stored after their migration from the source rock where they were formed. Their concentration occurs within the porous rocks, where fluid circulation is possible. Thus, hydrocarbons, along with formation water, accumulate in the pores of the rock, between the grains of the rock. Rocks that exhibit the necessary properties to serve as hydrocarbon reservoirs are usually sandstones and carbonate rocks (limestones and dolomites). Their size, shape, and internal properties (porosity, permeability) vary. Depending on the size of the grains, they are categorized as conglomerates, which have very large grains, sandstones or sands, which are formed from small, cemented grains, and siltstones or claystones, which consist of smaller grains. (Alamooti & Malekabadi, 2018)

### 1.2.2.3 Cap Rock (Seal)

For the formation of a hydrocarbon reservoir, the presence of a rock that functions as a seal is essential, creating a top barrier above the reservoir rock to prevent the natural, buoyancy-driven flow of hydrocarbons towards the surface. Rocks suitable for this purpose are fine-grained, polycrystalline rocks characterized by low permeability, such as clays and shales, salt

layers, and generally any rock that is sufficiently tight, like well-cemented limestones. Also, sealing can occur along fault zones and fault fronts. The presence of seal rocks in petroleum systems is crucial for the accumulation and formation of hydrocarbon reservoirs. Without them, hydrocarbons would continue their migration to the Earth's surface, where chemical processes under surface conditions, including bacterial activity, would eventually lead to the natural decomposition of the hydrocarbons.

#### 1.2.2.4 Trap Structure

The hydrocarbon trap structure is the combined geometric configuration of a permeable rock, like a reservoir rock, and an impermeable rock, like a cap rock. When these two types of rocks coexist and combine with the physio-chemical properties of subsurface fluids, they can lead to the accumulation of hydrocarbons and the formation of corresponding reservoirs. In the case of petroleum, as it is lighter and rises towards the surface through sedimentary rocks, it will not be trapped even if there are sealing rocks, unless they are characterized by an upward concave geometry. This means that the petroleum will continue its flow along the base of the seal until it reaches its edge and will then continue to the surface. As a result, it will not be able to accumulate to form a reservoir. Therefore, it is necessary for the seal rock to have a downward concave geometry in order to serve for the accumulation of hydrocarbon fluids.

Hydrocarbon trap structures can be categorized into structural traps and stratigraphic traps. Structural traps are formed from tectonic, gravitational, and diapiric processes. In the category of structural traps are anticlines, salt domes, diapiric structures, and folds. Stratigraphic traps are discontinuities within the basin fill or structures that existed from the primary depositional morphology.

#### 1.2.2.5 Timing and Migration

The term hydrocarbon migration refers to the slow but continuous upward flow of hydrocarbons from the source rocks where they emanate. Since hydrocarbons (oil and natural gas) are lighter than water, they tend to move towards the Earth's surface. Hydrocarbon migration can be either primary or secondary. Primary migration refers to the expulsion of hydrocarbons from the source rock where they are first formed. In contrast, secondary migration refers to the further upward flow of hydrocarbons in the subsurface until they reach the reservoir rock and the relevant structural trap. The permeability of each formation affects the velocity of hydrocarbon migration, as natural gas molecules are smaller and move faster than oil molecules. As hydrocarbons migrate upwards, they may encounter natural obstacles, such as impermeable rocks. In this case, their migration continues through fractures in the rock or by passing through neighboring permeable rocks. There are also quantities of hydrocarbons that remain in the source rock pores, as they cannot move upwards or dissolve in the formation water through which they pass—a phenomenon known as migration losses.

Considering the migration process, it becomes apparent that the presence of a sealing rock with the appropriate geometry to create a hydrocarbon trap structure is not sufficient. The timing of the trap's development also plays a crucial role. The timing of petroleum migration must be correlated with the deposition of the reservoir and seal, as well as the formation of structures within the sedimentary basin. This means that if hydrocarbon migration occurs before

the deposition of a suitable combination of reservoir and seal or before the formation of the appropriate trap geometries within the basin, the hydrocarbons will not be trapped.

As previously mentioned, regarding grain size, these sedimentary rocks can be conglomerates, which are characterized by very large grains, sandstones, which are characterized by small grains formed from the cementation of sand, and siltstones or claystones, that consist of smaller grains. A rock with a satisfactory level of porosity and appropriate permeability to store oil and natural gas can serve as a reservoir. However, there are other factors that play a crucial role in determining whether the exploitation of a hydrocarbon reservoir can be considered economically viable. Besides porosity, important parameters include permeability, volume, the amount and type of the accumulated hydrocarbons. The mentioned parameters belong to the petrophysical properties of reservoir rocks, which are important during the various stages of hydrocarbon exploration and exploitation. Petrophysical properties of reservoirs are estimated through measurements conducted through exploratory wells using standard logging techniques, core samples and seismic measurements, combined with the geological and geophysical structure of the area. In general, petrophysical properties include lithological composition, porosity, water saturation, permeability, density, electrical conductivity, thermal conductivity, natural radioactivity of formations, the natural NMR phenomenon, and solid mechanics. However, the basic petrophysical parameters of reservoirs are lithological composition, porosity, permeability, and saturation in water, oil, and natural gas. (Rajput & Pathak, 2025)

### 1.2.3 Lithology

Rocks that function as hydrocarbon reservoirs are categorized into two main types based on their lithology: clastic and carbonate rocks. Clastic rocks include sandstones and conglomerates, while carbonate rocks include limestones and dolomites. Among clastic rocks, more than 60% of rocks serving as reservoirs for giant fields worldwide are sandstones. Sandstones are sediments (sands) where the size of the clastic components ranges from 2 mm to 1/16 mm. After sedimentation, the loose sediment (sand) transforms into the corresponding rock (sandstone). Sediments characterized by clastic component sizes greater than 2 mm are called gravels (rounded fragments) or breccias (angular fragments). The rocks formed by diagenesis are respectively conglomerates and breccias. (Niazi et al., 2019)

### 1.2.4 Spatial Distribution of Fluids within the Reservoir

Hydrocarbon reservoirs generally contain three fluid phases with different chemical compositions that dynamically change during exploitation: a) the oil phase, b) the gas phase, and c) the water phase. Their initial spatial distribution at the start of production exploitation results from a combination of physical-chemical and geochemical processes that led to the formation of the reservoir over geological times. The primary factor is gravity, which typically leads to phase separation into distinct layers based on their specific gravity. Thus, in saturated reservoirs, the gas phase is initially located just below the impermeable geological structure trapping the reservoir underground, while directly below it is the liquid hydrocarbon phase. At the bottom of the reservoir is the water phase, which also contributes a significant portion of its natural energy. It should be noted that remnants of the initial water phase (before the accumulation and separation of hydrocarbons in the reservoir) are always present within the liquid and gas phases due to capillary phenomena preventing its complete removal from the rock during the migration phase of hydrocarbons to the reservoir.

### 1.2.5 Oil

The history of modern industrial society is closely tied to oil. Today, oil is the most widely used energy raw material and, consequently, has become an essential requirement for the functioning and development of the economy. It is established as the primary energy resource both because it has a high energy content compared to other conventional fuels and because it is in liquid phase under environmental conditions, making it easy to transport, handle, and store. Therefore, hydrocarbons (both oil and natural gas) are the primary energy source today and for the foreseeable future, as long as there is no sufficient and reliable alternative energy source available at a competitive cost. Their production on the surface, given that they are always under pressure and at great depth, is achieved through drilling. (Pathak, 2021)

### 1.2.6 Crude Oil

Crude oil historically refers to all hydrocarbons that have formed underground through natural geochemical processes. It consists of a mixture of hydrocarbons, largely alkanes (paraffins), naphthenes, aromatic hydrocarbons, and other non-hydrocarbon compounds ($H_2S$, sulfur compounds, nitrogen compounds, oxygen compounds). Crude oil is found in porous rocks within the upper layers of the Earth's crust. It originates from the diagenetic transformation of organic matter, primarily microscopic marine organisms such as algae, and zooplankton, which thrived in ancient marine environments. These organisms assimilated solar energy via photosynthesis. Their remains settled in anoxic sedimentary basins, such as ocean floors or riverbeds, where they were buried under layers of sediments (e.g., sand, mud). Specifically, the formation of crude oil appears to be negligible at temperatures below 150°F (65°C) and reaches its maximum value within the range of 225°F to 350°F (107°C to 176°C). This temperature range within which crude oil forms is known as the oil window.

### 1.2.7 Natural Gas

Natural gas is a gaseous mixture of saturated hydrocarbons containing a small number of carbon atoms. The primary component found in natural gas is methane ($CH_4$), although there are also quantities of ethane ($C_2H_6$), propane ($C_3H_8$), butane ($C_4H_{10}$), as well as carbon dioxide ($CO_2$), nitrogen ($N_2$), hydrogen ($H_2$), helium (He), and hydrogen sulfide ($H_2S$). Natural gas that contains no hydrocarbons other than methane, i.e., pure methane, is commonly referred to as dry natural gas. In contrast, natural gas containing other hydrocarbons besides methane is called wet natural gas. A key characteristic distinguishing the two is whether or not condensate forms at the surface during production. (Speight, 2017)

Regarding its properties, natural gas is colorless, invisible, and odorless, belonging to the second family of gaseous fuels. The combustion of natural gas, compared to oil, has less harmful environmental impacts. Natural gas serves as both a fuel and a raw material for the chemical industry. It is found under high pressure and forms similarly to oil. Its transportation to various areas does not require further processing, as natural gas fields are usually far from major consumption centers. However, chemical processing industries are usually located in the production area. The transportation of natural gas, if in gaseous form, is done through pipelines under high pressure, while if in liquid form, it is transported by ships.

Geographically, the production of natural gas differs from that of oil, as the Middle East produces only 10% of natural gas while holding one-third of global oil production. About 26% of natural gas is currently produced from offshore fields. In general, future increases in

natural gas production are expected globally, except in Europe, where the producing fields are nearing depletion.

### 1.2.8 Reservoir Evaluation

The use of natural oil and gas reserves is found in energy production through combustion, petrochemical material manufacturing, road construction, etc. From the above, it can be inferred that the role of hydrocarbons is crucial both internationally and locally. Specifically, in Greece, oil is still the primary fuel in the country's energy mix. According to the BP Energy Outlook 2024, petroleum products are still predominantly used in transportation; however, their share is expected to significantly decrease by 2050 due to the rise of electrification and alternative fuels. At the same time, natural gas continues to be extensively used for electricity generation, particularly in emerging economies, although a gradual decline in its demand is projected in scenarios of transition towards cleaner energy sources. (BP Energy Outlook, 2024)

After discovering a hydrocarbon field (following surface, geological, and geophysical surveys), it is necessary to collect all required data (through development drilling and reservoir delineation) to characterize both the fluid and the rock hosting it in the reservoir. The results of all these investigations will be considered and evaluated to design the optimal exploitation plan. Before approving the expenditure of substantial capital for the development of a field, the volume of hydrocarbons within the field and the percentage expected to be ultimately recovered according to the management plan are estimated. The number of production wells, their locations, and the optimal placement of perforations are decided, combining cost minimization with maximization of recovery. The production rates for each well are determined, and the specifications for the surface processing facilities are set. Finally, mathematical models in simulation programs such as MRST are employed to predict production and fluid movement at any given time within the porous medium of the reservoir.

### 1.2.9 Production of Reservoir Fluids

The production of hydrocarbons from a reservoir to the surface requires energy expenditure, which can either be derived from reservoir system itself or supplied externally. Characteristic scenarios include the expansion of pressurized reservoir fluids and the maintenance of reservoir pressure above bubble point through secondary oil production methods, such as water injection. Based on the source of energy used for production, we can distinguish the following stages:

#### 1.2.9.1 Primary Production

Primary recovery is the initial stage in the extraction of hydrocarbons, relying on natural reservoir energy. This stage relies on the natural pressure difference between the reservoir fluids and the wellbore to bring hydrocarbons to the surface. Physical driving processes like gas drive, water drive, or gravity drainage facilitate this process. The aim during primary production is to maintain operations as long as possible, as this method is the most cost-effective. Energy is derived from the reservoir-fluid system or, occasionally, from an associated water reservoir.

## Natural Drive Mechanisms

In hydrocarbon reservoirs, production is influenced by various natural drive mechanisms. These include the Solution Gas Drive, where gas dissolved in the oil helps push hydrocarbons to the production well; the Overlying Gas Zone, which can exert pressure from above; Natural Water Influx, where water naturally moves into the reservoir to displace oil; and Rock Compressibility, where the rock's ability to compress impacts the reservoir's dynamics. Often, multiple mechanisms work simultaneously to drive production, each contributing differently to the overall extraction process.

## Solution Gas Drive Mechanism

A reservoir with a solution gas drive mechanism relies primarily on the expansion of the oil phase and the release of gas dissolved in the oil to drive the oil towards the production well. During production from reservoirs of this type, two stages can be distinguished. In the first stage, the pressure is maintained above the saturation pressure of the oil ($P > Psat$), and no free gas phase exists within the reservoir's pores. In this scenario, all the gas produced at the surface is dissolved in the liquid phase under reservoir conditions. The second stage occurs when the formation pressure drops below the fluid's saturation pressure ($P < Psat$). At this point, gas is released and begins to coexist as a free phase within the reservoir's pores.

In cases where the dominant production mechanism is dissolved gas, the primary goal for reservoir engineers is to minimize gas production to preserve as much available energy in the reservoir for oil production. This is because gas compressibility is much higher than that of liquids (oil and water). The recoverability of reservoirs using the solution gas drive mechanism mainly depends on the fluid's PVT properties. When no other production mechanisms are in effect, the final recoverability of the reservoir typically does not exceed 30% of the initial hydrocarbon volume in the porous medium, with an average of around 15%. For reservoirs containing very heavy oil (black oil) the ultimate recovery rate may be less than 10%. (Rajput & Pathak, 2025)

## Gas Cap Drive Mechanism

In a reservoir with a gas cap drive mechanism, the theoretical principles are similar to those of the solution gas drive. As the reservoir pressure decreases due to fluid extraction, the overlying gas cap expands and maintains sufficient pressure on the oil, providing additional energy from the gas expansion. The interface between the gas and oil phases continuously moves downward because the liquid is saturated and cannot dissolve additional gas, leading to an increasing gas phase saturation as formation pressure drops and fluids are produced at the surface. Generally, the presence of a gas cap reduces the rate of pressure decline compared to a reservoir using only a dissolved gas drive. Special attention is required in the early stages of reservoir exploitation regarding the placement of production wells and perforations, as the formation already contains a second (gas) phase under initial pressure and temperature conditions. The primary goal for petroleum engineers is to correctly position the wells and perforations to achieve maximum oil production at the surface. Ideally, perforations should be located within the oil zone. Reservoirs with a gas cap drive mechanism typically exhibit a 1-10% higher recoverability compared to those producing solely with a solution gas drive.

### Water Drive Mechanism

Another natural drive mechanism involves natural water influx from an underlying aquifer. The impact of natural water influx on a reservoir's behavior depends largely on the size of the underlying aquifer. In reservoirs with a strong natural water influx mechanism, pressure remains relatively stable throughout the reservoir's life. However, if water production at the surface starts to increase significantly due to water breakthrough, immediate measures must be implemented to safeguard the viability of the extraction process. In some cases, a significant water column in the well can exert hydrostatic pressure on the formation sufficient to halt production.

### Compaction and Expansion Drive Mechanism

Another natural drive mechanism is due to the compressibility of both the porous medium and the water trapped within its pores. Fluid production from the reservoir leads to a gradual decrease in reservoir pressure due to the occurring expansion. This pressure reduction causes a change in the pore volume, resulting in the shrinkage of the formation's pores because of rock compressibility, known as compaction drive. Since hydrocarbon reservoirs contain water that cannot be removed, this leads to water expansion and the expulsion of oil from the pores. This mechanism contributes minimally to production compared to the above mechanisms when the rock compressibility is significantly lower than the fluid compressibility. (Pathak, 2021)

### Complex Drive Mechanism

When a reservoir produces under the influence of more than one drive mechanism, it is referred to as using a complex drive mechanism. The contribution of each mechanism to production is not constant but changes over time. If multiple drive mechanisms contribute simultaneously, the overall performance of the reservoir will depend mainly on the choice of well placements and the production rate from each. For example, in a reservoir influenced by both a gas cap drive and an underlying aquifer, the reservoir's performance will vary significantly depending on whether wells penetrate the lower or upper part of the formation or if perforations are ideally placed between the oil/gas and oil/water interfaces. Typically, reservoir simulation models are recommended for studying the behavior of formations with complex drive mechanisms (e.g., MRST). (Rajput & Pathak, 2025)

Single-phase flow refers to the movement of a single fluid phase through the rock matrix. In this context, while other phases, such as water at connate water saturation, may be present within the reservoir, only the primary fluid—typically oil or gas—is actively flowing through the rock. This scenario simplifies the modeling and analysis of reservoir behavior, as the dynamics of additional phases do not significantly influence the movement of the primary fluid. (Martin, 1996)

For example, the flow of single-phase oil in unsaturated oil reservoirs is straightforward to model, allowing for accurate predictions of reservoir productivity and characteristics. Single-phase flow models often assume that the fluid properties, such as viscosity and density, remain constant, which further simplifies the calculations. This simplification is particularly useful in initial reservoir assessments and early stages of development, where it helps in estimating the potential production rates and designing appropriate extraction techniques. (2023 International Conference on Energy Engineering, 2024)

Moreover, single-phase flow modeling provides a foundation for understanding more complex multi-phase flow scenarios. By mastering the principles of single-phase flow, engineers can better interpret how additional phases, such as water or gas, will interact with the primary fluid as they become significant in later stages of reservoir exploitation. This fundamental understanding is crucial for designing efficient recovery strategies and optimizing reservoir management practices.

In contrast, multiphase flow involves the simultaneous movement of two or more phases, such as oil, water, and gas, through the rock matrix. This type of flow is inherently more complex due to the intricate interactions between the different phases and their mutual effects on each other's movement. These interactions include phenomena such as capillary pressure, which affects how fluids are distributed within the porous medium.

The behavior of multiphase flow is significantly influenced by surface properties of the rock and fluids, including interfacial tension, wettability, and relative permeability. Interfacial tension dictates the extent to which different fluid phases interact at their boundaries, while wettability describes the affinity of the rock surface for different fluids, impacting fluid distribution and flow patterns. Relative permeability reflects how the presence of one fluid phase affects the flow of another, complicating the prediction of each phase's movement through the rock.

Furthermore, multiphase flow modeling must account for the dynamic changes in fluid saturation over time, which can lead to complex displacement processes, such as water or gas breakthrough. These phenomena can alter the flow paths and impact overall reservoir performance. Understanding and accurately modeling these parameters is crucial for optimizing extraction strategies, enhancing recovery factors, and managing reservoirs effectively. This knowledge allows engineers to design more efficient recovery methods and make informed decisions regarding reservoir development and management. (Rajput & Pathak, 2025)

### 1.2.9.2 Secondary Production

When primary recovery becomes less efficient due to pressure depletion in the reservoir, secondary recovery techniques are employed in order to maintain the pressure of the reservoir, typically above the bubble point. These methods involve introducing external energy to the system to enhance hydrocarbon recovery. Common techniques include water injection (waterflooding) and gas injection, which help push hydrocarbons toward the production wells. Secondary production maintains reservoir pressure and displaces oil or gas towards the wells, contributing to ultimate recovery in a cost-effective manner. This stage typically recovers about 15% to 40% of the original hydrocarbons in place.

#### Waterflooding

The maintenance of reservoir pressure through water injection (waterflooding) has historically been the first method of secondary oil recovery. It was first implemented in 1907 in the Bradford field in Pennsylvania, where it achieved great success due to the low viscosity of the reservoir oil, the low initial gas saturation, and the absence of an adjacent water reservoir. As a result, primary production was insignificant, and the application of this technique led to a significant oil recovery factor (Bradley & Gipson, 1987).

Today, this method is the most commonly used technique for oil production worldwide, often in combination with artificial lift techniques in production wells. It is mainly applied in

unsaturated reservoirs and gas condensate reservoirs with good horizontal hydraulic conductivity.

The water tends to accumulate in the lower layers of the reservoir due to its higher density, thus pushing the oil toward the production wells and forming a nearly flat interface between the two phases. The surface pressure is regulated to prevent exceeding the rock's fracture limit, and the process is completed when water reaches the production wells (water-cut).

### Gas Flooding

In saturated reservoirs with an overlying gas cap, gas is typically injected into the gas zone to maintain sufficient production pressure in the underlying oil layer. This method often relies on the recycling of gases produced from the reservoir, such as natural gas, nitrogen ($N_2$), and carbon dioxide ($CO_2$). These gases are partially soluble in oil and initially flow as a separate gas phase, primarily occupying the larger pores and fractures in the rock. As a result, the gas front often bypasses significant amounts of oil trapped in smaller pores, leading to a low recovery factor.

The dissolution of gases in the oil, when conditions are favorable, affects the physicochemical properties of the oil, such as viscosity, density, and the dew and bubble points. The reduction in viscosity increases the mobility of oil and consequently enhances production, especially when gas injection is alternated with water injection (Water Alternating Gas, WAG).

## 1.2.9.3 Tertiary Production

Tertiary recovery, also known as enhanced oil recovery (EOR), involves advanced methods to extract additional hydrocarbons. This stage includes:

#### Thermal Recovery

Thermal recovery is an advanced enhanced oil recovery (EOR) technique designed to extract heavy or viscous oils from challenging reservoirs by reducing their viscosity through heating. The primary method is steam injection, where steam is injected into the reservoir to lower the oil’s viscosity and enhance its flow. There are two main approaches: continuous steam injection and cyclic steam stimulation (huff and puff), each tailored to maintain reservoir temperature and improve oil mobility. (Alamooti & Malekabadi, 2018)

Another technique, in-situ combustion, involves igniting a portion of the oil within the reservoir to generate heat, reducing viscosity and increasing pressure to facilitate oil flow. Hot water injection, although less effective, can be used when steam generation is impractical or to complement other EOR methods.

Thermal recovery can significantly boost oil recovery by improving flow characteristics and maintaining reservoir pressure. However, it comes with high costs and energy demands, making its economic viability a concern. Effective reservoir management is crucial to avoid issues like steam breakthrough. Despite these challenges, thermal recovery is essential for maximizing the potential of heavy oil resources.

**Gas Injection**

Gas injection is an advanced enhanced oil recovery (EOR) technique that uses gases like carbon dioxide ($CO_2$) and nitrogen to improve oil extraction from reservoirs, especially after primary and secondary recovery methods. This method enhances the recovery factor by reducing oil viscosity, improving flow and maintaining reservoir pressure. (Olukoga, 2024)

Carbon dioxide injection is a widely used approach where $CO_2$ is injected into the reservoir to mix with or dissolve in the oil. This process reduces the oil's viscosity, making it easier to mobilize under a fixed pressure gradient. This injection can operate through miscible displacement, when $CO_2$ fully integrates with the oil, or immiscible displacement, where it creates a pressure gradient to displace the oil towards production wells.

Nitrogen injection is another important method, where nitrogen is used as a displacement fluid to displace oil towards production wells. Unlike $CO_2$, nitrogen does not dissolve in oil but helps maintain reservoir pressure and is often more abundant and cost-effective.

The benefits of gas injection include significantly increasing oil recovery, maintaining reservoir pressure and improving oil flow efficiency. However, challenges such as the high cost of sourcing and injecting gases, limited availability of $CO_2$ and the need for specialized infrastructure can impact the feasibility of this method. Effective reservoir management is crucial to avoid issues like gas breakthrough (gas-cut), where the injected gas reaches production wells prematurely.

In summary, gas injection is a vital EOR method that enhances oil recovery by using gases like nitrogen and $CO_2$ to reduce oil viscosity and maintain reservoir pressure. Despite the associated costs and challenges, it plays a key role in maximizing oil recovery and extending the productive life of oil fields.

**Chemical Injection**

Chemical injection is an advanced enhanced oil recovery (EOR) method that uses chemicals like polymers and surfactants to improve oil extraction. Polymers increase the viscosity of injected water, thus improving sweep efficiency by reducing water bypass and enhancing oil displacement. Surfactants reduce surface and interfacial tensions, allowing water to better penetrate and mobilize trapped oil, leading to higher recovery rates. Alkaline agents can be combined with surfactants to generate in-situ surfactants, further improving oil mobilization.

While chemical injection significantly enhances oil recovery by improving waterflooding efficiency, it is complex and costly. Its effectiveness depends on reservoir conditions, requiring careful characterization and monitoring. Despite these challenges, chemical injection remains a crucial tool for optimizing oil extraction and maximizing reservoir potential. (Das et al., 2025)

### 1.2.10 Porosity

Porosity ($\phi$) is a fundamental property that allows a reservoir to store oil, natural gas, and water and essentially represents the reservoir's storage capacity. There are different types of porosity, such as total (absolute) porosity and effective porosity, which refers to the percentage of interconnected pores. Total porosity is the ratio of the total pore volume ($V_b$) to

the bulk volume of the rock ($V_t$) and is expressed as a percentage or fraction of the rock's total volume. Effective porosity describes the ratio of interconnected pores to the total rock volume.

$$\phi\ (\%) = \frac{V_b}{V_t}$$

Within reservoir rocks, due to their complex internal structure, isolated pores may exist that, although contributing to the total porosity, do not connect with the main pore network. This results in isolated pores that do not participate in fluid flow within the reservoir. Effective porosity, which supports fluid flow and essentially represents the permeability, typically has a lower value than total porosity but is more significant for hydrocarbons as it represents the space that contains recoverable fluids.

Another important distinction in porosity is between primary and secondary porosity. Primary porosity refers to the voids present in the rock during its deposition, while secondary porosity forms after sediment deposition due to processes such as compaction and diagenesis. Secondary porosity includes features like recrystallization and fracturing. Sedimentary rocks usually have porosity values between 5-25%. Excellent reservoir rocks have porosity values above 25%, while those with 15-25% are considered good, 5-15% as satisfactory, and below 5% as low porosity.

Additionally, porosity depends on the sorting of the rock grains, with well-sorted grains leading to better porosity. Conversely, poorly sorted grains, where smaller grains fill spaces between larger grains, negatively impact porosity. (Ανδρέας Γιώτης, Δημήτρης Μαρινάκης, Μηχανική Κοιτασμάτων Υδρογονανθράκων)

### 1.2.11 Permeability

Alongside porosity, permeability (k) is also a crucial parameter for a reservoir rock. Permeability measures the ability of a fluid to flow through a porous medium and is a dynamic property of the rock (measured through flow experiments on reservoir samples), unlike porosity, which is a static property. Permeability is categorized into horizontal permeability (flow parallel to the bedding planes) and vertical permeability (flow perpendicular to the bedding planes). It is expressed by Darcy's Law:

$$q = \frac{k(P_1 - P_2)A}{\mu L}$$

Where:

- q = volumetric flow rate of the fluid ($m^3/s$)
- k = permeability of the porous medium ($m^2$)
- $P_1$ and $P_2$ = fluid pressure at the two ends of the porous medium (Pa)
- A = area of the flow ($m^2$)
- μ = dynamic viscosity of the fluid (Pa·s)
- L = length of the porous medium through which the fluid is flowing (m)

(Mahdian et al., 2020)

The non-SI unit of measurement for permeability is the Darcy (D), though the milli-darcy (mD) is more commonly used (1 D = 1000 mD).

The permeability of a reservoir depends on the effective porosity, which is influenced by grain size, shape, sorting, and the degree of cementation. Rounded or platy grains, and angular grains affect both horizontal and vertical permeability. Permeability is also affected by the type of clay and cement between sand grains, especially in the presence of fresh water. Some clays, like bentonites and montmorillonites, tend to swell in fresh water, leading to pore space blockage. In sedimentary rocks, the permeability can vary with measurement direction, with vertical permeability usually being lower than horizontal permeability. Typically, slightly elongated grains in sandstone are oriented parallel to flow directions, which can slightly increase permeability in the direction of elongation. Since a hydrocarbon reservoir involves multiphase flow (oil, gas, and water), the flow of each fluid depends on the proportions of the other fluids in the reservoir, referred to as saturation. Therefore, the relative permeability of each fluid in the system is crucial. Relative permeability is the ratio of the effective permeability of each phase to the absolute permeability. Absolute permeability is a property of the rock structure itself, not the fluids contained. For hydrocarbons, permeability values of 5-500 mD are suitable. Generally, permeability values of 100-1000 mD are considered excellent, 10-100 mD as good, 1-10 mD as satisfactory, and below 1 mD as low. (Tiab & Donaldson, 2024)

### 1.2.12 Anisotropy

Anisotropy describes a condition in which a porous medium exhibits different transport properties in different spatial directions. This directional dependence is particularly significant in geological formations, where characteristics such as permeability, formation resistivity factor, and breakthrough capillary pressure vary with direction. In an anisotropic medium, the flow of fluids, the electrical resistance, and the capillary pressure required to displace fluids are not uniform across all directions.

Geological and structural factors contribute to anisotropy in rock formations. For instance, sedimentary rocks may display higher permeability in horizontal directions due to layering and depositional processes. Similarly, the orientation of fractures, faults, and bedding planes can cause variations in rock properties depending on the direction of measurement.

In the context of reservoir engineering, anisotropy has several important implications. Permeability anisotropy, where horizontal permeability is often much greater than vertical permeability, significantly affects fluid flow patterns and extraction efficiency. Accurate modeling of these variations is crucial for designing effective reservoir management strategies and enhanced oil recovery techniques.

The formation resistivity factor, which is used in well logging and geophysical surveys, can also vary with direction. This variability must be considered to accurately interpret resistivity measurements and assess reservoir characteristics. Additionally, capillary pressure anisotropy, where the pressure required to displace fluids varies with direction, impacts fluid distribution and the effectiveness of reservoir management techniques such as waterflooding or gas injection.

Understanding anisotropy allows for enhanced reservoir modeling, providing a more accurate representation of reservoir behavior. This, in turn, improves predictions of fluid flow, reservoir performance, and the impact of various extraction strategies. By accounting for anisotropic properties, engineers can optimize extraction methods, such as tailoring horizontal drilling and well placement to the directional properties of the reservoir. Furthermore, the knowledge of anisotropy is crucial for designing effective enhanced oil recovery (EOR)

techniques. It enables the optimization of injection patterns and well placements to maximize oil or gas recovery. Recognizing and addressing anisotropy also helps in managing risks associated with fluid injection and production, such as uneven fluid distribution and premature water breakthrough.

In summary, anisotropy adds complexity to reservoir characterization and management. A thorough understanding and incorporation of these directional variations into reservoir models and extraction strategies lead to more accurate predictions, optimized recovery methods, and improved resource management.

### 1.2.13 Saturation

Saturation (S) is a fundamental petrophysical property of rocks that serve as hydrocarbon reservoirs, and it refers to the relative volume content of each phase within the rock. Essentially, saturation expresses the fraction or percentage of the pore volume occupied by a specific fluid, which can be water, oil, or gas. Practically, it represents the ratio of the total fluid volume to the pore volume. According to this perspective, each fluid in the reservoir has a distinct saturation.

Water, oil, and gas saturation determine the fluid distribution in reservoir rocks, influencing hydrocarbon extraction efficiency. High water or gas saturation can complicate recovery, requiring advanced management and EOR techniques to optimize production and maximize reservoir potential.

#### 1.2.13.4 Total Saturation

Total saturation encompasses the overall fluid content in the pore spaces of a reservoir rock, including oil, gas, and water. It provides a comprehensive view of the reservoir's fluid composition. For instance, if a reservoir rock has a total saturation of 100%, it means the entire pore space is filled with fluids, whether oil, gas, or water.

Understanding these different types of saturation is essential for evaluating the oil content of a reservoir. By analyzing each type, engineers can gauge how much oil can be extracted and the effort required for extraction. Thus, for oil, there is oil saturation ($S_o$), which expresses the volume of oil relative to the pore volume; gas saturation ($S_g$), which expresses the volume of gas relative to the pore volume; and water saturation ($S_w$), which represents the volume of water relative to the pore volume. The sum of these three saturations equals one ($S_o + S_g + S_w = 1$). It is worth noting that saturation is related to the pore volume, and that the saturation observed in each phase ranges between 0 and 100%. Ideally, because fluids have different densities, in a reservoir rock, the placement of fluids from top to bottom is as follows: gas, oil, water, with the lighter fluid moving upward.

Regarding water in the reservoir, it is distinguished into bottom-edge water and connate water. Connate water is seawater trapped within the pore spaces of sediments during deposition and lithification, long before the migration of oil into the reservoir rock. The forces that hold the water in the gas and oil zones are capillary forces. Connate water saturation ($S_{wc}$) is a key factor, as it reduces the space available between oil and gas. Generally, connate water is not distributed uniformly within the reservoir. It depends on other petrophysical properties of the reservoir, such as its lithology, permeability, and height from the free water surface.

Additionally, it is important to mention critical saturation, which applies to each fluid in the reservoir. For oil, this includes critical oil saturation ($S_{oc}$), which represents a specific

value that must be exceeded for oil flow to occur. At this saturation, oil remains in the reservoir pores and will not flow. Residual oil saturation ($S_{or}$) is also observed, referring to the amount of oil that remains in the reservoir during the process of oil displacement through the porous medium. The value of residual saturation is greater than the value of critical oil saturation ($S_{or}>S_{oc}$). The third type of oil-related saturation is movable oil saturation ($S_{om}$), which represents the ratio of the pore volume occupied by movable oil ($S_{om} = 1 – S_{wc} – S_{oc}$), where $S_{wc}$ is connate water saturation and Soc is critical oil saturation. Regarding gas, the critical gas saturation ($S_{gc}$) is identified first. In the reservoir, as the pressure drops below the bubble point ($P_b$), gas starts to separate from the oil phase, resulting in an increase in gas saturation as reservoir pressure decreases. The gas phase remains immobile until the saturation exceeds a specific value, above which gas begins to move. This specific value represents the critical gas saturation.

For water, critical water saturation ($S_{wc}$) is observed. The critical water saturation, along with connate water saturation, is extensively used in determining the maximum water saturation at which the water phase will remain immobile. (Iqbal & Kudapa, 2025)

## 1.3 Volumetric formation factors for reservoir fluids

### 1.3.1 Gas – Oil Contact

The term Gas-Oil Contact (GOC) refers to the surface or boundary within a hydrocarbon reservoir that contains both gas and oil, above which gas predominates and below which oil predominates. The area between these two phases is transitional, containing a mixture of gas and oil. (Schlumberger n.d.)

The precise location of the GOC is critical for the estimation of reserves and the management of the reservoir, as it influences production strategies and resource evaluation. The position of the GOC can be determined through geophysical measurements, density logs, resistivity logs, nuclear magnetic resonance (NMR) logging, and other methods. (Mihai & Ciuperca, 2019)

### 1.3.2 Solution Gas-Oil Ratio (Rs)

The volumetric factor of the Solution Gas-Oil Ratio, Rs, is defined as the ratio of the volume of gas that will be produced from a certain amount of oil in the reservoir when it is expanded from the reservoir conditions (Tr, Pr) to standard conditions (SC), to the volume of oil that would be produced from the same amount under standard conditions. That is:

$$R_s = \frac{V_{dg}^{sc}}{V_o^{sc}}$$

The Rs is dimensionless in SI units as a volume ratio, while in the mixed oil unit system, it is measured in (scf/stb). The Rs factor increases with the amount of dissolved gas in the oil.

As is evident in the following diagram: the increase in Rs with pressure, for values below bubble-point pressure, is due to the absorption of the overlying gas and the expansion of the liquid. After the bubble-point pressure, Rs remains constant, as no more gas is available to dissolve into the oil.

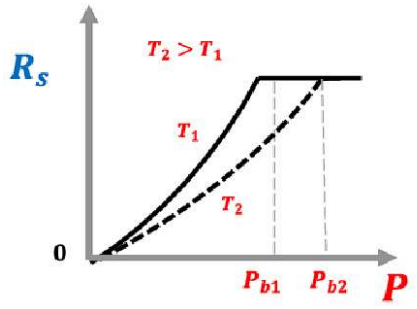


**Figure 2: An indicative change in the Rs factor with pressure at two different temperatures. (Andreas Giotis, & Marinakis, D. (2023). Μηχανική Κοιτασμάτων Υδρογονανθράκων.)**

### 1.3.3 Gas to Oil Ratio (GOR)

The term Gas to Oil Ratio (GOR) is a crucial indicator in hydrocarbon reservoir engineering, expressing the ratio of the volume of gas produced at the surface (under standard conditions — SC) to the volume of oil produced under the same conditions. It is a significant tool for evaluating the nature of a reservoir and categorizing the type of fluid it contains.

GOR can be mathematically expressed as follows:

$$\text{GOR} = \frac{\text{V}_{\text{gas}}}{\text{V}_{\text{oil}}} \ (\text{scf/stb})$$

(Alexandria Engineering Journal, 2023)

Where:

- $V_{gas}$: volume of produced gas (in standard cubic feet, scf)
- $V_{oil}$: volume of produced oil (in stock tank barrels, stb)

The value of GOR depends on two main factors:

1. Free gas present in the reservoir that reaches the surface.
2. Dissolved gas within the oil that is released as pressure decreases during production.

Analyzing GOR provides valuable insights into reservoir dynamics. A high GOR may indicate:

- The presence of large amounts of dissolved gas in the oil (high Rs).
- The flow of free gas from the upper section of the reservoir, especially if the well is perforated near the Gas Oil Contact (GOC).

Conversely, a low GOR may signal that the oil contains a small amount of dissolved gas or that drilling occurs in deeper sections, far from the gas zone. In specific reservoir categories, such as gas condensate reservoirs, GOR can range from 10,000 to 18,000 scf/stb

(equivalent to 60,000 to 100,000 cubic feet of gas per barrel of oil), indicating significant quantities of gas in the reservoir's fluid phases. Understanding and continuously monitoring GOR is vital for making informed production management decisions, as it can reveal critical changes in reservoir behavior, such as gas zone migration or gas influx into the well due to pressure drops.

### 1.3.4 Bottom Hole Pressure (BHP)

The term Bottom Hole Pressure (BHP) is the pressure at the bottom of a wellbore, where fluid (oil, gas, or water) flows from the reservoir into the well. It is a key parameter in reservoir engineering, directly influencing production rates and overall reservoir performance.

Understanding and accurately measuring the bottomhole pressure (BHP) is crucial for the proper management and exploitation of an oil or gas reservoir. Monitoring BHP allows for continuous evaluation of well productivity, as a drop in BHP can indicate reservoir depletion or increased permeability of the geological formation. Additionally, controlling BHP is essential for optimizing production, as maintaining the correct well pressure helps prevent issues such as gas or water coning in the production system. Continuous monitoring of BHP is also vital for detecting potential problems, such as wellbore blockages, gas breakout, or leaks. Overall, ongoing BHP monitoring enables effective reservoir management and maximizes hydrocarbon recovery, ensuring long-term production efficiency.

### 1.3.5 Gas Cut

The term gas cut refers to the condition where the reservoir fluid (usually oil) contains mixed gas (typically natural gas). In BHP (Bottomhole Pressure) diagrams for oil reservoirs, gas cut indicates the presence of gas within the reservoir fluid being pumped from the wellbore. This occurs when the well pressure decreases and the gas dissolves or is released into the liquid, causing the identification of gas cut.

The presence of gas cut can cause a reduction in the density of the fluid being pumped, as gas has a lower density than oil. In BHP diagrams, this change can lead to variations in the well pressure, and a sudden drop in pressure may indicate the release of gas from the reservoir fluid.

Gas cut can have several consequences, such as:

1. Reduction in oil production: The presence of gas in the fluid may reduce the amount of oil being extracted.
2. Changes in well pressure: Gas may alter the dynamic pressure of the well and lead to production losses or suboptimal well performance.
3. Requires further management: The detection of gas cut may require a reassessment of the extraction strategy and well control.

This term is used to monitor the quality of the extracted fluid and the ability to maintain production through proper management of well parameters.

# Chapter 2: Introduction to the MATLAB Reservoir Simulation Toolbox (MRST)

## 2.1 Prominent Reservoir Simulation Tools

Reservoir simulation is a critical tool in modern reservoir engineering, enabling the modeling of complex subsurface flow phenomena to optimize hydrocarbon recovery and explore porous media applications. Numerical simulators facilitate the analysis of multiphase fluid flow, pressure dynamics, and enhanced oil recovery (EOR) strategies, supporting both industrial operations and academic research. With a diverse range of simulation platforms available, each offering unique capabilities and target audiences, selecting the appropriate tool is essential for aligning with project goals and resources. This chapter provides a comparative overview of four prominent reservoir simulation tools—MATLAB Reservoir Simulation Toolbox (MRST), tNavigator, OPM Flow, and ResFrac—evaluating their strengths, limitations, and applicability. Following this comparison, we focus on MRST, which is used extensively in this thesis due to its flexibility and alignment with our academic context at the Technical University of Crete. Subsequent sections will detail MRST's installation, setup, and application in building customized simulation workflows, such as those for Water Alternating Gas (WAG) injection.

The MATLAB Reservoir Simulation Toolbox (MRST), developed by SINTEF Digital, is an open-source platform designed for rapid prototyping and academic research in porous media flow and reservoir simulation. Unlike conventional simulators with fixed workflows, MRST offers a modular framework that allows users to develop, test, and modify simulation components with significant flexibility. Built within the MATLAB environment, it leverages robust mathematical and visualization tools, making it suitable for researchers exploring novel numerical methods or complex physical models. MRST includes black-oil and compositional simulators compatible with industry-standard models, and despite its academic origins, it can handle large and complex cases. Supported by an active user community and comprehensive documentation, including tutorials and examples, MRST is widely adopted in research institutions (SINTEF, 2023, MathWorks, 2022).

In contrast, tNavigator, developed by Rock Flow Dynamics, is a commercial software emphasizing computational efficiency and integrated workflows for industrial applications. It combines static modeling, dynamic simulation, surface network optimization, and seismic interpretation within a unified platform. tNavigator's computational performance, achieved through advanced CPU and GPU parallelization, enables rapid execution of large-scale simulations. Its dynamic simulation interface supports real-time model interaction, facilitating tasks like history matching and decision-making. This makes tNavigator effective in industrial settings where time efficiency and cross-departmental integration are priorities, though its commercial licensing may limit accessibility (Rock Flow Dynamics, 2024).

OPM Flow, part of the Open Porous Media (OPM) initiative, is an open-source, fully implicit black-oil simulator focused on compatibility with industry standards, such as Eclipse input decks. It supports advanced physical models, including $CO_2$ storage, thermal simulation, and polymer flooding, and uses automatic differentiation to enable rapid development of new fluid models while ensuring numerical robustness. While less visually intuitive than commercial tools like tNavigator, OPM Flow is versatile for both academic and industrial

applications, particularly when cost and transparency are considerations. Its commitment to open science fosters collaboration and reproducibility (OPM Project, 2024).

ResFrac, a commercial tool, integrates hydraulic fracturing and reservoir simulation in a single environment, modeling the entire well life cycle, including multiphase flow, 3D fracture mechanics, proppant transport, poroelastic effects, and thermal phenomena. This comprehensive approach eliminates the need for separate fracture and reservoir models, enhancing consistency and reducing workflow complexity. ResFrac is particularly suited for unconventional reservoirs, EOR, and geothermal energy studies. Its cloud-based architecture and intuitive interface support scalability and collaboration, though its commercial nature requires licensing (ResFrac Corporation, 2024).

Comparing these tools, MRST excels in flexibility, transparency, and academic customization, making it ideal for research projects requiring rapid prototyping or novel numerical schemes. Its integration with MATLAB enhances accessibility for users familiar with the platform, though it lacks the optimized performance of commercial tools. tNavigator offers superior computational speed and operational integration, but its cost and proprietary nature may limit academic use. OPM Flow balances industrial compatibility and open access, serving as a middle ground between MRST's research focus and commercial robustness. ResFrac provides specialized capabilities for unconventional reservoirs, appealing to projects requiring detailed fracture and reservoir integration.

The decision to use MRST for this thesis is driven by practical and academic considerations aligned with our context as students at the Technical University of Crete. First, we have free access to MATLAB through institutional licenses, making MRST a cost-effective choice compared to commercial software like tNavigator or ResFrac, which require paid subscriptions. Second, our familiarity with MATLAB, developed through coursework and prior projects, reduces the learning curve and enables efficient development of customized simulation workflows. Third, MRST's modular structure allows us to implement and test custom grid structures, such as the tensor grids used in our WAG simulations, and tailor numerical methods to our research objectives. This flexibility is critical for exploring novel algorithms and physical models, such as those for multiphase flow and EOR. While tNavigator and ResFrac offer high performance and integrated workflows, their proprietary nature and cost make them less feasible for student-led research. OPM Flow, though open-source, requires familiarity with C++ and Eclipse input formats, which are less intuitive for MATLAB users. MRST's combination of accessibility, transparency, and support through a well-documented API and active research community makes it the optimal choice for this thesis, aligning with our academic goals and technical resources.

## 2.2 Installation of MATLAB

To proceed with the study and simulation of a reservoir using MRST, the installation of MATLAB is required. First, visit the official MathWorks website. In our case, MATLAB was installed through the Technical University of Crete, which provides licenses for its students. The Technical University of Crete offers the latest version of MATLAB, along with the Simulink package and all available toolboxes (link: MathWorks Products), through a new Campus Agreement subscription license. Otherwise, a personal license must be obtained at your own expense.

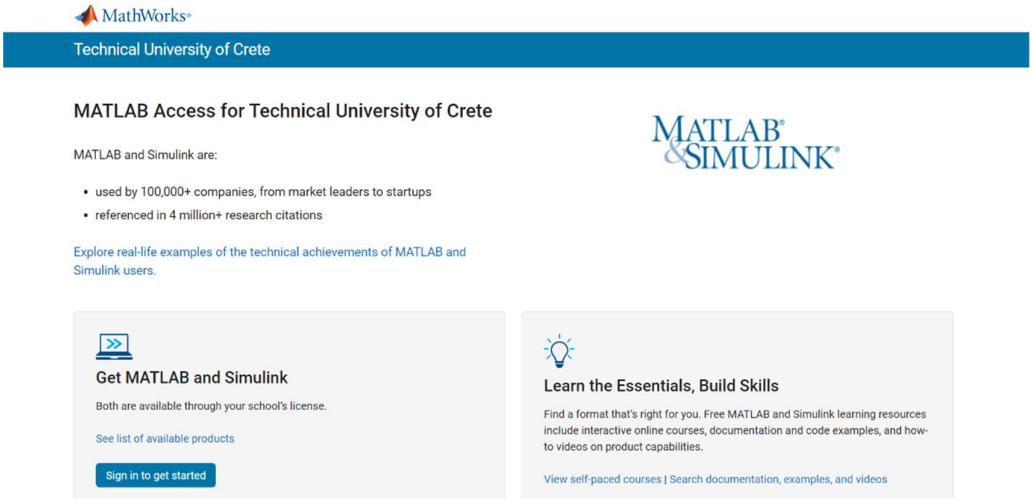


**Figure 2: The start page of MathWorks.**

After visiting the website, the first step is to log in with your account on the MathWorks Portal. In our case, we will enter our institutional email address (of the form @tuc.gr) and select '*Next*'.

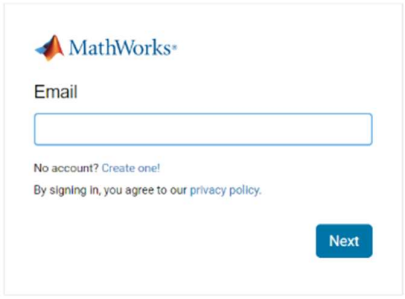


**Figure 3: Entering the email address.**

In the subsequent step, authentication through the Technical University of Crete is required. The process involves entering institutional credentials (username and password) using the institution's Single Sign-On (SSO) system. This step verifies the user's identity and confirms eligibility for software use under the institution's subscription license. After successful authentication, select '*Login*' to proceed.

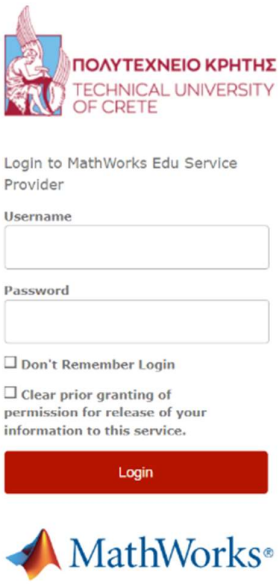


**Figure 4: Entering the institutional credentials username and password.**

Upon successful login, a confirmation message regarding the possession of the relevant license will be displayed. Select '*Get Started'* to proceed.

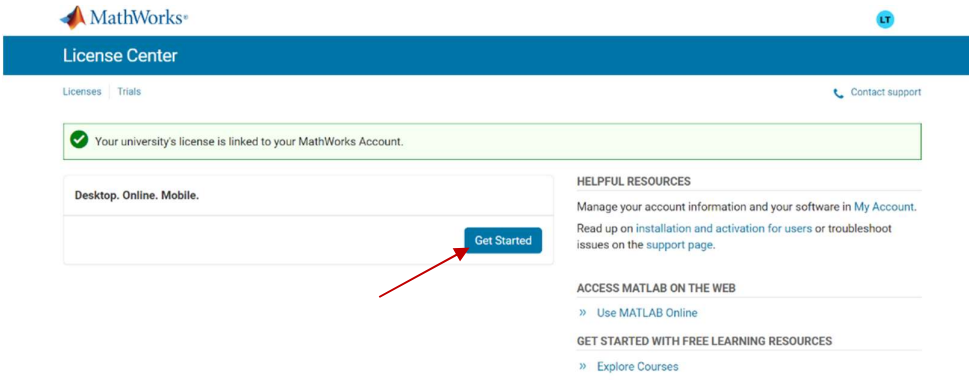


**Figure 5: The confirmation of the university's license.**

On the subsequent page, select '*Install MATLAB'* and then click on '*Download for Windows'*.

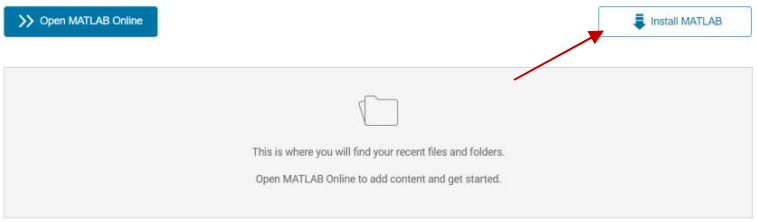


**Figure 6: Select the 'Install MATLAB' button.**

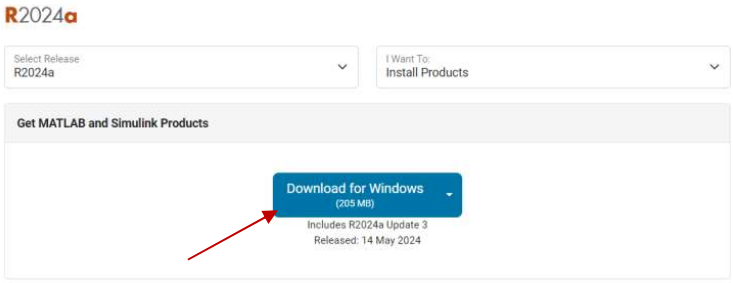


**Figure 7: Select the 'Download for Windows' button.**

The choice of the directory for installing MATLAB and subsequently MRST is deliberate. Specifically, create a folder named '*MATLAB'* in '*Local Disc (C:)' >> 'Program Files'*, as illustrated in the image below.

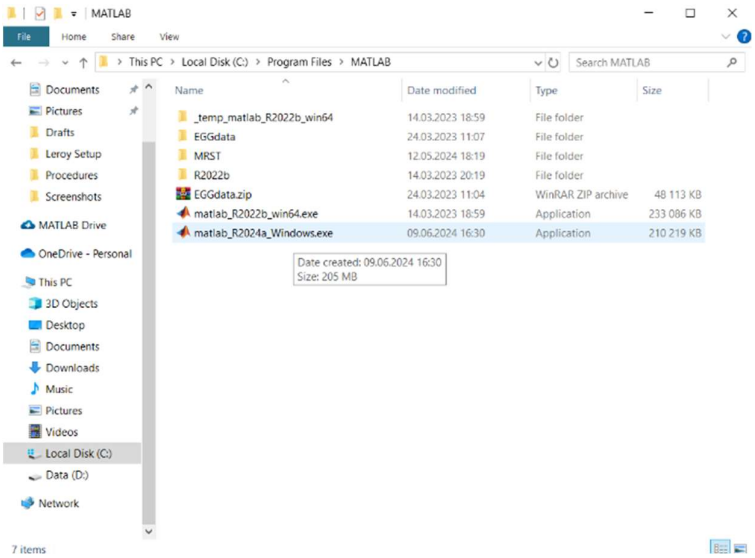


**Figure 8: The folder selection to save the installed file.**

Subsequently, in the folder we created, extract the downloaded file. The name '*R2024a*' reflects the version number of the software. The previous version we had installed, '*R2022b*', is also visible in the image above.

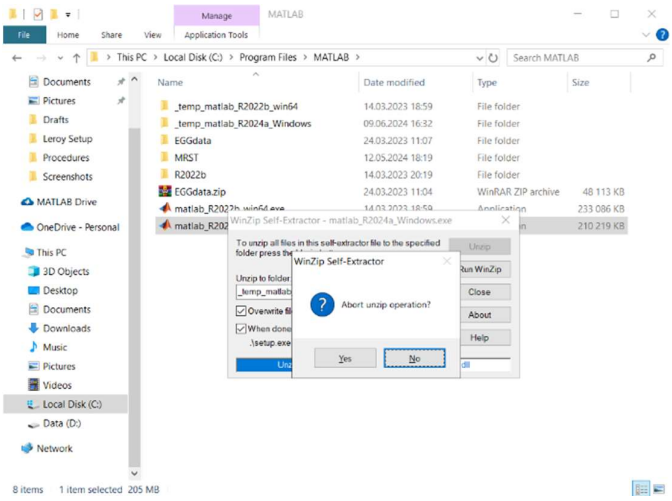


**Figure 9: The folder extraction at the existing saved folder.**

After extracting the file, proceed with the installation by signing in with the institutional email and authenticating with the Technical University of Crete, following the same process described earlier.

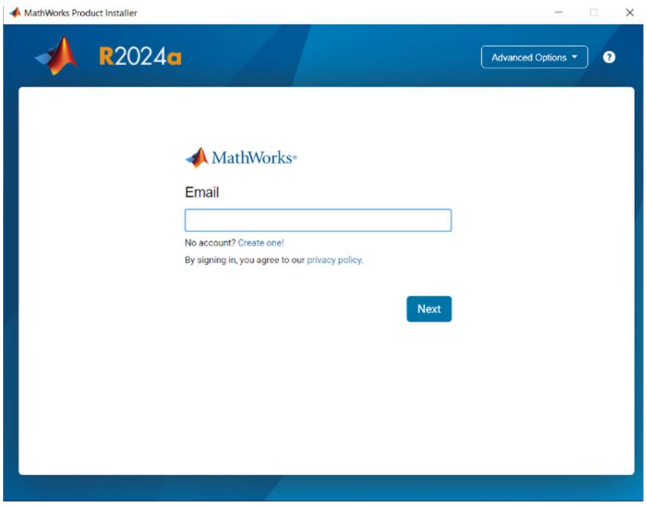


**Figure 10: Entering the email address.**

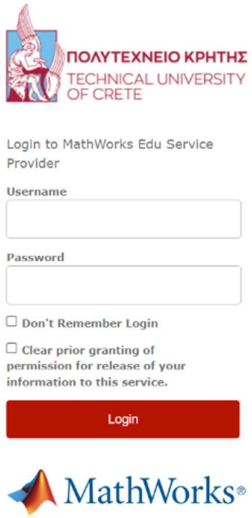


**Figure 11: Entering the institutional credentials 'username' and 'password'.**

Next, accept the terms of the license agreement by selecting '*Yes*' for the option '*Do you accept the terms of the license agreement*', and then click '*Next*' to proceed.

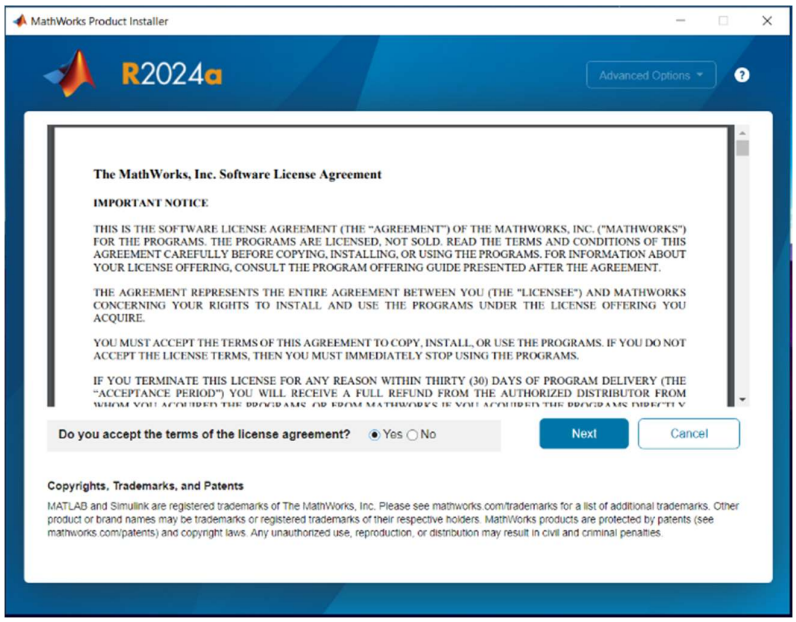


**Figure 12: Accepting the terms of the license agreement.**

In the next window, the '*LICENSING*' tab will display the type of institutional license we possess. Click '*Next*' to proceed.

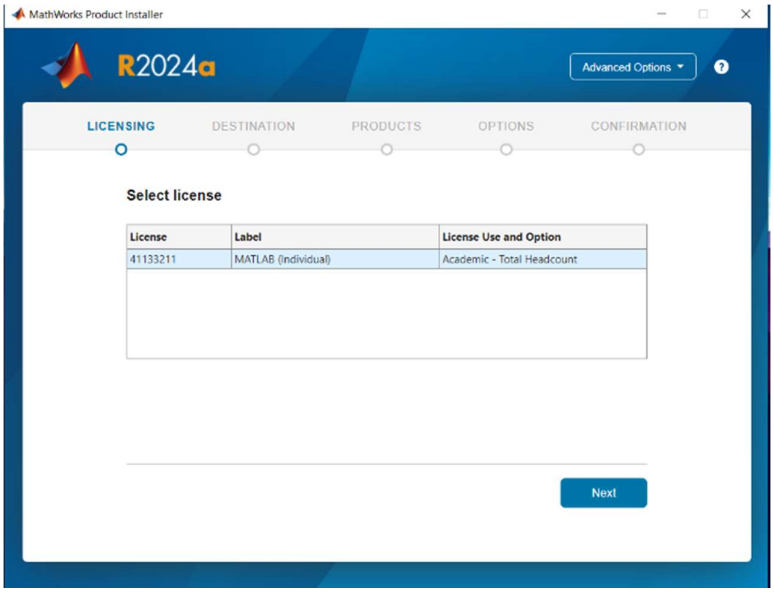


**Figure 13: Showing 'License Use and Option'.**

In the '*DESTINATION*' tab, select the installation folder that was created on the computer, and then click '*Next*' to proceed.

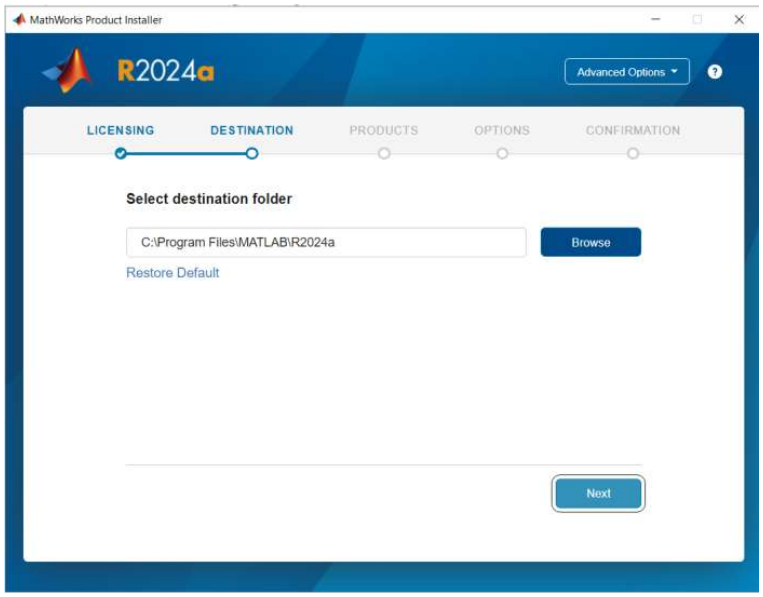


**Figure 14: Choosing the destination folder.**

In the '*PRODUCTS'* tab, select the MATLAB products and toolboxes you wish to install on your computer, and then click '*Next'* to proceed.

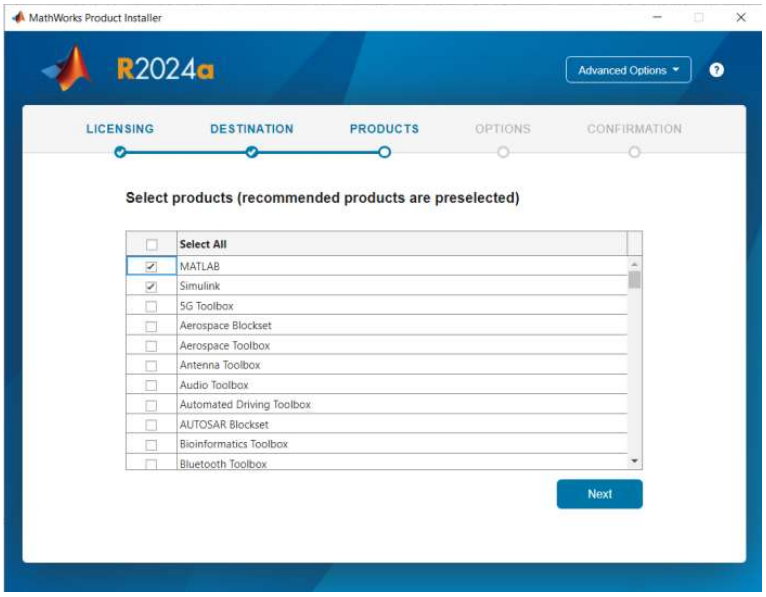


**Figure 15: Selecting products and toolboxes.**

In the '*OPTIONS*' tab, optionally choose to add a shortcut to the application on the desktop. In the '*CONFIRMATION*' tab, review and confirm your selections, then click '*Next*' to proceed.


**Figure 16: The confirmation form.**

Next, the process of downloading the necessary files and installing the software begins. Wait until the installation is completed, which may take several minutes.

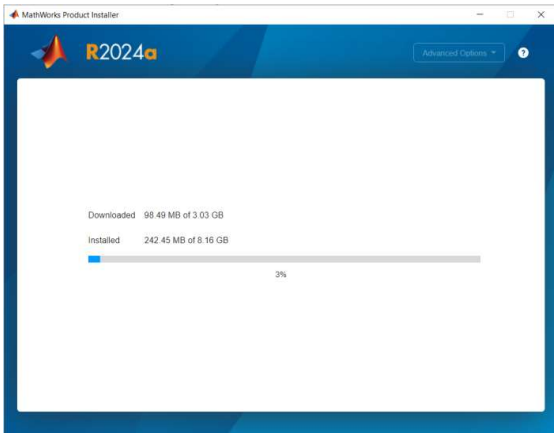


**Figure 17: Waiting until the installation is complete.**

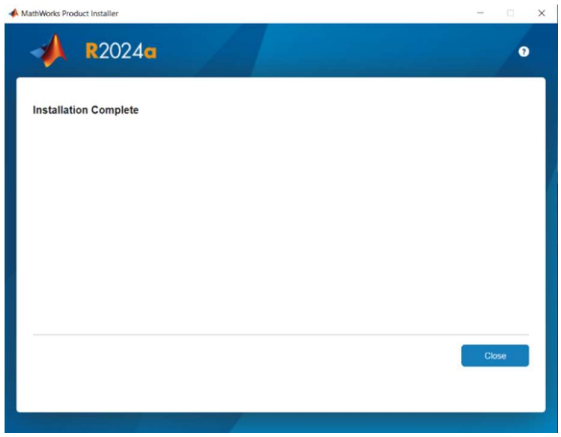


**Figure 18: Installation is completed.**

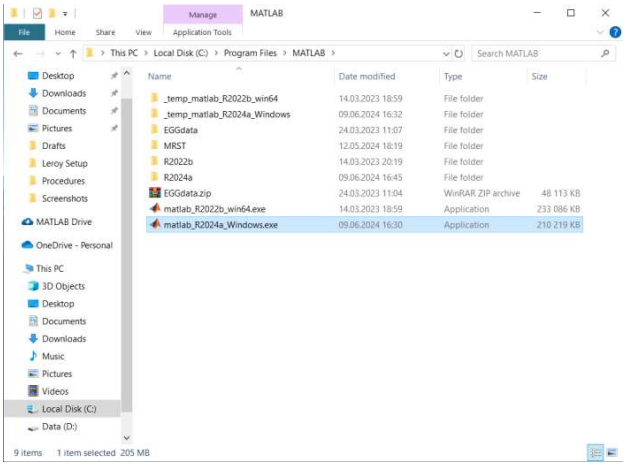


**Figure 19: The installed file appears in the default folder.**

After the installation is complete, initiate MATLAB. The software will prompt for institutional credentials to sign in. Once the sign-in process is completed, proceed to start MATLAB by selecting '*Run as administrator'*.

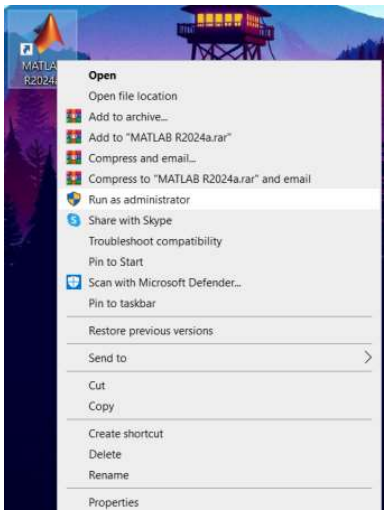


**Figure 20: Right click on the MATLAB folder and left click to 'Run as administrator'.**

The MATLAB environment consists of the following components. First, the '*Command Window*', which is the main area where users enter and execute commands or short pieces of code directly. Results from these commands are displayed immediately in this window. In addition, the '*Workspace*' shows all the variables created during the current session. It provides information such as variable names, their dimensions, and the types of data they contain. The area of '*Command History'* records all commands executed in the current and previous sessions. It allows users to easily repeat or review past commands. Another tool is *'Editor'*. This tool is used for writing, saving, and running scripts and functions. It features syntax highlighting, automatic indentation, and debugging tools to assist with code development. The '*Figures*' are graphics windows where plots and graphical representations are created and displayed. Each plot has its own window, which can contain one or more subplots. The '*Current Folder'* pane displays the directory you are currently working in and includes the files located within that folder. This facilitates navigation and selection of files for editing or execution. The '*Toolstrip*' contains tabs with tools and commands for various functions, such as managing the environment, importing data, creating graphs, and more. 'Path Browser' used for managing the folders on the MATLAB path, which are the directories MATLAB searches for scripts and functions. Through *'MATLAB Add-Ons'*, you can install additional tools, such as toolboxes, which extend MATLAB's capabilities for various applications. The MATLAB environment includes a comprehensive '*Help*' system with documentation, examples, tutorials, and search functions that assist users in better understanding MATLAB's capabilities.

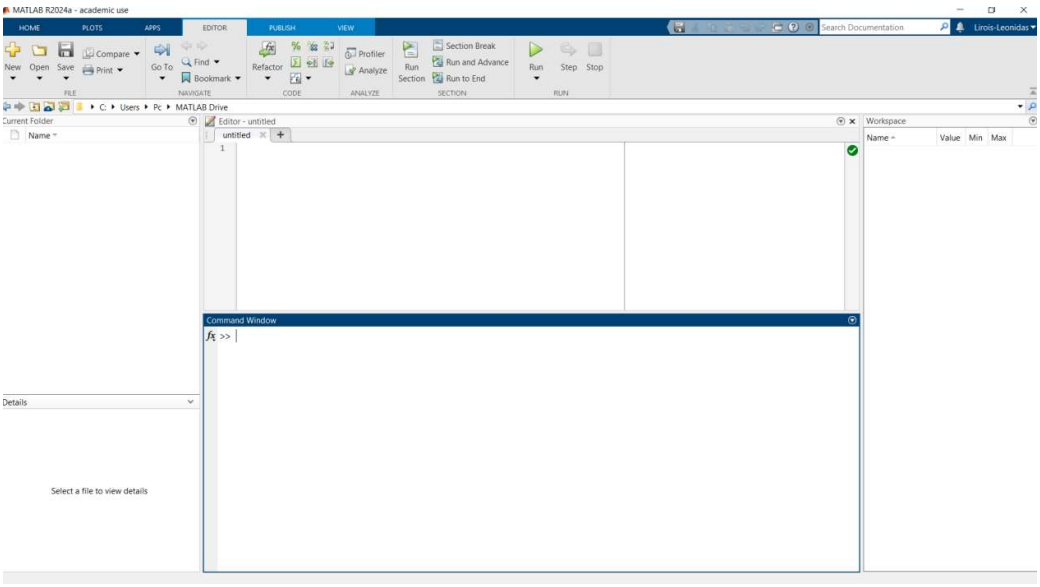


**Figure 21: MATLAB environment.**

For additional information, you can refer to the official installation guide provided by the Technical University of Crete at the following link: [Technical University of Crete MATLAB Installation Guide].

## 2.3 Installation of MRST

Following its selection for its flexibility and compatibility, this chapter guides the installation of the MATLAB Reservoir Simulation Toolbox (MRST). To install MRST, begin by visiting the official SINTEF website dedicated to MRST at [Sintef MRST Download]. From this page, you can select the option to download MRST, which will provide you with the latest version of the toolbox. This installation process is designed to be straightforward, allowing users to integrate MRST into their MATLAB environment efficiently.

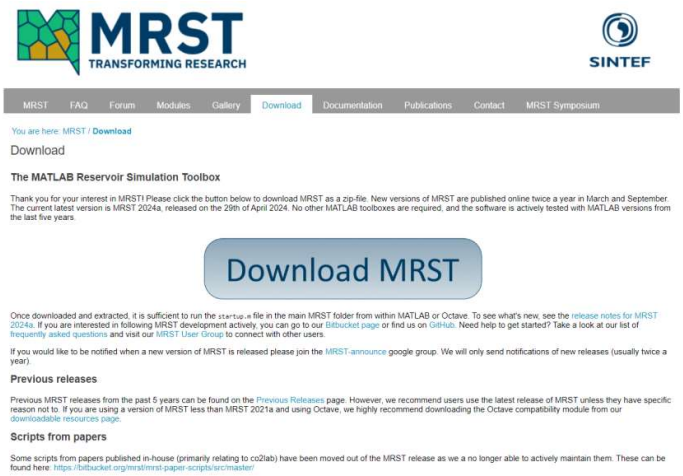


**Figure 22: The page of Sintef MRST Download.**

Subsequently, create a new directory within the previously established folder and name it '*MRST*'. Install the downloaded zip file, titled '*mrst-2024a.zip'*, into this newly created directory. Next, extract the contents of the zip file, which will produce a folder named '*mrst-2024a'*. The designation '*2024a*' is significant, as it indicates the version of the software.

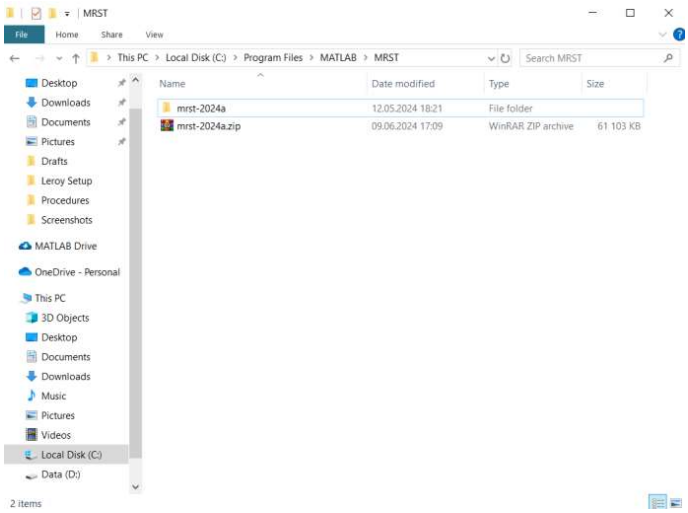


**Figure 23: The installed file of MRST.**

Next, launch the MATLAB environment and use the '*Browse for Folder'* option to upload the '*MRST*' directory that was installed by selecting '*Select Folder'*. The folder will then be visible in the '*Current Folder'* window.

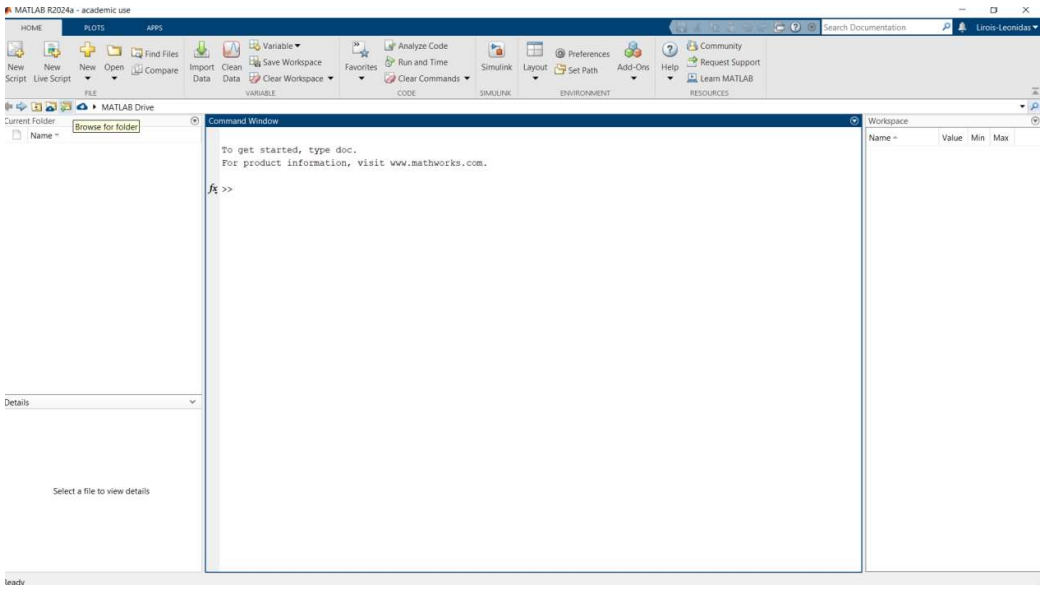


**Figure 24: Use the 'Browse for folder' to select a folder.**

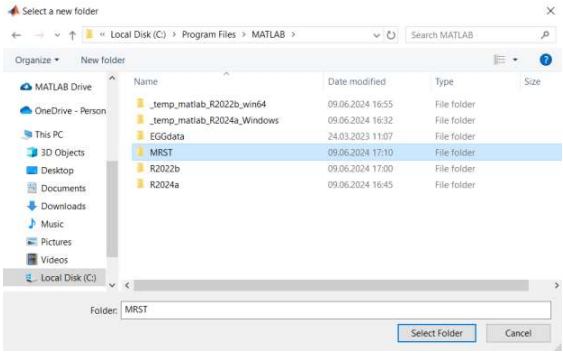


**Figure 25: Select the 'MRST' folder.**

Within the '*mrst-2024a*' directory, there is a file named '*startup.m*' which is essential for initializing MRST. Once you select and open this file in the Editor window, choose 'Run' to execute it.

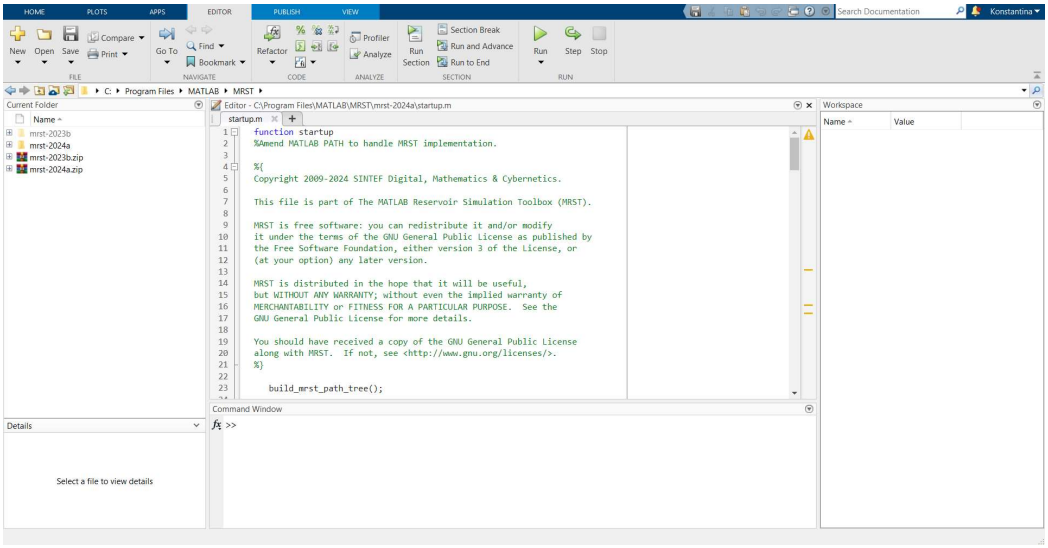


**Figure 26: Open and run the 'startup.m'.**

Executing the '*startup.m'* file initializes essential files in the MATLAB '*Command Window'*. This file plays a crucial role in configuring the MRST environment, facilitating preparation for reservoir simulations. Each time MATLAB is launched, '*startup.m'* is automatically executed to add all necessary MRST folders to the MATLAB path. This ensures that MRST functions and modules are immediately accessible without further configuration. Moreover, '*startup.m'* can be customized to activate frequently used modules, such as those related to black-oil modeling or other simulations, allowing users to start their work promptly. It may also include personalized settings, such as default measurement units or configuration of graph display parameters, making the entire process more efficient and tailored to the user's needs. Consequently, '*startup.m'* is instrumental in optimizing setup time and ensuring a smooth start to each MATLAB session, particularly for projects requiring MRST tools.

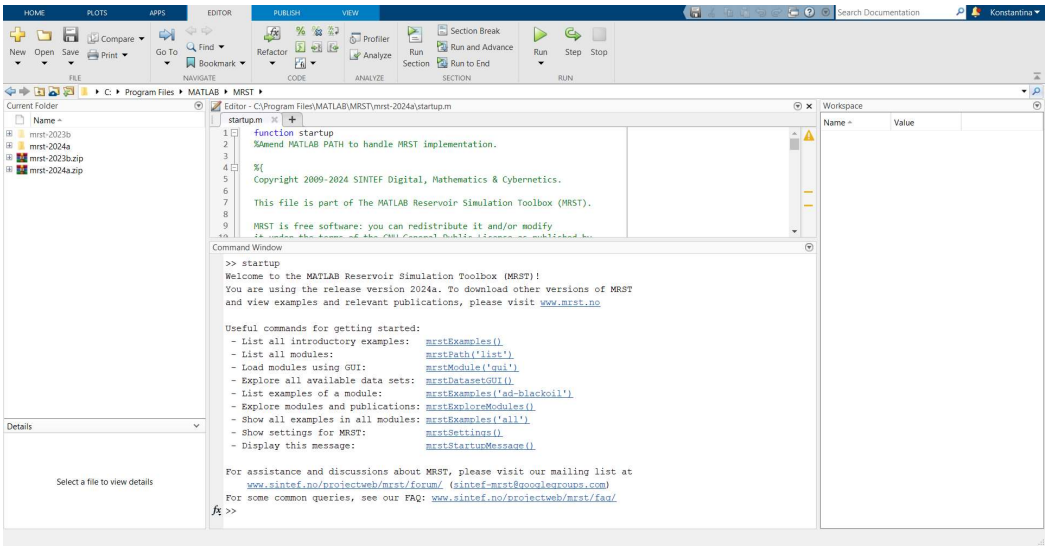


**Figure 27: The results at the 'Command Window'.**

The '*mrstModule('gui')*' function in MRST is used to manage the available MRST modules. When you use the mrstModule command, you can perform various actions, such as adding, removing, or viewing which modules are currently active.

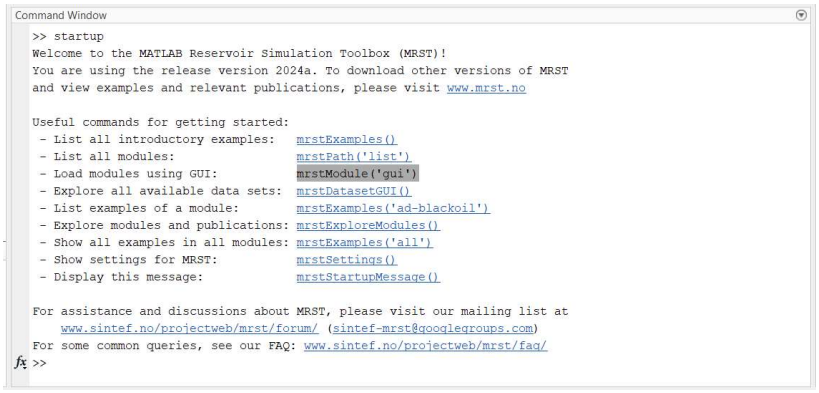


**Figure 28: Select the 'mrstModule('gui')'.**

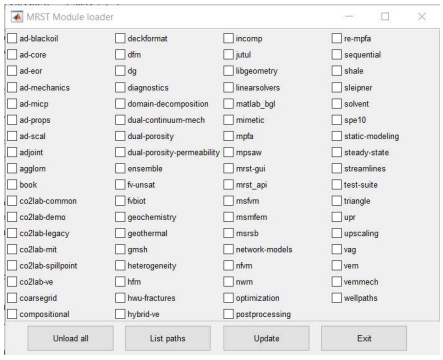


**Figure 29: The module's list.**

These details allow users to efficiently manage their MRST tools, ensuring that only the necessary modules for their work are loaded, thus preventing MATLAB from being overloaded with unnecessary functions.

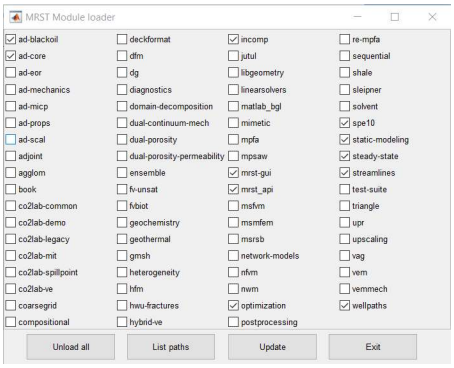


**Figure 30: Selecting modules.**

The '*mrstDatasetGUI*' is a graphical user interface within MRST designed to facilitate the loading, management, and visualization of reservoir simulation data. Through its user-friendly interface, it allows users to navigate and organize datasets without the need to write code. The GUI provides tools for visualizing grids, rock properties, phase flows, and other critical data, making the simulation process more accessible and efficient. It is an ideal tool for both novice users and experienced researchers looking to streamline their workflow.

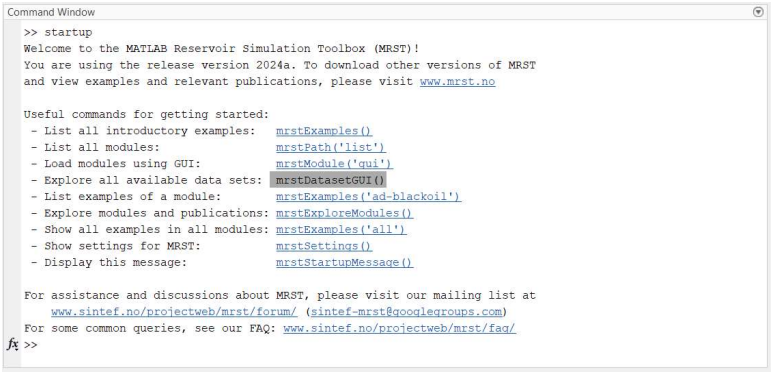


**Figure 31: Select the 'mrstDatasetGUI()' to explore all available data sets.**

Within the '*mrstDatasetGUI*' environment, select the option to download the '*SPE1*' dataset for subsequent analysis.

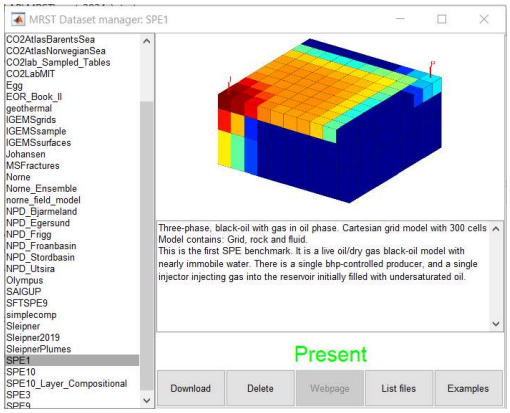


**Figure 32: Select the file 'SPE 1'.**

The '*mrstExploreModules()*'function in MRST provides a user-friendly graphical interface for exploring and managing the available modules. Through this interface, users can easily navigate through the modules, view information about their functionalities, and activate or deactivate the ones they need. This facilitates the management of dependencies between modules, ensuring that the workspace is properly configured for the required simulations. Overall, '*mrstExploreModules()'* simplifies the setup of MRST, making it more accessible and flexible for users.

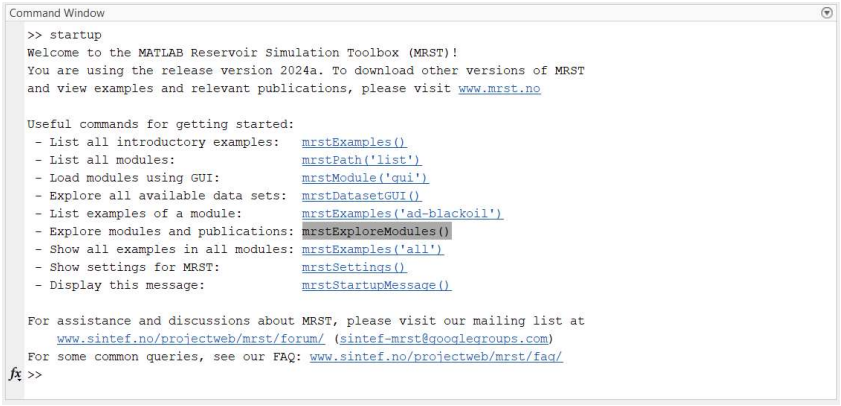


**Figure 33: Select the 'mrstExploreModules()' to explore modules and publications.**

By selecting '*mrstExploreModules()*', the following window opens, where you can choose '*ad-blackoil'>> 'spe1\blackoilTutorialSPE1.m'>> 'edit*' to download and open a file named '*blackoilTutorialSPE1.m'* in the Editor. Repeat the same process for the file '*simulateSPE1.m'*.

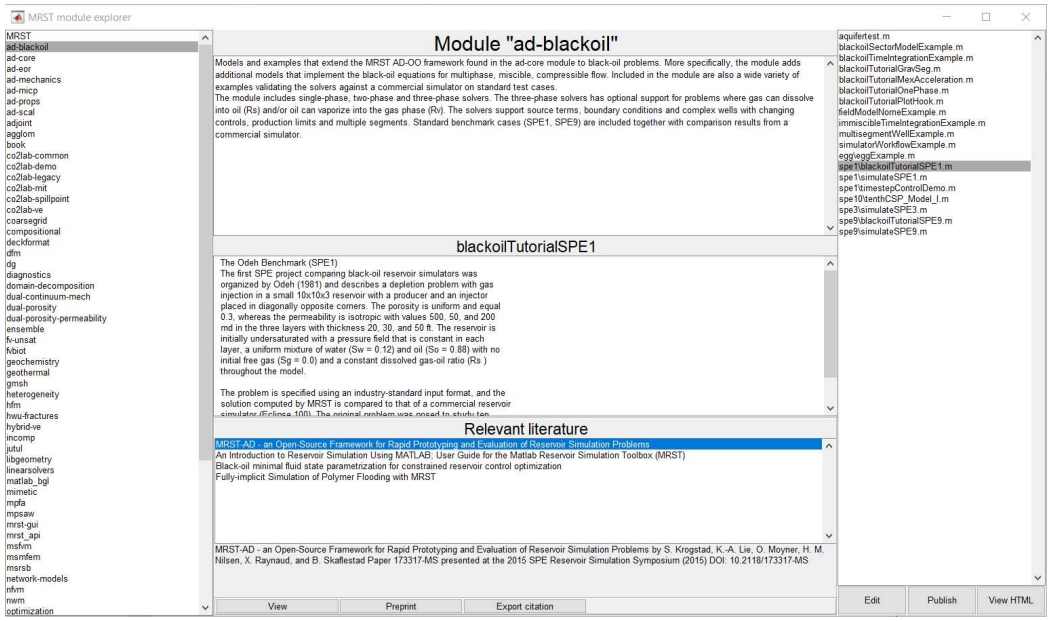


**Figure 34: The window of 'MRST module explorer'.**

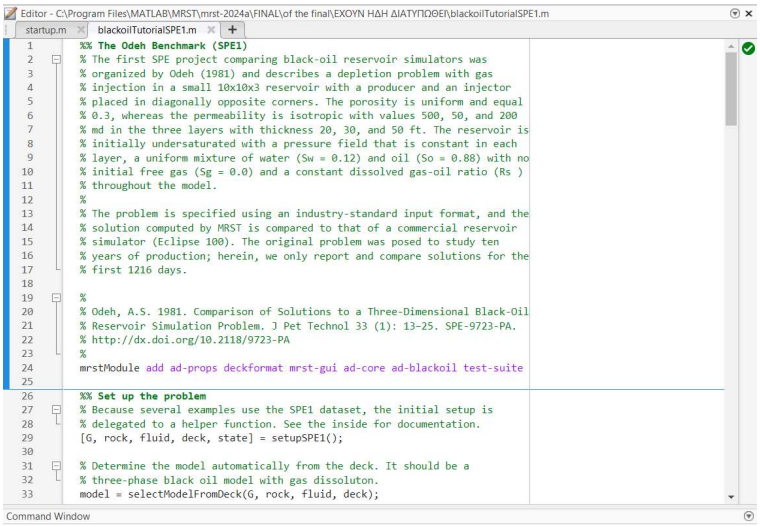


**Figure 35: Open the 'blackoilTutorialSPE1.m'.**

# Chapter 3: Introductory Codes and SPE 1 Analysis

## 3.1 Simple Square Grid

### 3.1.1 Workspace Preparation

```
clc;
clear all;
close all;
```

The first three lines of the code serve to prepare the MATLAB environment for the simulation. The command `clc` clears the command window, removing any previous outputs. The `clear all` command removes all variables from the workspace, ensuring no leftover data from prior runs could affect the current simulation. Lastly, `close all` closes any open figures, providing a clean slate for visualizations. This initial setup is crucial for ensuring accurate and reliable results throughout the execution of the script.

### 3.1.2 Creation of the grid

```
nx = 20;
ny = 20;
nz = 20;
Lx = 10;
Ly = 10;
Lz = 10;
G1 = cartGrid([nx, ny, nz], [Lx, Ly, Lz]);
G1 = computeGeometry(G1);
```

In this section, the code initializes a new Cartesian grid for a reservoir simulation. The variables `nx`, `ny`, and `nz` are set to 20, indicating the number of cells in the x, y, and z directions, respectively. The physical dimensions of the grid are defined by the variables `Lx`, `Ly`, and `Lz`, each set to 10 m. The command `cartGrid([nx, ny, nz], [Lx, Ly, Lz])` creates a structured grid with the specified dimensions. Following this, the `computeGeometry` function is employed to compute essential geometric information. This routine generates the basic geometry and topology of the grid, detailing how nodes connect to form faces, how faces connect to create cells, and how cells are linked via common faces. While this information suffices for many applications, additional geometric details, such as cell centroids, volumes, face areas, centroids, and normals, may be required in various cases. The computation of this information is straightforward for simplexes and Cartesian grids but can be complex for general polyhedral grids containing curved polygonal faces. Thus, this step is critical for accurate modeling and simulation of flow and transport within the reservoir.

### 3.1.3 Visualization of the grid

```
figure(1)
plotGrid(G1)
xlabel('Length (m)')
ylabel('Width (m)')
zlabel('Heigth (m)')
title('Basic Grid Visualization');
view(3)
shg
```

In this section, the code generates a 3D visualization of the newly created grid `G1`. The `figure(1)` command opens a new figure window for plotting. The `plotGrid(G1)` function is used to display the grid structure, providing a visual representation of the arrangement and connectivity of the cells. The axis labels are set using `xlabel`, `ylabel`, and `zlabel` to indicate the dimensions being represented: Length, Width, and Height, respectively. A title is added with the `title` function to describe the visualization as "Basic Grid Visualization." The `view(3)` command adjusts the perspective to a three-dimensional view, allowing for a better understanding of the grid's spatial layout. Finally, the `shg` command ensures that the current figure window is brought to the front, making it visible to the user.

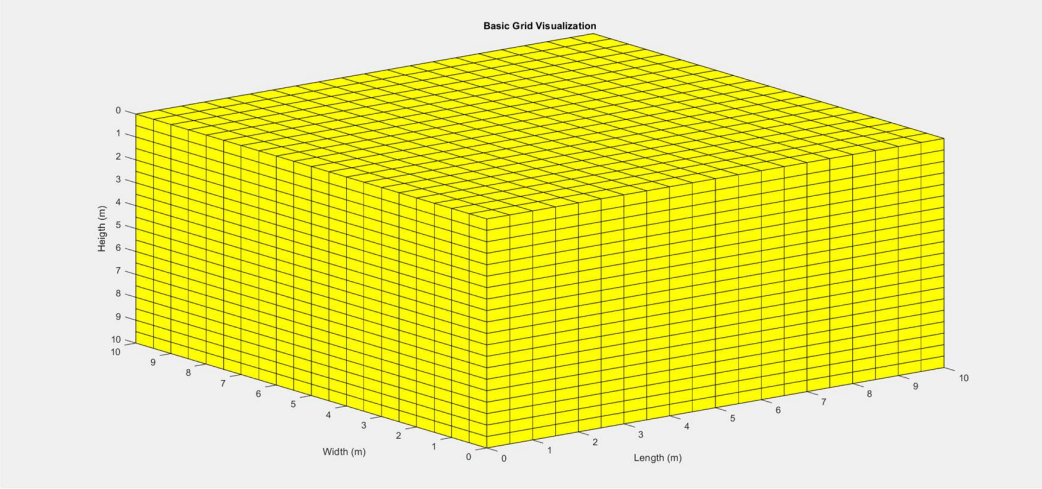


**Figure 36: Simple Square Grid Visualization.**

### 3.1.4 Creation and Visualization of the Tensor Grid

A tensor grid (or tensor-product grid) is a type of structured grid in computational modeling, particularly in numerical simulations like reservoir simulation, where the grid is formed by the Cartesian product of one-dimensional coordinate arrays in each spatial dimension.

```
figure(2)
xvalues=[(1:2:nx/3),(nx/2+1:2:nx)];
yvalues=[(1:2:ny/2),(ny/2+1:2:ny)];
zvalues=[(1:2:nz)];
G2=tensorGrid(xvalues,yvalues,zvalues);
plotGrid(G2, 'FaceColor','none')
xlabel('Length')
ylabel('width')
zlabel('heigth')
title('Tensor Grid Visualization');
view(3)
shg
```

This code constructs a tensor grid using specified x, y, and z values and then visualizes it. The `figure(2)` command opens a new figure window for this purpose. The `xvalues`, `yvalues`, and `zvalues` arrays are defined using ranges:

- `xvalues` includes every second index from 1 to a third of `nx` and from half of `nx` to `nx`
- `yvalues` includes every second index from 1 to half of ny and from half of `ny` to `ny`
- `zvalues` includes every second index from 1 to `nz`.

The `tensorGrid` function uses these arrays to create a grid `G2` that captures the desired structure. The `plotGrid(G2, 'FaceColor','none')` function displays the tensor grid with transparent faces, allowing for a clear view of the grid's arrangement. Axis labels are added with `xlabel`, `ylabel`, and `zlabel`, and a title is provided to describe the visualization as "Tensor Grid Visualization." The `view(3)` command adjusts the perspective to three dimensions, while `shg` ensures the figure is brought to the front.

The `tensorGrid` function constructs a tensor-product grid (also called a Cartesian product grid) from three input vectors of coordinates (`xvalues, yvalues, zvalues`). Each grid point is defined by a combination of one x-coordinate, one y-coordinate, and one z-coordinate:

$$Grid\ points = \{(x_i, y_i, z_k) | x_i \in xvalues, y_j \in yvalues, z_k \in zvalues\}$$

This results in a grid where each cell is a 3D rectangular prism, with faces aligned along the x, y, and z axes.

The grid is structured, meaning it has a regular topology where cells are organized in a logically rectangular array (`i, j, k`). Each cell's geometry is defined by the differences between consecutive coordinates in `xvalues, yvalues, and zvalues`.
The non-uniform spacing in `xvalues` and `yvalues` (due to the concatenation of two sequences) creates varying cell sizes, but the grid remains a tensor grid because it is still formed by the tensor product of the coordinate vectors.

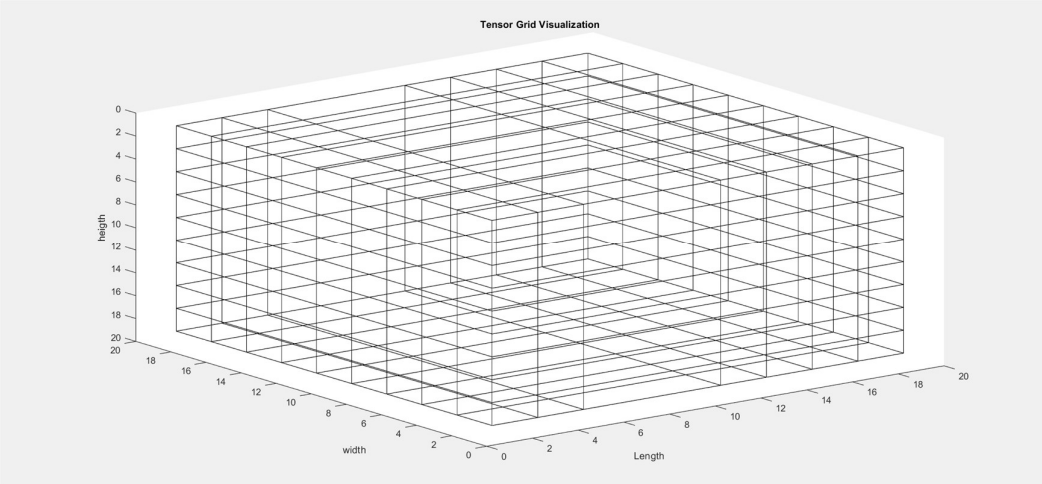


**Figure 37: Tensor Grid Visualization.**

### 3.1.5 Creation and Visualization of Grid with removed cells

```
figure(3);
G3 = computeGeometry(G1);
c = G3.cells.centroids;
center = [Lx/2, Ly/2, Lz/2];
dd = sqrt((c(:,1) - center(1)).^2 + 0.25*(c(:,2) - center(2)).^2 +
0.25*(c(:,3) - center(3)).^2);
G4 = removeCells(G1, dd> 5);
plotGrid(G4);
view(-70, -70);
axisequal;
xlim([0, Lx]);
ylim([0, Ly]);
zlim([0, Lz]);
xlabel('Length (m)');
ylabel('Width (m)');
zlabel('Height (m)');
title('Grid with Cells Removed Based on Distance');
```

In this section, the code processes the existing grid `G1` to remove cells based on their distance from a specified center point. The `figure(3)` command opens a new figure window for visualization. The `computeGeometry` function is called to compute the geometry of `G1`,

resulting in `G3`, which contains the centroids of the cells stored in `c`. A `center` point is defined at the middle of the grid dimensions, `[Lx/2, Ly/2, Lz/2]`. The code calculates the distance of each cell centroid from this center point using a modified distance formula that applies a weighting to the y and z coordinates: `dd = sqrt((c(:,1) - center(1)).^2 + 0.25*(c(:,2) - center(2)).^2 + 0.25*(c(:,3) - center(3)).^2)`. Cells that are further than a distance of 5 units from the center are identified for removal using `removeCells(G1, dd > 5)`, resulting in the new grid `G4`. The `plotGrid(G4)` function displays this modified grid. The `view(-70, -70)` command adjusts the viewpoint, while `axisequal` ensures equal scaling on all axes. Axis limits are set with `xlim`, `ylim`, and `zlim`, and axis labels along with a title are added to describe the visualization as 'Grid with Cells Removed Based on Distance'.

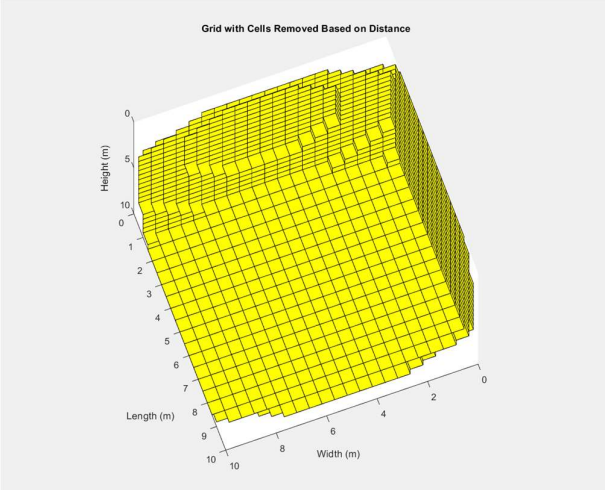


**Figure 38: Grid with cells removed based on distance.**

### 3.1.6 Log-Normal permeability layers visualization

```
figure(4)
FirstLayPerm=100;
SecondLayPerm=400;
ThirdLayPerm=5;
nxx=100;
nyy=30;
nzz=50;
G3=processGRDECL(simpleGrdecl([nxx nyy nzz],0.4));
perm=logNormLayers(G3.cartDims,[FirstLayPermSecondLayPermThirdLayPerm],
'indices',[1 uint8(nzz/4) uint8(nzz/2)  nzz+1]);
plotCellData(G3,perm,'EdgeColor','none');
xlabel('Length')
ylabel('width')
zlabel('heigth')
title('Log-Normal Permeability Layers Visualization');
view(3);
axis tight
```

In this section, the code visualizes permeability layers in a reservoir grid using a log-normal distribution. The `figure(4)` command opens a new figure window for the plot. Permeability values for three different layers are defined: `FirstLayPerm` is set to 100, `SecondLayPerm` to 400, and `ThirdLayPerm` to 5. The dimensions of the grid are defined with `nxx`, `nyy`, and `nzz`, set to 100, 30, and 50, respectively. The `processGRDECL` function is called with `simpleGrdecl([nxx, nyy, nzz], 0.4)` to generate a grid structure based on these dimensions.

The log-normal permeability layers are generated using the `logNormLayers` function, which takes the grid dimensions and permeability values, specifying indices at which the layer transitions occur: the first layer at the top, the second at one-quarter of the total height, and the third at half the height. The result is stored in `perm`. The `plotCellData(G3, perm, 'EdgeColor', 'none')` function visualizes the permeability distribution across the grid cells without displaying cell edges. Axis labels are added with `xlabel`, `ylabel`, and `zlabel`, and a title is set to describe the visualization as "Log-Normal Permeability Layers Visualization." The `view(3)` command adjusts the perspective to three dimensions, and `axistight` is used to minimize the whitespace around the plot.

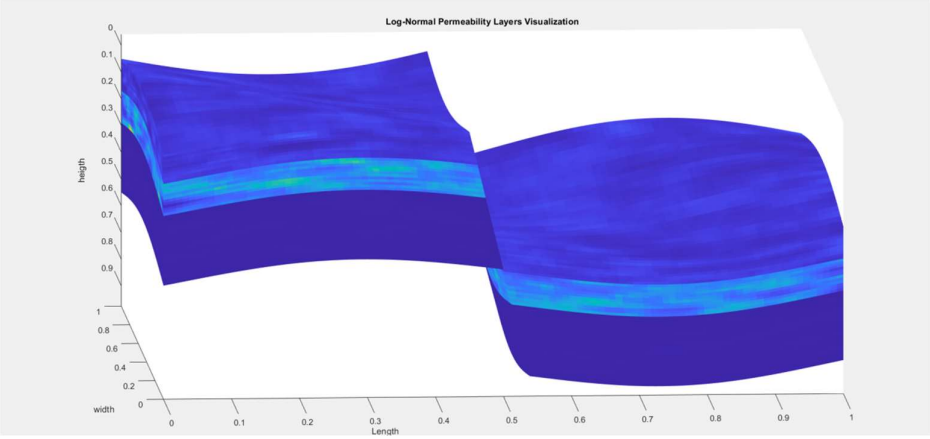


**Figure 39: Log-Normal permeability layers visualization (x-axis).**

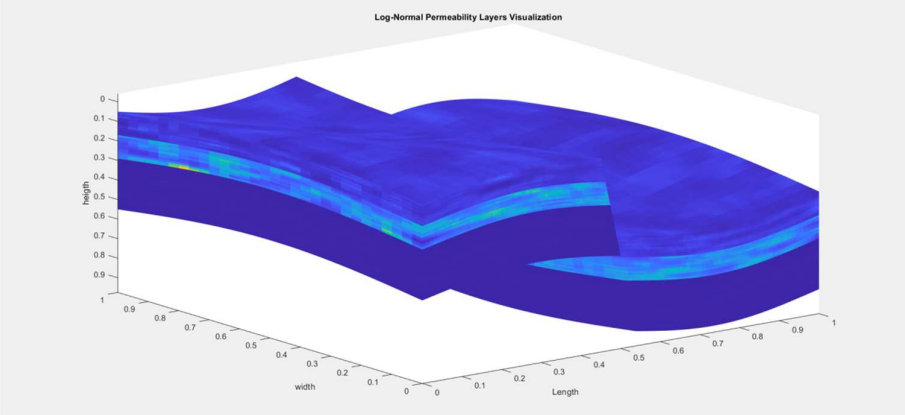


**Figure 40: Log-Normal permeability layers visualization.**

### 3.1.7 Grid with random porosity and permeability

```
figure(5)
minporos=0.15;
maxporos=0.30;
porosity=gaussianField(G1.cartDims,[minporos,maxporos],[15 15 15],1.5);
permeability=porosity.^3.*(1e-5)^2./(0.81*72*(1-porosity).^2);
rock1=makeRock(G1,permeability(:),porosity(:));
plotCellData(G1,rock1.poro,'EdgeColor','none')
xlabel('Length')
ylabel('Width')
zlabel('Heigth')
title('Rock Properties with Porosity and Permeability');
colorbar('horiz');
axis equal tight;
view(3)
shg
```

In this section, the code visualizes rock properties, specifically porosity and permeability, within the reservoir grid `G1`. The `figure(5)` command opens a new figure window for plotting. The minimum and maximum porosity values are set to `minporos = 0.15` and `maxporos = 0.30`. The `gaussianField` function generates a spatially variable porosity field within the grid dimensions defined by `G1.cartDims`. The parameters `[minporos, maxporos]` define the range of porosity values, while `[15 15 15]` specifies the correlation length, and `1.5` sets the standard deviation of the Gaussian distribution.

The permeability is calculated using the formula `permeability = porosity.^3 .* (1e-5)^2 ./ (0.81 * 72 * (1 - porosity).^2)`, which relates permeability to porosity. The `makeRock` function creates a rock model (`rock1`) using the generated permeability and porosity vectors. The `plotCellData(G1, rock1.poro, 'EdgeColor', 'none')` function visualizes the porosity distribution across the grid cells without displaying cell edges. Axis labels are added using `xlabel`, `ylabel`, and `zlabel`, and a title describes the visualization as "Rock Properties with Porosity and Permeability." A horizontal colorbar is included for reference, and `axisequal tight` ensures equal scaling and tight limits around the plot. The `view(3)` command adjusts the perspective to three dimensions, and `shg` brings the figure window to the front.

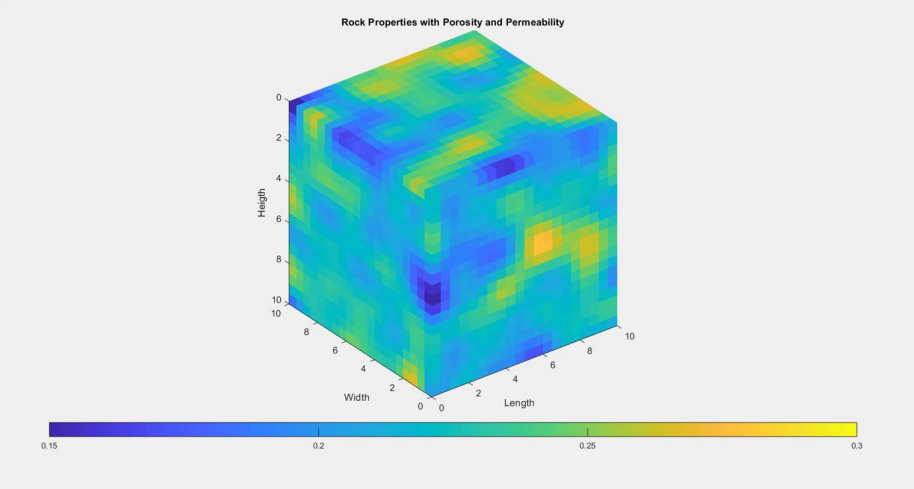


**Figure 41: Grid with random porosity and permeability.**

### 3.1.8 Grid with three different layers

```
figure(6)
FirstLayPerm=100;
SecondLayPerm=400;
ThirdLayPerm=50;
Lx=50;Ly=50;Lz=30;
G1=cartGrid([nx,ny,nz],[Lx,Ly,Lz]);
perm=logNormLayers(G1.cartDims,[FirstLayPermSecondLayPermThirdLayPerm],
'indices',[1 uint8(nz/4) uint8(nz/2)  nz+1],'sz',[15 15 15],'std',2);
plotCellData(G1,log10(perm),'EdgeColor','none');
view(3);
xlabel('Length')
ylabel('Width')
zlabel('Heigth')
axis tight equal
set(gca,'DataAspect', [1 1 1])
h=colorbar('horiz')
title('Log-Normal Permeability with Different Grid Resolution');
```

In this section, the code visualizes log-normal permeability values within a Cartesian grid while varying grid resolution. The `figure(6)` command opens a new figure window. Three permeability values are defined: `FirstLayPerm` as 100, `SecondLayPerm` as 400, and `ThirdLayPerm` as 50. The grid dimensions are set with `Lx`, `Ly`, and `Lz` to 50, 50, and 30 units, respectively. The grid `G1` is created using the `cartGrid` function with the specified dimensions.

The `logNormLayers` function generates permeability values based on a log-normal distribution. It takes the grid dimensions and the defined permeability values, with indices indicating where the layer transitions occur: at the top, one-quarter height, and half height. The parameters `'sz'` and `'std'` control the size and standard deviation of the log-normal distribution, respectively. The resulting permeability values are visualized using

`plotCellData(G1, log10(perm), 'EdgeColor', 'none')`, displaying the logarithm of permeability across the grid cells without cell edges.

The `view(3)` command sets the perspective to three dimensions, while axis labels are added using `xlabel`, `ylabel`, and `zlabel`. The `axistight equal` command ensures equal scaling on all axes, and the aspect ratio is adjusted with `set(gca, 'DataAspect', [1 1 1])`. A horizontal colorbar is included for reference, and a title is provided to describe the visualization as "Log-Normal Permeability with Different Grid Resolution.

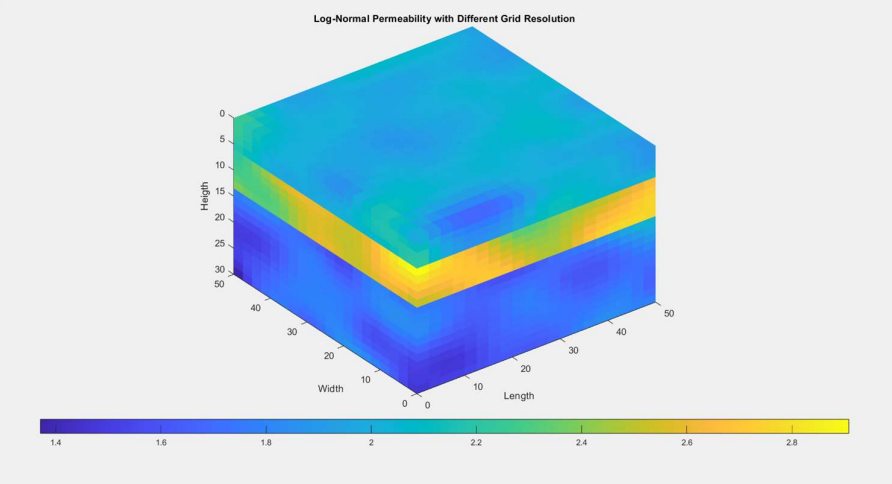


**Figure 42: Grid with three different permeability layers.**

### 3.1.9 Permeability Histogram

```
figure(7)
histogram(permeability,100)
xlabel('Permeability Values')
ylabel('Frequency')
title('Permeability Histogram');
```

In this section, the code generates a histogram to visualize the distribution of permeability values. The `figure(7)` command opens a new figure window for plotting. The `histogram(permeability, 100)` function creates a histogram with 100 bins, displaying the frequency of different permeability values calculated earlier. The x-axis is labeled as 'Permeability Values' using `xlabel`, while the y-axis is labeled 'Frequency' with `ylabel`. A title, "Permeability Histogram," is set using the `title` function to describe the plot. This visualization helps to understand the distribution and spread of permeability values within the reservoir model.

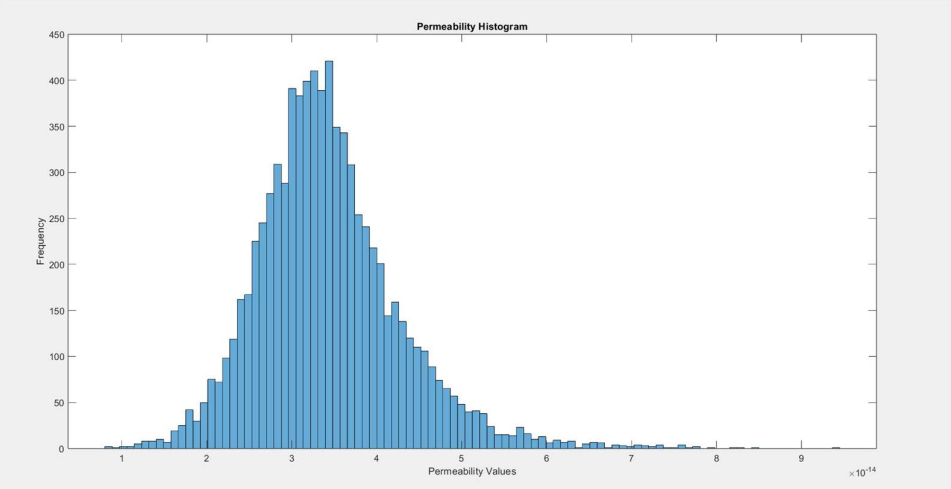


**Figure 43: Permeability histogram.**

### 3.1.10 Logarithmic permeability histogram

```
figure(8)
histogram(log10(perm),100)
xlabel('Log-Transformed Permeability Values')
ylabel('Frequency')
title('Histogram of Log-Transformed Permeability');
shg
```

In this section, the code creates a histogram to visualize the distribution of logarithmic permeability values. The `figure(8)` command opens a new figure window for the plot. The `histogram(log10(perm), 100)` function generates a histogram with 100 bins, representing the frequency of the logarithm (base 10) of the permeability values obtained from the earlier calculations. The x-axis is labeled as 'Log-Transformed Permeability Values' using the `xlabel` function, while the y-axis is labeled 'Frequency' with `ylabel`. A title is added with `title` to indicate that this is a "Histogram of Log-Transformed Permeability." This transformation provides insights into the distribution characteristics of the permeability values, often revealing patterns that are not immediately apparent in the original data. The `shg` command ensures that this figure window is brought to the front for viewing.

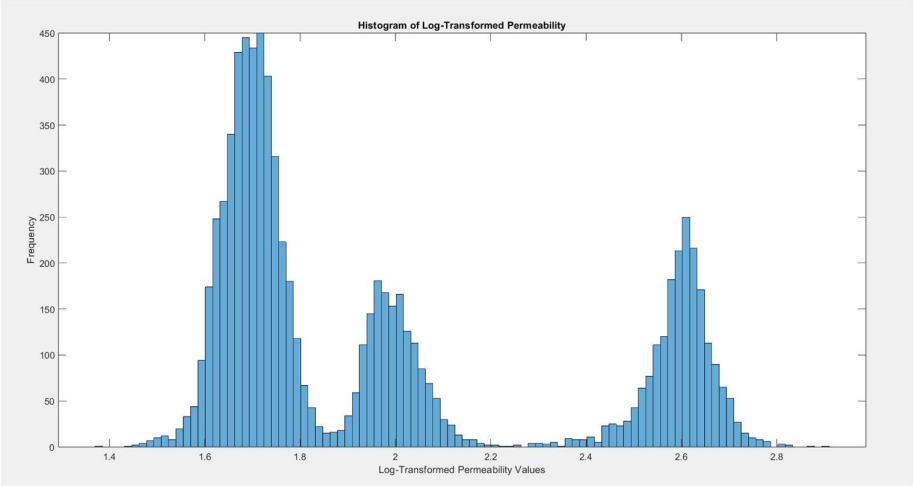


**Figure 44: Log-Transformed permeability histogram.**

### 3.1.11Comparison of Permeability Histograms

The first histogram displays the raw permeability values, while the second histogram shows the log-transformed permeability values. In the raw permeability histogram**,** we often observe a right-skewed distribution, where a majority of permeability values cluster at the lower end, with fewer higher values. This reflects the natural variability in reservoir properties, where most formations tend to have lower permeabilities with outliers representing high-permeability zones. In contrast, the log-transformed permeability histogram typically presents a more symmetric distribution. Log transformation helps to normalize the data, reducing the impact of extreme values. This transformation is particularly useful for statistical analyses, as it allows for better modeling of relationships between variables and simplifies comparisons. Overall, the log transformation often reveals underlying patterns and relationships that are less visible in the raw data, making it a valuable tool in reservoir characterization and analysis.

## 3.2 Reservoir Grid Creation and Well Setup

This MATLAB code is designed to model a three-dimensional grid representing a reservoir with anisotropic permeability, specifically for a dual-porosity system. It begins by setting up the grid dimensions and geometry, followed by defining permeability values for different sections of the reservoir. The permeability is adjusted to reflect variations throughout the grid, where top, middle, and bottom layers have distinct properties. Next, the code initializes fluid properties for water and oil, then creates two wells: an injector and a producer, each with specific flow rates and characteristics. Finally, the code visualizes the permeability distribution and well placements in a 3D plot, providing insights into the reservoir's behavior.

### 3.2.1 Workspace Preparation

```
clc;
clear all;
close all;
```

The first three lines of the code serve to prepare the MATLAB environment for the simulation. The command `clc` clears the command window, removing any previous outputs. The `clear all` command removes all variables from the workspace, ensuring no leftover data from prior runs could affect the current simulation. Lastly, `close all` closes any open figures, providing a clean slate for visualizations. This initial setup is crucial for ensuring accurate and reliable results throughout the execution of the script.

### 3.2.2 Creation of the Grid

```
nx = 10;
ny = 10;
nz = 10;
Lx = 10;
Ly = 10;
Lz = 10;
G = cartGrid([nx, ny, nz], [Lx, Ly, Lz]);
G = computeGeometry(G);
```

In this section, the code defines the dimensions of the grid for the reservoir simulation. The variables `nx`, `ny`, and `nz` represent the number of cells in the x, y, and z directions, respectively, each set to 10. The variables `Lx`, `Ly`, and `Lz` specify the physical lengths of the grid in each direction, also set to 10 meters. The `cartGrid` function is then called to create a Cartesian grid based on these specifications, generating a structured grid of cells. Following this, the `computeGeometry` function is employed to compute essential geometric information. This routine generates the basic geometry and topology of the grid, detailing how nodes connect to form faces, how faces connect to create cells, and how cells are linked via common faces. While this information suffices for many applications, additional geometric details, such as cell centroids, volumes, face areas, centroids, and normals, may be required in various cases. The computation of this information is straightforward for simplexes and Cartesian grids but can be complex for general polyhedral grids containing curved polygonal faces. Thus, this step is critical for accurate modeling and simulation of flow and transport within the reservoir.

### 3.2.3 Defining Anisotropic Permeability

```
K_base = 200;
K_top = 200;
K_middle = 250;
K_bottom = 270;
K = zeros(G.cells.num, 1);
```

In this section, the code initializes the parameters for anisotropic permeability within the reservoir. The variables `K_base`, `K_top`, `K_middle`, and `K_bottom` are defined with values of 200, 200, 250, and 270, respectively, representing the permeability in different layers of the reservoir. These values indicate that the top and base layers have the same permeability, while the middle layer has a higher permeability. The line `K = zeros(G.cells.num, 1)` creates a vector initialized to zero, which will hold the permeability values for each cell in the grid. This setup is essential for modeling fluid flow through the reservoir, as it reflects the varying permeability characteristics in different sections.

### 3.2.4 Defining Permeability for Each Node

```
K(1:nx*ny) = K_top;
K(nx*ny+1:nx*ny+nx*ny*6) = K_middle;
K(nx*ny+nx*ny*6+1:end) = K_bottom;
rock = makeRock(G, 200*milli*darcy, 0.25);
hT = computeTrans(G, rock);
```

In this section, the code assigns specific permeability values to different regions of the grid. The line `K(1:nx*ny) = K_top` sets the permeability of the top layer cells to the value defined as `K_top`. The next line, `K(nx*ny+1:nx*ny+nx*ny*6) = K_middle`, assigns the middle layer's permeability to the specified `K_middle` value, covering a range of cells. Finally, `K(nx*ny+nx*ny*6+1:end) = K_bottom` assigns the permeability of the bottom layer cells to `K_bottom`. This structured assignment allows for a clear differentiation of permeability throughout the reservoir. The code then uses the `makeRock` function to create a rock model with a specified permeability of 200 milli-darcy and a porosity of 0.25. Subsequently, the `computeTrans` function calculates the transmissibility of the grid based on the rock properties, preparing it for flow simulations.

### 3.2.5 Fluid Initialization

```
mrstModule add incomp
fluid = initSimpleFluid('mu',[1,10],'rho',[1000 700],'n',[2 2]);
```

In this section, the code initializes the fluid properties for the reservoir simulation. The command `mrstModule add incomp` activates the module for incompressible fluid flow within the MATLAB Reservoir Simulation Toolbox (MRST). The subsequent line creates a fluid model using the `initSimpleFluid` function. Here, the parameters `'mu'`, `'rho'`, and `'n'` are defined, where `'mu'` specifies the viscosities of the fluids, set to 1 cP for water and 10 cP for oil. The `'rho'` parameter defines the densities, with values of 1000 kg/m³ for water and 700 kg/m³ for oil. The `'n'` parameter indicates the viscosity and density exponents, both set to 2. This setup is essential for accurately modeling the behavior of the fluids within the reservoir, allowing for the simulation of flow dynamics between the water and oil phases.

### 3.2.6 Well Creation

```
W = verticalWell([], G, rock, 1, 1, 1:nz, 'Type', 'rate', 'Val',
1000*meter^3/day, 'Radius', 0.1, 'Name', 'Injector', 'Comp_i', 1);
W = verticalWell(W , G, rock, 10, 10, 1, 'Type', 'bhp', 'Val',
800*psia, 'Radius', 0.1, 'Name', 'Producer', 'Comp_i', 2);
```

In this section, the code defines two vertical wells within the reservoir model: an injector and a producer. The first line creates the injector well using the `verticalWell` function. It initializes the well with no prior data (indicated by the empty brackets) and associates it with the grid `G` and rock properties. The injector is positioned at the coordinates (1, 1) and extends through all vertical layers (`1:nz`). This well is designated as a 'rate' type, with a flow rate of 1000 cubic meters per day, a radius of 0.1 meters, and is identified as "Injector" with a component index of 1 for water. The second line similarly defines the producer well at the coordinates (10, 10), but this one is configured as a 'bhp' (bottom hole pressure) type, set to maintain a pressure of 800 psia. It also has a radius of 0.1 meters and is named "Producer" with a component index of 2 for oil. This configuration allows the simulation to capture the interactions between the injector and producer wells effectively.

### 3.2.7 Visualization of Permeability Distribution and Wells

```
figure(9);
plotCellData(G, K);
title('Permeability Distribution');
xlabel('Length (m)');
ylabel('Width (m)');
zlabel('Height (m)');
plotWell(G, W, 'radius', 0.2, 'height', 0.1, 'color', 'r', 'label',
'on', 'depth', 'type');
view(3);
cbar = colorbar;
cbar.Label.String = 'Permeability (mD)';
alpha(0.7);
```

In this section, the code generates a 3D plot to visualize the permeability distribution within the reservoir grid. The `figure(9)` command creates a new figure window. The `plotCellData(G, K)` function displays the permeability values assigned to each grid cell, allowing for a visual representation of how permeability varies throughout the reservoir. The plot is enhanced with titles and axis labels using `title`, `xlabel`, `ylabel`, and `zlabel` to clearly indicate the dimensions being represented. Additionally, the `plotWell` function overlays the injector and producer wells on the grid, specifying their radius, height, color, and labeling options for clarity. The `view(3)` command adjusts the view to a 3D perspective. A colorbar is added to the plot with `colorbar`, which provides a reference for interpreting permeability values, labeled as 'Permeability (mD)'. Finally, the `alpha(0.7)` function is used to set the transparency of the plot, allowing for better visualization of the wells against the permeability distribution.

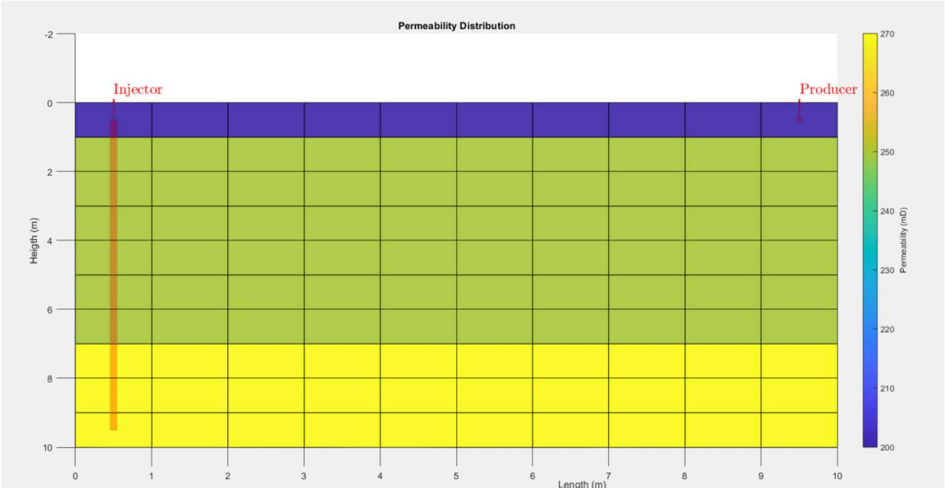


**Figure 45: Permeability distribution (x-axis).**

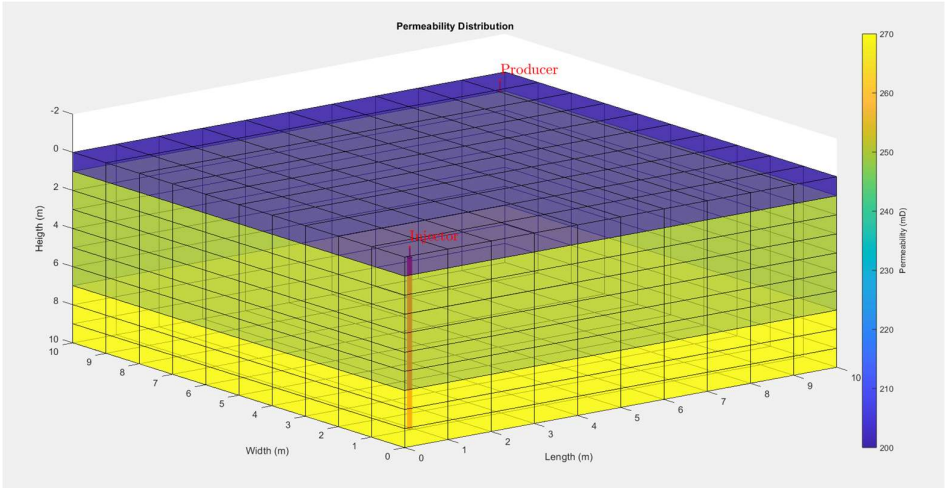


**Figure 46: Permeability distribution.**

## 3.3 Importing the SPE 1 Simulation

As mentioned earlier, it is necessary to run the *'startup.m'* file, and then we open '*mrstExploreModules()'*, where we select '*ad-blackoil'* and subsequently the '*spe1\blackoilTutorialSPE1.m'* and '*spe1\simulateSPE1.m'* files, and finally select '*EDIT'*.

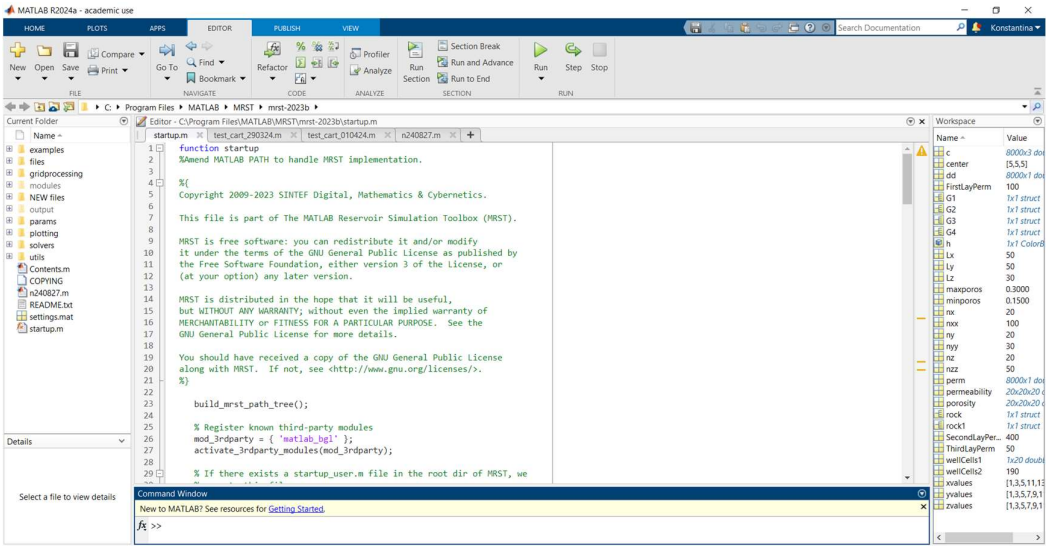


**Figure 47: Run 'startup.m'.**

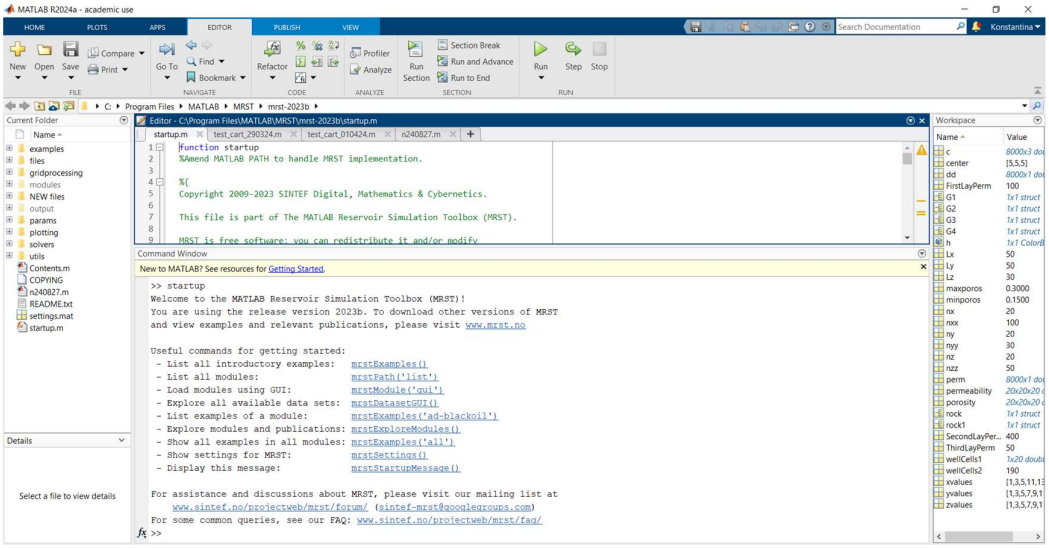


**Figure 48: Click the 'mrstExploreModules()'.**

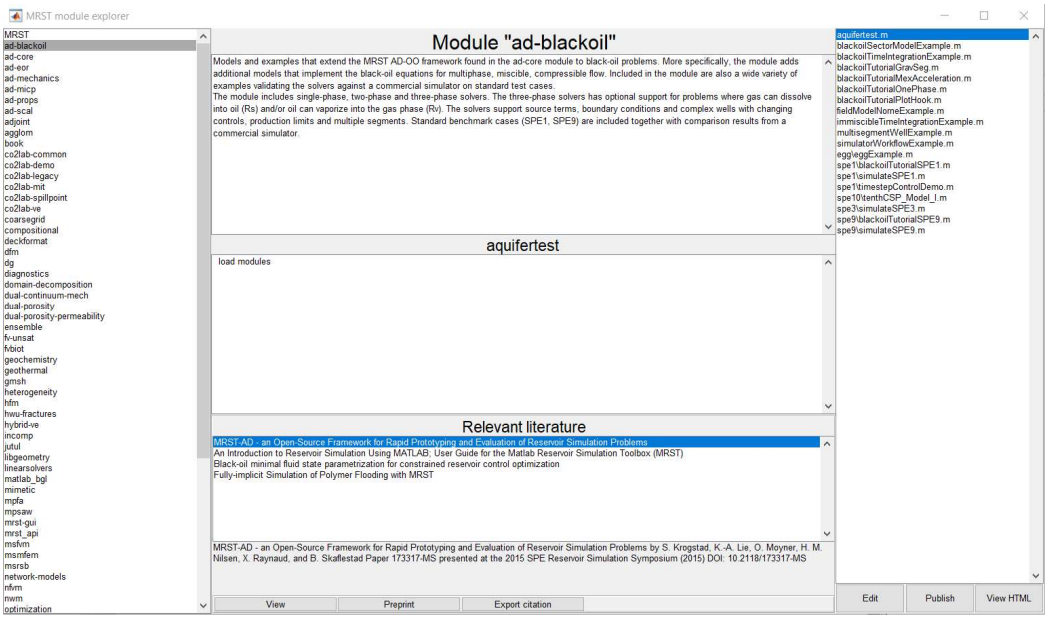


**Figure 49: Click the 'ad-blackoil'.**

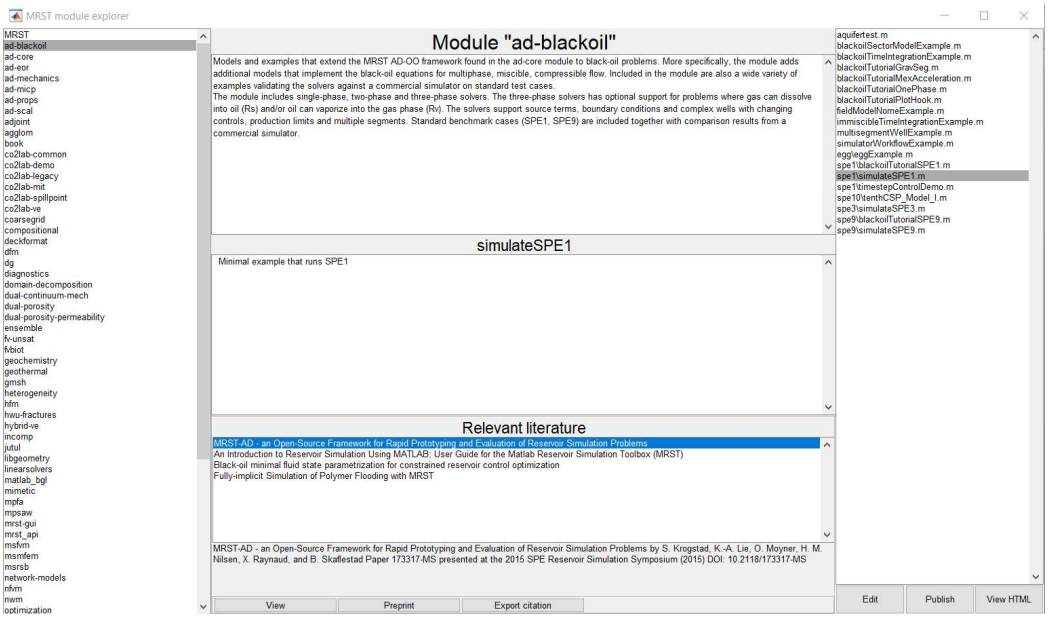


**Figure 50: Select 'spe1\blackoilTutorialSPE1.m' and 'spe1\simulateSPE1.m'.**

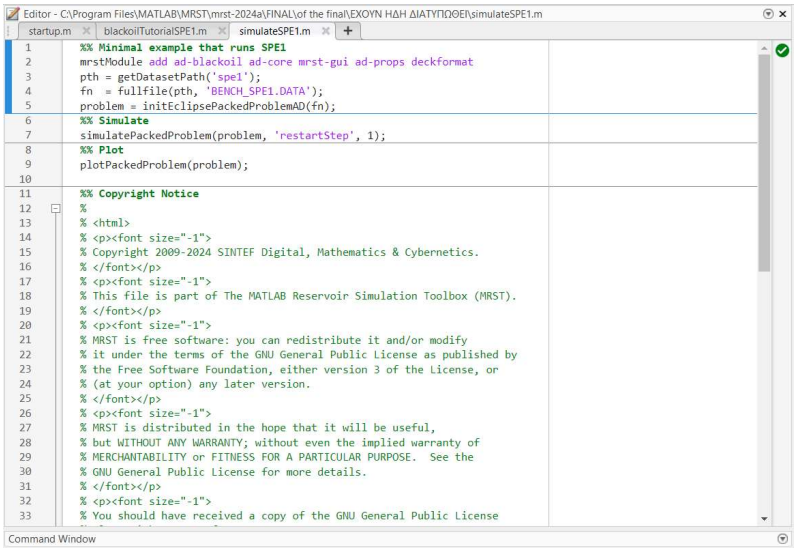


**Figure 51: The 'spe1\blackoil TutorialSPE1.m' and 'spe1\simulateSPE1.m' open at the editor.**

## 3.4 The Odeh Benchmark (SPE 1)

The Odeh Benchmark or SPE1 tutorial, developed by Dr. A.S. Odeh, is an introduction to the basic capabilities of the MATLAB Reservoir Simulation Toolbox (by Knut-Andreas Lie). Its main objective is implementing and solving a black-oil problem, which is a standard approach to multiphase fluid flow simulation (water, oil and gas). The MRST modules that comprise this black-oil tutorial set up a workflow that can visualize a reservoir simulation. Through the SPE1 tutorial, users gain hands-on experience with MRST, learning how to utilize its robust features for reservoir simulation. This tutorial serves as a practical introduction to

MRST's capabilities, demonstrating its effectiveness in handling complex simulation tasks and providing a solid foundation for more advanced reservoir modeling studies.

The SPE 1 problem models a three-phase black-oil reservoir with gas injection in a small 10x10x3 grid, simulating depletion over 1216 days. The reservoir has dimensions of 10000 ft x 10000 ft x 100 ft, with layer thicknesses of 20 ft in layer 1, 30 ft in layer 2, and 50 ft in layer 3. The rock properties include a uniform porosity of 0.3 (30% pore volume), and isotropic permeability varying by layer: 500 mD in layer 1, 50 mD in layer 2, and 200 mD in layer 3. The fluid properties consist of three phases: oil, water, and gas, with gas dissolution in oil. The initial conditions include water saturation (Sw = 0.12), oil saturation (So = 0.88), and no free gas initially (Sg = 0.0), while the dissolved gas-oil ratio (Rs) remains constant. The reservoir is unsaturated, with the initial pressure above bubble point, meaning the oil contains dissolved gas but no free gas phase. The reservoir has two wells: a gas injection well and a production well both located at diagonally opposite corners. The gas injection drives oil production, causing pressure depletion as well as gas dissolution and expansion.

The black-oil model is a widely adopted mathematical framework in reservoir simulation, used to describe the flow of three phases (water, oil, and gas) within a porous reservoir. It accounts for fluid compressibility, gas dissolution in oil, and multiphase flow dynamics, simplifying the complex compositional behavior of hydrocarbons into three distinct components:

- Water: A slightly compressible, single-phase fluid that does not dissolve gas.
- Oil: A liquid phase (crude oil) that may contain dissolved gas, treated as a single component with pressure-dependent properties.
- Gas: A free gas phase that can also dissolve into the oil phase.

The black-oil model relies on several assumptions tailored to the SPE 1 problem:

1. Three-Phase Flow: Water, oil, and gas coexist, with phase saturations satisfying:

$$S_o + S_w + S_g = 1$$

2. Gas Dissolution: Gas dissolves in the oil phase, quantified by the solution gas-oil ratio $R_s(P_o)$, which depends on oil pressure. No gas dissolution occurs in the water phase.
3. Compressibility: All fluids are compressible, with reservoir volumes related to standard conditions via formation volume factors $B_w$, $B_o$ and $B_g$, which are pressure-dependent.
4. Isothermal Conditions: The reservoir maintains a constant temperature, with no heat transfer.
5. Local Equilibrium: Phases are in instantaneous thermodynamic equilibrium, allowing immediate gas dissolution or liberation based on pressure changes.
6. Darcy Flow: Fluid flow adheres to Darcy's law, modified by phase-specific relative permeabilities $k_{ri}(S_i)$, which account for multiphase interactions.

These assumptions enable the black-oil model to effectively capture the dynamics of gas injection in SPE 1, where gas displaces oil toward a production well, altering pressure and saturation profiles.

The SPE 1 problem is implemented in MRST through the `blackoilTutorialSPE1.mat` example, utilizing the `ad-blackoil` module. The following numerical methods are employed to solve the black-oil model equations:

- Fully Implicit Solver: Solves the coupled pressure and saturation equations simultaneously, ensuring stability for the complex dynamics of SPE 1.
- Finite Volume Discretization: Discretizes the reservoir domain to conserve mass across control volumes, suitable for the heterogeneous permeability field.
- Cartesian Grid with Corner-Point Geometry: Represents the SPE 1 reservoir (10x10x3 cells) with a structured grid, allowing flexible cell shapes to model geological features.
- Time Stepping: Advances the simulation over 10 years using adaptive time steps to balance accuracy and computational efficiency.
- Nonlinear Solver (Newton-Raphson): Iteratively solves the nonlinear system of equations arising from the implicit formulation.
- Linear Solver (GMRES or Conjugate Gradient): Resolves linear systems within each Newton iteration, enhancing computational robustness.
- Automatic Differentiation: Computes Jacobians efficiently, enabling rapid and accurate linearization of the governing equations.
- Peaceman Well Models: Represents injector and producer wells, calculating flow rates based on well indices and pressure differences.

These methods, implemented in MRST, leverage the toolbox's flexibility to accurately simulate the SPE 1 gas injection scenario.

### 3.4.1 Governing Equations of the SPE 1 Simulation

#### 3.4.1.1 Mass Conservation

Uses component-based equations with  and $Rs$ to account for gas dissolution, particularly in the gas component equation.

Water: $\phi \frac{\partial}{\partial t}\left(\frac{\rho_w S_w}{B_w}\right) + \nabla \cdot \left(\frac{\rho_w u_w}{B_w}\right) = q_w$

Oil: $\phi \frac{\partial}{\partial t}\left(\frac{\rho_o S_o}{B_o}\right) + \nabla \cdot \left(\frac{\rho_o u_o}{B_o}\right) = q_o$

Gas: $\phi \frac{\partial}{\partial t}\left(\frac{\rho_g S_g}{B_g} + \frac{\rho_g R_s S_o}{B_o}\right) + \nabla \cdot \left(\frac{\rho_g u_g}{B_g} + \frac{\rho_g R_s u_o}{B_o}\right) = q_g$

Where:

- $\phi$: Porosity
- $\rho_w$, $\rho_o$, $\rho_g$: Density
- $S_w$, $S_o$, $S_g$: Saturation
- $B_w$, $B_o$, $B_g$: formation volume factors
- $R_s$: Dissolved Gas-Oil Ratio
- $u_w$, $u_o$, $u_g$: Flow Rate
- $q_w$, $q_o$, $q_g$: Production

(Folk, n.d., Kleppe, n.d., Chen, n.d.)

#### 3.4.1.2 Darcy's Law for Multiphase Flow

$$v_i = -\frac{k_{ri}(S_i)K}{\mu_i b_i}(\nabla P_i - \rho_i g \nabla z), i = \{w, o, g\}$$

Where:

- $v_i$: Darcy velocity (volumetric flow per unit area)
- K: Absolute permeability tensor
- g: Gravitational constant
- z: Elevation
- $k_{ri}(S_i)$: Relative permeability of phase i , dependent on saturation $S_i$
- $\mu_i$: Viscosity of phase i
- $b_i$: Formation volume factor of phase i
- $P_i$ : Pressure of phase i
- $\rho_i$: Density of phase i

(ScienceDirect, n.d.)

#### 3.4.1.3 Effective Saturation

$$S_o^* = \frac{S_o - S_{or}}{1 - S_{or} - S_{gc}}$$

$$S_g^* = \frac{S_g - S_{gc}}{1 - S_{or} - S_{gc}}$$

Where:

- $S_{or}$: residual oil saturation
- $S_{gc}$: critical gas saturation

(HIS Energy, n.d.)

#### 3.4.1.4 Relative Permeability

Oil phase: $k_{ro}(S_o^*) = k_{ro0} \cdot (1 - S_g^*)^{n_o}$

Gas phase: $k_{rg}(S_g^*) = k_{rg0} \cdot (S_g^*)^{n_g}$

(HIS Energy, n.d.)

#### 3.4.1.5 Capillary pressure

Gas-Oil: $P_g = P_o + P_c^{g,o}(S_g)$

(ScienceDirect, n.d.)

#### 3.4.1.6 Saturation Constraint

$$S_o + S_w + S_g = 1$$

This means that the pore space is fully saturated with water, oil and gas.

Where:

- $S_o$: Oil Saturation
- $S_w$: Water Saturation
- $S_g$: Gas Saturation

#### 3.4.1.7 Formation Volume Factors

$$B_i = \frac{V_i^{res}}{V_i^{st}}, i = \{w, o, g\}$$

Where:

- $B_i$: Formation Volume Factor for phase i
- $V_i^{res}$: Volume of the phase at reservoir conditions
- $V_i^{st}$: Volume of the phase at stock tank (surface) conditions

(ScienceDirect, n.d.)

#### 3.4.1.8 The Peaceman Well Model

The Peaceman well model computes well flow rates in SPE 1's black-oil simulation by linking the pressure difference between the wellbore and adjacent grid block to a precomputed well index (WI), incorporating well radius, grid dimensions, and permeability. Its efficiency, compatibility with Cartesian grids, and widespread use in industry make it ideal for modeling vertical wells in multiphase reservoir simulations.

$$q_i = WI \cdot \frac{k_{ri}(S_i)\rho_i}{\mu_i B_i} \cdot \left(P_{grid} - P_{bhp}\right), i = \{w, o, g\}$$

Where:

- $q_i$: Flow rate (in surface volume units)
- WI: Well index (depends on grid geometry and permeability)
- $k_{ri}(S_i)$: Relative permeability
- $\rho_i$: Phase density
- $\mu_i$: Viscosity
- $B_i$: Formation volume factor
- $P_{grid}$: Pressure in the grid block connected to the well
- $P_{bhp}$: Bottom-hole pressure of the well

Well Index: Used to calculate the well production/injection rate for each phase.

$$WI = \frac{2\pi kh}{\ln\left(\frac{r_e}{r_w}\right) + S}$$

Where:

- k: permeability ($m^2$)
- h: thickness of the reservoir layer
- $r_w$: wellbore radius
- $r_e$: equivalent radius of the grid block (depends on grid size)
- S: skin factor (can be zero if neglected)

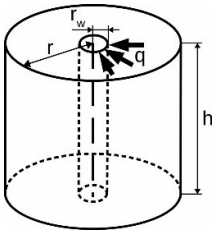


**Figure 40: Schematic representation of the radial inflow into a single well. (www.sciencedirect.com)**

### 3.4.2 Black Oil Tutorial SPE 1

```
mrstModule add ad-props deckformat mrst-gui ad-core ad-blackoil test-
suite
```

Beginning with the `mrstModuleadd` function the necessary modules are loaded so that all functionalities are available moving on. The `ad-props` and `ad-core` modules initialize the grid, petrophysical and fluid properties. The `deckformat` module is used to read and parse the SPE1 data deck, a standard input format that includes all essential data such as grid properties, fluid characteristics, and initial conditions.

The black-oil model, implemented with the `ad-blackoil` module, is set up to simulate the fluid flow dynamics within the reservoir. The model is solved using AD-based

solvers, which efficiently handle the non-linearities inherent in the governing equations. 'AD' stands for Automatic Differentiation, a computational technique used to evaluate derivatives of functions efficiently and accurately. In the context of reservoir simulation, AD-based solvers are specialized algorithms that use automatic differentiation to solve the complex mathematical equations governing fluid flow in the reservoir. Governing Equations are the mathematical equations that describe how fluids (oil, water, gas) move through the porous rock in a reservoir. For a black-oil model, these equations include mass conservation laws and Darcy's law for fluid flow, among others. These equations are often non-linear, meaning that the relationship between variables is not a simple straight line. Non-linearities arise from various factors such as changes in fluid properties (like viscosity and density), phase behavior (how oil, water, and gas interact), and the complex geometry of the reservoir.

Finally, the '*mrst-gui'* module provides interactive visualization tools to explore and analyze the simulation results, while the '*test-suite'* module offers a collection of test cases to validate the simulation's accuracy and reliability.

```
[G, rock, fluid, deck, state] = setupSPE1();
model = selectModelFromDeck(G, rock, fluid, deck);
model
schedule = convertDeckScheduleToMRST(model, deck);
```

The `setupSPE1()` function is a custom-designed routine that plays a crucial role in initializing and setting up the SPE1 simulation case within the MATLAB Reservoir Simulation Toolbox (MRST). This function performs several essential tasks to prepare the simulation environment. It begins by loading the data deck, which serves as the primary input source containing all the necessary simulation parameters. Following this, the function initializes the grid structure (`G`), which encapsulates the reservoir's geometry and topology, providing the spatial framework for the simulation. Additionally, it defines the petrophysical properties of the reservoir rock (`rock`), such as porosity and permeability, as well as the fluid properties (`fluid`), which describe the behavior and characteristics of the reservoir fluids, including oil, water, and gas. Finally, the function sets up the initial state (`state`) of the reservoir, detailing key parameters such as initial pressure, saturation levels, and other state variables that define the starting conditions of the simulation.

Following the initialization, the `selectModelFromDeck(G, rock, fluid, deck)` function is employed to select and initialize the most appropriate simulation model based on the detailed information provided in the data deck. This function analyzes the grid structure (`G`), rock properties (`rock`), fluid properties (`fluid`), and the data deck (`deck`) to determine the most suitable model for the simulation. The selected and initialized model is then stored in the output variable model, which represents the simulation framework that will be used to perform the fluid flow calculations.

Once the model has been selected, the `convertDeckScheduleToMRST(model, deck)` function is utilized to translate the scheduling information contained in the data deck into a format that MRST can interpret and execute. This scheduling information encompasses the sequence of simulation steps, including the time steps and control settings for wells and other operational parameters. The output variable schedule generated by this function represents the simulation schedule, which MRST will follow to control the simulation's progression and workflow.

```
figure;
plotCellData(G, convertTo(rock.perm(:,1), milli*darcy), ...
'FaceAlpha', 0.5, 'EdgeAlpha', 0.3, 'EdgeColor', 'k');
plotWell(G, schedule.control(1).W, 'radius',.5);
title('Permeability (mD)')
axistight, view(35, 40), colorbar('SouthOutside');
```

To visualize the permeability distribution within the reservoir grid, a new figure is initiated using the `figure;` command, which opens a blank canvas for plotting.

The `plotCellData(...)` function is then employed to plot the permeability data (`rock.perm(:,1)`) across the reservoir grid (`G`). The permeability values are converted to milli-darcy units using the `convertTo(rock.perm(:,1), milli*darcy)` function. The transparency of the plotted cell faces is controlled by the `'FaceAlpha', 0.5` parameter, setting it to 50%, while the transparency of the cell edges is set to 30% using `'EdgeAlpha', 0.3`. The edges of the cells are colored black, as specified by `'EdgeColor', 'k'`.

Next, the `plotWell(G, schedule.control(1).W, 'radius', .5);` function is used to plot the well locations on the grid (`G`). The well information is extracted from `schedule.control(1).W`, which likely contains details about the well control, including its position and type. The radius of the wells is set to 0.5 units, although the specific unit of measurement is not explicitly defined (typically grid units are used).

To enhance the plot's readability, a title is added using the `title('Permeability (mD)')` command, indicating that the data plotted represents permeability measured in milli-darcy (mD). The axis limits are adjusted to fit the data tightly using the `axis tight` command. The viewing perspective of the plot is set with `view(35, 40)`, which adjusts the azimuth and elevation angles to provide a suitable 3D perspective. Finally, a color bar is added below the plot using `colorbar('SouthOutside')`, which provides a reference scale for the plotted permeability values.

```
nls = NonLinearSolver('useLinesearch', true);
[wellSols, states, report] = simulateScheduleAD(state, model, schedule,
'nonlinearsolver', nls);
```

The creation of a `NonLinearSolver` object is a critical step in reservoir simulation, as it provides the necessary tools to solve the nonlinear systems of equations that arise from the complex interactions within the reservoir. The `NonLinearSolver` function initializes this object, and one of its key features is the `'useLinesearch', true` option. This option enables the use of a line search algorithm, which is essential for determining the optimal step size during the solution process. The use of a line search improves the convergence rate and stability of the solver, particularly in challenging simulations where the non-linearities are pronounced.

Following the setup of the solver, the `simulateScheduleAD` function is employed to execute the reservoir simulation over a predefined operational schedule using an Automatic Differentiation (AD) model. This function integrates several key components:

`state`: The initial conditions of the reservoir, including parameters such as pressure and saturation levels.

`model`: The reservoir model, encompassing the physical properties of the reservoir and the governing equations that dictate fluid flow and behavior.

`schedule`: The operational schedule, which defines the control actions over time, such as injection and production rates, and well settings.

`nls`(Nonlinear Solver): The previously created nonlinear solver is specified for use in the simulation, ensuring that the complex equations are solved with high accuracy and efficiency.

The output of the simulateScheduleAD function includes several critical results:

`wellSols`: This output contains detailed solutions for the wells at each time step, including data such as pressures and flow rates.

`states`: This output tracks the state of the reservoir, such as pressure and saturation distributions, throughout the simulation.

`report`: The report provides a comprehensive summary of the simulation, including convergence details, computational time, and other diagnostic information.

The `simulateScheduleAD` function plays a pivotal role in orchestrating the entire simulation process. By integrating the initial conditions, the physical model, and the operational schedule, it computes the evolution of the reservoir state and well performance over time. The use of a nonlinear solver is crucial in this context, as it ensures that the complex, nonlinear equations governing the reservoir behavior are solved with both accuracy and efficiency.

```
plotWellSols(wellSols, report.ReservoirTime)
figure;
plotToolbar(G, states)
plotWell(G, schedule.control(1).W)
axistight, view(-10, 60)
```

Effective visualization is crucial for understanding and analyzing well and reservoir simulations. This section explores the key functions and techniques used to visualize simulation results, focusing on well solutions, reservoir states, and plot adjustments.

The `plotWellSols` function is a vital tool for visualizing well simulation results over time. This function takes as input `wellSols`, the solutions generated by the `simulateScheduleAD` function. These solutions include essential data such as pressures and flow rates at various wells. Additionally, `plotWellSols` utilizes `report.ReservoirTime`, which denotes the specific times at which reservoir states were recorded throughout the simulation. By plotting well solutions, we can analyze well performance across the simulation period, observe trends such as changes in production or injection rates, and identify any issues like declining performance or unexpected anomalies.

To manage and organize visualizations effectively, the `figure` command is used to create new figure windows. This command ensures that new plots are drawn in a separate window, preserving the clarity and integrity of previous plots. This separation is particularly

beneficial when dealing with multiple visualizations, as it allows each figure to be independently customized and analyzed, thus maintaining clarity and ease of interpretation.

For interactive exploration of reservoir states, the `plotToolbar` function provides a useful tool. This function features a toolbar that allows users to interactively explore the states of the reservoir. It works with `G` and `states`. The interactivity offered by `plotToolbar` enables users to navigate through different time steps, adjust visualization settings, and gain a deeper understanding of reservoir dynamics.

Another important aspect of simulation visualization is plotting well locations on the reservoir grid. The `plotWell` function facilitates this by displaying well locations and configurations relative to the reservoir grid. It uses `G` for the grid representation and `schedule.control(1).W` for the well configuration from the initial control step of the simulation schedule. Visualizing wells in relation to the reservoir grid is crucial for understanding their spatial distribution and interactions with the reservoir, which aids in interpreting simulation results within the context of the reservoir's physical layout.

Finally, adjusting the appearance of plots is essential for enhancing readability and presentation. The `axis tight` command modifies the axis limits to fit the data closely within the plot, ensuring that the plot fits neatly within the figure window. The `view(-10, 60)` command sets the viewing angle for 3D plots, with -10 degrees for the azimuthal angle (rotating the plot around the vertical axis) and 60 degrees for the elevation angle (tilting the plot up or down). These adjustments improve the clarity of the plot, making it easier to interpret complex data.

```
Load SPE1_smry
inj = find([wellSols{1}.sign] == 1);
prod = find([wellSols{1}.sign] == -1);
ind = 2:size(smry.data,2);
[qWs, qOs, qGs, bhp] = wellSolToVector(wellSols);
T = convertTo(cumsum(schedule.step.val), year);
```

Effective analysis of simulation data relies on systematically loading, processing, and interpreting the results. This section outlines the key steps involved in preparing simulation summary data for further analysis and visualization.

To begin with, the load `SPE1_smry` command is used to load the `SPE1_smry` file, which contains summary data from a prior SPE1 reservoir simulation. This summary data encompasses critical results and performance metrics from the simulation, such as production rates, pressures, and other key indicators. The information from this file is instrumental for comparison with current results and for conducting detailed analyses.

Identifying the roles of different wells is crucial for effective reservoir management. The `find([wellSols{1}.sign] == 1)` function locates the indices of injector wells in the well solutions from the first time step. Similarly, `find([wellSols{1}.sign] == -1)` identifies the indices of producer wells. The sign property of well solutions typically indicates whether a well is an injector (with a value of 1) or a producer (with a value of -1). Differentiating between these well types is essential for analyzing their respective performances, as injectors and producers fulfill distinct roles in reservoir operations. Injectors are responsible for maintaining pressure by injecting fluids, while producers extract hydrocarbons from the reservoir.

The next step involves selecting relevant data columns from the summary file. The command `2:size(smry.data,2)` generates an array of indices starting from 2 up to the number of columns in `smry.data` . This approach is used to select columns of interest, possibly excluding the first column, which may contain headers or irrelevant initial data. By focusing on the pertinent columns, subsequent analysis can be more precise and relevant.

To facilitate the analysis of well solutions, it is useful to convert them into a vector form. The `wellSolToVector` function is employed for this purpose, transforming well solutions into vectors for water flow rates (`qWs`), oil flow rates (`qOs`), gas flow rates (`qGs`), and bottom-hole pressures (`bhp`). Converting well solutions to vectors simplifies data handling, making it easier to perform plotting, statistical analysis, and comparisons with other datasets.

Furthermore, translating simulation time into a more interpretable format is achieved through the `convertTo` function. This function converts cumulative simulation time into years using `cumsum(schedule.step.val)` to compute the cumulative sum of simulation step values. The result is an array of time values converted to years, which provides a more intuitive understanding of the simulation duration and aligns the simulation results with real-world time frames.

```
figure(3)
clf
ecl = convertFrom(smry.get('PRODUCER', 'WGOR', ind), 1000*ft^3/stb)';
mrst = qGs(:,prod)./qOs(:,prod);
mrstarg = {'LineWidth', 2};
eclarg = {'ro','MarkerSize',5,'MarkerFaceColor',[.6 .6 .6]};
hold on
plot(T, mrst, mrstarg{:})
plot(Tcomp, ecl, eclarg{:});
legend({'MRST', 'Eclipse'})
xlabel('Time [Years]')
```

To effectively compare and analyze simulation results, creating a clear and organized figure is essential. This section details the steps involved in generating and visualizing a comparison plot for Gas-Oil Ratios (GOR) derived from MRST and Eclipse simulation data.

The process begins with initializing a new figure window using the command `figure(3)`, which creates or selects the figure window with the identifier 3. To ensure that this figure is ready for new data, the `clf` command is employed to clear any existing plots and graphical elements from the figure. Starting with a clean slate prevents previous data from cluttering the new visualization and ensures that the plot accurately reflects the current data.

Next, the relevant data for analysis is extracted and converted. Using the command `smry.get('PRODUCER', 'WGOR', ind)`, Water-Gas-Oil Ratio (WGOR) data for producer wells is retrieved from the summary file. The variable `ind` specifies the columns of interest. This data is then converted to standard units of cubic feet per stock tank barrel ($ft^3/stb$) using `convertFrom(..., 1000*ft^3/stb)`, ensuring consistency in units for accurate comparison. The data is also transposed to match the required orientation for plotting.

In addition to converting summary data, the Gas-Oil Ratio for MRST is calculated using the well flow rates. Specifically, the gas flow rates for producer wells are extracted from the vector `qGs`, and the oil flow rates are extracted from `qOs`. The GOR is then computed by dividing the gas rates by the oil rates, yielding a key performance metric that reflects the

proportion of gas produced per unit of oil. This ratio is crucial for understanding reservoir behavior and fluid dynamics.

Setting the appropriate plotting arguments enhances the clarity of the visual representation. The plotting arguments for MRST data are defined using `mrstarg`, which specifies a line width of 2. For Eclipse data, `eclarg` specifies a red circle marker with a marker size of 5 and a gray face color. These visual settings help distinguish between MRST and Eclipse data in the final plot.

When plotting the data, the `hold on` command is used to retain the current plot, allowing multiple datasets to be displayed simultaneously without overwriting each other. The `plot(T, mrst, mrstarg{:})` command plots the MRST GOR data against time `T` with the specified arguments, while `plot(Tcomp, ecl, eclarg{:})` plots the Eclipse WGOR data against comparison time `Tcomp`. This side-by-side visualization enables a direct comparison of the two sets of simulation results.

Finally, annotations such as legends, axis labels, and titles are added to the plot for clarity. The `legend({'MRST', 'Eclipse'})` command labels the data series, making it clear which dataset corresponds to MRST and which to Eclipse. The `xlabel('Time [Years]')` command labels the x-axis to indicate that time is measured in years, and `title('Gas rate / Oil rate')` provides a descriptive title for the plot. These annotations enhance the plot's readability and ensure that the data is easily interpretable.

```
figure(4)
clf
ecl = convertFrom(smry.get('PRODUCER', 'WBHP', ind), psia)';
mrst = bhp(:,prod);
hold on
plot(T,     convertTo(mrst, barsa), mrstarg{:})
plot(Tcomp, convertTo(ecl, barsa), eclarg{:});
legend({'MRST', 'Eclipse'})
xlabel('Time [Years]')
ylabel('bar')
title('Bottom-hole pressure (producer)')
```

This segment of the code titled 'Plot producer bottom-hole pressure' is responsible for generating a plot that compares the bottom-hole pressure (BHP) of a producer well as computed by MRST and a commercial simulator, Eclipse.

The code begins by initializing the plot with the command `figure(4)`, which opens or activates a figure window labeled as number 4. This window is specifically designated to display the bottom-hole pressure plot. Following this, the `clf` command is used to clear any existing content within this figure, ensuring that the window is prepared for the new plot.

Next, the line `ecl = convertFrom(smry.get('PRODUCER', 'WBHP', ind), psia)';` is used to retrieve the bottom-hole pressure (WBHP) data for the producer well from the Eclipse simulation. The `smry.get` function extracts the relevant WBHP values over the specified time indices (`'ind'`), and the `convertFrom` function converts these values from their original units to psia (pounds per square inch absolute). The data is then transposed to ensure it is in the correct format for plotting. Similarly, the variable `mrst` is assigned the

MRST-computed bottom-hole pressure values for the producer well, which are stored in the `bhp` matrix, indexed by `prod`.

The `hold on` command is then used to retain the current plot, allowing multiple datasets to be plotted on the same figure without overwriting each other. This is crucial for displaying both the MRST and Eclipse data together. The first plot is generated with the line `plot(T, convertTo(mrst, barsa), mrstarg{:})`, which plots the MRST-computed bottom-hole pressure against time. Here, T represents the simulation time in years, and `convertTo(mrst, barsa)` converts the MRST bottom-hole pressure data into bars (the unit of pressure). The additional arguments `mrstarg{:}` are used for custom plot formatting, such as adjusting the line width. The second plot is created with the line `plot(Tcomp, convertTo(ecl, barsa), eclarg{:});`, which similarly plots the Eclipse-computed bottom-hole pressure against time using Tcomp for the time values and converting the Eclipse data (`'ecl'`) into bars. The arguments `eclarg{:}` specify additional plot formatting options, like marker style and color.

To finalize the plot, a legend is added with the command `legend({'MRST', 'Eclipse'})`, which distinguishes the lines corresponding to the MRST and Eclipse data. The x-axis and y-axis of the plot are labeled using `xlabel('Time [Years]')` and `ylabel('bar')`, indicating that the x-axis represents time in years and the y-axis represents pressure in bars. The plot is given the title `Bottom-hole pressure (producer)`, clearly indicating that it shows the bottom-hole pressure for the producer well.

This code segment ultimately creates a comparative plot of the bottom-hole pressure for the producer well, allowing for a visual inspection of the results from MRST and Eclipse to assess the accuracy and consistency of the MRST simulation.

```
figure(5)
clf
ecl = convertFrom(smry.get('INJECTOR', 'WBHP', ind), psia)';
mrst = bhp(:,inj);
hold on
plot(T,     convertTo(mrst, barsa), mrstarg{:})
plot(Tcomp, convertTo(ecl, barsa),  eclarg{:});
legend({'MRST', 'Eclipse'})
xlabel('Time [Years]')
ylabel('bar')
title('Bottom-hole pressure (injector)')
```

This segment of the code is focused on generating a plot that compares the bottom-hole pressure (BHP) of an injector well as computed by MRST and a commercial simulator, Eclipse.

The process begins with the `figure(5)` command, which opens or activates a figure window labeled as number 5. This window is dedicated to displaying the bottom-hole pressure plot for the injector well. The `clf` command follows, clearing any existing content within this figure to ensure that it is ready for the new plot.

The line `ecl = convertFrom(smry.get('INJECTOR', 'WBHP', ind), psia)';` is then used to retrieve the bottom-hole pressure data for the injector well from the Eclipse simulation. The `smry.get` function extracts the WBHP values for the injector over the specified time indices (`'ind'`), and the `convertFrom` function converts these values from their original units to psia (pounds per square inch absolute). The data is then transposed

to ensure it is formatted correctly for plotting. Similarly, the variable `mrst` is assigned the MRST-computed bottom-hole pressure values for the injector well, which are stored in the `bhp` matrix, indexed by `inj`.

The `hold on` command is used next to retain the current plot, allowing multiple datasets to be plotted on the same figure without overwriting each other. This is necessary for displaying both the MRST and Eclipse data together. The MRST-computed bottom-hole pressure is plotted against time with the command `plot(T, convertTo(mrst, barsa), mrstarg{:})`. Here, `T` represents the simulation time in years, and `convertTo(mrst, barsa)` converts the MRST bottom-hole pressure data into bars. The additional arguments `mrstarg{:}` are used to apply custom formatting, such as adjusting the line width. The Eclipse-computed bottom-hole pressure is then plotted with `plot(Tcomp, convertTo(ecl, barsa), eclarg{:});`, where `Tcomp` represents the time values and `convertTo(ecl, barsa)` converts the Eclipse data into bars. The arguments `eclarg{:}` specify additional plot formatting options, like marker style and color.

A legend is added to the plot with the command `legend({'MRST', 'Eclipse'})`, which helps distinguish between the lines representing MRST and Eclipse data. The axes of the plot are labeled using `xlabel('Time [Years]')` and `ylabel('bar')`, indicating that the x-axis represents time in years and the y-axis represents pressure in bars. The plot is titled `Bottom-hole pressure (injector)`, clearly indicating that it displays the bottom-hole pressure for the injector well.

This code segment ultimately creates a comparative plot of the bottom-hole pressure for the injector well, allowing for a visual inspection of the results from MRST and Eclipse to evaluate the accuracy and consistency of the MRST simulation.

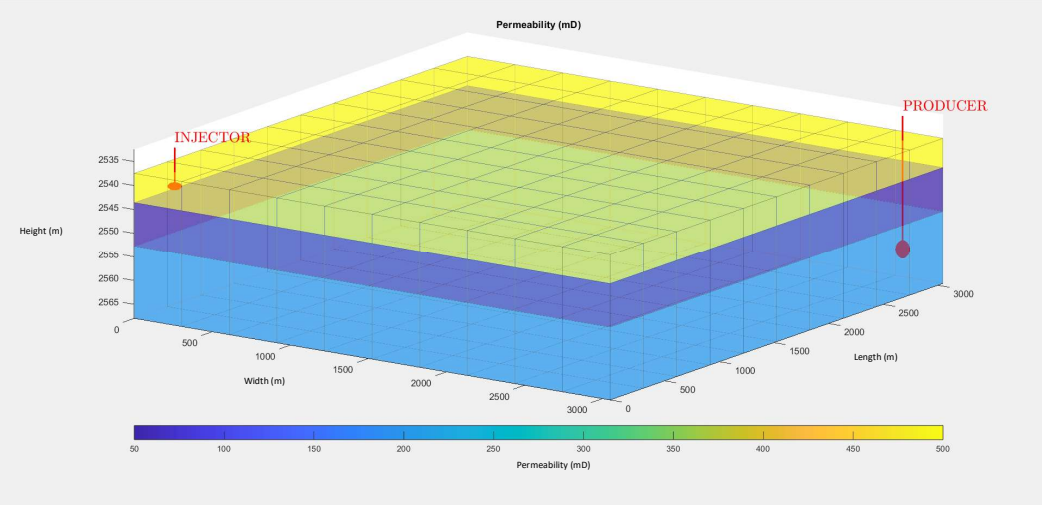


**Figure 52: Reservoir simulation grid.**

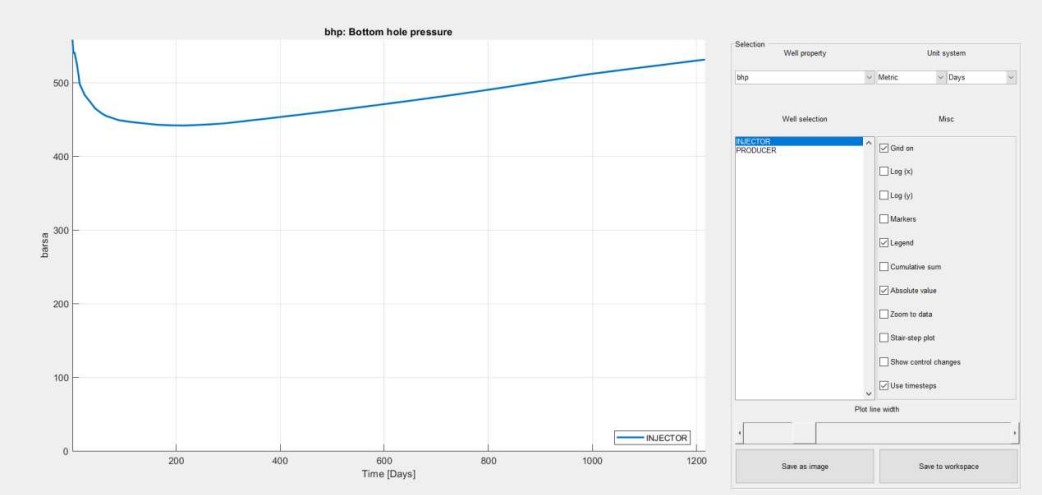


**Figure 53: Diagram of Bottom hole pressure - Time for the Injector.**

The curve presented in the diagram above illustrates the evolution of bottom hole pressure (bhp) for the injector well over time. Initially, a sharp pressure drop is observed: from approximately 540 bar, the pressure rapidly decreases within the first 50 days, reaching around 440 bar. This behavior may be linked to the reservoir's response to the initial operating conditions, such as the sudden fluid. Subsequently, and until around 400$^{th}$ day, the pressure remains relatively stable, fluctuating within the 430–440 bar range, indicating a gradual stabilization of the system. After 400$^{th}$ day, the curve exhibits a gradual but consistent increase in bottom hole pressure. From around 600$^{th}$ day onward, the upward trend becomes more pronounced, with pressure reaching approximately 510 bar by the end of the simulation period, near 1200$^{th}$ day. This behavior could be attributed to changes in injection flow rates, increased flow resistance within the reservoir, or adjustments in the operational strategy of the well. The overall shape of the curve reflects the dynamic interaction between the injection well and the reservoir characteristics throughout the simulation period.

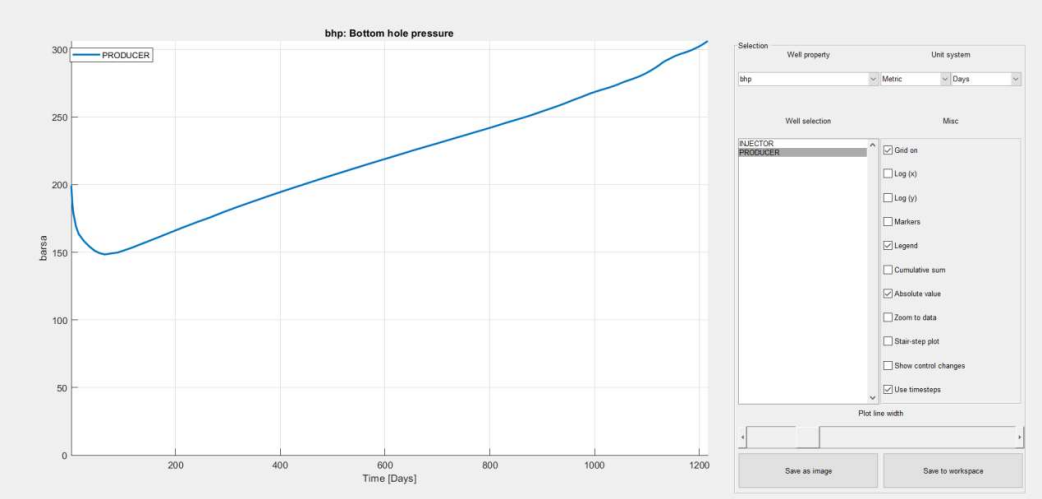


**Figure 54: Diagram of Bottom hole pressure -Time for the Producer.**

The curve presented in the diagram above illustrates the evolution of bottom hole pressure (bhp) for the producer well over time. Initially, a sharp pressure drop is observed: from approximately 200 bar, the pressure rapidly decreases within the first 50 days, reaching around 150 bar. This behavior may be linked to the reservoir's response to the initial operating conditions, such as the sudden gas injection. After 50$^{th}$ day, the curve exhibits a gradual but consistent increase in bottom hole pressure, with pressure reaching approximately 300 bar by the end of the simulation period, near 1200$^{th}$ day. This behavior could be attributed to changes in injection flow rates, increased flow resistance within the reservoir, or adjustments in the operational strategy of the well. The overall shape of the curve reflects the dynamic interaction between the production well and the reservoir characteristics throughout the simulation period.

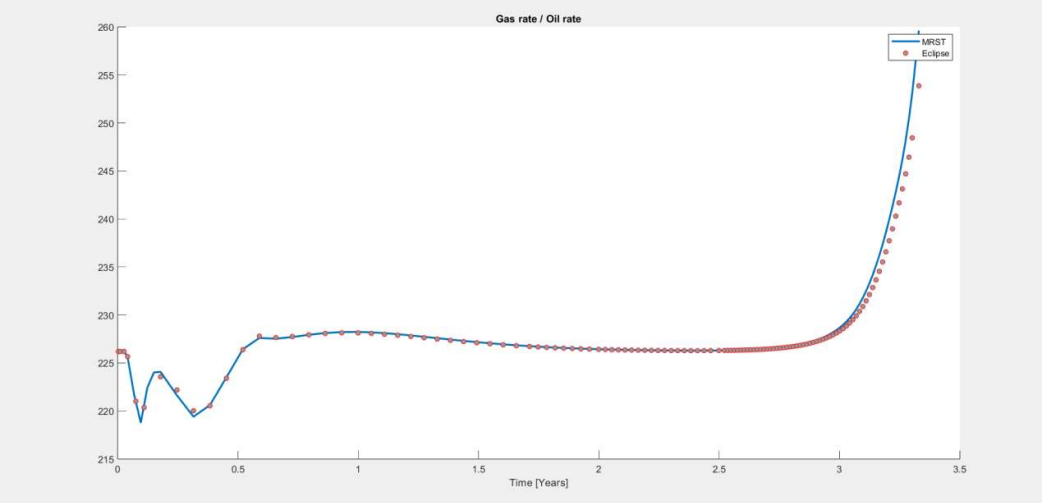


**Figure 55: Diagram of Gas to Oil ratio - Time.**

The GOR curve as a function of time shows significant fluctuations in the production of gas and oil from the reservoir. Specifically, for the period from 0 to 0.6 years (i.e., from 0 to 219 days), there is intense variation in the GOR values, which increase and decrease sharply. Specifically, for the period from 0 to 0.1 years (0 to 36.5 days), the initial GOR value is 226 scf/stb and drops to 219 scf/stb, registering an initial decrease. Then, for the period from 0.1 to 0.2 years (36.5 to 73 days), there is an increase in the GOR value, from 219 scf/stb to 224 scf/stb. This is followed by a period of decrease from 0.2 to 0.4 years (73 to 146 days), where the GOR reaches 220 scf/stb. After this fluctuation, the GOR value gradually increases for the period from 0.4 to 0.6 years (146 to 219 days), reaching 228 scf/stb. For the period from 0.6 to 3 years (219 to 1,095 days), the GOR value stabilizes, fluctuating around 227 scf/stb. Finally, for the period from 3 to 3.3 years (1,095 to 1,204.5 days), there is a sharp increase in GOR values, with the value starting at 227 scf/stb and rising to 260 scf/stb.

The physical significance of these changes is related to various factors affecting the reservoir. The intense fluctuations in GOR during the early stages of production (0 to 0.6 years) are likely due to changes in the reservoir's pressure and temperature. These changes affect how much gas dissolves in the oil and how the gas is released to the surface. The initial decrease in GOR is likely because the gas that was dissolved in the oil begins to be released as pressure drops, causing the gas to separate from the oil. Similarly, the stabilization of GOR from 0.6 to 3 years suggests that the reservoir has reached a steady production rate, with stable pressure and temperature conditions. After 3 years, a sharp increase in GOR is observed. This

corresponds to the gas breakthrough at the producer, due to continuous gas injection. The breakthrough leads to a disproportionate rise in gas production compared to oil, significantly raising the GOR and signaling reduced displacement efficiency and the onset of gas channeling.

The two diagrams below depict the bottom-hole pressure as a function of time (in years) for a producer and an injector well, respectively. The results compare simulations carried out using MRST (blue line) and the commercial software Eclipse (red circles), highlighting the consistency and accuracy of MRST.

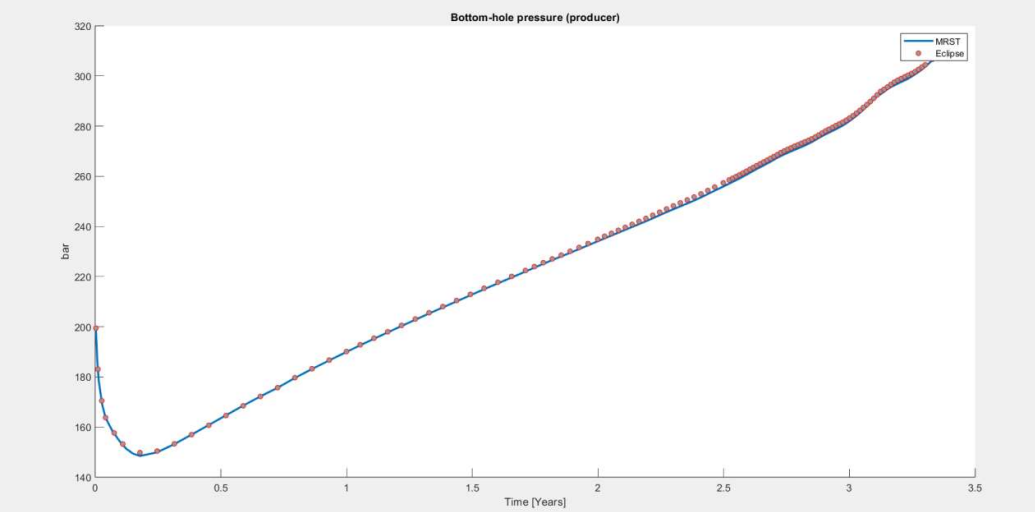


**Figure 56: Diagram of BHP - Time for the Producer.**

According to the diagram of Figure 56, the producer's pressure starts at approximately 195 bar and initially experiences a sharp decline down to around 145 bar just before 0.2 years. It then increases almost linearly, gradually reaching about 310 bar at the end of the simulation period (3.5 years). The comparison shows that MRST and Eclipse results align very closely, with only minor deviations appearing after 3 years.

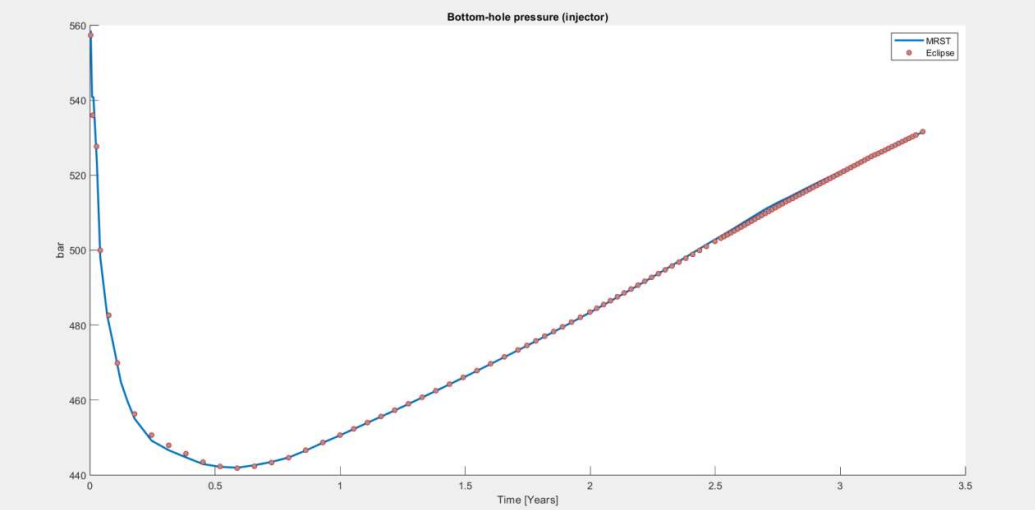


**Figure 57: Diagram of BHP- Time for the Injector.**

In this diagram (Figure 57), the pressure begins high, close to 560 bar, and sharply decreases until about 0.5 years, reaching a low of around 445 bar. From that point on, it gradually increases, reaching approximately 540 bar by year 3.5. As in the previous diagram, the MRST and Eclipse curves follow nearly identical trajectories, confirming the reliability of the simulation.

### 3.4.3 Comparison of SPE 1 Simulation for different Injection rates.

In this subsection, we read and process the code named 'BENCH_SPE1.DATA', which serves as the data source for the SPE1 simulation. The reason for examining the BENCH_SPE1.DATA file is to gain a deeper understanding of the SPE1 code and to observe how the reservoir responds to variations in the injection rate (STB/day). As will be understood, the following code modifications in the Bench code were made in line 394, where the initial value of 20,000 STB/day was changed to 25,000 STB/day and 30,000 STB/day.

The figures presented here are based on the initial conditions of the Bench code, meaning for an input flow rate of 20,000 STB/day.

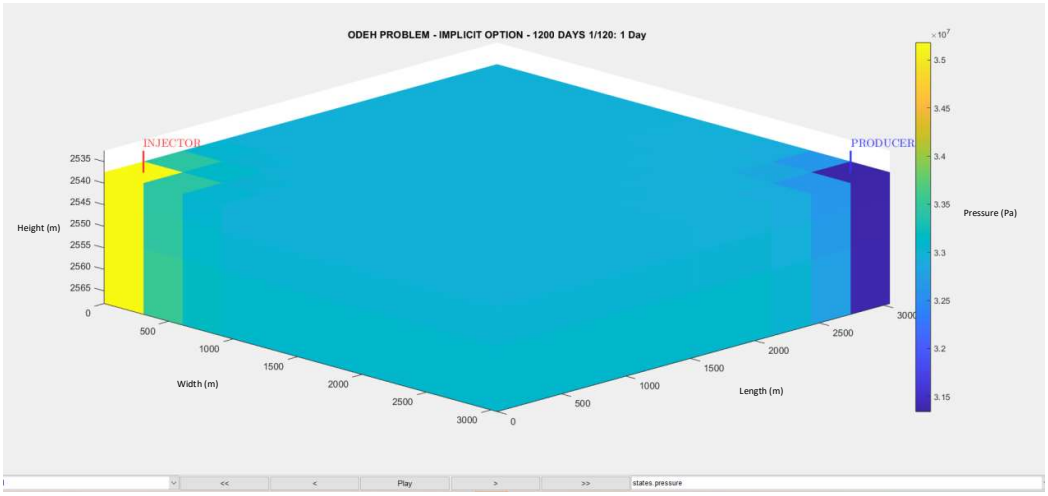


**Figure 58: Phase field dynamics of the reservoir (1st time step-beginning – 1st day).**

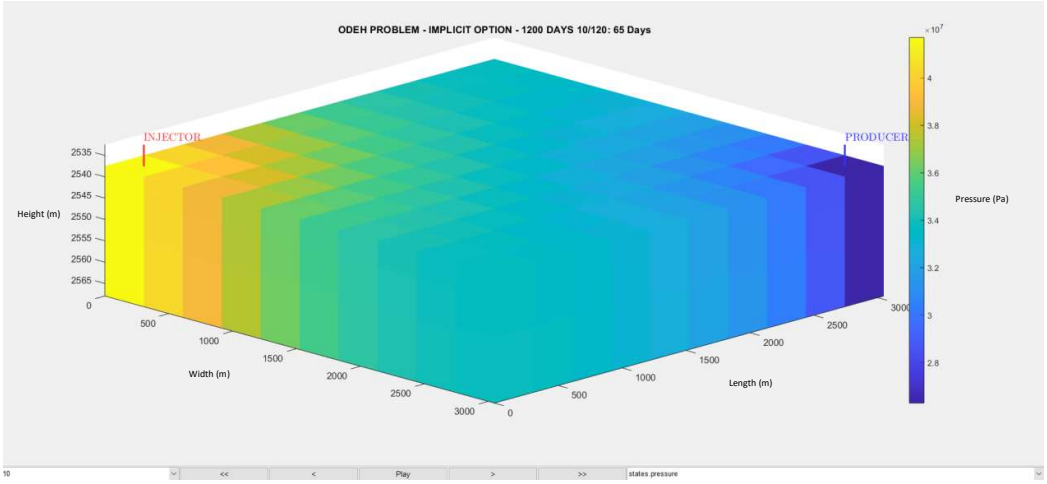


**Figure 59: Phase field dynamics of the reservoir (10th time step – 65th day).**

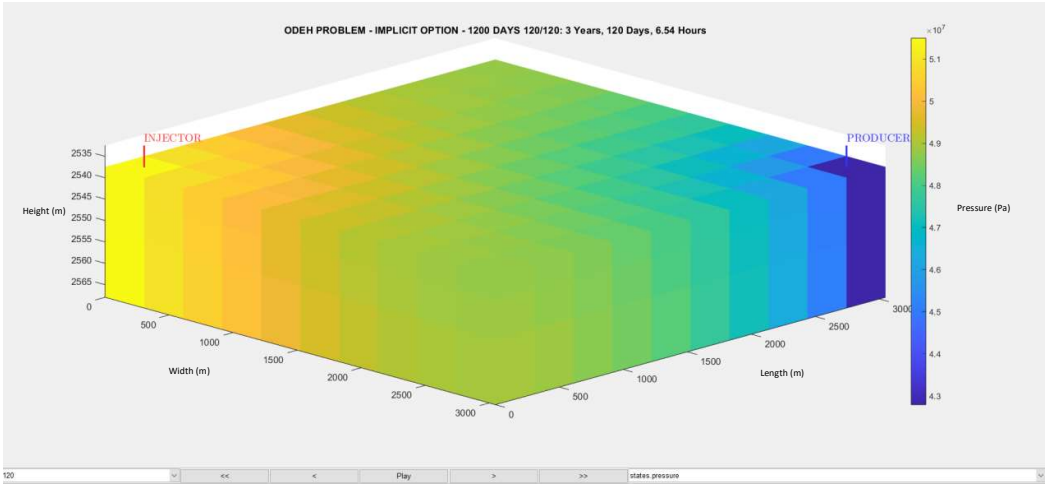


**Figure 60: Phase field dynamics of the reservoir (120th time step – 3 years and 120th days).**

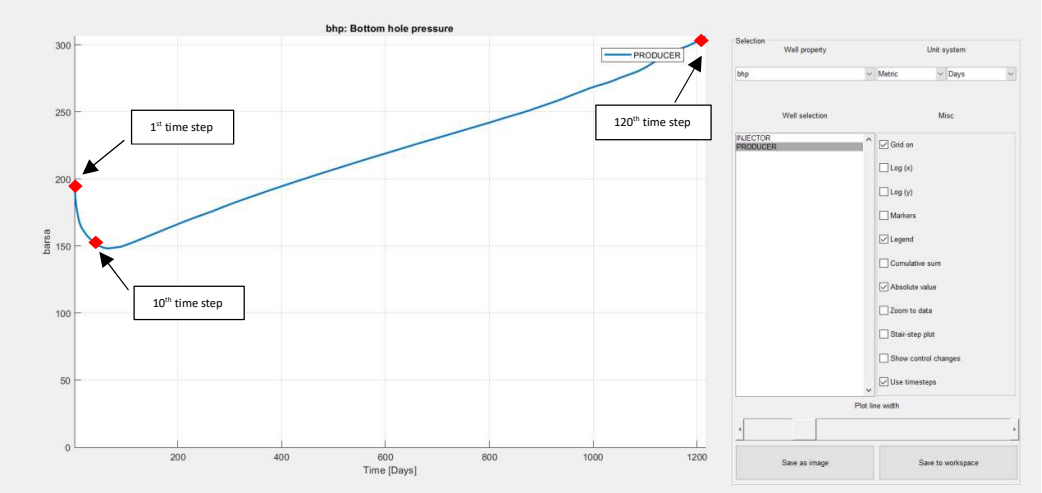


**Figure 61: BHP - Time for the Producer.**

The following figures are based on a change to the initial conditions of the Bench code, specifically for an increase in the input flow rate from 20,000 STB/day to 25,000 STB/day.

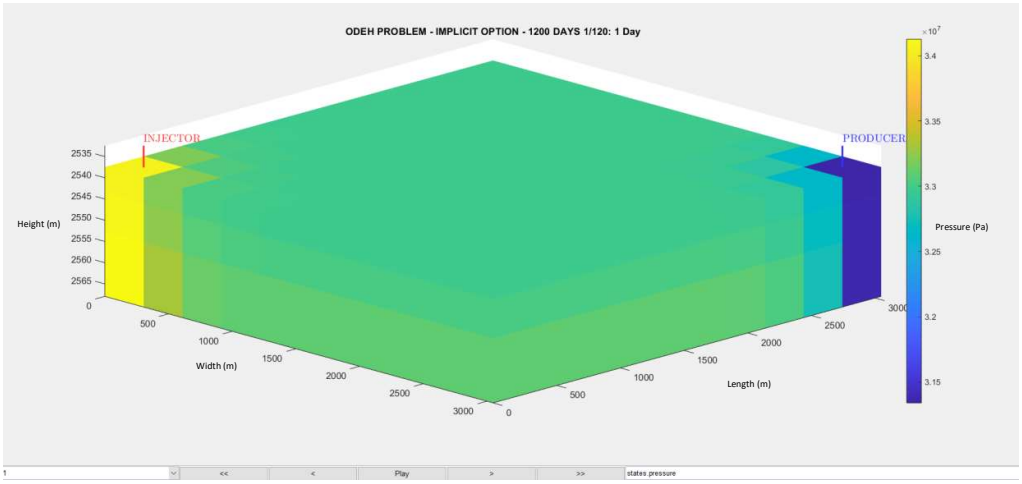


**Figure 62: Phase field dynamics of the reservoir (1st time step – 1st day).**

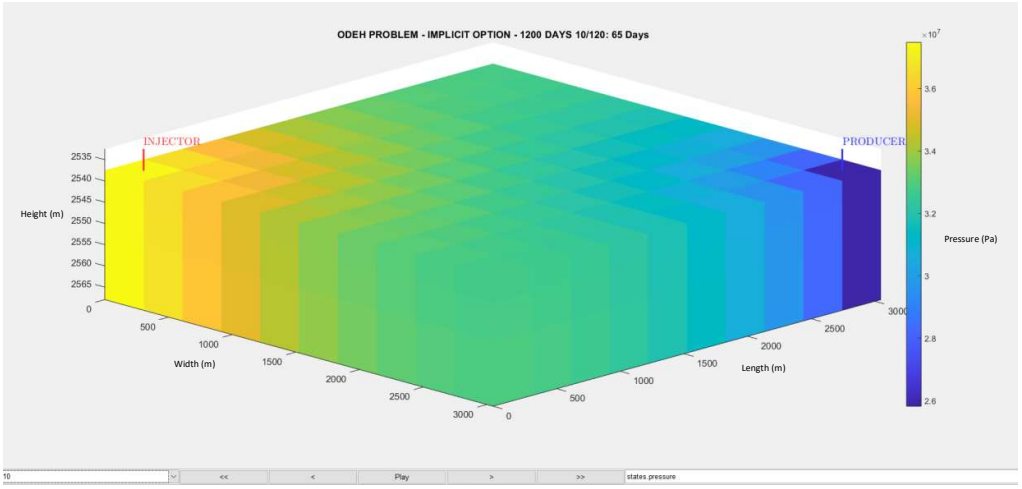


**Figure 63: Phase field dynamics of the reservoir (10th time step – 65th day).**

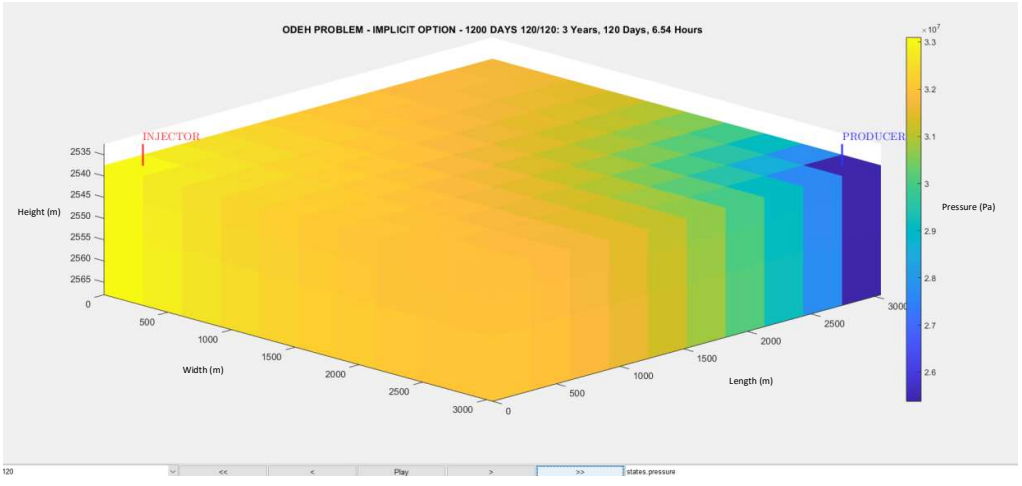


**Figure 64: Phase field dynamics of the reservoir (120th time step – 3 years and 120 days).**

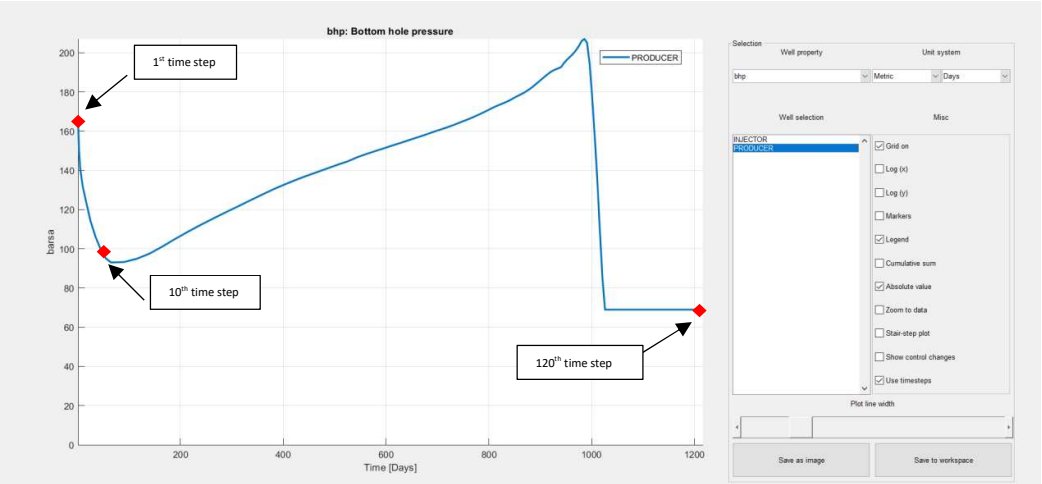


**Figure 65: BHP-Time for the Producer.**

In the same manner the input flow rate was changed from 20,000 STB/day to 30,000 STB/day, resulting in the following:

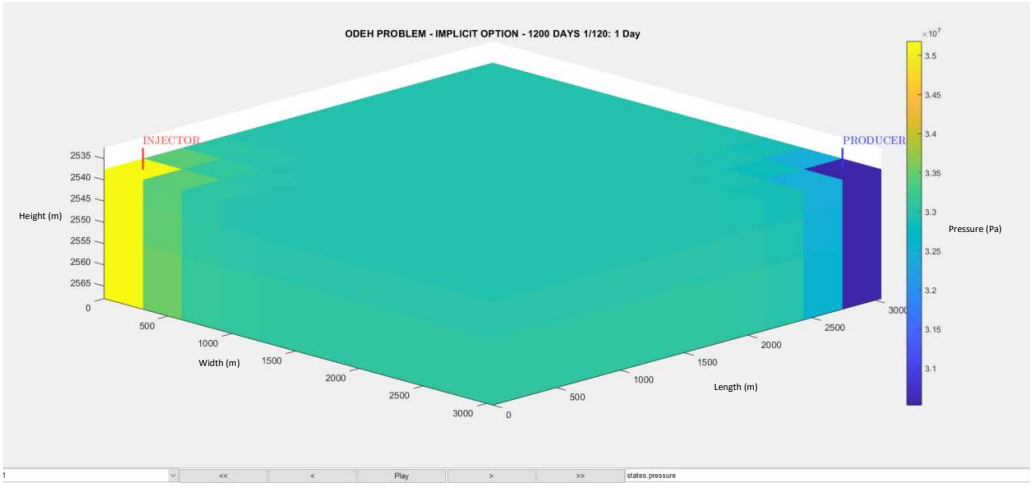


**Figure 66: Phase field dynamics of the reservoir (1st time step – 1st day).**

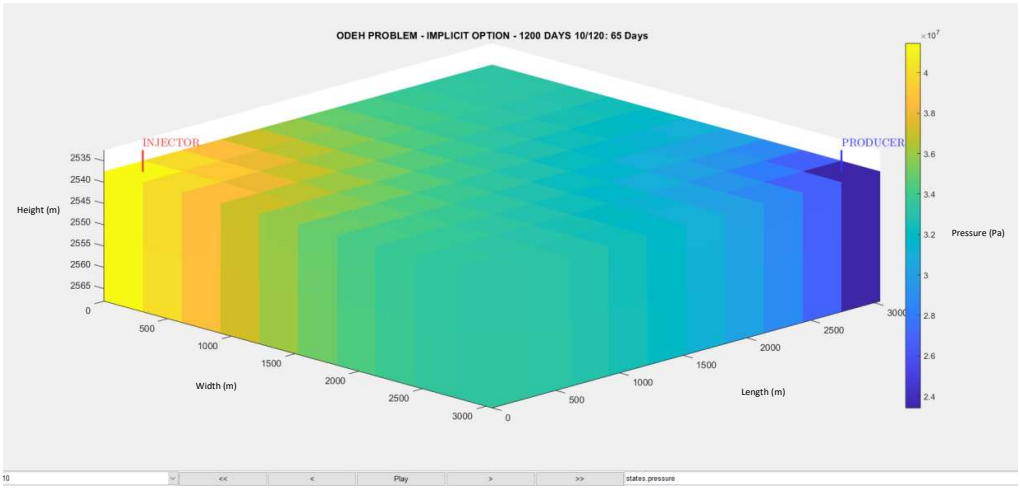


**Figure 67: Phase field dynamics of the reservoir (10th time step – 65th day).**

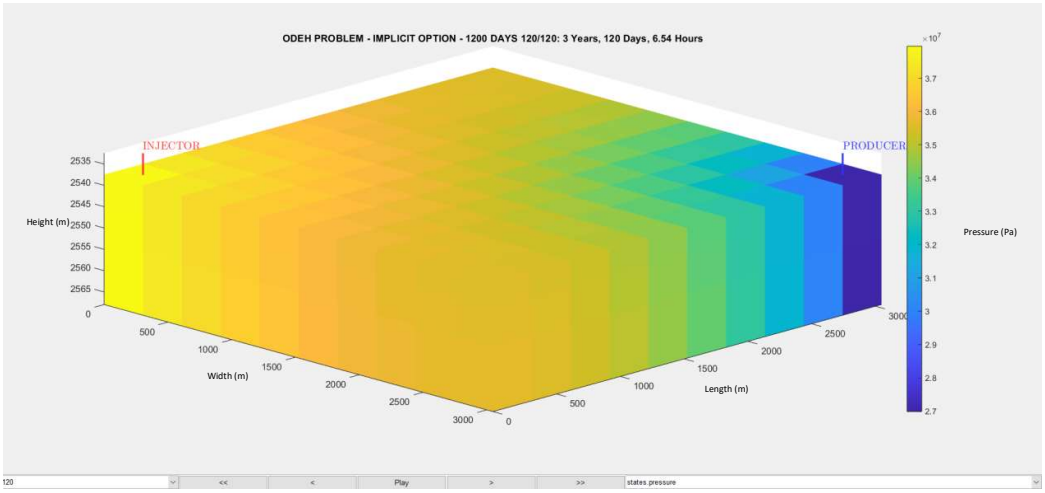


**Figure 68: Phase field dynamics of the reservoir (120th time step – 3 years and 120 days).**

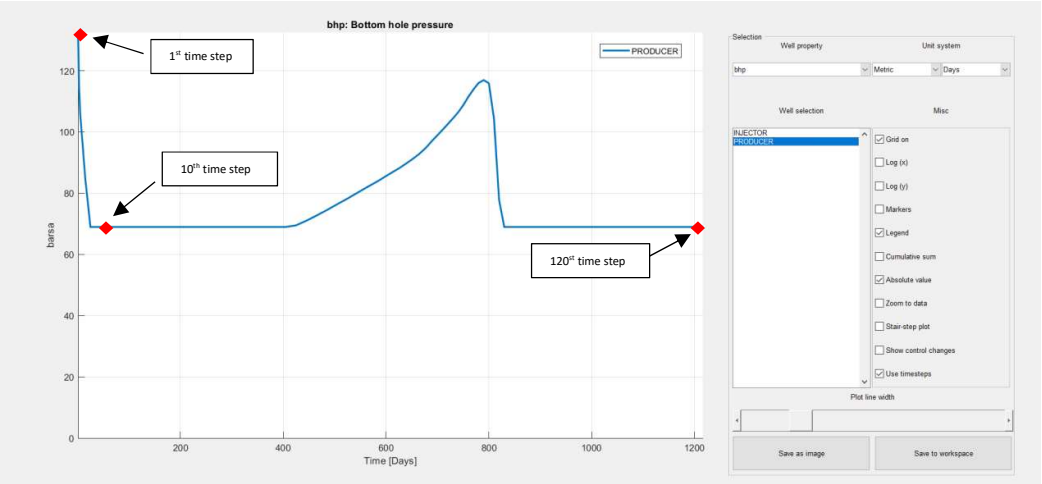


**Figure 69: BHP - Time for the Producer.**

The resulting BHP diagrams will now be compared to each other.

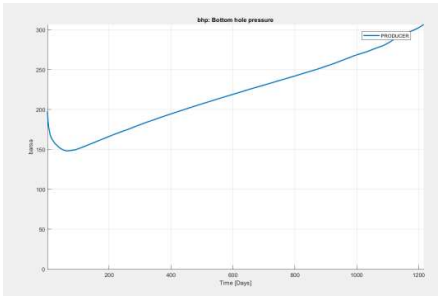


**Figure 70: BHP- Time for rate 20000 STB/day for the Producer.**

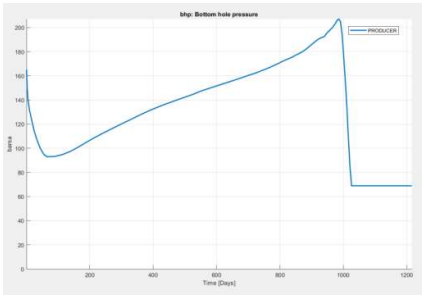


**Figure 71: BHP- Time for rate 25000 STB/day for the Producer.**

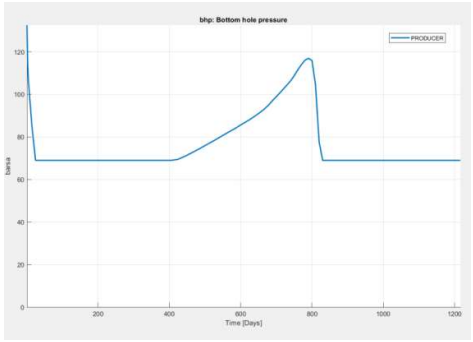


**Figure 72: BHP- Time for rate 30000 STB/day for the Producer.**

The BHP trends observed in all cases follow a general pattern: an initial decline, followed by a gradual increase, and, in the cases of higher production rates, a sudden drop. The

initial pressure reduction is attributed to reservoir depletion as production begins, while the subsequent increase indicates the effect of gas injection maintaining reservoir pressure. However, significant differences arise depending on the production rate, particularly in the later stages of production.

At a production rate of 20.000STB/day, BHP follows a relatively stable trend, with no abrupt fluctuations. The steady increase in BHP suggests that gas injection effectively supports the reservoir pressure, preventing rapid pressure declines. This indicates a controlled production environment where fluid withdrawal is balanced with pressure maintenance.

In contrast, at 25.000 and 30.000 STB/day production rates, BHP exhibits a sharp drop at approximately 1.000 and 800 days, respectively. This sudden decrease strongly suggests gas breakthrough, a phenomenon in which injected gas reaches the producer well and significantly alters fluid properties. As gas has a much lower density than reservoir fluids, its breakthrough leads to a reduction in hydrostatic pressure within the wellbore, causing the observed BHP collapse.

Gas injection plays a crucial role in sustaining reservoir pressure. In the early stages, it counteracts the pressure decline associated with production. However, as the production rate increases, the reservoir experiences greater pressure drawdown, accelerating gas migration from the injector to the producer.

The sharp BHP drops observed in the 25.000 and 30.000 STB/day cases indicate that free gas has reached the production well, leading to a sudden decrease in fluid column pressure. This suggests that higher production rates increase the likelihood of premature gas breakthrough, which can negatively impact oil recovery by reducing the well's ability to sustain liquid flow.

The analysis highlights the importance of optimizing production rates to balance hydrocarbon recovery while minimizing premature gas breakthrough. Lower production rates (e.g., 20.000STB/day) result in more stable BHP trends, indicating a well-controlled pressure regime. Conversely, higher production rates (25.000 and 30.000STB/day) lead to earlier and more severe gas breakthrough, as evidenced by the sharp BHP declines.

Further investigation is necessary to fully characterize the impact of gas injection, including an analysis of gas saturation distribution and gas-oil ratio (GOR) trends. Additionally, flow diagnostics could provide further insights into the dynamics of gas migration and breakthrough timing under different production scenarios.

# Chapter 4: The Water Alternating Gas (WAG) Simulation

The WAG (Water-Alternating-Gas) problem arises from the quest to find the optimal production strategy for reservoirs that utilize both water and gas injection for secondary oil recovery. The objective is to maximize oil production while minimizing costs and optimizing the efficiency of the recovery process. In this case, the reservoir is modeled with dimensions of 10 x 10 x 10 cells, covering a total area of 1000 x 1000 x 10 meters. The system includes 4 injectors and 1 producer well, with all five wells having the same. The injectors are placed at the four corners of the reservoir, while the producer is located at the center of the reservoir. The primary physical process under investigation is the alternating injection of water and gas into the reservoir, a method that aims to improve the displacement efficiency of oil by maintaining reservoir pressure and improving sweep efficiency. Additionally, the reservoir's porosity is randomly distributed across the grid, which introduces variability and complexity to the modeling process.

The key assumptions governing the WAG Simulations are the following:

- Three-phase immiscible flow (water, oil, gas) with no gas dissolution or oil vaporization (`disgas = false`, `vapoil = false`).
- The fluid model uses a three-phase immiscible model with quadratic Corey relative permeabilities ($k_{ri} \propto S_i^2$) and constant viscosities.
- The Kozeny-Carman equation used in the provided MRST codes for the WAG problem is a modified form to compute permeability ($k$) based on porosity ($\phi$).

(Blunt et al., 2001, Chen, n.d., Lie et al., 2012)

Based on the WAG problem, we created another two codes which isolate the injection of water and gas. The need to create the modified codes stems from the desire to study specific injection scenarios within the same reservoir as the original WAG problem. This approach allows for a more detailed investigation into the individual effects of water and gas injection on reservoir behavior and production efficiency. The reservoir, still modeled with the same dimensions retains the same rock properties, fluid characteristics, and well placements as the original WAG model. The objective now is to study the impact of water injection alone, without the interference of gas and vice versa to better understand how each injection affects oil displacement, pressure maintenance, and sweep efficiency.

Additionally, this isolated scenarios allow for comparisons to be made with the WAG problem, helping to identify which injection method, or combination of methods, results in the most effective production strategy. By isolating each injection method, we can better evaluate the unique challenges and advantages associated with each, contributing to more informed decision-making in reservoir management.

As seen in the following simulation (5.2), two numerical methods are employed to approach the solution of the reservoir flow equations: the Full Implicit (FIMP) and the Sequential method. Those are numerical techniques used in reservoir simulation to solve the coupled equations governing fluid flow. The FIMP method simultaneously solves the pressure and saturation equations, providing superior numerical stability and accuracy, particularly for complex scenarios like three-phase flow or enhanced oil recovery (EOR) processes. However, its computational cost is high due to the need to solve a large, nonlinear system of equations. Conversely, the Sequential method decouples the solution process, first solving the pressure

equation and then the saturation equations independently. This approach is computationally efficient and easier to implement but may compromise accuracy and stability, especially in highly coupled systems. The choice between FIMP and Sequential methods depends on the problem's complexity, required accuracy, and available computational resources.

## 4.1 Governing Equations of the Simulation

### 4.1.1 Mass Conservation

$$\frac{\partial}{\partial t}(\phi \rho_i S_i) + \nabla \cdot (\rho_i v_i) = q_i, i = \{w, o, g\}$$

Where:

- $\phi$: Porosity
- $\rho_i$: Density
- $S_i$: Saturation
- $v_i$: Darcy velocity (volumetric flow per unit area)
- $q_i$: Injection rate

(Durlofsky & Aziz, 2010, Peaceman, 1977)

### 4.1.2 Kozeny-Carman

$$k = \frac{\phi^3}{c \cdot S_s^2 \cdot (1-\phi)^2}$$

Where:

- k: Permeability ($m^2$)
- $\phi$: Porosity
- c: Kozeny constant
- $S_s$ : Specific surface area per unit volume of the solid grains ($m^{-1}$)
- $(1-\phi)^2$: Accounts for the solid fraction, affecting tortuosity and flow paths

In the codes this is then modified to: $k = \frac{\phi^3 \cdot (5 \cdot 10^{-5})^2}{0.81 \cdot 72 \cdot (1-\phi)^2}$

Where:

- The pore size is set to $(5 \cdot 10^{-5})^2 = 2.5 \cdot 10^{-9} m^2$ (typical sandstone grain size ~50µm)
- The Kozeny constant is set to $0.81 \cdot 72 = 58.32$
- Those values ensure a permeability range for 10 to 1000mD, indicative of a sandstone reservoir

(Carlson, 1997, Bear, 1972)

### 4.1.3 Darcy's Law

$$v_i = -\frac{k_{ri}}{\mu_i} K\nabla P_i, i = \{w, o, g\}$$

Where:

- $k_{ri}$: Relative permeability (modeled with Corey model: $k_{ri}=k_{ri}^0 S_i^n, n = 2$)
- $\mu_i$: Viscosity
- K: Absolute permeability tensor (derived from Kozeny-Carman Relation)
- $P_i$: Pressure (related via capillary pressure: $P_c = P_o - P_w, P_c = P_g - P_o$, but negligible here for simplicity)

(Durlofsky, n.d.)

### 4.1.4 Pressure Equation

$$\phi c_t \frac{\partial P}{\partial t} + \nabla \cdot (\sum_i \lambda_i K\nabla P) = \sum_i q_i , i = \{w, o, g\}$$

Where:

- $c_t$: Total compressibility (rock and fluid)
- $\lambda_i$: Phase Mobility ($\lambda_i=\frac{k_{ri}}{\mu_i}$)
- P: Reference Pressure

(Peaceman, 1977)

### 4.1.5 Saturation Equations

$$\phi \frac{\partial S_i}{\partial t} + \nabla \cdot (f_i v_t) = \frac{q_i}{\rho_i}, i = \{w, o, g\}$$

Where:

- $f_i$: Fractional flow function ($f_a = \frac{\lambda_a}{\sum_\beta \lambda_\beta}$)
- $v_t$: Total velocity ($v_t = \sum_i v_i$)

The saturation equations for water and gas are solved, with oil saturation computed as $S_o = 1 - S_w - S_g$

(Buckley & Leverett, 1942, Durlofsky & Aziz, 2010)

### 4.1.6 Well Models

- Injectors (rate controlled): $q_i = Q_{inj} \cdot compi_i, i = \{w, o, g\}$
- Producer (pressure controlled): $q_i = -WI \cdot \lambda_i \left(P_{bhp} - P_{cell}\right), i = \{w, o, g\}$

Where:

- WI: Well Index (computed based on well radius and permeability)
- $P_{bhp}$: Bottom Hole Pressure (set at 100bar)
- $P_{cell}$: Grid Cell Pressure

(Ewing, 1978)

### 4.1.7 Fluid Properties

- Density: $\rho_i = \rho_i^0 e^{\left(c_i (P - P_{ref})\right)}, i = \{o, g\}$ & $\rho_w$= 1000 kg/m$^3$

  (McCain, 1990)
- Viscosity: Constant for each phase
- Relative Permeability: Quadratic Corey Model: $k_{ra} = k_{ra}^0 \cdot \left(\frac{S_a - S_{ar}}{1 - \sum_\beta S_{\beta r}}\right)^{n_a}, a = \{w, o, g\}$

Where:

- $k_{ra}$: Relative permeability
- $k_{ra}^0$: Endpoint relative permeability (maximum value for maximum saturation, usually 1 for simplicity)
- $S_a$: Saturation
- $S_{ar}$: Residual saturation
- $1 - \sum_\beta S_{\beta r}$: Maximum mobile saturation range, accounting for residual saturations of all phases
- $n_a$: Corey exponent, is set to 2 where relative permeability increases more gradually at low saturations and more rapidly as saturation approaches the maximum mobile value. A quadratic model ($n$ = 2) provides a smooth, parabolic curve that balances realism and simplicity, suitable for modeling three-phase flow in a homogeneous or mildly heterogeneous reservoir.

This model describes how the permeability of a porous medium to a specific phase is reduced due to the presence of other phases, as a function of phase saturation. Below, I explain the quadratic Corey relative permeability model, its implementation in the codes, and the relevant equations, assumptions, and context.)

Some simplifications are implemented:

- $k_{ra}^{0} = 1$, meaning each phase can achieve a maximum permeability of 1 when it fully saturates the mobile pore space
- $S_{ar} = 0$, simplifying the normalized saturation to $S_a$
- No capillary pressure, so phase pressures are equal
- Three phase immiscible flow, so relative permeabilities depend only on phase saturations and not on phase interaction (like gas dissolution)

These conclude to: $k_{ra} = S_a^2$

(Amyx et al., 1960)

### 4.2 Code Analysis of WAG

```
clc;
clearall;
closeall;
```

The `clc` command clears the command window. It removes all the text and output previously displayed, making the command window empty. This is often done at the start of a script to provide a clean output screen for the results of the script's execution. It does not affect variables or functions in the workspace. The `clear all` command clears all variables, functions, and MEX files from the MATLAB workspace. This is useful to ensure that no leftover variables or functions from previous runs are present in the workspace, which could cause conflicts or unexpected behavior. This command ensures a fresh start by removing everything stored in the workspace, such as previously defined variables, functions, and loaded files. The `close all` command closes all figure windows that are currently open. This is particularly helpful if you are running a script that generates plots or figures, as it ensures that no old figures or plots remain open, preventing clutter. It provides a clean slate for visualizations generated during the current script execution.

Together, these three commands are used to prepare the MATLAB environment for a clean execution of the script. They ensure that the command window is cleared of any previous outputs, the workspace is emptied of any previous variables or functions and any open figures or plots are closed. This initialization step is particularly useful when running a script multiple times or when working in a shared environment, as it prevents old data or figures from interfering with the current execution.

```
mrstModuleadd ad-core ad-blackoil ad-props spe10 mrst-gui sequential
spe10 mrst-gui
cartDims = [10, 10, 10];
physDims = [1000, 1000, 10]*meter;
G = cartGrid(cartDims, physDims);
G = computeGeometry(G);
Gravity reset on
```

The code starts with the command `mrstModule add`, which loads several essential modules in the MATLAB Reservoir Simulation Toolbox (MRST). These modules are necessary for running a reservoir simulation. The modules included are:

`ad-core`: This module provides the core functionality for automatic differentiation. It allows the simulation to compute derivatives automatically, which is important for optimization, inversion, and sensitivity analysis in reservoir modeling.
`ad-blackoil`: This module is used to simulate a black oil reservoir. It contains the necessary routines for modeling fluid flow in oil reservoirs with three phases (oil, water, and gas).
`ad-props`: This module is responsible for calculating the reservoir's properties such as permeability and porosity. These properties are crucial for the simulation of fluid flow through the reservoir rock.
`spe10`: This module provides a specific benchmark dataset, the SPE10 (Society of Petroleum Engineers) model, which is often used for testing and comparing simulation methods. It includes predefined grid configurations and reservoir properties.
`mrst-gui`: This module provides the graphical user interface for visualizing the simulation results. It makes it easier to interpret the results by providing interactive visualizations of reservoir properties, fluid flow, and other relevant data.
`sequential`: This module is used for running simulations using sequential methods, which are less computationally demanding than fully implicit methods. Sequential simulation is often used in reservoir modeling to speed up calculations.

After loading these modules, the code proceeds to set up the simulation grid. The variable `cartDims = [10, 10, 10]` defines the grid's size in terms of the number of cells in each direction. Here, the grid is defined with 10 cells in the x, y, and z directions, meaning the grid has a total of 1,000 cells (10 × 10 × 10). These cells represent the discrete volume elements of the reservoir, each of which will be assigned a set of physical properties. The variable `physDims = [1000, 1000, 10]*meter` defines the physical dimensions of the grid, specifying that the reservoir will be modeled as a 1000 meter by 1000 meter area, with a depth of 10 meters. The `meter` unit ensures that the dimensions are correctly specified in meters. This configuration provides the physical size of the reservoir in which the fluid flow will be simulated. Next, the `cartGrid(cartDims, physDims)` function is used to create the grid object `G`. This function takes the number of cells in each direction (`cartDims`) and the physical dimensions of the grid (`physDims`) as inputs, and it creates a Cartesian grid that represents the spatial discretization of the reservoir. The resulting grid `G` will be used for the simulation of fluid flow and other reservoir properties. The `computeGeometry(G)` function is called to compute the geometry of the grid. This includes calculating important properties of the grid cells, such as cell volumes and surface areas. These geometric properties are essential for computing fluid flow, pressure drops, and other reservoir dynamics. For example, knowing the volume of each cell is necessary to compute the amount of fluid that can flow into or out of each cell during the simulation. Finally, the command `gravity reset on` is executed to enable gravity in the simulation. This is important because gravity plays a significant role in the behavior of fluids in a reservoir. By turning on gravity, the simulation accounts for the effect of gravitational forces on the fluids, which is especially important in multiphase flow models,

where oil, water, and gas interact. Enabling gravity ensures that the simulation will correctly model the vertical stratification of the fluids, with denser fluids (such as water) sinking and lighter fluids (such as gas) rising. This is a realistic representation of how fluids behave in a natural reservoir, where gravity affects the movement of fluids within the porous rock.

In summary, this code sets up the basic framework for the reservoir simulation by adding the necessary MRST modules, defining the grid dimensions and physical properties, and enabling gravity. This forms the foundation for further steps in the reservoir simulation, such as defining fluid properties, wells, and running the simulation itself.

```
rng(0);
poro = gaussianField(cartDims, [0.05, 0.3], 3, 8);
poro = poro(:);
perm = poro.^3 .* (5e-5)^2 ./ (0.81 * 72 * (1 - poro).^2);
```

This segment of the code is setting up the reservoir rock properties, specifically porosity and permeability, using random fields to simulate heterogeneity in the reservoir.

The command `rng(0)` sets the random number generator's seed to `0`. This ensures that the random processes involved in generating properties like porosity are reproducible. By fixing the seed, the same set of random values will be generated every time the code is run. This is important for debugging and testing, as it allows the simulation to produce consistent results across different runs.

The function `gaussianField(cartDims, [0.05, 0.3], 3, 8)` is generating a random field for porosity. The details of this function call are as follows:

`cartDims`: The dimensions of the grid are provided as `cartDims = [10, 10, 10]`, so the generated porosity field will have the same 10 × 10 × 10 grid structure defined earlier.

`[0.05, 0.3]`: These values define the mean and standard deviation for the porosity values. Specifically, the mean porosity is set to `0.05`, and the standard deviation is set to `0.3`.

The function generates random values that follow a Gaussian (normal) distribution with these properties. This means that most of the generated porosity values will cluster around 0.05, but some will be higher or lower, within the specified standard deviation. Porosity values typically range from 0 to 1, so the porosity generated here will be within a reasonable range.

`3`: This is the correlation length for the Gaussian field. It controls the spatial correlation of the porosity values in the reservoir grid. A higher value for the correlation length means that neighboring cells in the grid will tend to have similar porosity values, while a smaller value would make the porosity values more random and independent from neighboring cells.

`8`: This represents the seed for the Gaussian random field generator, which ensures that the same random field is produced every time the code is executed with this same seed.

Once this function is called, it returns a 3D matrix of porosity values (`poro`) that will be assigned to each grid cell in the reservoir model. The `(:)` operation flattens the matrix into a column vector, which is stored in the variable `poro`. This column vector is a 1D array of porosity values that corresponds to each grid cell in the reservoir.

The line `perm = poro.^3 .* (5e-5)^2 ./ (0.81 * 72 * (1 - poro).^2)` calculates the permeability field based on the porosity values that were just generated. In this case, permeability is modeled as a function of porosity.`poro.^3` models a relationship between porosity and permeability, where permeability increases with increasing porosity, but the relationship is nonlinear.`(5e-5)^2` represents a constant factor in the permeability calculation. This constant would be chosen based on the specific reservoir model and expected rock properties.

The `./ (0.81 * 72 * (1 - poro).^2)` denominator consists of several factors:

`0.81` is a constant factor used in the permeability equation, which represents specific geological or rock mechanical properties.
`72` is another constant that is related to the specific reservoir rock or simulation.
`(1 - poro).^2` models the fact that as porosity increases, permeability also increases, but this relationship is affected by the value of `1 - poro` which represents the fraction of the rock that is not pore space. As porosity increases, the value of `1 - poro` decreases, which reduces the overall permeability.

The resulting `perm` is a vector that contains the calculated permeability values for each grid cell in the reservoir. Permeability is generally measured in units such as Darcy or mD and the formula here models it as a function of porosity, accounting for the rock's characteristics.

This portion of the code generates a random 3D field of porosity values, based on a Gaussian distribution, and then calculates the permeability for each grid cell based on those porosity values. This process introduces heterogeneity into the reservoir model, which is a more realistic representation of actual subsurface conditions. The porosity and permeability values are used to define how fluids will flow through the reservoir in subsequent simulation steps.

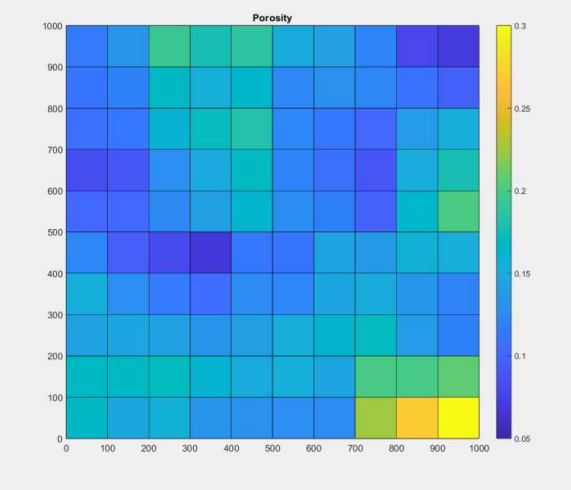


**Figure 73: Porosity Distribution.**

```
rock = makeRock(G, perm, poro);
figure(1); clf
plotCellData(G, rock.poro)
axisequal tight
colorbar
title('Porosity')
```

This segment of the code is related to setting up the reservoir rock properties and visualizing the generated porosity field. The first line calls the function `makeRock` to create a rock object, which will be used to store the rock properties (permeability and porosity) for the reservoir model. `G` contains the information about the dimensions and structure of the reservoir grid (as defined earlier in the code with `cartGrid`). This grid is used to map the porosity and permeability values to the individual cells in the reservoir. `perm` is the permeability vector that was computed earlier based on the porosity values. Each element in `perm` represents the permeability of a specific grid cell in the reservoir. `poro` is the porosity vector that was also generated earlier. It represents the porosity for each grid cell in the reservoir. The function `makeRock(G, perm, poro)` then creates a rock that associates the permeability (`perm`) and porosity (`poro`) values with the corresponding cells in the reservoir grid (`G`). The output, `rock`, is a structure containing the properties of the reservoir rock, including its porosity and permeability.

In the next line, `figure(1)` opens a figure window with the identifier `1`. `clf` stands for "clear figure." It clears the content of the figure window, ensuring that any previous plots or visualizations in that figure are removed, and the window is ready for a fresh plot.

Subsequently, `plotCellData` is a function used to visualize data on the reservoir grid (`G`). It displays the specified data (in this case, porosity) for each grid cell in the reservoir. The function takes two arguments `G` and `rock.poro`. The `rock` structure created earlier contains a field `poro`, which is the porosity for each grid cell. The function `plotCellData(G, rock.poro)` will display the porosity values in a color-coded way, where each grid cell is colored based on its porosity value. This allows the visualization of the spatial distribution of porosity within the reservoir.

The command `axis equal` ensures that the aspect ratio of the plot is equal, meaning the units are the same along both axes. This ensures that the grid cells are displayed in their correct proportions, rather than being stretched or compressed. `tight` is the next command which adjusts the plot's axes so that the entire grid is visible without any extra space around it. It ensures that the plot tightly fits the size of the data, so the grid fills the figure window.

The `colorbar` command adds a color bar to the figure. The color bar shows the range of values for the porosity data (the `rock.poro` values) and the corresponding colors used to represent them. Lastly the `tittle` command sets the title of the plot to `Porosity`.

This section of the code is used to visualize the porosity distribution within the reservoir grid. It creates a plot that shows the porosity of each grid cell using a color map, helping to visualize how the porosity is distributed spatially across the reservoir. The code also ensures that the figure is properly formatted, with the correct aspect ratio and a color bar for reference.

```
pv = poreVolume(G, rock);
T = 30*year;
irate = sum(pv)/(T*4);
```

This segment of the code calculates the pore volume of the reservoir, sets a time duration for the simulation, and computes the injection rate. The first line calculates the pore volume of the reservoir grid using the function `poreVolume`. The function takes two inputs `G` and `rock`. The function `poreVolume(G, rock)` computes the pore volume for each cell in the reservoir grid. The pore volume represents the amount of space in the rock that is available to hold fluid (such as oil, water, or gas). The result, stored in the variable `pv`, is a vector that contains the pore volume for each cell in the reservoir grid. This value is important for determining how much fluid can be stored in the reservoir.

The second line of code, `30*year`, defines time duration for the simulation specifying that the simulation will run for 30 years. The variable `T` is assigned the value corresponding to a 30-year period, which will be used later to determine how long the injection process will last in the simulation.

`irate` calculates the injection rate, which is the amount of fluid to be injected into the reservoir per year. The function `sum(pv)` computes the total pore volume of the reservoir by summing the pore volume of all the cells in the grid. This gives the overall pore volume of the entire reservoir, indicating how much fluid the entire reservoir can hold. The expression `T*4` multiplies the total simulation time (30 years) by 4. This factor represents the number of injection periods during the simulation, such as a quarterly injection schedule (four injections per year). The injection rate is then calculated by dividing the total pore volume of the reservoir (`sum(pv)`) by `T*4`, effectively giving the volume of fluid to be injected into the reservoir each year, averaged over four periods per year. This rate is important for determining how much fluid should be injected into the reservoir at each time step of the simulation.

```
makeInj = @(W, name, I, J, compi) verticalWell(W, G, rock, I, J, [],...
'Name', name, 'radius', 5*inch, 'sign', 1, 'Type', 'rate',...
'Val', irate, 'comp_i', compi);
```

This line of code defines the function handle `makeInj` for the creation of vertical injection wells in the reservoir. The function encapsulates the logic needed to define and add injection wells to the simulation in a concise and reusable manner.

The arguments of this function are:

`W`: The existing list of wells. New wells will be added to this list.

`name`: A string specifying the name of the well.

`I` and `J`: The grid cell indices in the `x` (I) and `y` (J) directions where the well is located.

`compi`: A vector specifying the injection composition (e.g., the proportions of water, oil, or gas being injected).

The function calls the `verticalWell` function, which is a utility in the MRST framework for defining vertical wells in the reservoir. The parameters passed to

`verticalWell` are `W`, `G`, `.rock`, `I`, `J` and `[]`. `[]` represents the K index (vertical grid index) of the well. By default, if empty, the well spans the entire depth of the reservoir.

Additional well properties are specified using name-value pairs:

`'Name', name`: Assigns the provided `name` to the well for identification purposes.
`'radius', 5*inch`: Sets the well radius to 5 inches. The `inch` constant converts inches to the appropriate unit of length (e.g., meters) used in the simulation.
`'sign', 1`: Specifies the well type. A positive `sign` indicates that this is an injector (injecting fluid into the reservoir), whereas a negative value would indicate a producer.
`'Type', 'rate'`: Sets the control type for the well to "rate," meaning the well will inject fluid at a specified rate.
`'Val', irate`: Defines the injection rate for the well. The `irate` value, calculated earlier, represents the volume of fluid to be injected.
`'comp_i', compi`: Specifies the composition of the injected fluid using the `compi` vector, which might include proportions of water, oil, and gas.

The `makeInj` function handle simplifies the process of defining vertical injection wells in the simulation. It accepts parameters for the list of wells, the well's name, location, and composition, and then uses these to create and add a well with predefined properties. The well is set up as an injector, with a specified rate and injection composition, enabling easy addition of wells to the reservoir model.

```
W = [];
W = makeInj(W, 'I1', 1,           1,           []);
W = makeInj(W, 'I3', cartDims(1), cartDims(2), []);
W = makeInj(W, 'I4', 1,           cartDims(2), []);
W = makeInj(W, 'I2', cartDims(1), 1,           []);
```

This section of the code initializes and sets up a list of injection wells in the reservoir model using the previously defined `makeInj` function handle. Each well is positioned at specific grid locations within the reservoir, with a unique name for identification. The wells are added to the list `W` sequentially, ensuring that the reservoir setup includes multiple injectors.

The variable `W` is initialized as an empty array. This serves as the starting point for the list of wells in the reservoir model. As new wells are defined, they are appended to this array. The first well is created using the `makeInj` function handle. It is named 'I1' and is located at the top-left corner of the reservoir grid, specifically at grid cell `(1, 1)`. The `[]` parameter for the composition indicates that no specific injection composition is provided at this stage, meaning it will use the default settings from the function handle. The second well, 'I3,' is added at the bottom-right corner of the reservoir grid. This corresponds to grid cell `(cartDims(1), cartDims(2))`, which uses the dimensions of the grid `cartDims` to calculate the exact location dynamically. This ensures flexibility if the grid dimensions are modified. The third well, 'I4,' is placed at the top-right corner of the reservoir, at grid cell `(1, cartDims(2))`. This ensures that there is an injector in the upper-right boundary of the grid, complementing

the other injector locations. The fourth well, 'I2,' is positioned at the bottom-left corner of the reservoir grid, at grid cell `(cartDims(1), 1)`. This completes the setup of the injectors by covering all four corners of the grid. By using the `makeInj` function handle, each well is created with the same set of properties, such as a fixed radius, injection rate (`irate`), and control type. The naming scheme ('I1,' 'I2,' etc.) allows for clear identification of the wells in subsequent processing steps.

This process results in a total of four injection wells, strategically distributed across the grid to ensure even injection coverage. Each well is stored in the array `W`, which now serves as the complete list of injectors for the simulation. This systematic setup is essential for defining the injection strategy and ensures balanced fluid displacement in the reservoir.

```
I = ceil(cartDims(1)/2);
J = ceil(cartDims(2)/2);
```

This segment of the code calculates the grid cell indices I and J, which are used to determine the central location of the reservoir grid in the x- and y-directions, respectively. These indices are computed dynamically based on the dimensions of the grid, `cartDims`, ensuring that the code adapts to different grid sizes.

The variable `cartDims(1)` represents the number of cells along the x-direction of the grid. By dividing this number by 2 and applying the `ceil` function, the code identifies the middle cell along the x-axis. The `ceil` function ensures that if the number of cells is even, the index points to the next higher cell, effectively choosing the center or slightly biased towards the larger half.

Similarly, the variable `cartDims(2)` represents the number of cells along the y-direction of the grid. Dividing this value by 2 and using the `ceil` function determines the middle cell along the y-axis. As with the x-direction, the `ceil` function ensures proper rounding to the next higher integer in case of even cell counts.

The variables I and J together identify the grid cell at the approximate center of the reservoir. By using dynamic calculations, the code ensures that the center cell is correctly identified regardless of the grid size, making the code more robust and adaptable.

```
W = verticalWell(W, G, rock, I, J, [], 'Name', 'P1', 'radius', 5*inch,
...
'Type', 'bhp', 'Val', 100*barsa, 'comp_i', [1, 1, 1]/3, 'Sign', -1);
```

This line creates a production well named `P1` in the reservoir simulation using the `verticalWell` function, specifying its physical properties, operational constraints, and position within the reservoir grid.

The well is named `P1` and is placed at specific grid indices I and J in the Cartesian reservoir grid (`G`). The third positional argument is left empty (`[]`), indicating that the well spans vertically through the full depth of the reservoir at that location. The well has

a radius of 5 inches, which represents the effective size of the wellbore. This parameter influences the flow dynamics near the well, such as pressure drop and fluid velocities. The well operates under a bottom-hole pressure (BHP) control mode, ensuring the simulator maintains a constant pressure of 100 barsa at the bottom of the well throughout the simulation. This control mode is commonly used to stabilize production or protect the reservoir from overdraw. The comp_i parameter defines the phase composition of fluids produced by the well. In this case, the composition is set to [1, 1, 1]/3, meaning the well is designed to handle equal proportions of water, oil, and gas. This is relevant for multi-phase flow simulations. The well is identified as a producer due to the 'Sign' parameter being set to −1.

This setup ensures the well operates realistically within the simulation, reflecting key parameters for production and multi-phase fluid handling.

```
[W_water, W_gas] = deal(W);
For i = 1:numel(W)
If W(i).sign < 0

continue
end
    W_water(i).compi = [1, 0, 0];
    W_gas(i).compi   = [0, 0, 1];
end
```

This section of code creates two configurations of the simulation's injection wells, one set for water injection (W_water) and another for gas injection (W_gas). It separates the well behavior into these two distinct categories while ensuring producers remain unaltered.

The code begins by creating two identical copies of the original wells array (W) using the deal function. These copies, W_water and W_gas, inherit all properties of the original wells. The idea is to modify each copy to serve as either water injectors or gas injectors, depending on the simulation's needs. A for loop iterates over all wells in the array W, represented by the index variable i. This loop ensures each well in the list is evaluated for its role in the simulation.

Within the loop, the condition if W(i).sign < 0 identifies wells that are producers based on their negative 'sign' value. These wells are skipped using the continue statement, ensuring no changes are made to their properties. This step is crucial, as only injector wells are intended to have their composition modified. For each injector well (sign > 0) water injectors (W_water) are set to inject only water by assigning their compi parameter to [1, 0, 0], indicating a fluid composition of 100% water. Gas injectors (W_gas) are set to inject only gas by assigning their compi parameter to [0, 0, 1], indicating a fluid composition of 100% gas.

This process ensures the simulation has two distinct sets of injection wells: one exclusively for water injection and the other exclusively for gas injection. These configurations can then be alternated in the simulation schedule, allowing the reservoir's response to each type of injection to be studied separately.

```
dT_target = 190*day;
dt = rampupTimesteps(T, dT_target, 10);
```

This section of code sets up the time-stepping scheme for the reservoir simulation, focusing on defining an appropriate sequence of time steps. This (`dT_target = 190*day`) defines the target duration for a single simulation time step as 190 days. This value represents the desired length of the time steps after an initial ramp-up phase.

The `rampupTimesteps` function generates an array of time step sizes that gradually increase from smaller initial steps to the target step size. Key parameters are:

`T`: The total simulation time, defined earlier in the script as 30 years
`dT_target`: The desired final time step size of 190 days
`10`: The number of initial smaller time steps during the ramp-up phase

The generated `dt` array ensures that the simulation begins with fine-grained time steps, capturing early transient behaviors effectively, and transitions to coarser time steps for the steady-state periods. This balance is critical for accurate and efficient simulation of reservoir dynamics.

```
schedule = struct();
schedule.control = [struct('W', W_water);...
struct('W', W_gas)];
```

This section of the code sets up the schedule for the reservoir simulation, defining how the injection wells operate over time. The schedule specifies the sequence and timing of controls for water and gas injection. `schedule = struct()` creates an empty structure, `schedule`, which will hold all the details of the simulation's operational plan, including controls and time steps. The `schedule.control` field defines two distinct operational states for the wells:

i. Water Injection Control: The first control uses the set of wells defined in `W_water`. In this state, only water is injected into the reservoir.
ii. Gas Injection Control: The second control uses the set of wells defined in `W_gas`. In this state, only gas is injected.

Each control is created as a structure with the field `'W'`, which holds the corresponding well configuration (`W_water` or `W_gas`). These controls represent the alternating injection strategies applied during the simulation.

```
schedule.step.val = dt;
schedule.step.control = (mod(cumsum(dt), 2*dT_target) >= dT_target) +
1;
```

This part of the code defines the time steps and assigns control switches for the schedule, specifying how and when the simulation alternates between water and gas injection.

The function `schedule.step.val = dt` assigns the array of time step sizes (`dt`) generated earlier to the `schedule.step.val` field. These values determine the duration of each simulation step, allowing for variable step lengths as defined by the ramp-up logic.

The next line alternates the injection control between water and gas based on cumulative time. `cumsum(dt)` computes the cumulative simulation time at each step by summing up the time steps in `dt`. This gives the total elapsed time at each step. `mod(cumsum(dt), 2*dT_target)` calculates the remainder when the cumulative time is divided by twice the target time step (`2*dT_target`). This creates a repeating cycle that resets every `2*dT_target`. `>= dT_target` checks whether the remainder is greater than or equal to `dT_target`. If true, the control switches to gas injection; otherwise, it remains in water injection. `+1` converts the binary result (0 or 1) into control indices (1 for water injection and 2 for gas injection). This logic ensures that water injection is active for the first `dT_target` days of each cycle and gas injection is active for the next `dT_target` days of each cycle.

The schedule alternates consistently every 90 days (or the equivalent of `dT_target` in this context), enabling systematic evaluation of both strategies' effects on the reservoir. This structured alternation is key to simulating alternating injection scenarios and analyzing their combined impact over the course of the simulation.

```
figure(2), clf
ctrl = repmat(schedule.step.control', 2, 1);
x = repmat(cumsum(dt/year)', 2, 1);
y = repmat([0; 1], 1, size(ctrl, 2));
surf(x, y, ctrl)
colormap(jet)
view(0, 90)
axis equal tight
set(gca, 'YTick', []);
xlabel('Time [year]')
title('Control changes over time: Red for gas injection, blue for
water')
```

This block of code creates a 3D surface plot to visualize the alternation between water and gas injection over time in the simulation. The plot provides a clear graphical representation of how the control strategy switches, with water and gas injections alternating during the simulation period.

The process begins by creating a new figure with the command `figure(2), clf,` which ensures that the figure window is cleared before any new data is plotted.

Next, the code prepares the data to be used in the plot. The `ctrl` variable is created by using `repmat(schedule.step.control', 2, 1).` This replicates the `schedule.step.control` array, which contains the control values (either water or gas injection) across time steps. By transposing (`'`) and repeating it twice, the structure of the data is modified to fit the 3D surface plot. Similarly, the x variable is constructed by calculating the cumulative sum of the time steps (`cumsum(dt/year)`) and then repeating it twice horizontally using `repmat`. This matrix represents the time progression of the simulation, with each column corresponding to a cumulative time step. For the vertical axis, `y` is created by using `repmat([0; 1], 1, size(ctrl, 2))`, which assigns alternating values of 0 and 1 to represent the two control states: 0 for water injection and 1 for gas injection.

Once the data is prepared, the `surf(x, y, ctrl)` command is used to generate the 3D surface plot. The `x` matrix corresponds to the time `axis` (in years), the `y` matrix corresponds to the injection type (water or gas), and the `ctrl` matrix represents the alternating control values for each time step. This creates a surface where the injection types are visually represented as different heights or colors along the time axis.

To enhance the visual presentation, the `colormap(jet)` command applies the `jet` colormap, which colors the surface according to the control values, making it easy to distinguish between water and gas injection. The `view(0, 90)` command adjusts the plot to a top-down view, ensuring that both the time and control states are clearly visible. The `axis equal tight` command ensures that the axes are scaled equally and that the plot tightly fits the data, removing any unnecessary space. The `set(gca, 'YTick', [])` command removes the y-axis ticks, as the y-axis is simply used to distinguish between water and gas.

Finally, the plot is labeled with `xlabel('Time [year]'),` indicating that the x-axis represents time in years, and a title is added with `title('Control changes over time: Red for gas injection, blue for water')`, explaining that the red and blue colors correspond to gas and water injection phases, respectively.

Overall, this plot visually demonstrates the alternating injection strategy, helping to better understand the timing and sequence of water and gas injection phases throughout the simulation.

```
fluid = initSimpleADIFluid('phases',    'WOG', ...
'rho',       [1000, 700, 250], ...
'n',         [2, 2, 2], ...
'c',         [0, 1e-4, 1e-3]/barsa, ...
'mu',        [1, 4, 0.25]*centi*poise ...
                          );
```

This line of code initializes a fluid model for a three-phase fluid system in a reservoir simulation using the `initSimpleADIFluid` function. The `'phases', 'WOG'` argument specifies the phases present in the fluid system. In this case, the phases are Water, Oil, and Gas, indicating that the reservoir fluid consists of these three immiscible phases. This is important for modeling multiphase flow, as each phase will have distinct physical properties that affect the reservoir behavior.

The `'rho', [1000, 700, 250]` argument defines the density of each fluid phase in kilograms per cubic meter (kg/m³). For water, the density is set to 1000 kg/m³, which is a typical value for fresh water at standard conditions. Oil is assigned a density of 700 kg/m³, reflecting a less dense fluid than water. Gas has the lowest density, set at 250 kg/m³, which is typical for natural gas. These densities are used in the simulation to calculate the pressure, flow rates, and other reservoir properties related to the fluids.

The `'n', [2, 2, 2]` argument specifies the exponents used in the relative permeability and capillary pressure functions for each phase. In this case, the exponent for each phase (water, oil, and gas) is set to 2. This indicates that the relative permeability of each phase follows a power-law relationship with an exponent of 2. This choice is typically used in simulations to represent the flow characteristics of the phases, as these exponents affect how each fluid interacts with the others, especially in porous media.

The `'c', [0, 1e-4, 1e-3]/barsa` parameter defines the capillary pressure between the phases. Capillary pressure is the pressure difference between two immiscible fluids in the reservoir. For this model, the capillary pressure between water and oil is set to zero, indicating no significant capillary pressure between these two phases. However, a small capillary pressure is defined between oil and gas (`1e-4` bar) and between water and gas (`1e-3` bar). These values reflect typical conditions in which gas is less miscible with both water and oil, creating capillary forces between these phases. The capillary pressure values are expressed in barsa (bar absolute), a unit of pressure.

Finally, the `'mu', [1, 4, 0.25]*centi*poise` argument defines the viscosity of each fluid phase in centipoise (cP), a unit of dynamic viscosity. Water is assigned a viscosity of 1 cP, which is typical for water at standard conditions. Oil has a viscosity of 4 cP, representing a more viscous fluid than water. Gas has the lowest viscosity at 0.25 cP, reflecting its more mobile nature compared to the liquid phases. These viscosity values are critical in determining the resistance to flow for each phase in the reservoir, impacting the overall fluid dynamics.

```
model = ThreePhaseBlackOilModel(G, rock, fluid, 'disgas', false,
'vapoil', false);
seqModel = getSequentialModelFromFI(model);
```

This line of code initializes a three-phase black oil model using the function `ThreePhaseBlackOilModel`. The model simulates the flow and interaction of three immiscible fluid phases water, oil, and gas within a reservoir. The black oil model is commonly used in reservoir simulations because it simplifies the behavior of these phases under typical reservoir conditions. It assumes that each phase behaves according to relatively simple properties, making it a practical choice for many engineering applications. The `G` variable represents the grid, which defines the reservoir's spatial discretization, while rock contains the `rock` properties, such as permeability and porosity. The `fluid` variable defines the fluid properties, such as density, viscosity, and capillary pressure, as initialized in the previous step. The additional parameters `'disgas', false` and `'vapoil', false` specify that the model should not include gas dissolution into oil or oil vaporization, respectively, meaning it assumes that gas and oil phases remain immiscible and do not mix under typical reservoir conditions. This setup allows for simulating multiphase flow in a more straightforward manner, focusing on water, oil, and gas interactions without additional complexities like phase changes.

The next line of code creates a sequential model from the fully implicit model defined earlier. This is done using the `getSequentialModelFromFI` function, which generates a model based on a sequential solution approach. The sequential model is particularly useful for handling large-scale reservoir simulations where efficiency is critical, as it allows for solving each phase's equations separately while still accounting for the interactions between phases. This transition from a fully implicit model to a sequential model enables the simulation to maintain accuracy while potentially reducing computational costs.

```
state = initResSol(G, 100*barsa, [0, 1, 0]);
[wsSeq, statesSeq] = simulateScheduleAD(state, seqModel, schedule);
[wsFIMP, statesFIMP] = simulateScheduleAD(state, model, schedule);
```

The first line of code initializes the reservoir state using the `initResSol` function. This function sets the initial conditions for the simulation, which include the pressure and fluid saturations in the reservoir. The parameter `G` refers to the grid that defines the geometry and discretization of the reservoir. The pressure is initialized to 100 bar (converted to barsa). This initial pressure is a crucial parameter as it influences fluid flow and the overall dynamics of the reservoir. The vector `[0, 1, 0]` defines the initial fluid saturations in the reservoir. Here, it indicates that the reservoir is initially fully saturated with oil and both water and gas have zero saturation, meaning no water or gas is present at the start. This assumption helps in modeling scenarios where a reservoir starts with oil as the dominant phase, which is typical in many oil reservoirs.

The next two lines of code simulate the reservoir's behavior over time using two different solution approaches. The function `simulateScheduleAD` is used to perform the simulation based on the predefined schedule (`schedule`), which controls the injection and production rates. The first simulation uses the sequential model `seqModel`, which solves the reservoir equations one phase at a time. This method is generally less computationally intensive and can be more efficient for large-scale simulations. The result of this simulation is stored in `wsSeq` (well solutions) and `statesSeq` (reservoir states at each time step). The second simulation uses the fully implicit model (`model`), which solves all the phases' equations simultaneously for greater accuracy but at a higher computational cost. The result of this simulation is stored in `wsFIMP` (well solutions) and `statesFIMP` (reservoir states at each time step). These two simulations allow for comparison between the two methods, highlighting differences in efficiency and accuracy when modeling reservoir behavior under the same conditions.

```
df = get(0, 'DefaultFigurePosition');
figure('position', df.*[1, 1, 2, 1])
names = {'qWs', 'qOs', 'qGs'};
holdon
t = cumsum(schedule.step.val)/day;
colors = lines(3);
```

This block of code is used to set up the visual representation of the results from the reservoir simulation, specifically for plotting the production rates of the three phases (water, oil, and gas) over time. The first line, `df = get(0, 'DefaultFigurePosition')`, retrieves the default figure position settings from the MATLAB environment. This ensures that the subsequent figure is created with the default size and position. The next line, `figure('position', df.*[1, 1, 2, 1])`, creates a new figure window and adjusts its size based on the default figure position, but it doubles the width (`2` in the `[1, 1, 2, 1]` scaling factor), effectively making the figure wider.

The line `names = {'qWs', 'qOs', 'qGs'}` defines a cell array containing the names of the three quantities to be plotted: `qWs` (water production rate), `qOs` (oil production rate), and `qGs` (gas production rate). These are the well outputs for each phase that were calculated during the simulation. The `hold on` command ensures that all the plots are drawn on the same figure without overwriting each other.

The time vector `t = cumsum(schedule.step.val)/day` computes the cumulative sum of the time steps (from the `schedule`), converting the time into days. This represents the total simulation time at each step, which is necessary for plotting the production rates over time. Finally, `colors = lines(3)` generates a set of three distinct colors to be used for plotting the three different phase production rates, ensuring that each phase (water, oil, gas) is clearly distinguishable in the plot.

```
For i = 1:numel(names)
qSeq = -getWellOutput(wsSeq, names{i}, 5);
qFIMP = -getWellOutput(wsFIMP, names{i}, 5);
stairs(t, qSeq*day, '-o', 'color', colors(i, :));
stairs(t, qFIMP*day, '.', 'color', colors(i, :), 'linewidth', 2);
end
```

This section of code is responsible for plotting the production rates of the different fluid phases (water, oil, and gas) over time, comparing the results from two simulation methods: the sequential model and the fully implicit model.

The loop `for i = 1:numel(names)` iterates over the elements in the `names` cell array, which contains the names of the quantities to be plotted: `qWs`, `qOs`, and `qGs`, representing the production rates of water, oil, and gas, respectively. For each iteration, the code retrieves the production rate data for the current phase from both the sequential and fully implicit simulations. Specifically, `qSeq = -getWellOutput(wsSeq, names{i}, 5)` retrieves the production rate for the current phase (`names{i}`), using the `getWellOutput` function on the sequential simulation results (`wsSeq`). The negative sign ensures that the production rates are positive, as the output is usually given as a negative value for produced fluids. Similarly, `qFIMP = -getWellOutput(wsFIMP, names{i}, 5)` retrieves the corresponding production rate for the fully implicit model (`wsFIMP`).

The `stairs(t, qSeq*day, '-o', 'color', colors(i, :))` function plots the sequential simulation results for the current phase using a stair-step plot, where `t` represents the time vector (in days), and `qSeq*day` converts the production rate into units of cubic meters per day. The plot is drawn with circular markers (`'-o'`) and uses the color specified in the `colors` array for each phase. Similarly, the line `stairs(t,`

`qFIMP*day, '.', 'color', colors(i, :), 'linewidth', 2)` plots the results from the fully implicit model with a dotted line (`'.'`) and a thicker line width (`'linewidth', 2`) to differentiate it from the sequential model's plot.

This loop, therefore, creates two sets of plots (one for the sequential model and one for the fully implicit model) for each phase (water, oil, and gas), allowing for a comparison of the production rates over time.

```
legend('Water, sequential', 'Water FIMP', 'Oil, sequential', ...
'Oil FIMP', 'Gas, sequential', 'Gas FIMP');
xlabel('Time [days]')
ylabel('Production at standard conditions [m^3/day]')
axis tight
```

This section of code is used to enhance the plot by adding labels, a legend, and adjusting the axis to ensure that the data is displayed clearly.

The `legend` function is used to label the different lines on the plot. Specifically, it labels the production rates for water, oil, and gas, separately for the sequential model and the fully implicit model (FIMP). The labels `'Water, sequential', 'Water FIMP', 'Oil, sequential', 'Oil FIMP', 'Gas, sequential'`, and `'Gas FIMP'` correspond to the respective lines that were plotted in the earlier part of the code. This helps distinguish between the results for each fluid phase and model type.

The `xlabel('Time [days]')` and `ylabel('Productionat standardconditions [m^3/day]')` functions are used to set the x-axis and y-axis labels, respectively. The x-axis represents time, in days, and the y-axis represents the production rate of each fluid phase in cubic meters per day ($m^3$/day) under standard conditions. Finally, the `axis tight` command adjusts the axis limits so that the plot is zoomed in tightly around the data, ensuring that the plot is clear and well-fitted without unnecessary space.

```
df = get(0, 'DefaultFigurePosition');
figure('position', df.*[1, 1, 2, 1])
names = {'qWs', 'qOs', 'qGs'};
holdon
t = cumsum(schedule.step.val)/day;
colors = lines(3);
```

This block of code is responsible for preparing the plot for the production rates of water, oil, and gas over time. The first line, `df = get(0, 'DefaultFigurePosition')`, retrieves the default figure position and size from MATLAB, which is typically the dimensions of the last active figure window. This value is then used in the next line to create a new figure with the command `figure('position', df.*[1, 1, 2, 1])`. Here, the size of the figure is adjusted by multiplying the default dimensions by `[1, 1, 2, 1]`, effectively making the figure wider, while keeping the height the same. This ensures that the new figure has an appropriate size for displaying the plots clearly. The line `names = {'qWs', 'qOs', 'qGs'};` defines a cell array containing the names of the three variables to be plotted: `qWs`, `qOs`, and `qGs`. The `hold on` command ensures that all the plots are drawn on

the same figure without erasing previous ones. The time vector `t = cumsum(schedule.step.val)/day` calculates the cumulative time steps, converting them into days, based on the time intervals specified in the `schedule` structure. Finally, the line `colors = lines(3);` generates a color map with three distinct colors (one for each phase: water, oil, and gas) to be used in the plots, making it easier to differentiate the production rates for each phase.

```
fori = 1:numel(names)
qSeq = -getWellOutput(wsSeq, names{i}, 5);
qFIMP = -getWellOutput(wsFIMP, names{i}, 5);
stairs(t, qSeq*day, '-o', 'color', colors(i, :));
stairs(t, qFIMP*day, '.', 'color', colors(i, :), 'linewidth', 2);
end
```

This code block is designed to plot the production rates of different fluid phases (water, oil, and gas) over time for both the sequential and fully implicit models. The loop `for i = 1:numel(names)` iterates through each element of the `names` array, which contains the names of the quantities to be plotted (`qWs`, `qOs`, and `qGs`. In each iteration, the code retrieves the production data for the current phase from both simulation results: the sequential model (`wsSeq`) and the fully implicit model (`wsFIMP`). The `getWellOutput(wsSeq, names{i}, 5)` function extracts the production rate for the current phase from the sequential model, while `getWellOutput(wsFIMP, names{i}, 5)` does the same for the fully implicit model. The negative sign `(-)` in front of both functions ensures that the production rates are positive.

After retrieving the data, the production rates are plotted using the `stairs` function. The line `stairs(t, qSeq*day, '-o', 'color', colors(i, :));` plots the results from the sequential model. It uses a stair-step plot to represent how production changes over time (`t`), with markers (`-o`) to indicate each data point and colors taken from the `colors(i, :)` array to distinguish each fluid phase. Similarly, the line `stairs(t, qFIMP*day, '.', 'color', colors(i, :), 'linewidth', 2);` plots the production rates from the fully implicit model using a dotted line (`'.'`) and a thicker line width (`'linewidth', 2`) for better visual distinction.

The loop iterates over all three fluid phases (water, oil, gas), generating separate plots for each one and allowing for a clear visual comparison between the sequential and fully implicit models across the three phases.

```
legend('Water, sequential', 'Water FIMP', 'Oil, sequential', ...
'Oil FIMP', 'Gas, sequential', 'Gas FIMP');
xlabel('Time [days]')
ylabel('Production at standard conditions [m^3/day]')
axistight
clf; hold on
colors = lines(4);
```

This section of the code enhances the plot by adding a legend, axis labels, and adjusting the plot's appearance for clarity. The `legend` function is used to provide descriptive labels for the plotted lines, distinguishing between the production rates of water, oil, and gas for both

the sequential and fully implicit models (FIMP). The labels are organized in pairs: one for the sequential model and one for the fully implicit model, making it easy to identify the corresponding lines for each fluid phase (water, oil, gas). Next, the `xlabel('Time [days]')` and `ylabel('Production at standard conditions [m^3/day]')` commands add appropriate labels to the x-axis and y-axis, respectively. The x-axis represents time in days, while the y-axis represents the production rate of each fluid phase in cubic meters per day under standard conditions, providing clear context for the data being plotted. The `axis tight` command ensures that the axis limits are automatically adjusted to fit the data tightly, eliminating unnecessary space around the plot and ensuring that the focus remains on the relevant data. The `clf; hold on` command clears the current figure and holds the plot open, allowing additional elements to be added without clearing the existing ones. Finally, the `colors = lines(4);` line defines a set of four distinct colors to be used for plotting, ensuring that each of the four sets of data (water, oil, gas, and the associated models) has a unique color for easy differentiation in the plot.

```
For i = 1:4
bhpSeq = getWellOutput(wsSeq, 'bhp', i)/barsa;
bhpFIMP = getWellOutput(wsFIMP, 'bhp', i)/barsa;

stairs(t, bhpSeq, '-o', 'color', colors(i, :));
stairs(t, bhpFIMP, '.', 'color', colors(i, :), 'linewidth', 2);
end
```

This block of code is used to plot the bottom-hole pressure (BHP) for the wells in both the sequential and fully implicit models over time. The loop `for i = 1:4` iterates over the four wells in the simulation, retrieving the BHP data for each well from both simulation results: the sequential model (`wsSeq`) and the fully implicit model (`wsFIMP`). The `getWellOutput(wsSeq, 'bhp', i)/barsa` function extracts the BHP data for the `i-th` well from the sequential model, while `getWellOutput(wsFIMP, 'bhp', i)/barsa` does the same for the fully implicit model. The division by barsa converts the BHP from Pascals (Pa) to bars, a more commonly used unit for pressure in this context.

The `stairs` function is then used to plot the BHP data for each well over time, with different line styles and colors. The command `stairs(t, bhpSeq, '-o', 'color', colors(i, :));` plots the BHP from the sequential model for the i-th well, using a solid line with circle markers `(-o)` and assigning it a color from the `colors(i, :)` array. Similarly, `stairs(t, bhpFIMP, '.', 'color', colors(i, :), 'linewidth', 2);` plots the BHP from the fully implicit model for the same well, using a dotted line (.) with a thicker line width (`linewidth`, 2) to visually differentiate it from the sequential model. This ensures that the BHP for each well in both models is clearly displayed and easily distinguishable, allowing for a comparison of how the bottom-hole pressure evolves over time for both model types.

```
xlabel('Time [days]')
ylabel('Injector bottom hole pressure [bar]')
axistight
```

This segment of code adds labels to the x-axis and y-axis of the. The command `xlabel('Time [days]')` labels the x-axis with `'Time [days]'`, indicating that the horizontal axis represents time measured in days. Similarly, `ylabel('Injector bottom hole pressure [bar]')` labels the y-axis with `'Injector bottom hole pressure [bar]'`, clarifying that the vertical axis represents the pressure at the bottom of the injection wells, measured in bar. This is a key piece of information for understanding the conditions of the wells during the simulation. Finally, the command `axis tight` adjusts the plot's axis limits so that the data fits snugly within the plot area, removing any unnecessary empty space around the plot.

```
plotWellSols({wsSeq, wsFIMP}, cumsum(schedule.step.val), ...
'datasetnames', {'Sequential', 'FIMP'}, 'linestyles', {'-', '-.'})
```

This line of code generates a plot that visualizes the well solutions for both the sequential and fully implicit models over time. The function `plotWellSols` is called with the simulation results for both models, provided as a cell array `{wsSeq, wsFIMP}`. The `wsSeq` and `wsFIMP` variables represent the well solutions from the sequential and fully implicit models, respectively. The second argument, `cumsum(schedule.step.val)`, calculates the cumulative sum of the timestep values (`schedule.step.val`), which represents the total time elapsed at each simulation step, converting the simulation steps into actual time units.

The `datasetnames` parameter is used to label the two datasets in the plot, assigning the names 'Sequential' and 'FIMP' to the respective models. The `linestyles` parameter defines the line styles for the two datasets, where `'-'` is used for the sequential model, and `'-. '` (a dashed-dotted line) is used for the fully implicit model. This visual differentiation helps to easily distinguish between the two sets of data in the plot. Overall, this line of code provides a clear comparison of well behavior over time for both models, using distinct line styles and labels to highlight the differences.

```
figure;
plotToolbar(G, statesSeq);
axistight
plotWell(G, W);
title('Sequential implicit')
```

This segment of code creates a figure to visualize the simulation results of the sequential implicit model. The `figure` command generates a new figure window. The next line, `plotToolbar(G, statesSeq)`, displays a 3D representation of the reservoir grid (`G`) along with the simulation results from the sequential model (`statesSeq`). The `plotToolbar` function visualizes the grid and the states of the reservoir at each time step, offering a comprehensive view of the model's behavior, such as fluid distributions and pressure fields. The `axis tight` command ensures that the plot is adjusted to fit the data without any extra space, making the display more focused and efficient.

The next line, `plotWell(G, W)`, overlays the well locations onto the grid visualization. This function places markers on the grid where the wells (represented by `W`) are located, giving a clear indication of where fluid production and injection activities are occurring in the reservoir. The inclusion of well locations is critical for understanding how the wells interact with the reservoir during the simulation. Finally, `title('Sequential implicit')` adds a title to the plot.

```
figure;
plotToolbar(G, statesFIMP);
axistight
plotWell(G, W);
title('Fully-implicit')
```

This section of code generates a new figure window and visualizes the simulation results of the fully implicit model. The `figure` command creates an empty figure for the upcoming plots. The function `plotToolbar(G, statesFIMP)` is then called to create a 3D visualization of the reservoir grid (`G`) along with the results from the fully implicit model (`statesFIMP`). This function plots the state of the reservoir at each time step, displaying the distribution of fluid properties such as pressure and saturation throughout the grid. This helps in understanding how the reservoir evolves over time during the simulation. The `axis tight` command ensures that the axis limits are adjusted to tightly encompass the plotted data.

Following this, `plotWell(G, W)` overlays the well locations on the reservoir grid. The `plotWell` function places markers at the positions of the wells, represented by the variable `W`, on the visualized grid. This allows the user to see where the wells are situated and observe how they interact with the reservoir during the simulation. Finally, the `title('Fully-implicit')` command adds a title to the plot.

The following diagrams illustrate the evolution of the Gas-Oil Ratio (GOR) during Water Alternating Gas (WAG) injection under two different gravity conditions: with gravity activated (Gravity On) and without gravity (Gravity Off). These conditions significantly influence the fluid dynamics and production behavior, as observed in the respective trends.

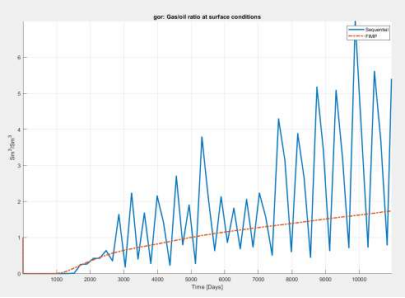

**Figure 74: Diagram of GOR - Time with Gravity ON.**

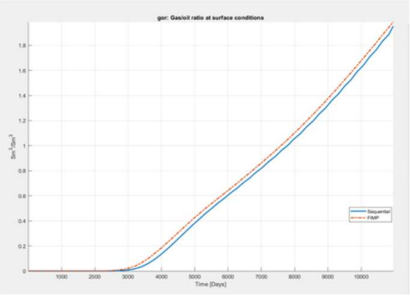

**Figure 75: Diagram of GOR - Time with Gravity OFF.**

In the first diagram (Gravity On), the GOR exhibits significant fluctuations throughout the WAG injection process. The peaks in the GOR likely correspond to the gas injection phases, while the drops represent the water injection phases. The activation of gravity amplifies the fluid flow dynamics, resulting in more pronounced variations. Gas, being lighter, migrates upward under gravity, which impacts production behavior and creates these observable fluctuations. This highlights how gravity influences phase distribution and flow characteristics during WAG injection.

In contrast, the second diagram (Gravity Off) displays a smoother GOR progression with fewer fluctuations. The absence of gravity reduces the dispersion effect between the gas and water phases, leading to a more balanced and steady production trend. Without buoyancy forces, the alternating gas and water injections are less dynamic, resulting in a more predictable and consistent GOR increase.

When comparing the two scenarios, several key differences emerge. Under Gravity On conditions, the GOR demonstrates larger fluctuations due to buoyancy forces affecting phase distribution. This creates chaotic and dynamic interactions between the phases, causing sharper transitions in the production trend. Conversely, in the Gravity Off case, the GOR evolves steadily, with minimal interactions between the gas and water phases. This smoother progression allows for easier prediction and analysis of the production trend.

Furthermore, the behavior of the WAG injection differs significantly between the two scenarios. With gravity activated, the alternating injection of gas and water causes more pronounced effects on flow behavior, leading to abrupt changes in the production trend. In the absence of gravity, the system responds more calmly to injection changes, avoiding sharp variations and maintaining stability.

In conclusion, the presence of gravity intensifies the interactions between the phases, increasing flow dynamics but also introducing higher uncertainty in production. On the other hand, the absence of gravity results in a more stable and smoother production trend, enhancing predictability and potentially improving efficiency during WAG injection.

The two provided plots (Figure 76 & 77) illustrate the well surface rate of oil under different simulation conditions for Water Alternating Gas (WAG) injection schemes. The first plot represents the case with gravity turned on, while the second corresponds to gravity turned off. Both simulations compare two numerical approaches: the Sequential method and the Fully Implicit Method for Pressure (FIMP).

The variables `qWs`, `qOs`, and `qGs` represent the surface production rates of water, oil, and gas, respectively, expressed in standard barrels per day (STB/day) for liquids and standard cubic feet per day (SCF/day) for gas. These outputs are extracted using the command:

```
[qWs, qOs, qGs, bhp] = wellSolToVector(wellSols)
```

Specifically:

`qWs`: corresponds to the surface water rate

`qOs`: to the surface oil rate

`qGs`: to the surface gas rate

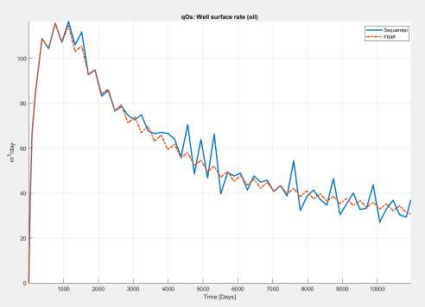


**Figure 76: Diagram of qOs - Time with Gravity ON.**

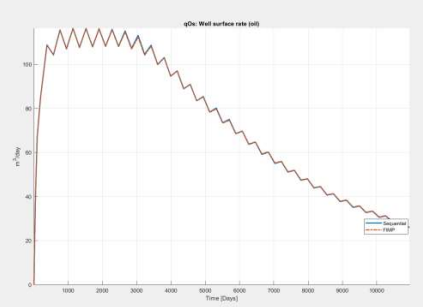


**Figure 77: Diagram of qOs - Time with Gravity OFF.**

In the first plot (gravity on), the oil production rate begins at a high value and steadily declines over time. This decline exhibits notable fluctuations, which can be attributed to the dynamics of WAG injection. The alternating injection of water and gas creates periodic changes in reservoir conditions, leading to the oscillations observed in the production curve. Gravity effects play a significant role in this scenario, as the heavier water sinks and the lighter gas rises, promoting oil displacement but also introducing instabilities. These fluctuations are more pronounced for the Sequential method, which appears more sensitive to the effects of gravity. In contrast, the FIMP method smooths out these instabilities, likely due to its more robust numerical stability. Over time, the production curves of both methods converge as the reservoir reaches a more stable flow regime.

In the second plot (gravity off), the oil production rate follows a much smoother decline, with minimal fluctuations compared to the gravity-on case. The absence of gravity eliminates buoyancy effects, resulting in a more uniform fluid displacement. This produces consistent and predictable reservoir behavior, as seen in the nearly identical curves for the Sequential and FIMP methods. The lack of gravity ensures a stable displacement front, although it may reduce the efficiency of early oil recovery compared to the gravity-on scenario. Both methods perform equally well in this case, indicating that gravity-driven dynamics are a key factor in differentiating their behavior.

When comparing the gravity-on and gravity-off scenarios, significant differences emerge. With gravity on, early oil recovery is enhanced due to buoyancy-driven displacement, but the production profile is less stable and exhibits larger fluctuations. This could lead to operational challenges and uneven recovery patterns over time. In contrast, the gravity-off case offers a smoother and more predictable decline in oil production, making it easier to manage but potentially less efficient in the early stages. The gravity-on scenario emphasizes the importance of segregation effects, while the gravity-off scenario highlights the benefits of uniform displacement.

In conclusion, these plots demonstrate the critical influence of gravity on reservoir dynamics during WAG injection. While gravity-on conditions enhance early production, they introduce instabilities that must be managed. Gravity-off conditions, on the other hand, prioritize stability and consistency at the cost of reduced early recovery efficiency. Both numerical approaches—Sequential and FIMP—perform reliably, but FIMP shows a slight advantage in stabilizing fluctuations under gravity-on conditions. This analysis underscores the importance of accounting for gravity effects when designing injection strategies and selecting numerical methods for simulation.

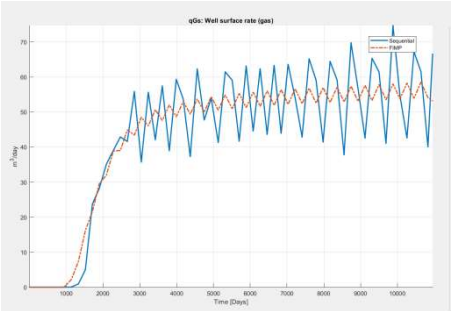


**Figure 78: Diagram of qGs -Time with Gravity ON.**

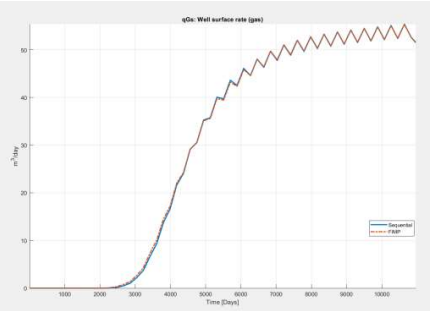


**Figure 79: Diagram of qGs - Time with Gravity OFF.**

These two graphs illustrate the gas production rates over time under two different scenarios for a Water-Alternating-Gas (WAG) injection process: one with gravity included (Figure 78) and one with gravity effects disabled (Figure 79). Both datasets, "Sequential" and "FIMP," are plotted to compare their performance.

In Figure 78, with gravity enabled, the gas production rate exhibits a periodic oscillatory behavior, which reflects the alternating nature of the injection process. The peaks and troughs correspond to the respective gas and water injection cycles. The inclusion of gravity leads to a more dynamic production profile as gravity segregation likely contributes to variations in the mobility and displacement of gas and water phases. Despite these fluctuations, the "FIMP" method follows the general trend of the "Sequential" approach, although with slightly smoother variations.

In contrast, Figure 79, with gravity effects disabled, shows a much more stable and predictable gas production profile. The absence of gravity-driven segregation simplifies the displacement process, resulting in more uniform and consistent gas production rates. Here, the periodic nature of the WAG process is still apparent, but the amplitude of the oscillations is significantly reduced. The two methods, "Sequential" and "FIMP," display almost identical behavior, indicating minimal differences in production performance under these conditions.

Overall, gravity significantly influences gas production dynamics in a WAG process, introducing greater variability and potentially enhancing phase segregation. The "FIMP" method appears robust under both scenarios, maintaining close agreement with the "Sequential" method while exhibiting smoother behavior.

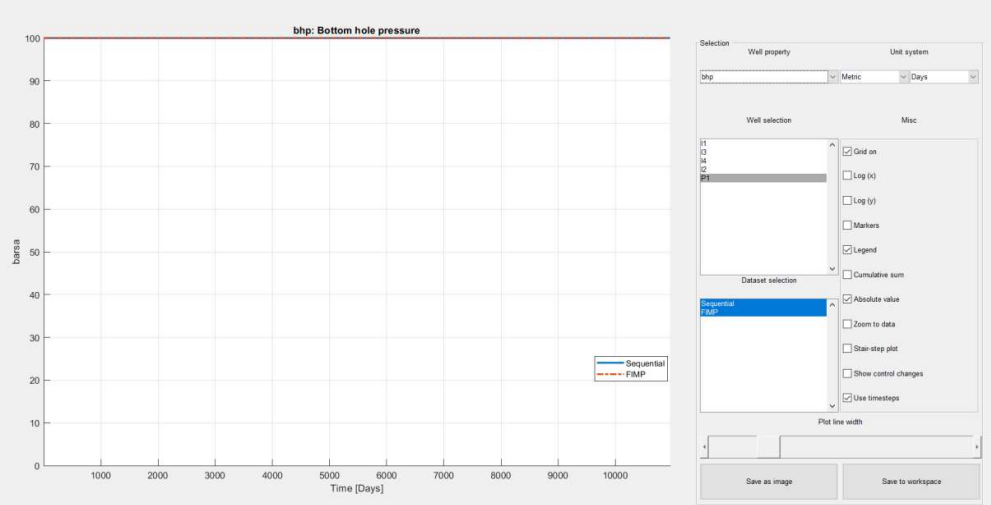


**Figure 80: Diagram of BHP - Time for the Producer.**

The bottom-hole pressure (bhp) of the production well remains constant at 100 barsa throughout the simulation. This is expected, as the well operates under a fixed bhp control, which was explicitly defined in the simulation setup using the command: `'Type', 'bhp', 'Val', 100*barsa`. Consequently, any variations in production rates are due to reservoir dynamics rather than changes in well pressure.

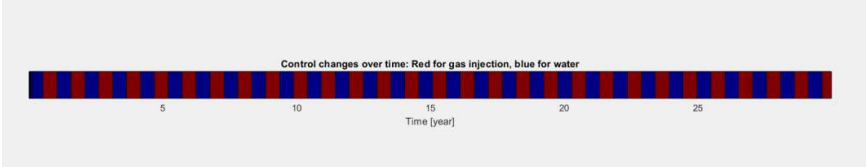


**Figure 81: Water (blue) and Gas (red) inflow alternation over time.**

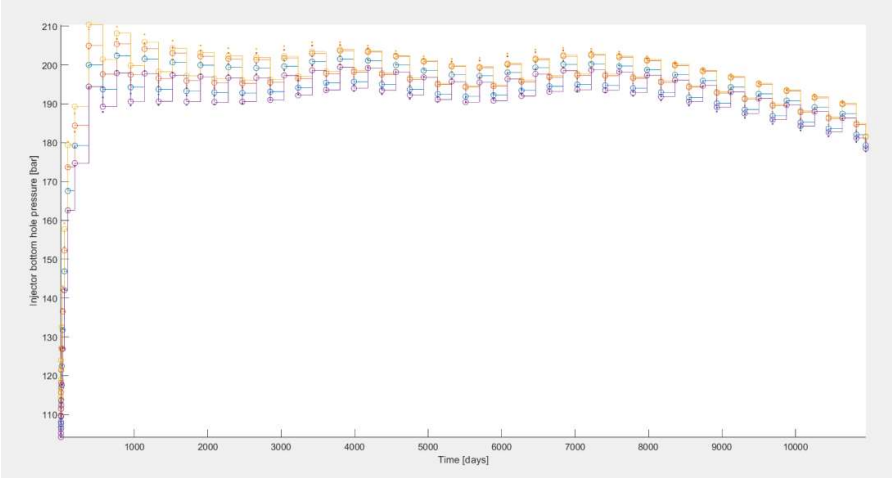


**Figure 82: Diagram of BHP – Time for the Injectors.**

In figure 82, four curves corresponding to the four injectors are observed. The highest curve, in yellow, corresponds to injector I4, the next one below it, in orange, corresponds to injector I3, the following one, in light blue, corresponds to injector I1, and the other curve, in purple, corresponds to injector I2. We notice that there is a fluctuation (waves) in the curves over time, which is due to the alternating presence of water and gas in the reservoir. It should be noted that the depth of the wells is the same as that of the producer, as they are defined in the code as vertical wells.

The simulation employs a target time step of 190 days, set by the variable `dT_target = 190*day`, meaning the reservoir state is updated at this interval. Initially, the time steps ramp up gradually during the first 10 iterations (using the `rampupTimesteps` function), allowing for stable convergence as fluid mobility evolve. After this ramp-up phase, the injection alternates every 190 days between water and gas, continuing this cyrle throughout the 30-year simulation. This structured alternation is responsible for the cyclic behavior seen in the BHP curves of the injectors.

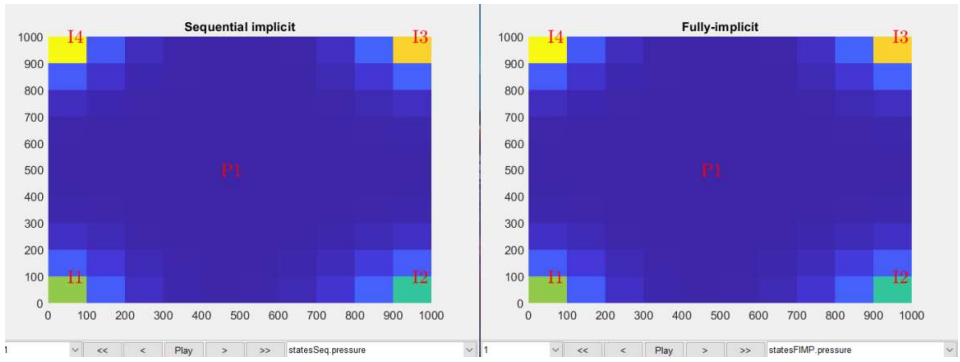


**Figure 83: Sequential Implicit (left) and Fully Implicit (right) (1st step).**

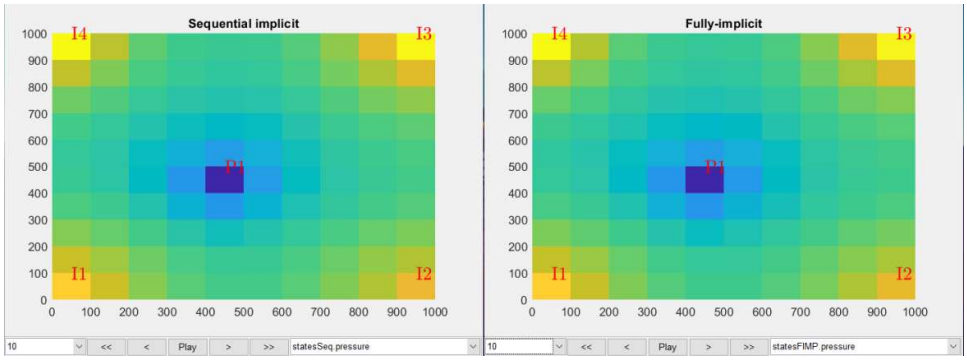


**Figure 84: Sequential Implicit (left) and Fully Implicit (right) (10th step).**

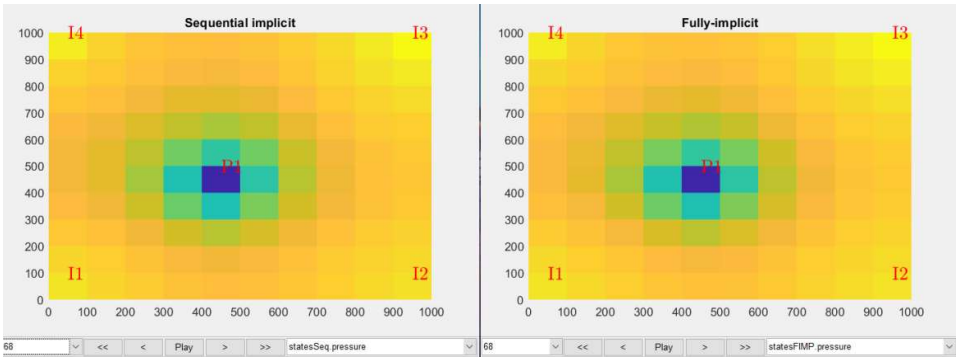


**Figure 85: Sequential Implicit (left) and Fully Implicit (right) (68th step).**

The simulation schedule alternates between water and gas injection, and results are visualized dynamically by plotting water and gas saturation across the reservoir over time.

```
MrstModule add ad-core ad-blackoil ad-props spe10 mrst-gui
   sequential;
```

This line ensures that the necessary MRST modules are loaded before running the simulation. The `mrstModule add` command is used to import specific modules required for reservoir modeling and simulation. The `ad-core` module provides fundamental functionalities for automatic differentiation-based simulations, while `ad-blackoil` implements the black-oil formulation for three-phase flow. The `ad-props` module includes property models for fluid and rock behavior, and `spe10` contains predefined datasets, particularly the SPE10 benchmark model commonly used for testing simulations. The `mrst-gui` module provides graphical tools for visualization, and `sequential` enables sequential solvers for multiphase flow simulations.

```
cartDims = [10, 10, 10];
physDims = [1000, 1000, 10]*meter;
G = cartGrid(cartDims, physDims);
G = computeGeometry(G);
Gravity reset off;
```

This section of the code sets up the reservoir grid, defines its physical dimensions, and configures fundamental properties related to rock and fluid behavior. The variable `cartDims = [10, 10, 10]` defines a Cartesian grid with 10 cells in each spatial direction (x, y, and z), resulting in a total of 1,000 cells in the reservoir model. The physical dimensions of the reservoir are specified using `physDims = [1000, 1000, 10]*meter`, meaning that the reservoir extends 1,000 meters in the x and y directions and 10 meters in the z direction. The command `G = cartGrid(cartDims, physDims)` creates a structured Cartesian

grid based on the specified number of cells and physical dimensions. This grid serves as the foundation for discretizing the reservoir and solving the governing flow equations. The next line, `G = computeGeometry(G)`, calculates geometric properties such as cell volumes, centroids, and face areas, which are necessary for numerical simulations. Finally, `gravity reset off` disables gravity in the simulation.

```
rng(0);
poro = gaussianField(cartDims, [0.05, 0.3], 3, 8);
poro = poro(:);
perm = poro.^3 .* (5e-5)^2 ./ (0.81 * 72 * (1 - poro).^2);
rock = makeRock(G, perm, poro);
```

This section of the code initializes the porosity and permeability properties of the reservoir rock. The command `rng(0)` sets the random number generator seed to zero, ensuring that the results are reproducible each time the script is run. This is important when generating random fields for porosity distribution.

The function `gaussianField(cartDims, [0.05, 0.3], 3, 8)` generates a spatially correlated random field for porosity with values ranging between 0.05 and 0.3. The parameters `3` and `8` control the correlation length and variability of the field. The generated porosity field is stored in the variable `poro`, and `poro = poro(:)` reshapes it into a column vector to ensure compatibility with further calculations.

The permeability (`perm`) is computed using an empirical relationship that depends on the porosity. The equation `perm = poro.^3 .* (5e-5)^2 ./ (0.81 * 72 * (1 - poro).^2)` is derived from the Kozeny-Carman equation, which relates permeability to porosity and grain size. In this case, permeability is proportional to the cube of the porosity and inversely proportional to the square of the term `(1 - poro)`, which represents the fraction of the rock that is not porous. The coefficient `(5e-5)^2` represents a characteristic length scale, while the denominator includes constants related to tortuosity and grain packing properties.

Finally, `rock = makeRock(G, perm, poro)` creates the rock property structure, associating the computed permeability and porosity values with the previously defined grid `G`. This structure will be used in subsequent calculations to model fluid flow in the reservoir.

```
fluid = initSimpleADIFluid('phases', 'WOG', 'rho', [1000, 700,
250], 'n', [2, 2, 2], 'c', [0, 1e-4, 1e-3]/barsa, 'mu', [1, 4,
0.25]*centi*poise);
model = ThreePhaseBlackOilModel(G, rock, fluid, 'disgas', false,
'vapoil', false);
state = initResSol(G, 100*barsa, [0, 1, 0]);
```

The command `fluid = initSimpleADIFluid(...)` initializes a simple three-phase fluid model using the automatic differentiation framework of MRST. The `'phases', 'WOG'` parameter specifies that the model consists of water (`W`), oil (`O`), and gas (`G`). The `'rho', [1000, 700, 250]` parameter defines the densities of water, oil, and gas in kg/m³, respectively. The `'n', [2, 2, 2]` parameter sets the relative permeability exponents for each phase, which control the shape of the relative permeability curves. The `'c', [0, 1e-4, 1e-3]/barsa` parameter defines the compressibility of each phase, with water being treated as incompressible (`0`), oil having a compressibility of `1e-4` per bar, and gas having a higher compressibility of `1e-3` per bar. The `'mu', [1, 4, 0.25]*centi*poise` parameter defines the dynamic viscosities of water, oil, and gas in centipoise, with water having a viscosity of `1 cP`, oil `4 cP`, and gas `0.25 cP`.

The line `model = ThreePhaseBlackOilModel(G, rock, fluid, 'disgas', false, 'vapoil', false)` creates a black-oil model for three-phase flow, using the previously defined grid `G`, rock properties `rock`, and fluid properties `fluid`. The parameters `'disgas', false` and `'vapoil', false` specify that dissolved gas in oil and vaporized oil in gas are not considered, meaning that the oil and gas phases are treated as independent, without mass transfer between them.

Finally, `state = initResSol(G, 100*barsa, [0, 1, 0])` initializes the reservoir state. The entire reservoir is assigned an initial pressure of `100 barsa` (100 bar), and the fluid saturations are set to `[0, 1, 0]`, meaning that the reservoir is initially fully saturated with oil (`1` for oil saturation), with no water (`0`) or gas (`0`).

```
T = 30*year;
pv = poreVolume(G, rock);
irate = sum(pv)/(T*4);
dT_target = 190*day;
dt = rampupTimesteps(T, dT_target, 10);
schedule.step.val = dt;
```

The variable `T = 30*year` sets the total simulation duration to 30 years. The command `pv = poreVolume(G, rock)` calculates the total pore volume of the reservoir by integrating the porosity values over the grid cells.

The injection rate `irate` is then computed using `irate = sum(pv)/(T*4)`. This formula distributes the total pore volume over the simulation time, assuming that four injectors will be operating continuously. This ensures that fluid is injected at a rate proportional to the reservoir's storage capacity over the given timeframe.

The variable `dT_target = 190*day` sets the target time step size to 190 days, defining the interval at which the reservoir state will be updated during the simulation. The command `dt = rampupTimesteps(T, dT_target, 10)` generates a time-stepping schedule that gradually increases the step size over the first `10` time steps before stabilizing at `dT_target`.

Finally, `schedule.step.val = dt` assigns the computed time step values to the simulation schedule, ensuring that the solver follows the defined time-stepping strategy.

```
W = [];
W = verticalWell(W, G, rock, 1, 1, [], 'Name', 'I1', 'radius',
5*inch, 'Type', 'rate', 'Val', irate, 'comp_i', [1, 0, 0],
'sign', 1);
W = verticalWell(W, G, rock, cartDims(1), 1, [], 'Name', 'I2',
'radius', 5*inch, 'Type', 'rate', 'Val', irate, 'comp_i', [1, 0,
0], 'sign', 1);
W = verticalWell(W, G, rock, 1, cartDims(2), [], 'Name', 'I3',
'radius', 5*inch, 'Type', 'rate', 'Val', irate, 'comp_i', [1, 0,
0], 'sign', 1);
W = verticalWell(W, G, rock, cartDims(1), cartDims(2), [],
'Name', 'I4', 'radius', 5*inch, 'Type', 'rate', 'Val', irate,
'comp_i', [1, 0, 0], 'sign', 1);
```

The variable `W = []` initializes an empty well structure. The function `verticalWell(...)` is then used to define four vertical injection wells at different positions in the grid. Each well is configured with specific properties such as well radius, injection rate, and injected fluid composition.

The first well is defined as `W = verticalWell(W, G, rock, 1, 1, [], 'Name', 'I1', 'radius', 5*inch, 'Type', 'rate', 'Val', irate, 'comp_i', [1, 0, 0], 'sign', 1)`. This places the well at `(i=1, j=1)`, meaning the lower-left corner of the grid in the x-y plane. The empty brackets `[]` indicate that the well spans the entire reservoir depth in the z-direction. The `'Name', 'I1'` assigns the well name as "I1". The well radius is set to `5 inches`, and it operates as a rate-controlled injector (`'Type', 'rate'`). The injection rate is given by `irate`, which was previously computed based on the pore volume and total simulation time. The parameter `'comp_i', [1, 0, 0]` specifies that this well injects only water (100% water, 0% oil, 0% gas). The `'sign', 1` parameter indicates that this is an injection well (positive sign for injectors, negative for producers).

Similar commands are used to define three additional injection wells at different locations:

- `I2` is placed at the lower-right corner `(i=cartDims(1), j=1)`.
- `I3` is placed at the upper-left corner `(i=1, j=cartDims(2))`.
- `I4` is placed at the upper-right corner `(i=cartDims(1), j=cartDims(2))`.

All four injection wells are configured to inject water at the same rate (`irate`) and are positioned symmetrically at the four corners of the reservoir model. These wells will help drive the displacement of oil toward the central production well, which will be defined later.

```
I = ceil(cartDims(1)/2);
J = ceil(cartDims(2)/2);
W = verticalWell(W, G, rock, I, J, [], 'Name', 'P1', 'radius',
5*inch, 'Type', 'bhp', 'Val', 100*barsa, 'comp_i', [0, 1, 0],
'sign', -1);
```

The variables `I = ceil(cartDims(1)/2)` and `J = ceil(cartDims(2)/2)` compute the coordinates for the center of the reservoir grid. The function `ceil()` is used to round up the values of `cartDims(1)/2` and `cartDims(2)/2`, ensuring that the well is placed at the middle of the x and y grid dimensions, respectively. This results in the production well being positioned at the center of the reservoir in the x-y plane.

The command `W = verticalWell(W, G, rock, I, J, [], 'Name', 'P1', 'radius', 5*inch, 'Type', 'bhp', 'Val', 100*barsa, 'comp_i', [0, 1, 0], 'sign', -1)` defines the production well with the following characteristics:

- It is named "P1" and placed at the grid location `(I, J)`, which corresponds to the central cell.
- The well radius is set to `5 inches`.
- The well type is defined as `'bhp'`, meaning that it is controlled by a bottom-hole pressure (BHP). The value `'Val', 100*barsa` specifies that the well operates at a bottom-hole pressure of 100 bar (converted from barsa, which is the unit for pressure).
- The `'comp_i', [0, 1, 0]` parameter defines the well's fluid composition, indicating that it only produces oil (100% oil, 0% water, 0% gas).
- The `'sign', -1` parameter indicates that this is a production well (negative sign for producers).

```
W_water = W;
W_gas = W;

fori = 1:numel(W)
  ifW(i).sign > 0
       W_gas(i).compi = [0, 0, 1];
  end
end
```

The line `W_water = W` creates a copy of the previously defined wells `W` for water injection, so the same wells will be used for injecting water. Similarly, `W_gas = W` creates another copy of the wells for gas injection. Initially, both `W_water` and `W_gas` are identical copies of `W`, which contains the four injection wells.

The `for i = 1:numel(W)` loop iterates over each well in the `W` array. Inside the loop, an `if` statement checks whether the well is an injector by evaluating `W(i).sign > 0`. This condition ensures that only the injectors (those with a positive `sign`) are modified.

For each injector well, the following line is executed: `W_gas(i).compi = [0, 0, 1]`. This sets the composition of the fluid being injected

into the gas injection wells. Specifically, `compi = [0, 0, 1]` means that these wells will inject 100% gas and no water or oil, overriding the initial composition for water injection.

As a result, `W_water` will maintain the original composition for water injection (100% water), and `W_gas` will be configured to inject only gas (100% gas). These modified well lists will then be used later in the simulation to alternate between water and gas injection.

```
schedule.control = [struct('W', W_water); struct('W', W_gas)];
schedule.step.control = (mod(cumsum(dt), 2*dT_target) >=
dT_target) + 1;
```

The command `schedule.control = [struct('W', W_water) struct('W', W_gas)]` sets up the control structure for the simulation, alternating between water injection and gas injection. The `schedule.control` array contains two structures, one for water injection (`W_water`) and one for gas injection (`W_gas`). Each structure defines the corresponding set of injection wells for each type of fluid. This ensures that the wells will alternate between injecting water and gas during the simulation.

The next line `schedule.step.control = (mod(cumsum(dt), 2*dT_target) >= dT_target) + 1` defines how the schedule alternates the injection types.

`cumsum(dt)` calculates the cumulative sum of the time steps, representing the total time elapsed during the simulation.

`mod(cumsum(dt), 2*dT_target)` computes the remainder when the cumulative time is divided by twice the target time step (`2*dT_target`). This ensures that the schedule alternates between water and gas injection at intervals of `dT_target`.

`(mod(cumsum(dt), 2*dT_target) >= dT_target)` checks if the current time has reached or exceeded half of the alternating period (`dT_target`). If true, it will assign a value of `1`, indicating that water injection is active; otherwise, it will assign a value of `2`, indicating that gas injection is active.

This logic ensures that the simulation alternates between water and gas injection based on the time steps, with each injection type being active for `dT_target` duration before switching to the other.

```
[ws, states] = simulateScheduleAD(state, model, schedule);
```

The command `[ws, states] = simulateScheduleAD(state, model, schedule)` invokes the `simulateScheduleAD` function to simulate the reservoir behavior over the defined time steps.

- `state` represents the initial conditions of the reservoir, including pressure and fluid saturations (initialized earlier in the code).

- `model` is the reservoir simulation model, which includes the grid, rock properties, fluid properties, and the flow model (in this case, a three-phase black oil model).
- `schedule` defines the time steps and control actions (in this case, alternating water and gas injection) during the simulation.

The function `simulateScheduleAD` runs the simulation and returns two outputs:

- `ws` contains the well data, including injection and production rates, for each time step.
- `states` contains the reservoir state at each time step, including the pressure and saturations for water, oil, and gas.

This step begins the simulation and stores the results in `ws` and `states`, which can later be used for analysis or visualization.

```
figure;
fori = 1:length(states)
clf;

subplot(1, 2, 1);
plotCellData(G, states{i}.s(:,1), 'EdgeColor', 'none');
colorbar;
caxis([0 1]);
xlabel('X-Distance [m]');
ylabel('Y-Distance [m]');
zlabel('Z-Distance [m]');
title(sprintf('Water Saturation at Timestep %d', i));
axis equal tight;

subplot(1, 2, 2);
plotCellData(G, states{i}.s(:,3), 'EdgeColor', 'none');
colorbar;
caxis([0 1]);
xlabel('X-Distance [m]');
ylabel('Y-Distance [m]');
zlabel('Z-Distance [m]');
title(sprintf('Gas Saturation at Timestep %d', i));
axis equal tight;

drawnow;
pause(0.1);

if~isvalid(gcf)
break;
end
end
```

This section of the code creates a dynamic visualization of the water and gas saturations in the reservoir over time, displaying the evolution of these parameters during the simulation.

The command `figure` initializes a new figure window for plotting the results. The `for i = 1:length(states)` loop iterates through the different states of the reservoir at each time step, as stored in the `states` array returned by the simulation. Each iteration represents the simulation at a particular time step.

Within each iteration, the code performs the following actions:

1. Water Saturation Visualization: The command `subplot(1, 2, 1)` creates a subplot on the left side of the figure to visualize the water saturation. `plotCellData(G, states{i}.s(:,1), 'EdgeColor', 'none')` plots the water saturation (`s(:,1)`, the first column of the saturation vector) on the reservoir grid. The colorbar is added with `colorbar`, and `caxis([0 1])` normalizes the color scale, ensuring that the saturation is shown on a scale from 0 (no water) to 1 (fully saturated with water). Axis labels are added for the x, y, and z dimensions with `xlabel('X-Distance [m]')`, `ylabel('Y-Distance [m]')`, and `zlabel('Z-Distance [m]')`. The title of the subplot, `title(sprintf('Water Saturation at Timestep %d', i))`, indicates the current time step. `axis equal tight` ensures that the axes are scaled equally and that the plot tightly fits the data.
2. Gas Saturation Visualization: Similarly, `subplot(1, 2, 2)` creates a subplot on the right side of the figure to visualize the gas saturation. `plotCellData(G, states{i}.s(:,3), 'EdgeColor', 'none')` plots the gas saturation (`s(:,3)`, the third column of the saturation vector) on the grid. The same color normalization (`caxis([0 1]);`), axis labeling, and title settings are applied for the gas saturation plot. The title is dynamically updated to show the current time step: `title(sprintf('Gas Saturation at Timestep %d', i))`.
3. Dynamic Plot Update: `drawnow` forces the figure to update in real-time, so the changes in the plots can be viewed interactively as the simulation progresses.
4. Pause and Check for Window Closure: The `pause(0.1)` command pauses the execution for 0.1 seconds, allowing time for the user to observe the plots before moving to the next time step. The condition `if ~isvalid(gcf)` checks if the figure window is still open. If the figure has been closed by the user, the loop breaks and stops the dynamic plotting.

To facilitate the extraction of saturation values, an empty matrix (`results = []`) is first initialized to store the results for each simulation time step. The variable `states`, which holds the simulation output, contains a series of reservoir states representing fluid distributions at different time instances.

The `for i = 1:length(states` loop iterates through all recorded time steps. Within each iteration, the fluid phase saturations are extracted using the field `s` from the `states` structure:

`states{i}.s(:,1)`: Retrieves water saturation values.
`states{i}.s(:,2)`: Retrieves oil saturation values.
`states{i}.s(:,3)`: Retrieves gas saturation values.

Each of these variables is an array containing the phase saturations for all grid cells at a given time step.

To obtain a representative measure of phase distribution across the entire reservoir, the mean saturation for each phase is computed using the `mean()` function:

`mean(water_saturation)`: Computes the average water saturation.
`mean(oil_saturation)`: Computes the average oil saturation.
`mean(gas_saturation)`: Computes the average gas saturation.

Since these values range between 0 and 1, they are converted into percentages by multiplying by 100 for improved interpretability.

The computed values are stored in a results matrix using `results = [results; i, avg_oil_saturation, avg_water_saturation, avg_gas_saturation]`.

Once all time steps have been processed, the accumulated results are converted into a MATLAB table using `results_table = array2table(results, 'VariableNames', {'TimeStep', 'Oil_%', 'Water_%', 'Gas_%'})`. The `VariableNames` parameter assigns column labels to enhance clarity. The table format is preferred because it allows for easy data manipulation, visualization, and export to external analysis tools. Finally, the `disp(results_table)` function is used to display the formatted results table:

| TimeStep | Oil_% | Water_% | Gas_% |
|---|---|---|---|
| 1 | 99.999 | 0.0014326 | 0 |
| 2 | 99.997 | 0.0028652 | 0 |
| 3 | 99.994 | 0.0057308 | 0 |
| 4 | 99.989 | 0.011464 | 0 |
| 5 | 99.977 | 0.022937 | 0 |
| 6 | 99.954 | 0.04591 | 0 |
| 7 | 99.908 | 0.091964 | 0 |
| 8 | 99.816 | 0.18448 | 0 |
| 9 | 99.629 | 0.3708 | 0 |
| 10 | 99.254 | 0.74646 | 0 |
| 11 | 98.615 | 0.75504 | 0.6295 |
| 12 | 97.066 | 2.2937 | 0.64002 |
| 13 | 95.742 | 2.3339 | 1.9238 |
| 14 | 94.145 | 3.8892 | 1.9653 |
| 15 | 92.781 | 3.9455 | 3.2733 |
| 16 | 91.168 | 5.5122 | 3.3201 |

**Figure 86: Table of Time Step - Saturations (%)**

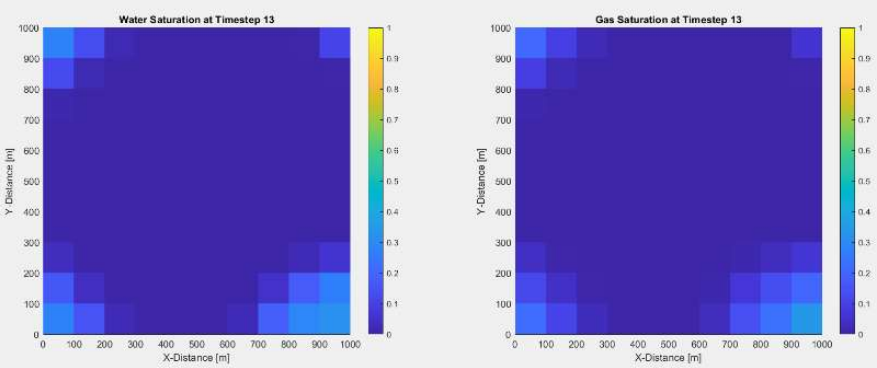


**Figure 87: Water and Gas Saturation at 13th Timestep.**

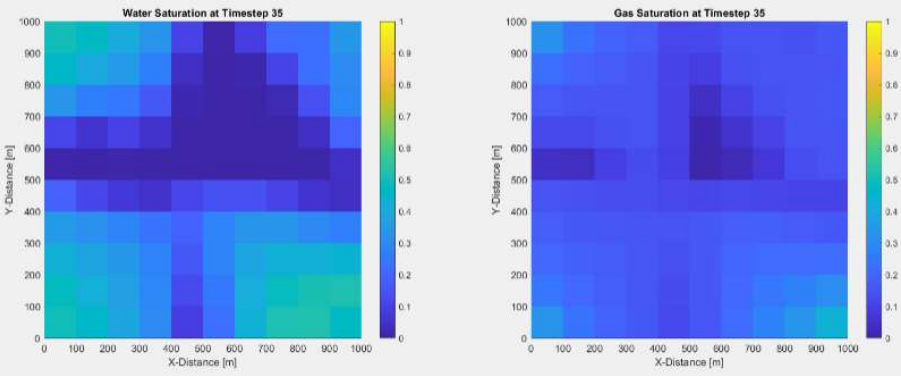


**Figure 88: Water and Gas Saturation at 35th Time step.**

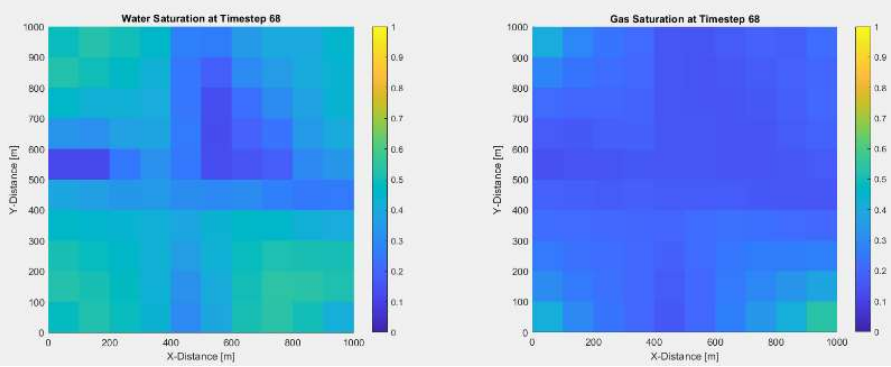


**Figure 89: Water and Gas Saturation at 68th Time step.**

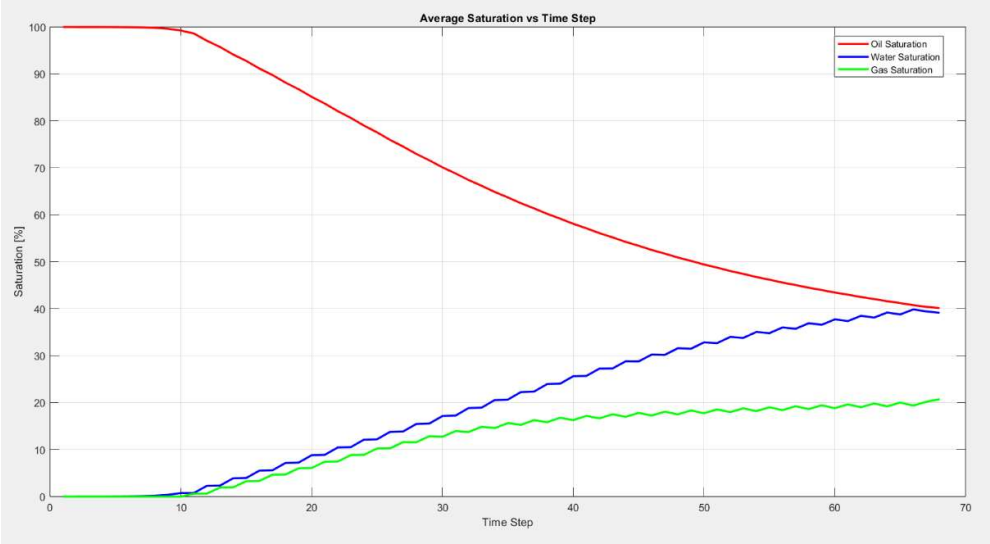


**Figure 90: Diagram of Saturation (%) – Time Steps**

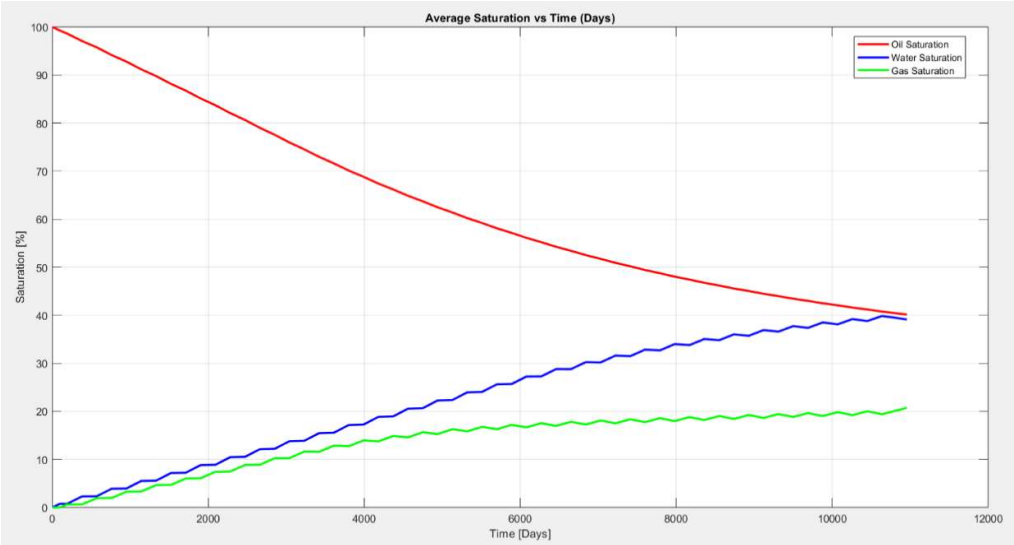


**Figure 91: Diagram of Saturation (%) - Time (days)**

The Saturation Diagram reveals the behavior of the three phases in the reservoir as a result of production, with a focus on the evolving interplay between oil displacement, water and gas injection.

At the start of the simulation, oil saturation is 100%, suggesting that the reservoir was initially entirely filled with oil. As time progresses, oil saturation declines steadily, primarily due to the displacement by injected water and gas. The sharp decrease in oil saturation is an indicator of successful oil extraction as a result of the water-alternating-gas (WAG) injection strategy.

The decline is nonlinear, with a steeper reduction in oil saturation during the first 20 time steps (from 99.99% to 85.13%). The largest decrease occurs between time steps 20-40, where oil saturation falls by ~27%. In the final phase (time steps 40-68), oil saturation continues to drop but at a slower rate, reaching ~40%. This suggests that the majority of the recoverable oil has been displaced, but some residual oil remains, possibly indicating that further enhanced oil recovery (EOR) techniques could be necessary.

Initially, water saturation is 0%, as the reservoir is entirely oil-filled. However, as water is injected into the reservoir, the water saturation begins to rise steadily. Water saturation increases at an accelerated rate between time steps 10-50. This rapid rise is indicative of the initial phases of water injection, where water is actively pushing oil towards the production wells.

The increase is characterized by three distinct phases:

- Initial Phase (1-10 time steps): A slow increase from 0% to approximately 0.75%.
- Middle Phase (10-40 time steps): A sharp increase from 0.75% to 25.6%, showing that water injection becomes more effective.
- Late Phase (40-68 time steps): A slower increase from 25.6% to 39.1%, indicating that the system is approaching an equilibrium, where the rate of water influx is balanced with oil displacement.

Gas saturation initially starts at 0%, and its gradual increase is indicative of gas injection. The gas breakthrough occurs around time step 11, marking the point where gas starts to replace oil in the reservoir. The gas saturation grows more slowly than water, which is typical since gas has a lower density and is less efficient at displacing oil compared to water. Gas saturation starts to rise, initially at a slow rate. By time step 40, gas saturation reaches ~16.3%, indicating significant gas mobility in the reservoir. From time step 50, the gas saturation stabilizes between 18-20%, suggesting that gas is either moving as free gas or is in solution (dissolved gas drive). This stabilization indicates that gas injection has reached a steady-state, with no further significant increase in gas saturation expected.

The graph clearly demonstrates that water and gas are actively displacing oil. Water reaches its breakthrough earlier than gas. The gas phase increases at a slower pace and stabilizes after a period of displacement. The oil saturation decline is nonlinear, with the most significant depletion occurring between time steps 20-40. This suggests that the WAG process, combining water and gas injections, is most effective during this period in displacing oil.

The system begins to reach steady-state conditions between time steps 60-68, as indicated by the plateau in both water and gas saturations. The rate of increase in water saturation slows down, and gas saturation stabilizes.

By the end of the simulation, oil saturation remains around 40%, indicating that while a significant amount of oil has been recovered (~60% reduction), further enhanced oil recovery methods may be necessary to improve the recovery factor.

### 4.3 Analysis of the Water-only Simulation

Moving on the WAG code will be modified to first simulate only water flow and then only gas flow. To achieve this, the code will be adjusted from the point where the two different flow types (gas and water) are introduced. By making this change, the effects of gas and water injection can be isolated, enabling a clearer understanding of how each fluid behaves in the reservoir under the given conditions.

```
mrstModule add ad-core ad-blackoil ad-props spe10 mrst-gui sequential
spe10 mrst-gui;
```

The required MRST modules are loaded to enable key functionalities. `ad-core` and ad-blackoil provide automatic differentiation and black-oil modeling capabilities. `ad-props` manages the rock and fluid properties, while `spe10` gives access to benchmark datasets. `mrst-gui` enables visualization, and sequential supports advanced well modeling.

```
cartDims = [10, 10, 10];
physDims = [1000, 1000, 10]*meter;
G = cartGrid(cartDims, physDims);
G = computeGeometry(G);
Gravity reset off;
```

A structured Cartesian grid of 10×10×10 cells is defined, representing a reservoir with physical dimensions of 1000m × 1000m × 10m. The `cartGrid` function generates the grid, and `computeGeometry` calculates its geometric properties, such as cell volumes and face areas. Gravity effects are ignored by setting gravity `off`.

```
rng(0);
poro = gaussianField(cartDims, [0.05, 0.3], 3, 8);
poro = poro(:);
perm = poro.^3 .* (5e-5)^2 ./ (0.81 * 72 * (1 - poro).^2);
```

The porosity field is generated using `gaussianField`, with values ranging between `0.05` and `0.3`. A fixed random seed ensures reproducibility. Permeability is calculated from porosity using a modified Kozeny-Carman relationship, linking porosity ϕ to permeability k. This step creates spatially correlated rock properties.

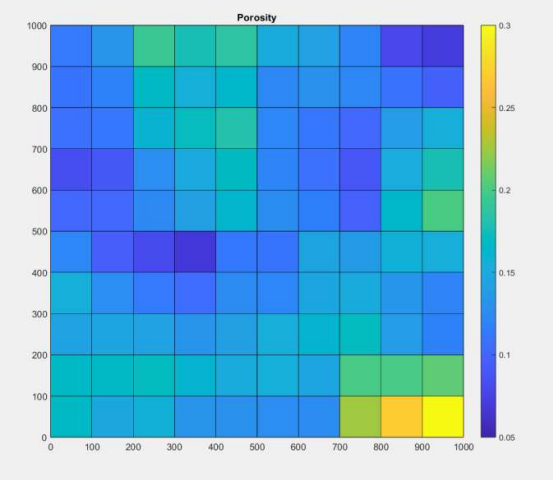


**Figure 92: Porosity Grid.**

```
rock = makeRock(G, perm, poro);
figure(1); clf;
plotCellData(G, rock.poro);
axis equal tight;
colorbar;
title('Porosity');
```

The porosity and permeability fields are combined into a rock structure using `makeRock`. The porosity distribution is visualized with `plotCellData`, ensuring the generated field aligns with the expected range and spatial characteristics.

```
pv = poreVolume(G, rock);
T = 30*year;
irate = sum(pv)/(T*4);
```

The pore volume for each grid cell is calculated using `poreVolume`, incorporating porosity and cell volume. The total injection rate is determined by dividing the cumulative pore volume over 30 years among four injectors.

```
makeInj = @(W, name, I, J, compi) verticalWell(W, G, rock, I, J, [],...
'Name', name, 'radius', 5*inch, 'sign', 1, 'Type', 'rate',...
'Val', irate, 'comp_i', compi);
W = [];
W = makeInj(W, 'I1', 1,           1,           []);
W = makeInj(W, 'I3', cartDims(1), cartDims(2), []);
W = makeInj(W, 'I4', 1,           cartDims(2), []);
W = makeInj(W, 'I2', cartDims(1), 1,           []);
```

Four injector wells are placed at the grid's corners, created using `verticalWell`. The injectors are configured with a fixed radius and injection rate. Their composition is left as a placeholder to allow future customization.

```
I = ceil(cartDims(1)/2);
J = ceil(cartDims(2)/2);
W = verticalWell(W, G, rock, I, J, [], 'Name', 'P1', 'radius', 5*inch,
...
'Type', 'bhp', 'Val', 100*barsa, 'comp_i', [1, 1, 1]/3, 'Sign', -1);
```

A producer well is placed at the grid center, with its bottom-hole pressure (BHP) fixed at 100 bars. The production composition is set to withdraw equal proportions of water, oil, and gas.

```
fluid = initSimpleADIFluid('phases', 'WOG', ...
'rho',    [1000, 700, 250], ...
'n',      [2, 2, 2], ...
'c',      [0, 1e-4, 1e-3]/barsa, ...
'mu',     [1, 4, 0.25]*centi*poise);
```

The fluid system is defined as a three-phase (Water-Oil-Gas) model. Density (`rho`), viscosity (`mu`), compressibility (`c`), and Corey exponents (`n`) are set for each phase. These parameters represent typical physical properties for subsurface fluids.

```
state = initResSol(G, 100*barsa, [0, 1, 0]);
```

The reservoir is initialized with a pressure of 100 bars and fully saturated with oil (`So=1, Sw=Sg=0`).

```
dT_target = 190*day;
dt = rampupTimesteps(T, dT_target, 10);
schedule.step.val = dt;
```

Simulation time steps are set using a ramp-up schedule, starting with small time steps and gradually increasing to a maximum of 190 days. This approach ensures numerical stability, particularly during early simulation periods.

```
fori = 1:numel(W)
ifW(i).sign > 0
W(i).compi = [1, 0, 0];
end
end
schedule.control = struct('W', W);
schedule.step.control = ones(size(schedule.step.val));
```

The injectors are configured to inject only water ([1,0,0][1, 0, 0][1,0,0]). The schedule structure specifies a single control configuration for all simulation time steps.

```
[wsWaterOnly, statesWaterOnly] = simulateScheduleAD(state, model,
schedule);
```

The water-only injection scenario is simulated using simulateScheduleAD, solving the governing equations for the defined reservoir, wells, and fluids.

```
figure;
plotToolbar(G, statesWaterOnly);
axistight;
plotWell(G, W);
title('Water-only injection');
```

The results are visualized using plotToolbar, showing the spatial evolution of fluid saturation and pressure over time. The well placement is also displayed.

**Comparison of GOR with and without Gravity:**

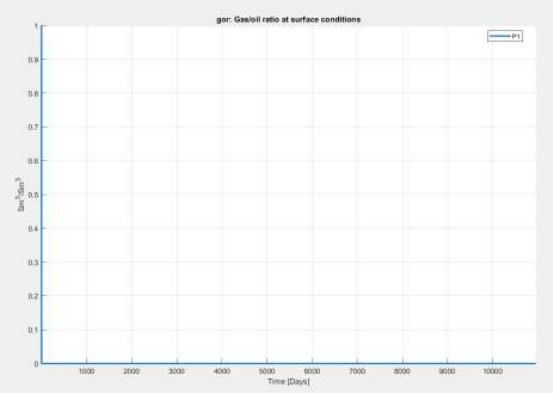


**Figure 93: GOR - GRAVITY ON.**

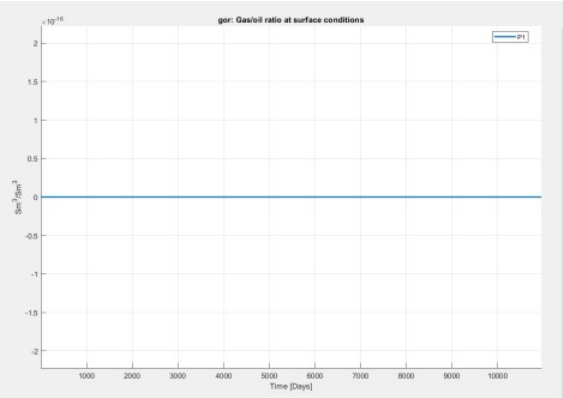


**Figure 94: GOR - GRAVITY OFF.**

The two graphs depict the gas-oil ratio (GOR) at surface conditions for the simulation, comparing scenarios with gravity on and gravity off. Unlike prior comparisons, both cases result in flat and consistent GOR trends throughout the entire simulation timeline.

In the first graph, where gravity is active, the GOR remains constantly zero for the entire duration of the simulation. This indicates that no gas is being produced alongside the oil. The flat line suggests that, under these gravity-influenced conditions, there is no gas breakthrough or associated gas production at the well during the modeled period.

In the second graph, where gravity is inactive, the GOR is also constant but exhibits a small, fixed negative value. This negative GOR is likely due to numerical artifacts or initial conditions within the model setup. Despite this discrepancy, the flat trend signifies a similar outcome—no gas is produced under the conditions simulated without the effects of gravity.

When comparing the two scenarios, it becomes evident that no gas production occurs in either case. This suggests that the reservoir dynamics or injection parameters prevent gas mobilization into the wellbore. The only noticeable difference between the two cases is the zero GOR observed in the gravity-on scenario versus the negative GOR in the gravity-off scenario. The latter is likely a result of computational or initialization errors rather than a physical phenomenon.

Overall, these results suggest that the specific design of the simulation, whether considering gravity effects or not, does not lead to gas liberation at surface conditions during the modeled timeline.

**Comparison of qOs with and without Gravity:**

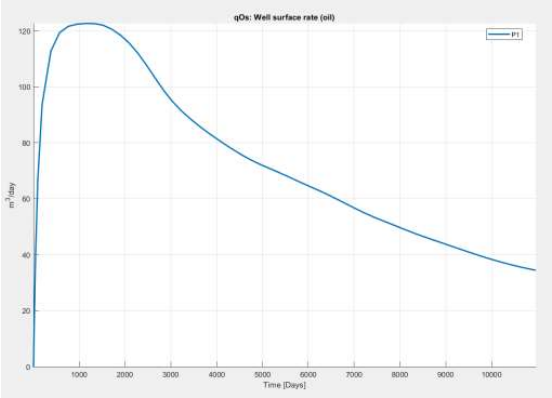


**Figure 95: QOS - GRAVITY ON.**

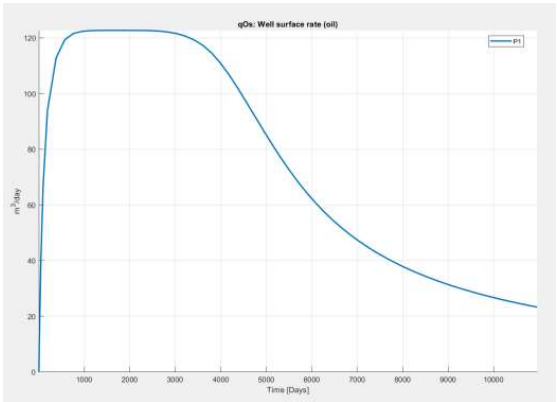


**Figure 96: QOS - GRAVITY OFF.**

The first graph shows the rate of oil surface flow (qOs) as a function of time (in days) under the influence of gravity (Gravity On). Initially, the flow rate increases from zero to approximately 120 m³/day. This initial increase is due to the reservoir's initial pressure and the efficient exploitation of the oil. After a maximum value around 1000 days, the flow rate starts to decrease gradually, reaching below 40 m³/day by the end of the 10,000-day period. This

decrease is associated with the drop in reservoir pressure and the increasing influence of gravity, which traps the oil in the lower zones of the reservoir, reducing the exploitation of the upper zones.

The second graph illustrates the same oil flow rate (qOs) when gravity is turned off (Gravity Off). The curve starts from zero and gradually increases to the same maximum value of approximately 120 $m^3$/day around 1000 days. However, the subsequent decrease is steeper compared to the gravity-on case, with the flow rate falling below 20 $m^3$/day by the end of the period. The faster decrease is due to the absence of gravity, which does not trap the oil in the lower zones, leading to faster depletion of the reservoir.

Comparing the two graphs, we observe that in the initial phase of both cases, the flow rate increases in a similar manner, reaching a maximum value. At this stage, the effect of gravity is negligible, as the reservoir pressure dominates. However, in the reduction phase, there are significant differences. In the gravity-on case, the decrease in the flow rate is more gradual, indicating that gravity assists in the exploitation of the lower zones. In contrast, with gravity off, the decrease is steeper as the production depends solely on the drop in reservoir pressure.

In the long term, gravity plays a crucial role in maintaining productivity. With gravity on, the oil remains in the lower zones, ensuring a longer production duration. Without gravity, the reservoir depletes faster, leading to lower overall output. Overall, the presence of gravity enhances long-term production efficiency, while its absence accelerates the depletion of reserves, highlighting the importance of gravity in reservoir management.

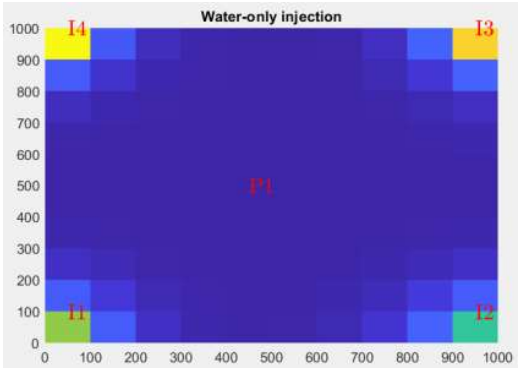


**Figure 97: Water-only injection - 1st Timestep.**

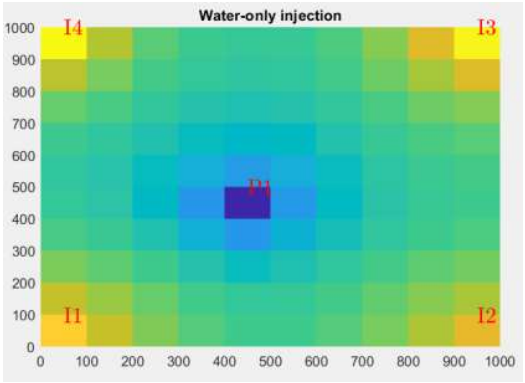


**Figure 98: Water-only injection - 10th Timestep.**

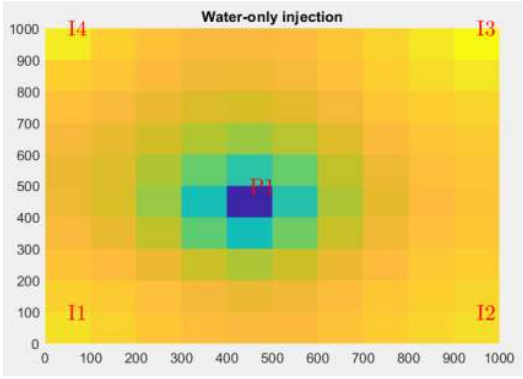


**Figure 99: Water-only injection - 68th Timestep.**

### 4.4 Analysis of the Gas-only Simulation

```
fluid = initSimpleADIFluid('phases', 'WOG', ...
'rho',    [1000, 700, 250], ...
'n',      [2, 2, 2], ...
'c',      [0, 1e-4, 1e-3]/barsa, ...
'mu',     [1, 4, 0.25]*centi*poise);
```

The `initSimpleADIFluid` function initializes a fluid model with three phases: Water (W), Oil (O), and Gas (G). Physical properties such as density (`rho`), viscosity (`mu`), Corey exponents (`n`), and compressibility (`c`) are defined for each phase. These parameters simulate realistic fluid flow and phase behavior within the reservoir.

```
model = ThreePhaseBlackOilModel(G, rock, fluid, 'disgas', false,
'vapoil', false);
```

The `ThreePhaseBlackOilModel` is created for immiscible three-phase flow simulation. Dissolved gas and vaporized oil options are disabled, focusing on phase interactions without mass transfer between phases. The model incorporates the previously defined grid, rock, and fluid properties.

```
state = initResSol(G, 100*barsa, [0, 1, 0]);
```

The reservoir is initialized using `initResSol` with a uniform pressure of 100 bars and complete oil saturation. The saturation vector `[0,1,0][0, 1, 0]` indicates no water or gas initially present in the reservoir.

```
dT_target = 190*day;
dt = rampupTimesteps(T, dT_target, 10);
schedule.step.val = dt;
```

Simulation time steps are configured to gradually increase using `rampupTimesteps`. This approach starts with smaller intervals to ensure stability during early phases, then transitions to larger time steps up to 190 days.

```
fori = 1:numel(W)
ifW(i).sign > 0
W(i).compi = [0, 0, 1];
end
end
schedule.control = struct('W', W);
schedule.step.control = ones(size(schedule.step.val));
```

The injectors are updated for gas-only injection by setting their composition vector to `[0,0,1][0, 0, 1]`. A single control scheme is applied for all time steps, ensuring consistent gas injection throughout the simulation.

```
[wsGasOnly, statesGasOnly] = simulateScheduleAD(state, model,
schedule);
```

The gas-only injection simulation is executed using `simulateScheduleAD`, producing results for pressure, saturation, and well performance over time.

```
figure;
plotToolbar(G, statesGasOnly);
axistight;
plotWell(G, W);
title('Gas-only injection');
```

The plotToolbar function visualizes the reservoir's temporal evolution, showing pressure and fluid saturations within the grid. Well placements are overlaid for context.

```
df = get(0, 'DefaultFigurePosition');
figure('position', df.*[1, 1, 2, 1]);
names = {'qWs', 'qOs', 'qGs'};
hold on;
t = cumsum(schedule.step.val)/day;
colors = lines(3);
ws = wsGasOnly
fori = 1:numel(names)
    q = -getWellOutput(wsGasOnly, names{i}, 5);
stairs(t, q*day, '-o', 'color', colors(i, :));
end
legend('Water', 'Oil', 'Gas');
xlabel('Time [days]');
ylabel('Production at standard conditions [m^3/day]');
axis tight;
```

Production rates for water, oil, and gas are extracted using `getWellOutput` and plotted against time. The plot provides insights into phase-specific production trends under gas-only injection. Each phase's production rate is displayed as a separate curve, making it easier to analyze individual contributions

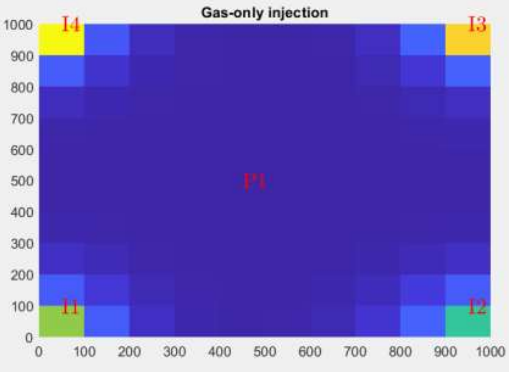


**Figure 100: Gas-only injection - 1st Timestep.**

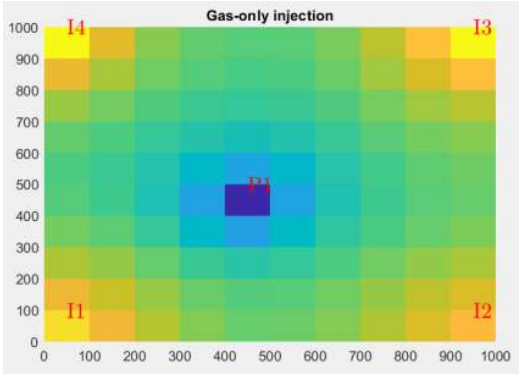


**Figure 101: Gas-only injection - 10th Timestep.**

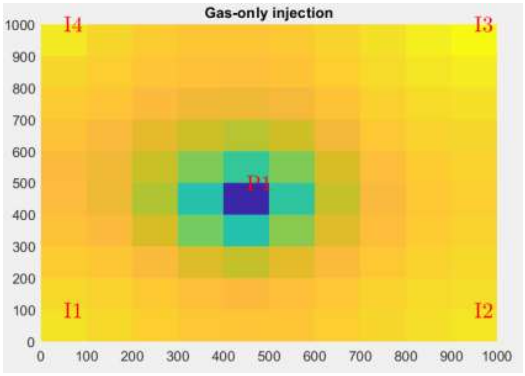


**Figure 102: Gas-only injection - 68th Timestep.**

## Comparison of GOR with and without Gravity:

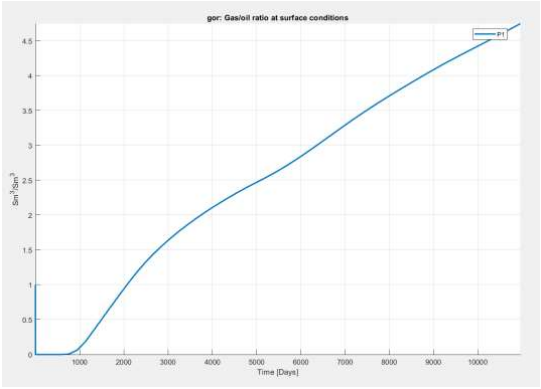


**Figure 103: GOR - GRAVITY ON.**

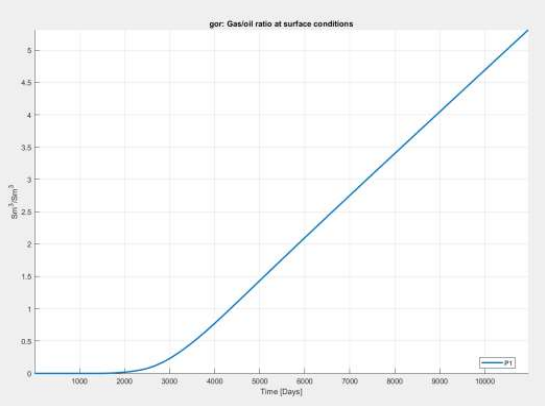


**Figure 104: GOR - GRAVITY OFF.**

These two diagrams show the change in GOR (Gas Oil Ratio) over time for two different cases: Gravity On and Gravity Off.

The first diagram (Gravity On) shows that the GOR starts from zero and gradually increases over time. The curve has an initial period of steeper growth until about 2000 days, after which the increase remains linear but with a gentler slope. This indicates that the presence of gravity contributes to the gradual separation between oil and gas, allowing gas to be recovered at an increasing rate as the oil-bearing layer depletes.

In the second diagram (Gravity Off), the graph clearly follows a linear form throughout the entire time period. The GOR increases steadily without any change in the slope of the curve. This happens because, without the influence of gravity, there is no natural force contributing to further separation between gas and oil. As a result, gas production increases in a more uniform manner.

In the case of Gravity On, the GOR increases more steeply at first due to the effect of gravity, which facilitates the accumulation of gas in the upper regions of the formation. After that, the curve becomes more stable and linear. In the case of Gravity Off, the curve maintains a constant linear form from start to finish, indicating the absence of any additional physical effects. In both cases, the final GOR value is nearly the same, reaching about 4.5 $Sm^3/Sm^3$ after 10,000 days. However, the path to this value differs significantly. In Gravity On, gravity contributes to creating a nonlinear curve at first, reflecting the acceleration of the separation process. In Gravity Off, the system evolves solely due to pressure, without the contribution of gravity.

The physical properties of the formation, such as porosity and permeability, can affect the rate of gas recovery. The pressure drop in the formation influences the GOR ratio, while the presence or absence of water can affect the mobility of the gas. The presence of gravity creates a more complex GOR development curve. The diagrams demonstrate the importance of gravity in the behavior of the formation and gas production. With gravity, the process is initially accelerated due to natural separation forces, while without gravity, the evolution is more stable and monotonic.

**Comparison of qGs with and without Gravity:**

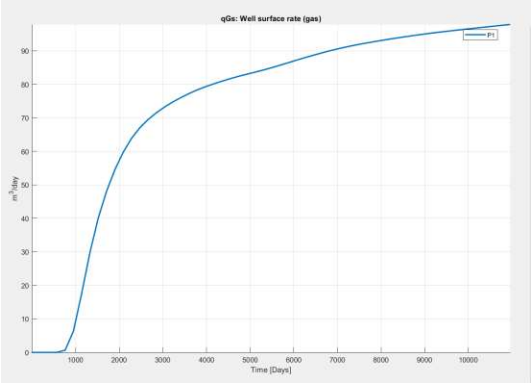


**Figure 105: qGs - GRAVITY ON.**

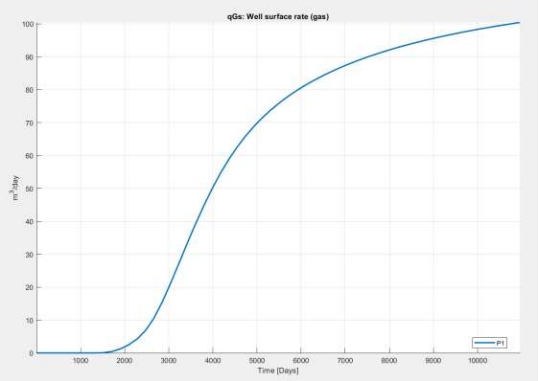


**Figure 106: qGs - GRAVITY OFF.**

In the first diagram (Gravity On), the gas flow rate (qGs) is shown as a function of time with gravity enabled. The curve shows a rapid increase during the initial production days (about up to 2000 days) due to the mobilization of gas from the formation into the reservoir. After the initial phase, there is a gradual decrease in the slope, with the flow stabilizing around 100 m³/day at the end of the period (10,000 days). The flow starts from zero and gradually reaches about 100 m³/day. The presence of gravity causes heavier phases (e.g., liquids) to settle, allowing the gas to move more freely towards the wellbore, affecting the flow. The increase in flow is relatively faster in the beginning due to the formation pressure driving gas production.

In the second diagram (Gravity Off), the behavior of the gas flow is shown without the contribution of gravity. The increase is smoother compared to the first diagram. The curve also reaches about 100 m³/day, but the stabilization is less steep and occurs more slowly. As in the first diagram, the flow starts from zero and ends up at about 100 m³/day. The absence of gravity reduces the mobilization of gas, as the system does not naturally separate the phases. The gas moves mainly through pressure forces. The process is more stable but slower, as there is no additional "help" from gravity to push the gas.

Comparing the two diagrams, in the first diagram (Gravity On), the flow increases more steeply in the beginning, while in the second diagram (Gravity Off), the increase is smoother and slower. The activation of gravity helps the gas move more quickly towards the

wellbore, accelerating production in the early days. In both cases, the flow stabilizes around the same maximum value (100 m³/day), indicating that the total volume of gas remains the same. In conclusion, gravity (gravity on) accelerates the initial gas production, but in both cases, the final output is comparable.

## 4.5 Comparison of Production between WAG, Water-only and Gas-only

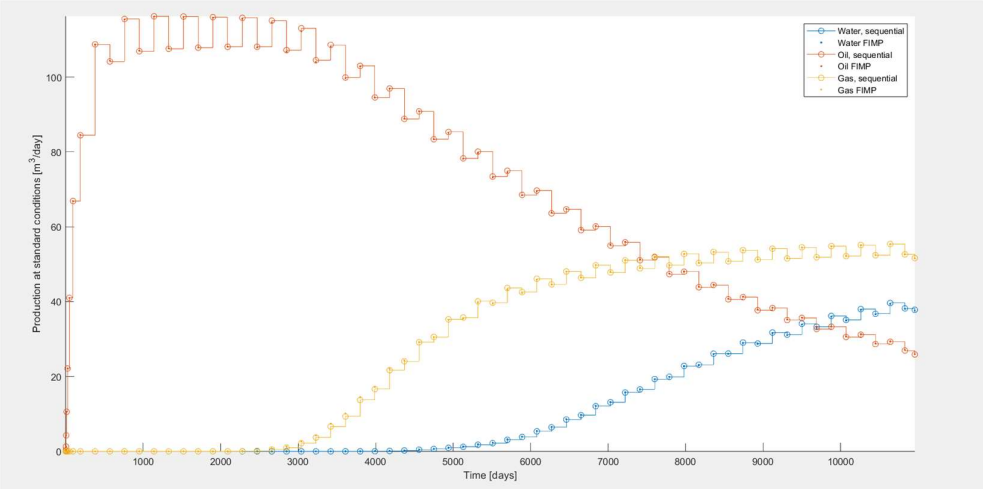


**Figure 107: Production with WAG.**

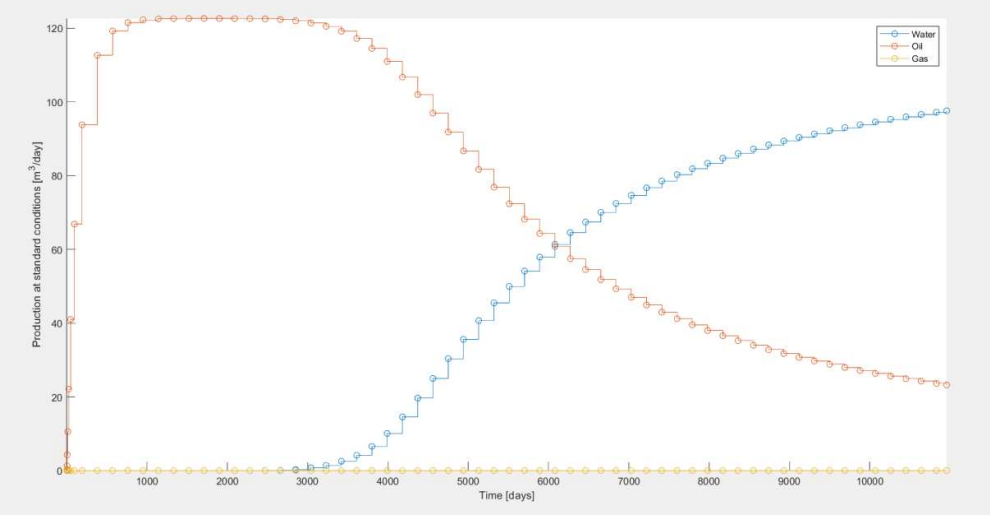


**Figure 108: Production with Water-only.**

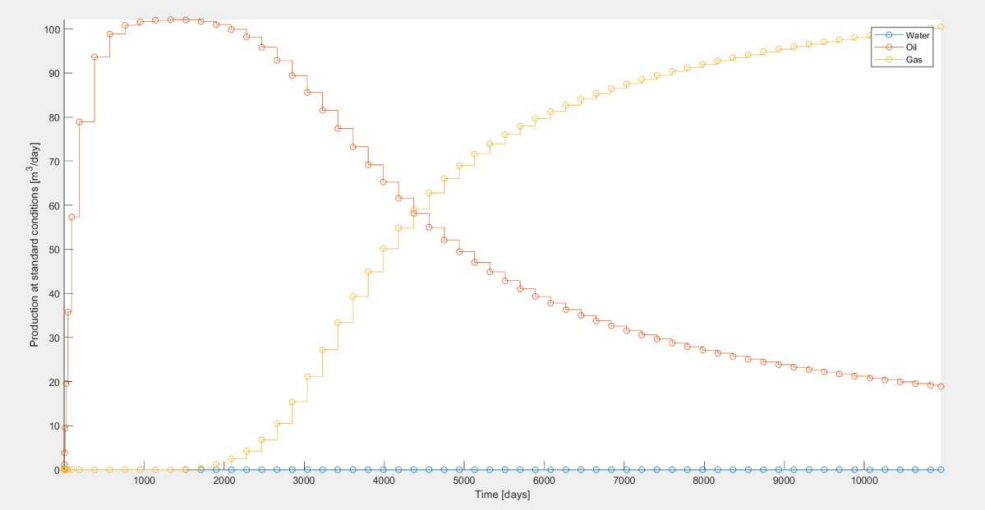


**Figure 109: Production with Gas-only.**

In the first diagram (left), we observe the WAG (Water-Alternating-Gas) injection method. It is evident that during the first 3,000 days of production, only oil is produced at the surface, with no water cut or gas cut observed. After the first 3,000 days, gas production begins at the surface, indicating the occurrence of gas cut. Finally, at around 5,000 days, water cut also takes place. From that point onward, water, gas, and oil are all produced at the surface. As oil production declines, the production of water and gas increases. This behavior is expected, as water and gas are continuously injected through the four injection wells.

In the second diagram (center), we observe the water injection method. Here, water cut occurs earlier than in the WAG case, around 3,000 days after production begins. At the same time, as expected, no gas cut is observed since there is no gas injection.

In the third diagram (right), we observe the gas injection method. Here, gas cut occurs much earlier than in the WAG case and earlier than water cut in the water injection scenario. This is due to the lower density of gas compared to water and oil, allowing it to migrate more quickly through the reservoir pores toward the surface.

## *Conclusions*

The engagement with MRST in this dissertation highlighted the significance of reservoir simulations as a fundamental tool for understanding hydrocarbon reservoir behavior and optimizing oil recovery strategies. At the same time, its suitability for constructing both simple and complex grids was reaffirmed.

As demonstrated through the processing of the SPE1 problem, MRST approaches the effectiveness of the industry-standard Eclipse software, which is widely employed by petroleum companies for reservoir exploitation simulations. By analyzing the parameters of the SPE1 problem, various scenarios were generated, and the corresponding results were extracted, serving as a guide for determining optimal production parameters.

At a subsequent stage, a more complex code was utilized, involving modifications to the WAG problem to develop two additional scenarios. The analysis of WAG revealed that water injection alone can enhance overall production; however, it may lead to premature water breakthrough in production wells. Conversely, gas injection improves oil displacement, but the uniformity of the flow is not always guaranteed.

A comparative assessment of the three scenarios indicated that the WAG method proved to be the most efficient, as it combines the advantages of both preceding techniques. Simulations demonstrated that alternating water and gas injection enables better control of the displacement front, ultimately maximizing hydrocarbon recovery. However, achieving peak efficiency necessitates the adjustment of operational parameters based on the specific characteristics of each reservoir.

Finally, this study underscores the need for further research and analysis, both to enhance mathematical modeling and to advance production strategies. The integration of optimization algorithms, such as gradient-based and genetic algorithms, for determining the optimal production rate and injection scheduling could lead to more efficient and controlled reservoir management. The use of these algorithms can improve the system's ability to respond to the continuously changing reservoir conditions, thereby enhancing overall production performance. Furthermore, simulating more realistic scenarios that include heterogeneities in the porous medium, permeability, and initial saturation would increase the complexity of the models, while also enhancing the fidelity of the results, making them more representative of real-world conditions. Expanding to three-dimensional geometries, such as complex and intricate grid geometries (e.g., "egg" type grids), and examining heterogeneous formations could offer new perspectives for improving the understanding and management of reservoirs. These extensions would increase the accuracy of the models and enable the optimization of production under realistic, complex conditions.

# References

## Greek Sources


1. Γιώτης, Α., & Μαρινάκης, Δ. (2023). *Μηχανική κοιτασμάτων υδρογονανθράκων*. Εκδόσεις Πολυτεχνείου Κρήτης.
2. Μπογάτσας, Χ. (n.d.). *Περιγραφή διαφόρων τύπων εξέδρων άντλησης πετρελαίου και ιδιαίτερα χαρακτηριστικά τους* [Master's thesis, Ακαδημία Εμπορικού Ναυτικού Μακεδονίας].
3. Πιάνος, Μ. (2021). *Υπολογιστική μελέτη συνθηκών ροής κατά τη δευτερογενή ανάκτηση πετρελαίου από ετερογενείς και διαστρωματωμένους ταμιευτήρες πετρελαίου* [Master's thesis, Πολυτεχνείο Κρήτης].
4. Σκετόπουλος, Θ. (2012). *Μελέτη παραγωγής πετρελαϊκών ρευστών από υπόγειους ταμιευτήρες με χρήση υπολογιστικού μοντέλου τύπου τανκ* [Master's thesis, Πολυτεχνείο Κρήτης].
5. Συμεωνίδης, Α. (2014). *Αποτίμηση παραγωγικού δυναμικού ταμιευτήρα της Ανατολικής Μεσογείου με τη μέθοδο ισοζυγίου μάζας* [Master's thesis, Πολυτεχνείο Κρήτης].
6. Ξανθοπούλου, Π. (2021). *Κατάταξη των πετρωμάτων – ταμιευτήρων πετρελαίου και φυσικού αερίου. Εισαγωγή στις βασικές αρχές εκτίμησης των αποθεμάτων* [Master's thesis, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης].


## International Sources


1. Adewumi, M. (n.d.). *Fundamentals of petroleum and natural gas engineering*. Pennsylvania State University.
2. Alamooti, A. M., & Malekabadi, F. K. (2018). *An introduction to enhanced oil recovery*. Petroleum Science and Technology.
3. American Association of Petroleum Geologists. (n.d.). *Reservoir modeling for simulation purposes*. AAPG Wiki. https://wiki.aapg.org/Reservoir_modeling_for_simulation_purposes
4. Amyx, J. W., Bass, D. M., & Whiting, R. L. (1960). *Petroleum reservoir engineering: Physical properties*. Journal of Petroleum Technology. https://doi.org/10.2118/1311-G
5. Awang, M. (2017). *ICIPEG 2016*. Springer.
6. Bear, J. (1972). *Dynamics of fluids in porous media*. Elsevier.
7. Bett, G. K. (2018). *Geochemical evaluation of the hydrocarbon potential of source rocks in the Anza Basin* [Master's thesis, University of Nairobi].
8. Blunt, M. J., Jackson, M. D., Piri, M., & Valvatne, P. H. (2001). *Review of WAG field experience*. Reservoir Evaluation & Engineering. https://doi.org/10.2118/74679-MS
9. Bradley, H. B., & Gipson, F. W. (1987). *Petroleum engineering handbook*. Society of Petroleum Engineers.
10. Brattekås, B., & Haugen, M. (2020). *Explicit tracking of CO2-flow at the core scale using micro-Positron Emission Tomography (μPET)*. Journal of Natural Gas Science and Engineering.

11. Buckley, W. F., & Leverett, S. E. (1942). *Mechanism of fluid displacement in sands*. Transactions of the AIME. https://doi.org/10.2118/942107-G
12. Carlson, M. R. (1997). *Practical reservoir simulation using spreadsheets*. Chemical Engineering Science. https://doi.org/10.1016/S0009-2509(97)80003-2
13. Chart Industries. (n.d.). *Where does crude oil come from? And 5 other things you should know about the Earth's 'black gold'*. https://www.chartindustries.com/Articles/Where-Does-Crude-Oil-Come-From-And-5-Other-Things
14. Chen, C. E. H. (n.d.). *Reservoir engineering fundamentals*. Rice University. http://www.owlnet.rice.edu/~ceng571/CHAP5.pdf
15. Chen, C. E. H. (n.d.). *Reservoir simulation*. Rice University. http://www.owlnet.rice.edu/~ceng571/CHAP6.pdf
16. Chen, Z. (2020). *Development of a toolbox for simulation of integrated reservoir-production system* [Master's thesis, Texas A&M University].
17. Das, A., Das, N., Pandey, P., & Pandey, P. (2025). *Microbial enhanced oil recovery: Process perspectives, challenges, and advanced technologies for its efficient applications and feasibility*. Archives of Microbiology.
18. Durlofsky, L. J. (n.d.). *Reservoir simulation and the mathematics of oil recovery*. SIAM News. https://www.siam.org/publications/siam-news/articles/reservoir-simulation-and-the-mathematics-of-oil-recovery/
19. Durlofsky, M. J., & Aziz, A. (2010). *Advanced techniques for reservoir simulation and modeling of porous media*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898718942
20. Energy Education. (n.d.). *Oil and gas reservoir*. https://energyeducation.ca/encyclopedia/Oil_and_gas_reservoir
21. Energy Education. (n.d.). *Oil and gas traps*. https://energyeducation.ca/encyclopedia/Oil_and_gas_traps
22. Ewing, R. E. (1978). *Interpretation of well-block pressures*. SPE Journal. https://doi.org/10.2118/5988-PA
23. Folk, A. (n.d.). *Reservoir simulation fundamentals*. Norwegian University of Science and Technology. https://andreas.folk.ntnu.no/papers/eacm-ressim.pdf
24. Glover, P. (n.d.). *Reservoir drives*. In *Formation evaluation MSc course notes*. University of Leeds.
25. Hu, X. (2017). *The physical properties of reservoir fluids*. In *Physics of petroleum reservoirs*. Springer.
26. IHS Energy. (n.d.). *Relative permeability correlations*. https://www.ihsenergy.ca/support/documentation_ca/Harmony_Enterprise/2019_3/content/html_files/ref_materials/calculations/relative_permeability_correlations.htm
27. Iqbal, M. I., & Kudapa, V. K. (2025). *Oil well production mechanism training manual on well production operations for non-production engineers (Oil and gas production operations)*. River Publishers.
28. Ishimwe, D. (2014). *Petroleum systems and elements of petroleum geology*. SPE Connect. https://connect.spe.org/blogs/donatien-ishimwe/2014/09/05/petroleum-systems-and-elements-of-petroleum-geology
29. Kleppe, J. (n.d.). *Reservoir engineering basics*. Norwegian University of Science and Technology. https://www.ipt.ntnu.no/~kleppe/TPG4150/text.pdf

30. Lie, K.-A. (2019). *An introduction to reservoir simulation using MATLAB/GNU Octave: User guide for the MATLAB Reservoir Simulation Toolbox (MRST)*. Cambridge University Press.
31. Lie, K.-A. (2023). *User guide to flow diagnostics post processing*. SINTEF.
32. Lie, K.-A., & Moyner, O. (2021). *Advanced modeling with the MATLAB Reservoir Simulation Toolbox*. Cambridge University Press.
33. Lie, K.-A., Krogstad, S., Ligaarden, I. S., Natvig, J. R., Nilsen, H., & Skaflestad, B. (2012). *MRST-AD – an open-source framework for rapid prototyping and evaluation of reservoir simulation problems*. Reservoir Evaluation & Engineering. https://www.researchgate.net/publication/273002665_MRST-AD_-_an_Open-Source_Framework_for_Rapid_Prototyping_and_Evaluation_of_Reservoir_Simulation_Problems
34. Magoon, L. B., & Beaumont, E. A. (1994). *The petroleum system*. American Association of Petroleum Geologists.
35. Mahdian, M., Huang, L. Y., Kirk, D. W., & Jia, C. Q. (2020). *Water permeability of monolithic wood biocarbon*. Microporous and Mesoporous Materials.
36. Martin, F. D. (1996). *Reservoir engineering*. In W. Lyons (Ed.), *Standard handbook of petroleum and natural gas engineering*. Gulf Professional Publishing.
37. MathWorks. (n.d.). *MRST overview*. https://www.mathworks.com/solutions/energy/mrst-matlab-reservoir-simulation-toolbox.html
38. McCain, W. D. (1990). *The properties of petroleum fluids* (2nd ed.). PennWell.
39. Mihai, B., & Ciuperca, C. L. (2019). *Identification of fluid contacts by using formation pressure data and geophysical well logs*. Geophysical Prospecting.
40. Moslehi, S. (n.d.). *Petroleum engineering tutorials* [Videos]. YouTube. https://www.youtube.com/@SajjadMoslehi
41. Mustafiz, S., & Islam, M. R. (2008). *State-of-the-art petroleum reservoir simulation*. Petroleum Science and Technology.
42. Niazi, A. M. K., Jahren, J., Mahmood, I., & Javaid, H. (2019). *Reservoir quality in the Jurassic sandstone reservoirs located in the Central Graben, North Sea*. Marine and Petroleum Geology.
43. Olukoga, T. A. (2024). *Machine learning applications for optimization of oil and gas system development* [Doctoral dissertation, University of Louisiana at Lafayette].
44. OPM Project. (n.d.). *OPM Flow*. https://opm-project.org/?page_id=19
45. Pathak, A. K. (2021). *Petroleum reservoir management - Considerations and practices*. CRC Press.
46. Peaceman, D. W. (1977). *Fundamentals of numerical reservoir simulation*. Elsevier.
47. Physics of petroleum reservoirs. (2017). Springer.
48. Raabe, G., & Jortner, S. (2022). *Universal well control*. Gulf Professional Publishing.
49. Rajput, S. (2016). *Geological controls for gas hydrates and unconventional book*. Baker Hughes.
50. Rajput, S., & Pathak, R. K. (2025). *Seismic exploration to reservoir excellence*. Springer.
51. ResFrac Corporation. (n.d.). *ResFrac simulator*. https://www.resfrac.com
52. Rock Flow Dynamics. (n.d.). *tNavigator*. https://rfdyn.com/software/
53. Sami Consulting. (2024). *bp energy outlook 2024*. bp. https://www.bp.com/content/dam/bp/business-sites/en/global/corporate/pdfs/energy-economics/energy-outlook/bp-energy-outlook-2024.pdf

54. Schlumberger. (n.d.). *Gas-oil contact (GOC)*. Schlumberger Oilfield Glossary. https://glossary.slb.com/en/terms/g/gas-oil_contact
55. ScienceDirect. (n.d.). *Capillary pressure*. https://www.sciencedirect.com/topics/earth-and-planetary-sciences/capillary-pressure
56. ScienceDirect. (n.d.). *Darcy's law*. https://www.sciencedirect.com/topics/engineering/darcys-law
57. ScienceDirect. (n.d.). *Formation volume factor*. https://www.sciencedirect.com/topics/engineering/formation-volume-factor
58. ScienceDirect. (n.d.). *Saturated oil reservoir*. https://www.sciencedirect.com/topics/engineering/saturated-oil-reservoir
59. SINTEF Digital. (n.d.). *MRST – MATLAB reservoir simulation toolbox*. https://www.sintef.no/projectweb/mrst/
60. Speight, J. G. (2017). *Rules of thumb for petroleum engineers*. Wiley.
61. Technical University of Crete. (n.d.). *MATLAB installation guide to personal computer*. https://www.tuc.gr/el/to-polytechneio/ilektronikes-ypiresies/matlab-logismiko-programmatismoy-kai-arithmitikis-ypologistikis
62. Terry, R. E., & Rogers, J. B. (2014). *Applied petroleum reservoir engineering* (3rd ed.). Prentice Hall.
63. Thakur, N. K. (n.d.). *Geological studies of hydrocarbon resources*. Council of Scientific and Industrial Research.
64. Tiab, D., & Donaldson, E. C. (2024). *Porosity and permeability*. Petroleum Science and Technology.
65. Top Dog Engineer. (n.d.). *Frictional pressure loss in a pipeline*.
66. Udegbunam, E. C. (2022). *Data-driven proxy to reservoir simulation*. Geoenergy Science and Engineering. https://doi.org/10.1016/j.geoen.2022.211513
67. 2023 International Conference on Energy Engineering. (2024). Springer.