Offshore Solvent-Based Huff ‘n’ Puff for Injector Well Improved Oil Recovery

by

© Tristan Strong

A Thesis submitted to the

School of Graduate Studies

in partial fulfillment of the requirements for the degree of

Master of Engineering

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

October 2017

St. John’s Newfoundland and Labrador

Abstract

The solvent-based huff ‘n’ puff process has been used with great success in heavy oils with CO_2 as the solvent. The aim of this work was to explore the use of natural gas as a solvent in the huff ‘n’ puff process and apply this to the Hibernia reservoir by creating a numerical reservoir simulator to complete this study. A one-dimensional compositional reservoir model was created using MATLAB. The simulator was developed to be able to use a Cartesian as well as radial co-ordinate system, allowing for simulation of multiple processes which aided in the validation of the model. The model uses a robust flash calculation which was tested against known experimental values, as were all fluid prediction models. The reservoir flow was compared to known analytical solutions, using both constant-rate and constant-pressure boundaries. This was done to ensure the simulator could adequately handle the required boundary conditions for simulation of the huff ‘n’ puff process.

Slim-tube experiments were simulated with Hibernia oil using realistic reservoir properties, in order to determine the minimum miscibility pressure for different gases to be tested in the huff ‘n’ puff process. Simulation of the huff ‘n’ puff was successful for the huff and puff phases, but issues were encountered when simulating the puff phase. It was found that it was not possible to model the three-phase huff ‘n’ puff process in the one-dimensional simulator that was developed. Although the huff ‘n’ puff process was not able to be modelled using the developed simulator, the simulator was validated on many different levels and there are many other useful processes that can be simulated using this model. It is also a great foundation for future work studying the huff ‘n’ puff and many other gas injection processes.

Acknowledgements

I would like to take this opportunity to express my appreciation and gratitude to my supervisors, Dr. Lesley James and Dr. Thormod Johansen. Thank you for your guidance in all aspects of my graduate program. It is through your guidance that I was able to learn so much and grow as an academic and professional, and I am very grateful for this. Without your technical expertise I would not have been able to complete this work.

I would also like to thank all the members of the Hibernia EOR research team. It is through our conversations and brainstorming sessions that I was able to get through some of the mental roadblocks that I had during this project.

Finally, I would like to thank HMDC, Chevron and RDC for financial support of this project.

Table of Contents

List of Figures

List of Tables

Nomenclature

a	Attraction parameter
b	Co-volume parameter
c_p	Compressibility of rock [Pa^{-1}]
c_w	Compressibility of water [Pa^{-1}]
$f_{i\alpha}$	Fugacity of component i in phase α [Pa, MPa]
f_w	Fractional flow of water
k	Absolute rock permeability [m^2]
$k_{r\alpha}$	Relative permeability of phase $(\alpha = w, o, g)$
$k_{ro\alpha}$	Relative permeability of oil with respect to phase α $(\alpha = w, g)$
J	Number of grid blocks
K	Equilibrium ratio
L	Liquid mole fraction of the hydrocarbons
M_α	Phase mobility [$(Pa \cdot s)^{-1}$]
M_T	Total mobility [$(Pa \cdot s)^{-1}$]
MW	Molecular Weight [g/mol]
n_α	Corey model exponent of phase $(\alpha = w, o, g)$
p_{bh}	Bottom-hole pressure [Pa, MPa]
p_α	Pressure of phase $(\alpha = w, o, g)$ [Pa, MPa]
p_c	Critical pressure [Pa, MPa]
$P_{co\alpha}$	Capillary pressure of α-phase with oil [Pa, MPa]

P_σ	Parachor
PV	Pore volume [m^3]
q_i	Molar flow rate of component i [(g·mol)/(m^3·s)]
q_α	Molar flow rate of phase $(\alpha = w, o, g)$ [(g·mol)/s]
Q_α	Volumetric flow rate of phase $(\alpha = w, o, g)$ [m^3/s]
r	Radius [m]
r_e	Drainage radius [m]
r_w	Wellbore radius [m]
R	Universal gas constant = 8.314 [J/(K·mol)]
S_{gc}	Critical gas saturation
S_{org}	Residual oil saturation to gas
S_{orw}	Residual oil saturation to water
$S_{w_{bt}}$	Water saturation at breakthrough
$\overline{S}_{w_{bt}}$	Average water saturation behind the front at breakthrough
S_{wc}	Connate water saturation
S_α	Saturation of phase $(\alpha = w, o, g)$
t	Time [s, hrs]
t_{bt}	Time to breakthrough [s, hrs]
T_c	Critical Temperature [K]
T_r	Reduced temperature [K]
T_α	Transmissibility of phase $(\alpha = w, o, g)$

u_α Volumetric flux of phase $(\alpha = w, o, g)$ [m/s]

v Volume [m^3]

v_c Critical volume [m^3]

V Vapor mole fraction of the hydrocarbons

W_{id} Dimensionless water influx

x Distance in primary direction [m]

$x_{i\alpha}$ Concentration of component i in phase α $(\alpha = o, g)$

z_i Mole fraction of component in the hydrocarbon mixture

Z_α Compressibility factor of phase $(\alpha = o, g)$

ε Newton-Raphson convergence factor

θ Water scaling factor

κ_{ij} Binary interaction between components i and j

μ_α Viscosity of phase $(\alpha = w, o, g)$ [Pa·s]

ξ_α Molar density of phase $(\alpha = w, o, g)$ [g·mol/m^3]

ρ_r Reduced density

σ_{og} Interfacial tension between oil and gas phases [N/m]

ϕ Porosity

$\varphi_{i\alpha}$ Fugacity coefficient of component in phase $(\alpha = o, g)$

ψ Accumulation term [g·mol/m^3]

ω Acentric factor

Superscripts

l Newton-Raphson pressure equation iteration level

n	Time level
s	Flash calculation iteration level

Subscripts

g	Gas
i	Component index
j	Grid block index
$j \pm 1/2$	Grid block interface index
o	Oil
w	Water
α	Phase index

Note: All equations are assumed to be in SI units unless otherwise noted.

Unit Conversions

Value	Field	SI	Field to SI	SI to Field
Area	acre	m^2	2.471 E-4	4.047 E+3
Compressibility	psi^{-1}	Pa^{-1}	6.895 E+3	1.450 E-4
Density	lb/ft	kg/m^3	6.243 E-2	1.602 E+1
Length	ft	m	3.281 E+0	3.048 E-1
Permeability	mD	m^2	1.013 E+15	9.869 E-16
Pressure	psi	Pa	1.450 E-4	6.895 E+3
Rate	stb/d	m^3/s	5.434 E+5	1.840 E-6
Viscosity	cP	Pa•s	1.000 E+3	1.000 E-3
Volume	barrel	m^3	6.290 E+0	1.590 E-1

Abbreviations

BCG	Block-Centered Grid
CFL	Courant-Friedrichs-Lewy
EOR	Enhanced Oil Recovery
EOS	Equation of State
IOR	Improved Oil Recovery
IMPECS	Implicit Pressure Explicit Composition and Saturation
IMPES	Implicit Pressure Explicit Saturation
LBC	Lohrenz, Bray and Clark
MMP	Minimum Miscibility Pressure
PDG	Point-Distributed Grid
PR	Peng Robinson
SRK	Suave-Redlich-Kwong

Chapter 1 Introduction

1.1 Overview of Reservoir Simulation

Reservoir simulation is used across the oil and gas industry for solving reservoir engineering problems (Abou-Kassem et al., 2013). These types of problems can cover all types of oil and gas recovery processes. A reservoir simulator mathematically models the behaviour of the physical fluids in the reservoir, as well as the reservoir rock itself. This means that in order to simulate an oil and gas reservoir, there first must be a mathematical model to describe the system. The mathematical model is based on laws of conservation of mass, momentum, and energy (Aziz and Settari, 1979). The numerical model describing an oil and gas reservoir draws from the basic laws governing fluid flow, and applies them to fluid flow in porous media. The development of the mathematical model for the work completed in this Thesis is described in detail in Chapter 3.

There are two types of reservoir simulation: black oil modelling and compositional modelling. Black oil modelling was developed first, as this is the simpler form of reservoir simulation. This does not take into account the composition of the oil, but instead assumes only three major components in a reservoir: water, oil, and gas. Typically, black oil modelling is used for modelling primary and secondary recovery. This method has been used with great success in reservoir simulation, and is still used today as it is adequate for modelling many recovery processes such as water injection and immiscible gas injection.

Compositional reservoir modelling is used to model more complicated reservoir processes which are referred to as tertiary recovery or enhanced oil recovery (Chen et al., 2006). The compositional reservoir models each component of the reservoir fluid individually, and is useful in examining complex processes such as miscible gas injection.

1.2 Purpose of Work

The purpose of this work is to examine the possibility of using the solvent-based huff 'n' puff process in conditions experienced offshore Newfoundland, and specifically to apply this process to injector wells for improved oil recovery (IOR). IOR involves increasing the production of a well after its production has begun to decline, which can include enhanced oil recovery (EOR) techniques. EOR encompasses any process that increases oil production, whether it be field wide or for a single well. Residual oil can be left in the vicinity of an injector well, limiting gas injectivity as well as leaving valuable oil unrecovered in the near well region. The production of this residual oil can be aided through the use of the solvent-based huff 'n' puff process.

The most common solvent used in solvent-based huff 'n' puff is CO_2. When running the CO_2 huff 'n' puff process onshore, the CO_2 will generally be provided directly from a pipeline or from CO_2 trucks. In general, there are no CO_2 pipelines running to offshore facilities, therefore CO_2 availability becomes an issue. The huff 'n' puff process has rarely been documented in offshore usage before, but in a CO_2 huff 'n' puff project offshore Vietnam one of the main problems was the availability of CO_2 (Ha et al., 2012). It can become quite costly to ship CO_2 offshore which can render the process economically unviable.

This work examines the use of natural gas as the solvent for the solvent-based huff 'n' puff process in a light oil reservoir. Natural gas has not been thoroughly studied for use in this process, and it is readily available in an offshore environment which could improve the economics of using this process, as well as provide a use for the natural gas produced in certain offshore environments. Although much of the literature review is for the CO_2 huff 'n' puff process, previous studies have

shown that some of the mechanisms which can lead to IOR could also apply to the natural gas huff ‘n’ puff process.

1.3 Scope of Thesis

A comprehensive literature review was completed regarding the huff ‘n’ puff process and how it works. Through this literature review, knowledge was gained on how the process works and what injection parameters are important to the process. The literature review on the huff ‘n’ puff process is summarized in Chapter 2. Completing this thorough review gave insight to what work has already been completed, as well as what would be useful to study.

In order to examine the possibility of using a solvent-based huff ‘n’ puff process offshore, a one-dimensional isothermal compositional reservoir simulator was created in MATLAB to simulate the process. The description of how this model was created is outlined in Chapter 3. This involved a very comprehensive study of reservoir simulation; many different textbooks were used to create the mathematical model for compositional near well reservoir simulation. A combination of research into reservoir simulation and knowledge of general numerical simulation and programming was required to complete this model.

The model inputs and boundary conditions were determined through literature review, and different parameters were studied to determine their effect on the natural gas huff ‘n’ puff process. The model used to evaluate the natural gas huff ‘n’ puff process along with the results and discussion are described in Chapter 4. Once the case studies were run, the conclusions and recommendations for future work were listed in Chapter 5. Figure 1.1 shows a concept map of the work completed in this thesis. The work started with a literature review of the huff ‘n’ puff process in parallel with a literature review of compositional reservoir simulation and numerical

simulation in MATLAB. Once knowledge on reservoir simulation in MATLAB was adequately developed, the compositional reservoir simulator was created and then validated. The literature review on the huff 'n' puff process aided in creating case studies to examine the natural gas huff 'n' puff process, and then finally these case studies were evaluated using the developed simulator.

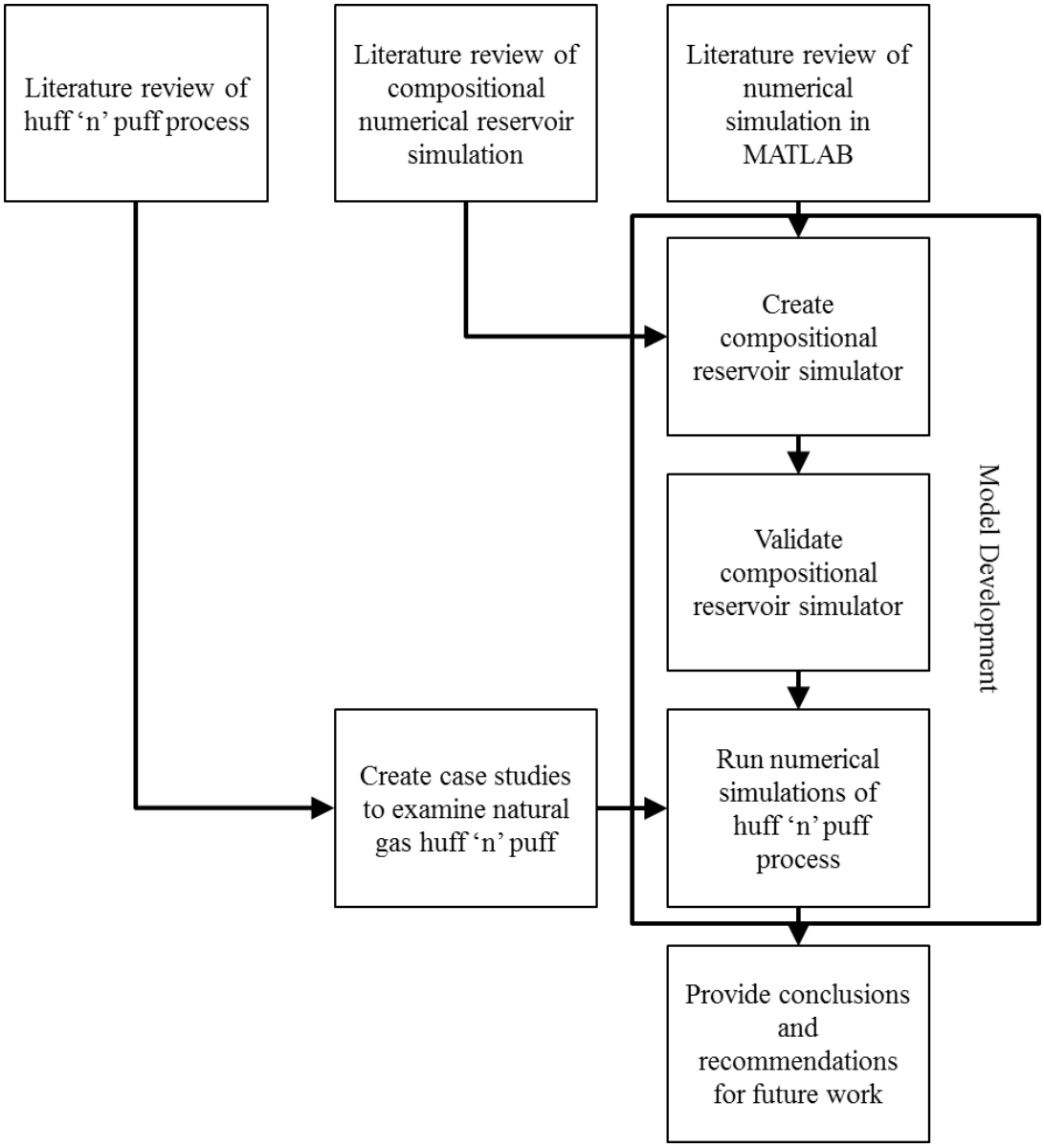


Figure 1.1 – Concept Map of Thesis

Chapter 2 Literature Review of Huff ‘n’ Puff Process

2.1 Background

When a well is shut-in due to economic constraints, residual oil is left in the vicinity of the well. An improved oil recovery (IOR) process known as the solvent-based huff ‘n’ puff process has been used to extend the life of wells as they near the end of their economic life. This method has become popular over recent years as it is easy to implement and generally does not require a large up front capital commitment, as long as the well is equipped for gas usage. It can be used as a typical EOR process and also an IOR process for residual oil well cleanup. The solvent-based huff ‘n’ puff process involves three stages; injection, a shut-in period, and production. There are various mechanisms which contribute to the IOR of this process, these are described in the proceeding section. The injection stage, known as the huff cycle, is when the solvent is injected into the well. The shut-in period allows for the solvent to interact with the formation oil. It is during this stage that some of the mechanisms of IOR, such as oil swelling and oil viscosity reduction, take place. When the shut-in period is over, the well is returned to production, which is known as the puff cycle. Huff ‘n’ puff is a cyclic solvent injection process; therefore this scheme can be repeated multiple times to increase the recovery factor. This process works as a single well EOR or an IOR method.

The primary solvent used in the solvent-based huff ‘n’ puff processes is CO_2 and mixtures of CO_2 with other components. CO_2 is widely used in EOR, and it has been investigated in terms of EOR since the 1950’s. Although the CO_2 huff ‘n’ puff process was not used until the 1970’s, CO_2 was still used for other EOR methods. The phase behaviour of CO_2 and paraffin systems was studied by Poettman and Katz (1945). The main mechanisms in which CO_2 could contribute to EOR were determined to be the swelling of oil, and the reduction of viscosity upon dissolution of CO_2 in the

oil. These mechanisms caused miscible CO_2 applications to become quite popular in the 1960's, as injecting CO_2 under miscible conditions allows for the highest solubility and increased mass transfer between the CO_2 and the oil. Thermal EOR methods were also popular at this time, with steam injection being widely used. Steam injection could be quite costly, and similar to miscible CO_2 applications steam injection could not penetrate deep enough to provide EOR for deeper wells (Khatib et al., 1981).

One method of steam injection which was used was what is called a steam huff 'n' puff process. This involved injecting steam, allowing it to soak, and then producing the oil. As with most thermal methods of EOR this was developed for use in heavy oil fields. The procedure used in the solvent-based huff 'n' puff process is very similar to the procedure which was used in the steam huff 'n' puff process. The solvent based huff 'n' puff process was also initially developed for use in heavy oil fields. Solvent-based huff 'n' puff was first seen in a patent by P.C. Keith in 1969, but this patent did not describe the process as it is used today. Keith described a cyclic injection of a mixture of CO_2 and steam, as at the time he believed steam may still be necessary to promote desirable EOR. The solvent-based huff 'n' puff process as it is used today was described in detail by Patton et al. (1982). There are a few key differences between the solvent-based huff 'n' puff process and miscible solvent flooding processes which had been used. The solvent-based huff 'n' puff process works in a single well, where miscible flooding is generally injected in one well, producing oil from another well sweeping the larger field. The huff 'n' puff process uses injection under immiscible conditions, which allows the solvent to propagate deeper into the reservoir than what could be achieved through miscible flooding. This enables the solvent to interact with more formation oil, which in turn increases the recovery factor in the near well region.

2.2 The Huff 'n' Puff Process

The huff 'n' puff process was developed in order to enhance the oil recovery in deeper wells. It is generally used as a single well IOR method. The solvent is injected in small treatments and does not typically travel more than 60 m from the injection well (Patton et al., 1982). There are three stages to the huff 'n' puff; injection, shut-in, and production. The injection stage involves injection of the solvent under immiscible conditions in order to bypass the oil and propagate deep into the reservoir through fingering and channeling (Liu et al., 2005). After the injection stage the drainage area of the near well region is pressurized before the shut-in period. The shut-in period is when the flow into the well is shut off, which allows the solvent to soak into the formation and oil and mass transfer occurs. The length of the shut-in period has been noted as an important parameter in the huff 'n' puff process (Mohammed-Singh et al., 2006), it can last up to several weeks (Liu et al., 2005). Although the thermodynamic conditions for miscibility may not be met, the solvent is generally still soluble in the oil. The solubility of CO_2 in oil has been shown to increase with pressure, as was studied by Barclay and Mishra (2016) when developing the following correlation for CO_2 solubility in light oils.

$$sol = (0.36913 - 0.00106T)\ln(p) + (0.01280 - 0.00160T) \tag{2.1}$$

where sol is the solubility of CO_2 as a mole fraction, p is pressure in MPa, and T is temperature in °C. Through this equation it is seen that the solubility of CO_2 in light oils is logarithmically proportional to pressure. At low pressures only a small portion of the solvent will dissolve in the oil, which is why it is important that the solvent contacts as much oil as possible through fingering and channeling (Miller, 1990). Diffusion can take a long time to reach equilibrium, which is why the shut-in period has been thought to be an important factor. After the well has been shut-in for an adequate period of time it is returned to production by reducing the pressure to operating

conditions. The oil surrounding the well has now mixed with the lighter injection gas and is easier to produce due to mechanisms discussed in the proceeding section. When the well is returned to operating conditions it will see an increase in oil recovery. The solvent-based huff 'n' puff process can be repeated multiple times to produce the remaining residual oil left in the vicinity of the well. This process has shown to have an economically viable increase in oil recovery after up to 3 cycles in the field (Mohammed-Singh et al., 2006). The number of cycles which are most favourable will depend on the economics of the individual project.

2.3 Mechanisms Contributing to EOR

There are many mechanisms that have been shown to contribute to the increase in oil recovery; those which have shown to be common amongst the majority of CO_2 huff 'n' puff processes are (Liu et al., 2005; Mohammed-Singh et al., 2006):

1. Oil swelling
2. Oil viscosity reduction
3. Gas relative permeability hysteresis
4. Gas penetration
5. Extraction of lighter components of oil by CO_2

Some mechanisms are common amongst both miscible and immiscible CO_2 EOR methods such as oil swelling and oil viscosity reduction. These have been known since the 1940's and were examined in early CO_2 EOR applications. Other mechanisms which are unique to immiscible CO_2 injection are; the extraction of lighter components of oil by CO_2 and gas penetration.

2.3.1 Oil Swelling

Swelling of oil has been noted to be an important recovery mechanism for the CO_2 huff 'n' puff process. Dissolution of CO_2 in the formation oil can cause the oil to swell, which can lead to IOR through mobilizing more oil. When producing a two-phase system, a higher oil swelling factor will increase the oil phase saturation and leave less residual oil in the reservoir (Liu et al., 2005). This effect is simulated through the equation of state flash calculation, described in section 3.4.3.

2.3.2 Oil Viscosity Reduction

Another mechanism contributing IOR of the CO_2 huff 'n' puff process is oil viscosity reduction. This is also caused by the dissolution of CO_2 into the formation oil. This mechanism is common to other CO_2 EOR processes as well, the reduced oil viscosity allows oil to flow more easily, improves the mobility ratio and similarly to the oil swelling effect the reduction of viscosity will reduce the residual oil saturation left in the reservoir (Liu et al., 2005). In the simulator, oil viscosity is calculated based on composition thus as the oil composition changes the viscosity accurately reflects these changes as described in section 3.4.4.

2.3.3 Gas Relative Permeability Hysteresis

During the huff 'n' puff process relative permeability hysteresis may be invoked during the production phase. Through interactions between the injection gas and formation water during the injection and shut-in phase, the gas-oil relative permeability function may experience hysteresis for the production phase (Liu et al., 2005). It has been noted during previous simulations that the gas relative permeability hysteresis has been a major cause of oil recovery in the huff 'n' puff process (Denoyelle and Lemonnier, 1987: Haines and Monger, 1990). As mentioned in Chapter 4, this mechanism is not included in this simulator due to complexity.

2.3.4 Gas Penetration

The CO_2 huff 'n' puff process has been primarily used as an immiscible process. The benefit of injecting the gas under immiscible conditions is that it allows the injection gas to penetrate much deeper into the reservoir than what would occur during a miscible injection. This allows the injection gas to come into contact with, and thus dissolve into more formation oil. In successive cycles of the huff 'n' puff process the CO_2 continues to penetrate further into the reservoir and contacting more and more formation oil (Khatib et al., 1981).

2.3.5 Extraction of Lighter Components by CO_2

In the huff 'n' puff process, the injection gas can strip away some intermediaries from the formation oil, and produce an enriched gas mixture to produce some of these intermediate components from the reservoir. Liu et. al (2005) have noted that these intermediaries can go as high as C_7 when using CO_2 as the injection gas. They also noted that the recovery of these components extracted by the injection gas can account for up to 20% of the hydrocarbon recovery by mole fraction. This mechanism is modelled through the phase effects of injection gas coming into contact with formation oil.

2.4 Previous Studies

Although the first field implementations of the solvent-based huff 'n' puff process were recorded in the 1960's (Palmer et al., 1986), the first laboratory study was conducted by Sayegh and Maini (1984). Their study, along with other early studies, was aimed to understand the process and what parameters affect the EOR of the process. The majority of studies have been conducted using CO_2, with some examining the effect of using different solvents. Overall, the parameters which have shown to have the greatest influence on the process are: injection pressure (Firouz and Torabi, 2012; Wang et al., 2013), injection rate (Karim et. al, 1992), injection volume (Monger and Coma,

1988; Hsu and Brugman, 1986), number of cycles (Wang et al., 2013; Hsu and Brugman, 1986), soaking time (Monger and Coma, 1988), and type of solvent used (Qazvini Firouz and Torabi, 2012; Sayegh et al., 1984).

2.4.1 Injection Pressure

The solvent based huff 'n' puff process is typically used as an immiscible injection process. Studies have examined this process over various ranges of miscibility, and have shown that in general immiscible injection provides better EOR than miscible injection in light oil (Monger and Coma, 1988; Monger et al., 1991). In these studies core floods were completed to examine the difference between injection of CO_2 under miscible and immiscible conditions, and it was determined that injecting under near miscible conditions produced the best results. In studies where all trials were done under immiscible conditions, an increased injection pressure provided better oil recovery for both heavy oil (Firouz and Torabi, 2012) and light oil (Wang et al., 2013) under laboratory conditions. CO_2 mixing with formation oil is necessary for improved oil recovery, and the solubility of CO_2 in oil is a function of pressure as described by equation (2.1). The higher pressure allows more solvent to dissolve in the formation oil, which improves oil recovery as noted by Asghari and Torabi (2007) where they ran a huff 'n' puff experiment injecting CO_2 in a slim tube filled with normal decane at different operating pressures. It was shown in their slim tube experiment that higher pressure (above the MMP) provided the best recovery factor, but even when operating below the MMP an increase in pressure improved the recovery factor. When increasing the operating pressure from 250 psi to 750 psi (both below the MMP) they saw an improvement of 14% in the recovery factor.

2.4.2 Injection Rate

One of the mechanisms which enhances oil recovery of the solvent-based huff 'n' puff process is gas penetration, where a higher injection rate would lead to higher gas penetration. Karim et. al (1992) studied the effect of injection rate on the huff 'n' puff process. This study was completed on 6 ft long, 2 in diameter cores of consolidated Berea sandstone. It was determined that an optimal injection rate was 140 cc/h. Injection rates which were higher and lower than this number were tested, but 140 cc/h yielded the best results. This study showed that lower injection rates caused the solvent to stay close to the injection site which negatively affected EOR, but when injection rates reached levels which were too high they negatively affected gas utilization. Injection rate was also studied by Wang et al. (2013). This study showed similar results but the results were not as measurable, which may have been attributed to the study being completed on a low permeability reservoir.

2.4.3 Injection Volume

An obvious parameter affecting solvent based huff 'n' puff process is the injection volume. The larger the volume of solvent injected, the more solvent which will be in contact with formation oil to promote EOR. This has been shown experimentally (Monger and Coma, 1988), as well as in a pure simulation study (Hsu and Brugman, 1986). In the simulation study by Hsu and Brugman it was shown that injection volume is the most important parameter affecting the increased oil recovery. Although an increase in injection volume positively affects oil recovery, it negatively affects gas utilization therefore needs to be optimized depending on the economics of a project.

2.4.4 Number of Cycles

The optimal number of cycles to be used for a solvent-based process can be difficult to determine. It has been shown that in general the incremental increased oil recovery (additional oil recovery

per cycle) drops after each cycle, through experimental studies (Wang et al., 2013) as well as simulation studies (Hsu and Brugman, 1986). However, it has been noted in another project that the peak oil production was after the 2nd and 3rd cycles (Qazvini Firouz and Torabi, 2012). The optimal number of cycles depends on the individual field, as well as economics of using the solvent-based huff 'n' puff process.

2.4.5 Soaking Time

Similarly to the number of cycles, the optimal soaking time can be difficult to determine. There have been some disagreements found in different studies. Sayegh and Maini (1984) found that increasing the soaking time did not significantly improve oil recovery, where Monger and Coma (1988) found that runs with a soak period produced more oil than runs without a soak period. In terms of increasing soaking time, it has been shown that differences in soak times do not have a significant change on oil recovery. Experimentally, (Firouz and Torabi, 2012) when changing the soak time from 24 to 48 hours, it was shown that it did not significantly improve the overall recovery factor. Through simulation (Hsu and Brugman, 1986), it was shown that increasing the soak time from 5 to 40 days did not have a significant increase on oil recovery. In the field a soak period is typically used when employing the solvent-based huff 'n' puff process. A study on Texas projects showed that a soak period of 2 to 3 weeks could produce as much oil as longer soak periods (Haskin and Alston, 1989), and a study on projects in Louisiana and Kentucky showed that the optimal soak period was 1 month (Thomas and Monger-McClure, 1991). The optimal soak period depends on field, as well as the economics of using the solvent-based huff 'n' puff process.

2.4.6 Solvents

Although CO_2 is the most popular solvent used in the solvent-based huff 'n' puff process, other solvents have been tested with varying results. In the early stages of the solvent-based huff 'n' puff

process organic solvents were also tested for heavy oil stimulation, but these lacked cost effectiveness. This is due to their inability to propagate deep into the reservoir (Patton et al., 1982). In the 1969 US patent, Keith submitted various EOR methods which were used at the time, one of which being an inert gas huff 'n' puff. The inert gas EOR method used a gas composition of typically 11%-15% CO_2 and 89%-85% N_2. Keith proposed that using pure CO_2 would provide better EOR than the inert gas.

In studies in more recent years it has been shown that indeed CO_2 produces better results than N_2 for heavy oils (Qazvini Firouz and Torabi, 2012; Sayegh et al., 1984), which is what the process was originally intended for. Liu et al. (2005) showed that CO_2 causes more swelling than N_2, as well as a greater decrease in viscosity of the oil, which are two of the main mechanisms that contribute to the EOR of the huff 'n' puff process. This is due to the higher solubility of CO_2 in the oil.

Another solvent which has been studied for use in the solvent based huff 'n' puff process is natural gas, although it has not been studied as extensively as CO_2 and N_2. A study on heavy oil (Firouz and Torabi, 2012) compared using pure methane against CO_2, as well as other hydrocarbons with CO_2 mixtures. This study concluded that CO_2 provides greater EOR than pure methane, but some mixtures of CO_2 and hydrocarbons can produce similar results to using pure CO_2. Shayegi et al. (1996) studied light oil comparing the use of pure methane and N_2 against CO_2, as well as mixtures of CO_2 with methane. This study determined that CO_2 and methane produce roughly the same recovery factors, N_2 only recovered half the oil that was recovered using CO_2 or methane.

There have also been a few studies examining the use of only natural gas for the solvent based huff 'n' puff process. Haines and Monger (1990) completed a study which focused solely on natural

gas for the solvent-based huff ‘n’ puff process. This study used natural gas as a solvent in the huff ‘n’ puff process in a light oil reservoir after waterflooding. Through coreflooding experiments and a field scale model this study showed that natural gas can provide favourable EOR in light oil fields. The natural gas was injected under immiscible conditions similar to the CO_2 process. The operational parameters affected the process in the same way as the CO_2 process, with the injection volume being an important parameter affecting the incremental oil recovery. Many of the same recovery mechanisms such as oil swelling and oil viscosity reduction were noted to have an effect on oil recovery during the natural gas process, which is similar to what has been seen in the CO_2 process. The natural gas huff ‘n’ puff process was tested in the field in Brazil (Lino, 1994). The purpose of this study was to substitute the CO_2 huff ‘n’ puff process with natural gas, to make the process applicable to a larger number of projects. The process was tested on various wells with different injection volumes and different soak times. The results showed that most wells had a positive incremental oil recovery while some had a negative incremental oil recovery, with the overall conclusion being that cyclic natural gas injection is a promising method to replace CO_2 injection where it is not feasible due to the expensive costs of using CO_2 in certain scenarios, such as operations offshore.

The previous studies mentioned using natural gas for the solvent based huff ‘n’ puff process applied to light oils. Studies have also been completed on heavy oils, which are what the process was originally intended for. A study by Wenlong et al. (2008) completed a laboratory experiment to determine how natural gas can dissolve in the heavy oil to decrease oil viscosity and increase oil flow. The use of the huff ‘n’ puff process in this paper contributed to foamy oil flows which enhanced oil recovery from a single well through similar mechanisms discussed previously such as reduction in viscosity and oil swelling. Another study examined the use of the natural gas huff

'n' puff process to maintain foamy oil production in a heavy oil reservoir (Sun and Zhang, 2014). This study also showed that natural gas cyclic injection improved oil recovery by creating foamy oil.

Chapter 3 Methodology

This section provides the methodology used to create a numerical reservoir simulation model. A system of equations was developed to describe the reservoir behaviour, and these equations were implemented in MATLAB to create a one-dimensional compositional reservoir simulator. The work flow for the development of this model is shown in Figure 3.1, there are three main pieces to this model: a numerical set of equations to model fluid flow in the reservoir, fluid property models, and well models. The numerical set of equations to model compositional fluid flow in the reservoir employs finite difference approximations. Finite difference approximations are commonly used in reservoir engineering to approximate non-linear equations. By solving for the pressure in each grid block across the reservoir model implicitly, the rest of the reservoir parameters can be updated explicitly in what is known as an implicit pressure explicit composition and saturation (IMPECS) method. This method, as well as the detailed solution method of the compositional simulator is described in detail in section 3.3, which also provides flow charts for the simulator in Figure 3.7 and Figure 3.8. The fluid property models encompass how the properties of the fluids in the reservoir change due to changes in the reservoir. These fluid property models are described in detail in section 3.4. The well models in a one dimensional radial model can be thought of as the boundary conditions for the model, boundary conditions are required in numerical simulations to model the boundary of the reservoir being simulated. These can either be modelled as real wells (such as at the injection/ production point) or virtual wells between the area being simulated and the rest of the reservoir. The well models are described in section 3.3.4. With all these pieces together to form the compositional numerical reservoir simulator, the model was then validated in section 3.5.

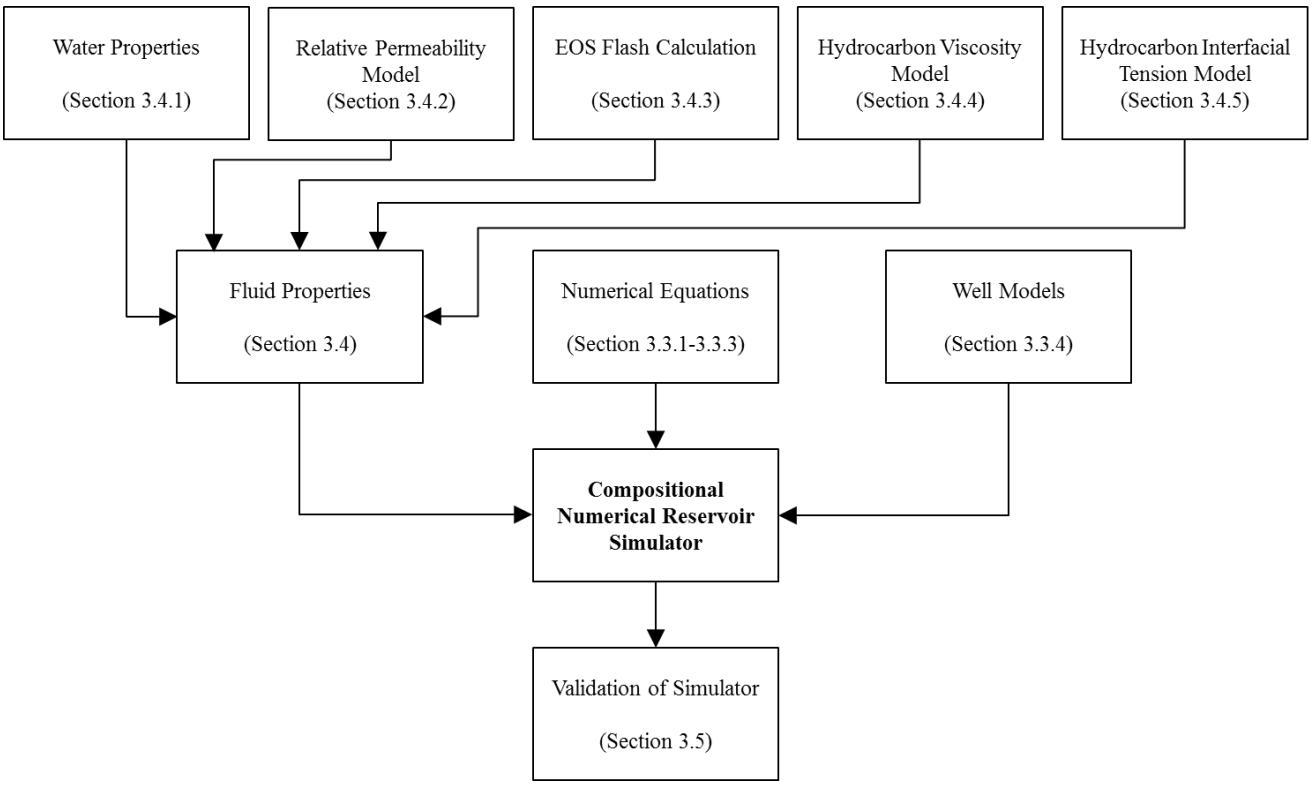


Figure 3.1 – Work Flow of Model Development

3.1 Compositional Modelling Equations

The simpler form of reservoir modelling is referred to as black oil modelling. In black oil modelling there are only three components, water and the two hydrocarbon components of oil and gas. Black oil models only have two hydrocarbon components, oil and gas, therefore mass transfer only occurs between the oil and gas phases. In compositional modelling the hydrocarbons are split into multiple components, and these components transfer mass between the oil and gas phase. Therefore, the compositional model is based on the conservation of mass of each component. Compositional modelling is typically used for gas injection processes, or any process where it is thought that inter-phase mass transfer may affect the reservoir modelling.

The equations for the conservation of mass of the water and hydrocarbon components are listed in equations (3.1) and (3.2) respectively (Kazemi et al., 1978):

$$\frac{\partial(\phi\xi_w S_w)}{\partial t} + \frac{\partial(\xi_w u_w)}{\partial x} = q_w \ , \tag{3.1}$$

$$\frac{\partial(\phi z_i[x_{io}\xi_o S_o + x_{ig}\xi_g S_g])}{\partial t} + \frac{\partial(x_{io}\xi_o u_o + x_{ig}\xi_g u_g)}{\partial x} = x_{io}q_o + x_{ig}q_g \tag{3.2}$$

$$i = 1, 2, ..., Nc \ ,$$

where ξ_α is molar density, S_α is α -phase saturation, $x_{i\alpha}$ is mole fraction of component i in phase c, q_α is molar flow rate and u_α is volumetric flux represented in one dimension by Darcy's law in equation (3.3):

$$u_\alpha = -\frac{kk_{r\alpha}}{\mu_\alpha}\left(\frac{\partial p_\alpha}{\partial x}\right) \qquad \alpha = w, o, g \ , \tag{3.3}$$

where k is rock permeability, $k_{r\alpha}$ is relative permeability, μ_α is viscosity, and p_α is pressure. Due to the fact that fluid flow is assumed to be slow relative to the inter-phase thermodynamic change, the reservoir is assumed to be in equilibrium at all times (Chen et al., 2006). Equilibrium relations are listed in equations (3.4)-(3.8).

$$f_{io} = f_{ig} \ , \tag{3.4}$$

$$L = \frac{\xi_o S_o}{\xi_o S_o + \xi_g S_g} \ , \tag{3.5}$$

$$V = \frac{\xi_g S_g}{\xi_o S_o + \xi_g S_g} \ , \tag{3.6}$$

$$z_i = x_{io}L + x_{ig}V \ , \tag{3.7}$$

$$K_i = \frac{x_{ig}}{x_{io}} \ , \tag{3.8}$$

where f is the fugacity, L is the liquid mole fraction of the hydrocarbons, V is the vapor mole fraction of the hydrocarbons, z_i is the total mole fraction of component i, and K_i is the equilibrium ratio of vapor to liquid in component i. Equation (3.4) is the fugacity relationship, that shows that it is assumed that each component is at equilibrium in both the oil and gas phases. Equation (3.5) and equation (3.6) respectively are used to determine the liquid and vapor mole fraction based on phase saturations and molar densities. Equation (3.7) is used to determine the total mole fraction of a component from its liquid and vapor parts. The constraint equations are as follows:

$$\sum_{i=1}^{Nc} z_i = \sum_{i=1}^{Nc} x_{io} = \sum_{i=1}^{Nc} x_{ig} = 1 \ , \tag{3.9}$$

$$L + V = 1 \ , \tag{3.10}$$

$$S_w + S_o + S_g = 1 \ , \tag{3.11}$$

$$p_w = p_o - P_{cow} \ , \tag{3.12}$$

$$p_g = p_o + P_{cog} \ , \tag{3.13}$$

where P_{cow} and P_{cog} are oil-water and gas-oil capillary pressures respectively. These constraint equations are used for determining water and gas pressures from the oil pressure, equation (3.12) and equation (3.13), and also to constrain that the summation of all mole fractions, phases, and saturations is equal to unity. Combining the fluid flow equations with the equilibrium relations and the constraint equations provides a system of equations which can be used to compositionally model a reservoir.

3.2 Numerical Reservoir Modelling

Numerical reservoir simulation typically employs a finite-difference approach to solve the differential equations involved in the mass transfer and fluid flow. This allows the reservoir to be

divided into grid blocks for simulation, which is known as discretization in space. The general method of discretization in space has been is shown in Figure 3.2 (Aziz and Settari, 1979).

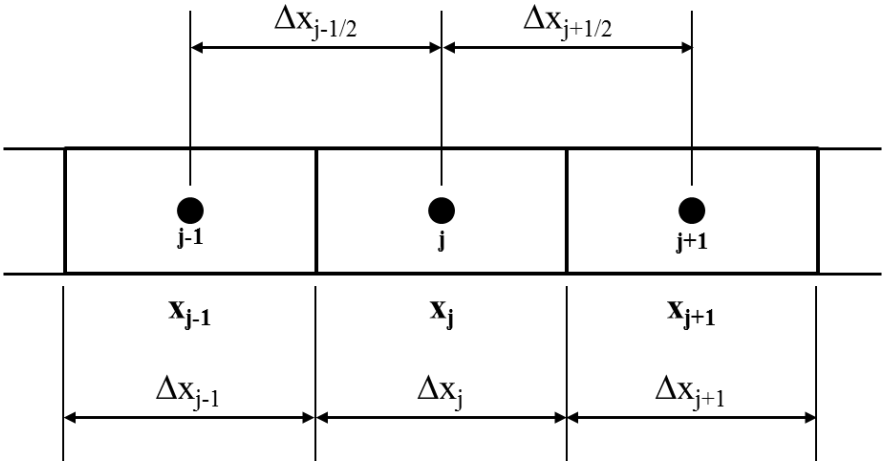


Figure 3.2 – General Discretization in Space

Two common methods of space discretization are the point-distributed grid (PDG) and the block-centered grid (BCG) approach, which are illustrated in Figure 3.3 and Figure 3.4 respectively (Aziz and Settari, 1979):

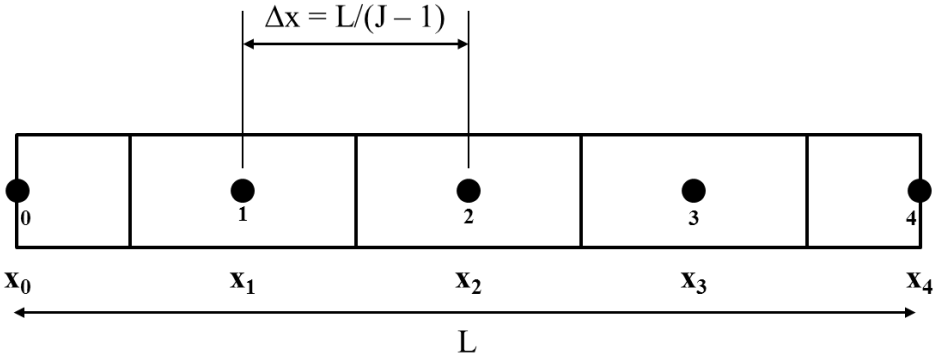


Figure 3.3 – Point-Distributed Grid

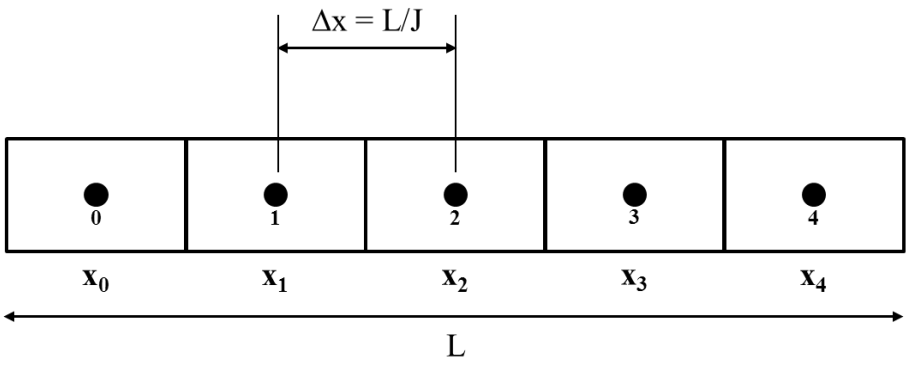


Figure 3.4 – Block-Centered Grid

In this figure L is the length of the model, and J is the total number of grid blocks. These illustrations show the discretization of a uniform grid, but either method can be extended to an irregular grid. In both methods the properties of the reservoir block are assumed to be acting in the center of the block, but for the PDG method the boundary blocks are only half as long compared to the BCG method. This allows the properties to be acting directly at the boundary when using the PDG method. The model in this thesis requires the use of an irregular grid (radial), for which the PDG method is more suited (Aziz and Settari, 1979).

The compositional numerical simulator was programmed to be able to use each of these types of discretization. There are two common co-ordinate systems used in reservoir simulation, Cartesian and radial. Since the huff 'n' puff is a single well model, this paper focuses on the development of the equations in radial geometry, although the simulator was programmed to also use Cartesian geometry for some validation work. Figure 3.5 and Figure 3.6 show a visual representation of each co-ordinate system in one dimension, respectively.

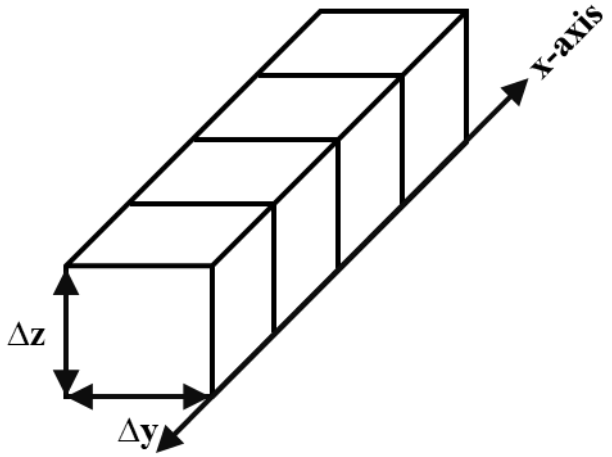


Figure 3.5 – Cartesian Geometry (1-D)

When using Cartesian geometry, Δx is the length of each grid block, Δy is the height and Δz is the depth. The length of each grid block can be spaced uniformly or irregularly.

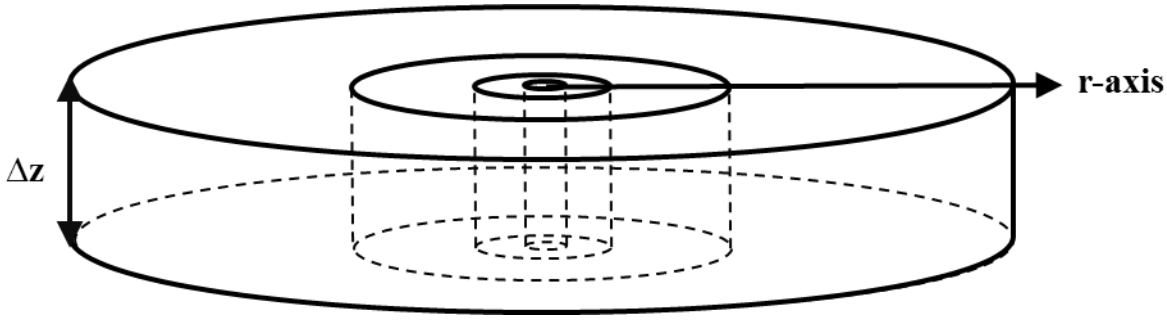


Figure 3.6 – Radial Geometry (1-D)

In order to create a radial grid, the grid blocks should be spaced logarithmically (Aziz and Settari, 1979). The grid block center radius r_{j+1} and grid block interface radius $r_{j\pm1/2}$ are calculated from equations (3.14) and (3.15) respectively, and Δz is the grid block depth.

$$r_{j+1} = r_j \left(\frac{r_e}{r_w} \right)^{1/(J-1)} \qquad j = 1, 2, ..., J \; ; \; r_1 = r_w , \tag{3.14}$$

$$r_{j+1/2} = \frac{r_{j+1} - r_j}{\ln(r_{j+1} / r_j)} \quad , \tag{3.15}$$

where r_e is the drainage radius, r_w is the wellbore radius, and J is the total number of grid blocks.

3.3 Numerical Compositional Model

This section provides the numerical solution to a one-dimensional radial compositional model. The model was developed for Cartesian co-ordinates as well for testing purposes, therefore the differences needed to convert the radial model to a Cartesian model are also listed. The simulation process requires the user to set an initial reservoir pressure, composition, temperature, and boundary conditions. The model uses an IMPECS method, which means at each time step in each grid block the pressure is calculated implicitly, and the concentrations and saturations are then updated explicitly. As discussed in section 3.3.1 the formulation of the pressure equation is based off a method developed by Nghiem et al. (1981). Through a flash calculation (described in detail in section 3.4.3) the liquid vapor split of each component in each grid block is determined, and with this information, viscosity, saturation, relative permeability, capillary pressure and transmissibility are computed in that order. The pressure equation is updated with the new parameters, and through a Newton-Raphson iteration this process is repeated until the new pressure across the system has been found. The summary of the overall solution process is shown in Figure

3.7 and the flow chart for the iteration process is shown in Figure 3.8. Appendix B has fully discretized versions of the necessary equations.

An initial reservoir pressure and oil composition are provided to the simulator, then all initial reservoir properties are calculated from there. At each time step Δp_o is solved using the Newton-Raphson iterative process, which then provides the new oil pressure. With this oil pressure reservoir properties are updated for each time step.

The iteration process for solving pressure at each time step begins with a guessed value of p_o^{n+1} (first guess is p_o^n), then the iterative process for solving for Δp_o takes place as shown in Figure 3.8. At the end of each iteration the pressure condition is checked to see if convergence has been achieved. If convergence has not been achieved, the iteration runs again until either the solution has converged or the maximum number of iterations has been reached

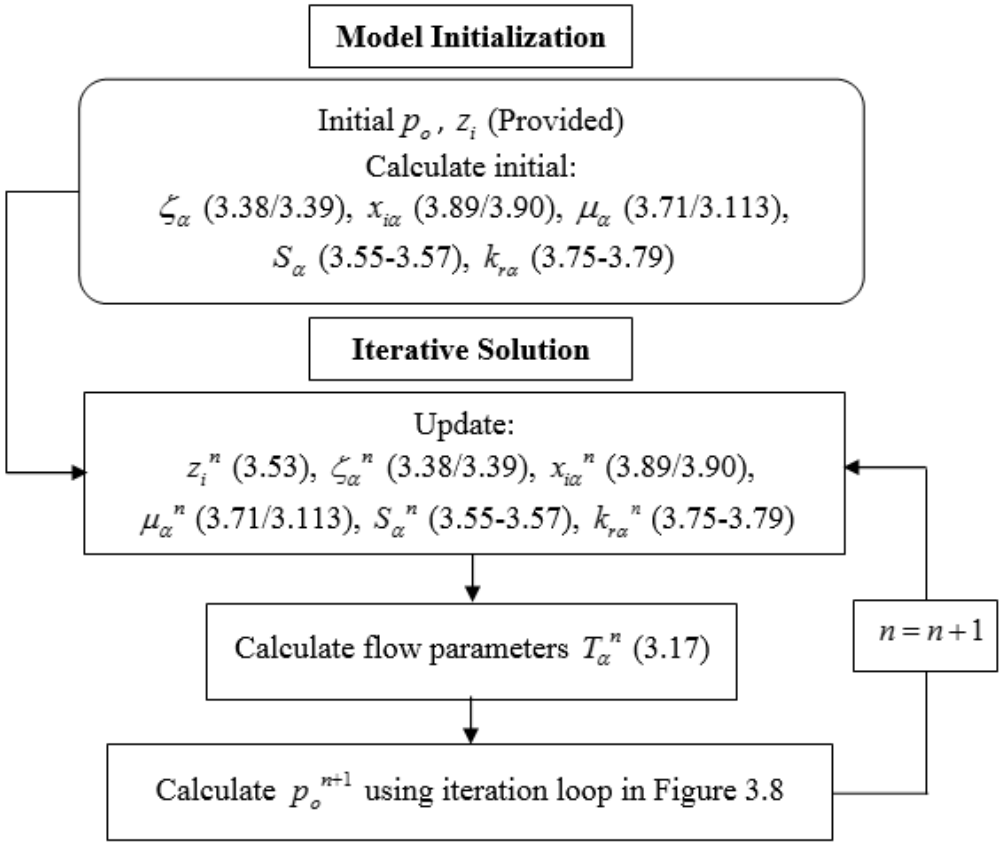


Figure 3.7 – Overall Solution Process Flow Chart

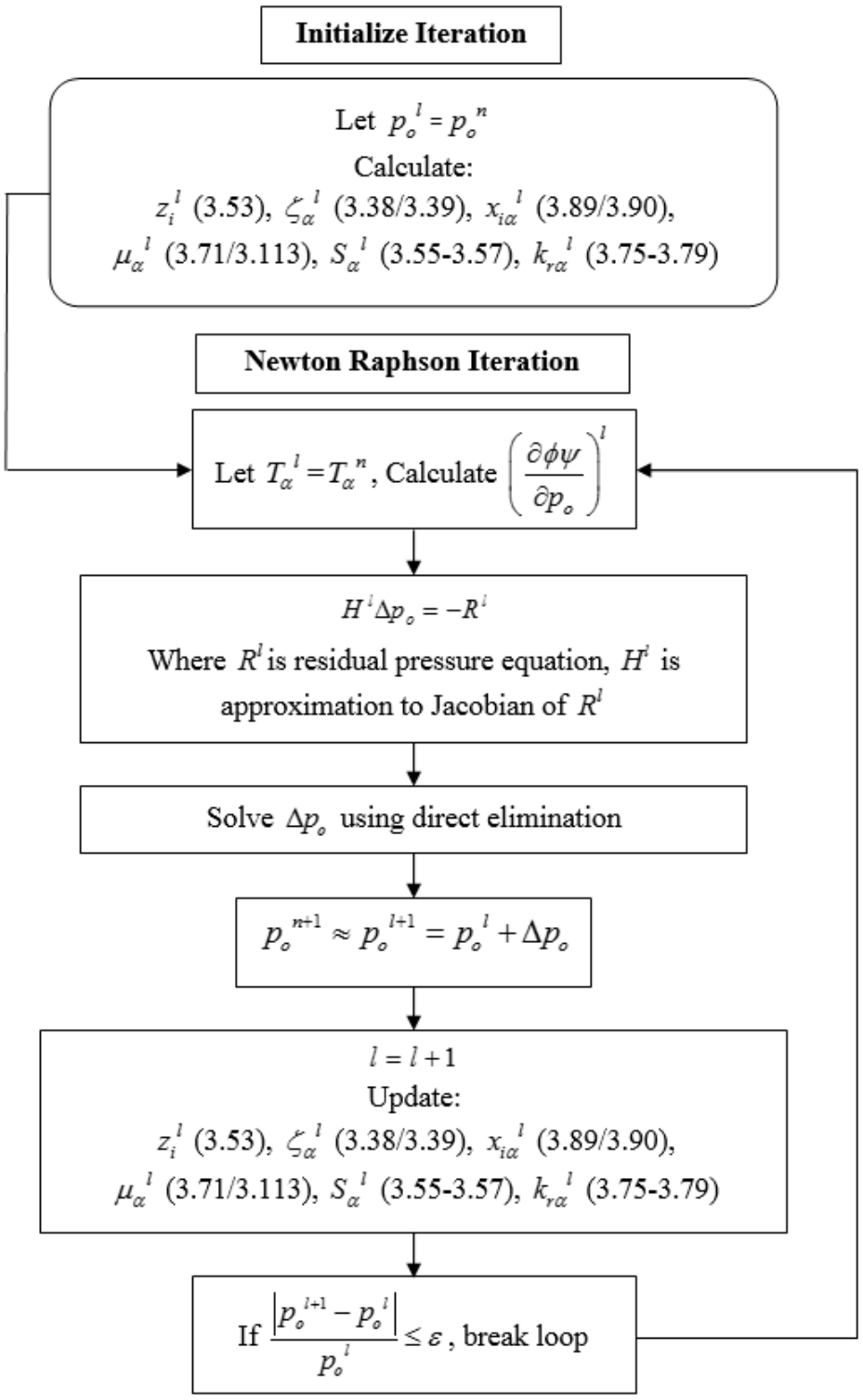


Figure 3.8 – Iteration Process Flow Chart

The model begins with the conservation of mass equation. Summing equation (3.2) over all components and applying the equilibrium relations gives:

$$\frac{\partial(\phi[\xi_o S_o + \xi_g S_g])}{\partial t} + \frac{\partial(\xi_o u_o + \xi_g u_g)}{\partial x} = q_o + q_g \ . \tag{3.16}$$

Phase transmissibility is defined as follows:

$$T_\alpha = \frac{k k_{r\alpha} \xi_\alpha}{\mu_\alpha} \ , \tag{3.17}$$

where the relative permeability of each phase is calculated using the Corey model (see section 3.4.2) or tabulated data combined with the standard Eclipse™ model (Schlumberger, 2014) as described in section 3.4.2. Applying the definition of transmissibility, and using a radial co-ordinate system equations (3.1) and (3.16) yield:

$$\frac{\partial(\phi \xi_w S_w)}{\partial t} - \frac{1}{r}\frac{\partial}{\partial r}\left[r T_w \left(\frac{\partial p_w}{\partial r} \right) \right] = q_w \tag{3.18}$$

and

$$\frac{\partial(\phi[\xi_o S_o + \xi_g S_g])}{\partial t} - \frac{1}{r}\frac{\partial}{\partial r}\left[r T_o \left(\frac{\partial p_o}{\partial r} \right) + r T_g \left(\frac{\partial p_g}{\partial r} \right) \right] = q_o + q_g \ , \tag{3.19}$$

which describe the water and oil/gas material balance, respectively. Equations (3.18) and (3.19) are added together to form the pressure equation developed in section 3.3.1, which can then be solved implicitly for pressure in the system if the phase transmissibilities and viscosities are evaluated explicitly at the previous time step as done in an implicit pressure explicit saturation (IMPES) or IMPECS formulation. In order to use a Cartesian co-ordinate system, any places in the model that use $\frac{1}{r}\frac{\partial}{\partial r}\left[r T_\alpha \left(\frac{\partial p_\alpha}{\partial r} \right) ... \right]$ should be replaced by $\frac{\partial}{\partial x}\left[T_\alpha \left(\frac{\partial p_\alpha}{\partial x} \right) ... \right]$.

3.3.1 Formulation of the Pressure Equation

There have been a few methods suggested in literature to solve the compositional pressure equation. The method used here is an alteration of the method suggested by Nghiem et al. (1981) in which they reviewed a method suggested by Kazemi et al. (1978), and made some variations in order to improve numerical stability. Their method allows for an iterative process to be applied to the pressure matrix which can be solved through direct elimination.

Applying equations (3.12) and (3.13) and multiplying the conservation of water equation by a constant parameter θ and adding it to the conservation of hydrocarbons yields:

$$\frac{\partial(\phi\psi)}{\partial t} - \frac{1}{r}\frac{\partial}{\partial r}\left[\theta r T_w\left(\frac{\partial p_o}{\partial r} - \frac{\partial P_{cow}}{\partial r}\right) + rT_o\left(\frac{\partial p_o}{\partial r}\right) + rT_g\left(\frac{\partial p_o}{\partial r} + \frac{\partial P_{cog}}{\partial r}\right)\right] = \theta q_w + q_o + q_g \tag{3.20}$$

where

$$\psi = \theta\xi_w S_w + \xi_o S_o + \xi_g S_g \ . \tag{3.21}$$

The numerical modelling of the well terms is described in section 3.3.3. Equation (3.20) is now the pressure equation which can be solved for p_o. The water equation is multiplied by the scaling factor θ in order to convert moles of water to moles of hydrocarbon. This scaling factor is calculated from:

$$\theta = \left.\frac{\xi_o S_o + \xi_g S_g}{\xi_w (S_o + S_g)}\right|_{t=0} \ . \tag{3.22}$$

Using this scaling factor helps in convergence (Nghiem et al., 1981). This scaling factor can be evaluated at any time and is not updated throughout the solution model, in this model it is evaluated at $t = 0$, and kept fixed thereafter.

3.3.2 Implicit Solution to the Pressure Equation

The pressure equation can be solved for p_o by discretization. First, the discretization in time of the accumulation term $\phi\psi$ yields:

$$\frac{\partial(\phi\psi)}{\partial t} = \frac{1}{\Delta t}\left(\phi^{n+1}\psi^{n+1} - \phi^{n}\psi^{n}\right) \tag{3.23}$$

where Δt is the time step, and superscript n refers to the interval of time. The time step is chosen through the Courant-Friedrichs-Lewy condition (CFL), which states that

$$\Delta t < \frac{\Delta x}{u} \ . \tag{3.24}$$

where u is total volumetric flux. The time step must be less than the length interval divided by the magnitude of the velocity (Courant et al., 1967). It is important to note that for a radial model the time step must be chosen for the smallest control volume in order to ensure convergence. As the reservoir blocks get closer to the wellbore the blocks get progressively smaller, therefore the time step will typically be smaller than when using Cartesian co-ordinates. The next step is to discretize in space using either the BCG or the PDG described in section 3.2. The subscript j refers to the center of the grid block, and the subscript $j \pm 1/2$ refers to the interface between grid blocks. The discretization of oil pressure in space is taken from the method for discretization of a cylindrical radial grid by Aziz and Settari (1979). Combining this with the discretization in time of the accumulation term provides the fully discretized pressure equation:

$$\begin{aligned}
&-\left\{\begin{array}{l}
\theta\left[T_{w_{j-1/2}}{}^{n}\left(p_{o_{j-1}}{}^{n+1} - P_{cow_{j-1}}{}^{n}\right) - (T_{w_{j+1/2}} + T_{w_{j-1/2}})^{n}\left(p_{o_j}{}^{n+1} - P_{cow_j}{}^{n}\right) + T_{w_{j+1/2}}{}^{n}\left(p_{o_{j+1}}{}^{n+1} - P_{cow_{j+1}}{}^{n}\right)\right] \\
+\left[T_{o_{j-1/2}}{}^{n} p_{o_{j-1}}{}^{n+1} - (T_{o_{j+1/2}} + T_{o_{j-1/2}})^{n} p_{o_j}{}^{n+1} + T_{o_{j+1/2}}{}^{n} p_{o_{j+1}}{}^{n+1}\right] \\
+\left[T_{g_{j-1/2}}{}^{n}\left(p_{o_{j-1}}{}^{n+1} + P_{cog_{j-1}}{}^{n}\right) - (T_{g_{j+1/2}} + T_{g_{j-1/2}})^{n}\left(p_{o_j}{}^{n+1} + P_{cog_j}{}^{n}\right) + T_{g_{j+1/2}}{}^{n}\left(p_{o_{j+1}}{}^{n+1} + P_{cog_{j+1}}{}^{n}\right)\right]
\end{array}\right\} \\
&+\frac{V_{r_j}}{\Delta t}\left(\phi_j{}^{n+1}\psi_j{}^{n+1} - \phi_j{}^{n}\psi_j{}^{n}\right) = \theta q_{w_j}{}^{n} + q_{o_j}{}^{n} + q_{g_j}{}^{n}
\end{aligned} \tag{3.25}$$

where V_r is the grid block volume and for a radial co-ordinate system:

$$T_{\alpha_{j+1/2}} = \frac{2\pi r_{j+1/2}\Delta z}{r_{j+1} - r_j} T_{\alpha_j} \text{ , and} \tag{3.26}$$

$$T_{\alpha_{j-1/2}} = \frac{2\pi r_{j-1/2}\Delta z}{r_j - r_{j-1}} T_{\alpha_{j-1}} \text{ .} \tag{3.27}$$

For a Cartesian co-ordinate system:

$$T_{\alpha_{j+1/2}} = \frac{2}{\frac{x_{j+1/2} - x_{j-1/2}}{\Delta y \Delta z} + \frac{x_{j+1\,1/2} - x_{j+1/2}}{\Delta y \Delta z}} T_{\alpha_j} \text{ , and} \tag{3.28}$$

$$T_{\alpha_{j-1/2}} = \frac{2}{\frac{x_{j+1/2} - x_{j-1/2}}{\Delta y \Delta z} + \frac{x_{j-1/2} - x_{j-1\,1/2}}{\Delta y \Delta z}} T_{\alpha_{j-1}} \text{ .} \tag{3.29}$$

As is typical with an IMPES solution method, the transmissibilities are evaluated at the previous time step n, and with a small enough time step, the capillary pressures can also be evaluated at the previous time step n. The oil pressure must be evaluated at time-step $n+1$. The following parameters are used to simplify the system:

$$D = \theta T_{w_{j+1/2}} + T_{o_{j+1/2}} + T_{g_{j+1/2}} \text{ ,} \tag{3.30}$$

$$E = \theta T_{w_{j-1/2}} + T_{o_{j-1/2}} + T_{g_{j-1/2}} \text{ ,} \tag{3.31}$$

$$F = -D - E \text{ ,} \tag{3.32}$$

where D, E, and F are parameters used to make the system of equations more readable.

Applying these simplifications to equation (3.25) gives what is known as the residual pressure equation which is:

$$R(p_{o_j}{}^{n+1}) = E_j p_{o_{j-1}}{}^{n+1} + F_j p_{o_j}{}^{n+1} + D_j p_{o_{j+1}}{}^{n+1} - \left(E_j P_{cow_{j-1}}{}^{n} + F_j P_{cow_j}{}^{n} + D_j P_{cow_{j+1}}{}^{n}\right) \tag{3.33}$$

$$+\left(E_j P_{cog_{j-1}}{}^{n} + F_j P_{cog_j}{}^{n} + D_j P_{cog_{j+1}}{}^{n}\right) + \theta q_{w_j} + q_{o_j} + q_{g_j} - \frac{V_{r_j}}{\Delta t}\left[\left(\phi\psi\right)_j{}^{n+1} - \left(\phi\psi\right)_j{}^{n}\right] \text{ .}$$

where R is the residual of the pressure equation. In order to solve for the values at the $n+1$ time-step, an iterative process must be employed. This system is non-linear in the primary variables therefore, the Newton-Raphson method is used for iteration to linearize the variables. Letting l represent the iteration level in the Newton-Raphson iteration, for a general variable υ:

$$\upsilon^{n+1} \approx \upsilon^{l+1} = \upsilon^{l} + \Delta\upsilon \tag{3.34}$$

rearranging shows:

$$\Delta\upsilon = \upsilon^{l+1} - \upsilon^{l} \tag{3.35}$$

Applying this to oil pressure yields:

$$p_o{}^{n+1} \approx p_o{}^{l+1} = p_o{}^{l} + \Delta p_o \tag{3.36}$$

A Taylor expansion can be used on the non-linear accumulation term $(\phi\psi)^{l+1}$:

$$\phi\psi(p_o)^{n+1} \approx \phi\psi(p_o)^{l+1} = \phi\psi(p_o)^{l} + \left(\frac{\partial\phi\psi}{\partial p_o}\right)^{l} \Delta p_o \ , \tag{3.37}$$

where $\frac{\partial\psi}{\partial p_o}$ can be approximated as (Nghiem et al., 1981) :

$$\left(\frac{\partial\psi}{\partial p_o}\right)^{l} = \left(\theta S_w \frac{\partial\xi_w}{\partial p_o} + S_o \frac{\partial\xi_o}{\partial p_o} + S_g \frac{\partial\xi_g}{\partial p_o}\right)^{l} \ . \tag{3.38}$$

Molar densities and porosity are related to oil pressure through equations (3.39)-(3.41):

$$\phi = \phi^{*}[1 + c_p(p_o - p^{*})] \ , \tag{3.39}$$

$$\xi_w = \xi_w{}^{*}[1 + c_w(p_o - p_w{}^{*})] \ , \tag{3.40}$$

$$\xi_\alpha = \frac{p_o}{Z_\alpha RT} \qquad \alpha = o, g \ , \tag{3.41}$$

where the * superscript indicates a parameter taken at time zero, c_p is the rock compressibility, c_w is the water compressibility, Z_α is the phase compressibility factor, R is the universal gas constant, and T is temperature. Using equation (3.38), $\left(\frac{\partial \phi\psi}{\partial p_o}\right)^l$ can now be evaluated as:

$$\left(\frac{\partial \phi\psi}{\partial p_o}\right)^l = c_p \phi^* \psi^l + \phi^l \left(\theta S_w c_w \xi_w^* + S_o \frac{\partial \xi_o}{\partial p_o} + S_g \frac{\partial \xi_g}{\partial p_g}\right)^l \tag{3.42}$$

where

$$\frac{\partial \xi_\alpha}{\partial p_o} = \frac{1}{RTZ_\alpha}\left(1 - \frac{p_o}{Z_\alpha}\frac{\partial Z_\alpha}{\partial p_o}\right) \qquad \alpha = o, g \ . \tag{3.43}$$

The accumulation term $(\phi\psi)^{l+1}$ is now a function which is linear in Δp_o. The expanded form of $\left(\frac{\partial \phi\psi}{\partial p_o}\right)^l$ is as follows:

$$\left(\frac{\partial \phi\psi}{\partial p_o}\right)^l = c_p \phi^* \psi^l + \phi^l \left(\theta S_w c_w \xi_w^* + S_o \frac{1}{RTZ_o}\left(1 - \frac{p_o}{Z_o}\frac{\partial Z_o}{\partial p_o}\right) + S_g \frac{1}{RTZ_g}\left(1 - \frac{p_o}{Z_o}\frac{\partial Z_g}{\partial p_o}\right)\right)^l \tag{3.44}$$

.

Applying equation (3.36) to equation (3.33) creates a linear system which can be solved for Δp_o over each iteration. The linear system is shown in equation (3.45):

$$H^l \Delta p_o = -R^l , \tag{3.45}$$

where R^l is the residual function in equation (3.33) and H^l is an approximation to the Jacobian of R^l. The matrix H^l can be evaluated through the following equations:

$$H_{jj}{}^l = R_j{}^l + \theta\left(\frac{\partial q_w}{\partial p_o}\right)^l + \left(\frac{\partial q_o}{\partial p_o}\right)^l + \left(\frac{\partial q_g}{\partial p_o}\right)^l - \left(\frac{\partial \phi\psi}{\partial p_o}\right)^l , \tag{3.46}$$

$$H_{j(j+1)}{}^l = D_j{}^l , \tag{3.47}$$

$$H_{j(j-1)}{}^{l} = E_j{}^{l} \,. \tag{3.48}$$

The value of $\frac{\partial q_\alpha}{\partial p_o}$ for various types of wells is described in section 3.3.3. For simplification of the system the constant G is defined as:

$$G^l = F^l + \theta\left(\frac{\partial q_w}{\partial p_o}\right)^l + \left(\frac{\partial q_o}{\partial p_o}\right)^l + \left(\frac{\partial q_g}{\partial p_o}\right)^l - \left(\frac{\partial \phi\psi}{\partial p_o}\right)^l . \tag{3.49}$$

The model has J grid blocks, implementing equation (3.45) over these J grid blocks, creates the system of equations shown in equation (3.50):

$$\begin{aligned} G_1{}^{l}\Delta p_{o_1} + D_1{}^{l}\Delta p_{o_1} &= -R_1{}^{l} \\ E_2{}^{l}\Delta p_{o_1} + G_2{}^{l}\Delta p_{o_2} + D_2{}^{l}\Delta p_{o_3} &= -R_2{}^{l} \\ &\vdots \\ E_J{}^{l}\Delta p_{o_{J-1}} + G_J{}^{l}\Delta p_{o_J} &= -R_J{}^{l} \,. \end{aligned} \tag{3.50}$$

Equation (3.50) in matrix form yields:

$$\begin{pmatrix} G_1{}^{l} & D_1{}^{l} & & \\ & \vdots & & \\ E_j{}^{l} & G_j{}^{l} & D_j{}^{l} & \\ & \vdots & & \\ & E_J{}^{l} & G_J{}^{l} & \end{pmatrix} \begin{pmatrix} \\ \Delta p_o \\ \\ \end{pmatrix} = \begin{pmatrix} -R_1{}^{l} \\ \\ -R_j{}^{l} \\ \\ -R_J{}^{l} \end{pmatrix}. \tag{3.51}$$

and in expanded form :

$$\begin{pmatrix} G_1^{\,l} & D_1^{\,l} & & & & & \\ E_2^{\,l} & G_2^{\,l} & D_2^{\,l} & & & & \\ & \vdots & \vdots & \vdots & & & \\ & & E_j^{\,l} & G_j^{\,l} & D_j^{\,l} & & \\ & & & \vdots & \vdots & \vdots & \\ & & & & E_{J-1}^{\,l} & G_{J-1}^{\,l} & D_{J-1}^{\,l} \\ & & & & & E_J^{\,l} & G_J^{\,l} \end{pmatrix} \begin{pmatrix} \Delta p_{o_1} \\ \Delta p_{o_2} \\ \vdots \\ \Delta p_{o_j} \\ \vdots \\ \Delta p_{o_{J-1}} \\ \Delta p_{o_J} \end{pmatrix} = \begin{pmatrix} -R_1^{\,l} \\ -R_2^{\,l} \\ \vdots \\ -R_j^{\,l} \\ \vdots \\ -R_{J-1}^{\,l} \\ -R_J^{\,l} \end{pmatrix}. \tag{3.52}$$

This is a linear system with a tri-diagonal coefficient matrix. This system is solved using direct elimination. The pressure in each grid block is then updated as follows:

$$p_o^{\,l+1} = p_o^{\,l} + \Delta p_o \,. \tag{3.53}$$

The Newton-Raphson iteration process is continued until the convergence criteria shown in equation (3.54) is met.

$$\frac{\left| p_o^{\,l+1} - p_o^{\,l} \right|}{p_o^{\,l}} \le \varepsilon \,. \tag{3.54}$$

With $p_o^{\,n+1}$ computed, the pressure in each phase can be easily updated for each time step through the given capillary pressure relations.

3.3.3 Explicit Solution to Compositions and Saturations

Once the pressure has been implicitly solved, the compositions and saturations can then be updated explicitly for the next iteration. The total mole fraction of component i, z_i, is updated by discretizing equation (3.2) with respect to time, and applying the assumptions which yields:

$$z_i^{\,l+1} = \frac{\frac{1}{r}\frac{\partial}{\partial r}\left[r x_{io}^{\,n} T_o^{\,n} \left(\frac{\partial p_o}{\partial r} \right) + r x_{ig}^{\,n} T_g^{\,n} \left(\frac{\partial p_o}{\partial r} + \frac{\partial P_{cog}}{\partial r} \right) \right] + \frac{V_r}{\Delta t} \phi^n z_i^{\,n} (\xi_o S_o + \xi_g S_g)^n + q_i}{\frac{V_r}{\Delta t} \phi^{l+1} (\xi_o S_o + \xi_g S_g)^{l+1}} \,, \tag{3.55}$$

where the denominator can be approximated as seen in equation (3.56) by discretizing equation (3.19) with respect to time:

$$\frac{V_r}{\Delta t}\phi^{l+1}(\xi_o S_o + \xi_g S_g)^{l+1} = \frac{1}{r}\frac{\partial}{\partial r}\left[rT_o{}^n\left(\frac{\partial p_o}{\partial r}\right) + rT_g{}^n\left(\frac{\partial p_o}{\partial r} + \frac{\partial P_{cog}}{\partial r}\right)\right] + \frac{V_r}{\Delta t}\phi^n(\xi_o S_o + \xi_g S_g)^n + q_o + q_g \quad . \tag{3.56}$$

With updated mole fractions of each component in each grid block, an equation of state flash calculation can be employed in each grid block to update the equilibrium ratios, phase mole fractions of each component, overall mole fraction of each phase and molar densities. A detailed description of the flash calculation is provided in section 3.4.3. The viscosities are then updated using the Lohrenz-Bray-Clark method (Lohrenz et al., 1964) which is described in section 3.4.4.

The next step is to update the saturation of each phase. The water saturation is updated by discretizing equation (3.18) with respect to time. This yields:

$$S_w{}^{l+1} = \frac{\frac{1}{r}\frac{\partial}{\partial r}\left[rT_w{}^n\left(\frac{\partial p_o}{\partial r} - \frac{\partial P_{cow}}{\partial r}\right)\right] + \frac{V_r}{\Delta t}\phi^n \xi_w{}^n S_w{}^n + q_w}{\frac{V_r}{\Delta t}\phi^{l+1}\xi_w{}^{l+1}}. \tag{3.57}$$

By substituting equations (3.5) and (3.6) into equation (3.11) the equations for updating oil and gas saturation are obtained, as shown in equations (3.58) and (3.59) respectively:

$$S_o{}^{l+1} = \left[\frac{(1-S_w)L\xi_g}{L\xi_g + (1-L)\xi_o}\right]^{l+1}, \tag{3.58}$$

$$S_g{}^{l+1} = \left[\frac{(1-S_w)(1-L)\xi_o}{L\xi_g + (1-L)\xi_o}\right]^{l+1}. \tag{3.59}$$

3.3.4 Well Models

The molar flow rates for each phase can be calculated from equation (3.60), and the molar flow rate of individual components can be calculated from (3.61) (Nghiem et al., 1981):

$$q_\alpha = \xi_\alpha Q_\alpha \ , \tag{3.60}$$

$$q_i = x_{io}\xi_o Q_o + x_{ig}\xi_g Q_g \ , \tag{3.61}$$

where Q_α is the volumetric flow rate of phase α, and q_i is the flow rate of component i. The model can accept constant flow rate wells, or constant bottom-hole pressure wells. These wells can be injection or production wells, as is described in the proceeding sections.

3.3.4.1 Injection Wells

For constant flow rate wells, $Q_{\alpha_{inj}}$ is specified. For constant bottom-hole pressure wells it is calculated through equation (3.62) (Kazemi et al., 1978):

$$Q_{\alpha_{inj}} = I_{inj} M_\alpha \left(p_{bh_{inj}} - p_o^{\,n+1} \right) , \tag{3.62}$$

where I is a shape factor for the well, M_α is the mobility of the injection phase and p_{bh} is the bottom hole pressure of the well. For constant rate wells $\frac{\partial q_\alpha}{\partial p_o}$ is zero, and for constant bottom-hole pressure wells $\frac{\partial q_\alpha}{\partial p_o}$ is:

$$\frac{\partial q_{\alpha_{inj}}}{\partial p_o} = -I_{inj}\xi_\alpha \ , \tag{3.63}$$

3.3.4.2 Production Wells

For constant flow rate wells phase rate, $Q_{\alpha_{prod}}$, can be calculated from the total rate, Q_{prod} ,using equation (3.64):

$$Q_{\alpha_{prod}} = \frac{M_\alpha}{M_T} Q_{prod} \ , \tag{3.64}$$

where M_T is the total mobility of fluids in the grid block. Phase mobility is defined as:

$$M_\alpha = \frac{k_{r\alpha}}{\mu_\alpha} \ . \tag{3.65}$$

Equations (3.66) and (3.67) can be used to calculate the value of $Q_{\alpha_{prod}}$ and $\frac{\partial q_{\alpha_{prod}}}{\partial p_o}$. This is similar to equations (3.62) and (3.63), with the difference being that the bottom-hole pressure will be set below the grid block pressure.

$$Q_{\alpha_{prod}} = I_{prod} M_\alpha \left(p_{bh_{prod}} - p_o^{n+1} \right) , \tag{3.66}$$

$$\frac{\partial q_{\alpha_{prod}}}{\partial p_o} = -I_{prod} \xi_\alpha \ . \tag{3.67}$$

3.4 Fluid Properties

In compositional modelling, there are many fluid properties which are functions of pressure and require updating throughout the solution process. This section provides the detailed method used for updating each fluid property. Each of these fluid properties is updated at every iteration of the pressure equation.

3.4.1 Water Properties

The compressibility and viscosity of formation water are both functions of pressure, temperature, and salinity. The compressibility of water is calculated as follows using field units (Danesh, 1998):

$$c_{wf} = 10^{-6} \varpi (C_0 + C_1 T + C_2 T^2) \ , \tag{3.68}$$

where c_{wf} is the isothermal compressibility of water in psi^{-1}, T is the temperature in °F, ϖ is the salinity correction factor and:

$$C_0 = 3.8546 - 0.000134\,p \ , \tag{3.69}$$

$$C_1 = -0.01052 + (4.77\times10^{-7})\,p \ , \tag{3.70}$$

$$C_2 = 3.9267\times10^{-5} - (8.8\times10^{-10})\,p \ , \tag{3.71}$$

where p is in psi. The salinity correction factor is calculated through the following equation:

$$\varpi = 1 + (-0.052 + (2.7\times10^{-4})T - (1.14\times10^{-6})T^2 + (1.121\times10^{-9})T^3)w_s \ , \tag{3.72}$$

where w_s is the salinity of the formation water (as fraction). As previously mentioned, the viscosity (cP) of formation water is also a function of pressure, temperature and salinity. The water viscosity can be calculated as follows (Danesh, 1998):

$$\mu_w = \mu_{wT}\frac{\mu_w}{\mu_{wT}} \ , \tag{3.73}$$

where μ_{wT} is the water viscosity at atmospheric conditions. This is calculated through:

$$\mu_{wT} = (109.574 - 8.40564w_s + 0.313314{w_s}^2 + (8.72213\times10^{-3}){w_s}^3)T^{-D} \ , \tag{3.74}$$

where T is in °F and:

$$D = 1.12166 - (2.63951\times10^{-2})w_s + (6.79461\times10^{-4}){w_s}^2 + (5.47119\times10^{-5}){w_s}^3 - (1.55586\times10^{-6}){w_s}^4 \ . \tag{3.75}$$

To relate the viscosity of water from atmospheric conditions to reservoir conditions, the following relationship is used:

$$\frac{\mu_w}{\mu_{wT}} = 0.9994 + (4.0295\times10^{-5})\,p + (3.1062\times10^{-9})\,p^2 \ , \tag{3.76}$$

where p is in psi.

3.4.2 Relative Permeability Model

The three-phase relative permeability model used in the simulation is the default model used in Eclipse™. This model is simple yet effective, and avoids typical problems seen in more complex three-phase permeability models. The basis of the model is that it assumes that water and gas are

completely segregated in a grid block, save from the connate water seen in the gas phase. This model treats the reservoir as if water and gas only flow relative to oil, and not each other. This allows the relative permeabilities of water and gas, k_{rw} and k_{rg}, to be calculated through a typical two-phase relative permeability curve as a function of S_L (equal to $S_o + S_{wc}$) or through tabulated data. The relative permeability of oil is defined by:

$$k_{ro} = \frac{S_g k_{rog} + (S_w - S_{wc}) k_{row}}{S_g + S_w - S_{wc}} , \tag{3.77}$$

where k_{rog} is the relative permeability of oil to gas, k_{row} is the relative permeability of oil to water, and S_{wc} is the connate water saturation. It can be seen from this definition that when $S_g = 0$ the relative permeability of oil will only be equal to k_{row}, and when $S_w = S_{wc}$ the relative permeability of oil will be k_{rog}. The two-phase functions which can be used to complete the relative permeability model are described using the Corey model as follows:

$$k_{row} = a_{ow} \left[\frac{1 - S_w - S_{orw}}{1 - S_{wc} - S_{orw}} \right]^{n_{ow}} , \tag{3.78}$$

$$k_{rog} = a_{og} \left[\frac{(S_o - S_w) - S_{wc} - S_{org}}{1 - S_{wc} - S_{org} - S_{gc}} \right]^{n_{og}} , \tag{3.79}$$

$$k_{rw} = a_w \left[\frac{S_w - S_{wc}}{1 - S_w - S_{wc}} \right]^{n_w} , \tag{3.80}$$

$$k_{rg} = a_g \left[\frac{S_g - S_{gc}}{1 - S_{wc} - S_{org} - S_{gc}} \right]^{n_g} , \tag{3.81}$$

where a_{ow}, a_{og}, a_w, and a_g are constants which are the end point value for their respective relative permeability curves. S_{gc} is the critical gas saturation and n_w, n_{og}, n_{ow}, and n_g are the Corey model

relative permeability exponents. The Corey model was chosen for correlating relative permeabilities as it is very popular in reservoir simulation.

3.4.3 Equation of State Flash Calculations

For compositional modelling of hydrocarbon mixtures, equations of state (EOS) are typically used to describe the volumetric behaviour and describe fluid phase behaviour (Danesh, 1998). Usually, a two parameter cubic equation of state is used to describe hydrocarbon systems. The first equation of state of this type was developed by van der Waals in 1873, since then others have taken the van der Waals EOS and improved upon it. Two common EOSs used in the oil and gas industry are Suave-Redlich-Kwong (SRK) and Peng-Robinson (PR). The SRK EOS was developed first, with Peng and Robinson (1976) taking the SRK EOS and modifying the attraction parameter to improve liquid density prediction (Danesh, 1998). Only the PR EOS is used for modelling in this paper, as it is more commonly used in reservoir simulation.

The PR equation of state takes the following form (Peng and Robinson, 1976):

$$p = \frac{RT}{v-b} - \frac{a}{v(v+b)+b(v-b)}\ , \tag{3.82}$$

where p is pressure, R is the universal gas constant, T is the temperature, v is the molar volume, a is the attraction parameter and b is the co-volume parameter. This can be applied to pure components as well as mixtures. When dealing with mixtures the attraction parameter a and the co-volume parameter b are calculated through mixing rules. The method to perform flash calculations is an iterative process using a two parameter EOS such as the PR EOS, and has been described in detail by Danesh (1998). With a known hydrocarbon composition, fluid properties, temperature and pressure, the phase behaviour can be determined by first determining the co-volume and attraction parameters, b_i and a_i respectively, for each component:

$$b_i = \frac{0.077796 R T_{c_i}}{p_{c_i}}, \tag{3.83}$$

$$a_i = ac_i (1 + m_i (1 - T_{r_i}^{0.5}))^2, \tag{3.84}$$

with a_i being a function of ac_i, m_i, and T_{r_i} where:

$$ac_i = \frac{0.457235 R^2 T_{c_i}^{\ 2}}{p_{c_i}^{\ 2}}, \tag{3.85}$$

and

$$m_i = 0.37464 + 1.5422\omega_i - 0.26992\omega_i^{\ 2}, \tag{3.86}$$

$$T_{r_i} = T / T_{c_i}, \tag{3.87}$$

where T_{c_i} and p_{c_i} are the critical temperature and critical pressure respectively, ω_i is the acentric factor and T_{r_i} is the reduced temperature of component i.

The next step is to determine the mole fractions of liquid and vapor in the mixture. This is done through first estimating equilibrium ratios of each component using the Wilson correlation (Wilson, 1968):

$$K_i = \frac{p_{c_i}}{p} \exp(5.37(1 + \omega_i)(1 - \frac{T_{c_i}}{T})), \tag{3.88}$$

where K_i is the equilibrium ratio of component i. With the first estimation of the equilibrium ratios the Rachford-Rice procedure described below can then be used to determine the mole fractions of liquid and vapor in the mixture. In order to stay consistent with the notation typically used in compositional reservoir modelling, liquid and vapor phases will henceforth be referred to as oil and gas phases respectively. In order The Rachford-Rice equation states that (Rachford and Rice, 1952):

$$\sum_{i=1}^{N_c} \frac{z_i(K_i - 1)}{1 + (K_i - 1)V} = 0, \tag{3.89}$$

where z_i is the composition of each component i and V is the gas split of the mixture. The Rachford-Rice equation can be solved using iterative methods to find V, which then allows the mole fractions of each component in each phase to be determined through equations (3.90)-(3.92):

$$L = 1 - V, \tag{3.90}$$

$$x_{io} = \frac{z_i}{1 + (K_i - 1)V}, \tag{3.91}$$

$$x_{ig} = \frac{K_i z_i}{1 + (K_i - 1)V}, \tag{3.92}$$

where n^o is the liquid split, x_{io} is the mole fraction of each component in the oil phase and x_{ig} is the mole fraction of each component in the vapor phase.

With the mole fraction of each phase known, the overall attraction and co-volume and attraction parameters can now be calculated in each phase. The equations for the overall co-volume and attraction parameters in each phase are:

$$b = \sum_{i=1}^{N_c} x_i b_i \ , \tag{3.93}$$

and

$$a = \sum_{i=1}^{N_c} \sum_{j=1}^{N_c} x_i x_j (1 - \kappa_{ij}) \sqrt{a_i a_j} \ , \tag{3.94}$$

where b is the overall liquid co-volume parameter, a is the overall liquid attraction parameter and κ_{ij} is the binary interaction parameter between two components i and j. In terms of compressibility, the PR EOS takes the form:

$$Z^3 - (1 - B)Z^2 + (A - 2B - 3B^2)Z - (AB - B^2 - B^3) = 0 \ , \tag{3.95}$$

where

$$A = \frac{ap}{(RT)^2} \ , \tag{3.96}$$

$$B = \frac{bp}{RT} \ , \tag{3.97}$$

where Z is the compressibility factor. Through numerical methods the three roots of equation (3.95) can be calculated. When calculating the compressibility factor of oil the lowest value of Z is taken, when calculating the compressibility factor of gas the highest value of Z is taken. The next step is to check for equilibrium, meaning:

$$f_{io} = f_{ig} \ , \tag{3.98}$$

where f_{io} is the oil fugacity of each component and f_{ig} is the gas fugacity of each component. To calculate the fugacities the following equation is used:

$$f_{i\alpha} = \varphi_{i\alpha} x_{i\alpha} p \ ; \ \ \alpha = o, g \ , \tag{3.99}$$

with:

$$\ln \varphi_{i\alpha} = \frac{b_i}{b}(Z_\alpha - 1) - \ln(Z_\alpha - B_\alpha) - \tag{3.100}$$

$$\frac{A_\alpha}{B_\alpha(-2\sqrt{2})}\left(\frac{2\sum_{j=1}^{N_c} x_{j\alpha}(1-\kappa_{ij\alpha})\sqrt{a_{i\alpha}a_{j\alpha}}}{a} - \frac{b_{i\alpha}}{b} \right) \ln\left(\frac{Z_\alpha + (1-\sqrt{2})B_\alpha}{Z_\alpha + (1+\sqrt{2})B_\alpha}\right) \ ,$$

where φ_i is the fugacity coefficient for each component i. Equation (3.101) is used to check for equilibrium, and if equilibrium is not reached the equilibrium ratio K is updated for each component using equation (3.102):

$$\sum_{i=1}^{Nc} \left(1 - \frac{f_{io}}{f_{ig}}\right)^2 \leq 10^{-12} \ , \tag{3.101}$$

$$K_i^{\,s+1} = K_i^{\,s} \left(\frac{f_{io}}{f_{ig}} \right)^s \ , \tag{3.102}$$

where s refers to the iteration level of the flash calculation. Iterations of equations (3.89)-(3.102) are repeated until equilibrium has been reached. Once equilibrium has been found the equilibrium mole fractions x_{io} and x_{ig} can be used to determine the mixture properties.

Before using the multiphase flash calculation the system is first checked to see if it is a single phase (Danesh, 1998). The stability check was proposed by Michelsen (1982), and is used to determine if the system will be a single phase or if the flash calculation is required. The following approach was presented by Whitson and Brulé (2000). First the overall system fugacity f_i is computed using them same equations listed above with the assumption of only one phase (hence no need to iterate and update for K). Then, a vapor-like second phase is created, and the second phase mole numbers and mole fractions are computed through the following equations:

$$Y_i = z_i K_i \ , \tag{3.103}$$

$$S_v = \sum_{i=1}^{Nc} Y_i \ , \tag{3.104}$$

$$y_i = \frac{Y_i}{S_V} \ , \tag{3.105}$$

where Y_i is the vapor-like phase mole numbers and S_V is the sum of the vapor-like phase mole numbers. This system is iterated as described above, but the new iteration criteria for updating equations are:

$$R_i = \frac{f_{iz}}{f_{ig}} \frac{1}{S_V} \ , \tag{3.106}$$

$$\sum_{i=1}^{Nc} (R_i^2 - 1)^2 \le 10^{-10} \ , \tag{3.107}$$

where R_i is a correction factor. The K values are then updated using equation (3.108).

$$K_i^{s+1} = K_i^s R_i \ . \tag{3.108}$$

The iteration process is terminated before convergence if the trivial solution is approached:

$$\sum_{i=1}^{Nc} (\ln K_i)^2 \leq 10^{-4} . \tag{3.109}$$

This process is repeated by creating a liquid-like second phase with equations (3.103) - (3.106) being replaced by equations (3.110) - (3.113) respectively.

$$X_i = z_i / K_i \ , \tag{3.110}$$

$$S_L = \sum_{i=1}^{Nc} X_i \ , \tag{3.111}$$

$$x_i = \frac{X_i}{S_L} \ , \tag{3.112}$$

$$R_i = \frac{f_{io}}{f_{iz}} S_L . \tag{3.113}$$

where X_i is the liquid-like phase mole numbers and S_L is the sum of the vapor-like phase mole numbers. The system is single phase stable if the tests yields that both S_V and S_L are less than 1, or if both tests yield trivial solutions, or if one test yields a trivial solution and the other test yields a sum less than one. If none of these conditions are met, the system is considered to have two phases and the two phase flash calculation described above is used.

3.4.4 Hydrocarbon Viscosity Model

A method for calculating the viscosity of hydrocarbon mixtures was developed by Lohrenz, Bray and Clark (1964), the method is commonly referred to as the LBC viscosity prediction method. Jossi et al. (1962) first proposed a model for predicting the viscosity of pure compounds, and Lohrenz, Bray and Clark extended this to hydrocarbon mixtures. The equation for predicting viscosity is:

$$\left[(\mu - \mu^0)\lambda + 10^{-4} \right]^{1/4} = a_1 + a_2 \rho_r + a_3 \rho_r^{\ 2} + a_4 \rho_r^{\ 3} + a_5 \rho_r^{\ 4} \ , \tag{3.114}$$

where ρ_r is the reduced density, μ^0 is the low pressure viscosity, λ is multiplied by viscosity to make it dimensionless in the equation, and the a values are the following constants:

$$\begin{aligned} a_1 &= 0.10230 \\ a_2 &= 0.023364 \\ a_3 &= 0.058533 \\ a_4 &= -0.040758 \\ a_5 &= 0.0093324 \end{aligned} \quad .$$

For hydrocarbon mixtures, μ^0 and λ are calculated through equations (3.115) and (3.116) respectively:

$$\mu^0{}_\alpha = \frac{\left[\sum_{i=1}^{Nc} x_{i\alpha}\mu_i{}^0 MW_i^{1/2}\right]}{\left[\sum_{i=1}^{Nc} x_{i\alpha} MW_i^{1/2}\right]} \quad , \tag{3.115}$$

$$\lambda_\alpha = \left(\sum_{i=1}^{Nc} x_{i\alpha} T_{c_i}\right)^{1/6} \left(\sum_{i=1}^{Nc} x_{i\alpha} MW_i\right)^{-1/2} \left(\sum_{i=1}^{Nc} x_{i\alpha} P_{c_i}\right)^{-2/3} \quad , \tag{3.116}$$

where MW_i is the molecular weight, and $\mu_i{}^0$ is the single component low pressure viscosity calculated through equation (3.117):

$$\begin{aligned} \mu_i{}^0 &= 34\times10^{-5} T_r{}^{0.94} / \lambda_i \qquad T_r \le 1.5 \\ \mu_i{}^0 &= 17.78\times10^{-5} (4.58T_r - 1.67)^{5/8} / \lambda_i \qquad T_r > 1.5 \\ \lambda_i &= T_c{}^{1/6} MW^{-1/2} P_c{}^{-2/3} \quad . \end{aligned} \tag{3.117}$$

Reduced density is calculated as:

$$\rho_{r_\alpha} = \frac{\sum_{i=1}^{Nc} x_{i\alpha} v_{c_i}}{v_\alpha} \quad , \tag{3.118}$$

where v_{c_i} is the critical molar volume of each component, and v is the molar volume determined from the EOS. The critical molar volume of the C_{7+} fraction can be calculated as:

$$v_{c_{(C_{7+})}} = 1.3468 + 9.4404\times10^{-4} MW_{C_{7+}} - 1.72641 SG_{C_{7+}} + 4.4083\times10^{-3} MW_{C_{7+}} SG_{C_{7+}} \quad (3.119)$$

,

where $SG_{C_{7+}}$ is the specific gravity of the C_{7+} fraction. The units used in this correlation are K for temperature, atm for pressure, mPa·s for viscosity, g/gmol for molecular weight and cm^3/mol for volume.

3.4.5 Hydrocarbon Interfacial Tension Model

The model used to predict interfacial tension between the oil and gas phases was developed by Weinaug and Katz (1943) for hydrocarbon mixtures by extending the method developed by Macleod and Sugden for pure components. The Weinaug and Katz method uses simple molar averaging of the parachor (Danesh, 1998). The equation for interfacial tension between the oil and gas phases can be seen in equation (3.120):

$$\sigma_{og} = \left[\sum_{i=1}^{Nc} P_{\sigma i} \left(x_{io}\xi_o - x_{ig}\xi_g \right) \right]^4 , \quad (3.120)$$

where $P_{\sigma i}$ is the parachor of component i.

3.5 Validation of Model

Each component of the compositional reservoir simulator was validated through published measured results and the reservoir flow in the simulator was validated through an analytical model. First the fluid property models were validated. This included the equation of state flash calculation, viscosity model, and hydrocarbon interfacial tension model. These models were validated against experimental and analytical solutions provided in *PVT and Phase Behaviour of Petroleum Reservoir Fluids* (Danesh, 1998). Then the reservoir flow itself was validated. Both constant pressure and constant rate boundaries were used for validation to ensure the model could handle both.

3.5.1 Validation of Flash Calculation

The PR EOS is used for flash calculations in the reservoir simulation. The flash calculation model was created as a function in MATLAB. In order to validate the results of the flash calculation MATLAB function, the outputs were compared with data provided in *PVT and Phase Behaviour of Petroleum Reservoir Fluids* (Danesh, 1998). The author used an analytical method to complete a flash calculation of two-component reservoir fluid with known composition and experimentally measured flash calculation outputs. This work was repeated to show that the flash calculation function created in MATLAB agrees with experimental data. The reservoir fluid is produced through a one stage separator at 344.3 K and 6.895 MPa. Table 3.1 shows the fluid composition, and Table 3.2 shows the flash calculation outputs using the different methods.

Table 3.1 – Two Component Flash Calculation Function Validation Fluid Composition

Component	Mole Fraction
C_1	0.60
nC_{10}	0.40

Table 3.2 – Two Component Flash Calculation Function Validation Results

	Results					
Method of Calculation	K_1	K_2	x_{1o}	x_{2o}	x_{1g}	x_{2g}
Experimental	4.005	0.0027	0.2496	0.7504	0.998	0.0020
Analytical Solution	3.8	0.0029	0.263	0.737	0.999	0.001
MATLAB Function	4.224	0.0028	0.2362	0.7638	0.9979	0.0021

These results show that the MATLAB function used for flash calculations is in agreement with the experimentally measured values (3.55% average error), and for this case predicts more accurately than the analytical solution (11.63% average error).

In order to validate the model for more than two components, the function was tested against a full array system using PVTsim software. The oil composition was taken from *PVT and Phase Behaviour of Petroleum Reservoir Fluids* (Danesh, 1998) and can be seen in Table 3.3.

Table 3.3 – Full Array Flash Calculation Function Validation Fluid Composition

Component	Mole Fraction
CO_2	0.0091
N_2	0.0016
C_1	0.3647
C_2	0.0967
C_3	0.0695
iC_4	0.0144
nC_4	0.0393
iC_5	0.0144
nC_5	0.0141
C_6	0.0433
C_{7+}	0.3329

C_{7+} properties: MW = 218 g/mol, Density = 851.5 kg/m^3

The flash calculation was performed in the MATLAB function using the full array, as well as grouping into three pseudo-components. The three pseudo-components were chosen as: CO_2, N_2 and C_1; C_2-C_6; and C_{7+}. The simple mixing rule was used to group the pseudo-component properties for the components lighter than heptane. The temperature used was the reservoir temperature of 378K, and the pressures used were the reservoir pressure of this oil listed in the textbook (28.37 MPa) and a pressure of 10 MPa to verify the MATLAB functions validity at low pressures. The results are shown in

Table 3.4.

Table 3.4 – Full Array Flash Calculation Function Validation Results

		Results					
P [MPa]	**Method of Calculation**	**L**	**V**	**Z_o**	**Z_g**	**ρ_o [kg/m^3]**	**ρ_g [kg/m^3]**
10	PVTsim	0.8297	0.1703	0.6218	0.8507	685.30	106.73
	No Grouping MATLAB	0.7293	0.2707	0.5919	0.8605	712.26	80.72
	Grouping MATLAB	0.7859	0.2141	0.5709	0.8269	687.50	88.91
28.37	PVTsim	1	0	1.2505	N/A	676.72	N/A
	No Grouping MATLAB	1	0	1.3422	N/A	682.77	N/A
	Grouping MATLAB	1	0	1.3548	N/A	674.37	N/A

It can be seen that the results of the MATLAB function using grouping match up quite well with PVTsim. Some of the deviations in density prediction can be caused by the estimation of the volume shift parameter for the C_{7+} fraction.

3.5.2 Validation of Viscosity Model

The viscosity model used in simulation is the LBC model, described in section 3.4.4. The model was created as a function in MATLAB. The viscosity model is validated through data provided in *PVT and Phase Behaviour of Petroleum Reservoir Fluids* (Danesh, 1998). In this textbook the author used the LBC viscosity prediction method to calculate the liquid viscosity of an oil with known composition and measured viscosity. This work was repeated to show that the viscosity model created in MATLAB agrees with the LBC prediction method as well as experimental data.

The liquid mixture is at 311 K and 20.68 MPa with a density of 0.368 g/cm^3. Table 3.5 shows the fluid composition, and Table 3.6 shows the viscosity using the different methods.

Table 3.5 – Viscosity Model Validation Fluid Composition

Component	Mole Fraction
C_1	0.593
C_3	0.3746
nC_8	0.0324

Table 3.6 – Viscosity Model Validation Fluid Viscosity

Method of Calculation	Viscosity [mPa.s]
Experimental	0.0510
LBC prediction Danesh Textbook	0.04684
MATLAB Function	0.04939

These results show that the MATLAB function used to predict viscosity is in agreement with the experimentally measured values.

3.5.3 Validation of Hydrocarbon Interfacial Tension Model

The model used in simulation for the interfacial tension between the oil and gas phases is described in section 3.4.5. The model was created as a function in MATLAB, and is validated through data provided in *PVT and Phase Behaviour of Petroleum Reservoir Fluids* (Danesh, 1998), similarly to the viscosity function validation. In this textbook the author used the Weinaug and Katz method to calculate the interfacial tension between the oil and gas phases of the hydrocarbons. This work was repeated to show that the interfacial tension model created in MATLAB agrees with the prediction method from the textbook as well as experimental data. The mixture is 60% C_1 and 40%

nC_{10} at 377.6 K and 23.59 MPa. Table 3.7 shows the fluid properties, and Table 3.8 shows the interfacial tension using the different methods.

Table 3.7 – Hydrocarbon Interfacial Tension Model Validation Properties

Phase	Density [g/cm^3]	Mole Fraction of Methane
C_1	0.5447	0.6000
C_3	0.1435	0.9825

Table 3.8 – Hydrocarbon Interfacial Tension Model Validation

Method of Calculation	Interfacial Tension [mN/m]
Experimental	2.4
Weinaug Katz prediction Danesh Textbook	1.708
MATLAB Function	1.708

These results show that the MATLAB function used to predict hydrocarbon interfacial tension is in decent agreement with the experimentally measured values, but in very good agreement with the prediction used by the authors of the textbook.

3.5.4 Validation of Reservoir Flow

The reservoir flow itself was validated through the comparison of the developed simulator to an analytical solution. Both constant rate boundaries and constant pressure boundaries were validated as these are each required to run the huff 'n' puff process.

3.5.4.1 Constant Rate

To validate the flow under constant rate boundaries fractional flow theory was used to validate the reservoir when a waterflood was run in a one-dimensional Cartesian model with just oil and connate water in the reservoir. The cases were developed to match work which was completed in *Fundamentals of Reservoir Engineering* (Dake, 1978). The following tables provide the

information from the textbook which were used to develop the fractional flow model to compare with simulator developed in MATLAB.

Table 3.9 – Relative Permeability Functions for Constant Rate Reservoir Validation

S_w	k_{rw}	k_{ro}
0.2	0	0.800
0.25	0.002	0.610
0.30	0.009	0.470
0.35	0.020	0.370
0.40	0.033	0.285
0.45	0.051	0.220
0.50	0.075	0.163
0.55	0.100	0.120
0.60	0.132	0.081
0.65	0.170	0.050
0.70	0.208	0.027
0.75	0.251	0.010
0.80	0.300	0

Table 3.10 – Formation Volume Factors for Constant Rate Reservoir Validation

Phase	FVF [rb/stb]
Water	1.0
Oil	1.3

Fractional flow models are developed for three cases listed in Table 3.11:

Table 3.11 – Viscosity Ratios for Constant Rate Reservoir Validation

Case	μ_w/μ_o
1	0.01
2	0.1

3	2.5

The fractional flow curves were developed using the same method that was used in the textbook, where:

$$f_w = \frac{1}{1 + \frac{\mu_w}{k_{rw}} \cdot \frac{k_{ro}}{\mu_o}} \quad . \tag{3.121}$$

Table 3.12 shows the tabular results of the fractional flow model, and shows the graphical fractional flow curves. Welge's graphical method (Welge, 1952) was used to determine the water saturation at breakthrough and the average water saturation behind the front, denoted by $S_{w_{bt}}$ and $\overline{S}_{w_{bt}}$ respectively.

Table 3.12 – Fractional Flow Models for Reservoir Validation

	Case 1 (μ_w/μ_o = 0.01)	**Case 2** (μ_w/μ_o = 0.1)	**Case 3** (μ_w/μ_o = 2.5)
S_w	**f_w**	**f_w**	**f_w**
0.2	0	0	0
0.25	0.24691	0.03175	0.00131
0.30	0.65693	0.16071	0.0076
0.35	0.84388	0.35088	0.02116
0.40	0.9205	0.53659	0.04427
0.45	0.95865	0.69863	0.08486
0.50	0.97873	0.82147	0.15544
0.55	0.98814	0.89286	0.25
0.60	0.9939	0.94218	0.39462
0.65	0.99707	0.97143	0.57627
0.70	0.9987	0.98719	0.75499
0.75	0.9996	0.99603	0.90942
0.80	1	1	1

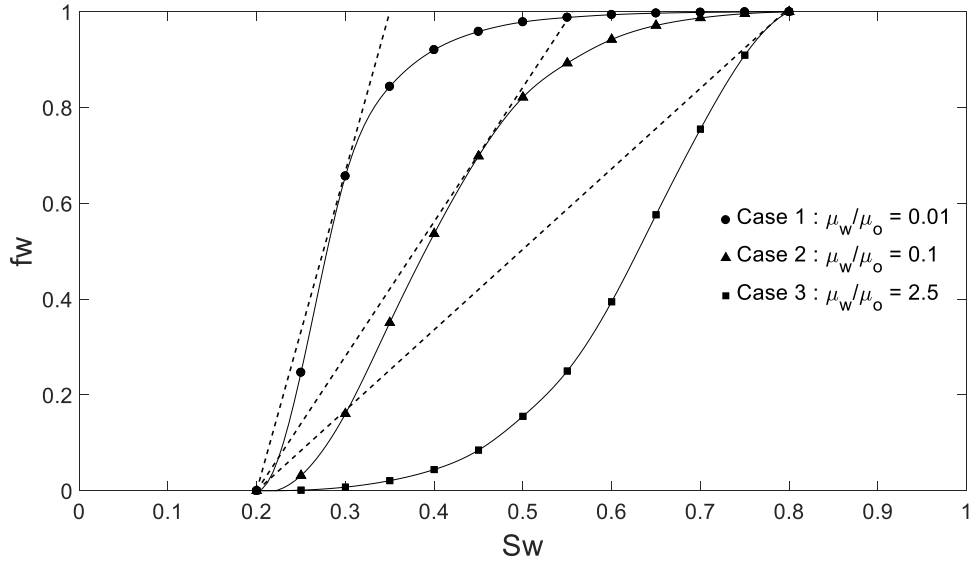


Figure 3.9 – Fractional Flow Functions for Constant Rate Reservoir Validation

Figure 3.9 shows the fractional flow functions plotted graphically for each of the three cases used in this validation. Welge's graphical technique was used to determine the water saturation and average water saturation behind the front at breakthrough. These are found by drawing a tangent to the fractional flow curve from the point of the connate water saturation (co-ordinates 0.2,0 on Figure 3.9). The saturation value when this tangent intersects the fractional flow curve is the average water saturation at breakthrough, and the saturation value where this tangent line intersects the line $f_w = 1$ represents the average saturation behind the front at breakthrough. The results of using Welge's graphical technique are provided in Table 3.13.

Table 3.13 – Results of Welge's Graphical Technique

Case	$S_{w_{bt}}$	$\overline{S}_{w_{bt}}$	μ_w / μ_o
1	0.28	0.34	0.01
2	0.45	0.55	0.1
3	0.80	0.80	2.5

The water saturation at break through and time to break through were the parameters used to validate the reservoir flow. The time to break through was calculated using equation (3.122)

$$t_{bt} = \frac{W_{id} PV}{Q} \ , \tag{3.122}$$

where PV is one pore volume, Q is the injection rate and W_{id} is the dimensionless water influx which is represented by:

$$W_{id} = \overline{\overline{S}}_{w_{bt}} - S_{wc} \ . \tag{3.123}$$

The reservoir flow in the compositional simulator created in MATLAB was then validated by running the three cases listed in *Fundamentals of Reservoir Engineering* and comparing the time to breakthrough and the water saturation at breakthrough. The model uses the same relative permeability function shown in Table 3.9, and the rest of the input parameters are shown in Table 3.14.

Table 3.14 – Input Parameters for Constant Rate Reservoir Flow Validation

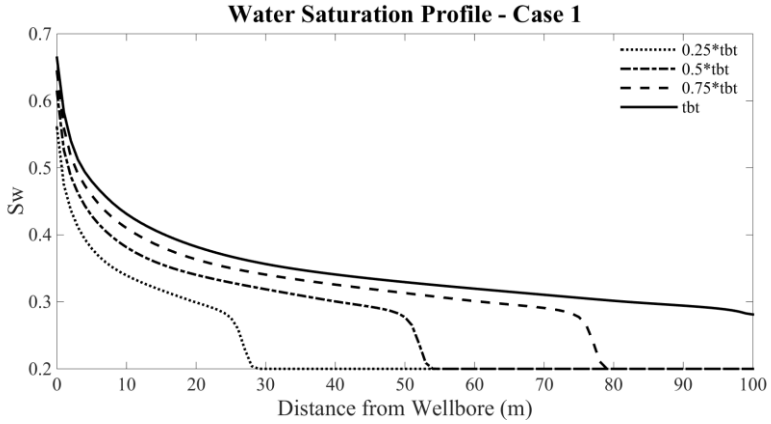
Input Parameter	Value
Oil Composition	100% nC_{10}
Porosity	18%
Length	100 m
Width	15 m
Depth	5 m
Water Injection Rate	0.00184 m^3/s

The oil viscosity changes with pressure as described earlier. In order to emulate the value of μ_w / μ_o for each case the oil viscosity was taken at the initial pressure, and the water viscosity was set as a constant fraction of the initial oil viscosity. The reservoir was divided into 100 grid blocks in order to accurately model the behaviour. The total pore volume is calculated through equation (3.124).

$$PV = \phi \Delta x \Delta y \Delta z = 0.18 \cdot 100 \cdot 15 \cdot 5 = 1350 m^3 \ , \tag{3.124}$$

Combining equations (3.124) and (3.123) with (3.122) allows for the solution of the water breakthrough time analytically, which is summarized for each case with the water saturation at breakthrough in Table 3.15.

The three cases were then run using the developed compositional simulator; the breakthrough time and the water saturation at breakthrough are also summarized in Table 3.15. The water saturation profiles of the three cases are shown in Figure 3.10. These water saturation profiles show that as the viscosity ratio increases, the time to breakthrough increases as well as the saturation at breakthrough. This is expected because the viscosity ratio is directly proportional to the mobility ratio, and with a lower mobility ratio in an injection process there is a more even front produced. This even front causes the time to breakthrough to increase as the water is not fingering into the reservoir. In the legend of each plot *tbt* represents the time to breakthrough.



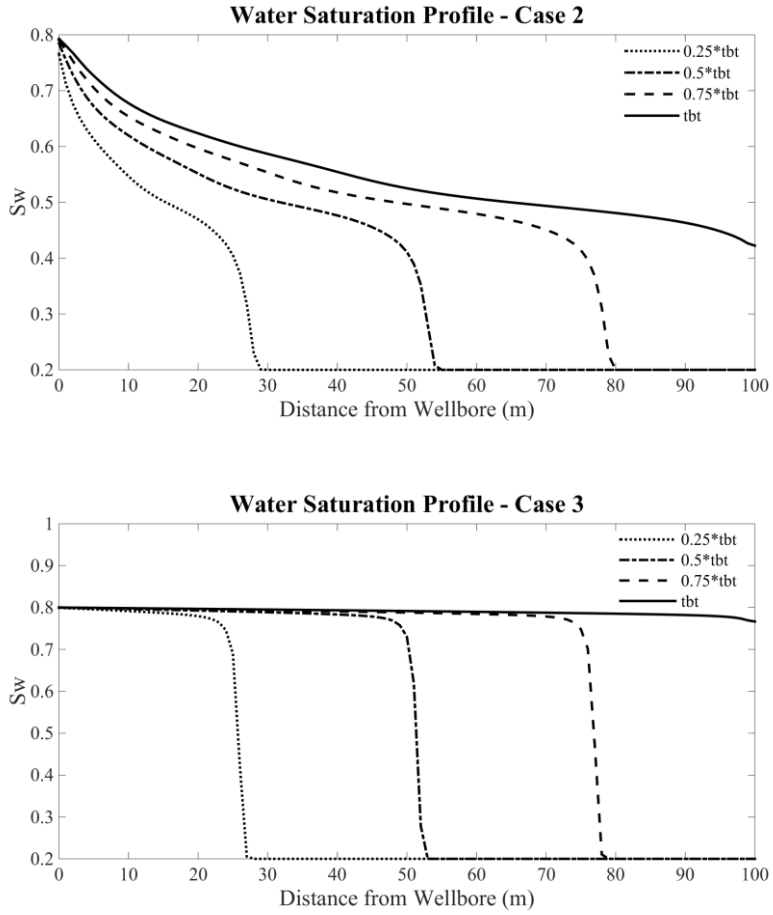


Figure 3.10 – Water Saturation Profiles for Constant Rate Reservoir Flow Validation

It can be seen from Table 3.15 that the time to breakthrough and water saturation at breakthrough match very well with the predicted values from the analytical solution. The reason for the discrepancies is discussed in section 3.5.4.3.

Table 3.15 – Analytical Breakthrough Time and Water Saturation

	Case 1			Case 2			Case 3		
	Analytical	**Simulator**	**Error**	**Analytical**	**Simulator**	**Error**	**Analytical**	**Simulator**	**Error**
$S_{w_{bt}}$	0.28	0.28	0.0%	0.45	0.42	6.7%	0.80	0.78	2.5%
t_{bt}	28.5 h	29.5 h	3.5%	71.3 h	72.0 h	1.0%	122.3 h	123.0 h	0.6%

3.5.4.2 Constant Pressure

To validate the flow under constant pressure boundaries, an analytical solution developed by Johansen and James (2012) was used which is an extension of the classical Buckley-Leverett Theory. The results of the analytical solution were developed by Yang (2014) and are simply presented here for comparison to the numerical model. A one-dimensional Cartesian reservoir model with just oil and connate water in the reservoir was waterflooded. The Corey model as described in section 3.4.2 was used for relative permeability.

Table 3.16 – Input Parameters for Constant Pressure Reservoir Flow Validation

Input Parameter	Value
Oil Composition	100% nC_{10}
Porosity	18%
Permeability	1E-12 m^2
Length	100 m
Inlet Pressure	21 MPa
Outlet Pressure	17 MPa
Water Injection Rate	0.00184 m^3/s
Residual Oil Saturation	0.3
Connate Water Saturation	0.25

Table 3.17 – Relative Permeability Information for Constant Pressure Reservoir Validation

Relative Permeability Parameter	Value
Model	Corey
n_w, n_o	2
$k_{rw_{\max}}$	0.2
$k_{ro_{\max}}$	0.8

Table 3.18 – Case Information for Constant Pressure Reservoir Validation

Case	μ_w	μ_o	Analytical t_{bt}
1	0.001 Pa·s	0.02 Pa·s	28.5 days
2	0.02 Pa·s	0.001 Pa·s	122.7 days

The cases were then run using the developed compositional simulator; water saturation profiles for each case can be seen in Figure 3.11. In each plot *tbt* represents the time to breakthrough.

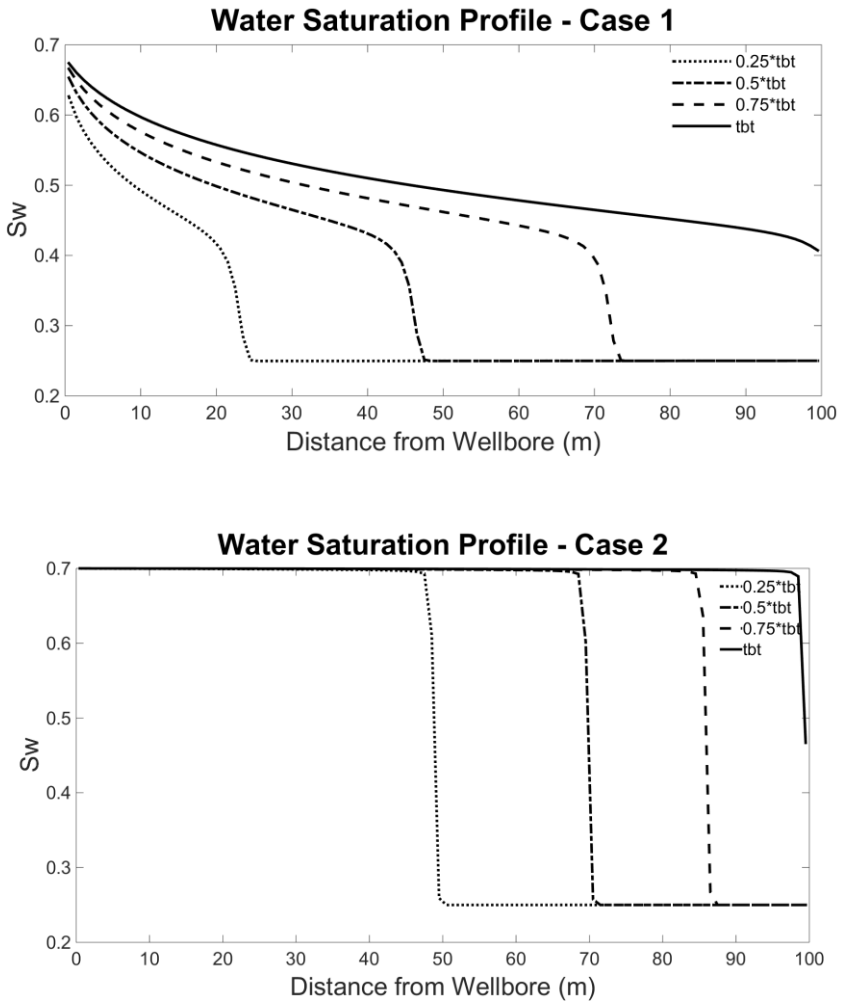


Figure 3.11 – Water Saturation Profiles for Constant Pressure Reservoir Flow Validation

The breakthrough time from the simulator for Case 1 was 27 days and for Case 2 it was 121 days. Similar to the constant rate validation, it can be seen that in the constant pressure model the simulation with the lower mobility ratio causes a more even water front to move through the reservoir, which increases the time to breakthrough. In Case 1, the higher mobility ratio, the water fingers into the reservoir hence causing an early breakthrough. By comparing the simulated

breakthrough times to the analytical breakthrough times shown in Table 3.18, it can be seen the numerical simulator matches the analytical solution very well. The discrepancies are discussed in section 3.5.4.3.

3.5.4.3 Cause of Error

In both the constant rate and constant pressure validation the numerical simulator did not match the analytical solution exactly, but the results were very close. This discrepancy can be attributed in both cases primarily to numerical dispersion. The breakthrough front is not a clean front in the numerical model, which makes the exact breakthrough time impossible to determine. Figure 3.12 shows the water saturation of Case 2 of the constant rate injection at 30 hours. The analytical solution is shown along with the results of the numerical simulator using different numbers of grid blocks. It can be seen that as the number of grid blocks increases the simulator gets closer and closer to the analytical solution, but this is at the cost of simulation time.

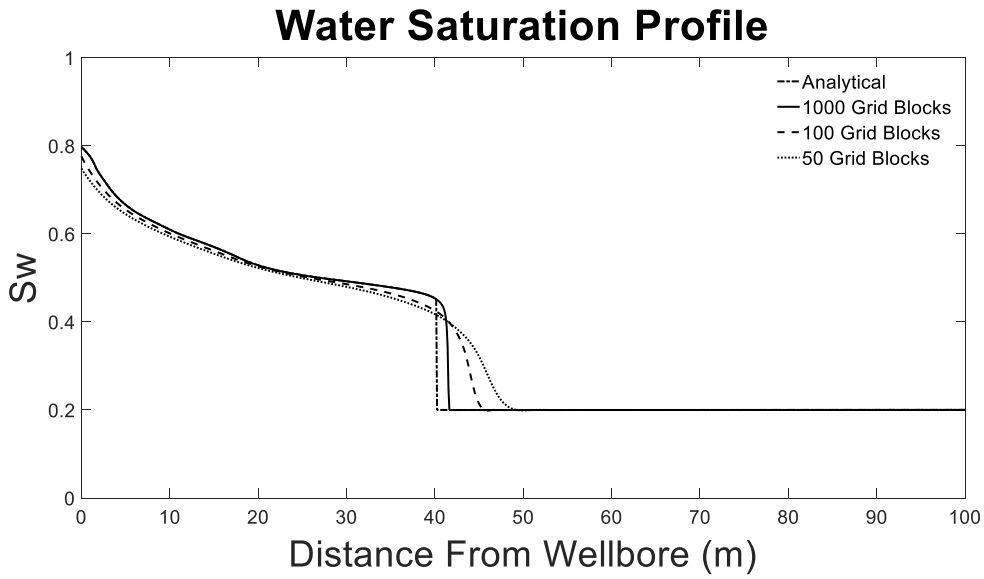


Figure 3.12 – Numerical Dispersion Error

Chapter 4 Numerical Study of Natural Gas Huff ‘n’ Puff

This section provides the simulations completed on the natural gas huff ‘n’ puff process using the simulator developed in the preceding chapter. The results of the simulations are presented, and the conclusions and recommendations from the numerical study are discussed in the proceeding section.

4.1 Model Properties

This section will describe properties used for the case study to evaluate the natural gas huff ‘n’ puff. Reservoir properties are kept constant throughout the simulations as the area of interest is only in the near well region. Absolute permeability and porosity are not being studied. Therefore, they remain constant throughout the study as well. General assumptions for all simulations are:

1. One dimensional flow
2. Permeability and porosity are constant
3. Capillary pressures are negligible
4. Reservoir temperature does not change
5. Hysteresis does not take place
6. Bottom hole pressure and well rates can be set at the reservoir

It is important to note that in a real huff ‘n’ puff process hysteresis would play an important role in oil recovery, but due to complexity it was not modelled for this study. It has been noted in literature (Liu et al., 2005) that the hysteresis effect can contribute to enhanced oil recovery. The relative permeability of the puff phase will be modeled more accurately using hysteresis, thus giving more accurate production values when comparing different injection parameters of the huff

'n' puff process. It is also assumed that well rates and pressures can be set at the reservoir, this is not realistic but it is an adequate assumption to study the huff 'n' puff process.

4.1.1 Reservoir Properties

The reservoir geometry used for the huff 'n' puff simulations is the one-dimensional radial geometry described in Chapter 3. Reservoir properties can be seen in Table 4.1, with a schematic seen in Figure 4.1. The wellbore boundary begins with constant rate injection, and then the bottom-hole pressure is set to a constant value for production. The drainage boundary is set to a constant pressure boundary. Drainage radius is set at 4000 m. The drainage boundary is set to be large to ensure the constant pressure boundary is valid and no pressure response from the wellbore will be felt at the drainage radius. The Corey model, as described in Chapter 3.4.2, is used to describe the relative permeability of the field.

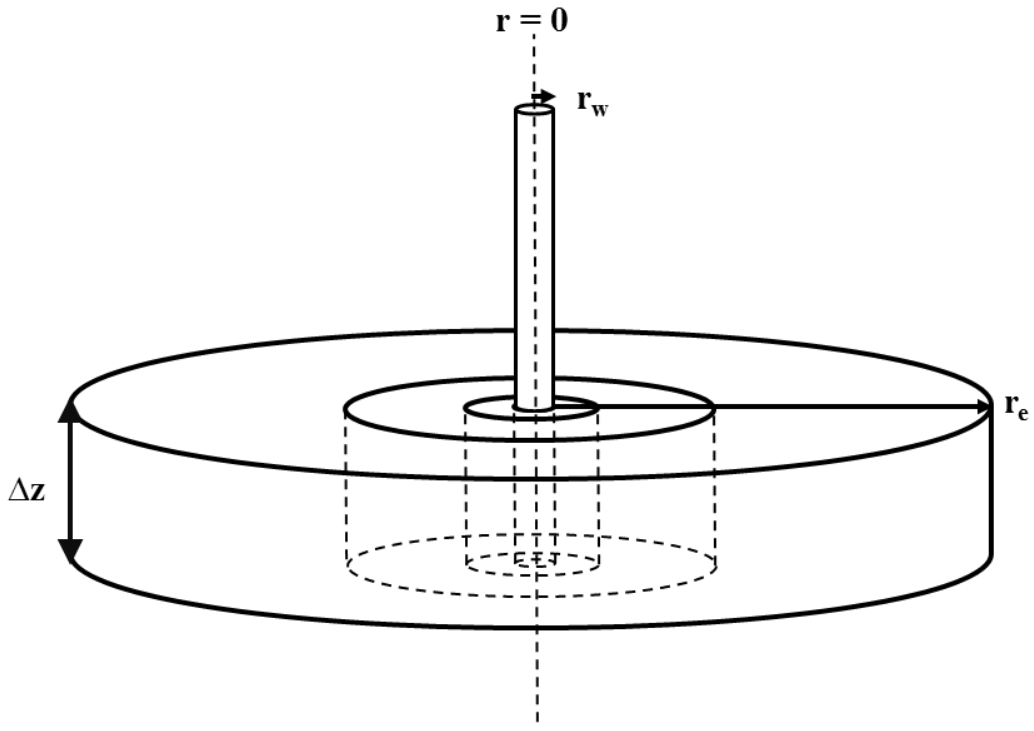


Figure 4.1 – Reservoir Schematic

The tables below list the reservoir properties as well as the relative permeability data used for all simulations. The reservoir and fluid properties are based on internal Hibernia Documentation, and the relative permeability data is taken from the Hebron Development Plan (2011).

Table 4.1 – Reservoir Properties

Parameter	Units	Value
Permeability	mD	500
Porosity	-	0.18
Wellbore Radius	m	1
Drainage Radius	m	4000
Thickness	m	5
Temperature	K	373.15
Initial Pressure	MPa	45

Table 4.2 – Relative Permeability Data

Parameter	Value
S_{wc}	0.18
S_{orw}	0.20
S_{org}	0.15
S_{gc}	0.03
$k_{r_{w\max}}$	0.3
$k_{r_{ow\max}}$, $k_{r_{og\max}}$	1
$k_{r_{g\max}}$	0.9
n_w	2
n_{ow}	2
n_{og}	2
n_g	2

4.1.2 Phase Behaviour Analysis

The composition of the reservoir fluid used for all case studies can be seen in Table 4.3.

Table 4.3 – Reservoir Fluid Properties

Component	Mole Fraction
N_2	0.0031
CO_2	0.0082
H_2S	0
C_1	0.4249
C_2	0.0467
C_3	0.046
iC_4	0.0082
nC_4	0.0217
iC_5	0.0097
nC_5	0.0126
nC_6	0.0333
C_{7+}	0.3856

C_{7+}: MW = 261.34 g/mol, Density = 881.18 kg/m^3

In order to study the effect of different gas compositions on the huff 'n' puff process three different gas compositions were examined. Table 4.4 shows a representation of the natural gas found in the Hibernia B-16 block, Table 4.5 shows a gas composition where intermediates have been added to the natural gas for enrichment to lower the minimum miscibility pressure (MMP) with the reservoir fluid, and Table 4.6 shows a composition which is further enriched to lower the MMP even further.

Table 4.4 – Gas 1: Hibernia Natural Gas (C_1 = 0.9, Intermediates = 0.1)

Component	Mole Fraction
C_1	0.90
C_2	0.05
C_3	0.025
iC_4	0.0125
nC_4	0.0125

Table 4.5 – Gas 2: First Enrichment (C_1 = 0.8, Intermediates = 0.2)

Component	Mole Fraction
C_1	0.80
C_2	0.10
C_3	0.05
iC_4	0.025
nC_4	0.025

Table 4.6 – Gas 3: Second Enrichment (C_1 = 0.7, Intermediates = 0.3)

Component	Mole Fraction
C_1	0.70
C_2	0.15
C_3	0.075
iC_4	0.0375
nC_4	0.0375

To determine the MMP of each gas with the reservoir oil, a slim tube experiment was simulated. The developed model simulates one dimensional flow, which is what is required for the slim tube simulation. A 12 m slim tube with a 6.3 mm radius was used for simulation, which is a standard slim tube mode (Danesh, 1998). The slim tube was initially filled with reservoir oil. The rock properties of the reservoir were used for the slim tube simulation. The results of each simulation can be seen in the following figures.

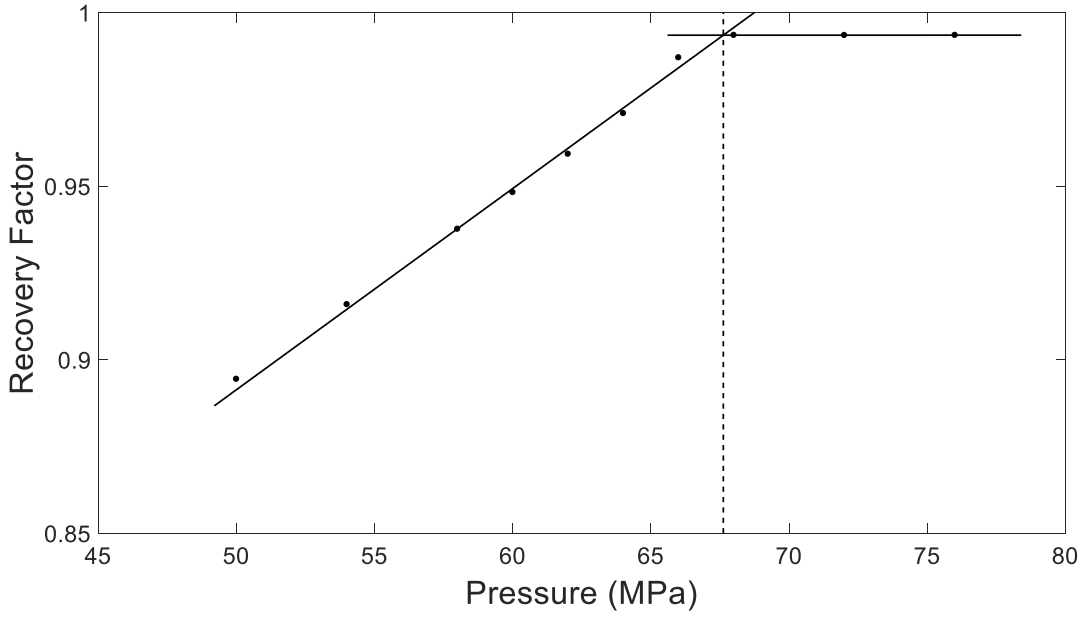


Figure 4.2 – MMP of Gas 1 (C_1 = 0.9, Intermediates = 0.1)

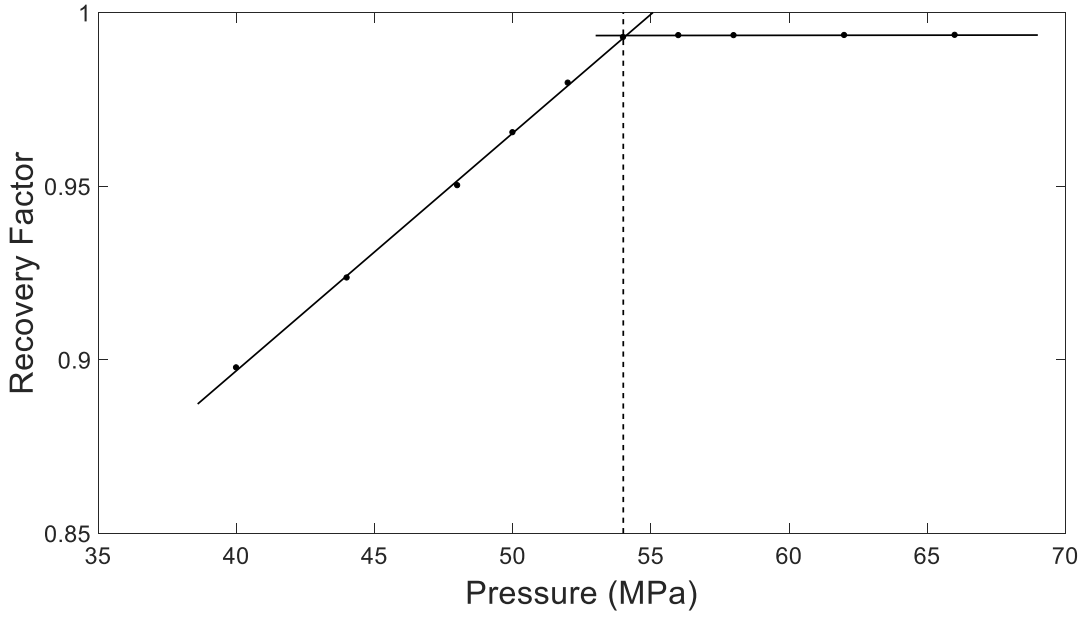


Figure 4.3 – MMP of Gas 2 (C_1 = 0.8, Intermediates = 0.2)

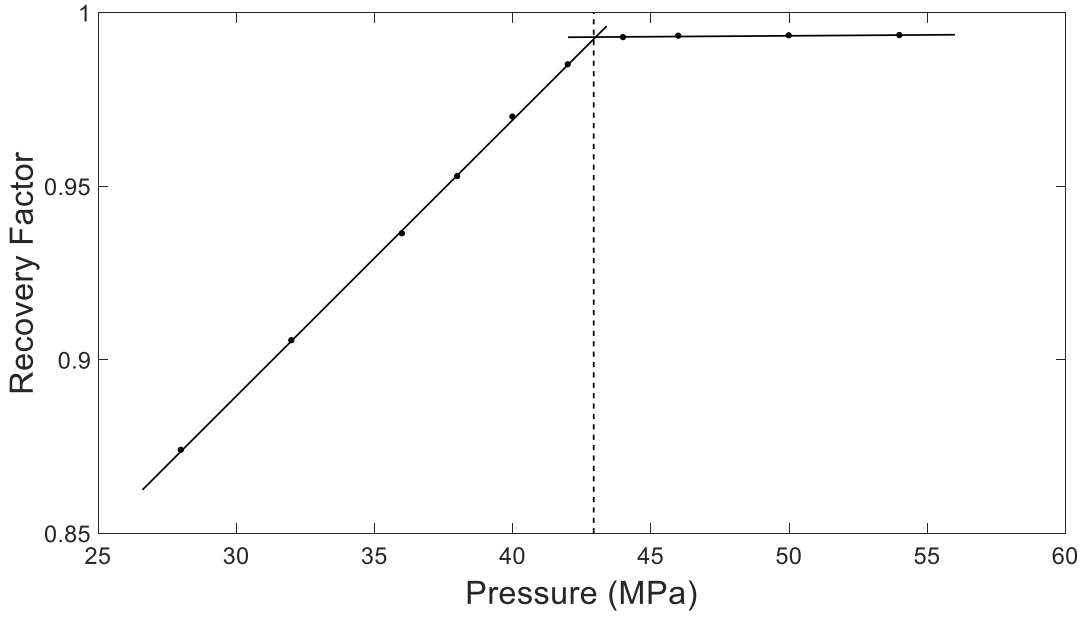


Figure 4.4 – MMP of Gas 3 (C_1 = 0.7, Intermediates = 0.3)

It can be seen from Figure 4.2 - Figure 4.4 that the MMP of Gas 1, 2 and 3 with the reservoir oil is 67.6 MPa, 54.2 MPa, and 43.0 MPa respectively. This will allow the process to be studied under fully immiscible conditions, an intermediate condition, and fully miscible conditions.

Figure 4.5 - Figure 4.7 show the pore volumes injected plotted against the recovery factor for each of the injection gases. Each plot shows the result of injecting the gas under immiscible and then miscible conditions. In immiscible injection, the gas break through occurs earlier than in miscible injection. The plots show that for each case the immiscible injection does indeed have an earlier gas break through than the miscible injection, and after breakthrough recovery slows down. Hence for each case the recovery is higher when injecting under miscible conditions.

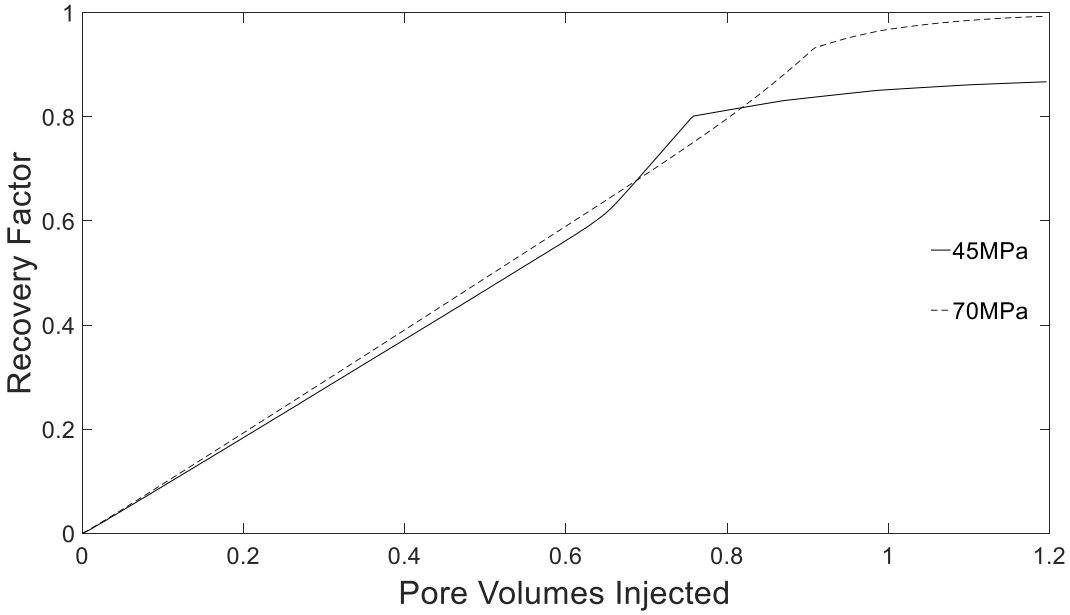


Figure 4.5 – Recovery Profile Gas 1 (C_1 = 0.9, Intermediates = 0.1)

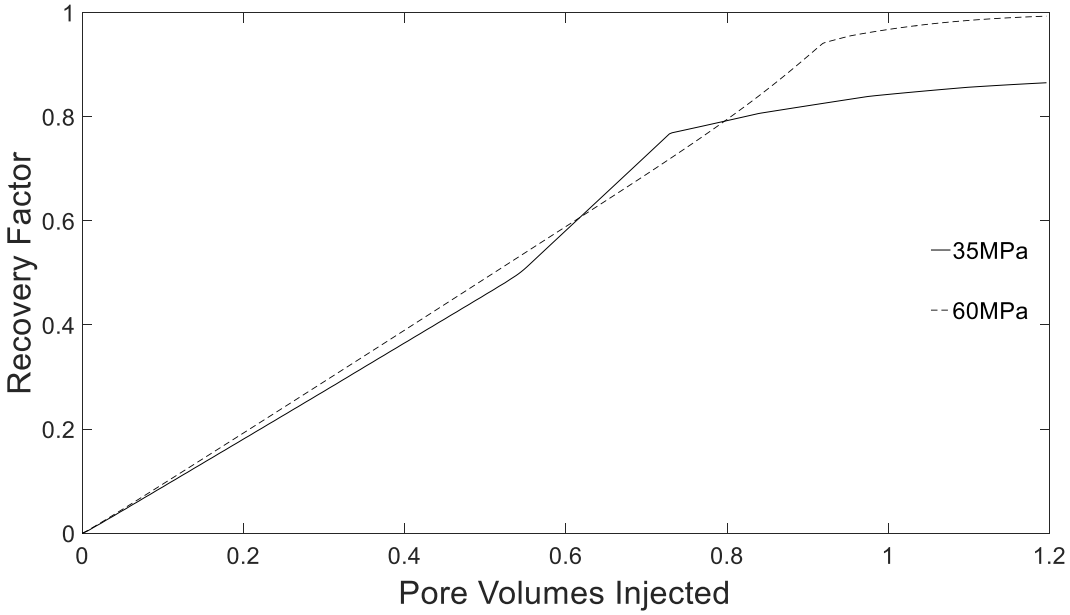


Figure 4.6 – Recovery Profile Gas 2 (C_1 = 0.8, Intermediates = 0.2)

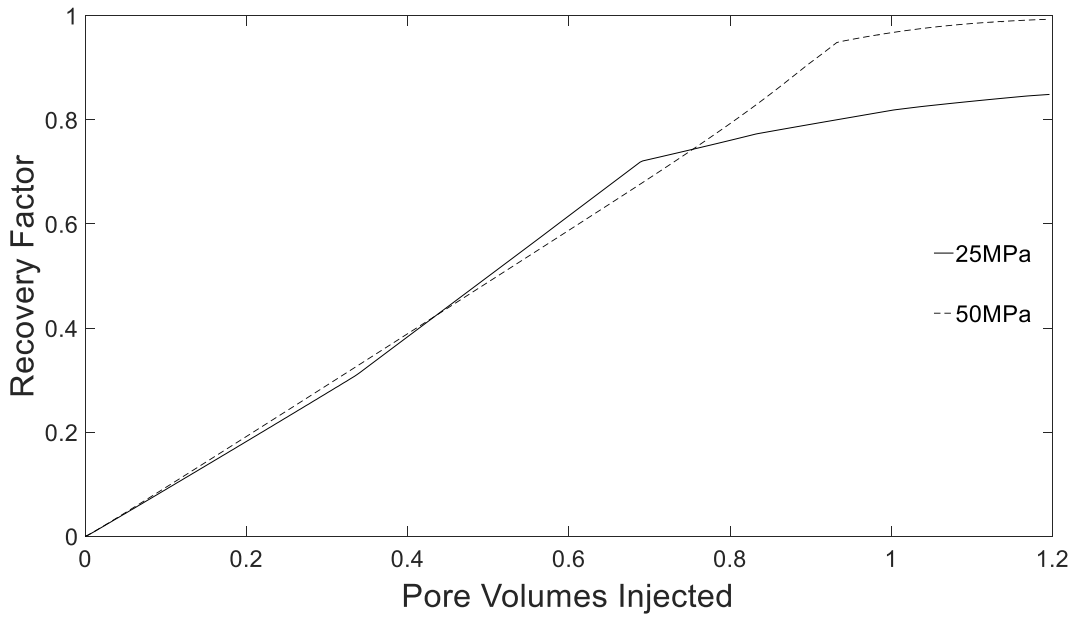


Figure 4.7 – Recovery Profile Gas 3 (C_1 = 0.7, Intermediates = 0.3)

The following plots show a comparison of an immiscible injection vs a miscible injection for each of the three injection gases. The methane and plus fraction components are also shown to illustrate what is happening in the slim tube. In each case 0.3 pore volumes of injection gas was injected into the slim tube in order to display the differences between the miscible and immiscible injection. In the legend; T represents the simulation time, with $T0$ representing the state of the value initially before the slim tube displacement. For each plot the immiscible injection is shown on the left and the miscible injection is shown on the right.

Gas 1 had the highest concentration of methane, and as expected it also had the highest MMP of 67.7 MPa. The plots in Figure 4.8 - Figure 4.10 show an injection at 45 MPa vs 70 MPa. At 45 MPa the injection is immiscible, and the methane is shown bypassing the oil. At 70 MPa, the miscible displacement, the methane moves along in an even front.

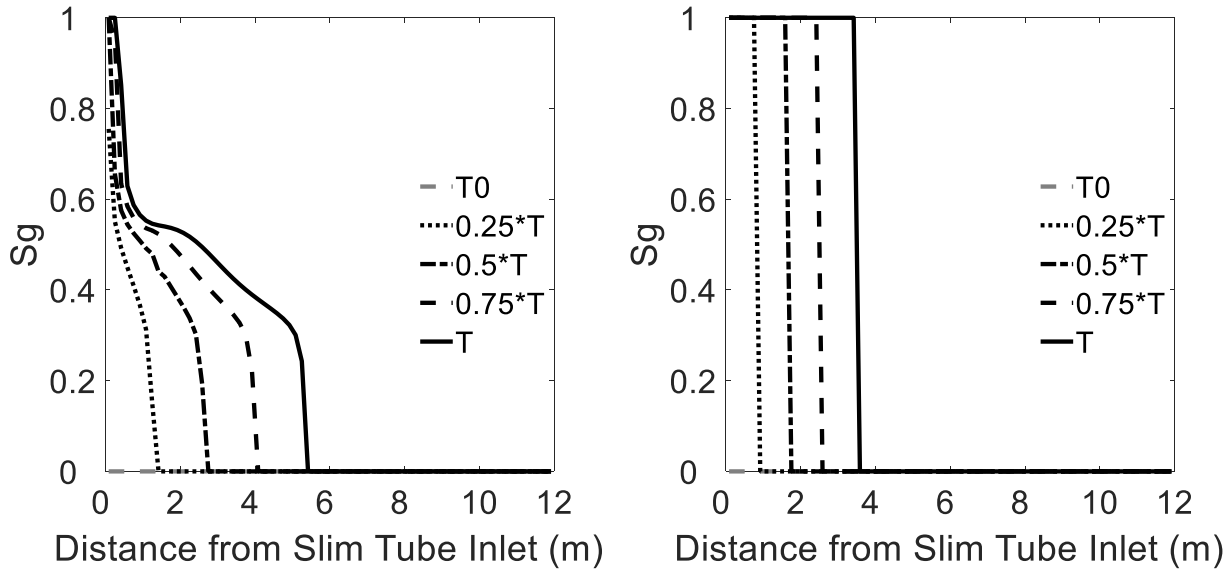


Figure 4.8 – Gas Saturation in Slim Tube Displaced by Gas 1 (C_1 = 0.9, Intm. = 0.1)

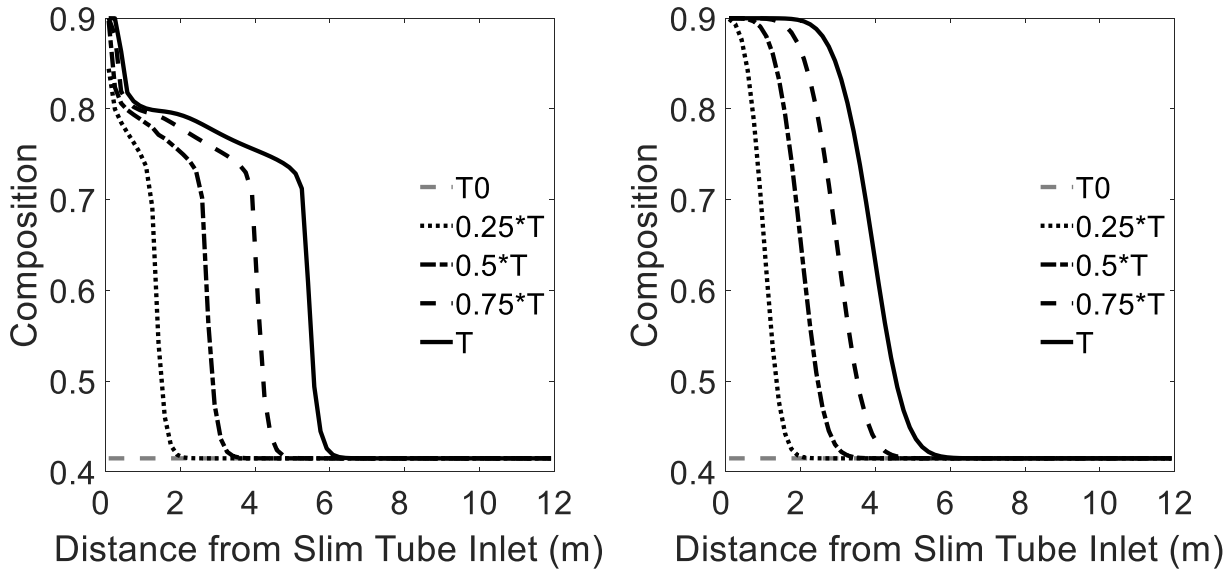


Figure 4.9 – C_1 Composition in Slim Tube Displaced by Gas 1 (C_1 = 0.9, Intm. = 0.1)

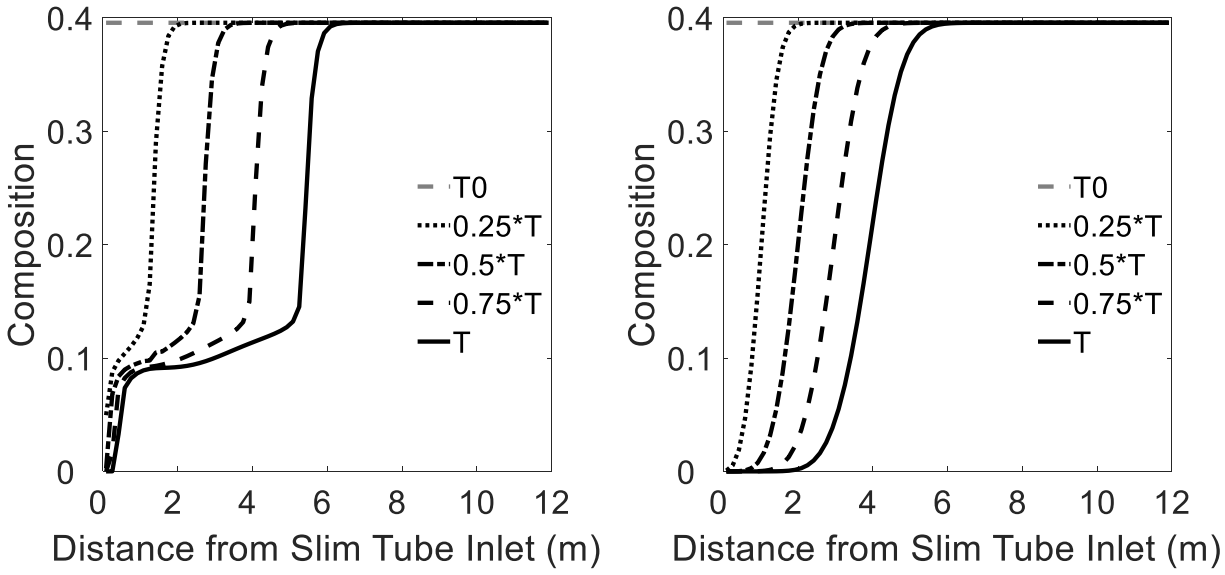


Figure 4.10 – C_{7+} Composition in Slim Tube Displaced by Gas 1 (C_1 = 0.9, Intm. = 0.1)

Gas 2 was the first level of enrichment, which lowered the MMP to 54.2 MPa. The plots in Figure 4.11 to Figure 4.13 show an injection at 35 MPa vs 60 MPa. At 35 MPa the injection is immiscible, and the methane is shown bypassing the oil even more so than in the immiscible injection for Gas 1. At 60 MPa, the miscible displacement, the methane injection travels through the slim tube once again at an even front.

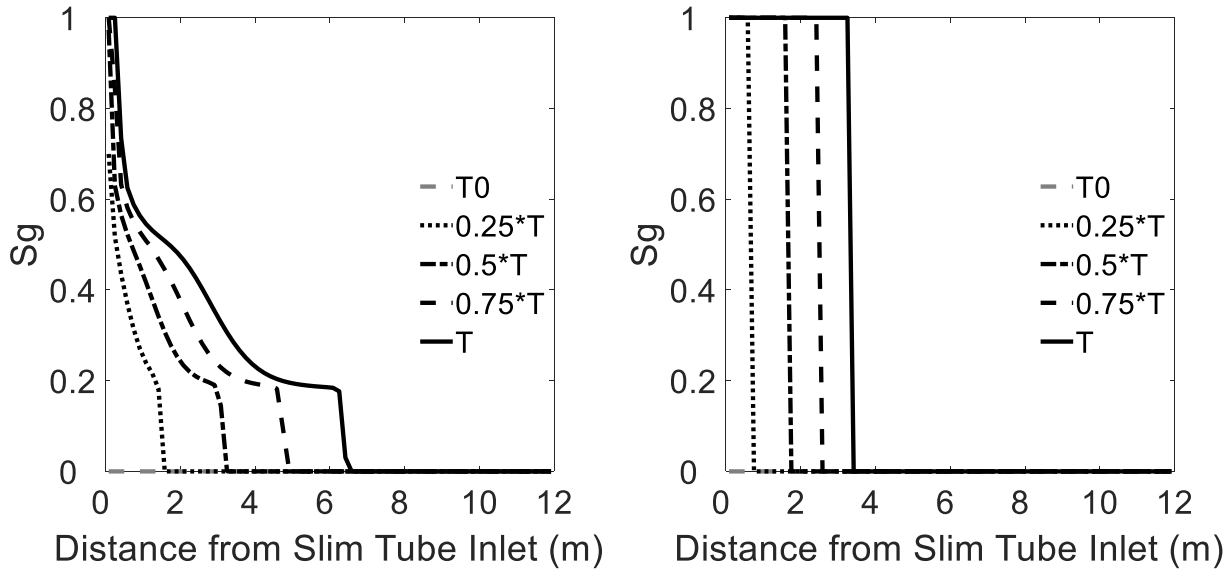


Figure 4.11 – Gas Saturation in Slim Tube Displaced by Gas 2 (C_1 = 0.8, Intm. = 0.2)

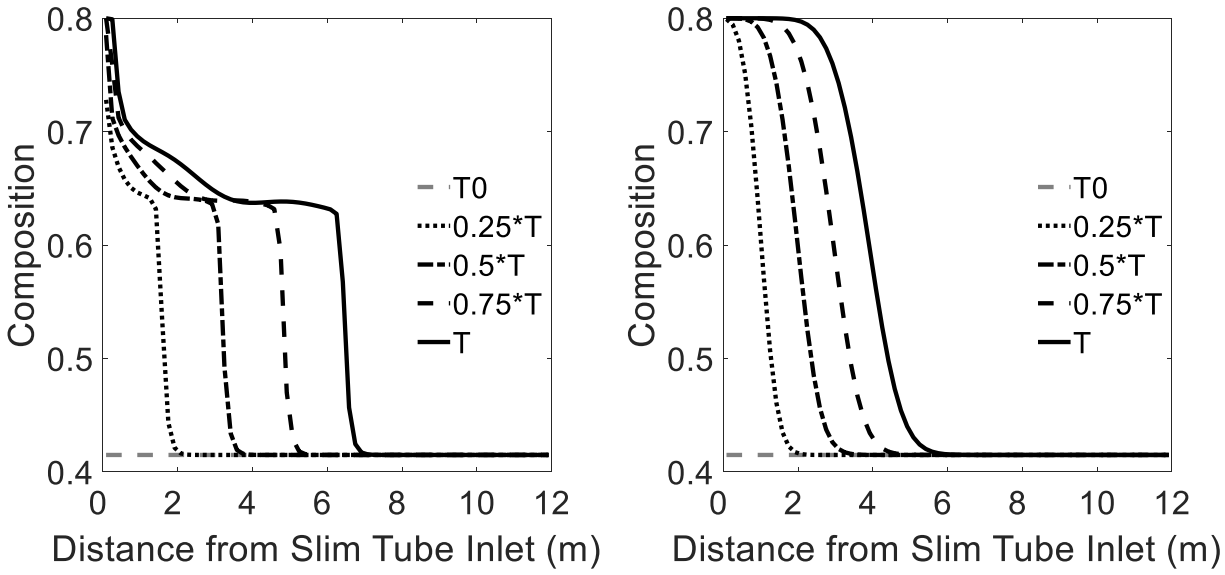


Figure 4.12 – C_1 Composition in Slim Tube Displaced by Gas 2 (C_1 = 0.8, Intm. = 0.2)

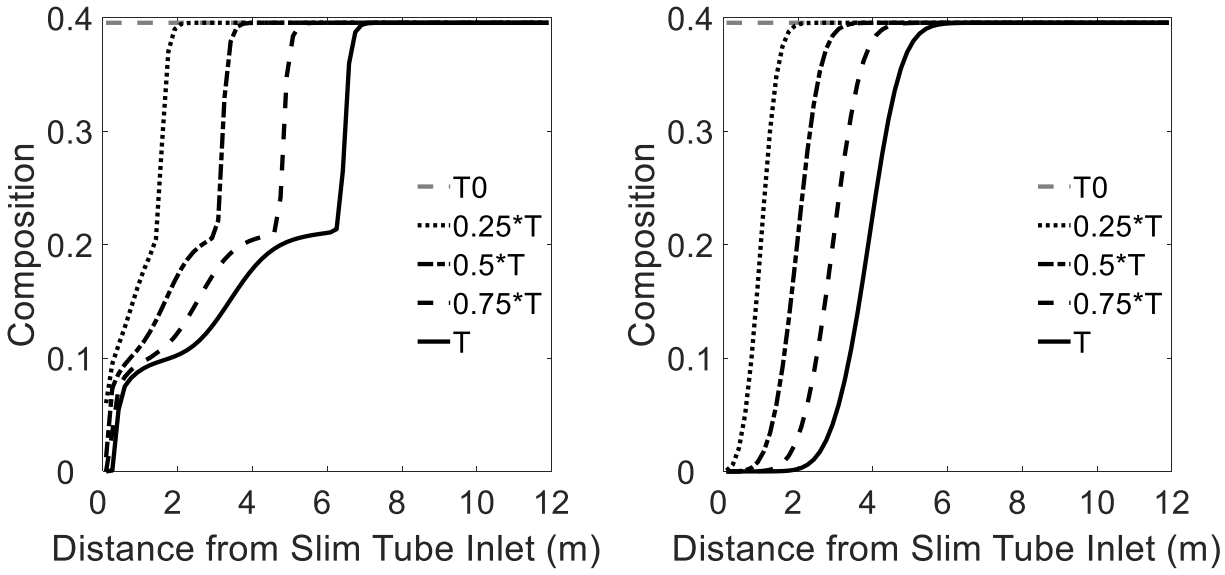


Figure 4.13 – C_{7+} Composition in Slim Tube Displaced by Gas 2 (C_1 = 0.8, Intm. = 0.2)

Gas 3 was the second level of enrichment, which lowered the MMP even further to 43.0 MPa. The plots in Figure 4.14 to Figure 4.16 show an injection at 25 MPa vs 50 MPa. At 25 MPa the injection is immiscible, and the methane is shown bypassing the oil even more so than in the immiscible injection for the other two gases. At 50 MPa, the miscible displacement, the methane once again moves along in an even front..

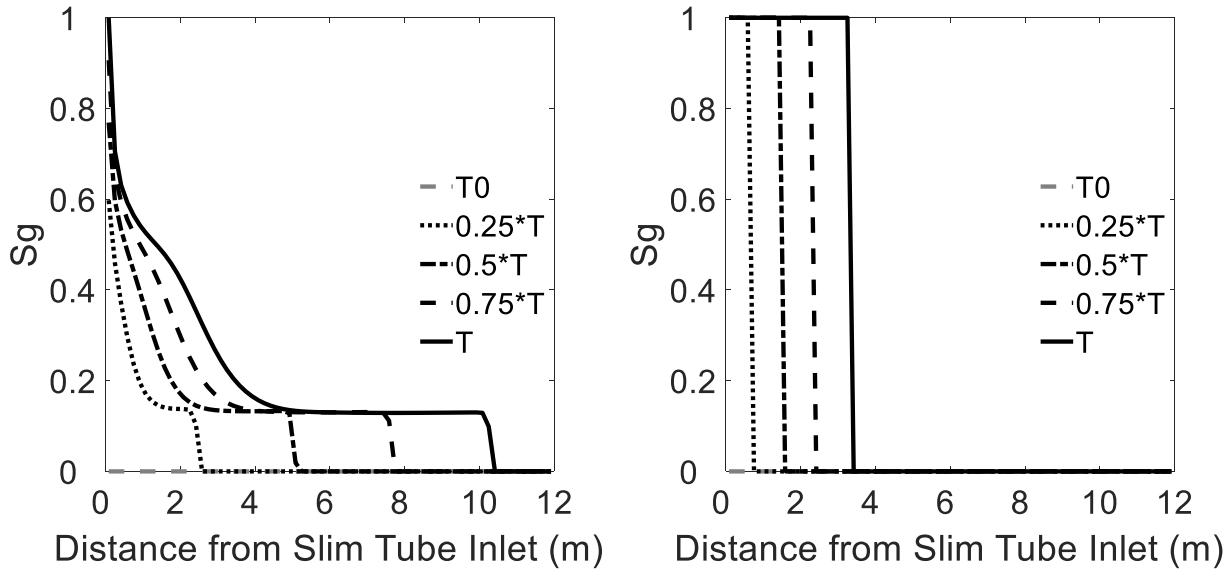


Figure 4.14 – Gas Saturation in Slim Tube Displaced by Gas 3 (C_1 = 0.7, Intm. = 0.3)

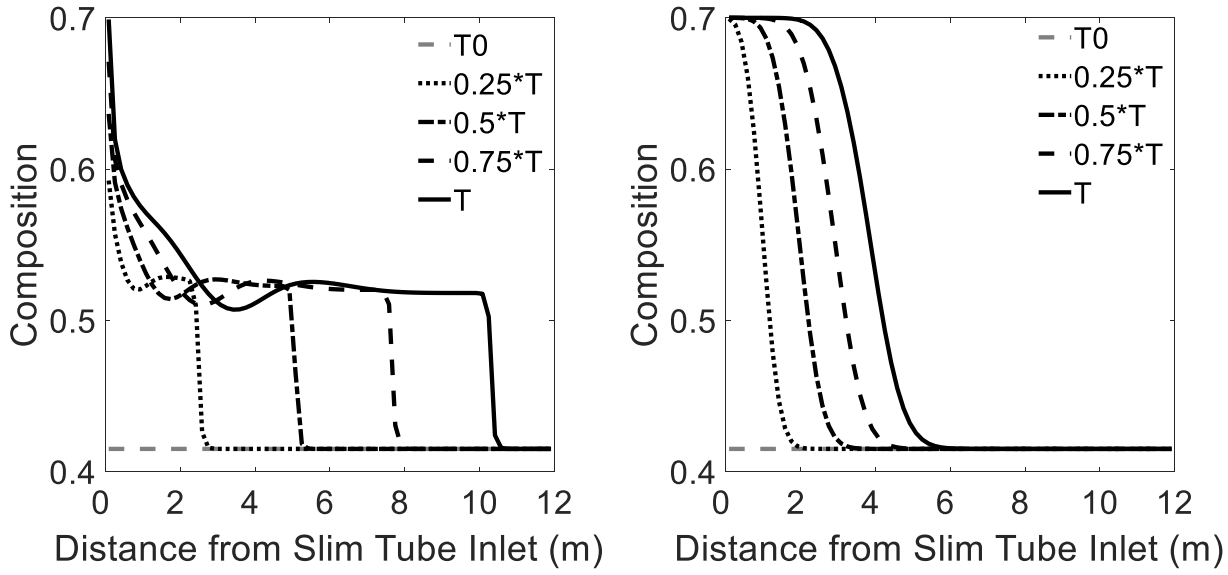


Figure 4.15 – C_1 Composition in Slim Tube Displaced by Gas 3 (C_1 = 0.7, Intm. = 0.3)

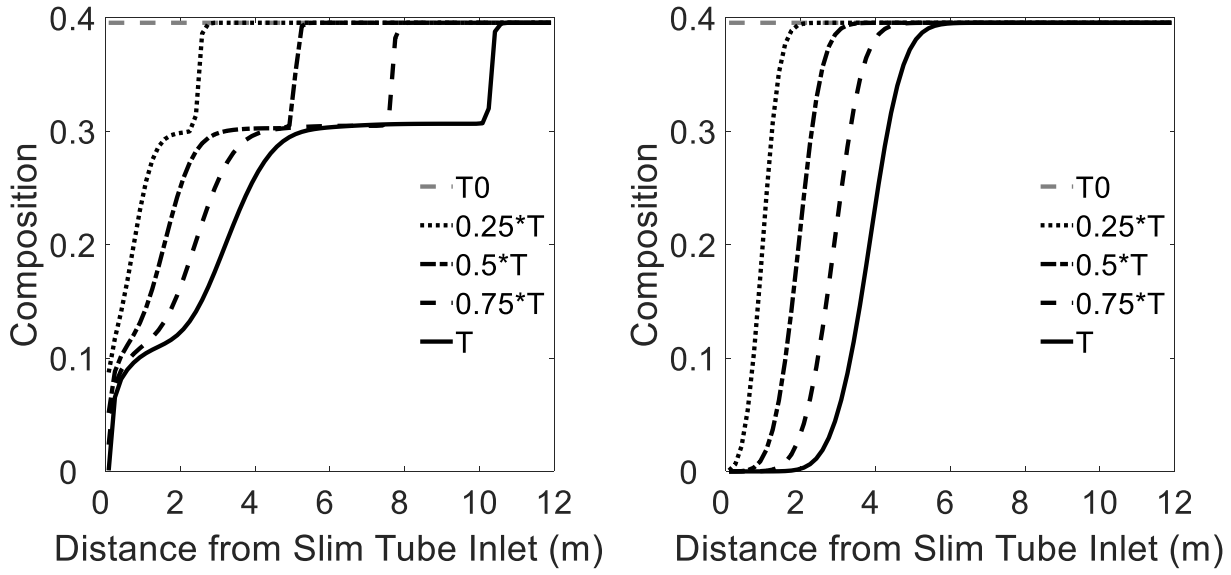


Figure 4.16 – C_{7+} Composition in Slim Tube Displaced by Gas 3 (C_1 = 0.7, Intm. = 0.3)

Overall, the results of these slim tube simulation experiments determined the MMP for each of the three injection gases to be studied for the huff 'n' puff, and helped to show that the developed simulator is able to model multi-component gas injection into multi-component reservoir oil in one dimension.

4.2 Case Study Results

The reservoir properties described in Table 4.1 were used for the huff 'n' puff case study simulations. Gas 1, Gas 2, and Gas 3 as described in Table 4.4 - Table 4.6 respectively were used as injection gases for the simulation. The simulation used an initial water saturation throughout the reservoir of 75%, in order to simulate a watered-out reservoir after a long period of water injection. As mentioned in Chapter 2, there are three phases to the huff 'n' puff process: huff (injection), shut-in, and puff (production). All of this happens in the same well, therefore since the case study was performed on a one-dimensional reservoir simulator all injection and production

occurred at the same radial point in the grid. The schedule was as follows: 8 hours of constant-rate injection for the huff phase at 0.00184 m^3/s (roughly 1000 bbl/d), followed by two hours of shut-in, and finally 14 hours of production for the puff phase with the wellbore pressure set to 44 MPa. The plots showing the state of the reservoir after each phase are shown in Figure 4.18 – Figure 4.20. The legend for these plots is shown in Figure 4.17; each plot shows the state of the parameter at the beginning of the phase, and then shows the state of the parameter across the time for that phase, with the simulation time for that phase being represented by T and $T0$ representing the beginning of the phase. For each phase and each injection gas, there is a full set of near-well plots showing what is happening near the wellbore, as this is the area of interest. There is also a full-scale pressure plot to show how the pressure of the radial reservoir system is behaving. Example plots with further instruction on how to interpret them are provided in Appendix C.

4.2.1 Huff Phase

The huff phase simulation was successful for each of the injection gases. For Gas 1, which had the least intermediates and the highest MMP, it was an immiscible injection. Figure 4.18(c) - Figure 4.18(e) show the near well saturation plots. The plots show that the hydrocarbons are injected immiscibly in the gas phase, partially bypassing the oil in the near well region. This is as expected from the work completed in the previous section, as the MMP for Gas 1 is higher than the reservoir pressure for the case study. The spike seen in oil saturation is caused by the injected light hydrocarbons reaching into the reservoir, but the light hydrocarbon composition is not high enough to cause a gaseous mixture. The composition plots support the immiscible injection, as the C_1 fraction is seen to be displacing the C_{7+} fraction away from the near wellbore region in an immiscible front. Of the three injection gases, using Gas 1 allowed the C_1 to travel the furthest into the reservoir travelling roughly 60m.

The injection phase of Gas 2 appeared mostly as expected. Gas 2 had been enriched with some intermediate components, and the MMP of Gas 2 with the reservoir oil was found to be roughly 54.2 MPa. This injection was thus still immiscible, although closer to the miscible region. The near well saturation plots in Figure 4.19(c) - Figure 4.19(e) show that the water is displaced immiscibly from the near well region, while the oil and gas saturation plots also show an immiscible injection. The spike in oil saturation at roughly 40m is also caused by the injected light hydrocarbons reaching into the reservoir but the light hydrocarbon composition is not yet high enough to form a gaseous mixture. At a point in the reservoir around 18m from the wellbore, it can be seen in Figure 4.19(d) and Figure 4.19(e) that the hydrocarbon mixture is only in the single oil phase, as opposed to the two-phase region like most of the surrounding reservoir. This can be explained by examining the composition plots in Figure 4.19(l) - Figure 4.19(v). At this point (~18m) in the reservoir, there is a slight decrease in C_1 composition, at the same point there is a slight increase in C_{7+} composition. This is coupled with a peak in composition of some of the heavier injected intermediaries (C_3 and C_4). It is these composition levels that cause the small one phase region in the reservoir. Where this injection was closer to the injection gas MMP than what was done with Gas 1, the injection process was more miscible and hence the injected C_1 did not travel as far as in the first injection. For Gas 2, the injected C_1 travelled roughly 50m into the reservoir.

Gas 3 had the highest level of intermediate components of the three injection gases. The MMP of Gas 3 and the reservoir oil was found to be roughly 43.0 MPa. The saturation profiles in Figure 4.20(c) - Figure 4.20(e) show a miscible injection front. The gas is injected in an even front displacing the water and oil from the near well region. The familiar spike in oil saturation is caused by the same effect of the previous two injection gases, and that is the light injected hydrocarbons travelling into regions where the light hydrocarbon composition is not yet high enough to cause

the phase to change to gas. The composition plots in Figure 4.20(l) - Figure 4.20(v) clearly show an even miscible injection, and in this injection the C_1 penetrates the least into the reservoir, as it travels roughly 45m deep.

For Gas 1, Gas 2, and Gas 3, the near well pressure plots can be seen in Figure 4.18(a), Figure 4.19(a) and Figure 4.20(a) respectively. The full-scale pressure plots are seen in Figure 4.18(b), Figure 4.19(b) and Figure 4.20(b) respectively. For the huff phase for each of the injection gases, the full-scale pressure plots show a radial pressure profile, with a very small pressure gradient towards the reservoir boundary which validates the assumption of a constant pressure boundary at the drainage radius. The near-well pressure plots all show that the pressure gradient in the near well region is affected by the gas injection, as gas is injected the oil pressure decreases. The pressure at the wellbore is decreasing due to this gas injection, it is a constant-rate injection as opposed to a constant-pressure injection. This is an expected behaviour, and is seen in all three injection gases. The oil viscosity plots for the three injection gases are shown in Figure 4.18(j), Figure 4.19(j) and Figure 4.20(j) respectively. In each of these plots, the oil viscosity reduces as the injection gas enters the reservoir. The reduction in oil viscosity follows the gas saturation roughly, therefore for Gas 1 there is a less even immiscible front, Gas 3 has an even front, and Gas 2 is in-between. The reduction of oil viscosity is due to the injection of lighter hydrocarbon components into the reservoir oil, this is an expected behaviour. After the huff phase, in each of the three injection gases it can be seen that the near well region has a lower oil viscosity where the injection gas has reached, and further from the wellbore the oil viscosity increases. As discussed in Chapter 2, oil viscosity reduction is one of the primary recovery mechanisms of the huff ‘n’ puff process.

In general, the hydrocarbon composition plots across each of the three injection gases show the three levels of miscibility. For each of the injection gases C_1 is the main component. By examining the C_1 composition plots, it is easy to see that for Gas 1 there is an immiscible uneven front, while for Gas 3 there is an even miscible front moving through the reservoir. The compositions throughout each reservoir follow a similar trend, where the only place in the reservoir a composition change occurs is due to the gas injection. The differences in the composition plots lie in the differences in injection gas composition, and the difference in miscibility of each injection.

Overall, the injection phase for the huff 'n' puff process behaved similar to what was seen in the previous section for the slim tube displacements. The displacements did not appear as smooth and the difference between different levels of miscibility was not as obvious; this could be attributed to the fact that the model for the huff 'n' puff process is in three phases with a high initial water saturation.

4.2.2 Shut-In Phase

The shut-in phase displayed roughly the same behaviour for each of the injection gases. The injection gas did not travel much further into the reservoir after shut-in as seen in Figure 4.18(d), Figure 4.19(d), and Figure 4.20(d). The full-scale pressure distribution plots are shown in Figure 4.18(b), Figure 4.19(b), and Figure 4.20(b). It can be seen that the system behaves as expected during shut-in, where the pressure levels out across the system after being elevated during the injection phase.

4.2.3 Puff Phase

It was noted that during the puff phase that for each injection gas, the simulator appeared to be unable to successfully model the huff 'n' puff process. For Gas 1, the gas saturation plot in Figure

4.18(e) shows that the gas is being produced at the wellbore, until the remaining gas saturation in around the wellbore is reduced to the critical gas saturation. Looking at the corresponding composition plots in Figure 4.18(l) - Figure 4.18(v), the C_1 fraction is seen to reduce as expected and then level off when the critical gas saturation has been reached. The limitations of the simulator are seen in the water and oil saturation plots in Figure 4.18(c) and Figure 4.18(d) respectively. The water saturation is elevated to $1\text{-}S_{orw}$, and due to the water saturation reaching this level the oil flow into the wellbore is essentially shut off as seen in the oil relative permeability plot in Figure 4.18(g). The same effect happens to the other two injection gases, Gas 2 and Gas 3. In both these cases the gas is produced until the gas saturation is reduced to critical gas saturation, and the oil saturation is reduced to S_{orw} which shuts off oil production to the wellbore.

Due to this effect, the injection with different gases cannot be properly compared, as the effective oil flow shut-off stops production. This oil flow shut-off is what would be physically expected when the oil saturation reaches residual oil, so this is not necessarily an error in the simulator but may be showing the limitations of this simulator when attempting to model the three-phase huff 'n' puff process in one dimension. Limitations of the simulator and how they may affect simulating the huff 'n' puff process are discussed in the proceeding section.

4.2.4 Discussion of Limitations

In general, during the simulation of the huff 'n' puff process all trends in reservoir property changes are what would be expected due to physical phenomenon. The work completed in section 3.5 provided many validations on the reservoir property models as well as the reservoir flow itself. The reservoir behaviour noted in the slim tube simulations, as well as the huff and shut-in phases

of the huff ‘n’ puff process were as expected. It is only during the puff phase while modelling the huff n’ puff process that the reservoir simulator seemed unable to successfully model the process. After a thorough evaluation of the model, the current hypothesis is that the huff ‘n’ puff process of a watered-out reservoir cannot be accurately modelled in the developed one-dimensional simulator. All other validations and tests of the model have shown that the model appears to accurately model reservoir behaviour, therefore the assumption is that the simulator is accurately modelling reservoir behaviour in the puff phase, but due to the simulator only having one dimension it cannot model the desired process. In true applications of the huff ‘n’ puff process there are complex flow patterns in multiple dimensions, where in a one dimensional model the flow patterns of the three phases are of course limited to the singular dimension which is likely what is causing the observed oil flow shut-off. It is likely that an extension of this model into multiple dimensions would allow for an accurate simulation of the process.

Other assumptions may have also limited the simulator from successfully modelling the huff ‘n’ puff process. The water properties were only modelled to be affected by pressure in this isothermal constant salinity model. The effect of gas injection into the water was not taken into account, which could alter the results. The assumption was also made that capillary pressures were negligible, in order to simplify the model. The effect of capillary pressures could play a role in the modelling of the huff ‘n’ puff process which may also need to be considered.

Although the simulation limitations did not allow for proper comparison of different injection parameters for the huff ‘n’ puff process, the work done to create this model was very substantial and also creates great opportunity for future work in this field, as will be mentioned in the proceeding chapter.

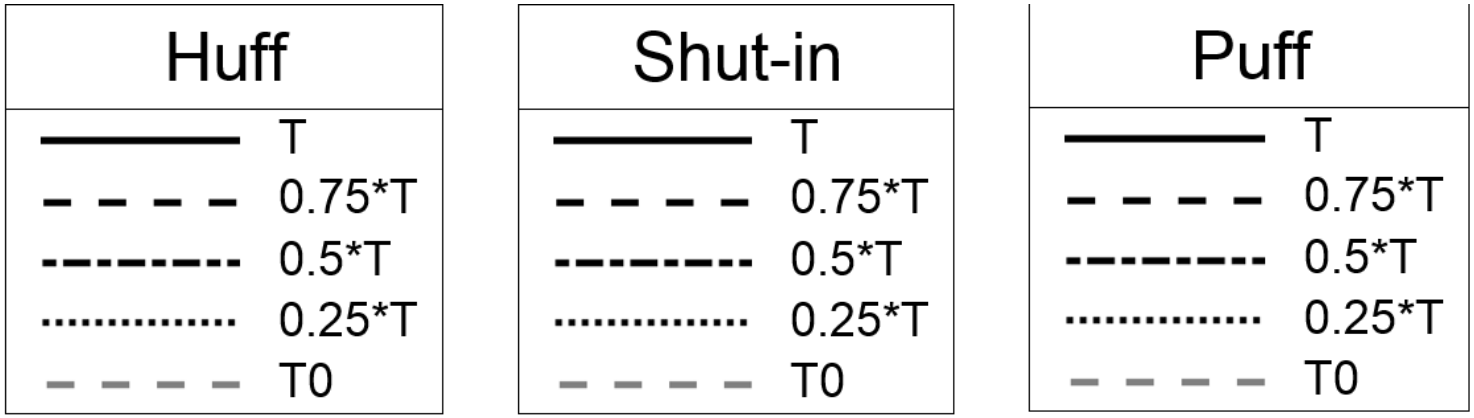


Figure 4.17 – Legend for Huff 'n' Puff plots

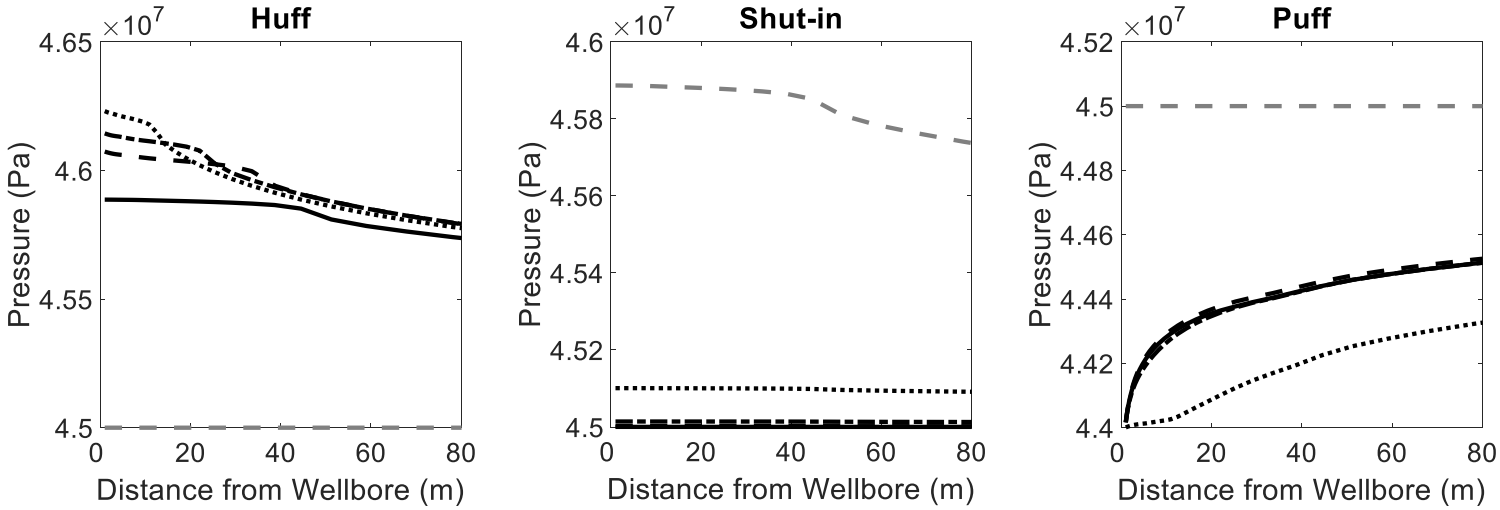


Figure 4.18(a) – Near Well Pressure Distribution for Gas 1

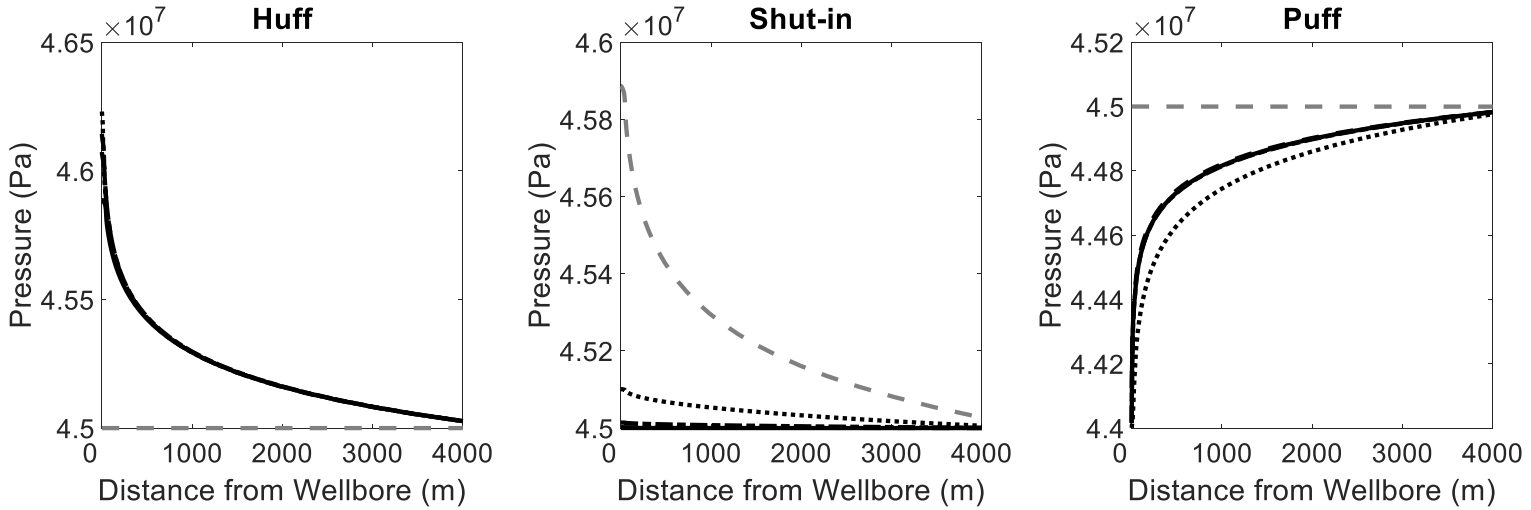


Figure 4.18(b) – Full Scale Pressure Distribution for Gas 1

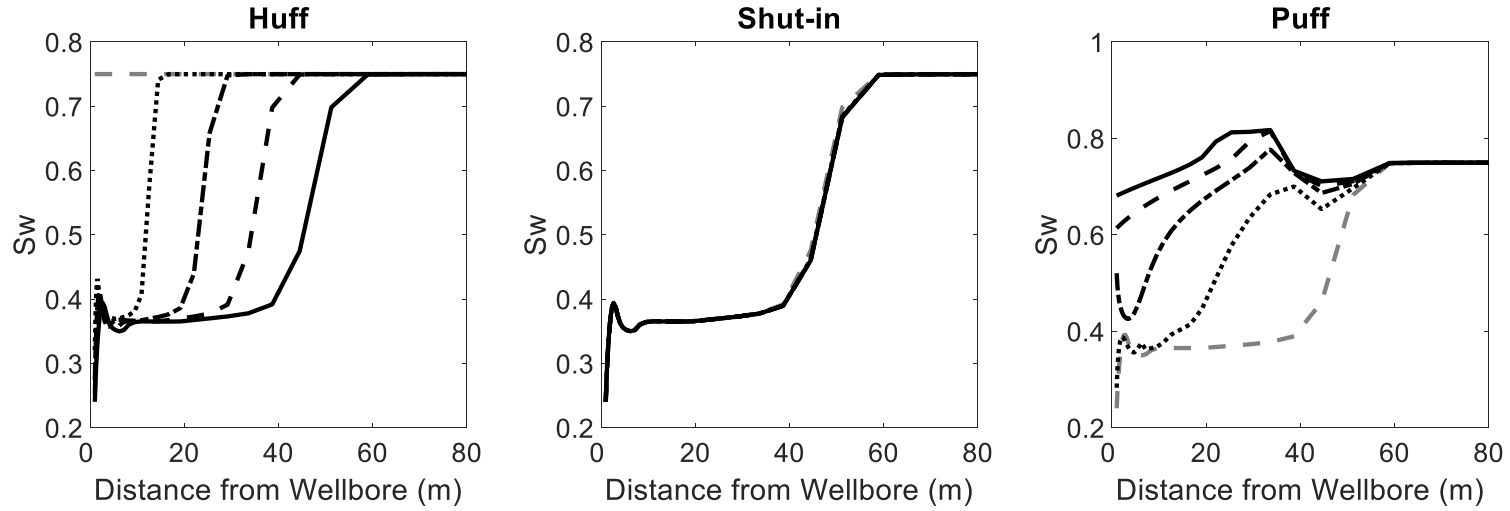


Figure 4.18(c) – Near Well Water Saturation Profile for Gas 1

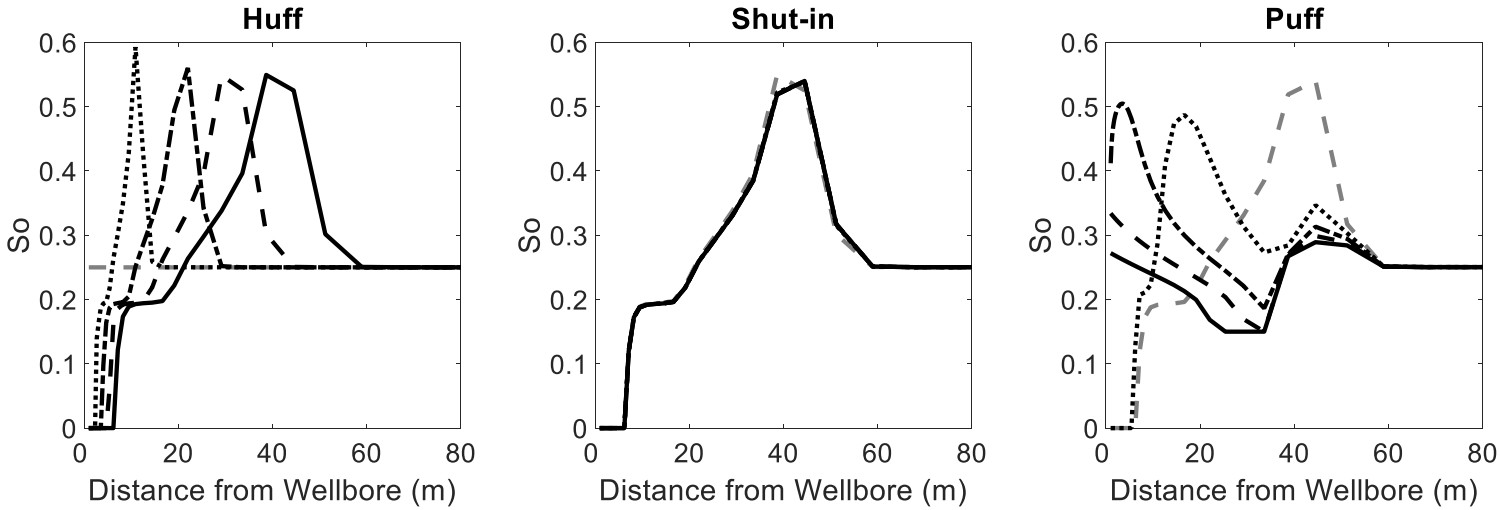


Figure 4.18(d) – Near Well Oil Saturation Profile for Gas 1

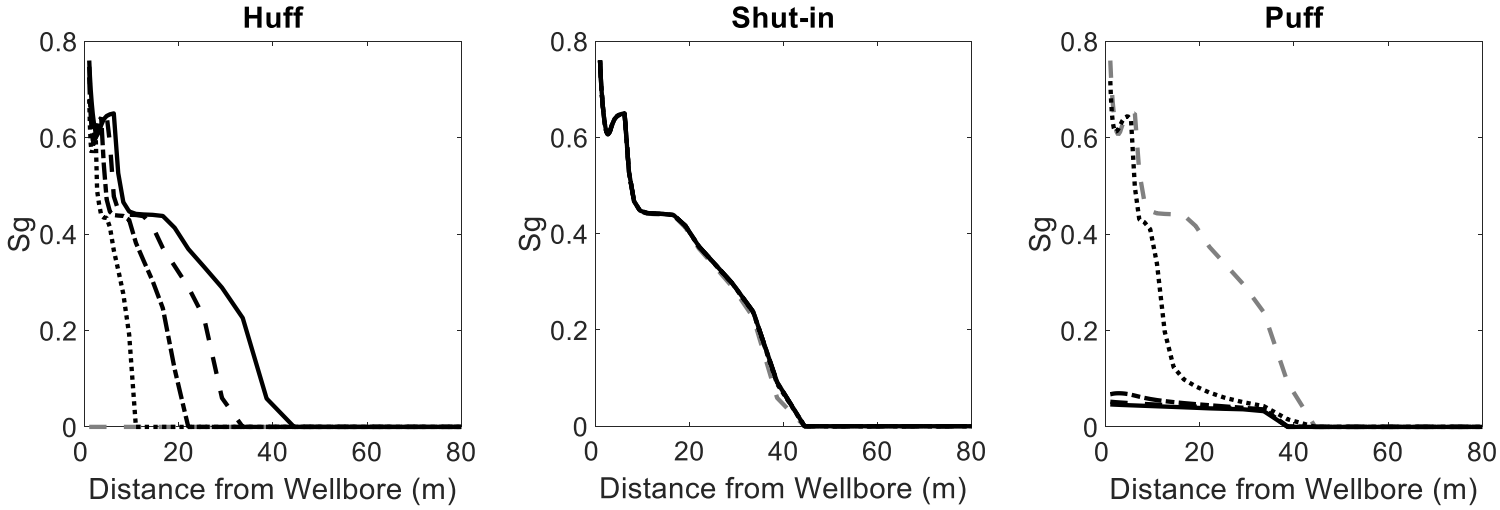


Figure 4.18(e) – Near Well Gas Saturation Profile for Gas 1

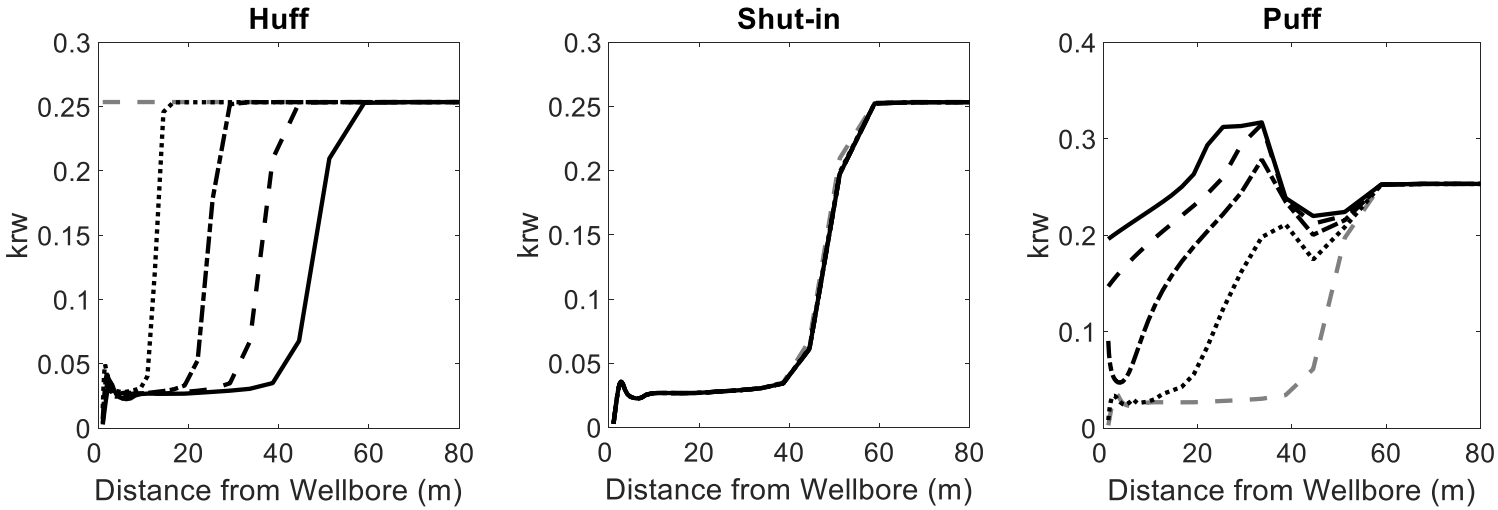


Figure 4.18(f) – Near Well Water Relative Permeability Profile for Gas 1

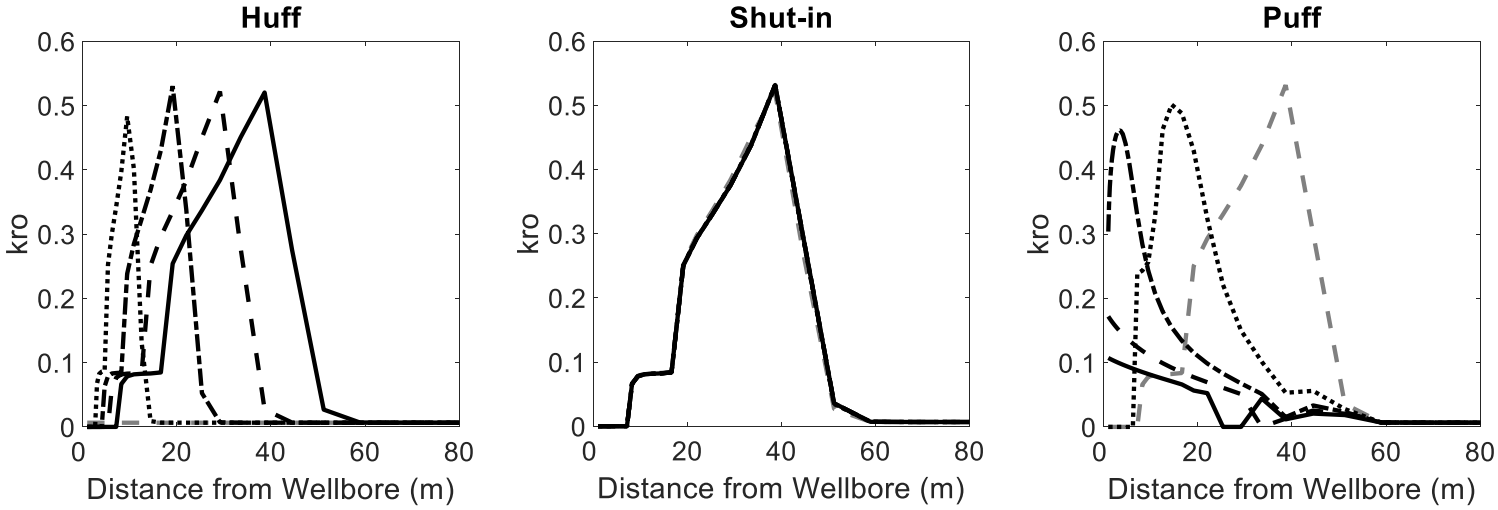


Figure 4.18(g) – Near Well Oil Relative Permeability Profile for Gas 1

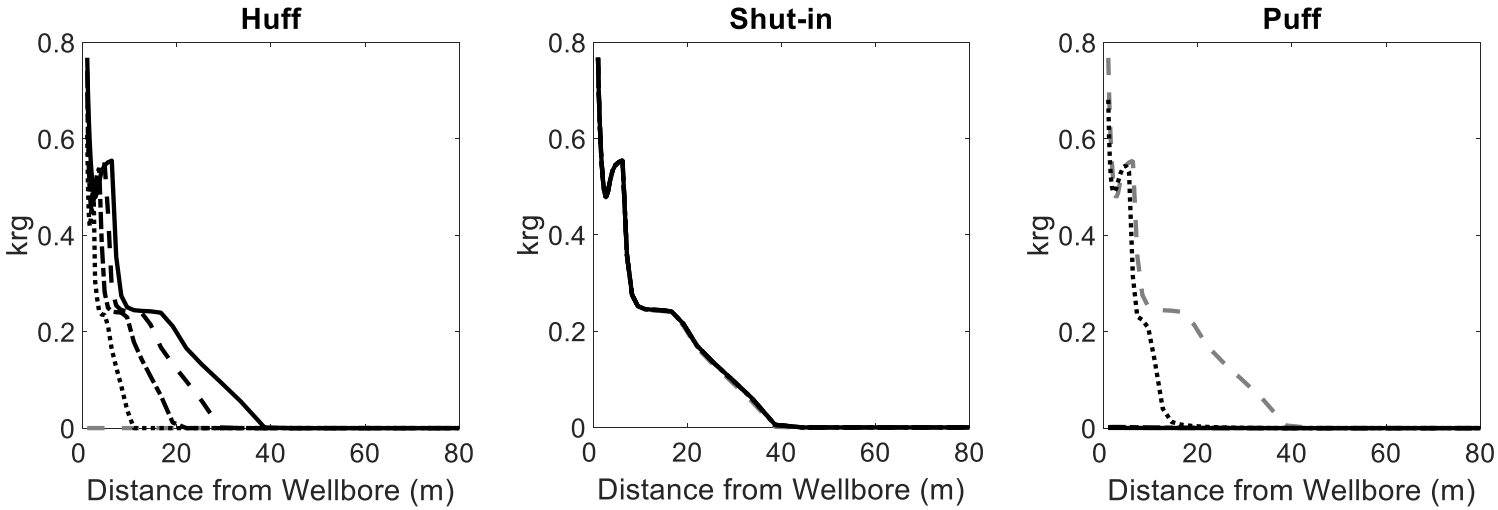


Figure 4.18(h) – Near Well Gas Relative Permeability Profile for Gas 1

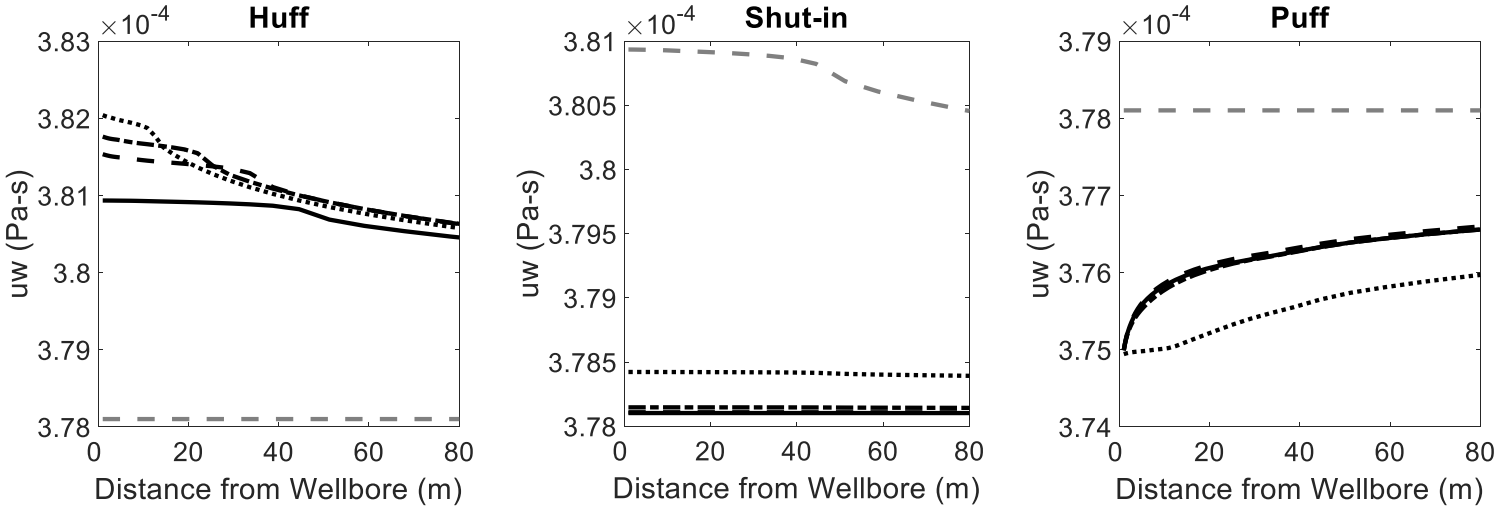


Figure 4.18(i) – Near Well Water Viscosity Profile for Gas 1

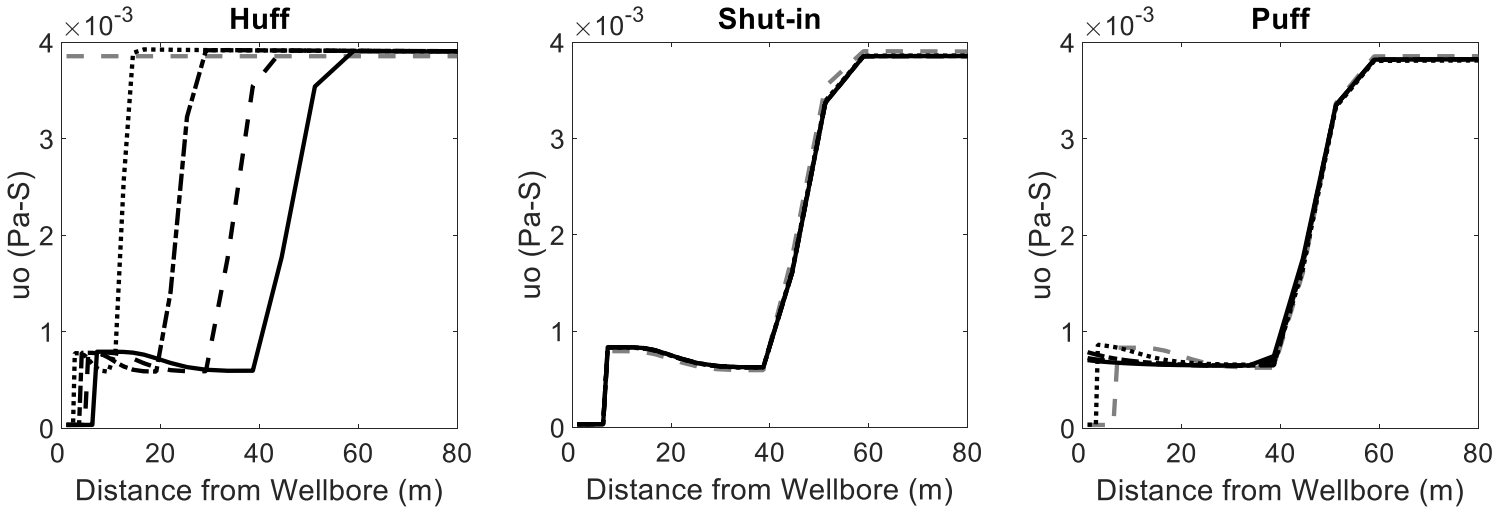


Figure 4.18(j) – Near Well Oil Viscosity Profile for Gas 1

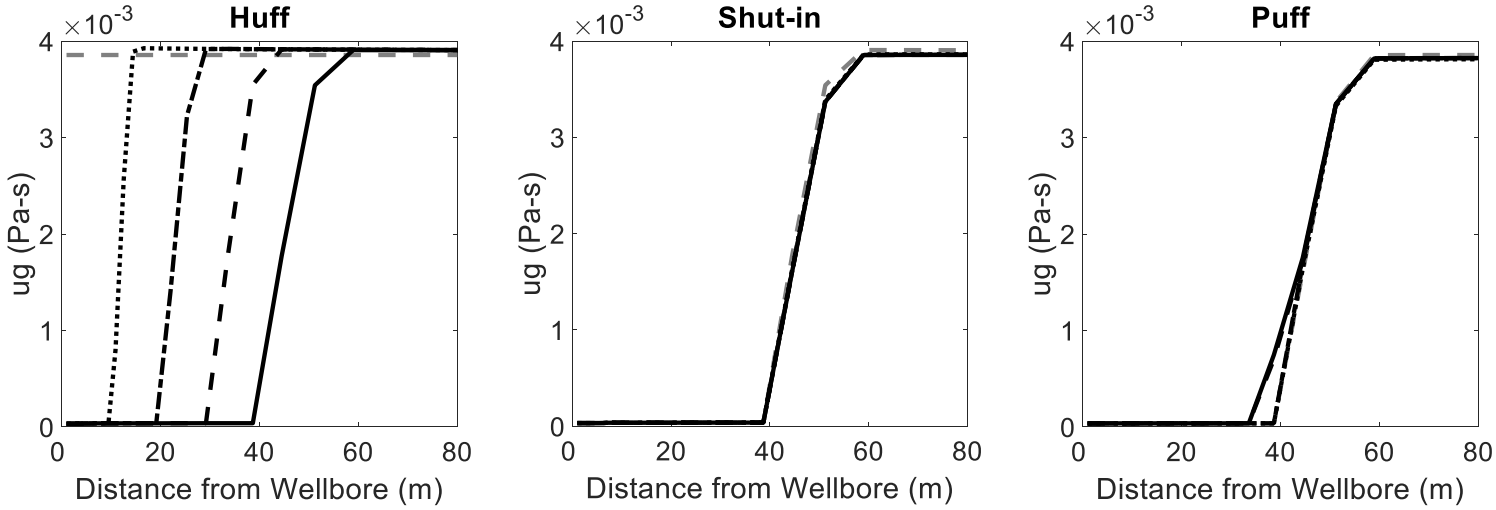


Figure 4.18(k) – Near Well Gas Viscosity Profile for Gas 1

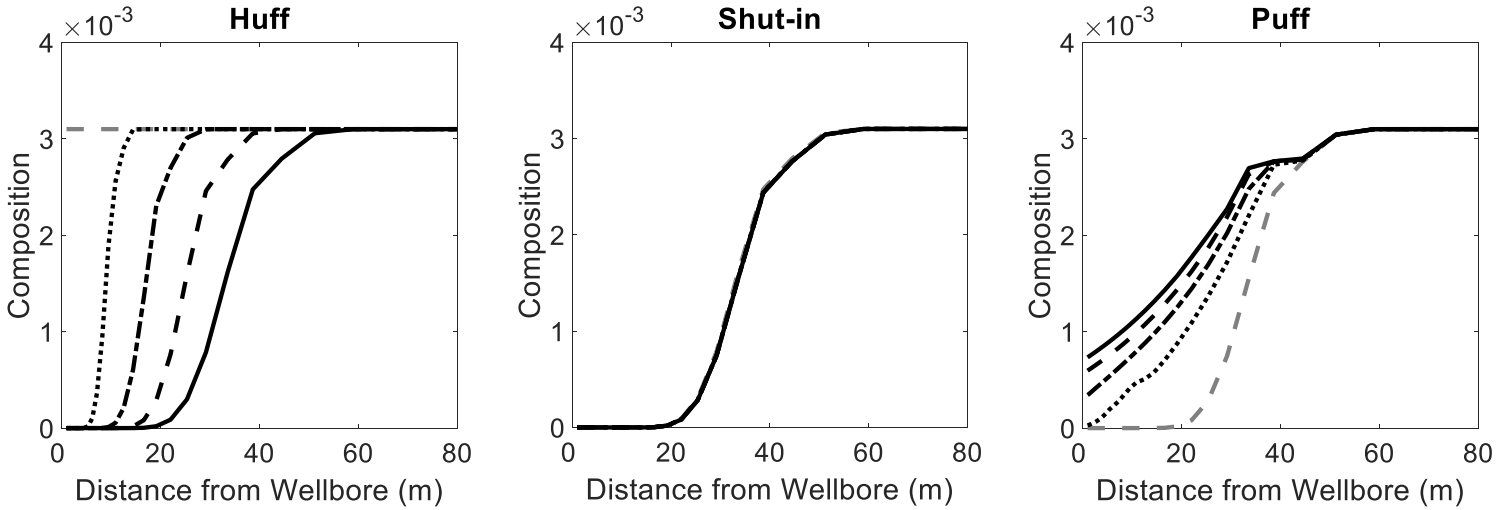


Figure 4.18(l) – Near Well N_2 Composition Profile for Gas 1

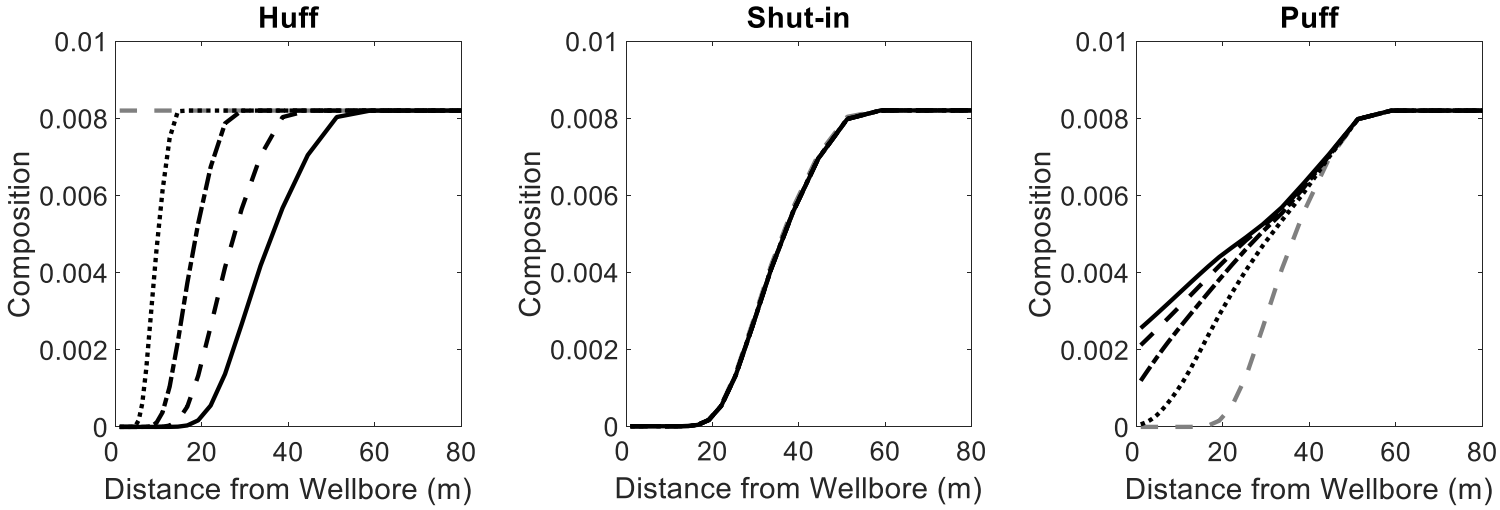


Figure 4.18(m) – Near Well CO_2 Composition Profile for Gas 1

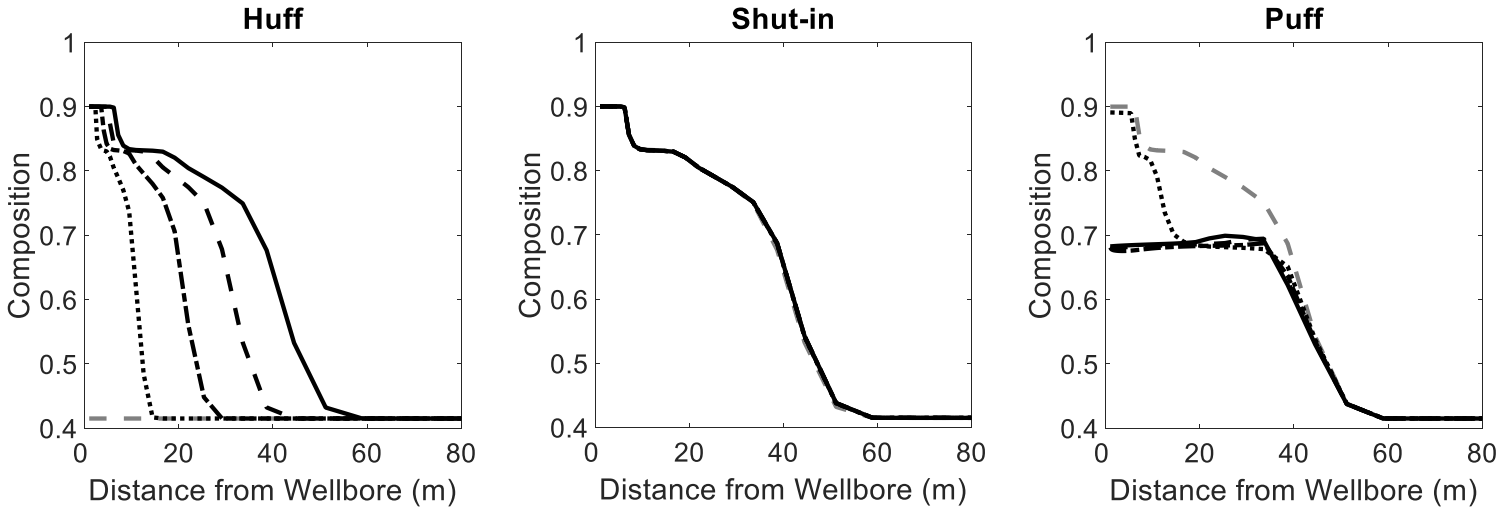


Figure 4.18(n) – Near Well C_1 Composition Profile for Gas 1

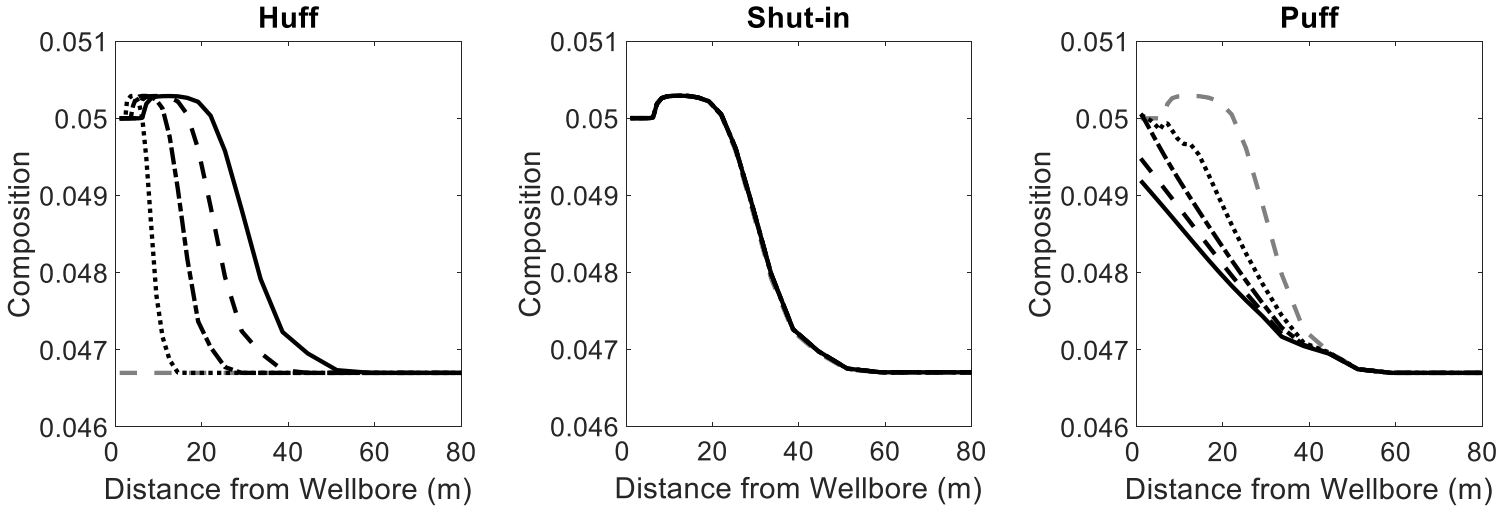


Figure 4.18(o) – Near Well C_2 Composition Profile for Gas 1

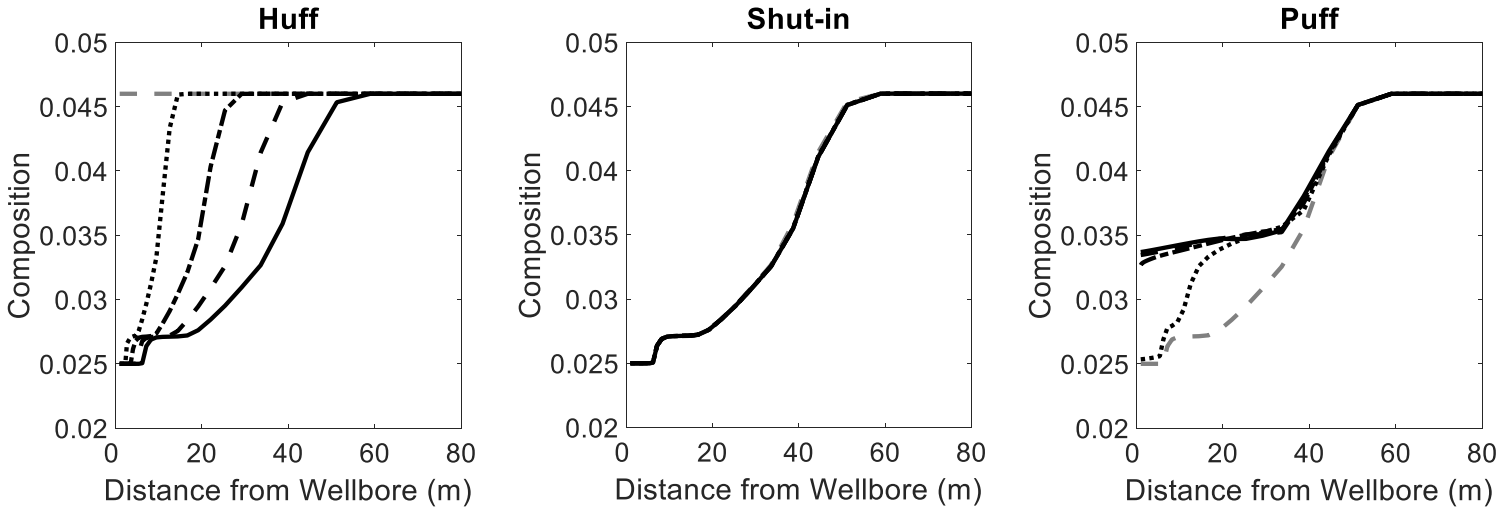


Figure 4.18(p) – Near Well C_3 Composition Profile for Gas 1

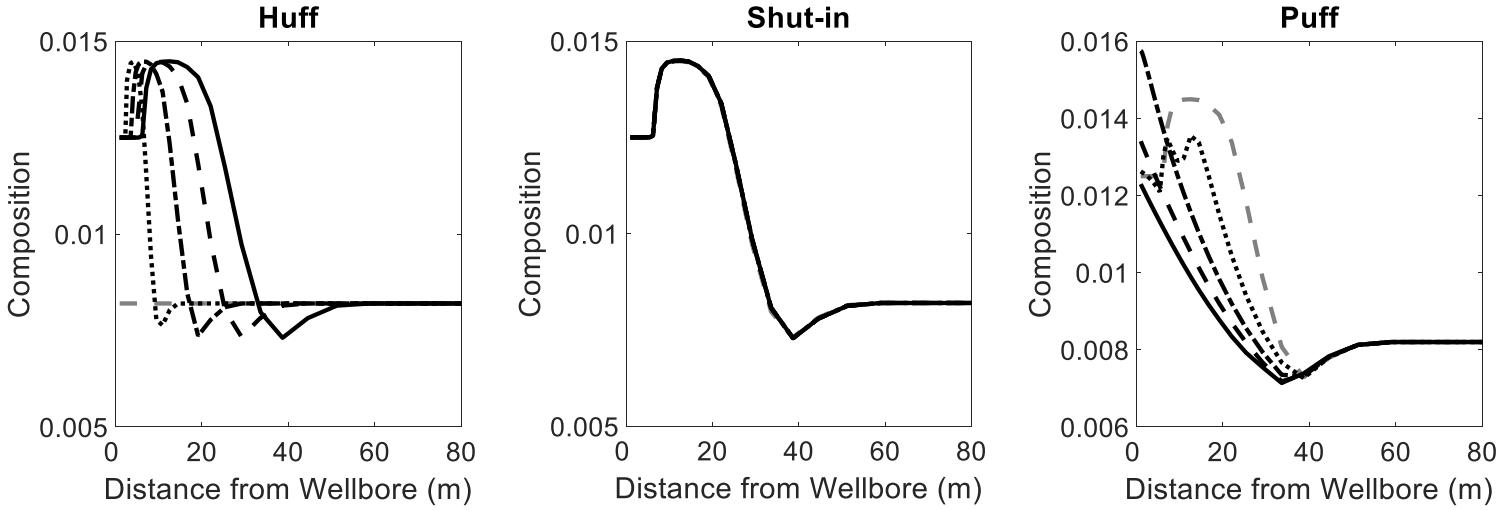


Figure 4.18(q) – Near Well iC_4 Composition Profile for Gas 1

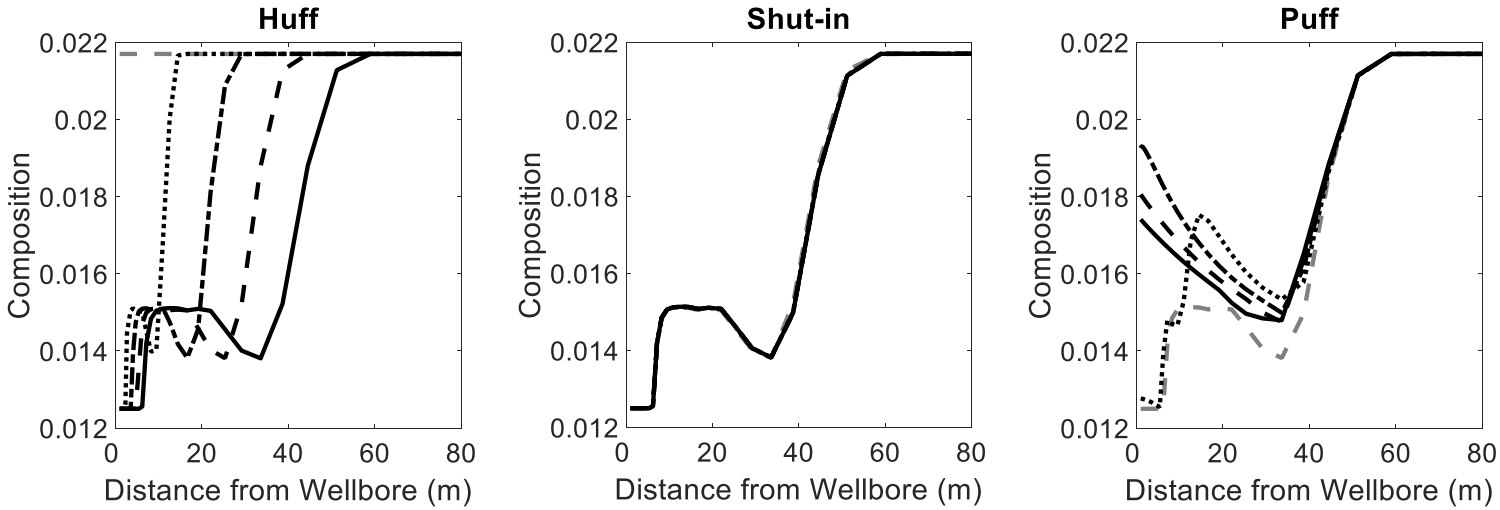


Figure 4.18(r) – Near Well nC_4 Composition Profile for Gas 1

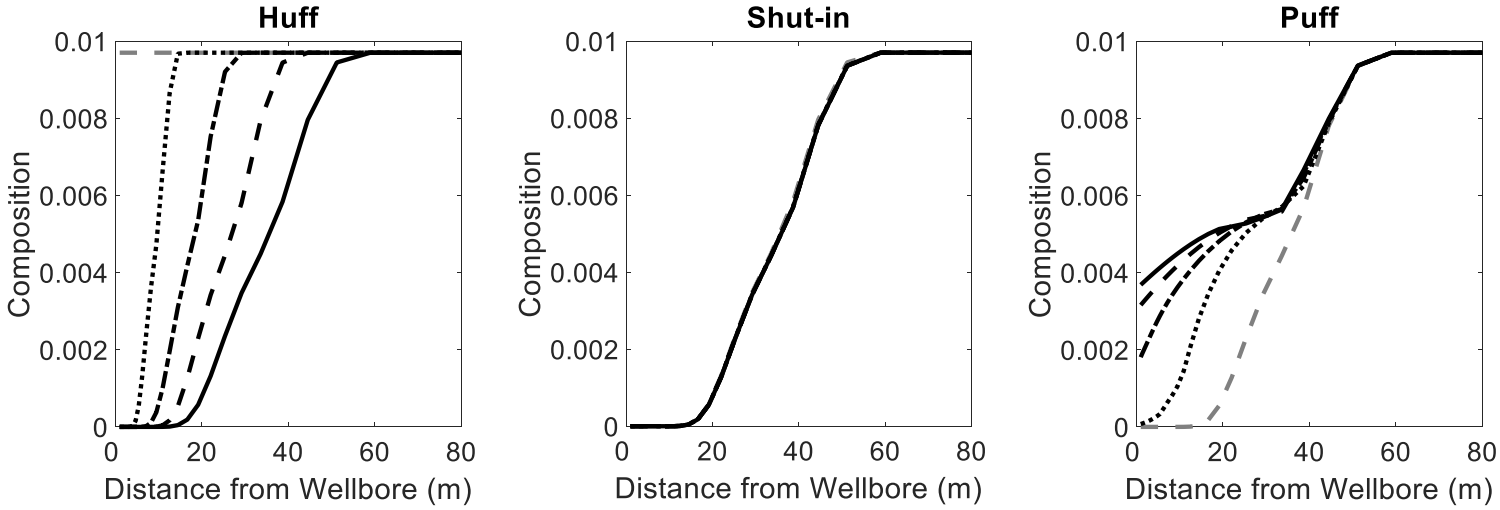


Figure 4.18(s) – Near Well iC_5 Composition Profile for Gas 1

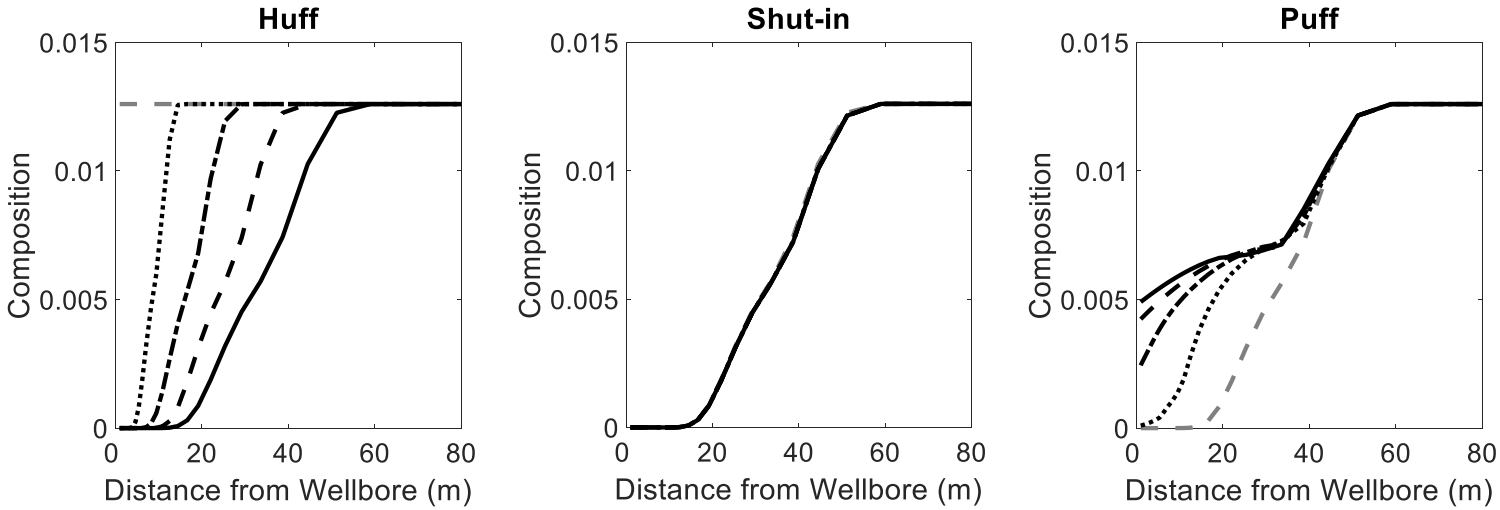


Figure 4.18(t) – Near Well nC_5 Composition Profile for Gas 1

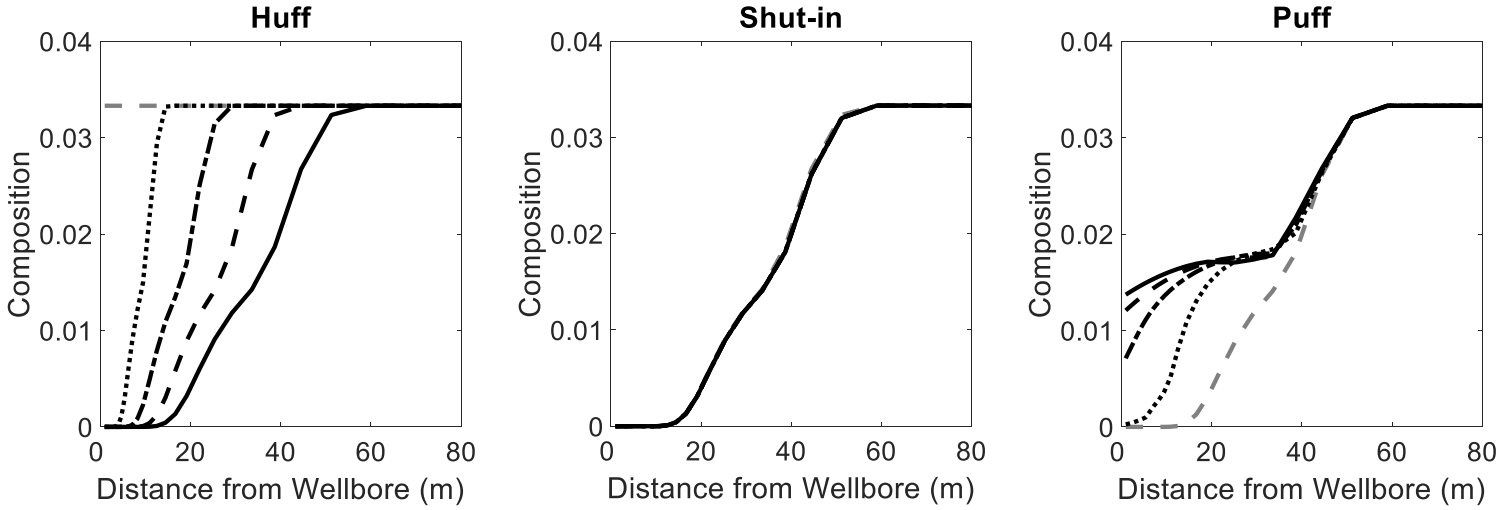


Figure 4.18(u) – Near Well nC_6 Composition Profile for Gas 1

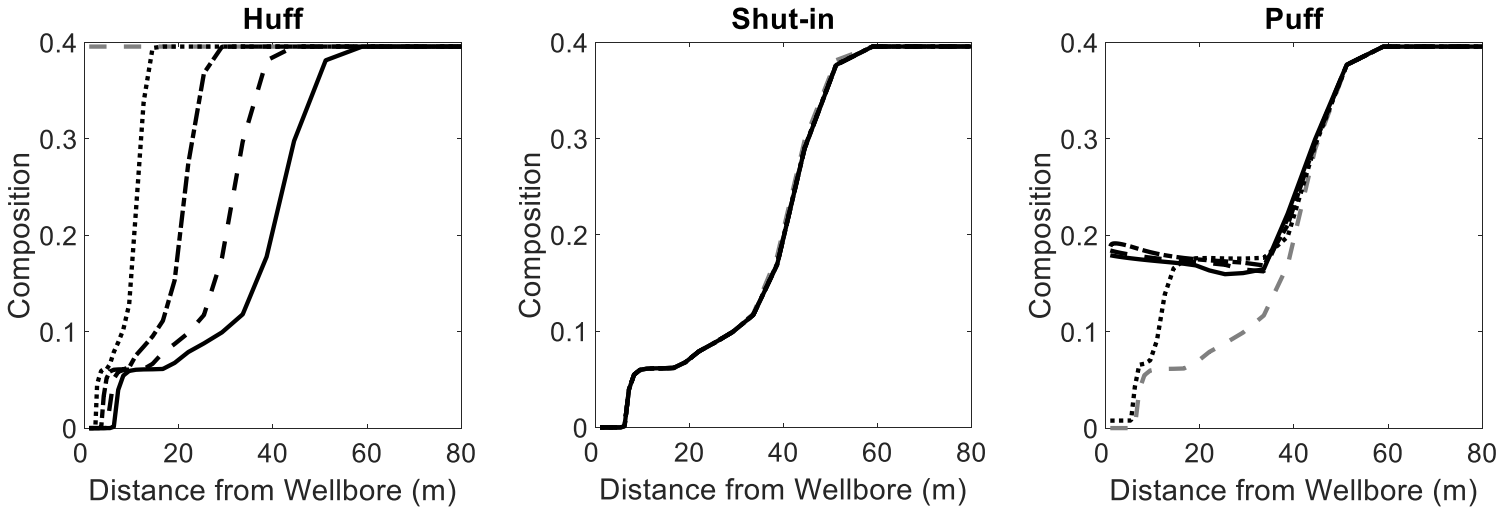


Figure 4.18(v) – Near Well C_{7+} Composition Profile for Gas 1

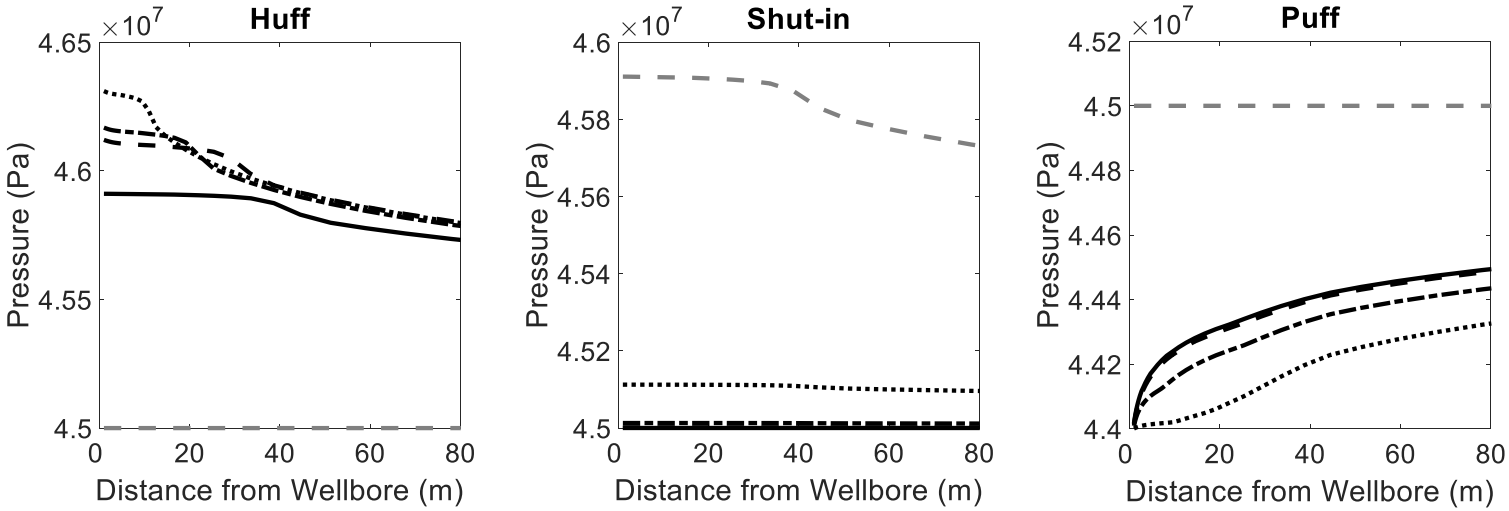


Figure 4.19(a) – Near Well Pressure Distribution for Gas 2

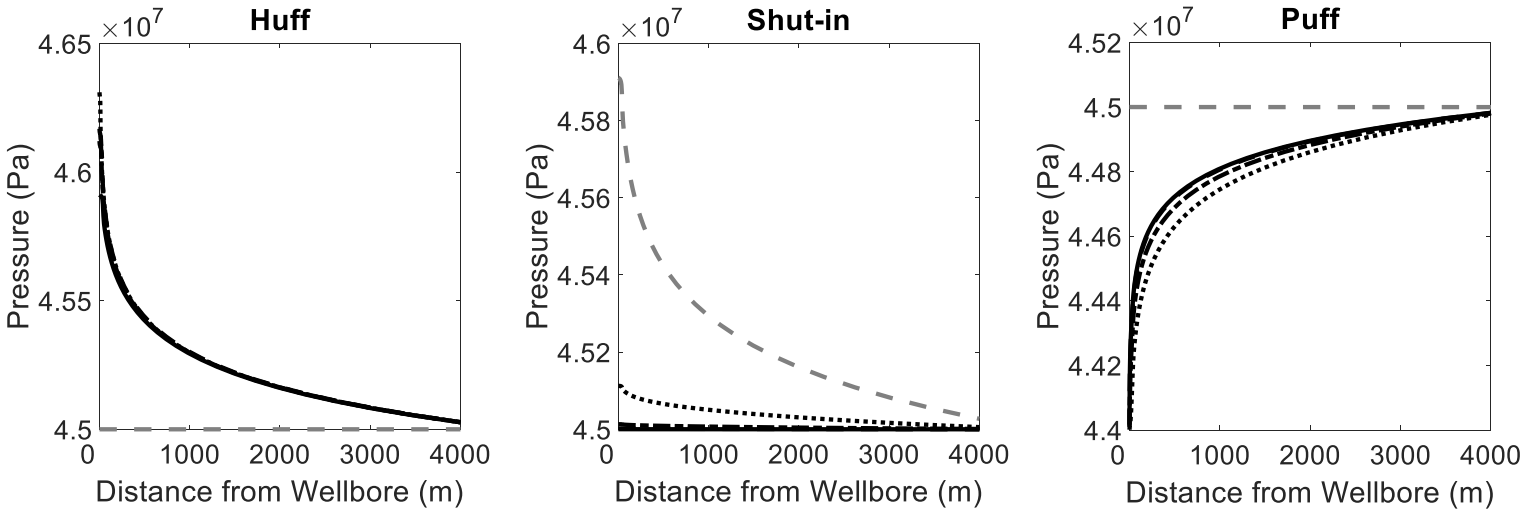


Figure 4.19(b) – Full Scale Pressure Distribution for Gas 2

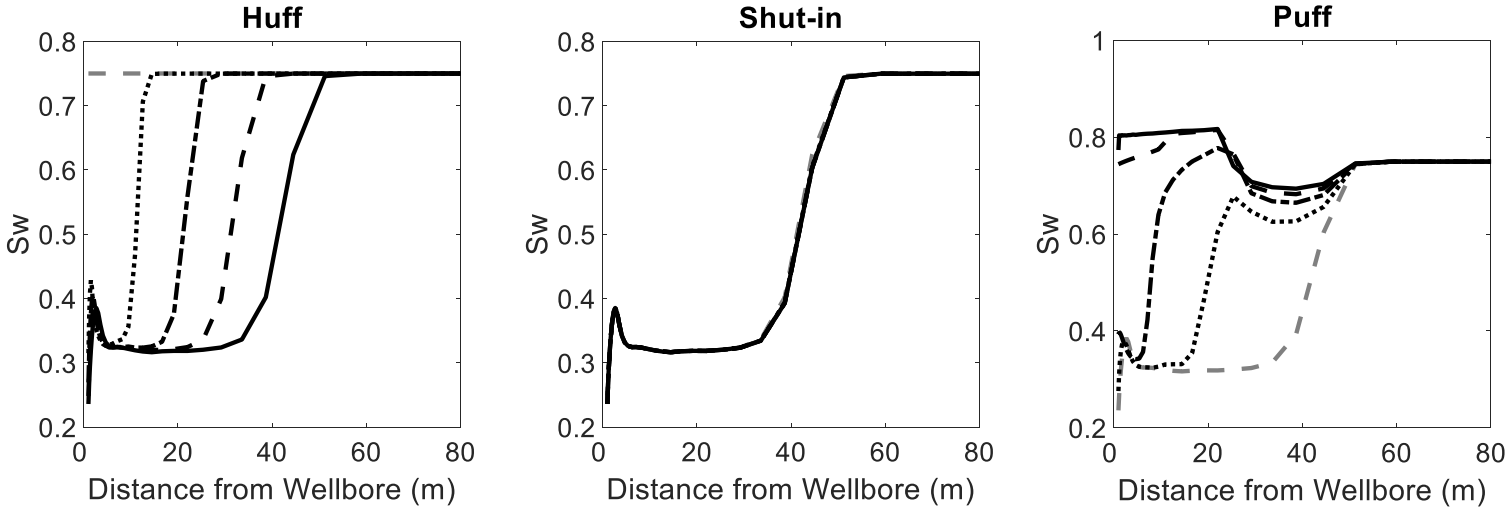


Figure 4.19(c) – Near Well Water Saturation Profile for Gas 2

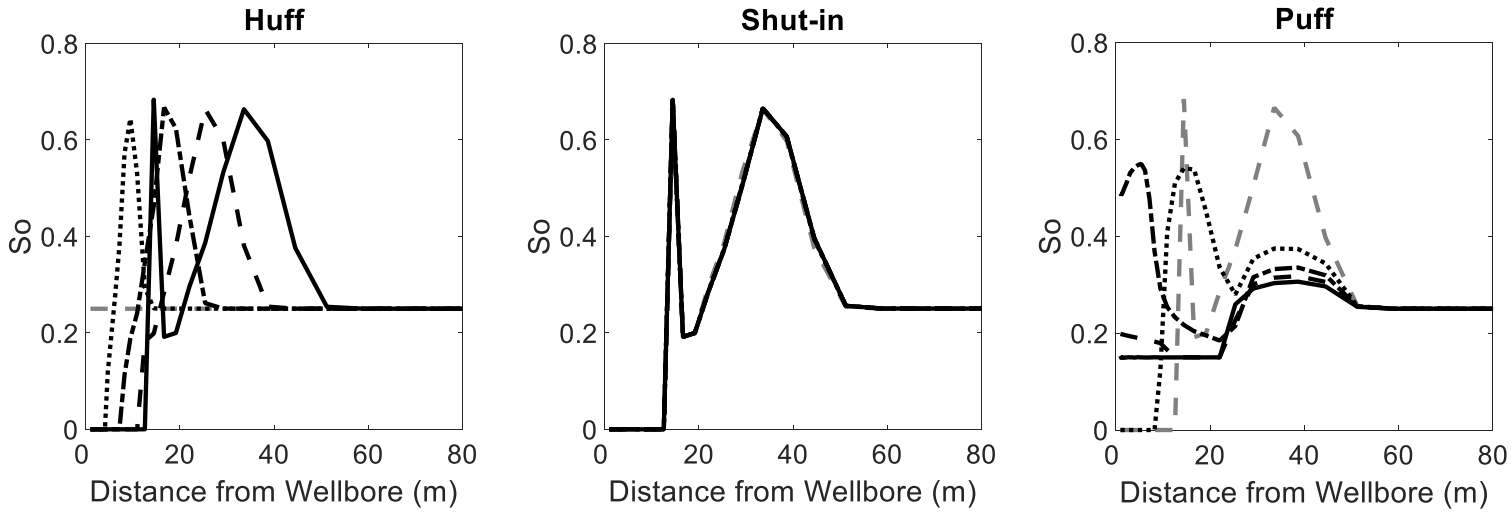


Figure 4.19(d) – Near Well Oil Saturation Profile for Gas 2

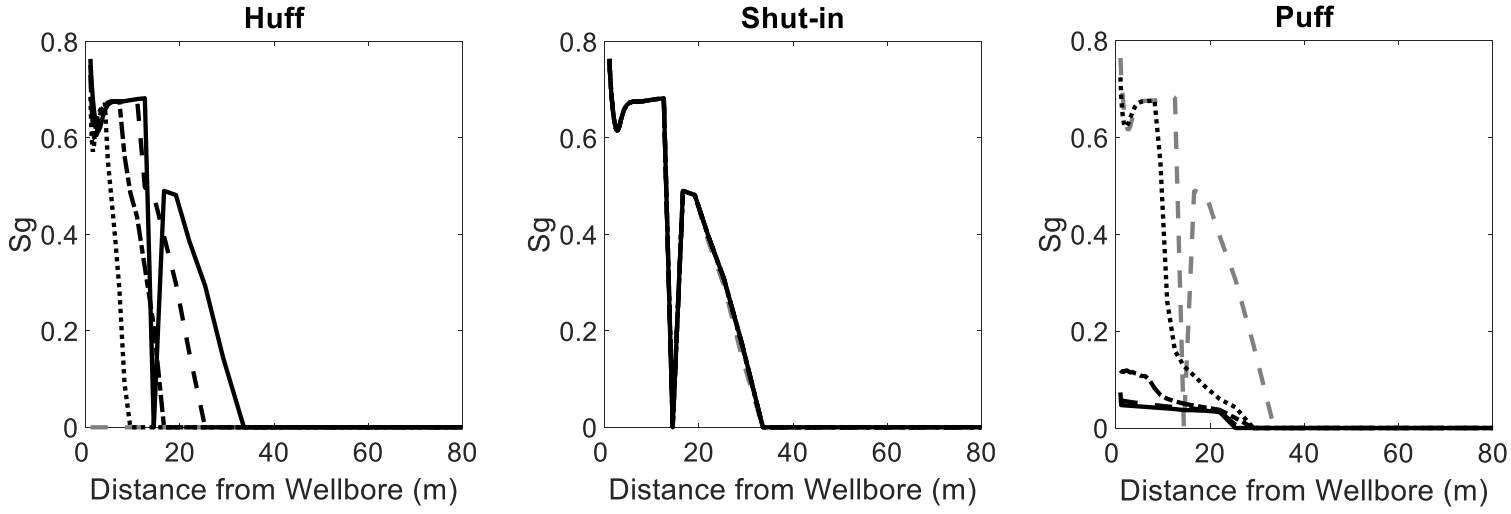


Figure 4.19(e) – Near Well Gas Saturation Profile for Gas 2

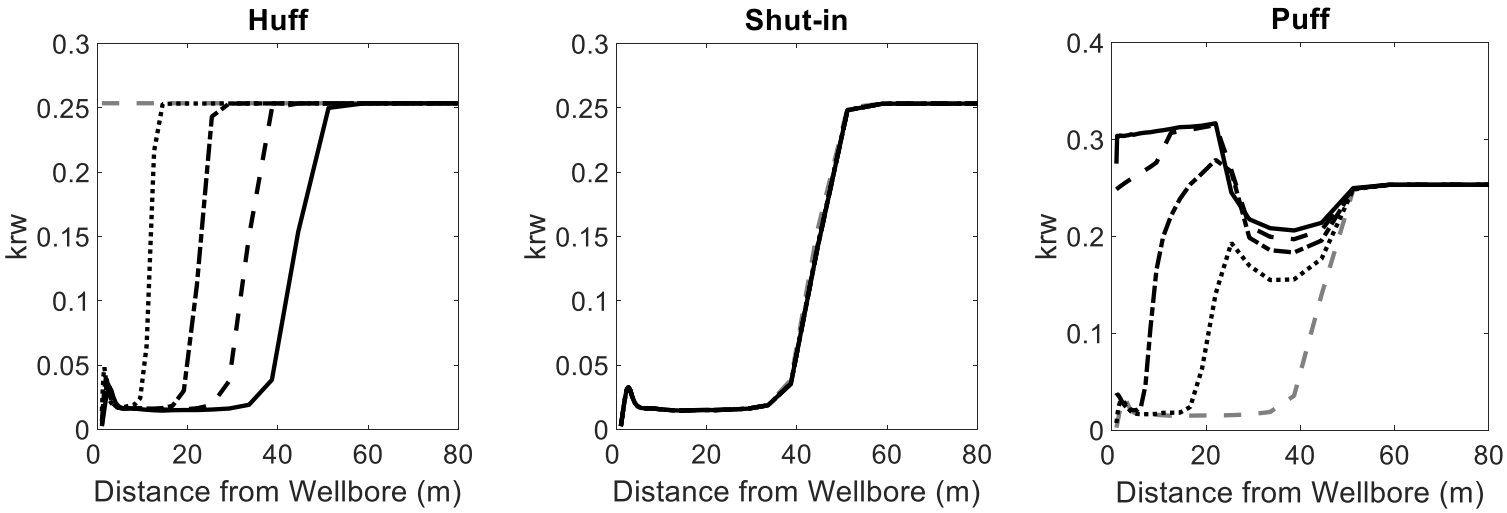


Figure 4.19(f) – Near Well Water Relative Permeability Profile for Gas 2

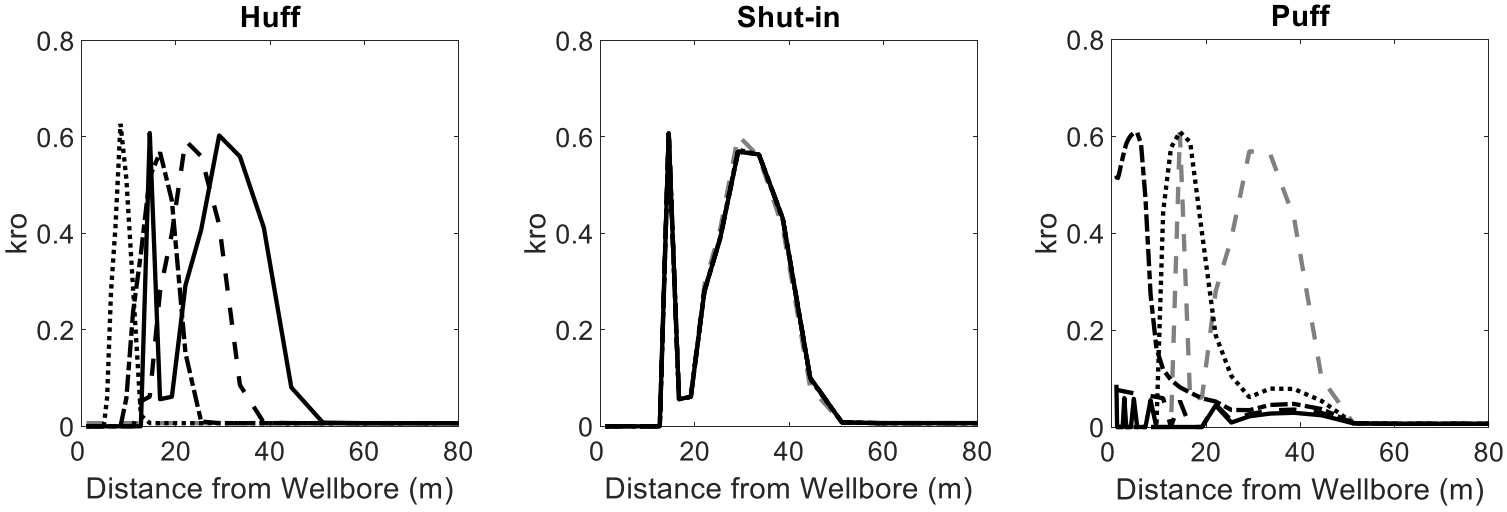


Figure 4.19(g) – Near Well Oil Relative Permeability Profile for Gas 2

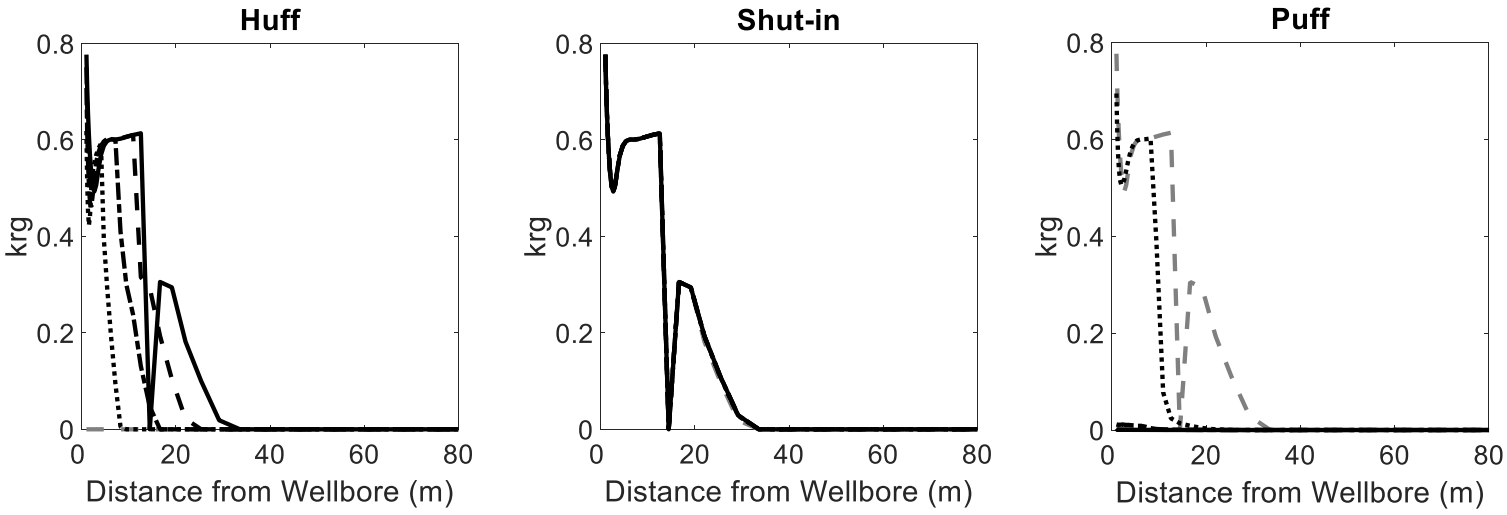


Figure 4.19(h) – Near Well Gas Relative Permeability Profile for Gas 2

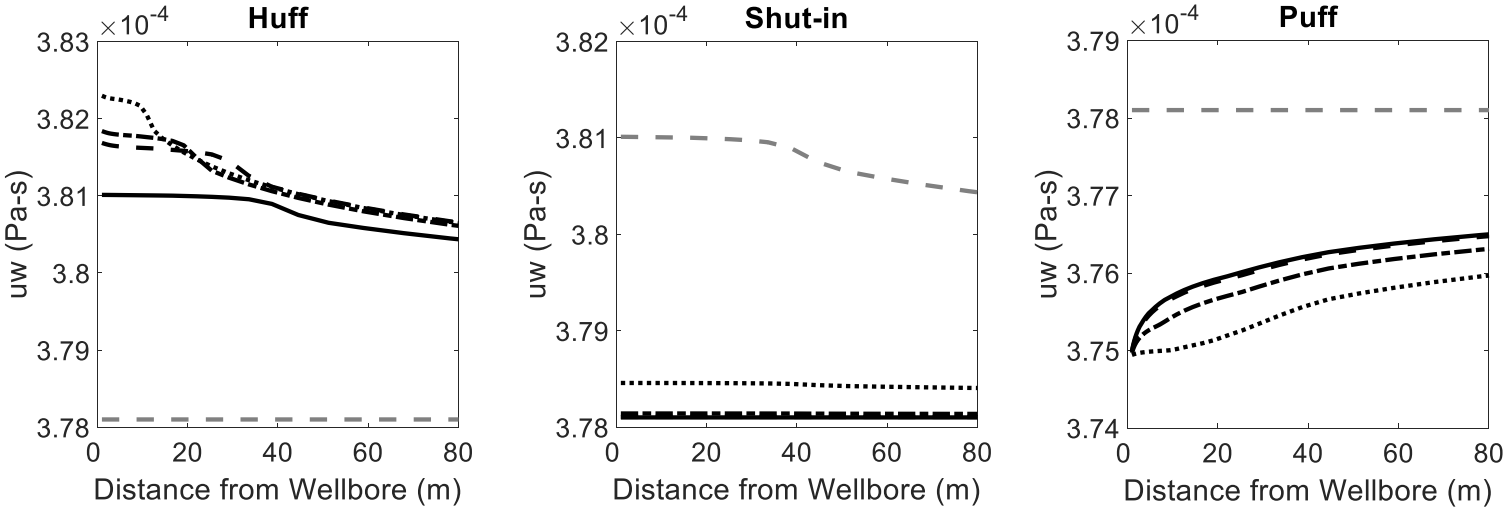


Figure 4.19(i) – Near Well Water Viscosity Profile for Gas 2

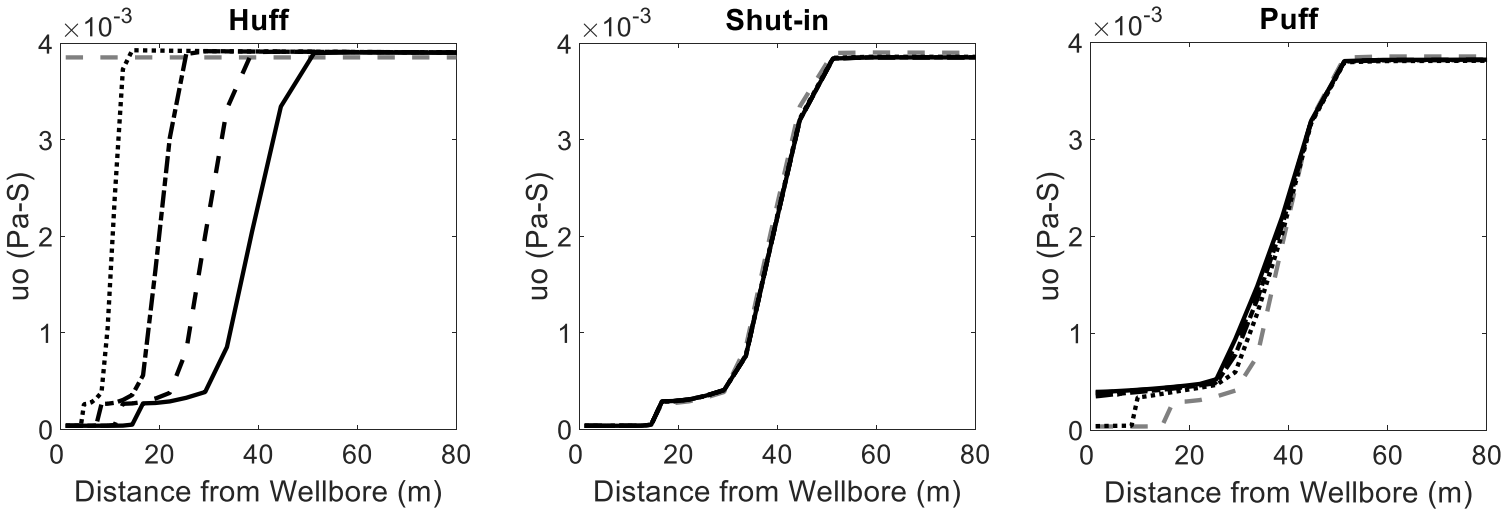


Figure 4.19(j) – Near Well Oil Viscosity Profile for Gas 2

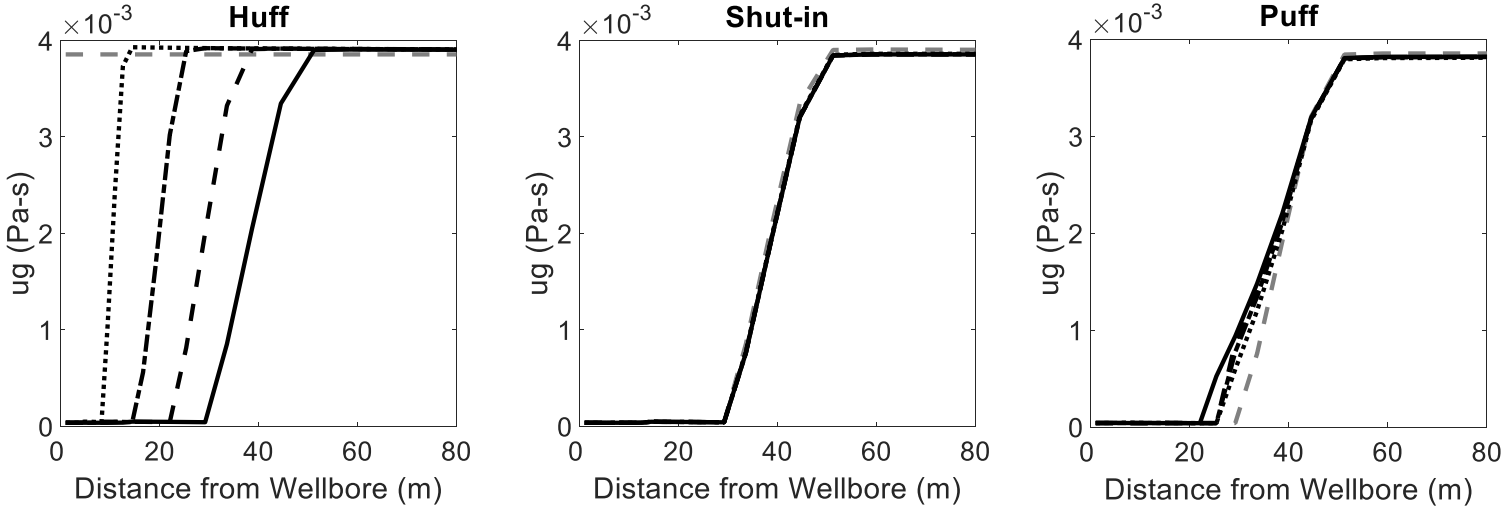


Figure 4.19(k) – Near Well Gas Viscosity Profile for Gas 2

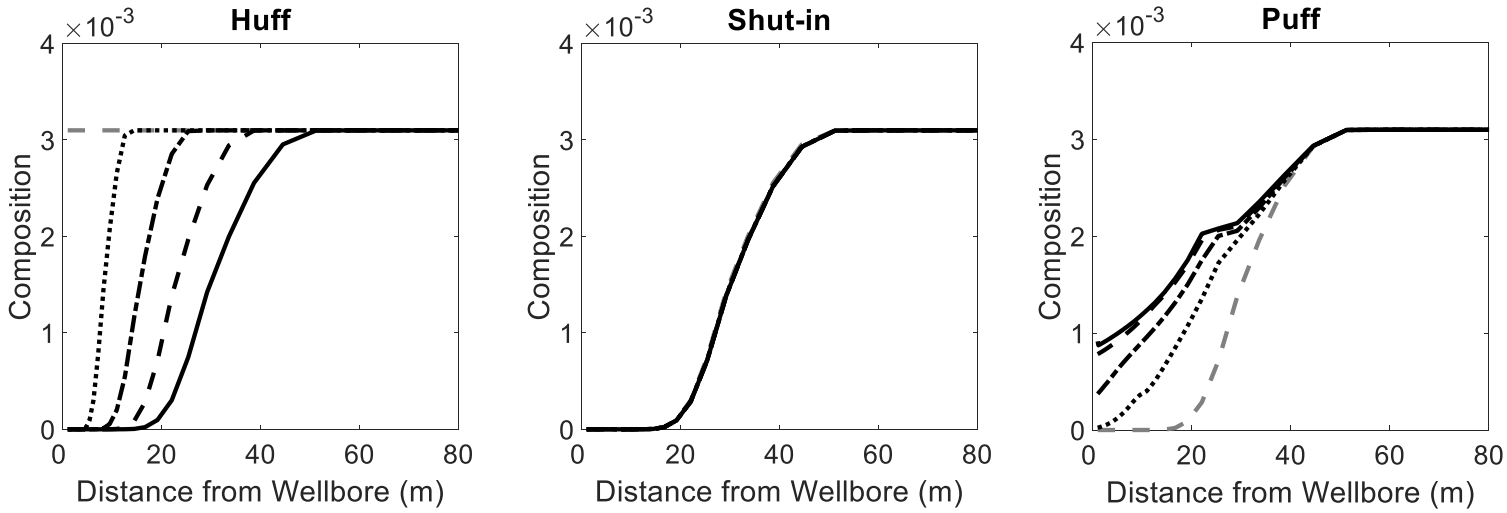


Figure 4.19(l) – Near Well N_2 Composition Profile for Gas 2

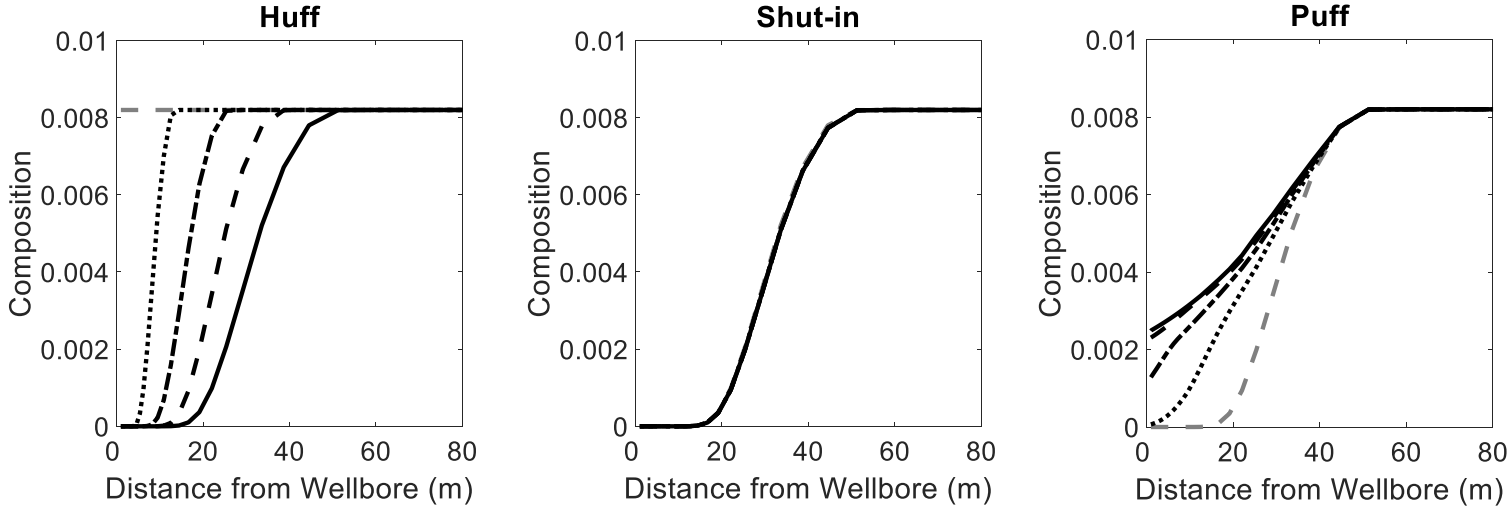


Figure 4.19(m) – Near Well CO_2 Composition Profile for Gas 2

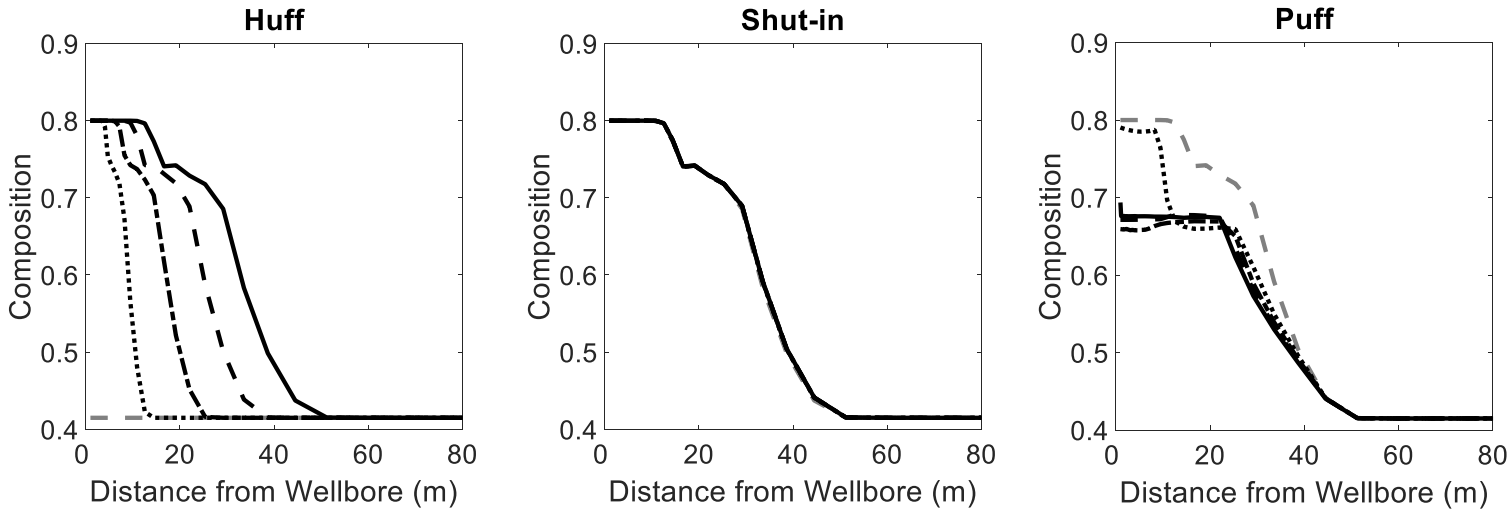


Figure 4.19(n) – Near Well C_1 Composition Profile for Gas 2

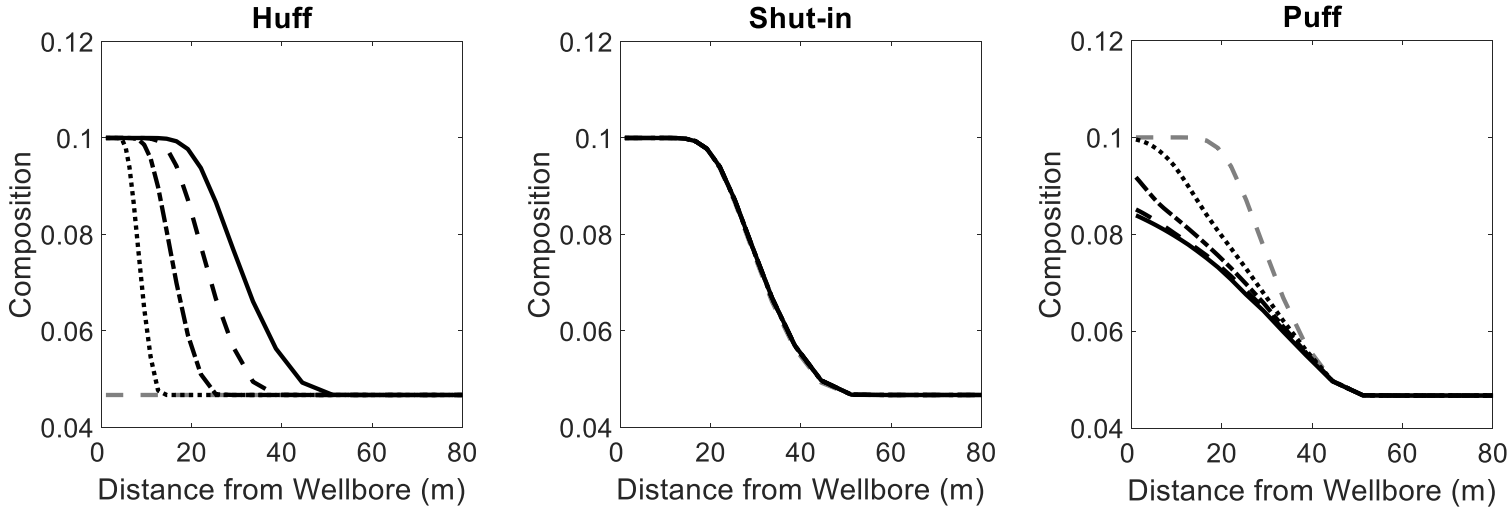


Figure 4.19(o) – Near Well C_2 Composition Profile for Gas 2

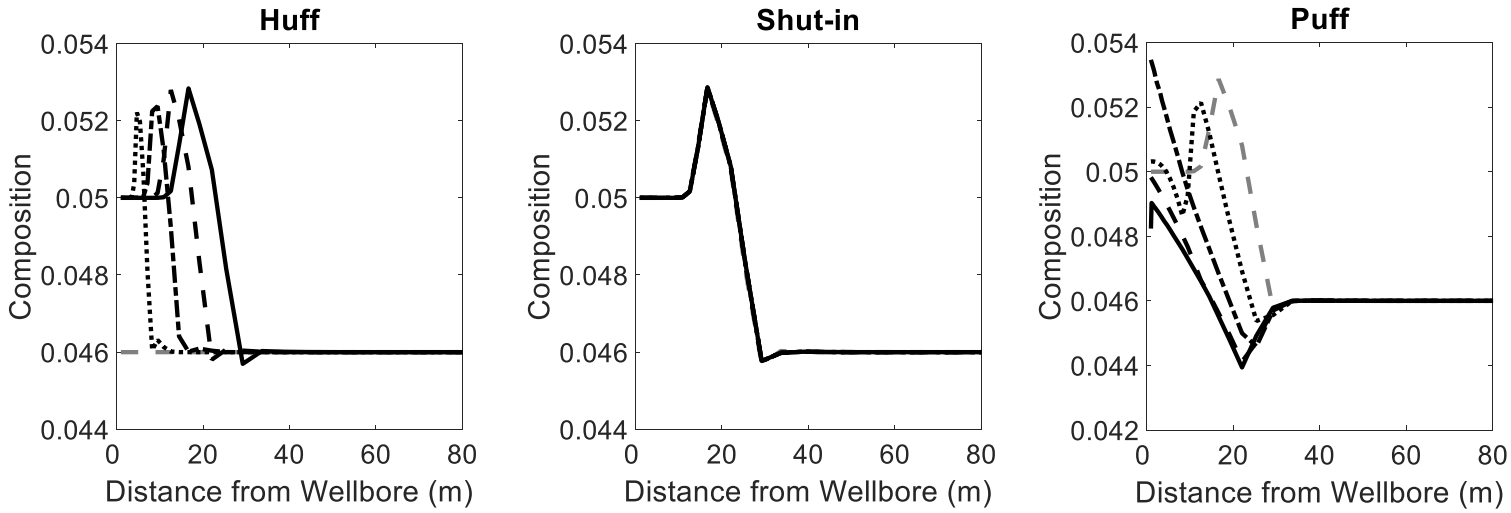


Figure 4.19(p) – Near Well C_3 Composition Profile for Gas 2

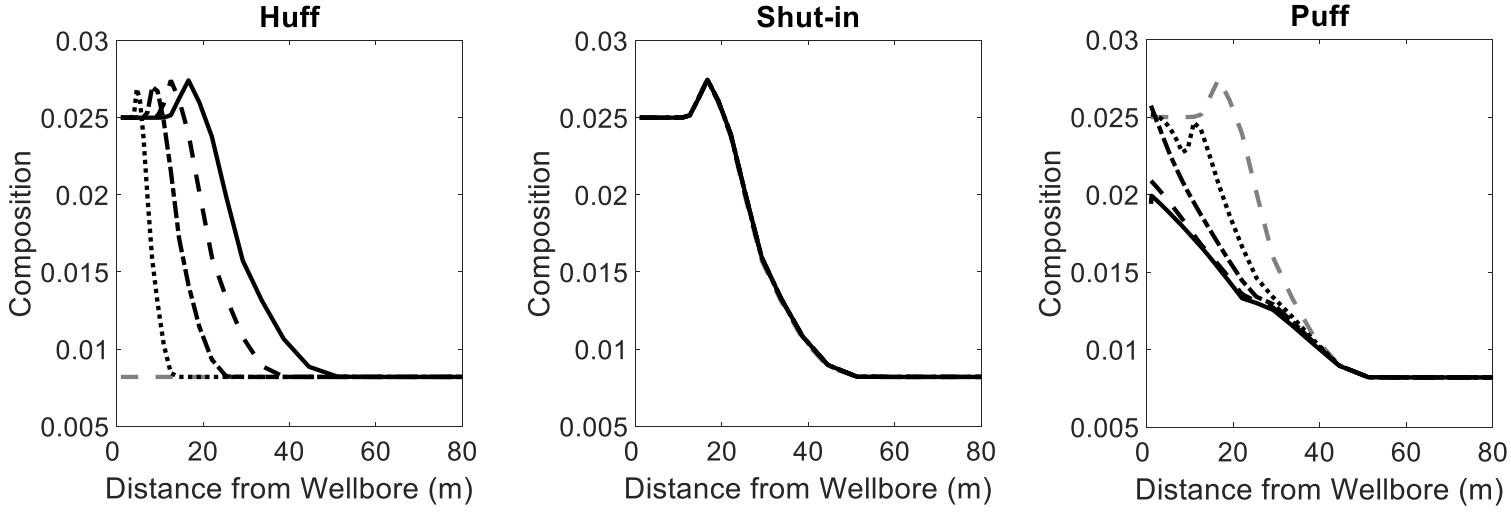


Figure 4.19(q) – Near Well iC_4 Composition Profile for Gas 2

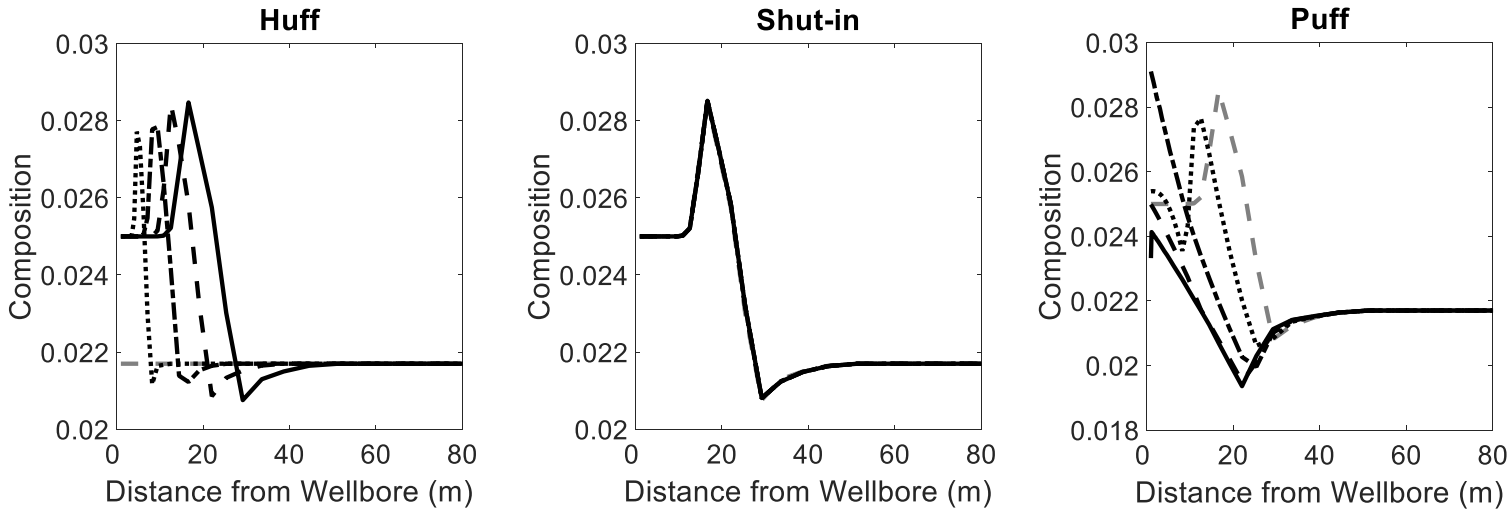


Figure 4.19(r) – Near Well nC_4 Composition Profile for Gas 2

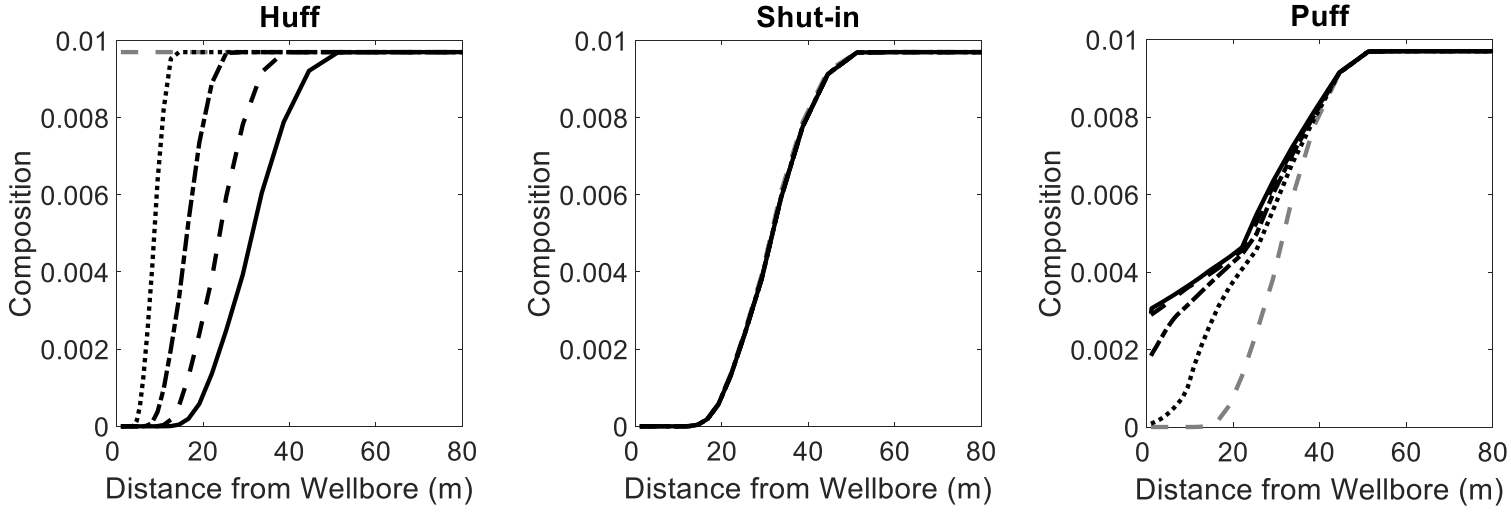


Figure 4.19(s) – Near Well iC_5 Composition Profile for Gas 2

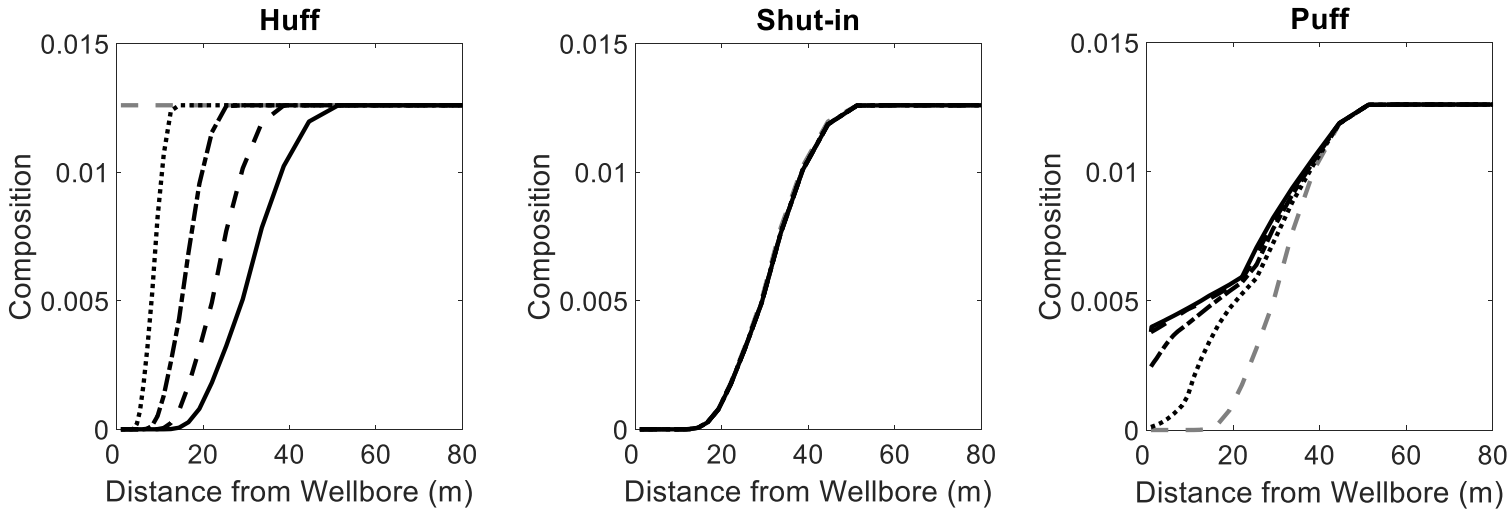


Figure 4.19(t) – Near Well nC_5 Composition Profile for Gas 2

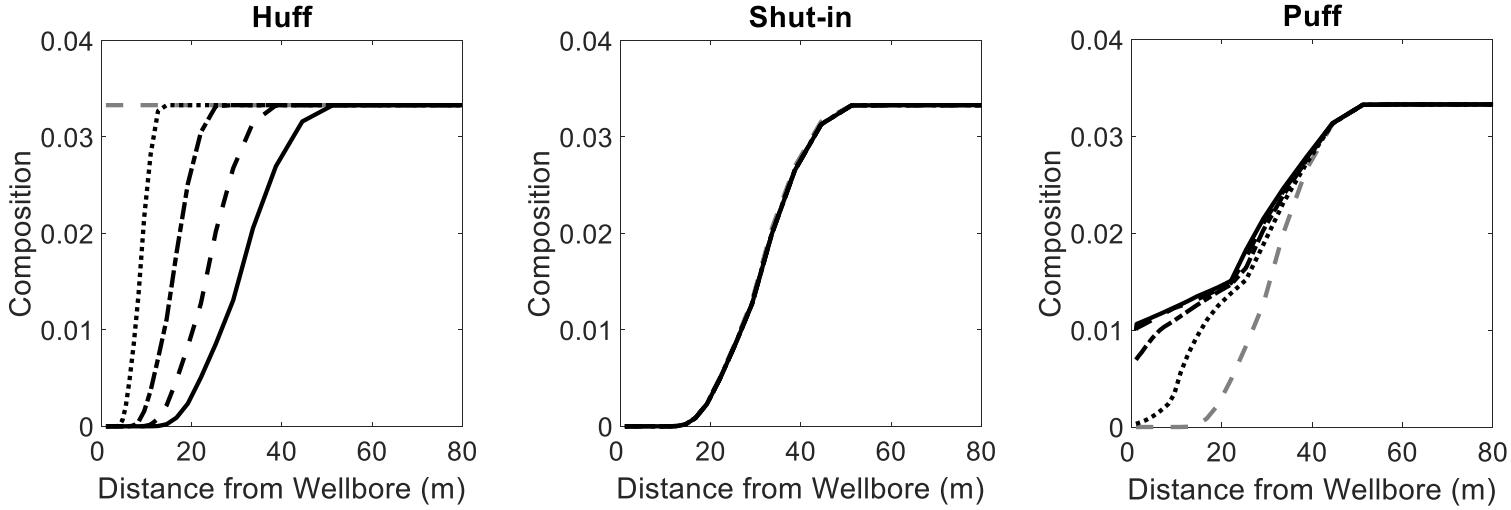


Figure 4.19(u) – Near Well nC_6 Composition Profile for Gas 2

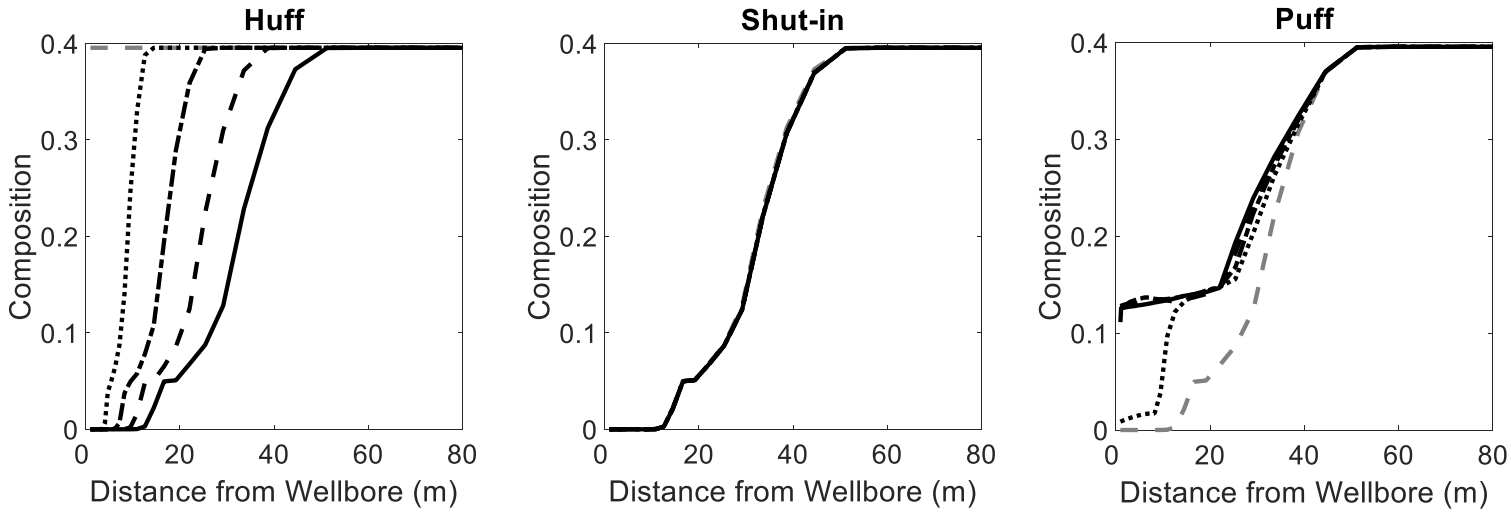


Figure 4.19(v) – Near Well C_{7+} Composition Profile for Gas 2

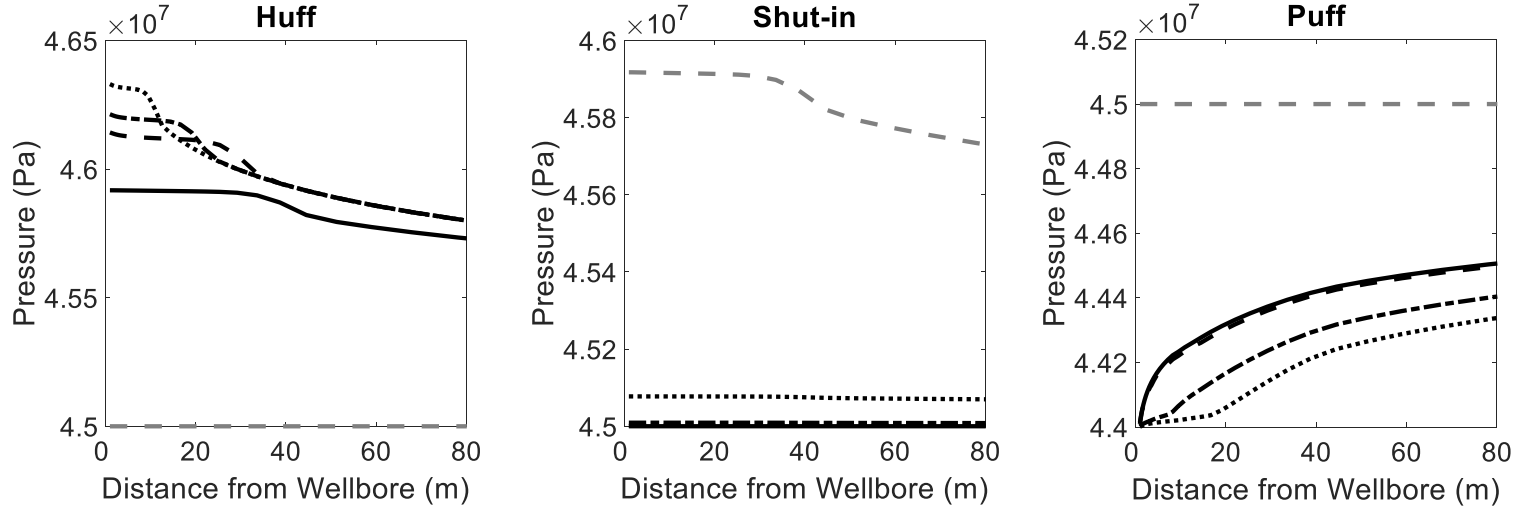


Figure 4.20(a) – Near Well Pressure Distribution for Gas 3

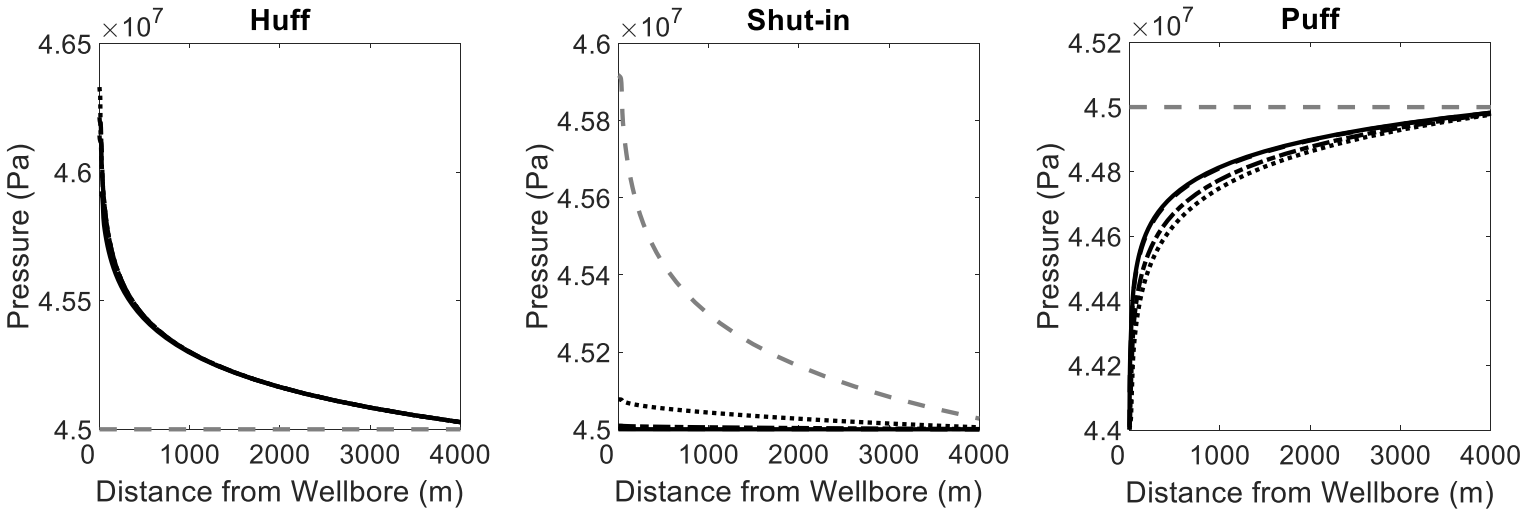


Figure 4.20(b) – Full Scale Pressure Distribution for Gas 3

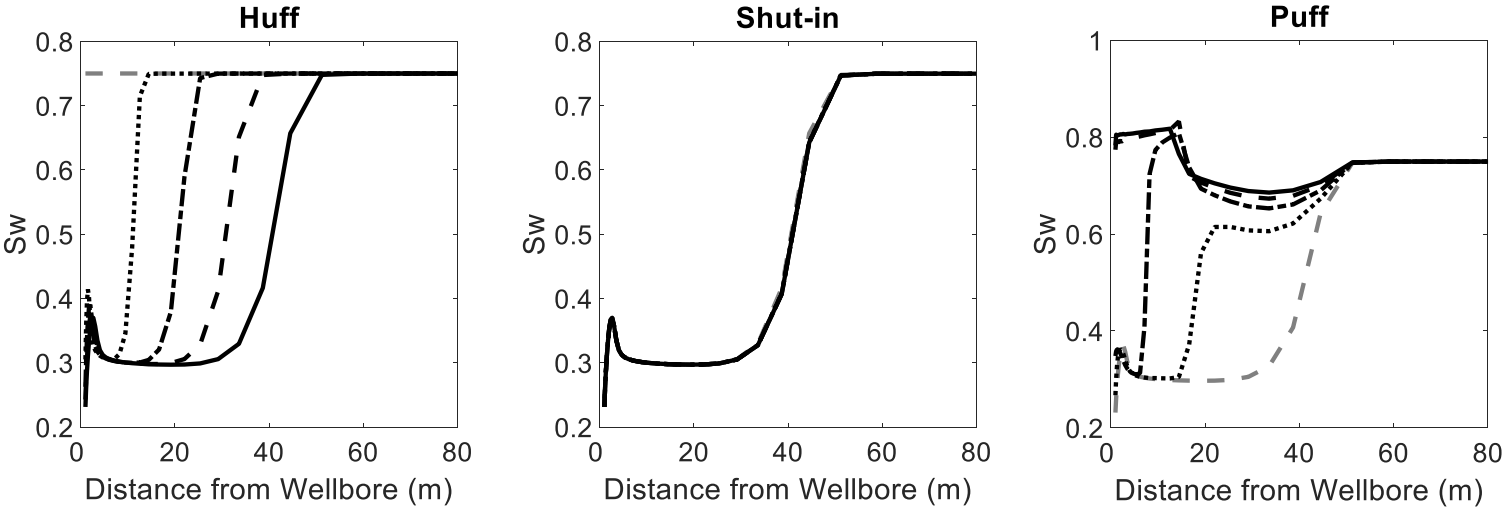


Figure 4.20(c) – Near Well Water Saturation Profile for Gas 3

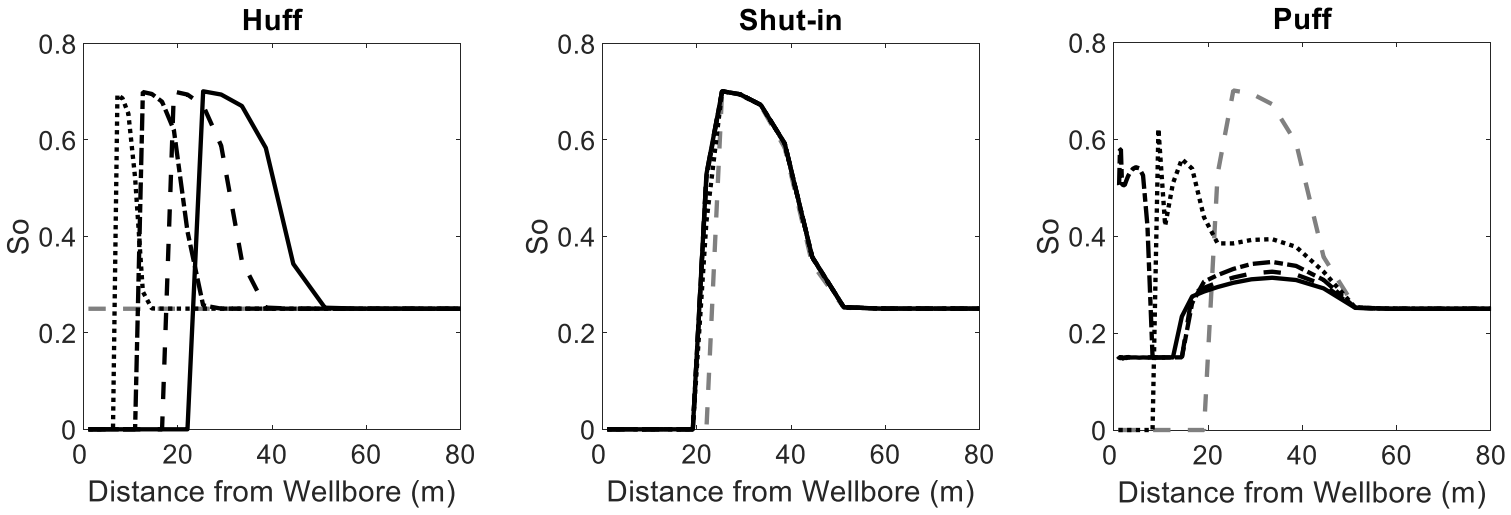


Figure 4.20(d) – Near Well Oil Saturation Profile for Gas 3

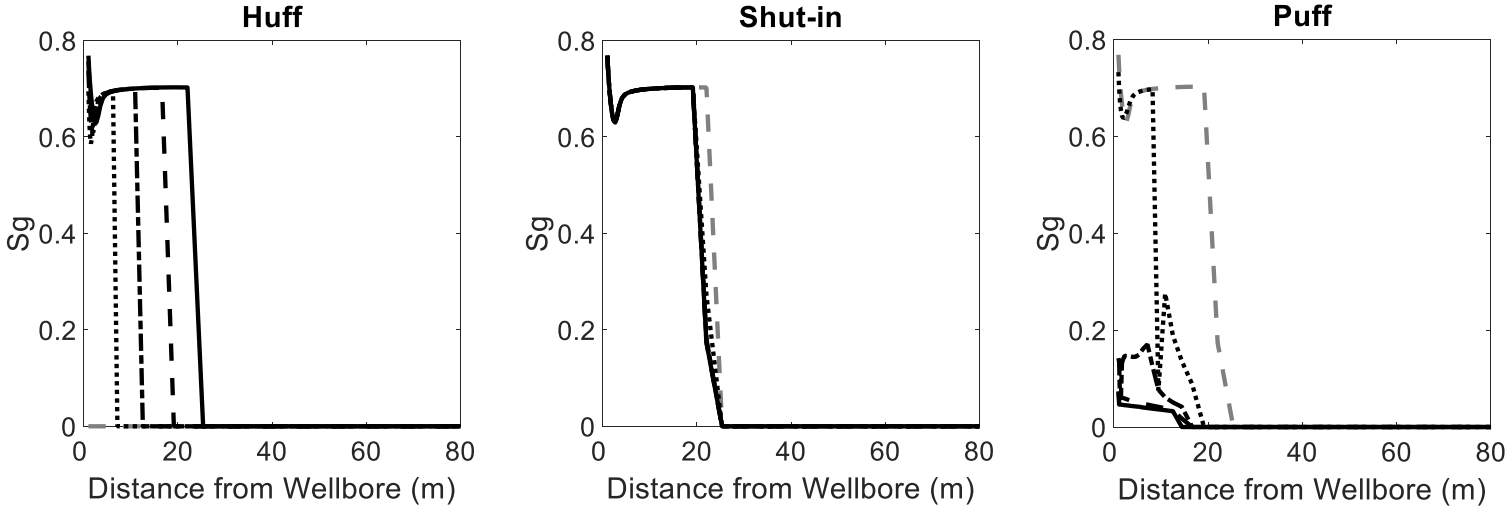


Figure 4.20(e) – Near Well Gas Saturation Profile for Gas 3

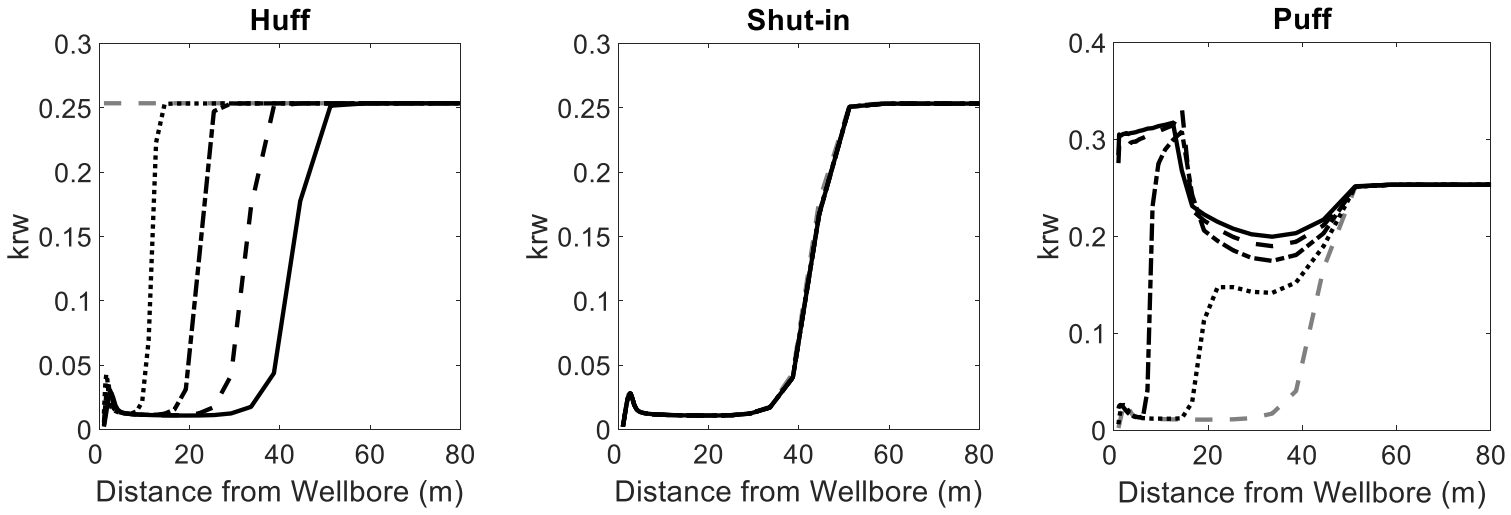


Figure 4.20(f) – Near Well Water Relative Permeability Profile for Gas 3

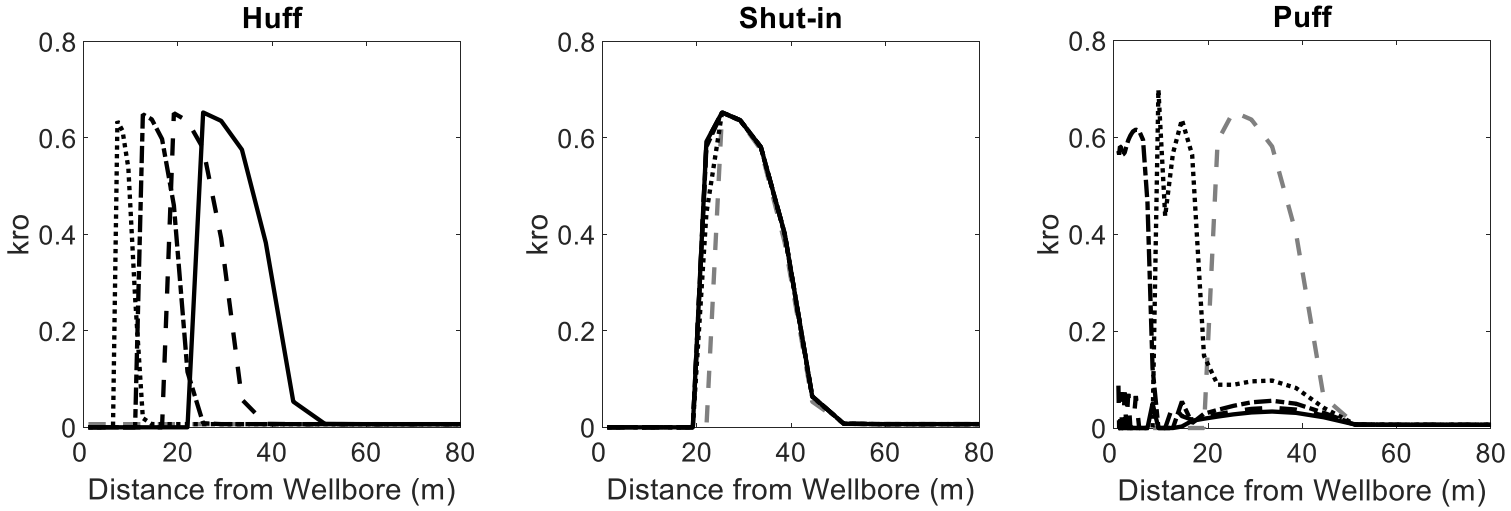


Figure 4.20(g) – Near Well Oil Relative Permeability Profile for Gas 3

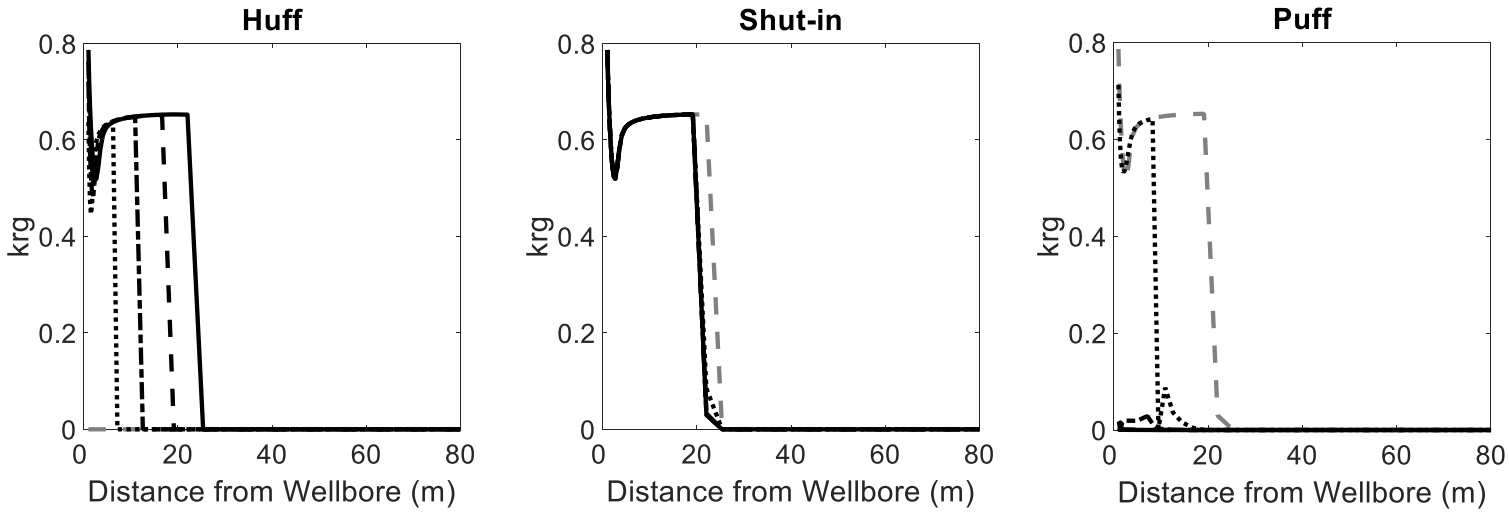


Figure 4.20(h) – Near Well Gas Relative Permeability Profile for Gas 3

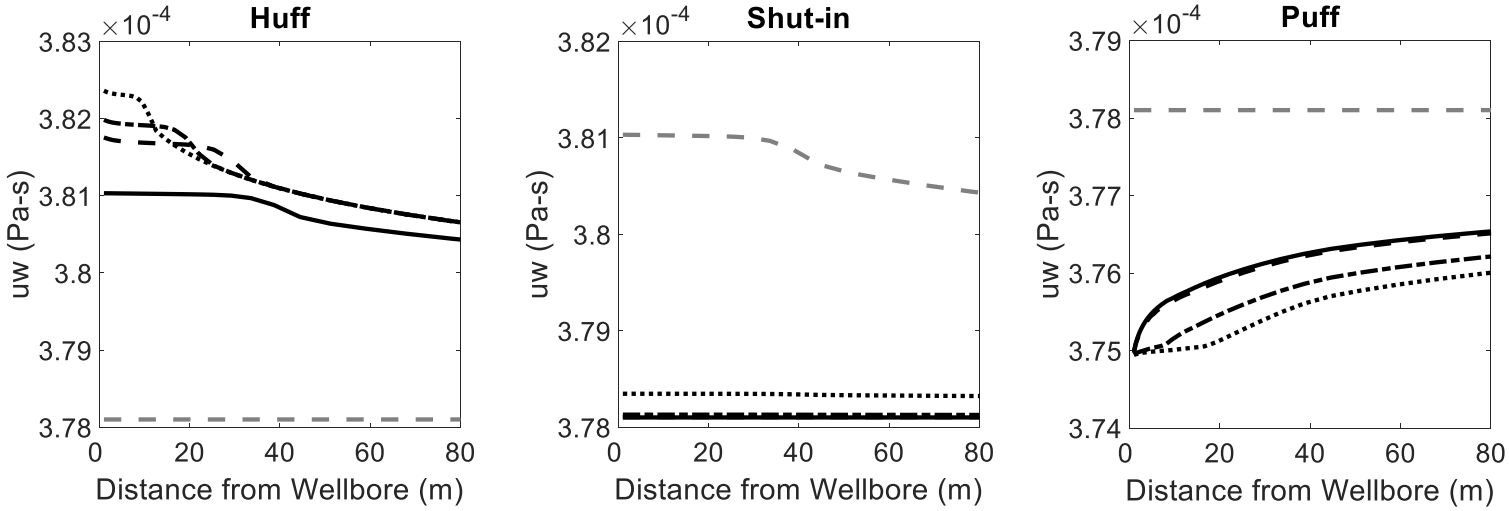


Figure 4.20(i) – Near Well Water Viscosity Profile for Gas 3

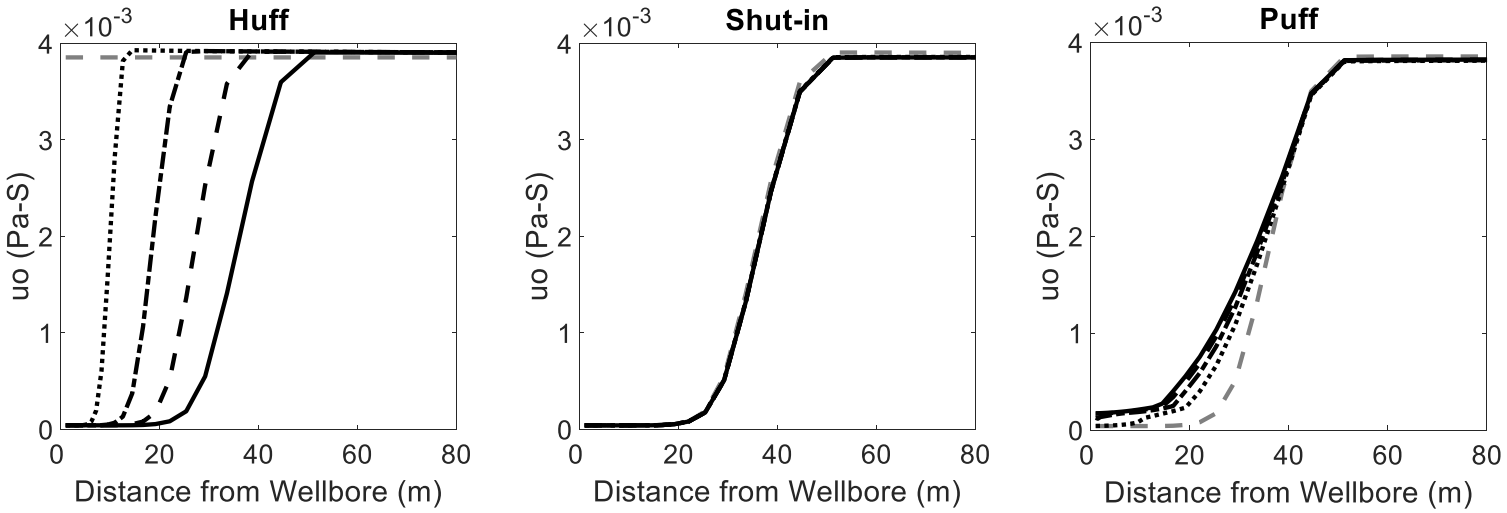


Figure 4.20(j) – Near Well Oil Viscosity Profile for Gas 3

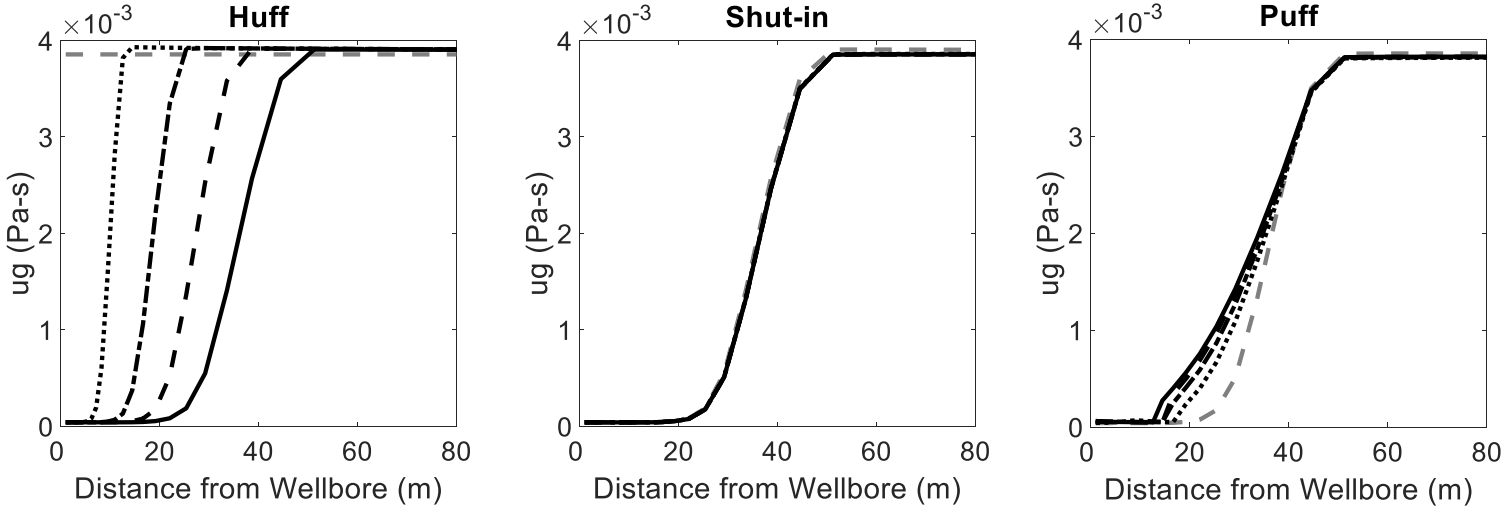


Figure 4.20(k) – Near Well Gas Viscosity Profile for Gas 3

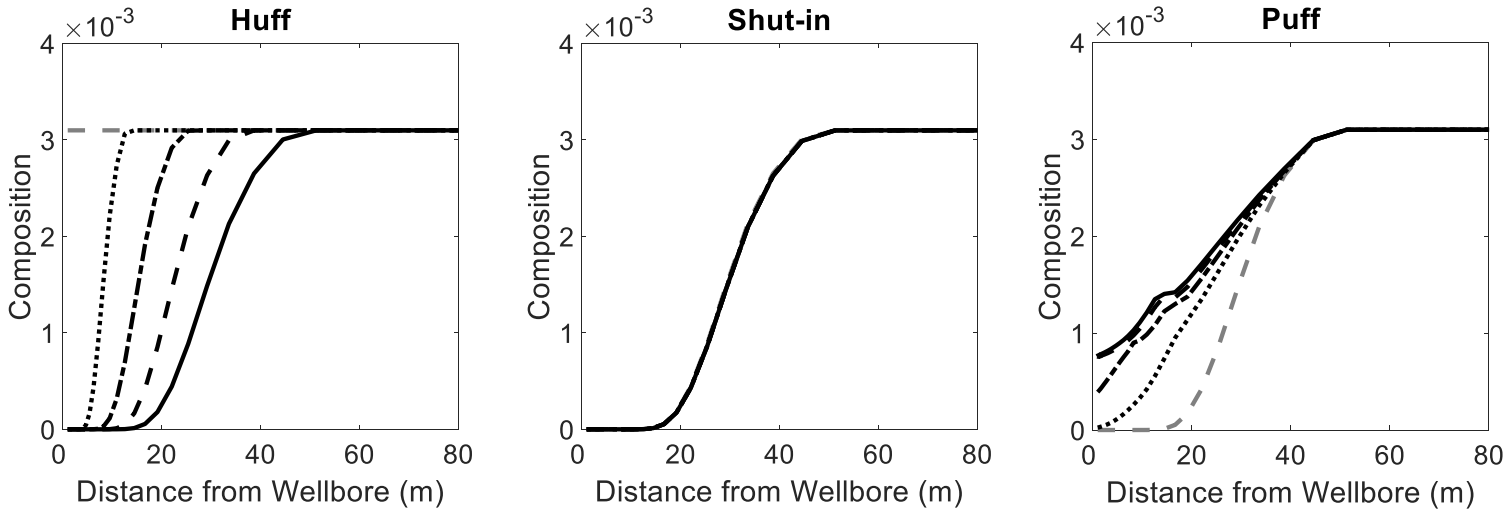


Figure 4.20(l) – Near Well N_2 Composition Profile for Gas 3

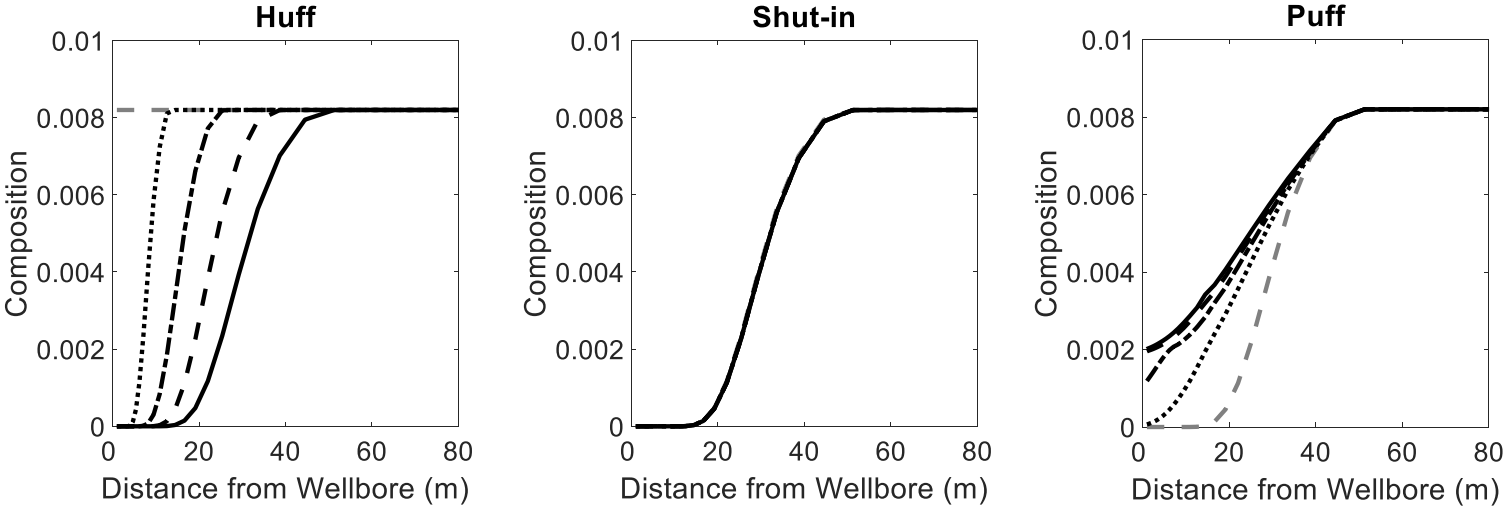


Figure 4.20(m) – Near Well CO_2 Composition Profile for Gas 3

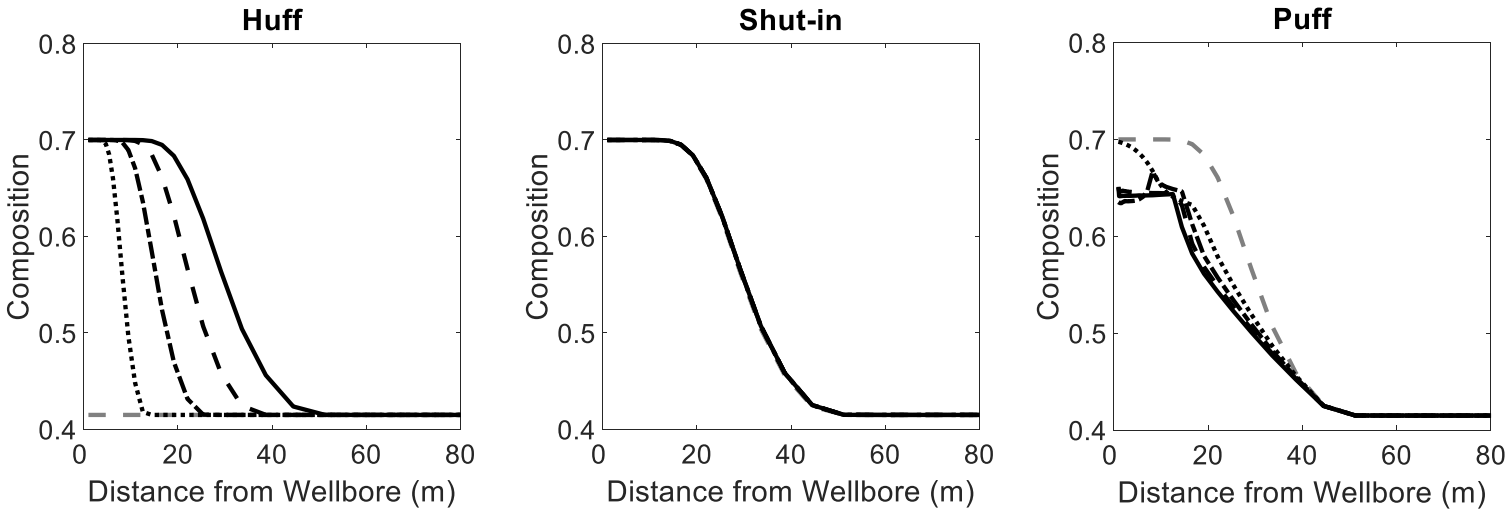


Figure 4.20(n) – Near Well C_1 Composition Profile for Gas 3

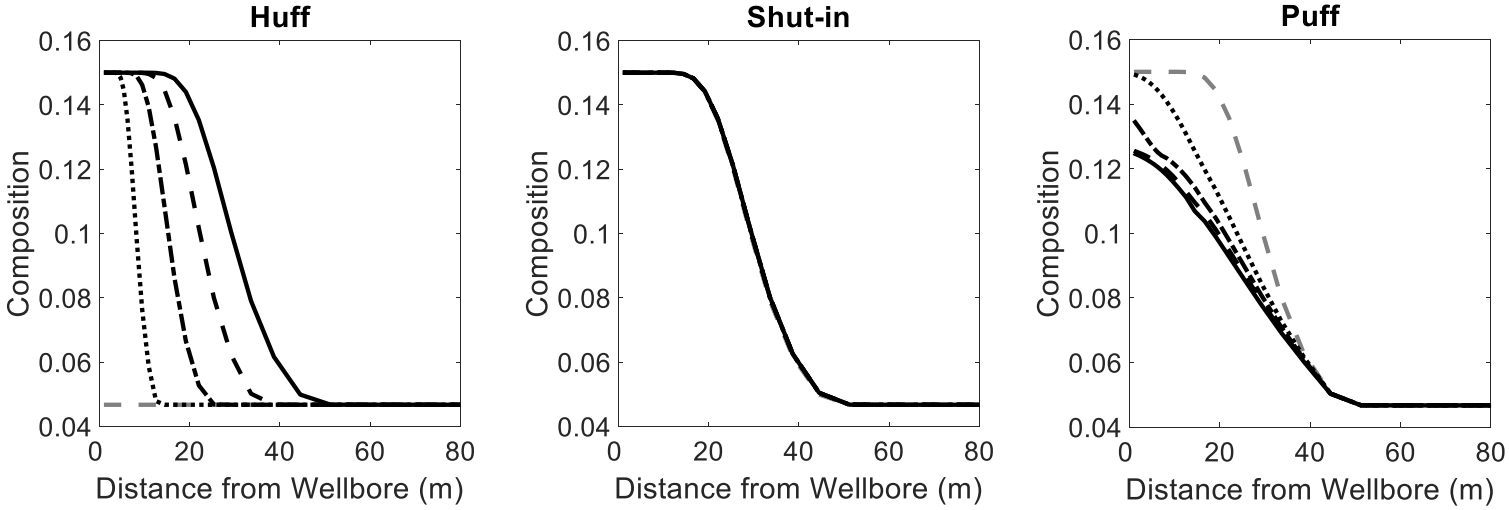


Figure 4.20(o) – Near Well C_2 Composition Profile for Gas 3

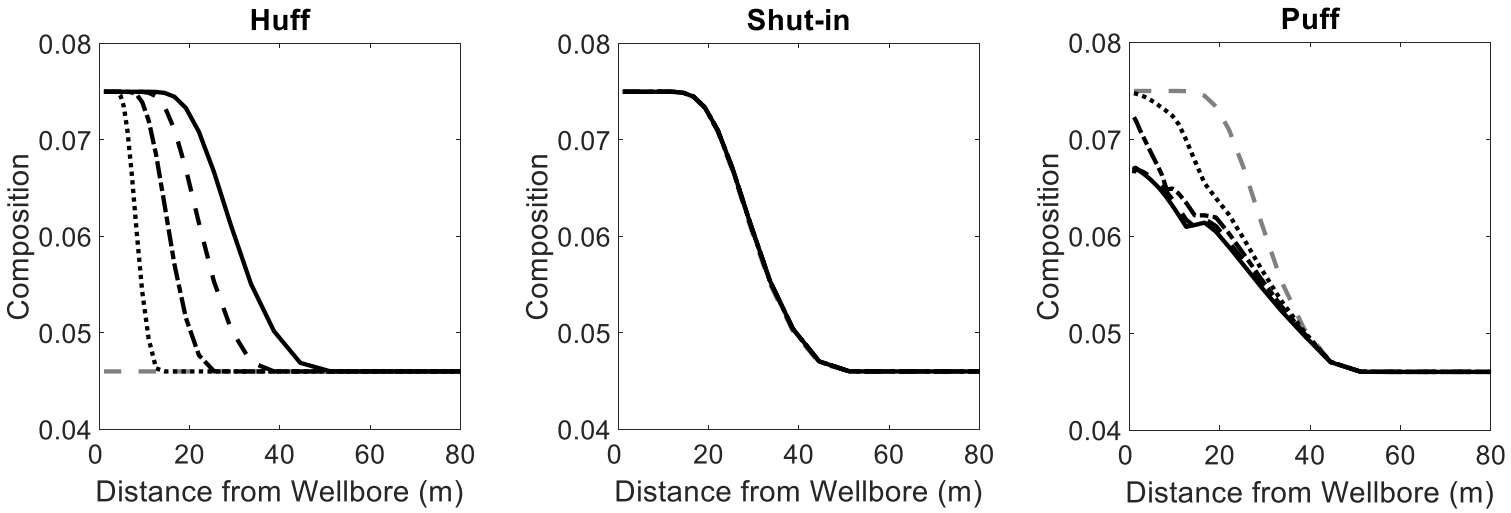


Figure 4.20(p) – Near Well C_3 Composition Profile for Gas 3

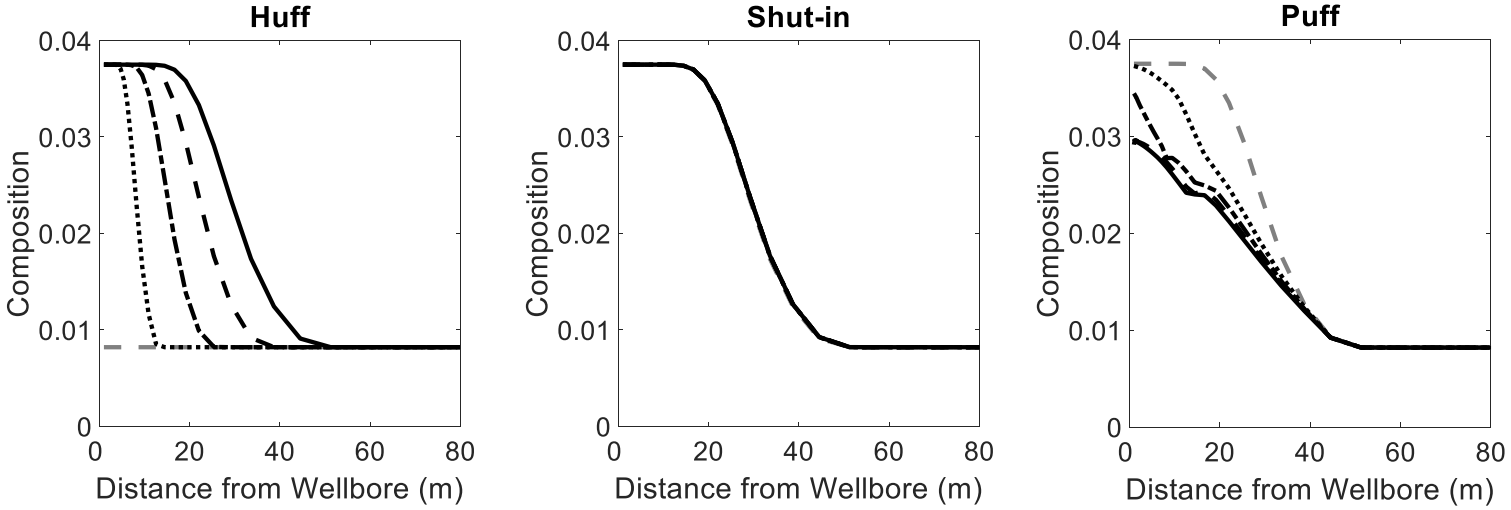


Figure 4.20(q) – Near Well iC_4 Composition Profile for Gas 3

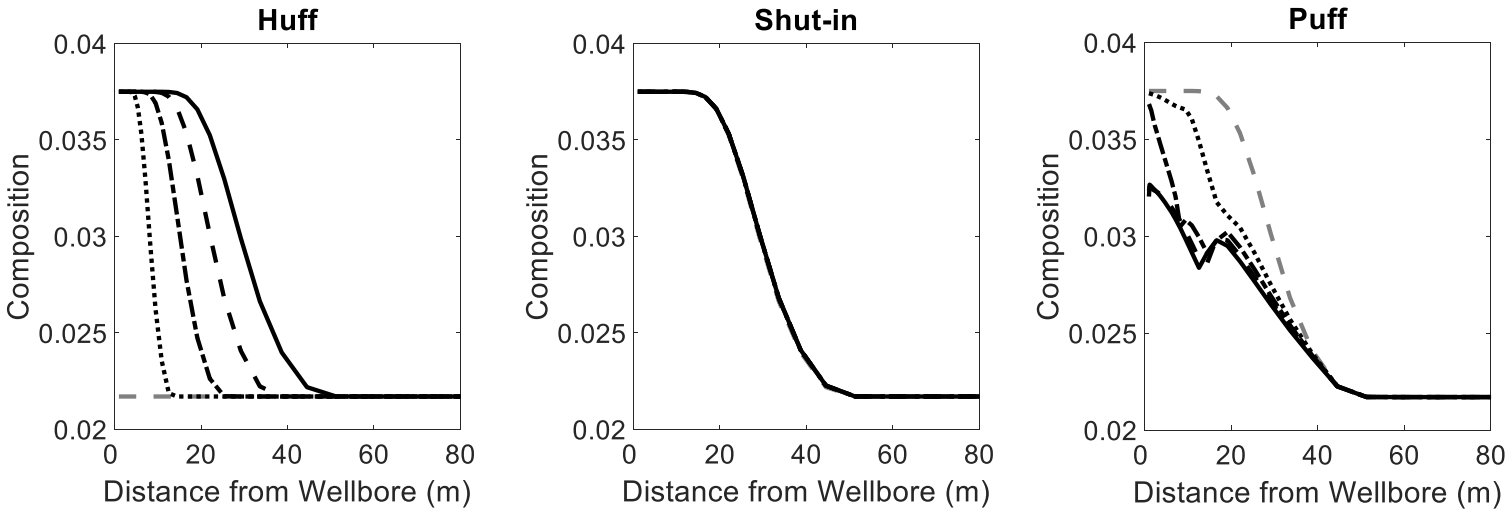


Figure 4.20(r) – Near Well nC_4 Composition Profile for Gas 3

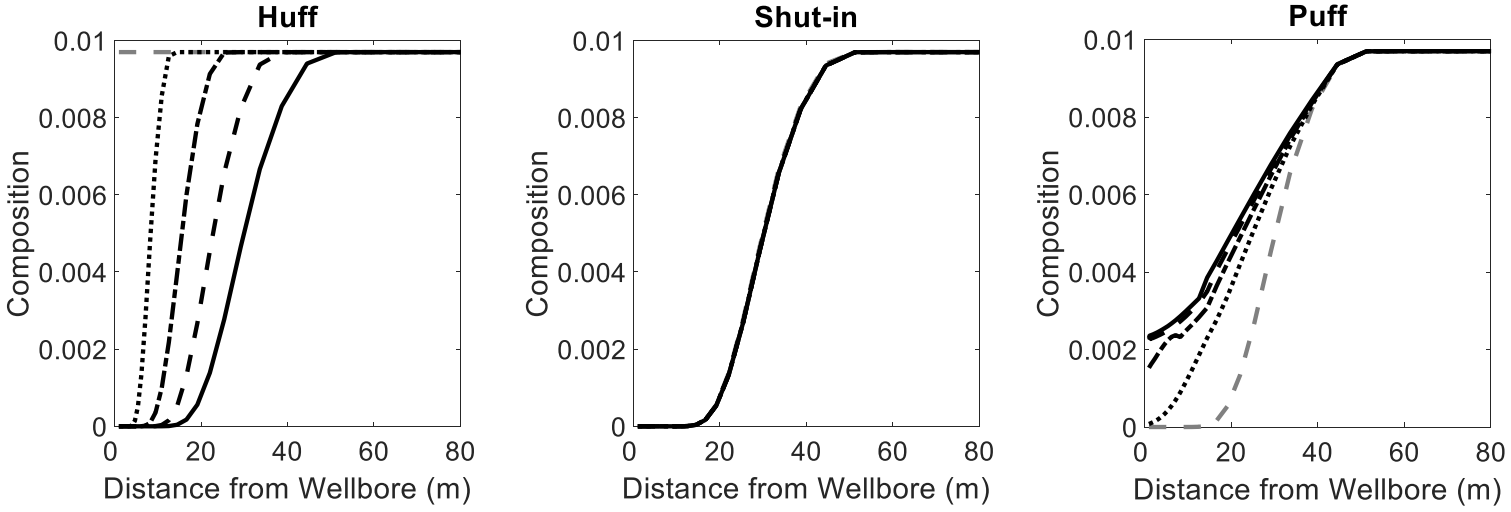


Figure 4.20(s) – Near Well iC_5 Composition Profile for Gas 3

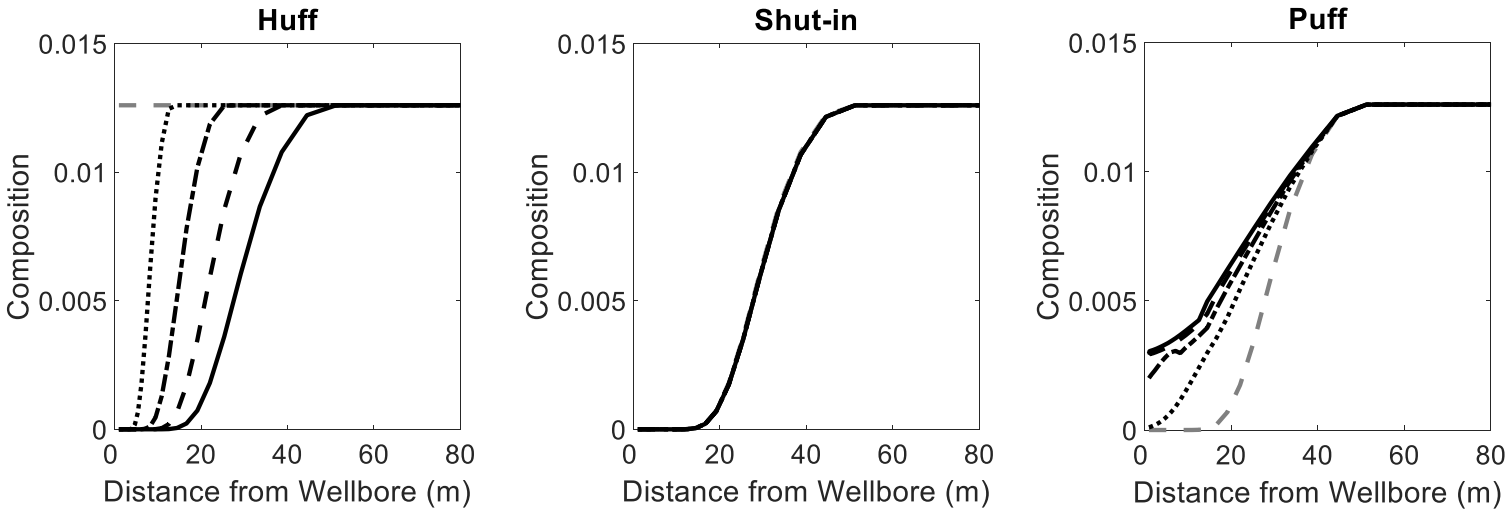


Figure 4.20(t) – Near Well nC_5 Composition Profile for Gas 3

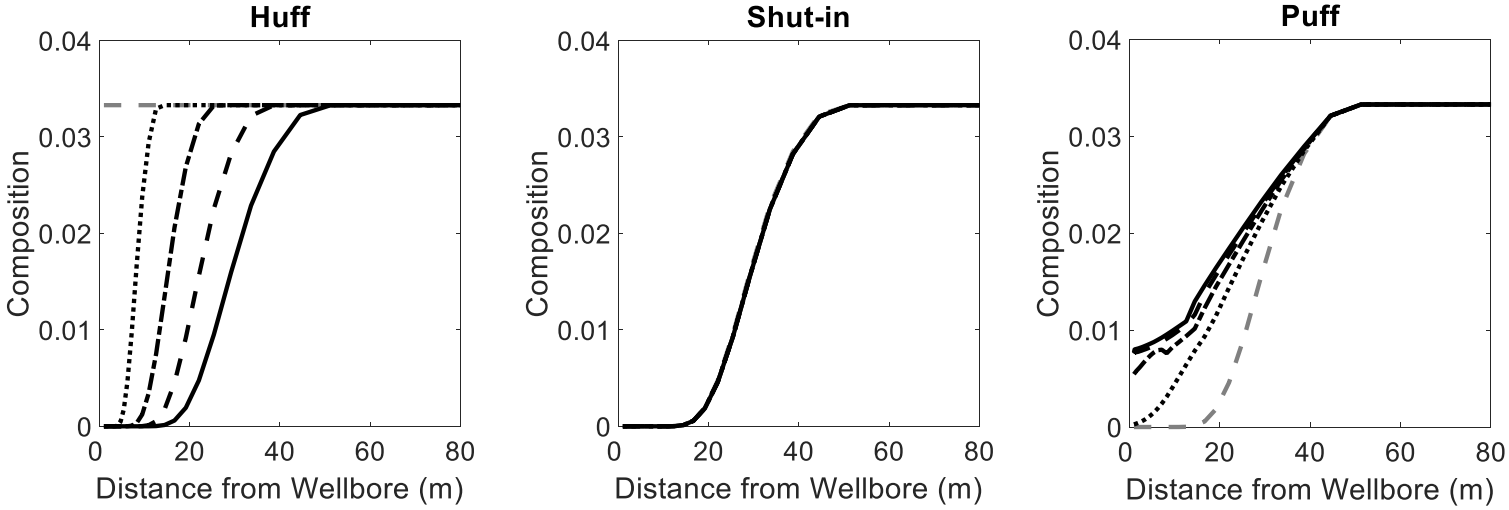


Figure 4.20(u) – Near Well nC_6 Composition Profile for Gas 3

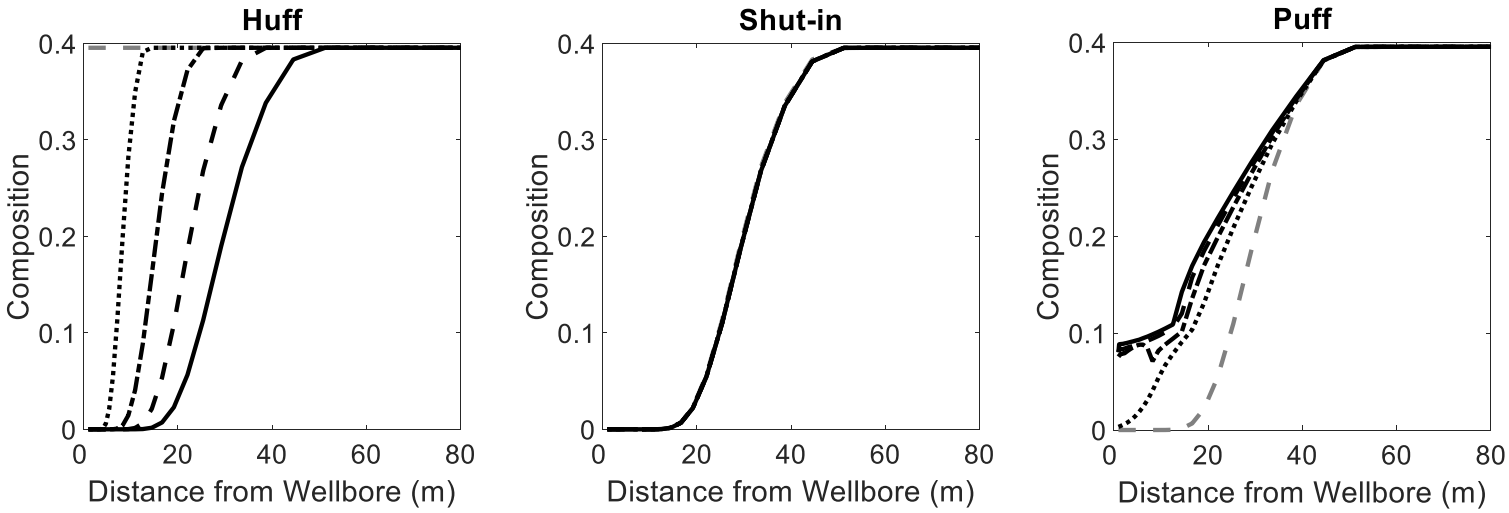


Figure 4.20(v) – Near Well C_{7+} Composition Profile for Gas 3

Chapter 5 Conclusions and Recommendations

5.1 Conclusions

This work focused on creating a numerical model which could compositionally simulate three phases in one dimension, in order to model the natural gas huff 'n' puff process. A one-dimensional fully compositional reservoir simulator using a flash calculation was developed using MATLAB. The model was based on conservation of mass, and many fluid property prediction models were incorporated to complete the model. All fluid property prediction models, including the flash calculation, were validated against known experimental data; and the reservoir flow was validated against analytical results for multiple types of boundary conditions.

The reservoir model properties were developed in order to simulate Hibernia reservoir oil. Three different injection gases were used to study the effects of miscibility on the huff 'n' puff process. Slim tube simulations were run to examine these miscibility effects. These slim tube simulations proved that the developed model could simulate multi-component gas injection into the multi-component reservoir oil. The recovery profiles of these slim tube simulations provided valuable information, and were used to determine the MMP of the various injection gases with the reservoir oil.

The three injection gases were then used to evaluate the natural gas huff 'n' puff process. During the huff phase, the simulator appeared to accurately model reservoir behaviour. The near-well and full field pressure profiles were what would be expected for a radial model, and seemed to validate the choice of boundary conditions and reservoir grid properties. The gas injection mimicked what was expected for each of the various miscibility cases. For each injection gas the various changes in phase saturations could be explained by physical phenomenon that were modelled in the

simulator. Each component flowed as expected, and due to the lighter hydrocarbon components entering the reservoir oil, the oil viscosity was seen to reduce accordingly. The shut-in phase also appeared to be modelled correctly. The gas front did not travel much further after gas injection was stopped, and pressure in the system leveled out as expected. The puff phase of the simulation was where the simulator seemed to meet its limitations in modelling the huff 'n' puff process. The production of the gas phase seemed accurate, as the gas composition is reduced to the critical gas saturation where it would no longer flow. This behaviour is mirrored in the composition plots. It is the production of oil and water that stops the simulator from allowing a full simulation of the huff 'n' puff process. With each injection gas, the oil saturation is reduced down to residual oil which essentially shuts off oil flow to the reservoir. When the oil saturation reaches residual oil, reduction of oil flow to zero mirrors the physical phenomenon expected, but does not allow for a true simulation of the huff 'n' puff process. The results of the simulations suggest that the simulator is accurately modelling reservoir behaviour, but the limitations of the model prevent the simulator from adequately modelling the huff 'n' puff process. The main limitation of the developed simulator in modelling the huff 'n' puff process appears to be that the complex production phase of the three-phase huff 'n' puff process cannot be accurately modelled in one-dimension.

Although it was shown that the three-phase huff 'n' puff process could not be accurately modelled in this one-dimensional simulator using radial geometry, the model which was developed can be built upon and has the opportunity for a wide variety of future work.

5.2 Recommendations

The work completed during this thesis has created a great opportunity for future work in reservoir simulation. The model as it currently stands has been validated on many levels and has shown the

ability to simulate waterflooding, slim tube experiments, and gas injection in one dimension (using radial and Cartesian co-ordinates). Some future works which could build onto this work are:

- Perform a one-dimensional huff ‘n’ puff experiment using a micro-model in order to determine if the process can be performed properly in only one dimension.
- Extend this model into two or three dimensions, this will likely allow for proper simulation of the huff ‘n’ puff process, and potentially resolve issues experienced in this work.
- Add relative permeability hysteresis to the model. This will allow for a more accurate simulation of the huff ‘n’ puff process as well as other EOR processes.
- Add a dispersion model to further study the interaction between injection gas and formation oil during shut-in phase.
- Improve water phase modelling to incorporate EOR recovery processes such as carbonated water injection into the model.
- Simulate different types of EOR processes. With the Cartesian and radial gridding built into the model, it allows for a wide variety of different processes to be simulated; waterflooding, gas injection, and WAG are all possibilities to be simulated with this model.
- Compare the model against experimental data (i.e. a slim-tube experiment). This model will allow for easy testing of any new correlations developed for fluid property prediction.

Bibliography

Abou-Kassem, Jamal H. Farouq Ali, S.M. Islam, M. Rafig. (2006). *Petroleum Reservoir Simulation - A Basic Approach*. Gulf Publishing Company.

Asghari, K., & Torabi, F. (2007). Laboratory experimental results of huff-and-puff CO2 flooding in a fractured core system. *Society of Petroleum Engineers*. doi:10.2118/110577-MS

Aziz, K., & Settari, A. (1979). *Petroleum reservoir simulation.* Applied Science Publishers.

Barclay, T. H., & Mishra, S. (2016). New correlations for CO2-oil solubility and viscosity reduction for light oils. *Journal of Petroleum Exploration and Production Technology*. doi:1001.1007/s13202-016-0233-y

Bardon, C., Corlay, P., Longeron, D., & Miller, B. (1994). CO2 huff 'n' puff revives shallow light-oil-depleted reservoirs. *Society of Petroleum Engineers*. doi:10.2118/22650-PA

Chen, Z., Huan, G., & Ma, Y. (2006). *Computational methods for multiphase flows in porous media.* Society for Industrial and Applied Mathematics.

Courant, R., Friedrichs, K., & Lewy, H. (1967). On the partial difference equations of mathematical physics. *IBM J.Res.Dev., 11*(2), 215-234. doi:10.1147/rd.112.0215

Danesh, A. (1998). *PVT and phase behaviour of petroleum reservoir fluids.* Elsevier Science.

Denoyelle, L. C., & Lemonnier, P. (1987). Simulation of CO2 huff ‘n’ puff using relative permeability hysteresis. *Society of Petroleum Engineers*. doi:10.2118/16710-MS

ExxonMobil (2011). *Hebron project development plan.* C-NLOPB.

Ha, G. T. et al. (2012*).* Design & implementation of CO2 huff-n-puff operation in a vietnam offshore field. *Society of Petroleum Engineers*. doi:10.2118/161835-MS

Haines, H. K., & Monger, T. G. (1990). A laboratory study of natural gas huff `n' puff. *Society of Petroleum Engineers.* doi:10.2118/21576-MS

Hara, S. K., & Christman, P. G. (1993). Investigation of a cyclic countercurrent light-Oil/CO2 immiscible process. *Society of Petroleum Engineers.* doi:10.2118/20207-PA

Haskin, H. K., & Alston, R. B. (1989). An evaluation of CO2 huff 'n' puff tests in Texas. *Society of Petroleum Engineers*. doi:10.2118/15502-PA

Holmes, M. (2013). *tridiag.m* (Version 1.0) [Computer program]. Available at http://www.mathworks.com/matlabcentral/fileexchange/40722-tridiag-m?focused=3780252&tab=function (Accessed 10 August 2015)

Hsu, H., & Brugman, R. J. (1986). CO2 huff-puff simulation using a compositional reservoir simulator. *Society of Petroleum Engineers.* doi:10.2118/15503-MS

Johansen, T. (2008). *Principles of reservoir engineering.*

Karim, F., Berzins, T. V., Schenewerk, P. A., Bassiouni, Z. A., & Wolcott, J. M. (1992). Light oil recovery from cyclic CO2 injection: Influence of drive gas, CO2 injection rate, and reservoir dip. *Society of Petroleum Engineers.* doi:10.2118/24336-MS

Kazemi, H., Vestal, C. R., & Shank, D. G. (1978). An efficient multicomponent numerical simulator. *Society of Petroleum Engineers.* doi:10.2118/6890-PA

Keith, P. C. (1969). Combination methods involving the making of gaseous carbon dioxide and its use in crude oil recovery. Retrieved from http://www.google.co.in/patents/US3442332

Khatib, A. K., Earlougher, R. C., & Kantar, K. (1981). CO2 injection as an immiscible application for enhanced recovery in heavy oil reservoirs. *Society of Petroleum Engineers.* doi:10.2118/9928-MS

Liu, H., Wang, M. C., Zhou, X., & Zhang, Y. P. (2005). EOS simulation for CO2 huff-n-puff process. *Petroleum Society of Canada.* doi:10.2118/2005-120

Lohrenz, J., Bray, B. G., & Clark, C. R. (1964). Calculating viscosities of reservoir fluids from their compositions. *Society of Petroleum Engineers.* doi:10.2118/915-PA

Miller, B. J. (1990). Design and results of a shallow, light oilfield-wide application of CO2 huff 'n' puff process. *Society of Petroleum Engineers.* doi:10.2118/20268-MS

Mohammed-Singh, L., Singhal, A. K., & Sim, S. S. (2006). Screening criteria for CO2 huff 'n' puff operations. *Society of Petroleum Engineers.* doi:10.2118/100044-MS

Monger, T. G., & Coma, J. M. (1988). A laboratory and field evaluation of the CO2 huff 'n' puff process for light-oil recovery. *Society of Petroleum Engineers.* doi:10.2118/15501-PA

Monger, T. G., Ramos, J. C., & Thomas, J. (1991). Light oil recovery from cyclic CO2 injection: Influence of low pressures impure CO2, and reservoir gas. *Society of Petroleum Engineers.* doi:10.2118/18084-PA

Nghiem, L. X., Fong, D. K., & Aziz, K. (1981). Compositional modeling with an equation of state (includes associated papers 10894 and 10903). *Society of Petroleum Engineers.* doi:10.2118/9306-PA

Olenick, S., Schroeder, F. A., Haines, H. K., & Monger-McClure, T. (1992). Cyclic CO2 injection for heavy-oil recovery in halfmoon field: Laboratory evaluation and pilot performance. *Society of Petroleum Engineers.* doi:10.2118/24645-MS

Palmer, F. S., Landry, R. W., & Bou-Mikael, S. (1986). Design and implementation of immiscible carbon dioxide displacement projects (CO2 huff-puff) in south louisiana. *Society of Petroleum Engineers.* doi:10.2118/15497-MS

Patton, J. T., Coats, K. H., & Spence, K. (1982). Carbon dioxide well stimulation: Part 1-A parametric study. *Society of Petroleum Engineers.* doi:10.2118/9228-PA

Peng, D., & Robinson, D. B. (1976). A new two-constant equation of state. *Industrial & Engineering Chemistry Fundamentals, 15*(1), 59-64. doi:10.1021/i160057a011

Poettmann, F. H., & Katz, D. L. V. (1945). *Phase behavior of binary carbon dioxide - paraffin systems.* University of Michigan.

Qazvini Firouz, A., & Torabi, F. (2012). Feasibility study of solvent-based huff-n-puff method (cyclic solvent injection) to enhance heavy oil recovery. *Society of Petroleum Engineers.* doi:10.2118/157853-MS

Rachford, H. H., & Rice, J. D. (1952). Procedure for use of electrical digital computers in calculating flash vaporization hydrocarbon equilibrium. *Journal of Petroleum Technology*.

Sayegh, S. G., & Maini, B. B. (1984). Laboratory evaluation of the CO huff-N-puff process for heavy oil reservoirs. *Society of Petroleum Engineers*. doi:10.2118/84-03-02

Schlumberger. (2014). *ECLIPSE technical description* (2014 1st ed.)

Shayegi, S., Jin, Z., Schenewerk, P., & Wolcott, J. (1996). Improved cyclic stimulation using gas mixtures. *Society of Petroleum Engineers.* doi:10.2118/36687-MS

Thomas, G. A., & Monger-McClure, T. (1991). Feasibility of cyclic CO2 injection for light-oil recovery. *Society of Petroleum Engineers.* doi:10.2118/20208-PA

Thomas, J., Berzins, T. V., Monger, T. G., & Bassiouni, Z. A. (1990). Light oil recovery from cyclic CO2 injection: Influence of gravity segregation and remaining oil. *Society of Petroleum Engineers.* doi:10.2118/20531-MS

Wang, Z., Ma, J., Gao, R., Zeng, F., Huang, C., Tontiwachwuthikul, P., et al. (2013). Optimizing cyclic CO2 injection for low- permeability oil reservoirs through experimental study. *Society of Petroleum Engineers.* doi:10.2118/167193-MS

Welge, H. J. (1952). A Simplified Method for Computing Oil Recovery by Gas or Water Drive. *Society of Petroleum Engineers*. doi:10.2118/124-G

Welker, J. R. (1963). Physical Properties of Carbonated Oils. *Society of Petroleum Engineers*. doi:10.2118/567-PA

Wilson, G. (1968). A modified redlich-kwong equation of state, application to general physical data calculations. *Presented at AIChE National Meeting,*

Whitson, C. H., Brulé, M. R., & Society of, P. E. (2000). *Phase behavior*. Henry L. Doherty Memorial Fund of AIME, Society of Petroleum Engineers.

Yang, H. (2014). *A study of water and carbonated water injection with constant pressure Boundaries.* Memorial University of Newfoundland

Zhang, Y. P., Sayegh, S. G., Huang, S., & Dong, M. (2006). Laboratory investigation of enhanced light-oil recovery by CO/Flue gas huff-n-puff process. *Society of Petroleum Engineers.* doi:10.2118/06-02-01

Appendix A – Reservoir Simulator Code

```
% ------------------------------------------------------------------------ %
% main.m executes the reservoir simulator
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

close all
clear variables
clc

format long

tic

%% Reservoir Information
%% Rock Properties
phi0 = 0.18;
perm0 = 500*(9.869233E-16); % [m^2]
cp = 0; %[Pa^-1]
D = 1000;
g = 9.81; % [m/s^2]

%% Relative Permeability
% Relative Permeability Type (Correlation / Tabular)
RelPermtype = 'Correlation';
% Critical Saturations
Swc = 0.18;
Sorw = 0.20;
Sorg = 0.15;
Sgc = 0.03;
% Maximum relative permeabilties
krowmax = 1;
krogmax = 1;
krwmax = 0.3;
krgmax = 0.9;
% Corey model exponents
nw = 2;
now = 2;
nog = 2;
ng = 2;
% Tabular
Swtab = [0    0.1  0.2  0.25 0.3  0.35 0.4  0.45 0.5  ...
         0.55 0.6  0.65 0.7  0.75 0.8  0.9  1];
krwtab = [0     0     0     0.002 0.009 0.02  0.033 0.051 0.075 ...
          0.1   0.132 0.17  0.208 0.251 0.3   0.3   0.3];
krowtab = [0.8    0.8    0.8    0.61  0.47  0.37  0.285 0.22  0.163 ...
           0.12   0.081  0.05   0.027 0.01  0     0     0];
Sgtab = [0 0.05 0.2 0.5 0.6 0.82 1];
krgtab = [0 0.022 0.11 0.44 0.56 1 1];
krogtab = [0.7 0.62 0.12 0 0 0 0];

%% Capillary Pressure
% Capillary Pressure Type (Zero / Correlation / Tabular)
```

```
Pcowtype = 'Zero';
% Might be needed
dPcowdSw = 0.00E6;
% Still need junk to fill in for tabular
Sw2tab= [0 .1 .2 .25 .3 .4 .5 .6 .7 .8 .9 1];
Pcowtab= [110304 110304 110304 59288.4 41364 24542 16683 10892 6549 0 0 0];

%% Fluid Properties
%% Reservoir Fluid / PVT
% Initial reservoir fluid composition
comp = {{'N2'}, {'CO2'},{'C1'} ,{'C2'} ,{'C3'} ,{'iC4'}, ...
        {'nC4'},{'iC5'},{'nC5'},{'nC6'},{'C7+'}};
z0 = [0.0031,0.0082,0.4149,0.0467,0.046 ,0.0082, ...
      0.0217,0.0097,0.0126,0.0333,0.3956];
Nc = size(z0, 2);
% C7+ Information
Tc7=760.37; % [K]
Pc7=1.602; % [MPa]
MW7=261.34; % [g/mol]
vc7=1.09;
SE7=1-(2.258/(MW7^0.1823));
w7=0.8429;
pchor7=0.324*(Tc7^(1/4))*(vc7^(7/8));
%% Water
% Salinity
ws = 0.1;
% Base density, molar density, viscosity
densw0 = 1000; % [kg/m^3]
moldensw0=55.6E3; % [mol/m^3]
viscw0 = 0.001; % [Pa-s]

%% Grid
% Co-ordinate system and grid block type
coordsys = 'Radial';
gridtype = 'BCG';
% Grid Size (Must Include Both)
% Cartesian / Radial (for Radial xinlet cannot be zero)
xinlet = 1; % [m]
xoutlet = 4000; % [m]
dy = 15; % [m]
dz = 5; % [m]
% Number of Gridblocks
numG = 60;
% Slim Tube Radius
STr = 0.0063; % [m]
% Grid Property Calculations
[grid,gridHalf,A,dz,Vr] = ...
GridProps(numG,xinlet,xoutlet,coordsys,gridtype,dy,dz,STr);
porev = phi0*sum(Vr(2:numG+1));

%% Schedule
% Injection Phase and Composition
process = 'Huff Puff';
injphase = 'g';
zin = [0 0 0.9 0.05 0.025 0.0125 0.0125 0 0 0 0];
```

```
% Used to ensure all components in reservoir fluid and injection gas
% are included for fluid properties
for k = 1:Nc
    z_props{k} = z0(k) + zin(k);
end
[z_tot,w,Tc,Pc,MW,vc,SE,pchor,Nc,Ncg,num_gr,z_gr] = ...
FluidProps(comp,z_props,w7,Tc7,Pc7,MW7,vc7,SE7,pchor7);
Kbin = BIP(z_tot,Nc,Ncg,num_gr,z_gr);

% Injection/ Production Rates and Pressures
% Rates
Qin = 0.00184; % [m^3/s]
Qout = -Qin; % [m^3/s]
% Pressures
pbh = 17E6; %[Pa]
pe = 45E6; %[Pa]

% Well types (Boundary Conditions)
welltypein = 'Rate';
welltypeout = 'Pressure';

% Simulation time
simulationtime = 24*3600;
% Shut-in and Production times (Huff 'n' Puff)
shuttime = 8*3600; % [s]
prodtime = shuttime + 2*3600; % [s]

% Time Step and calculations for iteration;
tstep_inj = 60; % [s]
tstep_shut = 60; %[s]
tstep_prod = 5; % [s]
% Calculations for iteration

if(strcmp(process,'Huff Puff') && simulationtime > shuttime && ...
   simulationtime > prodtime)

    total_inj_time = shuttime;
    total_shut_time = prodtime - shuttime;
    total_prod_time = simulationtime - prodtime;

    numtsteps = floor(total_inj_time/tstep_inj + ...
                      total_shut_time/tstep_shut + ...
                      total_prod_time/tstep_prod);
    startshut = floor(shuttime/tstep_inj);
    startprod = startshut + floor((prodtime - shuttime)/tstep_prod);
else
    total_inj_time = simulationtime;
    numtsteps = floor(total_inj_time/tstep_inj);
    startshut = numtsteps + 1;
    startprod = startshut +1;
end

%% Reporting
% Time Interval for reporting
interval = tstep_inj;
```

```
% Grid Block for plotting (Useful in Huff 'n' Puff)
plotblock = numG;

%% Initial Conditions
% Initial Pressure / Temperature
p0 = 45E6; %[Pa]
T = 373.15; %[K]
R = 8.3145; %[(Pa*m^3)/(mol K)]
pref = p0; % used for converting pd to p
% Initial Saturations
Sw0 = 0.75;
%% Simulation
%% Iteration control
maxNRiter = 25;
tolNR = 10E-6; tolMBE = 10E-9;
%% (MATLAB) Initialize Variables
% Time Variables
simtime=0;
runtime=0;
% Injection / Production
% Pore Volumes
PVinj = 0;
PVprod = 0;
% Moles
nInj = zeros(1,Nc);
nProd = zeros(1,Nc);
nInit = zeros(numG + 2, Nc);
% Water
waterInj = 0;
waterProd = 0;
waterInit = zeros(1,numG + 2);
%% Simulation Variables
%% Regular
p = zeros(1,numG + 2); phi = zeros(1,numG + 2); perm = zeros(1,numG + 2);
z = zeros(numG + 2, Nc);
xo = zeros(numG + 2, Nc); xg = zeros(numG + 2, Nc);
K = zeros(numG + 2, Nc); L = zeros(1,numG + 2); V = zeros(1,numG + 2);
Zo = zeros(1,numG + 2); Zg = zeros(1,numG + 2);
dmoldensodp = zeros(1,numG + 2); dmoldensgdp = zeros(1,numG + 2);
volo = zeros(1,numG + 2); volg = zeros(1,numG + 2);
Sw = zeros(1,numG + 2); So = zeros(1,numG + 2); Sg = zeros(1,numG + 2);
Tw = zeros(1,numG + 2); To = zeros(1,numG + 2); Tg = zeros(1,numG + 2);
krw = zeros(1,numG + 2); kro = zeros(1,numG + 2); krg = zeros(1,numG + 2);
viscw = zeros(1,numG + 2); visco = zeros(1,numG + 2);
viscg = zeros(1,numG + 2);
densw = zeros(1,numG + 2); denso = zeros(1,numG + 2);
densg = zeros(1,numG + 2);
moldensw = zeros(1,numG + 2); moldenso = zeros(1,numG + 2);
moldensg = zeros(1,numG + 2);
cw = zeros(1,numG + 2); Pcow = zeros(1,numG + 2);
Pcog = zeros(1,numG +2); ogIFT = zeros(1,numG + 2);
psi = zeros(1,numG + 2);
vk =  zeros(numG + 2, Nc); nHuff = zeros(1,Nc); nPuff = zeros(1,Nc);
%% Newton-Raphson Iteration
p_NR = zeros(maxNRiter,numG + 2); phi_NR = zeros(maxNRiter,numG + 2);
perm_NR = zeros(maxNRiter,numG + 2);
```

```
z_NR = zeros(numG + 2, Nc, maxNRiter);
xo_NR = zeros(numG + 2, Nc, maxNRiter);
xg_NR = zeros(numG + 2, Nc, maxNRiter);
K_NR = zeros(numG + 2, Nc, maxNRiter);
L_NR = zeros(maxNRiter,numG + 2); V_NR = zeros(maxNRiter,numG + 2);
Zo_NR = zeros(maxNRiter,numG + 2); Zg_NR = zeros(maxNRiter,numG + 2);
dmoldensodp_NR = zeros(maxNRiter,numG + 2);
dmoldensgdp_NR = zeros(maxNRiter,numG + 2);
volo_NR = zeros(maxNRiter,numG + 2); volg_NR = zeros(maxNRiter,numG + 2);
Sw_NR = zeros(maxNRiter,numG + 2); So_NR = zeros(maxNRiter,numG + 2);
Sg_NR = zeros(maxNRiter,numG + 2);
Tw_NR = zeros(maxNRiter,numG + 2); To_NR = zeros(maxNRiter,numG + 2);
Tg_NR = zeros(maxNRiter,numG + 2);
Twplus_NR = zeros(maxNRiter,numG + 2);
Toplus_NR = zeros(maxNRiter,numG + 2);
Tgplus_NR = zeros(maxNRiter,numG + 2);
Twminus_NR = zeros(maxNRiter,numG + 2);
Tominus_NR = zeros(maxNRiter,numG + 2);
Tgminus_NR = zeros(maxNRiter,numG + 2);
Tplus_NR = zeros(maxNRiter,numG + 2);
Tminus_NR = zeros(maxNRiter,numG + 2);
krw_NR = zeros(maxNRiter,numG + 2); kro_NR = zeros(maxNRiter,numG + 2);
krg_NR = zeros(maxNRiter,numG + 2);
viscw_NR = zeros(maxNRiter,numG + 2); visco_NR = zeros(maxNRiter,numG + 2);
viscg_NR = zeros(maxNRiter,numG + 2);
densw_NR = zeros(maxNRiter,numG + 2); denso_NR = zeros(maxNRiter,numG + 2);
densg_NR = zeros(maxNRiter,numG + 2);
moldensw_NR = zeros(maxNRiter,numG + 2);
moldenso_NR = zeros(maxNRiter,numG + 2);
moldensg_NR = zeros(maxNRiter,numG + 2);
cw_NR = zeros(maxNRiter,numG + 2); Pcow_NR = zeros(maxNRiter,numG + 2);
ogIFT_NR = zeros(maxNRiter,numG + 2); psi_NR = zeros(maxNRiter,numG + 2);
dir_NR = zeros(maxNRiter,numG + 2); Gin_NR = zeros(1,maxNRiter);
Gout_NR = zeros(1,maxNRiter);
Qw_NR = zeros(maxNRiter,numG + 2); Qo_NR = zeros(maxNRiter,numG + 2);
Qg_NR = zeros(maxNRiter,numG + 2);
qw_NR = zeros(maxNRiter,numG + 2); qo_NR = zeros(maxNRiter,numG + 2);
qg_NR = zeros(maxNRiter,numG + 2);
WIw_NR = zeros(maxNRiter,numG + 2); WIo_NR = zeros(maxNRiter,numG + 2);
WIg_NR = zeros(maxNRiter,numG + 2);
qk_NR = zeros(numG + 2, Nc, maxNRiter);
moldenswellw_NR = zeros(maxNRiter,numG + 2);
moldenswello_NR = zeros(maxNRiter,numG + 2);
moldenswellg_NR = zeros(maxNRiter,numG + 2);
waterNew_NR = zeros(maxNRiter,numG + 2);
totalWaterNew_NR = zeros(1,maxNRiter); waterInj_NR = zeros(1,maxNRiter);
waterProd_NR = zeros(1,maxNRiter);
MBEw_NR = zeros(1,maxNRiter); nNew_NR = zeros(numG + 2, Nc, maxNRiter);
totalnNew_NR = zeros(maxNRiter,Nc); nInj_NR = zeros(maxNRiter,Nc);
nProd_NR = zeros(maxNRiter,Nc); MBEn_NR = zeros(maxNRiter,Nc);

dpsiphidp_NR = zeros(maxNRiter,numG + 2);
dqwdp_NR = zeros(maxNRiter,numG + 2); dqodp_NR = zeros(maxNRiter,numG + 2);
dqgdp_NR = zeros(maxNRiter,numG + 2);
R_NR = zeros(maxNRiter,numG + 2);
a = zeros(1,numG+2); d = zeros(1,numG+2);
```

```
b = zeros(1,numG+1); c = zeros(1,numG+1);
iter = zeros(1,numtsteps); errorp = zeros(1,numtsteps);
errorMBEw = zeros(1,numtsteps);
changep = zeros(1,numG + 2); maxchangep = zeros(1, maxNRiter);
errorMBEz = zeros(numG + 2, Nc, numtsteps);
errormaxMBEz = zeros(1,numtsteps); errorF = zeros(1,numtsteps);
%% Reporting
rstep = 0;
numrsteps = ceil(simulationtime / interval);
p_r = zeros(numrsteps,numG + 2); z_r = zeros(numG + 2, Nc,numrsteps);
xo_r = zeros(numG + 2, Nc,numrsteps); xg_r = zeros(numG + 2, Nc,numrsteps);
Sw_r = zeros(numrsteps,numG + 2); So_r = zeros(numrsteps,numG + 2);
Sg_r = zeros(numrsteps,numG + 2);
krw_r = zeros(numrsteps,numG + 2); kro_r = zeros(numrsteps,numG + 2);
krg_r = zeros(numrsteps,numG + 2);
viscw_r = zeros(numrsteps,numG + 2); visco_r = zeros(numrsteps,numG + 2);
viscg_r = zeros(numrsteps,numG + 2);

%% Initialize Model
% Rock Properties
phi(2:numG+1) = phi0;
perm(2:numG+1) = perm0;

% Pressure and Saturations
p(1:numG+2) = p0;
Sw(2:numG+1) = Sw0;
z(2:numG+1,:) = repmat(z0,numG,1);

% Fluid Properties
for i = 2:numG+1
    % Hydrocarbon Properties
    [xo(i,:),xg(i,:),L(i),V(i),K(i,:),Zo(i),Zg(i), ...
    dmoldensodp(i),dmoldensgdp(i),volo(i),volg(i), ...
    moldenso(i),moldensg(i),denso(i),densg(i)] = ...
    PRFlash(Nc,z(i,:),R,w,Tc,Pc,MW,Kbin,T,p(i),SE);

    [visco(i),viscg(i)] = ...
    LBCVisc(Nc,xo(i,:),xg(i,:),Tc,Pc,MW,T,vc,volo(i),volg(i));
    [ogIFT(i)] = ...
    MacSugIFT(Nc,xo(i,:),xg(i,:),pchor,moldenso(i),moldensg(i));
    % Water Properties
    [cw(i),viscw(i),densw(i),moldensw(i)] = ...
    WaterProps(T,p(i),ws,moldensw0,densw0,p0);
    % Saturations
    if Sw(i) == 1
        So(i) = 0;
        Sg(i) = 0;
    else
        So(i)=((1-Sw(i))*moldensg(i)*L(i)) / ...
              (L(i)*moldensg(i)+V(i)*moldenso(i));
        Sg(i)=1-Sw(i)-So(i);
    end
end
% Initial Fluid in place (moles)
for i = 2:numG+1
    waterInit(i) = Vr(i)*phi(i)*moldensw(i)*Sw(i);
```

```
    for k = 1:Nc
        nInit(i,k) = Vr(i)*phi(i)*(moldenso(i)*So(i) + ...
                     moldensg(i)*Sg(i))*z(i,k);
    end
end
totalWaterInit = sum(waterInit);
totalnInit = sum(nInit);
% Because it is time = 0
waterNew = waterInit; totalWaterNew = totalWaterInit;
nNew = nInit; totalnNew = totalnInit;
URF = 0;
% Calculate theta (Water conversion parameter)
if Sw0 == 1
    theta = 0.2;
else
    mid = floor(numG/2);
    theta=(moldenso(mid)*So(mid)+moldensg(mid)*Sg(mid)) / ...
          (moldensw(mid)*(So(mid)+Sg(mid)));
end

% Rel Perms, Capillary Pressures, Accumulation term, Transmissibilities
for i = 2:numG+1
    % Relative Permeabilities
    [krw(i),kro(i),krg(i)] = RelPerm(krwmax,krowmax,krogmax,krgmax, ...
    Sw(i),So(i),Sg(i),Swc,Sorw,Sorg,Sgc,nw,now,nog,ng,RelPermtype, ...
    Swtab,krwtab,krowtab,Sgtab,krgtab,krogtab);
    % Capillary Pressures
    [Pcow(i)] = WaterCapillary(Sw(i),Sw2tab,Pcowtab,Pcowtype);
    % Accumulation term
    psi(i) = theta*moldensw(i)*Sw(i)+moldenso(i)*So(i)+moldensg(i)*Sg(i);
    % Transmissibilities
    Tw(i) = (perm(i)*krw(i)*moldensw(i))/(viscw(i));
    To(i) = (perm(i)*kro(i)*moldenso(i))/(visco(i));
    Tg(i) = (perm(i)*krg(i)*moldensg(i))/(viscg(i));
end
% Directional Transmissibilities
[dir,Gin,Gout,Twplus,Twminus,Toplus,Tominus,Tgplus,Tgminus, ...
Tplus,Tminus,Gplus,Gminus] = DirectionalTrans(p,numG,Tw,To,Tg,grid, ...
                             gridHalf,coordsys,A,dz,theta,1,startprod);
% Well Terms
[Qw,Qo,Qg,qw,qo,qg,qk,WIw,WIo,WIg, ...
moldenswellw,moldenswello,moldenswellg,p] = ...
WellModels(1,startshut,startprod,injphase,numG,R,Nc,zin,w,Tc,Pc,MW, ...
Kbin,T,p,SE,krwmax,krowmax,krogmax,krgmax,Swc,Sorw,Sorg,Sgc,nw,now,nog, ...
ng,RelPermtype,Swtab,krwtab,krowtab,Sgtab,krgtab,krogtab,viscw,visco, ...
viscg,densw0,moldensw0,moldensw,moldenso,moldensg,xo,xg,pe,pbh, ...
Gin,Gout,vc,krw,kro,krg,welltypein,welltypeout,Qin,Qout,z0,Sw0,ws,p0,perm);


%% Run iterative simulation

for n=1:numtsteps
    % Simulation phase
    if n == 1
        t = tstep_inj;
        phase = 'Injection';
```

```
    elseif n == startshut
        t = tstep_shut;
        phase = 'Shut-in';
    elseif n == startprod
        t = tstep_prod;
        phase = 'Production';
    end
    % Display Simulation Status
    fprintf('Current phase: %s. The iteration number is %d out of %d using a
time step of %f s, and %f s of simulation time have passed. Run time has been
%f s.\n',phase,n,numtsteps,t,simtime,runtime);

    if n == 1
        % Used to write initial conditions to report
    elseif n == startprod
        welltypein = 'Pressure';
        pbh = p0 - 1E6;
        [Qw,Qo,Qg,qw,qo,qg,qk,WIw,WIo,WIg, ...
        moldenswellw,moldenswello,moldenswellg,p] = ...
        WellModels(n,startshut,startprod,injphase,numG,R,Nc,zin,w,Tc, ...
        Pc,MW,Kbin,T,p,SE,krwmax,krowmax,krogmax,krgmax,Swc,Sorw,Sorg, ...
        Sgc,nw,now,nog,ng,RelPermtype,Swtab,krwtab,krowtab,Sgtab, ...
        krgtab,krogtab,viscw,visco,viscg,densw0,moldensw0,moldensw, ...
        moldenso,moldensg,xo,xg,pe,pbh,Gin,Gout,vc,krw,kro,krg, ...
        welltypein,welltypeout,Qin,Qout,z0,Sw0,ws,p0,perm);

        [dir,Gin,Gout,Twplus,Twminus,Toplus,Tominus,Tgplus,Tgminus, ...
        Tplus,Tminus,Gplus,Gminus] = DirectionalTrans(p,numG,Tw,To,Tg, ...
                                    grid,gridHalf,coordsys,A,dz,theta, ...
                                    n,startprod);
        t = tstep_prod;
    else

    % Reset iteration variables
    l = 0;
    endNR = 0;
    deltapd=zeros(1,numG+2);
    deltap=zeros(1,numG+2);
    error = 2*tolNR;
    %% Newton Raphson Iteration
    while 1==1
        l=l+1;
        %% Update Parameters
        if l == 1
            % Set variables at old time level
            p_NR(l,:) = p;
            z_NR(:,:,l) = z;
            dir_NR(l,:) = dir;
        else

            % Update Pressure
            [Lsolve,Usolve]=lu(A_matrix); % use LU factorization for speed
            deltapd = Usolve\(Lsolve\d(2:numG+1)');
            deltap(2:numG+1) = deltapd*pref;
```

```
    for i=1:numG+2
        p_NR(l,i) = p_NR(l-1,i)+deltap(i);
    end

    % Check for convergence
    if l > 2 % Minimum of 2 iterations
        for i=2:numG+1
            changep(i) = (p_NR(l,i)-p_NR(l-1,i))/p_NR(l-1,i);
        end
        maxchangep(l) = max(abs(changep));

        if ((l >= maxNRiter) || (maxchangep(l) < tolNR))
            endNR = l;
        end
    end

    % Update flow rates with new pressures
    [Qw,Qo,Qg,qw,qo,qg,qk,WIw,WIo,WIg,moldenswellw, ...
     moldenswello,moldenswellg,p] = ...
    WellModels(n,startshut,startprod,injphase,numG,R,Nc,zin, ...
    w,Tc,Pc,MW,Kbin,T,p_NR(l,:),SE,krwmax,krowmax,krogmax, ...
    krgmax,Swc,Sorw,Sorg,Sgc,nw,now,nog,ng,RelPermtype,Swtab, ...
    krwtab,krowtab,Sgtab,krgtab,krogtab,viscw,visco,viscg, ...
    densw0,moldensw0,moldensw,moldenso,moldensg,xo,xg,pe,pbh, ...
    Gin,Gout,vc,krw,kro,krg,welltypein,welltypeout,Qin,Qout,z0, ...
    Sw0,ws,p0,perm);

    % Directional Transmissibilities (Update direction, use
    % values from old time level)
    [dir,Gin,Gout,Twplus,Twminus,Toplus,Tominus,Tgplus, ...
     Tgminus,Tplus,Tminus,Gplus,Gminus] = ...
    DirectionalTrans(p_NR(l,:),numG,Tw,To,Tg,grid,gridHalf,...
                     coordsys,A,dz,theta,n,startprod);

    % Update Compositions
    z_NR(:,:,l) = UpdateZ(phi,z,xo,xg,moldenso,moldensg,So,Sg, ...
                          Toplus,Tominus,Tgplus,Tgminus, ...
                          p_NR(l,:),dir,Nc,t,numG,Vr, qo,qg,qk);

end

% Update Rock Properties
phi_NR(l,2:numG+1) = phi0;
perm_NR(l,2:numG+1) = perm0;
% Update Fluid Properties
for i = 2:numG+1
    % Hydrocarbon Properties
    [xo_NR(i,:,l),xg_NR(i,:,l),L_NR(l,i),V_NR(l,i),K_NR(i,:,l), ...
     Zo_NR(l,i),Zg_NR(l,i),dmoldensodp_NR(l,i), ...
     dmoldensgdp_NR(l,i),volo_NR(l,i),volg_NR(l,i), ...
     moldenso_NR(l,i),moldensg_NR(l,i),denso_NR(l,i), ...
     densg_NR(l,i)] = ...
    PRFlash(Nc,z_NR(i,:,l),R,w,Tc,Pc,MW,Kbin, T,p_NR(l,i),SE);

    [visco_NR(l,i),viscg_NR(l,i)] = ...
```

```
    LBCVisc(Nc,xo_NR(i,:,l),xg_NR(i,:,l),Tc,Pc,MW,T,vc, ...
            volo_NR(l,i),volg_NR(l,i));

    [ogIFT_NR(l,i)] = MacSugIFT(Nc,xo_NR(i,:,l),xg_NR(i,:,l), ...
                                pchor,moldenso_NR(l,i), ...
                                moldensg_NR(l,i));
    % Water Properties
    [cw_NR(l,i),viscw_NR(l,i),densw_NR(l,i),moldensw_NR(l,i)] = ...
    WaterProps(T,p_NR(l,i),ws,moldensw0,densw0,p0);
end
% Update Saturations
[Sw_NR(l,:),So_NR(l,:),Sg_NR(l,:)] = ...
UpdateSaturations(t,numG,Twminus,Twplus,p_NR(l,:),Pcow, ...
                  L_NR(l,:),V_NR(l,:),moldenso_NR(l,:), ...
                  moldensg_NR(l,:),moldensw_NR(l,:),moldensw, ...
                  phi_NR(l,:),phi,Sw,Vr,dir_NR(l,:),qw);

% Update Rel Perms, Capillary Pressures, Accumulation term, Trans
for i = 2:numG+1
    % Relative Permeabilities
    [krw_NR(l,i),kro_NR(l,i),krg_NR(l,i)] = ...
    RelPerm(krwmax,krowmax,krogmax,krgmax,Sw_NR(l,i), ...
            So_NR(l,i),Sg_NR(l,i),Swc,Sorw,Sorg,Sgc,nw,now,nog, ...
            ng,RelPermtype,Swtab,krwtab,krowtab,Sgtab,krgtab, ...
            krogtab);

    % Capillary Pressures
    [Pcow_NR(l,i)] = ...
    WaterCapillary(Sw_NR(l,i),Sw2tab,Pcowtab,Pcowtype);

    % Accumulation term
    psi_NR(l,i) = theta*moldensw_NR(l,i)*Sw_NR(l,i) ...
                  + moldenso_NR(l,i)*So_NR(l,i) ...
                  + moldensg_NR(l,i)*Sg_NR(l,i);

    % Transmissibilities
    Tw_NR(l,i) = (perm_NR(l,i)*krw_NR(l,i)*moldensw_NR(l,i)) / ...
                 (viscw_NR(l,i));
    To_NR(l,i) = (perm_NR(l,i)*kro_NR(l,i)*moldenso_NR(l,i)) / ...
                 (visco_NR(l,i));
    Tg_NR(l,i) = (perm_NR(l,i)*krg_NR(l,i)*moldensg_NR(l,i)) / ...
                 (viscg_NR(l,i));
end

% Directional Transmissibilities
[dir_NR(l,:),Gin_NR(l,:),Gout_NR(l,:),Twplus_NR(l,:), ...
 Twminus_NR(l,:),Toplus_NR(l,:),Tominus_NR(l,:),Tgplus_NR(l,:), ...
 Tgminus_NR(l,:),Tplus_NR(l,:),Tminus_NR(l,:)] = ...
DirectionalTrans(p_NR(l,:),numG,Tw_NR(l,:),To_NR(l,:), ...
                 Tg_NR(l,:),grid,gridHalf,coordsys,A,dz,theta, ...
                 n,startprod);
% Well Terms
[Qw_NR(l,:),Qo_NR(l,:),Qg_NR(l,:),qw_NR(l,:),qo_NR(l,:), ...
 qg_NR(l,:),qk_NR(:,:,l),WIw_NR(l,:),WIo_NR(l,:),WIg_NR(l,:), ...
 moldenswellw_NR(l,:),moldenswello_NR(l,:), ...
```

```
 moldenswellg_NR(l,:),p_NR(l,:)] = ...
WellModels(n,startshut,startprod,injphase,numG,R,Nc,zin,w,Tc, ...
           Pc,MW,Kbin,T,p_NR(l,:),SE,krwmax,krowmax,krogmax, ...
           krgmax,Swc,Sorw,Sorg,Sgc,nw,now,nog,ng,RelPermtype, ...
           Swtab,krwtab,krowtab,Sgtab,krgtab,krogtab, ...
           viscw_NR(l,:),visco_NR(l,:),viscg_NR(l,:),densw0, ...
           moldensw0,moldensw_NR(l,:),moldenso_NR(l,:), ...
           moldensg_NR(l,:),xo_NR(:,:,l),xg_NR(:,:,l),pe,pbh, ...
           Gin,Gout,vc,krw_NR(l,:),kro_NR(l,:),krg_NR(l,:), ...
           welltypein,welltypeout,Qin,Qout,z0,Sw0,ws,p0,perm);

%% Material Balance Check
% Water
for i = 2:numG+1
    waterNew_NR(l,i) = Vr(i)*phi_NR(l,i)*moldensw_NR(l,i) ...
                       *Sw_NR(l,i);
end
totalWaterNew_NR(l) = sum(waterNew_NR(l,:));
qwtemp = qw_NR(l,:);
waterInj_NR(l) = waterInj+sum(qwtemp(qwtemp>0)*t);
waterProd_NR(l) = waterProd+sum(qwtemp(qwtemp<0)*t);
MBEw_NR(l) = (totalWaterInit+waterInj_NR(l) + waterProd_NR(l) ...
             - totalWaterNew_NR(l))/totalWaterInit;
% Hydrocarbons
for k = 1:Nc
    for i = 2:numG+1
        nNew_NR(i,k,l) = Vr(i)*phi_NR(l,i)*(moldenso_NR(l,i) ...
                         * So_NR(l,i) ...
                         +moldensg_NR(l,i)*Sg_NR(l,i))*z_NR(i,k,l);
    end
    totalnNew_NR(l,k) = sum(nNew_NR(:,k,l));
    qktemp = qk_NR(:,k,l);
    nInj_NR(l,k) = nInj(k)+sum(qktemp(qktemp>0)*t);
    nProd_NR(l,k) = nProd(k)+sum(qktemp(qktemp<0)*t);
    MBEn_NR(l,k) = (totalnInit(k)+nInj_NR(l,k)+nProd_NR(l,k) ...
                   -totalnNew_NR(l,k))/totalnInit(k);
end

%% Convergence Check
% Check if convergence has been achieved
if l == endNR
    break;
end

%% Pressure Equation Setup
% Differentials
for i=2:numG+1
    % Accumulation term
    dpsiphidp_NR(l,i) = phi(i)*(theta*Sw_NR(l,i)*cw_NR(l,i) ...
    *moldensw0+So_NR(l,i)*dmoldensodp_NR(l,i)...
    + Sg_NR(l,i)*dmoldensgdp_NR(l,i));
    % Well terms
    dqwdp_NR(l,i)= -WIw_NR(l,i)*moldenswellw_NR(l,i);
    dqodp_NR(l,i)= -WIo_NR(l,i)*moldenswello_NR(l,i);
    dqgdp_NR(l,i)= -WIg_NR(l,i)*moldenswellg_NR(l,i);
end
```

```
    % Parameters for readability
    for i = 2:numG+1
        D_NR(l,i) = theta*Twplus(i)+Toplus(i)+Tgplus(i);
        E_NR(l,i) = theta*Twminus(i)+Tominus(i)+Tgminus(i);
        F_NR(l,i) = -D_NR(l,i)-E_NR(l,i);
    end

    % Residual Pressure Equation
    for i = 2:numG+1
        R_NR(l,i) =   E_NR(l,i)*p_NR(l,i-dir(i)) ...
                    + F_NR(l,i)*p_NR(l,i) ....
                    + D_NR(l,i)*p_NR(l,i+dir(i)) ....
                    - ( E_NR(l,i)*Pcow(i-dir(i)) ...
                    + F_NR(l,i)*Pcow(i) ...
                    + D_NR(l,i)*Pcow(i+dir(i)) ) ...
                    + ( E_NR(l,i)*Pcog(i-dir(i)) ...
                    + F_NR(l,i)*Pcog(i) ...
                    + D_NR(l,i)*Pcog(i+dir(i)) ) ...
                    + theta*qw_NR(l,i)+qo_NR(l,i)+qg_NR(l,i) ...
                    -(Vr(i)/t)*(phi_NR(l,i)*psi_NR(l,i)-phi(i)*psi(i));
    end

    % Set up Matrix
    for i=2:numG+1
        a(i) = (F_NR(l,i)+theta*dqwdp_NR(l,i)+dqodp_NR(l,i) ...
               + dqgdp_NR(l,i)-(Vr(i)/t)*dpsiphidp_NR(l,i))*pref;
        d(i) = -R_NR(l,i);
    end
    for i = 2:numG
        if dir(i) > 0
            b(i) = (E_NR(l,i+1))*pref;
            c(i) = (D_NR(l,i))*pref;
        elseif dir(i) < 0
            b(i) = (D_NR(l,i+1))*pref;
            c(i) = (E_NR(l,i))*pref;
        end
    end

    A_matrix = diag(a(2:numG+1))+diag(c(2:numG),1)+diag(b(2:numG),-1);

end

% Keep track of iterations and error from iteration
iter(n) = l;
errorp(n) = maxchangep(l); errorF(n) = max(abs(R_NR(l,:)));

%% Update reservoir at new time step
%% Update Variables
p = p_NR(l,:); phi = phi_NR(l,:); perm = perm_NR(l,:);
z = z_NR(:,:,l);
cw = cw_NR(l,:); Pcow = Pcow_NR(l,:);
xo = xo_NR(:,:,l); xg = xg_NR(:,:,l);
L = L_NR(l,:); V = V_NR(l,:);
volo = volo_NR(l,:); volg = volg_NR(l,:);
```

```
    moldensw = moldensw_NR(l,:); moldenso = moldenso_NR(l,:);
    moldensg = moldensg_NR(l,:);
    densw = densw_NR(l,:); denso = denso_NR(l,:); densg = densg_NR(l,:);
    viscw = viscw_NR(l,:); visco = visco_NR(l,:); viscg = viscg_NR(l,:);
    ogIFT = ogIFT_NR(l,:);
    Sw = Sw_NR(l,:); So = So_NR(l,:); Sg = Sg_NR(l,:);
    krw = krw_NR(l,:); kro = kro_NR(l,:); krg = krg_NR(l,:);
    Tw = Tw_NR(l,:); To = To_NR(l,:); Tg = Tg_NR(l,:);
    Twminus = Twminus_NR(l,:); Tominus = Tominus_NR(l,:);
    Tgminus = Tgminus_NR(l,:); Tminus = Tminus_NR(l,:);
    Twplus = Twplus_NR(l,:); Toplus = Toplus_NR(l,:);
    Tgplus = Tgplus_NR(l,:); Tplus = Tplus_NR(l,:);
    psi = psi_NR(l,:);

    dir = dir_NR(l,:);
    Qw = Qw_NR(l,:); Qo = Qo_NR(l,:); Qg = Qg_NR(l,:);
    qw = qw_NR(l,:); qo = qo_NR(l,:); qg = qg_NR(l,:);
    WIw =  WIw_NR(l,:); WIo = WIo_NR(l,:); WIg = WIg_NR(l,:);
    qk = qk_NR(:,:,l);
    moldenswellw = moldenswellw_NR(l,:);
    moldenswello = moldenswello_NR(l,:);
    moldenswellg = moldenswellg_NR(l,:);
    %% Update Material Balance
    waterNew = waterNew_NR(l,:); totalWaterNew = totalWaterNew_NR(l);
    waterInj = waterInj_NR(l); waterProd = waterProd_NR(l);
    MBEw = MBEw_NR(l);
    nNew = nNew_NR(:,:,l); totalnNew = totalnNew_NR(l,:);
    nInj = nInj_NR(l,:); nProd = nProd_NR(l,:);
    MBEn = MBEn_NR(l,:);
    %% Calculate Recovery Factor
    vk = vk + qk*t;

    [URF,nPuff,nHuff] = RecoveryFactor(process,totalnInit,Nc,vk,numG, ...
                                       nInj,n,nHuff,nPuff,startprod);
    end
    %% Write Variables to Report
    % Only write to report at specified interval
    if (mod(simtime,interval) == 0)
        rstep=rstep+1;
        p_r(rstep,:) = p;
        z_r(:,:,rstep) = z; xo_r(:,:,rstep) = xo; xg_r(:,:,rstep) = xg;
        Sw_r(rstep,:) = Sw; So_r(rstep,:) = So; Sg_r(rstep,:) = Sg;
        krw_r(rstep,:) = krw; kro_r(rstep,:) = kro; krg_r(rstep,:) = krg;
        viscw_r(rstep,:) = viscw;
        visco_r(rstep,:) = visco;
        viscg_r(rstep,:) = viscg;
    end
    if n == startprod

    end
    %% Update simulation and run times for display
    runtime = toc;
    simtime = simtime+t;
    clc
end
```

```
toc
```

```
% ------------------------------------------------------------------------ %
% BIP.m computes binary interaction parameters for individual components
% or groups
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [Kbin] = BIP(z,Nc,Nc_g,num_g,z_g)

Kbin = zeros(Nc,Nc); Kbin_g = zeros(length(Nc_g),length(Nc_g));

Kbin0 = [0      0      0.0311 0.0515 0.0852 0.1    0.0711  ...
         0.1    0.1    0.1496 0.1441 0.15   0.155  0.155 ; ...
         0      0      0.107  0.1322 0.1241 0.14   0.1333  ...
         0.14   0.14   0.145  0.145  0.14   0.0145 0.0145; ...
         0.0311 0.0107 0      0.0026 0.014  0.0256 0.0133  ...
         0.0056 0.0236 0.0422 0.0352 0.047  0.0474 0.05  ; ...
         0.0515 0.1322 0.0026 0      0.0011 0.0067 0.0096  ...
         0.008  0.0078 0.014  0.015  0.016  0.019  0.03  ; ...
         0.0852 0.1241 0.014  0.0011 0      0.0078 0.0033  ...
         0.0111 0.012  0.0267 0.056  0.059  0.007  0.02  ; ...
         0.1    0.14   0.0256 0.0067 0.0078 0      0       ...
         0.004  0.002  0.024  0.025  0.026  0.006  0.01  ; ...
         0.0711 0.1333 0.0133 0.0096 0.0033 0      0       ...
         0.017  0.017  0.0174 0.019  0.012  0.01   0.001 ; ...
         0.1    0.14   0.0056 0.008  0.0111 0.004  0.017   ...
         0      0      0      0      0      0      0     ; ...
         0.1    0.14   0.0236 0.0078 0.012  0.002  0.017   ...
         0      0      0      0      0      0      0     ; ...
         0.1496 0.145  0.0422 0.014  0.0267 0.024  0.0174  ...
         0      0      0      0      0      0      0     ; ...
         0.1441 0.145  0.0352 0.015  0.056  0.025  0.019   ...
         0      0      0      0      0      0      0     ; ...
         0.15   0.14   0.047  0.016  0.059  0.026  0.012   ...
         0      0      0      0      0      0      0     ; ...
         0.155  0.0145 0.0474 0.019  0.007  0.006  0.01    ...
         0      0      0      0      0      0      0     ; ...
         0.155  0.0145 0.05   0.03   0.02   0.01   0.001   ...
         0      0      0      0      0      0      0     ];

for i=1:Nc
    for j=1:Nc
        for k=1:Nc_g(i)
            for l=1:Nc_g(j)
            Kbin_g(l) = Kbin0(num_g(i,k),num_g(j,l));
            var1(l) = z_g(j,l)*Kbin_g(l);
            end
            sumvar1(k) = z_g(i,k)*sum(var1(1:Nc_g(j)));
        end
        sumvar2(j) = sum(sumvar1(1:Nc_g(i)));
        Kbin(i,j) = sumvar2(j)/(z(i)*z(j));
    end
end

Kbin(logical(eye(size(Kbin)))) = 0;
```

```
end
```

```
% ------------------------------------------------------------------------ %
% DirectionalTrans.m computes directional transmissibilities and associated
% parameters
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [dir,Gin,Gout,Twplus,Twminus,Toplus,Tominus,...
          Tgplus,Tgminus,Tplus,Tminus,Gplus,Gminus] = ...
          DirectionalTrans(p,numG,Tw,To,Tg,grid,gridHalf, ...
          coordsys,A,dz,theta,n,startprod)

cart = strcmp(coordsys,'Cartesian');
rad = strcmp(coordsys,'Radial');
slim = strcmp(coordsys,'Slim Tube');

Twplus = zeros(1,numG+2); Twminus = zeros(1,numG+2);
Toplus = zeros(1,numG+2); Tominus = zeros(1,numG+2);
Tgplus = zeros(1,numG+2); Tgminus = zeros(1,numG+2);
Gplus = zeros(1,numG+2); Gminus = zeros(1,numG+2);
dir = zeros(1,numG+2);

for i=2:numG+1
    if p(i+1) <= p(i)
        dir(i) = 1;
    elseif p(i+1) > p(i)
        dir(i) = -1;
    end
    if n < startprod
        dir(2:numG+1) = 1;
    else
        dir(2:numG+1) = -1;
    end
    % Cartesian/ Slimtube
    if cart == 1 || slim == 1
        Gplus(i) = 2 / (((gridHalf(i+1)-gridHalf(i))/A(i) ...
                    + ((gridHalf(i+dir(i)+1)-gridHalf(i+dir(i))) / ...
                    A(i+dir(i)))));
        Gminus(i) = 2 / (((gridHalf(i+1)-gridHalf(i))/A(i) ...
                    + ((gridHalf(i-dir(i)+1)-gridHalf(i-dir(i))) / ...
                    A(i-dir(i)))));

    % Radial
    elseif rad == 1
        if n < startprod
            Gplus(i) = (2*pi*gridHalf(i)*dz(i))/(grid(i+1)-grid(i));
            Gminus(i) = (2*pi*gridHalf(i-1)*dz(i))/(grid(i)-grid(i-1));
        else
            Gplus(i) = (2*pi*gridHalf(i-1)*dz(i))/abs((grid(i)-grid(i-1)));
            Gminus(i) = (2*pi*gridHalf(i)*dz(i))/abs((grid(i+1)-grid(i)));
        end

    end

    Twplus(i) = Gplus(i)*Tw(i);
```

```
    Twminus(i) = Gminus(i)*Tw(i-dir(i));
    Toplus(i) = Gplus(i)*To(i);
    Tominus(i) = Gminus(i)*To(i-dir(i));
    Tgplus(i) = Gplus(i)*Tg(i);
    Tgminus(i) = Gminus(i)*Tg(i-dir(i));

    if dir(2) == 1
        Twminus(2) = 0; Tominus(2) = 0; Tgminus(2) = 0;
    elseif dir(2) == -1
        Twplus(2) = 0; Toplus(2) = 0; Tgplus(2) = 0;
    end

    if dir(numG+1) == 1
        Twplus(numG+1) = 0; Toplus(numG+1) = 0; Tgplus(numG+1) = 0;
    elseif dir(numG+1) == -1
        Twminus(numG+1) = 0; Tominus(numG+1) = 0; Tgminus(numG+1) = 0;
    end

end
Tplus = theta*Twplus + Toplus + Tgplus;
Tminus = theta*Twminus + Tominus + Tgminus;

Gin = Gminus(2); Gout = Gplus(numG+1);

end
```

```
% ------------------------------------------------------------------------ %
% FluidProps.m provides fluid properties for a hydrocarbon mixture
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [z,w,Tc,Pc,MW,vc,SE,pchor,Nc,Ncg,num_g,z_g] = ...
         FluidProps(comp,zin,w7,Tc7,Pc7,MW7,vc7,SE7,pchor7)

Nc = numel(comp);
comp0 = {{ 'N2'},{'CO2'},{ 'C1'},{ 'C2'},{ 'C3'},{'iC4'},{'nC4'}, ...
         {'iC5'},{'nC5'},{'nC6'},{'nC7'},{'nC8'},{'nC9'},{'nC10'}};
w0 =  [0.0403 0.2276 0.0115 0.0995 0.1523 0.1770 0.2002 ...
       0.2275 0.2515 0.3013 0.3495 0.3996 0.4435 0.4923];
Tc0 = [126.1  304.19 190.56 305.32 369.83 408.14 425.12 ...
       460.43 469.7  507.6  540.2  568.7  594.6  617.7 ]; %[K]
Pc0 = [3.394 7.382 4.599 4.872 4.248 3.648 3.796 ...
       3.381 3.370 3.025 2.740 2.490 2.290 2.110]; %[MPa]
MW0 = [ 28.014  44.010  16.043  30.070  44.096  58.123  58.123 ...
        72.150  72.150  86.177 100.204 114.231 128.258 142.285]; %[kg/kmol]
vc0 = [0.0901 0.0940 0.0986 0.1455 0.2000 0.2627 0.2550 ...
       0.3058 0.3130 0.371  0.428  0.486  0.544  0.600]; %[m^3/kgmol]
SE0 = [ 0        0       -0.1540  -0.1002 -0.08501 -0.07935 -0.06413 ...
       -0.04350 -0.04183 -0.01478  0       0        0        0      ];
pchor0 = [41    78    77    108   150.3 181.5 189.9 ...
          225   231.5 271   312.5 351.5 393   433.5];

Ncg=zeros(1,Nc);

z=zeros(1,Nc); w=zeros(1,Nc); Tc=zeros(1,Nc); Pc=zeros(1,Nc);
MW=zeros(1,Nc); vc=zeros(1,Nc); SE=zeros(1,Nc); pchor=zeros(1,Nc);
c7plus=zeros(1,Nc); num=zeros(1,Nc);

z_g=zeros(Nc,Nc); w_g=zeros(1,Nc); Tc_g=zeros(1,Nc); Pc_g=zeros(1,Nc);
MW_g=zeros(1,Nc); vc_g=zeros(1,Nc); SE_g=zeros(1,Nc); pchor_g=zeros(1,Nc);
c7plus_g=zeros(1,Nc); num_g=zeros(Nc,Nc);

for i=1:Nc
    j=1;
    Ncg(i) = numel(comp{i});
    % Single Component
    if Ncg(i) == 1
        for k=1:length(comp0)
            if strcmp(comp{i},comp0{k}) == 1
                num(i) = k;
                num_g(i,j) = k;
            end
            if strcmp(comp{i},'C7+') == 1
                c7plus(i) = 1;
                num_g(i,j) = 14;
            end
        end

         z(i) = zin{i};
         z_g(i,j) = z(i);
```

```
     if c7plus(i) == 1
        w(i) = w7;
        Tc(i) = Tc7;
        Pc(i) = Pc7;
        MW(i) = MW7;
        vc(i) = vc7;
        SE(i) = SE7;
        pchor(i) = pchor7;
     else
        w(i) = w0(num(i));
        Tc(i) = Tc0(num(i));
        Pc(i) = Pc0(num(i));
        MW(i) = MW0(num(i));
        vc(i) = vc0(num(i));
        SE(i) = SE0(num(i));
        pchor(i) = pchor0(num(i));
     end
% Group
else

    group = comp{i};
    for j=1:Ncg(i)
        for k=1:length(comp0)
            if strcmp(group(j),comp0{k}) == 1
                num_g(i,j) = k;
            end
            if strcmp(group(j),'C7+') == 1
               c7plus_g(j) = 1;
               num_g(i,j) = 14;
            end
        end

         zgroup = zin{i};
         if c7plus(i) == 1
            w(i) = w7;
            Tc(i) = Tc7;
            Pc(i) = Pc7;
            MW(i) = MW7;
            vc(i) = vc7;
            SE(i) = SE7;
            pchor(i) = pchor7;
         else
            w_g(j) = w0(num_g(i,j));
            Tc_g(j) = Tc0(num_g(i,j));
            Pc_g(j) = Pc0(num_g(i,j));
            MW_g(j) = MW0(num_g(i,j));
            vc_g(j) = vc0(num_g(i,j));
            SE_g(j) = SE0(num_g(i,j));
            pchor_g(j) = pchor0(num_g(i,j));
         end

    z_g(i,j) = zgroup(j);
    end
    z(i) = sum(z_g(i,:));
    MW(i) = sum(z_g(i,:).*MW_g)/sum(z_g(i,:));
    w(i) = sum(z_g(i,:).*w_g)/sum(z_g(i,:));
```

```
        Tc(i) = sum(z_g(i,:).*Tc_g)/sum(z_g(i,:));
        Pc(i) = sum(z_g(i,:).*Pc_g)/sum(z_g(i,:));
        vc(i) = sum(z_g(i,:).*vc_g)/sum(z_g(i,:));
        SE(i) = sum(z_g(i,:).*SE_g)/sum(z_g(i,:));
        pchor(i) = sum(z_g(i,:).*pchor_g)/sum(z_g(i,:));
    end

end

for i=1:Nc
if z(i) == 0
    MW(i) = 0;
    w(i) = 0;
    Tc(i) = 0;
    Pc(i) = 0;
    vc(i) = 0;
    MW(i) = 0;
    SE(i) = 0;
    pchor(i) = 0;
end

end
```

```
% ------------------------------------------------------------------------ %
% GridProps.m provides grid properties for a reservoir model
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [grid,gridHalf,A,dz,Vr] = ...
         GridProps(numG,xinlet,xoutlet,coordsys,gridtype,c1,c2,c3)

cart = strcmp(coordsys,'Cartesian');
rad = strcmp(coordsys,'Radial');
slim = strcmp(coordsys,'Slim Tube');

pdg = strcmp(gridtype,'PDG');
bcg = strcmp(gridtype,'BCG');

grid = zeros(1,numG + 2);
gridHalf = zeros(1,numG + 3);
r = zeros(1,numG);
Vr = zeros(1,numG + 2);

% Cartesian/ Slimtube
if cart == 1 || slim == 1
    dy(1:numG+3)=c1;
    dz(1:numG+3)=c2;
    STr(1:numG+3) = c3;

    % Block-Centered Grid
    if bcg == 1
        % Size of Gridblocks
        dx=(xoutlet-xinlet)/numG;

        % Grid blocks and Grid Boundaries
        grid(1)=xinlet-0.5*dx;
        for i=2:numG+1
            grid(i)= grid(i-1)+dx;
        end
        grid(numG+2) = grid(numG+1)+dx;

        gridHalf(1)=xinlet-dx;
        for i=2:numG+2
            gridHalf(i)= gridHalf(i-1)+dx;
        end
        gridHalf(numG+3) = gridHalf(numG+2)+dx;

    elseif pdg == 1
        % Size of Gridblocks
        dx=(xoutlet-xinlet)/(numG-1);

        % Grid blocks and Grid Boundaries
        grid(1)=xinlet - 0.5*dx;
        grid(2)=xinlet;
        for i=3:numG+1
            grid(i)= grid(i-1)+dx;
```

```
        end
        grid(numG+2) = grid(numG+1)+0.5*dx;

        gridHalf(1) = xinlet-dx;
        gridHalf(2) = xinlet;
        gridHalf(3) = xinlet + 0.5*dx;
        for i=4:numG+1
            gridHalf(i)= gridHalf(i-1)+dx;
        end
        gridHalf(numG+2) = gridHalf(numG+1)+0.5*dx;
        gridHalf(numG+3) = gridHalf(numG+2)+dx;

    end


% Radial
elseif rad == 1
    rw=xinlet;
    re=xoutlet;
    dz(1:numG+3)=c2;

    r(1) = rw;
    for i=2:numG
        r(i) = r(i-1)*(re/rw)^(1/(numG-1));
    end

    % Block-Centered Grid
    if bcg == 1
        % Grid blocks and Grid Boundaries
        grid(2:numG+1) = r;
        grid(1) = grid(2)/(re/rw)^(1/(numG-1));
        grid(numG+2) = grid(numG+1)*(re/rw)^(1/(numG-1));

        gridin = grid(1)/(re/rw)^(1/(numG-1));
        gridout = grid(numG+2)*(re/rw)^(1/(numG-1));

        gridHalf(1)=(grid(1)-gridin)/log(grid(1)/gridin);
        for i=2:numG+2
            gridHalf(i)= (grid(i)-grid(i-1))/log(grid(i)/grid(i-1));
        end
        gridHalf(numG+3) = (gridout-grid(numG+2)) / ...
                            log(gridout/grid(numG+2));

    elseif pdg == 1
        grid(2:numG+1) = r;
        grid(1) = grid(2)/(re/rw)^(1/(numG-1));
        grid(numG+2) = grid(numG+1)*(re/rw)^(1/(numG-1));

        gridin = grid(1)/(re/rw)^(1/(numG-1));
        gridout = grid(numG+2)*(re/rw)^(1/(numG-1));

        gridHalf(1)=(grid(1)-gridin)/log(grid(1)/gridin);
        for i=2:numG+2
            gridHalf(i)= (grid(i)-grid(i-1))/log(grid(i)/grid(i-1));
```

```
        end
        gridHalf(numG+3) = (gridout-grid(numG+2)) / ...
                            log(gridout/grid(numG+2));

    end

end

% Areas
if cart == 1
    A = dy.*dz;
elseif slim ==1
    A = pi*STr.*STr;
elseif rad == 1
    A = pi.*dz;
end
% Volumes
for i=1:numG+2
    Vr(i) = (gridHalf(i+1)-gridHalf(i))*A(i);
end

end
```

```
% ------------------------------------------------------------------------ %
% LBCVisc.m uses the Lohrenz-Bray-Clark method for visocisity prediction of
% hydrocarbon mixtures
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [visco,viscg] = ...
         LBCVisc(Nc,xoinput,xginput,Tc,Pc,MW,T,vc,volo,volg)

% Calculate uo and lambda for each component
xo = xoinput;
xg = xginput;
if sum(xo) == 0
    xo = xg;
    volo = volg;
elseif sum(xg) ==0
    xg=xo;
    volg=volo;
end
for i=1:Nc
if xo(i) == 0
    Tc(i) = 0;
    Pc(i) = 0;
    MW(i) = 0;
    vc(i) = 0;
end
end
xo(xo==0) = [];
xg(xg==0) = [];
Tc(Tc==0) = [];
Pc(Pc==0) = [];
MW(MW==0) = [];
vc(vc==0) = [];
Nchold = numel(xo);

if sum(xo)==0&&sum(xg)==0
    visco=10E-8;
    viscg=10E-8;
else

% Reduced Temperature
Tr = T./Tc ;
% Pc in atmospheres
Pcatm = Pc./0.101325; %[atm]
% Lambda calculation
lambda = Tc.^(1/6).*MW.^(-1/2).*Pcatm.^(-2/3);
% uo calculation
for i = 1:Nchold
    if Tr(i) <= 1.5
        u0(i) = (34E-5)*(Tr(i)^0.94)/lambda(i); %[mPa-s]
    else
        u0(i) = (17.78E-5)*((4.58*Tr(i)-1.67)^(5/8))/lambda(i); %[mPa-s]
    end
end
```

```
% Calculate uo and lambda for oil and gas using Herning-Zipperer mixing
% rule
% Oil u0 and lambda
u0o = sum(xo.*u0.*sqrt(MW))/sum(xo.*sqrt(MW)); %[mPa-s]
lambdao = ((sum(xo.*Tc))^(1/6))*((sum(xo.*MW))^(-1/2)) * ...
          ((sum(xo.*Pcatm))^(-2/3));
% Gas u0 and lambda
u0g = sum(xg.*u0.*sqrt(MW))/sum(xg.*sqrt(MW)); %[mPa-s]
lambdag = ((sum(xg.*Tc))^(1/6))*((sum(xg.*MW))^(-1/2)) * ...
          ((sum(xg.*Pcatm))^(-2/3));

% Calculate oil and gas viscosities

% Convert critical volume to m^3/mol
vcrit = vc/(10^3); %[m^3/mol]
% Oil reduced density
pro = sum(xo.*vcrit)/volo;
% Gas reduced density
prg = sum(xg.*vcrit)/volg;
% Oil and gas viscosities
a1 = 0.10230; a2 = 0.023364; a3 = 0.058533;
a4 = -0.040758; a5 = 0.0093324;
% Oil viscosity
visco = (u0o+(1/lambdao)*(a1+a2*pro+a3*pro^2+a4*pro^3+a5*pro^4)^4 - ...
        (10^-4))/1000; %[Pa-s]
% Gas viscosity
viscg = (u0g+(1/lambdag)*(a1+a2*prg+a3*prg^2+a4*prg^3+a5*prg^4)^4 - ...
        (10^-4))/1000; %[Pa-s]
end

end
```

```
% ------------------------------------------------------------------------ %
% MacSugIFT.m Extension to multiple components of Macleod-Sugden method
% for calculating IFT
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [ift] = MacSugIFT(Nc,xoinput,xginput,pchor,densoinput,densginput)

% Calulate uo and lambda for each component
xo = xoinput;
xg = xginput;
if sum(xo) == 0
    xo = xg;
elseif sum(xg ==0)
    xg=xo;
end
denso = densoinput/(100^3);
densg = densginput/(100^3);
for i=1:Nc
if xo(i) == 0
    pchor(i) = 0;
end
end
xo(xo==0) = [];
xg(xg==0) = [];
pchor(pchor==0) = [];

if sum(xo)==0&&sum(xg)==0
    ift=10E-16;
    return
end

ift = (sum(pchor.*(xo.*denso - xg.*densg)))^4;

end
```

```
% ------------------------------------------------------------------------ %
% PRFlash.m uses the Peng-Robinson implementation of an EoS flash
% calculation
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [x,y,L,V,K,Zl,Zv,ddensldp,ddensvdp,voll,volv, ...
         moldensl,moldensv,densl,densv] = ...
         PRFlash(Nc,zinput,R,w,Tc,Pc,MW,Kbin,T,p,SE)

% Calculate general parameters of the EoS
z = zinput;
zlim = 0;
for i=1:Nc
    % Deal with composition equal to zero
    if z(i) <= zlim
        z(i) = 0;
        w(i) = 1000;
        Tc(i) = 0;
        Pc(i) = 0;
        MW(i) = 0;
        SE(i) = 1000;
        Kbin(i,:) = 1000;
        Kbin(:,i) = 1000;
    end

    if sum(z)==0
        x=zeros(1,Nc);
        y=zeros(1,Nc);
        L=0;
        V=0;
        K=zeros(1,Nc);
        Zl=0;
        Zv=0;
        dZldp=0;
        dZvdp=0;
        voll=0;
        volv=0;
        moldensl=0;
        moldensv=0;
        densl=0;
        densv=0;
        return;
    end

end
z(z==0) = [];
Nc_hold = numel(z);
z=z/sum(z(1:Nc_hold));
w(w==1000) = [];
Tc(Tc==0) = [];
Pc(Pc==0) = [];
MW(MW==0) = [];
SE(SE==1000) = [];
Kbin(all(Kbin==1000,1),:)=[];
```

```
Kbin = Kbin';
Kbin(all(Kbin==1000,2),:)=[];

% Reduced Temperature
Tr = T./Tc;
% Convert Pressure from Pa to MPa
P = p;
Pc = Pc*(10^6);
% Parameters of the EoS
ac = 0.457235*(R^2).*(Tc.^2)./Pc; % [MPa(m^3/kgmol)^2]
b = 0.077796*R*Tc./Pc; % [m^3/kgmol]
m = 0.37464 + 1.5422*w - 0.26992*w.^2;
alpha=(1+m.*(1-Tr.^0.5)).^2;
a = ac.*alpha; % [MPa(m^3/kgmol)^2]

c = SE.*b;

% Make a first guess at equilibrium ratios
% Wilson Correlation for estimating K-values (Regular Wilson)
Kwil = (Pc/P).*exp(5.37*(1+w).*(1-(Tc/T)));

%Check Stability

%Single Phase
btz = sum(z.*b); %[m^3/kgmol]
atz = sum((z.*a.^.5*(1-Kbin)).*(z.*a.^.5)); %[MPa(m^3/kgmol)^2]
Az = atz*P/(R*T)^2;
Bz = btz*P/(R*T);

zerz = roots([1 -(1-Bz) (Az-Bz*2-3*Bz^2) -(Az.*Bz-Bz^2-Bz^3)]);
Zz = zerz(imag(zerz)==0);
Zz = Zz(Zz>=0);

Bjz = b/btz;
Ajz = 2*(a.^.5).*((1-Kbin)*(z.*a.^.5)')'/atz;
phez = exp(b*(Zz-1)/btz-log(Zz-Bz)+((Az./(Bz.*(2*2^.5))).*(Ajz-Bjz)) ...
       .*log((Zz+(1-2^.5)*Bz)/(Zz+(1+2^.5)*Bz)));
phez = phez.*exp(-c*P/(R*T));
fz = z.*phez*P;

if sum(MW.*z) < 45
    vapor = 1;
else
    vapor = 0;
end

singlephase = 1;

while singlephase == 1

    %Create Vapor like phase
    K = Kwil;
    trivialv = 0;
    errorv = 1;
```

```
while errorv>10^(-10)
    Y = z.*K;
    Sv = sum(Y);
    y = Y/Sv;

    % Vapor mixture parameters
    bty = sum(y.*b); %[m^3/kgmol]
    aty = sum((y.*a.^.5*(1-Kbin)).*(y.*a.^.5)); %[MPa(m^3/kgmol)^2]
    Ay = aty*P/(R*T)^2;
    By = bty*P/(R*T);

    % Solve for Compressibility Factors
    % Z^3 - (1-B)*Z^2 + (A-2*B-3*B^2)*Z - (A*B-B^2-B^3) = 0
    z_v = roots([1 -(1-By) (Ay-By*2-3*By^2) -(Ay.*By-By^2-By^3)]);
    for i=1:3
        real_v(i)=isreal(z_v(i));
    end
    Zv = max(z_v'.*real_v);
    % Variables to solve for fugacity coefficients
    Bjv = b/bty;
    Ajv = 2*(a.^.5).*((1-Kbin)*(y.*a.^.5)')'/aty;
    % Calculate fugacity coefficients
    phev = exp(b*(Zv-1)/bty-log(Zv-By)+((Ay./(By.*(2*2^.5))) ...
           .*(Ajv-Bjv)).*log((Zv+(1-2^.5)*By)/(Zv+(1+2^.5)*By)));
    phev = phev.*exp(-c*P/(R*T));
    % Calculate fugacities
    fv = y.*phev*P;

    Rcorr = (fz./fv)*(1/Sv);
    K = K.*Rcorr;
    errorv = sum((Rcorr-1).^2);
    if (sum((log(K)).^2)< 10^-4)
        trivialv = 1;
        break
    end
end

if (Sv > 1) && (trivialv == 0)
    singlephase = 0;
    break
end

% Create liquid like phase
K = Kwil;
triviall = 0;
errorl = 1;
while errorl>10^(-10)
    Y = z./K;
    Sl = sum(Y);
    x = Y./Sl;

    % Vapor mixture parameters
    btx = sum(x.*b); %[m^3/kgmol]
    atx = sum((x.*a.^.5*(1-Kbin)).*(x.*a.^.5)); %[MPa(m^3/kgmol)^2]
    Ax = atx*P/(R*T)^2;
```

```
        Bx = btx*P/(R*T);

        % Solve for Compressibility Factors
        % Z^3 - (1-B)*Z^2 + (A-2*B-3*B^2)*Z - (A*B-B^2-B^3) = 0
        z_l = roots([1 -(1-Bx) (Ax-Bx*2-3*Bx^2) -(Ax.*Bx-Bx^2-Bx^3)]);
        for i=1:3
            real_l(i)=isreal(z_l(i));
        end
        Zl_real=(z_l'.*real_l);
        h = 1;
        for j=1:3
            if Zl_real(j)~=0 && Zl_real(j) > 0.1
                Zl_temp(h) = Zl_real(j);
                h=h+1;
            end
        end
        Zl = min(Zl_temp);

        % Variables to solve for fugacity coefficients
        Bjl = b/btx;
        Ajl = 2*(a.^.5).*((1-Kbin)*(x.*a.^.5)')'/atx;
        % Calculate fugacity coefficients
        phel = exp(b*(Zl-1)/btx-log(Zl-Bx)+((Ax./(Bx.*(2*2^.5))) ...
               .*(Ajl-Bjl)).*log((Zl+(1-2^.5)*Bx)/(Zl+(1+2^.5)*Bx)));
        phel = phel.*exp(-c*P/(R*T));
        % Calculate fugacities
        fl = x.*phel*P;

        Rcorr = (fl./fz)*Sl;
        K = K.*Rcorr;
        errorl = sum((Rcorr-1).^2);
        if (sum((log(K)).^2)< 10^-4)
            triviall = 1;
            break
        end
    end

    if (Sl > 1) && (triviall == 0)
        singlephase = 0;
        break
    end

    break

end

if singlephase == 1
    if vapor == 1
        y = z;
        x = y*0;
        V = 1;
        L = 0;
        Zv = Zz;
        Zl = 1;
        aty = atz;
```

```
        bty = btz;
        Ay = Az;
        By = Bz;
        atx = 0;
        btx = 0;
        Ax = 0;
        Bx = 0;
    else
        x = z;
        y = x*0;
        L = 1;
        V = 0;
        Zl = Zz;
        Zv = 1;
        atx = atz;
        btx = btz;
        Ax = Az;
        Bx = Bz;
        aty = 0;
        bty = 0;
        Ay = 0;
        By = 0;
    end

else

% Perform iterations to match fugacities
K = Kwil;
error = 1;
epszl = 0.1;
V = 0.5; % guess vapor split V = 0.5
while error>10^(-12)
    % Solve the Ratchford-Rice Equation for molar volumes and phase
    % fractions
    err = 1;
    while err>10^(-8)
    rr = sum(z.*(K-1)./(1+(K-1)*V));
    rrprime = (-sum(z.*((K-1).^2)./((1+(K-1)*V)).^2));
    Vold=V;
    V = Vold-rr/rrprime; % Iterate using Newton-Raphson method
    err=(V-Vold)/Vold;
    if V<0
       V = 0;
       break
    end
    if V>1
       V = 1;
       break
    end
    end
L = 1-V;
x = z./(1+(K-1)*V);
y = K.*x;
% Caculate phase mixture parameters
% Liquid mixture parameters
btx = sum(x.*b); %[m^3/kgmol]
```

```
atx = sum((x.*a.^.5*(1-Kbin)).*(x.*a.^.5)); %[MPa(m^3/kgmol)^2]
Ax = atx*P/(R*T)^2;
Bx = btx*P/(R*T);
% Vapor mixture parameters
bty = sum(y.*b); %[m^3/kgmol]
aty = sum((y.*a.^.5*(1-Kbin)).*(y.*a.^.5)); %[MPa(m^3/kgmol)^2]
Ay = aty*P/(R*T)^2;
By = bty*P/(R*T);
% Solve for Compressibility Factors
% Z^3 - (1-B)*Z^2 + (A-2*B-3*B^2)*Z - (A*B-B^2-B^3) = 0
z_l = roots([1 -(1-Bx) (Ax-Bx*2-3*Bx^2) -(Ax.*Bx-Bx^2-Bx^3)]);
z_v = roots([1 -(1-By) (Ay-By*2-3*By^2) -(Ay.*By-By^2-By^3)]);
for i=1:3
    real_l(i)=isreal(z_l(i));
    real_v(i)=isreal(z_v(i));
end
Zv = max(z_v'.*real_v);
Zl_real=(z_l'.*real_l);
h = 1;
for j=1:3
    if Zl_real(j)~=0 && Zl_real(j) > epszl
        Zl_temp(h) = Zl_real(j);
        h=h+1;
    end
end
Zl = min(Zl_temp);

% Variables to solve for fugacity coefficients
Bjl = b/btx;
Ajl = 2*(a.^.5).*((1-Kbin)*(x.*a.^.5)')'/atx;
Bjv = b/bty;
Ajv = 2*(a.^.5).*((1-Kbin)*(y.*a.^.5)')'/aty;
% Calculate fugacity coefficients
phel = exp(b*(Zl-1)/btx-log(Zl-Bx)+((Ax./(Bx.*(2*2^.5))).*(Ajl-Bjl)) ...
       .*log((Zl+(1-2^.5)*Bx)/(Zl+(1+2^.5)*Bx)));
phev = exp(b*(Zv-1)/bty-log(Zv-By)+((Ay./(By.*(2*2^.5))).*(Ajv-Bjv)) ...
       .*log((Zv+(1-2^.5)*By)/(Zv+(1+2^.5)*By)));
% Calculate fugacities
fl = x.*phel*P;
fv = y.*phev*P;
% Check for convergence and update equilibrium ratios
error = sum((1-fl/fv).^2);
K = K.*(fl./fv);
end

end

% Density of each phase
if sum(x) == 0
    voll = 1; %[m^3/mol]
    moldensl = 1; %[mol/m^3]
    densl = 1;
    ddensldp = 1;
else
    voll = (Zl*R*T/P - sum(x.*c)); %[m^3/mol]
    moldensl = 1/voll; %[mol/m^3]
```

```
    densl = (sum(x.*MW)/1000)/voll; %[kg/m^3]
    dAxdp=(atx/(R*T)^2);
    dBxdp = (btx/(R*T));
    dvolldp=-R*T/((voll-btx)^2)+2*atx*(voll+btx) / ...
            ((voll^2+2*btx*voll-btx^2)^2);
    dZldp = 1/(R*T)*(voll+1/dvolldp*p);
    ddensldp = (1/(R*T*Zl))*(1-(p/Zl)*dZldp);
end
if sum(y) == 0
    volv = 1; %[m^3/mol]
    moldensv = 1; %[mol/m^3]
    densv = 1;
    ddensvdp = 1;
else
    volv = (Zv*R*T/P - sum(y.*c)); %[m^3/mol]
    moldensv = 1/volv; %[mol/m^3]
    densv = (sum(y.*MW/1000))/volv; %[kg/m^3]
    dAydp=(aty/(R*T)^2);
    dBydp = (bty/(R*T));
    dvolvdp=-R*T/((volv-bty)^2)+2*aty*(volv+bty) / ...
            ((volv^2+2*bty*volv-bty^2)^2);
    dZvdp = 1/(R*T)*(volv+1/dvolvdp*p);
    ddensvdp = (1/(R*T*Zv))*(1-(p/Zv)*dZvdp);
end

% Convert back to full Nc of components
s=1;
for i=1:Nc
    if zinput(i) <= zlim
        xhold(i) = 0;
        yhold(i) = 0;
    else
        xhold(i) = x(s);
        yhold(i) = y(s);
        s=s+1;
    end
end

x=xhold;
y=yhold;

end
```

```
% ------------------------------------------------------------------------ %
% QComponent.m computes the flow of each individual component of a
% hydrocarbon mixture
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %
function qk = QComponent(Qo,Qg,densoin,densgin,xoin,xgin)
qk = densoin*Qo*xoin+densgin*Qg*xgin;
end
```

```
% ------------------------------------------------------------------------ %
% RecoveryFactor.m is used to calculate and monitor the recovery
% factor for different injection processes
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [URF,nPuff,nHuff] = ...
         RecoveryFactor(process,totalnInit,Nc,vk,numG,nInj, ...
         n,nHuff,nPuff,startprod)

huffpuff = strcmp(process,'Huff Puff');
displacement = strcmp(process,'Displacement');

% Displacement Process
if displacement == 1
    molesOut = -vk(numG + 1,:);
    for k=1:Nc
        if molesOut(k) > totalnInit(k)
        molesOut(k) = totalnInit(k);
        end
    end
    URF = sum(molesOut)/sum(totalnInit);
% Huff 'n' Puff process
elseif huffpuff == 1
    if n < startprod
        nHuff = nInj;
        URF = 0;
    else
        nPuff = nInj - nHuff;
        molesOut = zeros(1,Nc);
        for k=1:Nc
            if nPuff(k) > nHuff(k)
            molesOut(k) = nPuff(k)-nHuff(k);
            end
        end
        URF = sum(molesOut)/sum(totalnInit);
    end
end

end
```

```
% ------------------------------------------------------------------------ %
% RelPerm.m uses the Eclipse standard model for 3-Phase relative
% permeability
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [krw,kro,krg] = RelPerm(krwmax,krowmax,krogmax,krgmax,Sw,So, ...
                         Sg,Swc,Sorw,Sorg,Sgc,nw,now,nog,ng, ...
                         relpermtype,Swtab,krwtab,krowtab,Sgtab, ...
                         krgtab,krogtab)

tabular = strcmp(relpermtype,'Tabular');

if tabular == 1
    krw = interp1(Swtab,krwtab,Sw,'pchip');
    krow = interp1(Swtab,krowtab,Sw,'pchip');
    krg = interp1(Sgtab,krgtab,Sg,'pchip');
    krog = interp1(Sgtab,krogtab,Sg,'pchip');
else
    if Sw == 1
        kro = 0;
        krw = krwmax;
        krg = 0;
        return;
    elseif So == 1
        kro = krogmax;
        krw = 0;
        krg = 0;
        return;
    else
    % Relative permeability of oil with water and gas
    if So<Sorw
        krow = 0;
    else
        krow = krowmax*((1-Sw-Sorw)/(1-Swc-Sorw))^now;
    end

    if So<Sorg
        krog = 0;
    else
        krog = krogmax*((1-Sg-Sorg-Swc)/(1-Swc-Sorg))^nog;
    end
    % Relative permeability of each phase
    if Sw<Swc
        krw = 0;
    else
        krw = krwmax*((Sw-Swc)/(1-Swc-Sorw))^nw;
    end

    end

    if Sg<Sgc
        krg = 0;
    else
```

```
        krg = krgmax*((Sg-Sgc)/(1-Swc-Sgc))^ng;
    end
end

if Sg<10E-6 % Use this condition to stablize model, otherwise a very
            % small value of Sg could make the alpha eqn unstable
    alpha = 0;
else
    alpha = Sg/(Sg+Sw-Swc);
end
kro = alpha*krog+(1-alpha)*krow;

end
```

```
% ------------------------------------------------------------------------ %
% tridiag.m is an implementation of the Thomas Algorithm for solution
% of tridiagonal matrices
%
% Author: Mark Holmes
% Retrieved from http://www.mathworks.com/matlabcentral/fileexchange/ ...
% 40722-tridiag-m?focused=3780252&tab=function on August 10, 2015
% ------------------------------------------------------------------------ %

function y = tridiag(a,b,c,f)

n = length(f);
v = zeros(n,1);
y = v;
w = a(1);
y(1) = f(1)/w;
for i=2:n
    v(i-1) = c(i-1)/w;
    w = a(i) - b(i)*v(i-1);
    y(i) = ( f(i) - b(i)*y(i-1) )/w;
end
for j=n-1:-1:1
   y(j) = y(j) - v(j)*y(j+1);
end

end
```

```
% ------------------------------------------------------------------------ %
% UpdateSaturations.m is used to calculate saturations at each time step
% and each iteration
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [Sw,So,Sg] = UpdateSaturations(t,numG,Twminus,Twplus,p,Pcow,L, ...
                      V,moldenso,moldensg,moldensw,moldenswn,phi,phin, ...
                      Swn,Vr,dir,qw)
Sw = zeros(1,numG+2);
So = zeros(1,numG+2);
Sg = zeros(1,numG+2);

for i = 2:numG+1
    Sw(i)=(...
          Twminus(i)*(p(i-dir(i))-Pcow(i-dir(i))) ...
          + (-Twminus(i)-Twplus(i))*(p(i)-Pcow(i)) ...
          + Twplus(i)*(p(i+dir(i))-Pcow(i+dir(i))) ...
          + qw(i) + (Vr(i)/t)*phin(i)*moldenswn(i)*Swn(i) ...
          ) ...
          / ...
          ( ...
          (Vr(i)/t)*phi(i)*moldensw(i) ...
          );
    if Sw(i) == 1
        So(i) = 0;
        Sg(i) = 0;
    else
        So(i)=((1-Sw(i))*moldensg(i)*L(i)) / ...
              (L(i)*moldensg(i)+V(i)*moldenso(i));
        Sg(i)=1-Sw(i)-So(i);
    end
end

end
```

```
% ------------------------------------------------------------------------ %
% UpdateZ.m is used to calculate compositions at each time step and each
% iteration
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [znew] = UpdateZ(phi,zn,xo,xg,moldenso,moldensg,So,Sg,Toplus, ...
                  Tominus,Tgplus,Tgminus,p,dir,Nc,t,numG,Vr,qo,qg,qk)

znew = zeros(numG + 2, Nc);

for i=2:numG+1
    for k=1:Nc
        znew(i,k) = (...
                    Tominus(i)*xo(i-dir(i),k)*p(i-dir(i)) ...
                    + (-Tominus(i)*xo(i-dir(i),k) ...
                    - Toplus(i)*xo(i,k))*p(i) ...
                    + Toplus(i)*xo(i,k)*p(i+dir(i)) ...
                    + Tgminus(i)*xg(i-dir(i),k)*p(i-dir(i)) ...
                    + (-Tgminus(i)*xg(i-dir(i),k) ...
                    - Tgplus(i)*xg(i,k))*p(i) ...
                    + Tgplus(i)*xg(i,k)*p(i+dir(i)) ...
                    + qk(i,k) + (Vr(i)/t)*phi(i)*(moldenso(i)*So(i) ...
                    + moldensg(i)*Sg(i))*zn(i,k) ...
                    ) ...
                    / ...
                    ( ...
                    Tominus(i)*p(i-dir(i)) ...
                    + (-Tominus(i)-Toplus(i))*p(i) ...
                    + Toplus(i)*p(i+dir(i)) ...
                    + Tgminus(i)*p(i-dir(i)) ...
                    + (-Tgminus(i)-Tgplus(i))*p(i) ...
                    + Tgplus(i)*(p(i+dir(i))) ...
                    + qo(i) + qg(i) + (Vr(i)/t)*phi(i)*( ...
                    moldenso(i)*So(i)+moldensg(i)*Sg(i)) ...
                    );
    end
end

end
```

```
% ------------------------------------------------------------------------ %
% WaterCapillary.m is used for calculating water capillary pressure
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [Pcow] = WaterCapillary(Sw,Swtab,Pcowtab,Pcowtype)
tabular = strcmp(Pcowtype,'Tabular');
zeroPcow = strcmp(Pcowtype,'Zero');

if tabular==1
    Pcow = interp1(Swtab,Pcowtab,Sw,'pchip');
elseif zeroPcow==1
    Pcow = 0;
else
    Pcow = 0;
end

end
```

```
% ------------------------------------------------------------------------ %
% WaterProps.m computes properties of formation water
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [cw,viscw,densw,moldensw] = WaterProps(T,p,ws,moldensw0,densw0,p0)
T_F = T*(9/5)-459.67; % [F]
p_psia = p/6894.75728;

% Compressibility
C0 = 3.8546-0.000134*p_psia;
C1 = -0.01052+(4.77E-7)*p_psia;
C2 = (3.9267E-5)-(8.8E-10)*p_psia;

cw_psia = (10^-6)*(C0+C1*T_F+C2*T_F^2);
corrfact = 1+(-0.052+(2.7E-4)*T_F-(1.14E-6)*(T_F^2)+(1.121E-9)*(T_F^3))*ws;
cw_psia = cw_psia*corrfact;

cw = cw_psia/6894.75728;

% Viscosity
D_ws = 1.12166-(2.63951E-2)*ws+(6.79461E-4)*ws^2+(5.47119E-5)*ws^3 ...
       -(1.55586E-6)*ws^4;
viscwT = ((109.574-8.450564*ws+0.313314*ws^2 ...
         +(8.72213E-3)*ws^3)*T_F^-D_ws)/1000;

viscw = (0.9994+(4.0295E-5)*p_psia+(3.1062E-9)*p_psia^2)*viscwT;

% Density and Molar Density
moldensw = moldensw0*(1+cw*(p-p0));
densw = densw0*(1+cw*(p-p0));
end
```

```
% ------------------------------------------------------------------------ %
% WellModels.m is used to calculate the well terms for real wells as well
% as simulated wells at reservoir boundaries
%
% Author: Tristan Strong
% ------------------------------------------------------------------------ %

function [Qw,Qo,Qg,qw,qo,qg,qk,WIw,WIo,WIg, ...
          moldenswellw,moldenswello,moldenswellg,p] = ...
          WellModels(n,startshut,startprod,injphase,numG,R,Nc,zin,w,Tc, ...
          Pc,MW,Kbin,T,p,SE,krwmax,krowmax,krogmax,krgmax,Swc,Sorw,Sorg, ...
          Sgc,nw,now,nog,ng,RelPermtype,Swtab,krwtab,krowtab,Sgtab, ...
          krgtab,krogtab,viscw,visco,viscg,densw0,moldensw0,moldensw, ...
          moldenso,moldensg,xo,xg,pe,pbh,Gin,Gout,vc,krw,kro,krg, ...
          welltypein,welltypeout,Qin,Qout,z0,Sw0,ws,p0,perm)

constpressin = strcmp(welltypein,'Pressure');
constratein = strcmp(welltypein,'Rate');
constpressout = strcmp(welltypeout,'Pressure');
constrateout = strcmp(welltypeout,'Rate');
water = strcmp(injphase,'w');
oil = strcmp(injphase,'o');
gas = strcmp(injphase,'g');

Qw = zeros(1,numG + 2); Qo = zeros(1,numG + 2); Qg = zeros(1,numG + 2);
qw = zeros(1,numG + 2); qo = zeros(1,numG + 2); qg = zeros(1,numG + 2);
WIw = zeros(1,numG + 2); WIo = zeros(1,numG + 2); WIg = zeros(1,numG + 2);
qk = zeros(numG + 2, Nc);
moldenswellw = zeros(1,numG + 2);
moldenswello = zeros(1,numG + 2);
moldenswellg = zeros(1,numG + 2);

Iinj = perm(2)*Gin;
Iprod = perm(numG+1)*Gout;

if water == 1
    Swinj = 1;
    zin = zeros(1,Nc);
else
    Swinj = 0;
end


if n < startshut
    % Inlet
    % Water Properties
    [cwin,viscwin,denswin,moldenswin] = ...
        WaterProps(T,p(2),ws,moldensw0,densw0,p0);
    Swin = Swinj;
    % Hydrocarbon Properties
    if sum(zin)==0
        xoin = zeros(1,Nc); xgin = zeros(1,Nc);
        moldensoin = 0; moldensgin = 0;
        viscoin = 0; viscgin = 0;
        Soin = 0;
```

```
    Sgin = 0;
else
    [xoin,xgin,Lin,Vin,Kin,Zoin,Zgin,dZodpin,dZgdpin, ...
    voloin,volgin,moldensoin,moldensgin,densoin,densgin] = ...
    PRFlash(Nc,zin,R,w,Tc,Pc,MW,Kbin,T,p(2),SE);

    [viscoin,viscgin] = ...
    LBCVisc(Nc,xoin,xgin,Tc,Pc,MW,T,vc,voloin,volgin);

    Soin =((1-Swin)*moldensgin*Lin)/(Lin*moldensgin+Vin*moldensoin);
    Sgin = 1-Swin-Soin;
end

if (sum(zin) + Swinj)==0
    WIinjw = 0; WIinjo = 0; WIinjg = 0;
else
    [krwin,kroin,krgin] = ...
    RelPerm(krwmax,krowmax,krogmax,krgmax,Swin,Soin,Sgin, ...
    Swc,Sorw,Sorg,Sgc,nw,now,nog,ng,RelPermtype, ...
    Swtab,krwtab,krowtab,Sgtab,krgtab,krogtab);

    WIinjw=Iinj*(krwin/viscwin);
    if sum(zin)==0
        WIinjo=0;
        WIinjg=0;
    else
        WIinjo=Iinj*(kroin/viscoin);
        WIinjg=Iinj*(krgin/viscgin);
    end
end

if constpressin == 1
    Qwin=WIinjw*(pbh-p(2));
    Qoin=WIinjo*(pbh-p(2));
    Qgin=WIinjg*(pbh-p(2));
    p(1) = pbh;
elseif constratein == 1
    WIinjw = 0; WIinjo = 0; WIinjg = 0;
    Qwin=Qin*water; Qoin=Qin*oil; Qgin=Qin*gas;
    if Qin > 0
        p(1) = p(2) + 1E6;
    else
        p(1) = p(2) - 1E6;
    end
end
qwin=moldenswin*Qwin; qoin=moldensoin*Qoin; qgin=moldensgin*Qgin;
qkin = QComponent(Qoin,Qgin,moldensoin,moldensgin,xoin,xgin);
moldensw_bh = moldenswin;
moldenso_bh = moldensoin;
moldensg_bh = moldensgin;

%Outlet
WIprodw=Iprod*(krw(numG+1)/viscw(numG+1));
WIprodo=Iprod*(kro(numG+1)/visco(numG+1));
WIprodg=Iprod*(krg(numG+1)/viscg(numG+1));
```

```
    if constpressout == 1
        Qwout=WIprodw*(pe-p(numG+1));
        Qoout=WIprodo*(pe-p(numG+1));
        Qgout =WIprodg*(pe-p(numG+1));
        p(numG+2) = pe;
    elseif constrateout == 1
        WIprodw = 0; WIprodo = 0; WIprodg = 0;
        Qwout=((krw(numG+1)/viscw(numG+1)) / ...
              ((krw(numG+1)/viscw(numG+1))+(kro(numG+1)/visco(numG+1))+ ...
              (krg(numG+1)/viscg(numG+1))))*Qout;
        Qoout=((kro(numG+1)/visco(numG+1)) / ...
              ((krw(numG+1)/viscw(numG+1))+(kro(numG+1)/visco(numG+1))+ ...
              (krg(numG+1)/viscg(numG+1))))*Qout;
        Qgout=((krg(numG+1)/viscg(numG+1))/ ...
              ((krw(numG+1)/viscw(numG+1))+(kro(numG+1)/visco(numG+1))+ ...
              (krg(numG+1)/viscg(numG+1))))*Qout;
        if Qout < 0
            p(numG+2) = p(numG+1) - 1E6;
        else
            p(numG+2) = p(numG+1) + 1E6;
        end
    end
    qwout=moldensw(numG+1)*Qwout;
    qoout=moldenso(numG+1)*Qoout;
    qgout=moldensg(numG+1)*Qgout;
    qkout = QComponent(Qoout,Qgout,moldenso(numG+1),moldensg(numG+1), ...
            xo(numG+1,:),xg(numG+1,:));

    moldensw_e = moldensw(numG+1);
    moldenso_e = moldenso(numG+1);
    moldensg_e = moldensg(numG+1);

elseif n>=startshut && n<startprod

    WIinjw = 0; WIinjo = 0; WIinjg = 0;
    WIprodw=Iprod*(krw(numG+1)/viscw(numG+1));
    WIprodo=Iprod*(kro(numG+1)/visco(numG+1));
    WIprodg=Iprod*(krg(numG+1)/viscg(numG+1));
    Qwin = 0; Qoin = 0; Qgin = 0;
    Qwout=WIprodw*(pe-p(numG+1));
    Qoout=WIprodo*(pe-p(numG+1));
    Qgout = WIprodg*(pe-p(numG+1));
    qwin=0; qoin=0; qgin=0;
    qkin = QComponent(Qoin,Qgin,moldenso(2),moldensg(2),xo(2,:),xg(2,:));
    qwout=moldensw(numG+1)*Qwout;
    qoout=moldenso(numG+1)*Qoout;
    qgout=moldensg(numG+1)*Qgout;
    qkout = QComponent(Qoout,Qgout,moldenso(numG+1),moldensg(numG+1), ...
            xo(numG+1,:),xg(numG+1,:));
    moldensw_bh = 0; moldenso_bh = 0; moldensg_bh = 0;
    moldensw_e = moldensw(numG+1);
    moldenso_e = moldenso(numG+1);
    moldensg_e = moldensg(numG+1);
```

```
else
    %Inlet
    WIinjw=Iinj*(krw(2)/viscw(2));
    WIinjo=Iinj*(kro(2)/visco(2));
    WIinjg=Iinj*(krg(2)/viscg(2));

    if constpressin == 1
        Qwin=WIinjw*(pbh-p(2));
        Qoin=WIinjo*(pbh-p(2));
        Qgin =WIinjg*(pbh-p(2));
        p(1) = pbh;
    elseif constratein == 1
        WIinjw = 0; WIinjo = 0; WIinjg = 0;
        Qwin=((krw(2)/viscw(2))/ ...
             ((krw(2)/viscw(2))+(kro(2)/visco(2))+(krg(2)/viscg(2))))*Qin;
        Qoin=((kro(2)/visco(2))/ ...
             ((krw(2)/viscw(2))+(kro(2)/visco(2))+(krg(2)/viscg(2))))*Qin;
        Qgin=((krg(2)/viscg(2))/ ...
             ((krw(2)/viscw(2))+(kro(2)/visco(2))+(krg(2)/viscg(2))))*Qin;
        if Qin < 0
            p(1) = p(2) - 1E6;
        else
            p(1) = p(2) + 1E6;
        end
    end
    qwin=moldensw(2)*Qwin; qoin=moldenso(2)*Qoin; qgin=moldensg(2)*Qgin;
    qkin = QComponent(Qoin,Qgin,moldenso(2),moldensg(2),xo(2,:),xg(2,:));

    moldensw_bh = moldensw(2);
    moldenso_bh = moldenso(2);
    moldensg_bh = moldensg(2);

    % Outlet
    % Water Properties
    [cwout,viscwout,denswout,moldenswout] = ...
    WaterProps(T,p(numG+1),ws,moldensw0,densw0,p0);
    Swout = Sw0;
    zout = z0;
    % Hydrocarbon Properties
    if sum(zout)==0
        xoout = zeros(1,Nc); xgout = zeros(1,Nc);
        moldensoout = 0; moldensgout = 0;
        viscoout = 0; viscgout = 0;
        Soout = 0;
        Sgout = 0;
    else
        [xoout,xgout,Lout,Vout,Kout,Zoout,Zgout,dZodpout,dZgdpout, ...
        voloout,volgout,moldensoout,moldensgout,densoout,densgout] = ...
        PRFlash(Nc,zout,R,w,Tc,Pc,MW,Kbin,T,p(numG+2),SE);
        [viscoout,viscgout] = ...
        LBCVisc(Nc,xoout,xgout,Tc,Pc,MW,T,vc,voloout,volgout);
        Soout =((1-Swout)*moldensgout*Lout) / ...
               (Lout*moldensgout+Vout*moldensoout);
        Sgout = 1-Swout-Soout;
    end
```

```
    if (sum(zout) + Swout)==0
        WIprodw = 0;
        WIprodo = 0;
        WIprodg = 0;
        krwout = 0;
        kroout = 0;
        krgout = 0;
    else
        [krwout,kroout,krgout] = RelPerm(krwmax,krowmax,krogmax,krgmax, ...
        Swout,Soout,Sgout,Swc,Sorw,Sorg,Sgc,nw,now,nog,ng,RelPermtype, ...
        Swtab,krwtab,krowtab,Sgtab,krgtab,krogtab);

        WIprodw=Iprod*(krwout/viscwout);
        if sum(zout)==0
            WIprodo=0;
            WIprodg=0;
        else
            WIprodo=Iprod*(kroout/viscoout);
            WIprodg=Iprod*(krgout/viscgout);
        end
    end

    if constpressout == 1
        Qwout=WIprodw*(pe-p(numG+1));
        Qoout=WIprodo*(pe-p(numG+1));
        Qgout=WIprodg*(pe-p(numG+1));
        p(numG+2) = pe;
    elseif constrateout == 1
        WIprodw = 0;
        WIprodo = 0;
        WIprodg = 0;
        Qwout=Qout*water;
        Qoout=Qout*oil;
        Qgout=Qout*gas;
        if Qout > 0
            p(numG+2) = p(numG+1) + 1E6;
        else
            p(numG+2) = p(numG+1) - 1E6;
        end
    end
    qwout=moldenswout*Qwout;
    qoout=moldensoout*Qoout;
    qgout=moldensgout*Qgout;
    qkout = QComponent(Qoout,Qgout,moldensoout,moldensgout,xoout,xgout);
    moldensw_e = moldenswout;
    moldenso_e = moldensoout;
    moldensg_e = moldensgout;
end

Qw(2) = Qwin; Qw(numG+1) = Qwout; qw(2) = qwin; qw(numG+1) = qwout;
WIw(2) = WIinjw; WIw(numG+1) = WIprodw;
moldenswellw(2) = moldensw_bh; moldenswellw(numG+1) = moldensw_e;
Qo(2) = Qoin; Qo(numG+1) = Qoout; qo(2) = qoin; qo(numG+1) = qoout;
WIo(2) = WIinjo; WIo(numG+1) = WIprodo;
moldenswello(2) = moldenso_bh; moldenswello(numG+1) = moldenso_e;
```

```
Qg(2) = Qgin; Qg(numG+1) = Qgout; qg(2) = qgin; qg(numG+1) = qgout;
WIg(2) = WIinjg; WIg(numG+1) = WIprodg;
moldenswellg(2) = moldensg_bh; moldenswellg(numG+1) = moldensg_e;
qk(2,:) = qkin; qk(numG+1,:) = qkout;
end
```

Appendix B – Discretized Equations

Discretized Pressure Equation (3.25)

$$
-\left\{\begin{array}{l}
\theta\left[T_{w_{j-1/2}}{}^{n}\left(p_{o_{j-1}}{}^{n+1}-P_{cow_{j-1}}{}^{n}\right)-(T_{w_{j+1/2}}+T_{w_{j-1/2}})^{n}\left(p_{o_j}{}^{n+1}-P_{cow_j}{}^{n}\right)+T_{w_{j+1/2}}{}^{n}\left(p_{o_{j+1}}{}^{n+1}-P_{cow_{j+1}}{}^{n}\right)\right] \\
+\left[T_{o_{j-1/2}}{}^{n}p_{o_{j-1}}{}^{n+1}-(T_{o_{j+1/2}}+T_{o_{j-1/2}})^{n}p_{o_j}{}^{n+1}+T_{o_{j+1/2}}{}^{n}p_{o_{j+1}}{}^{n+1}\right] \\
+\left[T_{g_{j-1/2}}{}^{n}\left(p_{o_{j-1}}{}^{n+1}+P_{cog_{j-1}}{}^{n}\right)-(T_{g_{j+1/2}}+T_{g_{j-1/2}})^{n}\left(p_{o_j}{}^{n+1}+P_{cog_j}{}^{n}\right)+T_{g_{j+1/2}}{}^{n}\left(p_{o_{j+1}}{}^{n+1}+P_{cog_{j+1}}{}^{n}\right)\right]
\end{array}\right\}
$$

$$
+\frac{V_{r_j}}{\Delta t}\left(\phi_j{}^{n+1}\psi_j{}^{n+1}-\phi_j{}^{n}\psi_j{}^{n}\right)=\theta q_{w_j}{}^{n}+q_{o_j}{}^{n}+q_{g_j}{}^{n}
$$

Discretized Form of Composition Equation (3.55)

$$
z_i{}^{l+1}=\frac{\left\{\begin{array}{l}
x_{io}\left[T_{o_{j-1/2}}{}^{n}p_{o_{j-1}}{}^{n+1}-(T_{o_{j+1/2}}+T_{o_{j-1/2}})^{n}p_{o_j}{}^{n+1}+T_{o_{j+1/2}}{}^{n}p_{o_{j+1}}{}^{n+1}\right] \\
+x_{ig}\left[T_{g_{j-1/2}}{}^{n}\left(p_{o_{j-1}}{}^{n+1}+P_{cog_{j-1}}{}^{n}\right)-(T_{g_{j+1/2}}+T_{g_{j-1/2}})^{n}\left(p_{o_j}{}^{n+1}+P_{cog_j}{}^{n}\right)+T_{g_{j+1/2}}{}^{n}\left(p_{o_{j+1}}{}^{n+1}+P_{cog_{j+1}}{}^{n}\right)\right]
\end{array}\right\}+\frac{V_r}{\Delta t}\phi^{n}z_i{}^{n}(\xi_o S_o+\xi_g S_g)^{n}+q_i}{\left\{\begin{array}{l}
\left[T_{o_{j-1/2}}{}^{n}p_{o_{j-1}}{}^{n+1}-(T_{o_{j+1/2}}+T_{o_{j-1/2}})^{n}p_{o_j}{}^{n+1}+T_{o_{j+1/2}}{}^{n}p_{o_{j+1}}{}^{n+1}\right] \\
+\left[T_{g_{j-1/2}}{}^{n}\left(p_{o_{j-1}}{}^{n+1}+P_{cog_{j-1}}{}^{n}\right)-(T_{g_{j+1/2}}+T_{g_{j-1/2}})^{n}\left(p_{o_j}{}^{n+1}+P_{cog_j}{}^{n}\right)+T_{g_{j+1/2}}{}^{n}\left(p_{o_{j+1}}{}^{n+1}+P_{cog_{j+1}}{}^{n}\right)\right]
\end{array}\right\}+\frac{V_r}{\Delta t}\phi^{n}(\xi_o S_o+\xi_g S_g)^{n}+q_o+q_g}
$$

Discretized and Expanded form of Pressure Update Equation (3.50)

$$\left[\frac{2\pi r_{j-1/2}\Delta z}{r_j - r_{j-1}}(\theta T_{w_{j-1}} + T_{o_{j-1}} + T_{g_{j-1}})^l\right]\Delta p_{o_{j-1}} + \left\{-\left[\frac{2\pi r_{j-1/2}\Delta z}{r_j - r_{j-1}}(\theta T_{w_{j-1}} + T_{o_{j-1}} + T_{g_{j-1}})^l\right] - \left[\frac{2\pi r_{j+1/2}\Delta z}{r_{j+1} - r_j}(\theta T_{w_j} + T_{o_j} + T_{g_j})^l\right]\right\}\Delta p_{o_j}$$

$$+\left[\frac{2\pi r_{j+1/2}\Delta z}{r_{j+1} - r_j}(\theta T_{w_j} + T_{o_j} + T_{g_j})^l\right]\Delta p_{o_{j+1}} = \left[\frac{2\pi r_{j-1/2}\Delta z}{r_j - r_{j-1}}(\theta T_{w_{j-1}} + T_{o_{j-1}} + T_{g_{j-1}})^l\right] {p_{o_{j-1}}}^l +$$

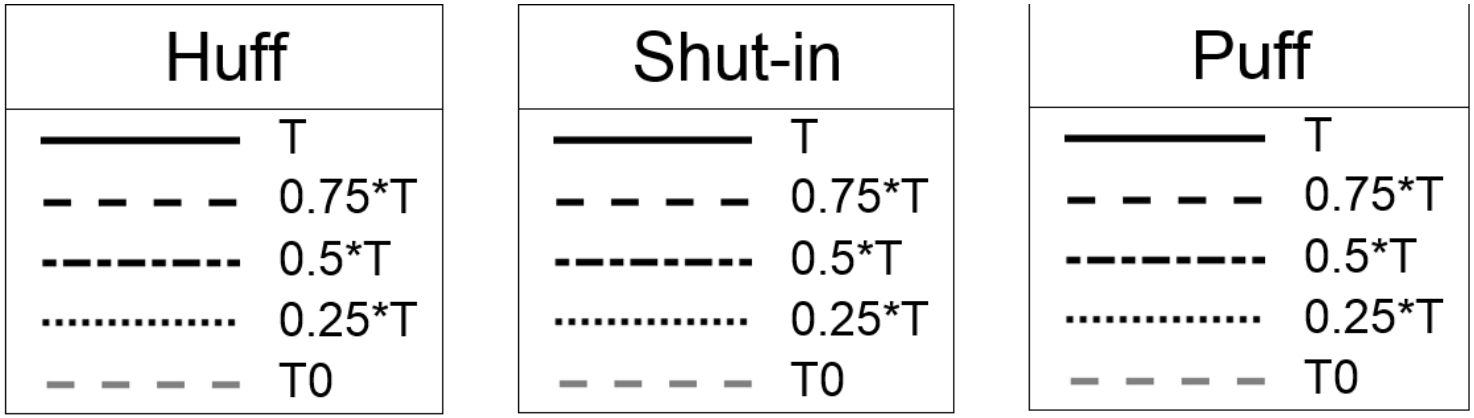
$$\left\{-\left[\frac{2\pi r_{j-1/2}\Delta z}{r_j - r_{j-1}}(\theta T_{w_{j-1}} + T_{o_{j-1}} + T_{g_{j-1}})^l\right] - \left[\frac{2\pi r_{j+1/2}\Delta z}{r_{j+1} - r_j}(\theta T_{w_j} + T_{o_j} + T_{g_j})^l\right]\right\} {p_{o_j}}^l + \left[\frac{2\pi r_{j+1/2}\Delta z}{r_{j+1} - r_j}(\theta T_{w_j} + T_{o_j} + T_{g_j})^l\right] {p_{o_{j+1}}}^l$$

$$-\left(\begin{array}{l}\left[\frac{2\pi r_{j-1/2}\Delta z}{r_j - r_{j-1}}(\theta T_{w_{j-1}} + T_{o_{j-1}} + T_{g_{j-1}})^l\right] {P_{cow}}_{j-1}^{\ n} + \left\{-\left[\frac{2\pi r_{j-1/2}\Delta z}{r_j - r_{j-1}}(\theta T_{w_{j-1}} + T_{o_{j-1}} + T_{g_{j-1}})^l\right] - \left[\frac{2\pi r_{j+1/2}\Delta z}{r_{j+1} - r_j}(\theta T_{w_j} + T_{o_j} + T_{g_j})^l\right]\right\} {P_{cow}}_j^{\ n} \\ +\left[\frac{2\pi r_{j+1/2}\Delta z}{r_{j+1} - r_j}(\theta T_{w_j} + T_{o_j} + T_{g_j})^l\right] {P_{cow}}_{j+1}^{\ n}\end{array}\right)$$

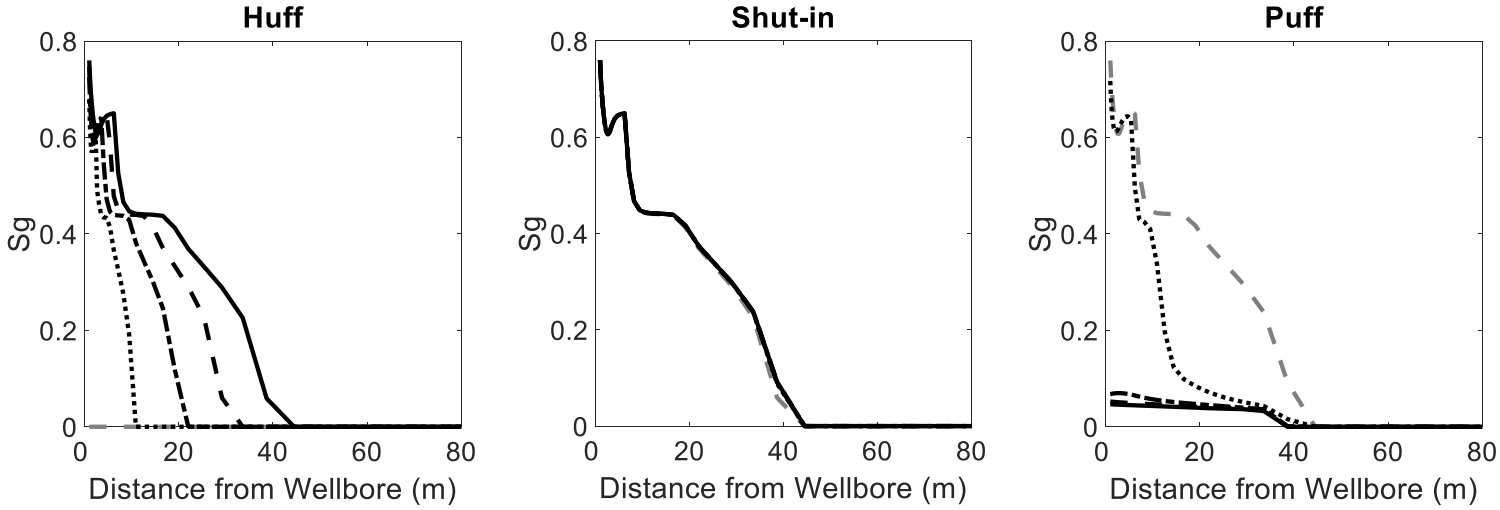
$$+\left(\begin{array}{l}\left[\frac{2\pi r_{j-1/2}\Delta z}{r_j - r_{j-1}}(\theta T_{w_{j-1}} + T_{o_{j-1}} + T_{g_{j-1}})^l\right] {P_{cog}}_{j-1}^{\ n} + \left\{-\left[\frac{2\pi r_{j-1/2}\Delta z}{r_j - r_{j-1}}(\theta T_{w_{j-1}} + T_{o_{j-1}} + T_{g_{j-1}})^l\right] - \left[\frac{2\pi r_{j+1/2}\Delta z}{r_{j+1} - r_j}(\theta T_{w_j} + T_{o_j} + T_{g_j})^l\right]\right\} {P_{cog}}_j^{\ n} \\ +\left[\frac{2\pi r_{j+1/2}\Delta z}{r_{j+1} - r_j}(\theta T_{w_j} + T_{o_j} + T_{g_j})^l\right] {P_{cog}}_{j+1}^{\ n}\end{array}\right)$$

$$+\theta {q_{w_j}}^l + {q_{o_j}}^l + {q_{g_j}}^l - \frac{V_{r_j}}{\Delta t}\left[\left(\phi\psi\right)_j^{\ l} - \left(\phi\psi\right)_j^{\ n}\right]$$

Appendix C – Example Results Plot



Legend



Example Plots

The legend which is provided in Chapter 4 is repeated here for reference, and a sample of example plots of gas saturation are provided for description on how to interpret these plots. The plots are all shown in a row of 3 plots, the first plot showing the huff phase, the second plot showing the shut-in phase, and the third plot showing the puff phase. Each plot spans a time interval of the simulation according to the corresponding phase of the process. The first plot (huff phase) covers the time from the beginning of the simulation, until the end of the huff phase. The second plot (shut-in phase) covers the time from the beginning of the shut-in phase (which is the end of the huff phase) until the end of the shut-in phase. The third plot (puff phase) covers the time from the beginning of the puff phase (which is the end of the shut-in phase) until the end of the simulation. The legend is the same for each of the plots, where each plot has 5 separate lines. $T0$ shows the state of the parameter at the beginning of the phase of the process, and then there are four lines showing the parameter at 4 fractions of the length of the simulation time of that phase, represented by T. The state of the parameter is shown at one quarter, one half, three quarters, and then the end of the simulation time of each phase. The state at time T of one phase, is the same as the state of the parameter at time $T0$ of the subsequent phase.

For instance, at the end of the shut-in phase, the gas saturation in the example plots is the same as the beginning of the puff phase. In general, the huff phase is when gas is injected, and by looking at the 4 lines corresponding to different fractions of the simulation time of the huff phase in the example plots, the gas can be seen travelling into the reservoir. In these simulations, the shut-in phase typically just levels out the pressure in the reservoir, which is why not much change is seen in the shut-in phase. The puff phase is where oil and

gas are produced, which is why in the example plots the gas which was seen to be injected into the reservoir in the huff phase, is now moving towards the wellbore as it is being produced.