IMPES MODELING OF VOLUMETRIC DRY GAS RESERVOIRS WITH MOBILE WATER

A Thesis

by

SAEED FORGHANY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2004

Major Subject: Petroleum Engineering

IMPES MODELING OF VOLUMETRIC DRY GAS RESERVOIRS WITH MOBILE WATER

A Thesis

by

SAEED FORGHANY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

David S. Schechter (Chair of Committee)	Robert R. Berg (Member)
Thomas A. Blasingame (Member)	Stephen Holditch (Head of Department)

May 2004

Major Subject: Petroleum Engineering

ABSTRACT

IMPES Modeling of Volumetric Dry Gas Reservoirs with Mobile Water. (May 2004)

Saeed Forghany, B.Sc., Sahand University of Technology (Iran)

Chair of Advisory Committee: Dr. David. S. Schechter

As the importance of natural gas as a resource increases, the importance of volumetric dry gas reservoirs with mobile water as the dominant gas reservoir types will also increase.

This research developed an efficient, user-friendly simulation program specifically designed to model two-phase flow of gas and water in these reservoirs.

Since fluid compression and viscous forces are the dominant parameters that control fluid movement in a dry gas reservoir, we used the Implicit Pressure and Explicit Saturation (IMPES) formulation of flow equations in which neither gravity nor capillary pressure terms are pertinent. Therefore, the IMPES approach showed greater stability for all cases considered in this work. The developed simulator is a Visual Basic Application (VBA) code for which the users can obsereve the results in a pertinent Microsoft Excel file.

This program allows users to study the depletion behavior of volumetric dry gas reservoirs with mobile water as efficiently and accurately as is now possible in more expensive commercially available reservoir simulators. The program was validated by comparing the results with a well-recognized commercial reservoir simulator (CMG). The results of a battery of tests of this simulator matched very well with results of the commercial reservoir simulator for all tested schemes including different simulation plans; reservoir, grid and fluid data; and well configurations.

The observed applicability of the program suggests when dealing with volumetric dry gas reservoirs with mobile water there is no need to employ more expensive commercial reservoir simulators, as the program can reliably be used for any simulation scheme of this case. Furthermore, the program can later be applied in a more robust reservoir simulator as the part that handles dry gas cases.

DEDICATION

To my parents and to my little brother for all of their love, care and enthusiasm.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere appreciation to the people who have assisted me throughout my studies.

I would specifically like to thank my advisor, Dr. David Schechter for his guidance and encouragement throughout my research. His patience over the period of doing my research is also greatly appreciated.

I would also like to acknowledge Dr. Thomas Blasingame and Dr. Robert Berg for their participation in my research as members of my advisory committee.

I also greatly appreciate the considerable aid of Dr. Erwin Putra in the Naturally Fractured Reservoirs group whose helpful comments always helped me make progress in my research.

Finally, I want to really thank all my friends especially the ones in the Naturally Fractured Reservoirs group for making my graduate years pleasant.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

Natural gas is becoming an increasingly important source of the world's energy. In recent years, natural gas use has grown the fastest of all the fossil fuels, and it will continue to grow rapidly for several decades. World gas consumption grows by 3.3 percent/year compared with 2.2 percent/year for oil and 2.1 percent/year for coal. This higher growth rate can be attributed to several factors such as the fact that natural gas, including unconventional gas, is available in abundant quantities in many parts of the world and also the lower price of gas relative to other fuels makes it attractive to many gas operators and consumers.[1] **Fig. 1.1** demonstrates the world supply and consumption of natural gas over the past few decades up to the present time. From these trends and also this fact that oil production has already passed its peak it can be easily concluded that even with this same tendency, natural gas will be the main source of energy that is going to power the world in the next few decades. Regarding this fact, the significance of developing tools for handling gas reservoirs is specified. A numerical simulation program is a tool which, when properly applied, can provide an estimation of reservoir performance under a variety of user specified conditions and constraints.

1.1 Background

Reservoir simulation is the art and science of using numerical techniques to solve the equations for mass flow in porous media, considering the appropriate initial and boundary conditions.[2, 3] Thanks to different technical advances such as gridding, fluid modeling, numerical approximations, linear and nonlinear solvers, reservoir and geolog-

---

This thesis follows the style and format of the *Journal of SPE Reservoir Evaluation and Engineering.*

ical modeling, etc. simulators are getting more accurate, realistic, robust and user-friendly. Simulation of three-dimensional flow of different phases in a reservoir requires solving the system of coupled, nonlinear partial differential equations. These equations arise from application of the conservation of mass principle to an oil-water-gas system.

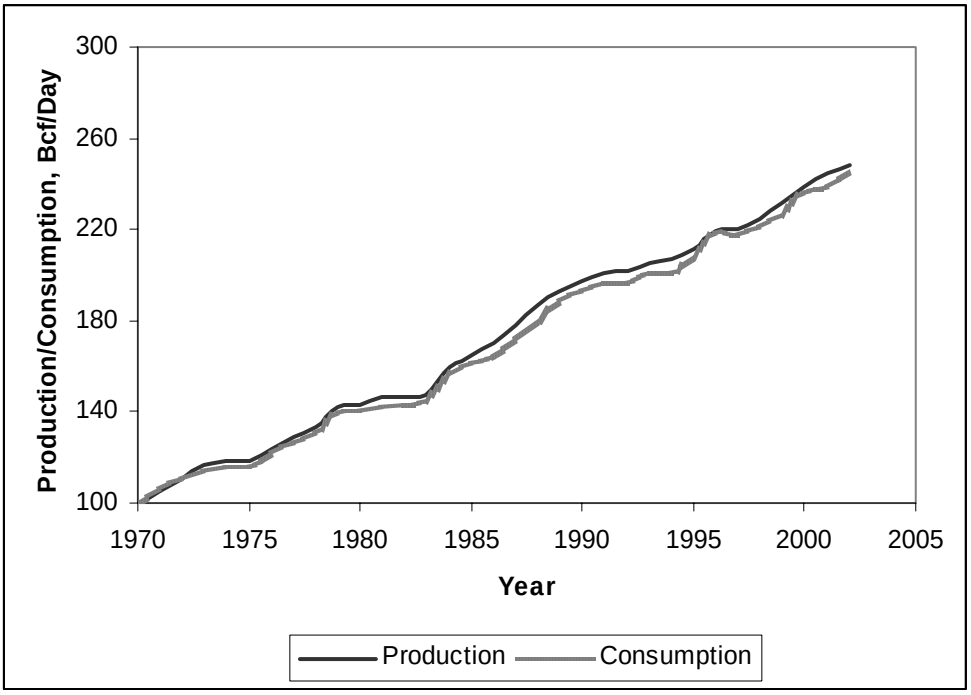


Fig. 1.1- World supply and consumption of natural gas (*from bp.com*[4])

For most practical situations the flow equations can not be solved analytically. Instead, the partial differential equations are approximated by algebraic equations known as finite difference equations. The finite difference equations are obtained by replacing derivatives with approximations derived from truncated Taylor series expansions.[5, 6] In this study, we develop a numerical reservoir simulator that handles dry gas reservoirs

and we have validated this model by comparing the results with a commercially available reservoir simulator. [7]The significance of the work is discussed in the following section. When we use "dry gas", we are referring to a reservoir gas made up primarily of methane with some intermediate-weight hydrocarbon molecules. The dry-gas-phase diagram in **Fig. 1.2** indicates that, because of this composition, dry gases do not undergo phase changes following a pressure reduction and therefore are solely gases in the reservoir and at the surface separator conditions. In this sense, "dry" does not refer to the absence of water but indicates that no liquid hydrocarbons form in the reservoir, wellbore or surface equipment during production.

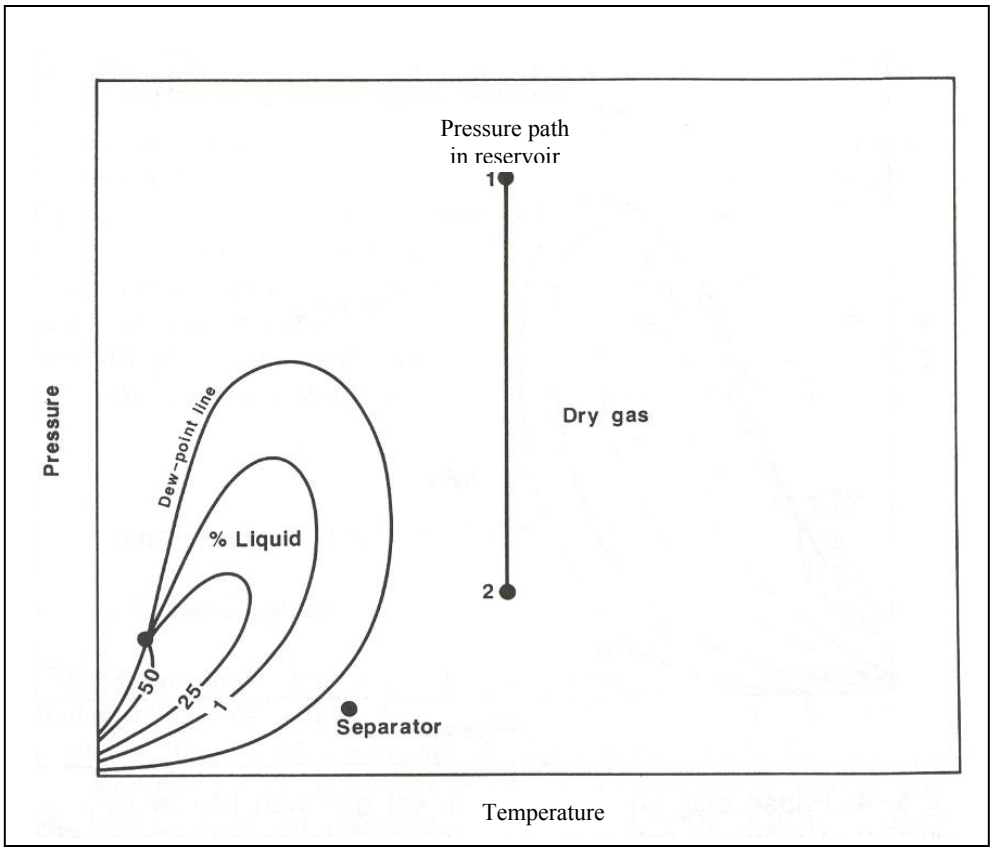


Fig. 1.2- Phase diagram for a dry gas reservoir (from McCain[11])

Dry gas reservoirs are categorized into the following titles:

1. Dry gas volumetric reservoirs.
2. Dry gas reservoirs with water influx.
3. Dry gas volumetric geopressured reservoirs.[10]

Volumetric dry gas reservoirs: A volumetric dry gas reservoir, as the name implies, is completely enclosed by low-permeability or completely impermeable barriers and does not receive pressure support from external sources, such as an encroaching aquifer. In addition, if the expansion of rock and the connate water are negligible, then the primary source of pressure maintenance is gas expansion resulting from gas production and the subsequent pressure reduction. In a volumetric dry gas reservoir the reservoir PV occupied by gas remains constant over the productive life of the reservoir.

Dry gas reservoirs with water influx: Many gas reservoirs are not completely closed but are subjected to some natural water influx from an aquifer. Water encroachment occurs when the pressure at the reservoir/aquifer boundary is reduced following gas production from the reservoir. In gas reservoirs with water influx, pore volume decreases by an amount equal to the net volume of water entering the reservoir and the remaining unproduced.

Dry gas volumetric geopressured reservoirs: In deep, geopressured gas reservoirs the compressibility of the gas is much smaller than that of volumetric reservoirs and does not totally dominate production performance. In geopressured systems the compressibility of the rock and water may be just as large as the gas. Some investigators have postulated that water will be released from shales as the reservoir compacts during depletion. [11]This would result in an internal water drive similar to aquifer influx. Because the reservoir rock is usually highly compressible and undercompacted, the decrease in pore volume during depletion may be very non-linear. Along with the rock compressibility, the absolute permeability may also decrease with declining pressure.

The creation of an abnormally pressured reservoir requires unusual geologic conditions. The reservoir is isolated from hydrostatic communication with the surface and is usually

geothermal as well. The isolation could result from shale totally surrounding the sand or from faulting, either of which would coplicate reservoir performance and analysis.

For abnormally or *geopressured* reservoirs, pressure gradients often approach values equal to the overburden pressure gradient (i.e., ~1.0 psi/ft). [8, 9]

Among these types of dry gas reservoirs, in this study we will focus on volumetric reservoir.

1.2 Research Methodology

This research is primarily accomplished by developing a computer program. The program is a 1900-line-long *VBA* code that takes the input data from a *Notepad* format file. This program simulates dry gas volumetric reservoirs. The program handles the general case of a volumetric dry gas reservoir including but not limited to any combination of boundary conditions, wells, production/injection plans, reservoir dimensions, reservoir life, etc. The user can easily make any required changes both in input data and in some settings of the program in the input file and then run the program just like any commercial simulator. Once the run is complete, numeric results and values are all stored in another *Notepad* file that can immediately be retrieved by the user. All graphic results are plotted and/or tabulated in the *MS Excel* file which is attached to the Visual Basic module. Different attributes of the code are discussed in detail in the next chapter.

The developed simulator is then validated by comparing the results with CMG software for some different simulation schemes. [8]

The comparisons showed that in worse cases the difference of the code results from CMG results is equal to or less than 0.8 percent, 0.9 percent, 0.5 percent, 0.4 percent and 0.4 percent for q_g, q_w, p_{wf}, G_p and W_p respectively. Therefore the simulator can be used when dealing with dry gas reservoirs with confidence. This code, afterwards, can be included in a more robust reservoir simulator as the part that handles dry gas cases. Because it provides reliable results for what it has been designed for.

The main significance of developing simulation programs is that this activity provides better understanding of what a commercially available simulator. Developing simulation codes helps a reservoir engineer analyze the results of a simulation case more reasonably and consequentially, this permits the reservoir engineer to make more realistic decisions. Some reservoir engineers view simulation software as a "black box". [5, 13, 14] Developing a simulation code helps the reservoir engineer to understand reservoir simulator is an engineering tool and must be applied with appropriate engineering judgment.[15]

In this report the study has been divided into chapters. In Chapter II, we provide a detailed description of the code and pertinent input and output files. Chapter III consists of the derivation diffusivity equation, the gas material balance equations, and the final IMPES flow equation that is applied in the coding. Chapter IV discusses and analyzes the results of the developed simulator we came up with for a few schemes and also comparison of them with results of the same sets of cases in CMG. The conclusions will follow this chapter. A listing of the *Visual Basic* code is included in the appendix.

CHAPTER II

IMPES FORMULATION FOR TWO-PHASE FLOW

In this chapter we will first derive the diffusivity equation from basic reservoir engineering relationships. The manipulation of the material balance equations in order to derive the final IMPES flow equations will follow. The averaging of flow equation parameters will be discussed as well.

2.1 Diffusivity Equation

In order to use differential; equation for predicting the behavior of a reservoir it is necessary to solve these relations subject to the appropriate boundary conditions. Only for the simplest cases involving homogeneous reservoirs and very regular boundaries (such as a circular boundary around a single well) can solutions be obtained by the classic methods of mathematical physics.[5] The set of the difference equations that are amenable to solution by computers constitute a numerical model.

The three basic reservoir engineering relationships that initiate the manipulation that leads us to gas flow equations are: [15, 16]

1. Darcy's law

$$q_g = c_x \frac{kA}{B_{gi}\mu_i}\frac{dp}{dx}$$ (Differential form for linear gas flow) ..... (2.1)

2. Mass Continuity Equation

$$\nabla \bullet (\rho_g \vec{u}) = -\frac{\partial(\phi\rho)}{\partial t}$$ ……..…… (2.2)

3. Equation of State

$\rho = f(p)$ (Isothermal) ……..…….. (2.3)

The derivation of diffusivity equation is based on incorporating these three relationships. For isothermal condition, fluid compressibility is defined as:

$$c = -\frac{1}{V}\frac{dV}{dp} = \frac{1}{\rho}\frac{d\rho}{dp} \quad \text{……. (2.4)}$$

and the rock compressibility is

$$c_f = \frac{1}{\phi}\frac{d\phi}{dp} \quad \text{…….....……..... (2.5)}$$

The derivation of the diffusivity equation for gas flow starts by substituting Darcy's law into the continuity equation which yields the generalized density form of the diffusivity equation: [16]

$$\nabla \bullet \left[\frac{\rho k}{\mu}\right] = \frac{\partial(\phi\rho)}{\partial t} \quad \text{…….....……..... (2.6)}$$

Recalling the definition of gas density, we have

$$\rho_g = \frac{pM}{zRT} \quad \text{.....(2.7)}$$

Substituting Eq. 2.7 into Eq. 2.6, and eliminating the $\frac{M}{RT}$ terms, we obtain

$$\nabla \bullet \left[\frac{k}{\mu}\frac{p}{z}\nabla p\right] = \frac{\partial}{\partial t}\left[\phi\frac{p}{z}\right] \quad \text{.....(2.8)}$$

If we assume that the effective permeability, *k*, is constant which is a very reasonable assumption for gas reservoirs and we expand the righthand-side term using the product rule, then Eq. 2.8 becomes:

$$\nabla \bullet \left[\frac{p}{\mu z}\nabla p\right] = \frac{1}{k}\frac{\partial}{\partial t}\left[\phi\frac{p}{z}\right] = \frac{1}{k}\left[\frac{p}{z}\frac{\partial\phi}{\partial t} + \phi\frac{\partial}{\partial t}\left[\frac{p}{z}\right]\right] \quad \text{.....(2.9)}$$

Expanding the time derivative terms using the chain rule yields

$$\nabla \bullet \left[\frac{p}{\mu z} \nabla p \right] = \frac{1}{k} \left[\frac{p}{z} \frac{\partial \phi}{\partial t} \frac{\partial p}{\partial t} + \phi \frac{\partial}{\partial p} \left[\frac{p}{z} \right] \frac{\partial p}{\partial t} \right] \quad \text{.....(2.10)}$$

Factoring out the porosity, we have

$$\nabla \bullet \left[\frac{p}{\mu z} \nabla p \right] = \frac{\phi}{k} \left[\frac{p}{z} \frac{1}{\phi} \frac{\partial \phi}{\partial p} \frac{\partial p}{\partial t} + \frac{\partial}{\partial p} \left[\frac{p}{z} \right] \frac{\partial p}{\partial t} \right] \quad \text{.....(2.11)}$$

Recalling the definition of pore-volume compressibility, c_f, we have

$$c_f = \frac{1}{\phi} \frac{\partial \phi}{\partial p} \quad \text{.....(2.12)}$$

Substituting Eq. 2.12 into Eq. 2.11, we obtain

$$\nabla \bullet \left[\frac{p}{\mu z} \nabla p \right] = \frac{\phi}{k} \left[\frac{p}{z} c_f \frac{\partial p}{\partial t} + \frac{\partial}{\partial p} \left[\frac{p}{z} \right] \frac{\partial p}{\partial t} \right] \quad \text{.....(2.13)}$$

Recalling the definition of isothermal gas compressibility, c_g, we have

$$c_g = \frac{1}{p} - \frac{1}{z} \left[\frac{\partial z}{\partial p} \right]_T \quad \text{.....(2.14)}$$

The alternative form of the definition of gas compressibility is (again for isothermal conditions, but dropping the *T* subscript)

$$c_g = \frac{z}{p} \frac{\partial}{\partial p} \left[\frac{p}{z} \right] \quad \text{.....(2.15)}$$

Rearranging Eq. 2.15, we have

$$\frac{\partial}{\partial p} \left[\frac{p}{z} \right] = \frac{p}{z} c_g \quad \text{.....(2.16)}$$

Substituting Eq. 2.16 into Eq. 2.17, we obtain

$$\nabla \bullet \left[\frac{p}{\mu z} \nabla p \right] = \frac{\phi}{k} \left[\frac{p}{z} c_f \frac{\partial p}{\partial t} + \frac{p}{z} c_g \frac{\partial p}{\partial t} \right] \quad \text{.....(2.17)}$$

Recalling the definition of total compressibility, $c_t = c_g + c_f$, and substituting this identity into Eq. 2.17 gives us

$$\nabla \bullet \left[\frac{p}{\mu z} \nabla p \right] = \frac{\phi c_t}{k} \frac{p}{z} \frac{\partial p}{\partial t} \quad \text{.....(2.18)}$$

Eq. 2.18 is the generalaized diffusivity equation for gas flow. [16]

In order to solve problems which involve this equation, the finite difference method can be used. This equation is discretized into the following finite difference form:

$$\frac{p_{i-1}^{n+1} - 2\,p_i^{n+1} + p_{i+1}^{n+1}}{(\Delta x)^2} = \frac{\phi\mu c}{k}\,\frac{p_i^{n+1} - p_i^{n}}{\Delta t} \qquad \text{..... (2.19)}$$

The *n* superscript indicates the *old* time level. All of the unknowns have already been solved at the n^{th} time level. The *n+1* superscript indicates the *new* time level. We want to solve for these unknown values at the new time level.

Eq. 2.19 is called an *implicit* finite difference equation since it involves more than one unknown. Three unknowns, p_{i-1}^{n+1}, p_i^{n+1}, and p_{i+1}^{n+1} occur because we chose the *n+1* time level to discretize the left-hand side of the equation. A template of this finite difference equation is shown in **Fig. 2.1**.

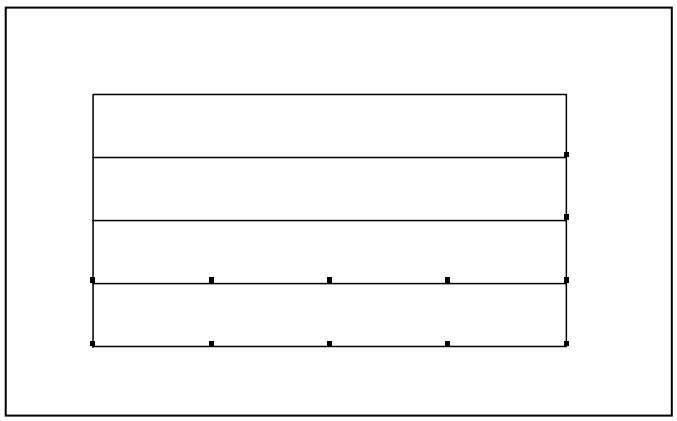


Fig. 2.1- Finite difference template used in the finite difference equation.

We know the value of *p* at the *n*-time level and we are trying to determine the values of *p* at the *n+1* time level.

2.2 Flow Equations

By replacing the differential equations with difference equations, the partial differential equations that describe fluid flow in reservoirs can be solved numerically. Discretization of differential equations subdivides distance and time into definite, specified increments. In this section we will manipulate the differential equations to treat the reservoir as if it were composed of discrete volume elements.[15, 17] The discrete approach we are going to use amounts to discretizing the continuity equation. Each gridblock has a definite boundary and the pressure represents the average pressure in the gridblock for material balance purposes. For simplicity the derivation of the finite difference equations for 1-D gas flow (two phases) are shown here in this section. When deriving IMPES flow equations for the code the 3-D case will be handled.

Discretization starts with developing a finite difference equation for the flow of gas and water in a grid system. This finite difference equation conserves mass, so it is called the material balance equation for gridblock. Beginning with a statement of the continuity, material balance equation would be:

Net rate of Flow in (scf/D) = Rate of Accumulation (scf/D)

If the system is defined as having constant density at standard conditions, the units of this equation will be in standard cubic feet, scf, rather than working with mass.

The pore volume of the gridblock ***i*** is:

$$V_p = \Delta x \Delta y h \phi \quad \text{.....} \quad (2.20)$$

The gas-in-place can be calculated as:

$$GIP = \frac{V_p S_g}{B_g} \quad \text{.....} \quad (2.21)$$

where

GIP = standard (stock tank) gas in place, scf

V_p = pore volume of the gridblock, rcf

S_g = average gas saturation of the gridblock, fraction

B_g = formation volume factor at the average gridblock pressure, rcf/scf

ϕ = average porosity of the gridblock, fraction

The rate of accumulation of gas including the production terms during the timestep is:

$$= \frac{1}{\Delta t}\left[\left(\frac{V_p S_g}{B_g}\right)^{n+1} - \left(\frac{V_p S_g}{B_g}\right)^{n}\right] + q_g \quad \text{(2.22)}$$

The quantities *Vp*, *Sg*, and *Bg* are evaluated at the time indicated by the superscripts, before and after the time step. The quantity in brackets is the accumulation of oil in the gridblock during the timestep. Dividing by Δt puts the right-hand side on the rate basis. The left side of the continuity equation deals with flow rates. It can be stated as:

$$\text{Net rate of flow in} = q_{left} + q_{right} \quad \text{(2.23)}$$

Where positive *q* is flow into the gridblock, negative flow is out of the gridblock. Usually fluid is flowing through the gridblock, so one term is positive and the other is negative. Our material balance equation can now be given as:

$$q_{lft} + q_{rht} = \frac{1}{\Delta t}\left[\left(\frac{V_p S_g}{B_g}\right)^{n+1} - \left(\frac{V_p S_g}{B_g}\right)^{n}\right] + q_g \quad \text{(2.24)}$$

Now, we need an expression for flowrate. We use Darcy's law for flow between the centers of the gridblocks. The flow distance, Δx, is the distance between the centers of the gridblocks. The gridblock pressures are taken to be at the center of the gridblocks.
Flow from the right, from gridblock i+1 to gridblock i, is

$$q_{rht}(scf/D) = \frac{u_{rht} A}{B_g} \quad \text{(2.25)}$$

$$q_{right} = \left(\frac{0.00633 k k_{rg}}{\mu_g B_g}\right)\left(\frac{p_{i+1} - p_i}{\Delta x}\right)(\Delta y h)$$

$$= \left(\frac{0.00633 k h \Delta y}{\Delta x}\right)\left(\frac{k_r}{B\mu}\right)_g (p_{i+1} - p_i) \quad \text{(2.26)}$$

The first factor is constant with time. It also applies to both phases. This is called "transmissibility" and is saved separately in the computations.
Now we define:

$$T_{i+1/2} = 0.00633\frac{kh\Delta y}{\Delta x} \quad \text{……..…….....…} \quad (2.27)$$

The subscript *i+1/2* indicates that the coefficient applies between gridblocks *i* and *i+1*. We will replace *i+1/2* with E, for the "east" direction. The notation for transmissibility can be represented as follows:

$$T_E = \frac{0.00633kh\Delta y}{\Delta x} \quad \text{……..……..……} \quad (2.28)$$

For a 3-D flow, we will use the following directional notation:

i+1/2= E

i-1/2 =W

j+1/2=N

j-1/2 =S

k+1/2=B*k-1/2* =T

The next factor in Eq. 2.23 is called mobility, λ . Its value changes with time and is defined as:

$$\lambda_{gi+1/2} = \left(\frac{k_r}{B\mu}\right)_{g\,i+1/2} = \left(\frac{k_r}{B\mu}\right)_{gE} \quad \text{……..……} \quad (2.29)$$

where:

$$(Bg)_E = (Bg_i + Bg_{i+1})/2$$

$$(\mu_g)_E = (\mu_{gi} + \mu_{gi+1})/2$$

$$(k_{rg})_E = \text{upstream } k_{rg}$$

2.3 Averaging of Flow Equation Terms

The following elements of the diffusivity equation need to be averaged:

1. Absolute permeability
2. Relative permeability of both phases
3. Viscosity of both phases
4. Porosity[3]

There is no unique way to choose the values of $\lambda_{i+1/2}, k_{i+1/2}$ etc. In general the values are averaged in such a way that they give the most accurate values possible for the flow rate and accumulation terms. In this case, from literature the properties are averaged as given in **Table 2.1**. The methodology of averaging is presented in *Aziz* and *Settari*'s book.

Table 2.1-Averaging of parameters.

Averaged Parameter	**Method of Averaging**	**Units**
Absolute Permeability	Harmonic Averaging	md
Relative Permeability	Upstream Weighting	-
Porosity	Arithmetic Averaging	-
Viscosity	Arithmetic Averaging	cp
Formation Volume Factor	Arithmetic Averaging	rcf/scf – STB/scf

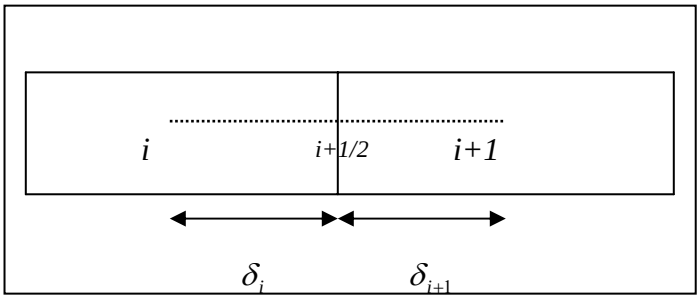


Fig. 2.2- Harmonic averaging of permeability

In case of a single fluid flow, by summing the flow rate from grid center *i* to block boundary *i*+1/2 to the flow rate from block boundary *i*+1/2 to block center *i*+1 and then comparing the result with the flow rate from *i* to *i*+1 (**Fig. 2.3**), one can see that the averaged equation for permeability term in east direction will look like this[3]:

$$k_{i+1/2} = \frac{2k_i k_{i+1}}{k_i + k_{i+1}} \quad \text{..... (2.30)}$$

But for relative permeabilities upstream weighting is applied which is a consequence of the hyperbolic nature of the problem. *Raithby* showed that the upstream weighting leads to an accurate solution. The upstream weighting is defined as follows[3, 17]:

$k_{rg} = k_{rg}(S_{gi})$ if flow is from *i* to *i*+1.

And $k_{rg} = k_{rg}(S_{gi+1})$ if flow is from *i*+1 to *i*.

This method of weighting takes effect when interpolating for gas and water relative permeabilities.

The pressure dependent properties, viscosities and formation volume factors are assumed to be arithmetically averaged since these properties are not variable in our case. A simple arithmetic average is also used for porosity.

2.4 Material Balance Equation

Our material balance equation now has the following form:

$$\left(\frac{k_r}{B\mu}\right)_{gW} T_W\left(p_{i\text{-}1} - p_i\right) + \left(\frac{k_r}{B\mu}\right)_{gE} T_E\left(p_{i+1} - p_i\right) + \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.31)$$

$$\left(\frac{k_r}{B\mu}\right)_{gN} T_N\left(p_{i+1} - p_i\right) + \left(\frac{k_r}{B\mu}\right)_{gS} T_S\left(p_{i+1} - p_i\right) +$$

$$\left(\frac{k_r}{B\mu}\right)_{gT} T_T\left(p_{i+1} - p_i\right) + \left(\frac{k_r}{B\mu}\right)_{gB} T_B\left(p_{i+1} - p_i\right) = \frac{1}{\Delta t}\left[\left(\frac{V_p S_g}{B_g}\right)^{n+1} - \left(\frac{V_p S_g}{B_g}\right)^{n}\right] + q_g$$

We now consolidate the notation by defining the gas symmetrical flow coefficient as follows (for the east direction for instance):

$$a_{gE} = \left(\frac{k_r}{B\mu}\right)_{gE} T_E \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.32)$$

The Gas Material Balance Equation now has a simpler form. The resulting 3-D finite difference equation is:

$$\begin{aligned}
&a_{gE}(p_{i+1} - p_{ijk}) \\
&+ \; a_{gW}(p_{i-1} - p_{ijk}) \\
&+ \; a_{gN}(p_{j+1} - p_{ijk}) \\
&+ \; a_{gS}(p_{j-1} - p_{ijk}) \\
&+ \; a_{gB}(p_{k+1} - p_{ijk}) \\
&+ \; a_{gT}(p_{k-1} - p_{ijk}) = \frac{1}{\Delta t}\left[\left(\frac{V_p S_g}{B_g}\right)^{n+1} - \left(\frac{V_p S_g}{B_g}\right)^{n}\right] + q_g \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.33)
\end{aligned}$$

The equation can be simplified further by defining a general difference operator as follows:

$$\Delta a_g \Delta p = a_{gE}(p_{i+1} - p_{ijk}) + a_{gW}(p_{i-1} - p_{ijk}) + a_{gN}(p_{j+1} - p_{ijk}) + a_{gS}(p_{j-1} - p_{ijk}) + a_{gB}(p_{k+1} - p_{ijk}) + a_{gT}(p_{k-1} - p_{ijk}) \quad \text{.....(2.34)}$$

The general Gas Material Balance Equation can then be written as:

$$\Delta a_g \Delta p = \frac{1}{\Delta t}\left[\left(\frac{V_p S_g}{B_g}\right)^{n+1} - \left(\frac{V_p S_g}{B_g}\right)^{n}\right] + q_g \quad \text{..... (2.35)}$$

The general Water Material Balance Equation may also similarly be derived as:

$$\Delta a_w \Delta p = \frac{1}{\Delta t}\left[\left(\frac{V_p S_w}{B_w}\right)^{n+1} - \left(\frac{V_p S_w}{B_w}\right)^{n}\right] + q_w \quad \text{..... (2.36)}$$

2.5 IMPES Formulation

The choice of the method for solving the flow equations in a reservoir simulator will control the ease of use, accuracy, and to some degree the cost of the simulator. Therefore choosing the right method has to be done with extensive insight.

Several options exist for picking the dependent variables in multi-phase problems. In two-phase problems, the most common option is to solve for one phase pressure (p_g here in this case) and two saturations. Different studies of formulations lead us to the **Im**plicit **P**ressure, **E**xplicit **S**aturation (IMPES) procedure which involves solving first implicitly (as required for stability) for the gas pressure at each point and then solving explicitly for the saturations. IMPES is the most commonly used sequential approach among other manipulation techniques. In sequential methods the equation are

manipulated to separate the solution of the pressure equation from that of the saturation equation. Its appeal is a result of greatly reduced computing requirements, because it avoids the simultaneous implicit solution for several unknowns at each gridpint.[12, 18, 19] So IMPES method is chosen to derive the final flow equations that are to be discretized for coding in this research.

For the flow of a gas in a dry gas reservoir (in the absence of any condensate or oil), fluid compression and viscous forces control fluid movement. So gravity and capillary forces are not pertinent and besides capillary pressure may not be applied in this case since there is no oil or condensate.[20, 21] Since there is going to be no gravity terms or p_c in the equations, IMPES approach will have less stability limitations and can definitely be used more efficiently and this another motivation that makes us feel even more confident about using IMPES for this code.

For our case, the following main assumption for IMPES method will be taken into account.

1. 2 phase model (gas and water)
2. A plus and a minus sign refer to production and injection cases respectively.
3. No gravity terms
4. No water influx from an aquifer

The steps for the IMPES method are:

Step 1. Calculate coefficients to the pressure equation

Step 2. Matrix solution of the pressure equation for all p^{n+1}

Step 3. Explicit (point-by-point) solution of S_g^{n+1}, S_w^{n+1}

In this procedure, the saturations are eliminated by adding the individual phase material balance equations. The resultant equation has only one unknown, a phase pressure which is obtained by simultaneous solution of a set of equations. Then saturations are determined explicitly by solving material balance equations.

For the 3-d system, rearranging the two finite difference equations we came up with yields these saturation equations:

$$S_g^{n+1} = \frac{B_g^{n+1}}{V_p^{n+1}}\left[\left(\frac{V_p S_g}{B_g}\right)^n + \Delta t\left(\Delta a_g \Delta p^{n+1} - q_g\right)\right] \quad (2.37)$$

$$S_w^{n+1} = \frac{B_w^{n+1}}{V_p^{n+1}}\left[\left(\frac{V_p S_w}{B_w}\right)^n + \Delta t\left(\Delta a_w \Delta p^{n+1} - q_w\right)\right] \quad (2.38)$$

Total saturation must be equal to unity,

$$S_w^{n+1} + S_g^{n+1} = 1 \quad (2.39)$$

Eliminating unknown saturation terms using the recent equation, we'll have:

$$V_p^{n+1} = B_w^{n+1}\left[\left(\frac{V_p S_w}{B_w}\right)^n + \Delta t\left(\Delta a_w \Delta p^{n+1} - q_w\right)\right] +$$

$$B_g^{n+1}\left[\left(\frac{V_p S_g}{B_g}\right)^n + \Delta t\left(\Delta a_g \Delta p^{n+1} - q_g\right)\right] \quad (2.40)$$

Expanding this equation, we obtain

$$V_p^{n+1} = B_w^{n+1}\left[\Delta a_w \Delta p^{n+1} - q_w\right]\Delta t + \frac{B_w^{n+1}}{B_w^n}\left(V_p S_w\right)^n +$$

$$B_g^{n+1}\left[\Delta a_g \Delta p^{n+1} - q_g\right]\Delta t + \frac{B_g^{n+1}}{B_g^n}\left(V_p S_g\right)^n \quad (2.41)$$

Gas and water compressibilities are respectively:

$$c_g = -\frac{1}{B_g^n}\frac{\Delta B_g}{\Delta p} \quad (2.42)$$

$$c_w = -\frac{1}{B_w^n}\frac{\Delta B_w}{\Delta p} \quad \text{.....(2.43)}$$

Rearranging these equations yields

$$\frac{B_g^{n+1}}{B_g^n} = 1 - c_g(p^{n+1} - p^n) \quad \text{..... (2.44)}$$

$$\frac{B_w^{n+1}}{B_w^n} = 1 - c_w(p^{n+1} - p^n) \quad \text{..... (2.45)}$$

Formation compressibility is defined as:

$$c_f = \frac{1}{V_p^n}\frac{\Delta V_p}{\Delta p} \quad \text{..... (2.46)}$$

$$\rightarrow \quad \frac{V_p^{n+1}}{V_p^n} = 1 + c_f(p^{n+1} - p^n) \quad \text{..... (2.47)}$$

By substituting the compressibility equations into the pore volume equation we obtain:

$$B_w^{n+1}\left[\Delta a_w \Delta p^{n+1} - q_w\right] + B_g^{n+1}\left[\Delta a_g \Delta p^{n+1} - q_g\right] =$$

$$\frac{1}{\Delta t}\left\{V_p^{n+1} - \left[1 - c_w\left(p^{n+1} - p^n\right)\right]\left(V_p S_w\right)^n - \left[1 - c_g\left(p^{n+1} - p^n\right)\right]\left(V_p S_g\right)^n\right\} \quad \text{..... (2.48)}$$

By Simplifying Right Hand Side of this equation and substituting it into previous equation, we obtain:

$$\text{RHS} = \frac{V_p^n}{\Delta t}\left\{1 - (S_w^n + S_g^n) + (c_f + c_w S_w^n + c_g S_g^n)(p^{n+1} - p^n)\right\} \quad \text{..... (2.49)}$$

Total compressibility is defined as $c_t = c_f + c_w S_w^n + c_g S_g^n$ ..... (2.50)

So, we'll have:

$$B_w^{n+1}\left[\Delta a_w \Delta p^{n+1} - q_w\right] + B_g^{n+1}\left[\Delta a_g \Delta p^{n+1} - q_g\right] = \frac{V_p^n c_t}{\Delta t}\left(p^{n+1} - p^n\right) \quad \text{..... (2.51)}$$

If the total rate is defined as $q_t = B_w^{n+1} q_w + B_g^{n+1} q_g$, then recent relationship can be rearranged to obtain the final form which is as follows:

$$B_w^{n+1} \Delta a_w \Delta p^{n+1} + B_g^{n+1} \Delta a_g \Delta p^{n+1} = \frac{V_p{}^n c_t}{\Delta t}\left(p^{n+1} - p^n\right) \pm B_w{}^{n+1} q_w \pm B_g{}^{n+1} q_g \text{ ..... (2.52)}$$

The elements of ***A*** matrix and ***B*** matrix that are to be discretized in two separate subroutines in the code can be shown as:

$$a_W = a_{wW} B_w{}^{n+1} + a_{gW} B_g{}^{n+1} \text{ ..... (2.53)}$$

$$a_E = a_{wE} B_w{}^{n+1} + a_{gE} B_g{}^{n+1} \text{ ..... (2.54)}$$

$$a_S = a_{wS} B_w{}^{n+1} + a_{gS} B_g{}^{n+1} \text{ ..... (2.55)}$$

$$a_N = a_{wN} B_w{}^{n+1} + a_{gN} B_g{}^{n+1} \text{ ..... (2.56)}$$

$$a_T = a_{wT} B_w{}^{n+1} + a_{gT} B_g{}^{n+1} \text{ ..... (2.57)}$$

$$a_B = a_{wB} B_w{}^{n+1} + a_{gB} B_g{}^{n+1} \text{ ..... (2.58)}$$

$$a_C = -(a_{wW} + a_{wE} + a_{wS} + a_{wN} + a_{wT} + a_{wB}) B_w{}^{n+1} - (a_{gW} + a_{gE} + a_{gS} + a_{gN} + a_{gT} + a_{gB}) B_g{}^{n+1} - \frac{V_p{}^n C_t}{\Delta t} \text{ ..... (2.59)}$$

$$b = -\frac{V_p{}^n C_t}{\Delta t} P_i^n \pm q_w B_w{}^{n+1} \pm q_g B_g{}^{n+1} \text{ ..... (2.60)}$$

Pore volume and Chord slope relationships that are used in above derivations are respectively:

$$V_p = \phi \Delta x \Delta y \Delta z \quad \text{(2.61)}$$

$$V_p^{\,n+1} = V_p^{\,n}\left[1 + C_f (p^{n+1} - p^n)\right] \quad \text{(2.62)}$$

$$B_w^{\,n+1} = B_w\left[1 + C_w (p^{n+1} - p^n)\right] \quad \text{(2.63)}$$

$$B_g^{\,n+1} = B_g\left[1 + C_g (p^{n+1} - p^n)\right] \quad \text{(2.64)}$$

CHAPTER III

PROGRAM CHARACTERISTICS AND PROPERTIES

In this chapter the main attributes of the developed simulator as well as the input and output units are discussed. The simulator is a VBA code which is coupled to the pertinent Excel file. It evaluates/forecasts the declining regime of volumetric dry gas reservoirs for two-phase (gas and water), 3-D models over the productive life of them.

3.1 VBA Code Algorithm

The 3-D, two-phase code that is developed is an IMPES manipulation of gas and water flow equations. The code is a convoluted structure of different subroutines that are all embodied by a main subroutine called *Main* that controls the order of the run of the subroutines and loops them over each timestep until the last timestep is reached. Each of these subroutines does a specific task when it is reached in the order it is placed within the main loop or when it is called by another subroutine. Some subroutines may be called more than once. Some basic tasks such as interpolations and averagings are accomplished in functions instead of subroutines. **Fig. 3.1** exhibits the flowchart of the code. This diagram is the basic algorithm of the code and does not represent all of the subroutines. We will go through the code algorithm within one single timestep following above flowchart.

Read Data: Once the program starts running the first subroutine in the time loop runs which takes care of reading reservoir, wells, PVT, relative permeabilities and all data required for the program to run from the input file. This subroutine is written in such a way that it is capable of accepting either uniform or irregular grids. Gridblocks can be of different dimensions in either of *x*, *y* or *z* directions or any combination of them. The use of irregular grid spacing is essential in models.

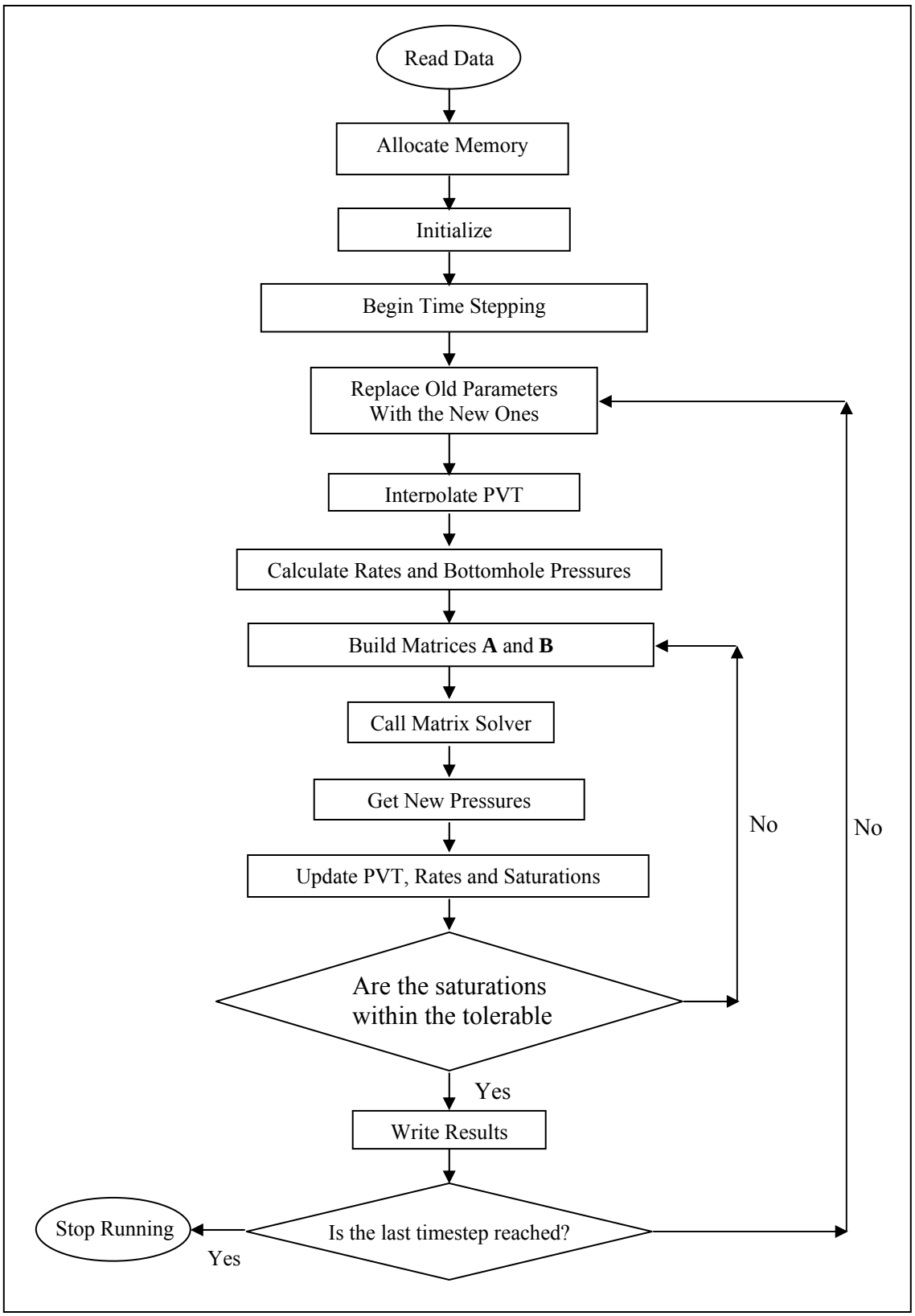


Fig. 3.1- Flowchart of the 3-D, 2-phase code

In many practical problems it is necessary to refine the grid in certain parts of the reservoir in order to obtain desired accuracy. For example, local refinement is often necessary around the wellbore. On the other hand it is often possible to use coarser grid over area where pressure and saturation change slowly. Irregular grid is also advantageous in cross-sectional and 3-D simulation of stratified reservoirs where the vertical gridblocks are chosen according to reservoir stratification. In practice we always want to keep the grid as coarse as possible (especially in 3-D simulations).[3]

In order for the code to be able to handle different gridblock sizes in a particular direction, the method of grid construction is *point-distributed* (as shown in **Fig. 3.2**) in

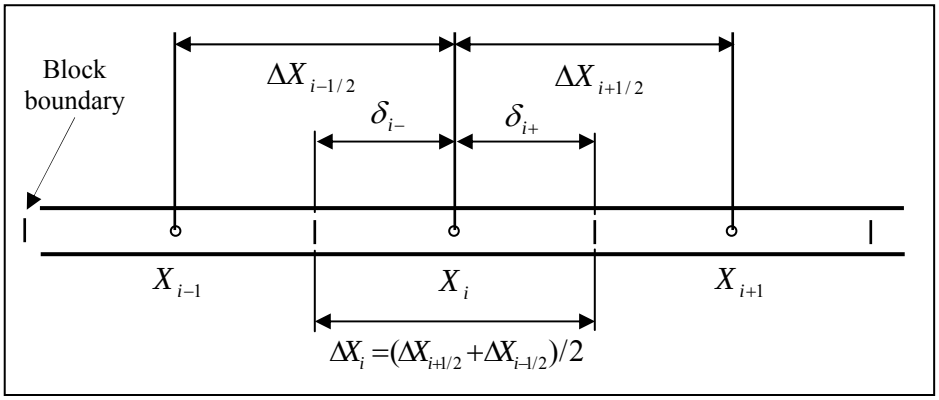


Fig. 3.2- *Point-distributed* system of gridding accounts for irregular grids (from Aziz and Settari[3])

which the grid points are selected first and the block boundaries are placed half-way between the grid points:

$$\delta_{i+} = (\Delta X_{i+1/2})/2 \quad \text{..... (3.1)}$$

$$\delta_{i-} = (\Delta X_{i-1/2})/2 \quad \text{..... (3.2)}$$

Unlike many commercial simulators, the code can handle models of any sizes. All the user has to do to define the size of a problem is to enter the number of gridblocks in each direction if the model is supposed to be uniform in that particular direction. But if the model has to be non-uniform in a particular direction the user must enter the sizes of all gridblocks in that direction.

Porosity and permeabilities can also be configured differently in different directions. In most cases permeability in *z* direction is considerably less than what it is in other directions.

The **PVT** table used for interpolation PVT properties of a given reservoir's gas and water can handle pressures starting from standard conditions up to 4,000 psi and the units for this table are tabulated in **Table 3.1**.

Table 3.1- Units for the PVT properties used in the input file

Pressure	B_w	μ_w	C_w	B_g	μ_g	C_g
psi	*rcf/scf*	*cp*	*1/psi*	*rcf/scf*	*cp*	*1/psi*

Allocate Memory: The next thing after reading the input set of data is allocating required memory for each variable.

3.1.1 Initial Conditions

To complete the mathematical description of a reservoir (following Eq. 2.49 in chapter II), it is necessary to specify initial conditions.[12] **Initialize** subroutine is the place for

basing the initial conditions to begin the timestep sequence. For the initial conditions, n=0, a value is specified for pressure, gas saturation and water saturation and then these parameters have to be initialized in every node by assigning the values to three-dimensional arrays through a $3 \times 3 \times 3$ spatial loop. Pore volumes are calculated for each gridblock and the summation of all of them is stored as the initial reservoir pore volume. Formation volume factors, viscosities and compressibility values are located by interpolation. Total compressibility is calculated using the existing saturations and phase compressibilities according to the following formula from the previous chapter:

$$c_t = c_f + c_w S_w^n + c_g S_g^n \quad \text{..... (2.47)}$$

Initialize is also where *total fluids in place* are calculated. Relative permeability to each phase is calculated by interpolation with recent saturations.

Begin Time Stepping: Now, the calculations are ready to get started with the time loop. The time loop is a *do loop* which repeats the following steps until it reaches the last timestep.

Replace Old Parameters With the New Ones: At this stage the new pressures calculated at the end of the previous timestep are designated as old values at the beginning of the new timestep. The same procedure is also applied for pore volumes, saturations and formation volume factors. The existing pressures are used to update viscosity, compressibility, total compressibility and relative permeabilites. Here viscosity, compressibility and relative permeabilites are interpolated. Using the

interpolated values of relative permeability, FVF and viscosity, the mobilities are evaluated. Also transmissibilities need to be evaluated for all directions (west, east, south, north, top and bottom) at this same stage, because they are going to be used in the *A* matrix.

3.1.2 Well Rates and Pressures

The well equations use pressures at the center of the gridblocks. These pressures represent material balance average pressures in the gridblock. However, if a well is located in the center of a gridblock, the gridblock pressure, $p_{i,j}$ is not the wellbore flowing pressure, p_{wf}. These equations compute the gas flow from gridblock to gridblock. So if a well is located in a cell, we need additional equations to relate the well performance to the cell variables. Steady state flow occurs within a cell and uses *Peaceman*'s equations[15]:

$$q_g = J_{model}\left(\frac{k_r}{B\mu}\right)_g^n \left(p_{i,j,k}^{n+1} - p_{wf}\right) \quad \ldots\ldots (3.3)$$

$$q_w = J_{model}\left(\frac{k_r}{B\mu}\right)_w^n \left(p_{i,j,k}^{n+1} - p_{wf}\right) \quad \ldots\ldots (3.4)$$

The fluid and rock properties are the same as for the cells. We now have 2 equations with 3 unknowns: q_w, q_g and p_{wf}. This means that the user must specify one of these unknowns which is going to be the condition under which the well will produce. For example, if the user specifies q_g, then q_w and p_{wf} are calculated. Similarly if we specify p_{wf}, then we can calculate q_w and q_g from the above equations.

In Peaceman's equations J_{model} is called "productivity index" or "well index" and is defined as follows:

$$J_{model} = \frac{2\pi\,(0.00633)\,kh}{\ln r_o / r_w + s} \quad \ldots\ldots (3.5)$$

Where r_o is calculated using the following equations:

1) When $\Delta x = \Delta y$, $k_x = k_y$,

$$r_o = 0.2\Delta X \quad \text{.....(3.6)}$$

2) Otherwise,

$$r_o = \frac{0.28\left[\sqrt{k_y/k_x}\,(\Delta x)^2 + \sqrt{k_x/k_y}\,(\Delta y)^2\right]^{1/2}}{(k_y/k_x)^{1/4} + (k_x/k_y)^{1/4}} \quad \text{.....(3.7)}$$

There are essentially two methods for representing a well in a simulator: by rate constraint or by pressure constraint.[5, 22] Both constraint methods are contained in the code and are summarized below.

Well BHP and Rates for Constant Rate Constraint: In this representation rates may be specified for injectors or producers. If the rate of any one phase is specified then the rate of the other phases can be calculated with obtaining the bottomhole pressure first as follows:

$$p_{wf} = p_i - \frac{q_\beta}{J\lambda_\beta} \quad \text{.....(3.8)}$$

$$q_\alpha = J\lambda_\alpha (p_i - p_{wf}) \quad \text{.....(3.9)}$$

where β is the phase whose rate is known and α is the phase whose rate is unknown.

Well Rates for Constant Bottomhole Pressure Constraint: If the bottomhole pressure and well productivity index are known then the rate of any phase can be

obtained as in Eq. 3.10. This calculation is done in a different subroutine from the subroutine for the previous constraint:

$$q_\alpha = J\lambda_\alpha (p_i - p_{wf}) \quad \text{..... (3.10)}$$

3.1.3 Boundary Conditions

For our 3-D case, at the left and right boundaries, we need to specify equations other than the discretized form of the diffusivity equation derived in chapter II, *i.e.* Eq. 2.16:

$$\frac{p_{i-1}^{n+1} - 2\,p_i^{n+1} + p_{i+1}^{n+1}}{(\Delta x)^2} = \frac{\phi\mu c}{k}\frac{p_i^{n+1} - p_i^n}{\Delta t} \quad \text{..... (2.16)}$$

The usual boundary condition is called a "no-flow" boundary condition or the *Neumann* condition.[8]In other words, no fluid flows across the outer boundaries. A frequent assumption in reservoir simulation is that the reservoir lies within some closed boundary across which there is no flow, and that fluid injection and production takes place at wells located at points within the interior of the reservoir.[12, 23] This condition quite fits the main assumption of developing this simulator which is handling **volumetric** dry gas reservoirs. Because in this type of reservoirs there is no flow or pressure communication between the reservoir and the adjacent media.

We should note that at each well, either the pressure or the flow rate for a phase is specified and this specification is, in fact, the most important part of the boundary conditions[12] which was detailed in the previous section. The boundary condition relations that apply in the discretized form of the final flow equation will come in the following section.

3.1.4 Matrices A and B

We now can state all the equations that are to be solved simultaneously for each timestep.

Recalling the discretized form of the diffusivity equation we derived in chapter II, we have:

$$\frac{p_{i-1}^{n+1} - 2\,p_i^{n+1} + p_{i+1}^{n+1}}{(\Delta x)^2} = \frac{\phi\mu c}{k}\frac{p_i^{n+1} - p_i^{n}}{\Delta t} \quad \text{..... (2.16)}$$

For our 3-D system, according to the directions defined in **Fig. 3.3**, this equation can be expanded as follows:

$$a_w(w, g) \,.\, (P_{i-1,j,k} - P_{i,j,k}) + a_e(w, g) \,.\, (P_{i+1,j,k} - P_{i,j,k}) + a_s(w, g) \,.\, (P_{i,j-1,k} - P_{j,j,k}) +$$

$$a_n(w, g) \,.\, (P_{i,j+1,k} - P_{j,j,k}) + a_b(w, g) \,.\, (P_{i,j,k-1} - P_{j,j,k}) + a_t(w, g) \,.\, (P_{i,j,k+1} - P_{j,j,k}) =$$

$$V_p^{\,n}\; C_t(P_{i,j,k}^{\,n+1} - P_{i,j,k}^{\,n})/\Delta t \quad \text{..... (3.11)}$$

$\Rightarrow$

$$a_w(w, g)\, P_{i-1,j,k} - a_w(w, g)\, P_{i,j,k} + a_e(w, g)\, P_{i+1,j,k} - a_e(w, g)\, P_{i,j,k} + a_s(w, g) P_{i,j-1,k} -$$

$$a_s(w, g)\, P_{i,j,k} + a_n(w, g)\, P_{i,j+1,k} - a_n(w, g)\, P_{i,j,k} + a_b(w, g)\, P_{i,j,k-1} - a_b(w,g)\, P_{i,j,k} +$$

$$a_t(w, g)\, P_{i,j,k+1} - a_t(w, g)\, P_{i,j,k} - V_p^{\,n}\, C_t P_{i,j,k}^{\,n+1}/\Delta t = V_p^{\,n}\, C_t\, P_{i,j,k}^{\,n}/\Delta t \quad \text{..... (3.12)}$$

From this point forward we change the notation from $a_w(w, g)$ simply to a_W and $a_e(w, g)$ to a_E and etc. Now we define a_C for central gridblocks as follows:

$$a_C = -(a_W + a_E + a_S + a_N + a_T + a_B) - \frac{V_p^{\,n} C_t}{\Delta t} \quad \text{..... (3.13)}$$

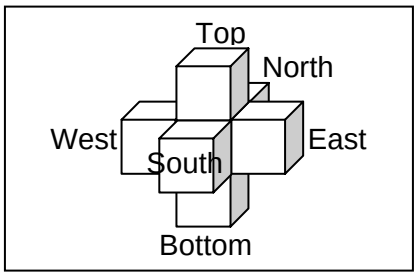


Fig. 3.3- Spatial definition of directions for expanding the diffusivity equation

Therefore Eq. 3.12 is simplified to the following format:

$$a_W \, P_{i-1,j,k} + a_E \, P_{i+1,j,k} + a_S \, P_{i,j-1,k} + a_N \, P_{i,j+1,k} + a_B \, P_{i,j,k-1} + a_T \, P_{i,j,k+1} + a_C \, P_{i,j,k} =$$

$$V_p{}^n \, C_t \, P_{i,j,k}{}^n / \Delta t \quad \text{..... (3.14)}$$

Recent equation makes up a system of simultaneous linear equations with respect to the unknown pressures. The number of unknowns and equations is defined by the user is the number of gridblocks in a given model. For ease of the equations, let us define α as the following:

$$\beta = V_p{}^n C_t / \Delta t \quad \text{..... (3.15)}$$

For example if there are 14 gridblocks in a model, the equations will look like the following:

$$a_C P_1^{n+1} + a_E P_2^{n+1} + a_N P_4^{n+1} = -\beta P_1^n \quad \text{..... (3.16)}$$

$$a_W P_1^{n+1} + a_C P_2^{n+1} + a_E P_3^{n+1} + a_N P_5^{n+1} + a_T P_8^{n+1} = -\beta P_2^n \quad \text{.....(3.17)}$$

$$a_W P_2^{n+1} + a_C P_3^{n+1} + a_E P_4^{n+1} + a_N P_6^{n+1} + a_T P_9^{n+1} = -\beta P_3^{n} \quad \text{..... (3.18)}$$

. .

. .

. .

$$a_B P_7^{n+1} + a_S P_{10}^{n+!} + a_W P_{12}^{n+1} + a_C P_{13}^{n+1} + a_E P_{14}^{n+1} = -\beta P_{13}^{n} \quad \text{..... (3.19)}$$

$$a_B P_8^{n+1} + a_S P_{11}^{n+1} + a_W P_{13}^{n+1} + a_C P_{14}^{n+1} = -\beta P_{14}^{n} \quad \text{..... (3.20)}$$

Therefore, for this 14-gridblock model, we have 14 equations and 14 unknowns. The first and last equations in this set are the governing equations for boundary conditions. This set of equations can be represented by a *matrix equation*, which can simply be written as:

$$A\vec{p} = \vec{B} \quad \text{..... (3.21)}$$

where A is the coefficient matrix and $\vec{p}$ and $\vec{B}$ are column vectors. Therefore, the set of equations can be shown as **Fig. 3.4**. $\vec{B}$ consists of the right hand side terms of the equations which are all known. In this matrix presentation a C represents a central flow coefficient; a W represents a west flow coefficient and etc.

Now that we made up our matrix presentation of the flow equations, we need to see what each coefficient is. For our IMPES formulation all of the coefficients in the left hand side matrix are Eqs. 2.50 through 2.56 in chapter II. The values in the B matrix for the IMPES formulation for the perforated blocks are computed using Eq. 2.57 in chapter II which has the total flow rate term in addition to β here.

In the subroutines of matrices A and B in the code, required average values for formation volume factors and viscosities are calculated. The direction of flow is determined to take the upstream relative permeabilites. For the constant bottomhole pressure case, B vector will have the part of the total rate, which has the old pressure vector as a multiplier. But the one which has the new pressure vector as a multiplier will go into A matrix to the central flow coefficient.

$$
\begin{bmatrix}
C & E & \dots & N & \dots & \dots & T & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
W & C & E & \dots & N & \dots & \dots & T & \dots & \dots & \dots & \dots & \dots & \dots \\
\dots & W & C & E & \dots & N & \dots & \dots & T & \dots & \dots & \dots & \dots & \dots \\
S & \dots & W & C & E & \dots & N & \dots & \dots & T & \dots & \dots & \dots & \dots \\
\dots & S & \dots & W & C & E & \dots & N & \dots & \dots & T & \dots & \dots & \dots \\
\dots & \dots & S & \dots & W & C & E & \dots & N & \dots & \dots & T & \dots & \dots \\
B & \dots & \dots & S & \dots & W & C & E & \dots & N & \dots & \dots & T & \dots \\
\dots & B & \dots & \dots & S & \dots & W & C & E & \dots & N & \dots & \dots & T \\
\dots & \dots & B & \dots & \dots & S & \dots & W & C & E & \dots & N & \dots & \dots \\
\dots & \dots & \dots & B & \dots & \dots & S & \dots & W & C & E & \dots & N & \dots \\
\dots & \dots & \dots & \dots & B & \dots & \dots & S & \dots & W & C & E & \dots & N \\
\dots & \dots & \dots & \dots & \dots & B & \dots & \dots & S & \dots & W & C & E & \dots \\
\dots & \dots & \dots & \dots & \dots & \dots & B & \dots & \dots & S & \dots & W & C & E \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & B & \dots & \dots & S & \dots & W & C
\end{bmatrix}
\begin{Bmatrix}
P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \\ P_8 \\ P_9 \\ P_{10} \\ P_{11} \\ P_{12} \\ P_{13} \\ P_{14}
\end{Bmatrix}^{n+1}
=
\begin{Bmatrix}
-\beta P_1 \\ -\beta P_2 \\ -\beta P_3 \\ -\beta P_4 \\ -\beta P_5 \\ -\beta P_6 \\ -\beta P_7 \\ -\beta P_8 \\ -\beta P_9 \\ -\beta P_{10} \\ -\beta P_{11} \\ -\beta P_{12} \\ -\beta P_{13} \\ -\beta P_{14}
\end{Bmatrix}^{n}
$$

Fig. 3.4- The matrix of equations for a 14-gridblock sample model

3.1.5 Matrix Solver

Once matrices *A* and *B* are built, the next task the time loop does is to solve the matrices by calling the matrix solver subroutine. Solver gets the flow coefficients from the *A* matrix and the known values of right hand side from *B* vector and returns the new pressure values.

The solution of the pressure equation can either be very simple or very difficult, depending on the physical problem. Almost all 3-D problems are considered relatively difficult to difficult and the effort required to solve the pressure equation becomes very significant in relation to the rest of the reservoir simulation problem. It is not unusual for the computing cost of solving Eq. 3.21 to be as high as 80% to 90% of the total reservoir simulation cost. The rest of the reservoir simulation solution, other than the solution of equation, is relatively constant in the computation time and effort required. This means that the overall

cost of reservoir simulation is often directly dependent on the ease with which we can solve Eq. 3.21.[15]

Solution methods are either **iterative** or **direct**. The basis of an iterative method is the development of an "approximate" solution to the system of equations. The approximation is replaced systematically until the answers converge to "the correct" answer. In a direct method, as the name implies, the algorithm that is used "solves" the equations exactly and will give a correct answer in a fixed number of answers.[12]

Because of the drastic increase in computational effort as the grid size increases in 3-D problems, there exists a grid size, such that for any grid size larger than this, an iterative method would have a computational advantage over a direct method. More importantly, perhaps, is the fact that direct methods require large amounts of storage for the coefficient matrix *A.* Iterative methods, on the other hand, are particularly well suited for large, spare systems of equations.[5]

According to *Vinsome* (1976), the most commonly used procedure among iterative methods, is **Orthomin**. This method is a minimization process conceptually based on the **conjugate-gradient** numerical method and converges faster than any other iterative method and also it is insensitive to the number of equations. Another great advantage of Orthomin approach is that it is applicable to non-symmetric sparse matrices. The method is so called because it uses both orthogonalizations, and minimization to achieve a high rate of convergence.[24]

Update PVT Data, Rates and Saturations: Rate for each phase is updated using the new pressure for constant bottomhole pressure case. For this rate, update $J_{\mathrm{mod}el}$ and mobility terms are kept the same. Formation volume factors and pore volumes are updated using chord slopes and they are named as "new". Notice that these new values will still be named as "new" at the beginning of the new time step and the values, which are named as old, will be calculated by interpolation. Saturations are updated as the last stage just before the timestep-cut procedure.

3.1.6 Timestep-Cut

In order to assure the IMPES formulation will converge for whatever input data a user might enter, there is a need for a **timestep-cut** procedure.[25] Once the new values of pressures are obtained and the rates and saturations are updated the main time loop goes through a timestep-cut inner loop to evaluate whether the timestep has to be decreased or not. Timestep, however, is not the only entry that can be changed to control the convergence of the solution. There is also another entry in the input file called *ncuts* and the user has the option of establishing some value for it before a run. The default value for the *ncuts* in this code is 3. In most commercially available simulators this value is 4. *ncuts* helps the timestep-cut procedure control the number of required reductions in the timestep (if any). Furthermore, it controls how much reduction is required for the timestep size to get the fastest possible convergence with the largest timestep each time there is a need for a reduction. The circumstance of the timestep-cut procedure is exhibited in **Fig. 3.5**. In this algorithm, *counter* is a variable initially set to zero for comparing with *ncuts* in the first **if** condition (algorithm in **Fig. 3.5**). This variable is first set to zero each time before the start of the timestep-cut loop.

Check Well Constraints: Once the timestep size is fixed (if required) in the timestep-cut inner loop, the main loop makes the simulator proceed into calculating the cumulative production of gas and water. Meanwhile, for wells with constant rate constraint, if the calculated bottomhole pressure turns out to be less than the minimum allowable BHP determined in the input file, the code changes the constraint from constant rate to constant pressure and then from this point forward will go through this constraint for calculating the rates.

Write Results: At this stage all of the results for this particular timestep are calculated and stored in memory and are ready to be written in both the pertinent Excel file and in the output Notepad file. So while the simulator proceeds in running, the results are being either written or plotted in the output units.

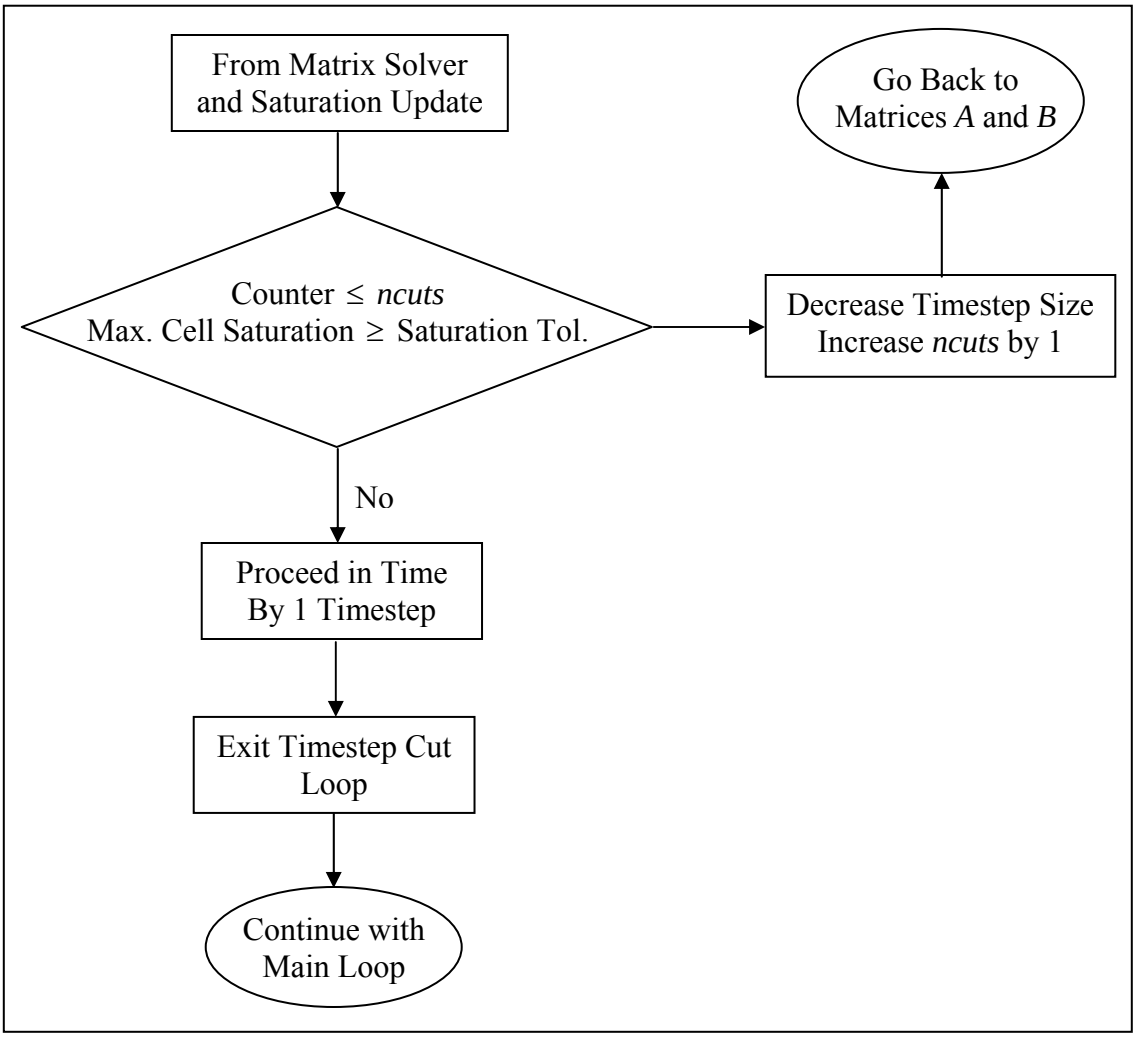


Fig. 3.5- Algorithm for the timestep-cut loop within the main time loop

Proceed with Timestepping: At this stage if the last timestep has not been reached yet, the code goes back to the beginning of the time loop where the old parameters are replaced with new ones until the required simulation time is reached.

3.2 Output Units

Once the run is complete all graphical results for different timesteps and/or wellbores are plotted in the pertinent Excel file. These graphical results include well gas rate, water rate and bottomhole pressure, average reservoir pressure, cumulative production, schematic of the grid model, reservoir cell pressures and saturations. Initial gas and water in place are also given in this Excel file. Well constraint, reservoir pressure, gas saturation, water saturation and total saturation for all timesteps are tabulated in separate worksheets and the user can make any desirable projection of them for either the analysis of reservoir performance or comparison purposes. The user can both view the existing plots and make new combination of desirable plots upon their need.

The code also generates a Notepad file as another output unit. This file contains all numeric results such as cell pressures, gas and water saturations, for all timesteps and all layers, fluids in place, cumulative production and also well entries from the input file.

CHAPTER IV

VALIDATION AND ANALYSIS OF RESULTS

The goal of any numerical-model study is the analysis and/or prediction of reservoir performance in more detail and with more accuracy than is possible with simple techniques such as extrapolation.[12] Therefore, in this chapter we will go through the results of two distinct simulation cases done by both the 3-D, 2-phase code and CMG in order to make comparisons between the two simulators. Comparing the runs of a newly-developed simulator with a commercially available simulator is usually the best way to confirm the validity of estimates. For validation of this code, CMG software is used which is a well-recognized simulator package in the oil and gas industry. In general having matches of fluid movement parameters including gas rates, water rates, cumulative productions and water/gas ratios (WGR's), between the two simulators is the strongest verification of the validity of techniques, formulations and assumptions concerning the newly-developed simulator. In this chapter, in addition to these parameters, the results of bottomhole and average reservoir pressures at times for both simulators are shown. The analysis of the results pertaining original fluids in place is also included in this chapter.

4.1 Simulation Schemes

In order to show the reliability of the code, two different schemes are considered to run. The first case is a production plan including three producer wells, with three different constraints. The second one is a simple injection-production plan with one injector and one producer well. Both plans have the same number of gridblocks and gridblock size. The model dimensions are $20 \times 20 \times 10$. **Fig. 4.1** shows the model configuration and also the location of wells for the first simulation scheme.

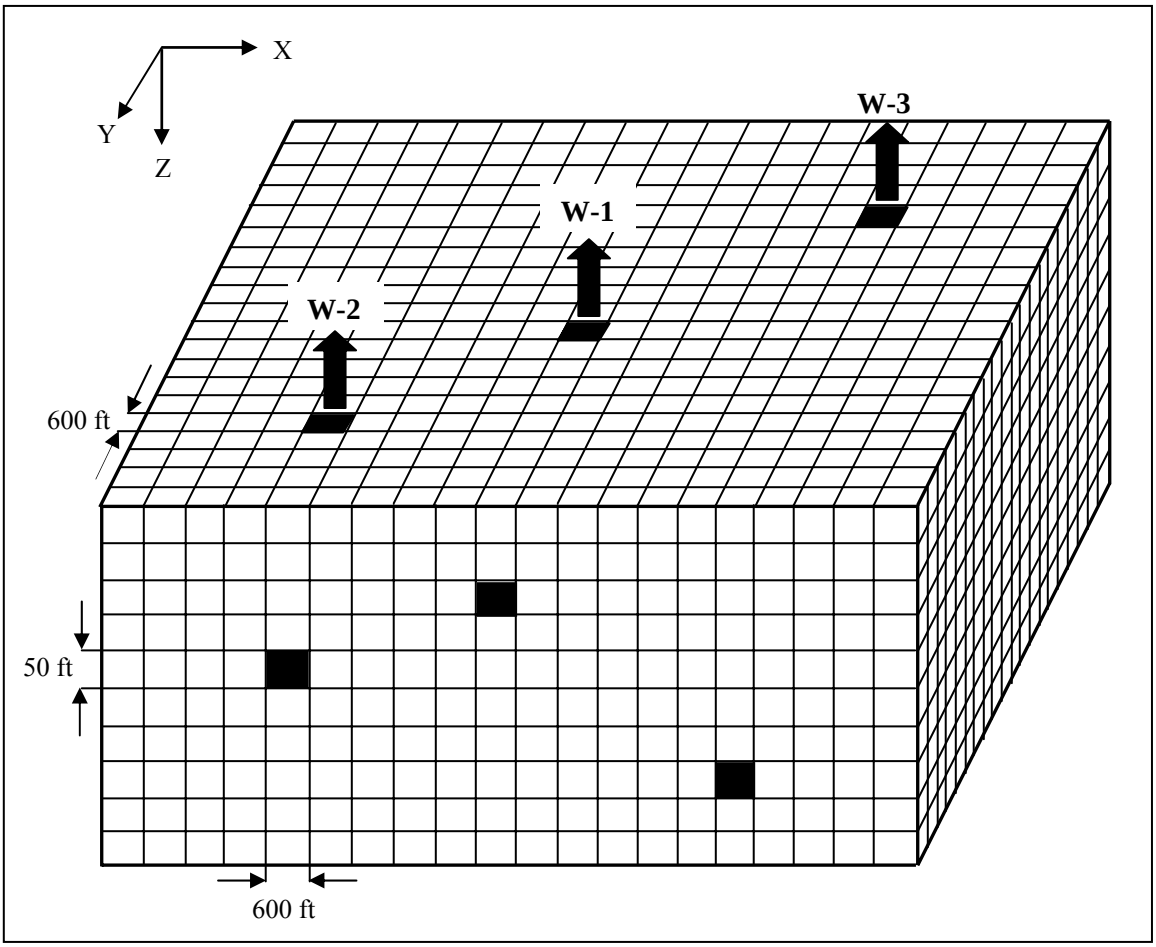


Fig. 4.1–Grid system and locations of production wells for case one

Table 4.1–Wells entries for case one

Well ID	X	Y	Z	Well Type	Constraint	Constraint Value	Min. BHP
W-1	10	10	3	Producer	Const. Gas Rate	10,000 Mscf	1,000 psia
W-2	5	5	5	Producer	Const. BHP	2,500 psia	1,000 psia
W-3	15	15	8	Producer	Const. Water Rate	40 STB	1,000 psia

The constraints and other entries of wells for case one (three producer wells) are tabulated in **Table 4.1**. Also all reservoir and model data are tabulated in **Table 4.2**.

Table 4.2–Reservoir input data

X_D	Y_D	Gridblock thickness	ϕ	k_x	k_y	Vertical permeability	p_i	S_{wi}	T	γ_g
600 ft	600 ft	50 ft	0.25	30 md	30 md	5 md	3,000 psia	0.25	210˚F	0.7

Table 4.3 shows the relative permeability data selected for running both cases. These sets of data for water and gas are demonstrated in **Figs. 4.2** and **4.3** respectively.

Table 4.3–Relative permeability data

S_w	K_{rw}	S_g	k_{rg}
0.1	0	0	0
0.3	0.024	0.1	0
0.4	0.056	0.3	0.04
0.5	0.116	0.4	0.072
0.6	0.192	0.55	0.144
0.83	0.488	0.65	0.248
1	1	0.85	0.532
		1	1

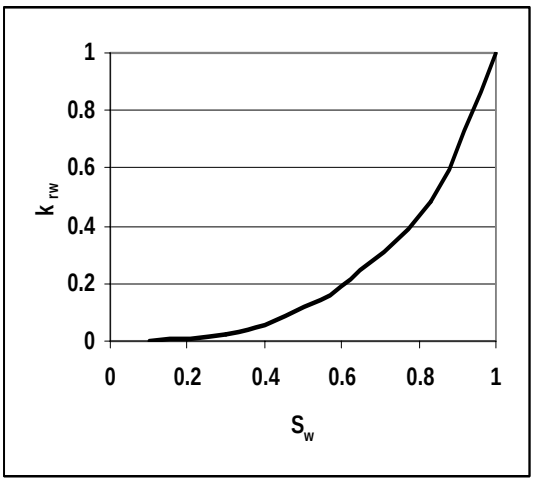


Fig. 4.2–Water relative permeability used in cases 1 and 2

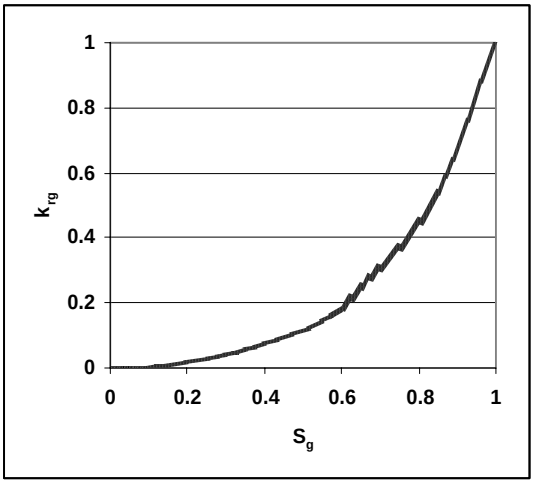


Fig. 4.3–Gas relative permeability used in cases 1 and 2

4.2 Validation of Plan 1: Three-Producer Case

Plots of different fluid movement parameters as well as, P_{wf} , $\overline{p}$ and GWR versus time

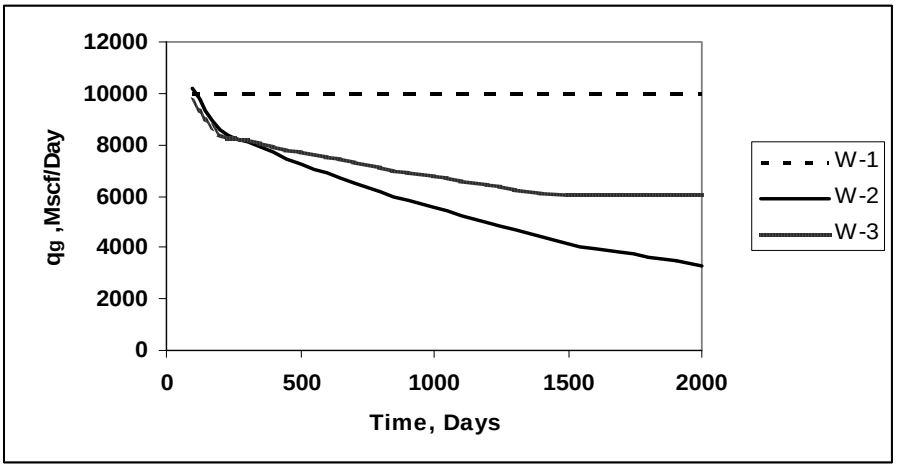


Fig. 4.4– Individual well gas rates

are made and incorporated to the same plots simulated by CMG package in order to show the validity of the results of the code.

Under the specified constraints and deliverability conditions for the three wells in plan one, the general depletion behavior of the assumed reservoir can be seen from **Fig. 4.4** for the gas rates from the code. **Fig. 4.5** is CMG's gas rate plot for well *W-2* (constant P_{wf} well) shown as a sample of plots made by CMG. For the rest of the results, comparisons are made by putting the plot generated by the code together with one generated by CMG to show the matches in more detail.

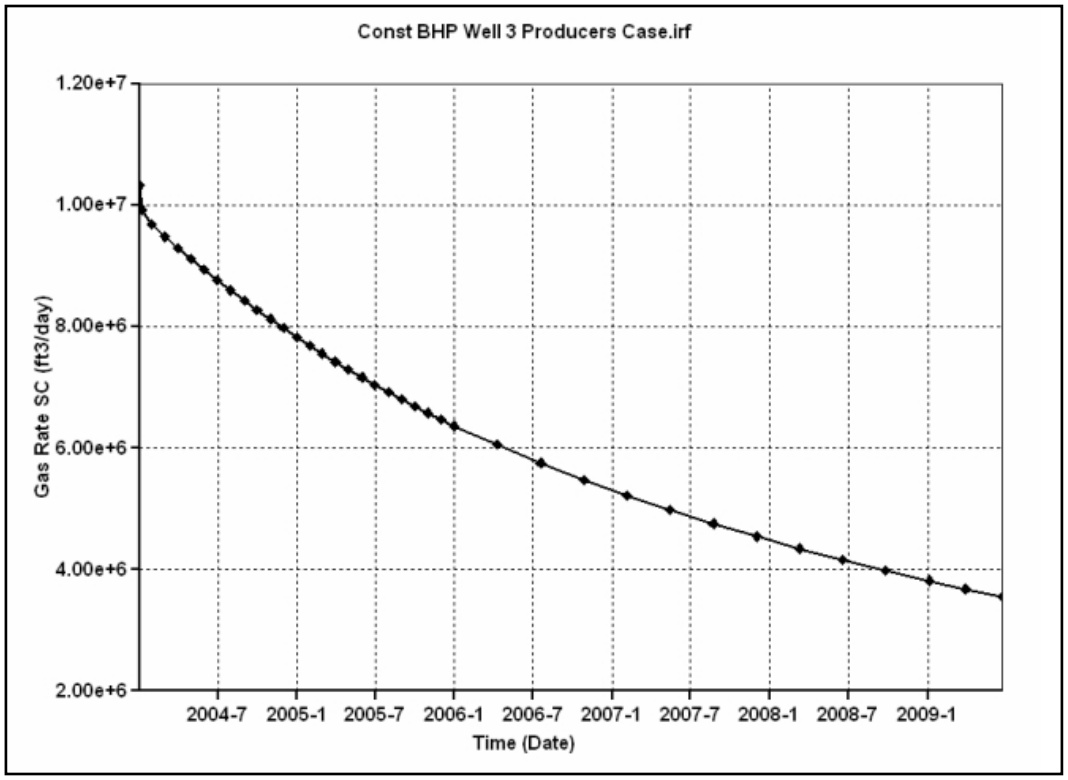


Fig. 4.5– Gas rate plot for well *W-2* generated by CMG

The first match can be observed from **Fig. 4.6** which shows the satisfactory match of gas production rate for the constant bottomhole pressure well (*W-2*), simulated by both the code and CMG.

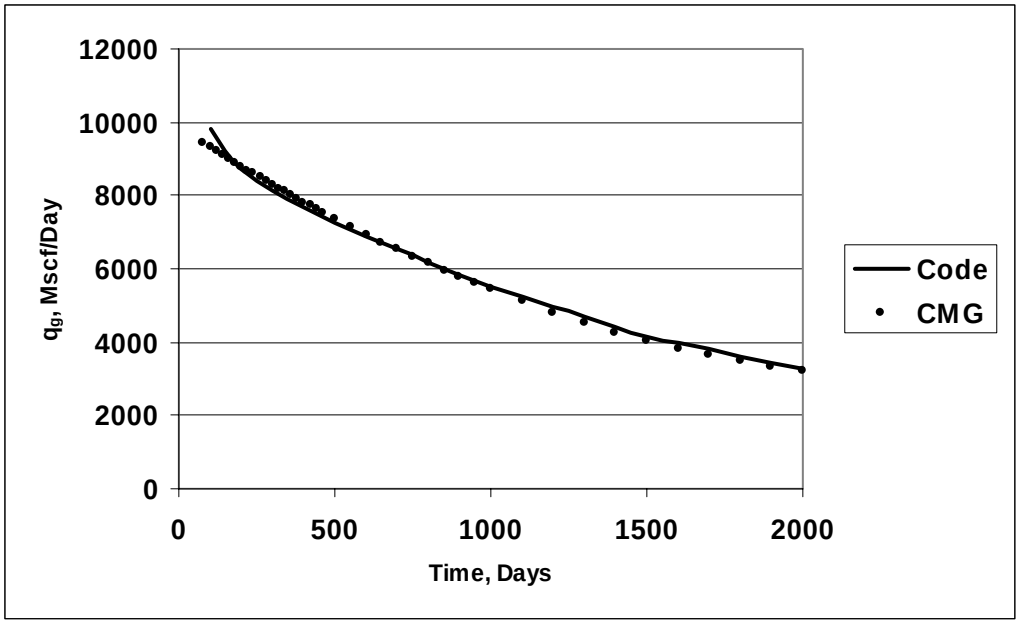


Fig. 4.6– Gas rate plots for well *W-2* generated by both the code and CMG

Fig. 4.7 shows the same sets of graphs for the well with constant water rate (*W-3*). The slight difference seen at the beginning of the two plots is probably caused by different timestep sizing at initial timesteps between the two simulators as the dotted line for CMG shows that it has taken very small timesteps at the beginning of the run to converge. The timestep size in the 3-D, 2-phase code is 100 days for all of the shown plots in this section. Since the code is equipped with the timestep-cut procedure, it can be concluded that there has been no need for a cut in timestep size in this particular run, because if there has, the code would have taken smaller timesteps.

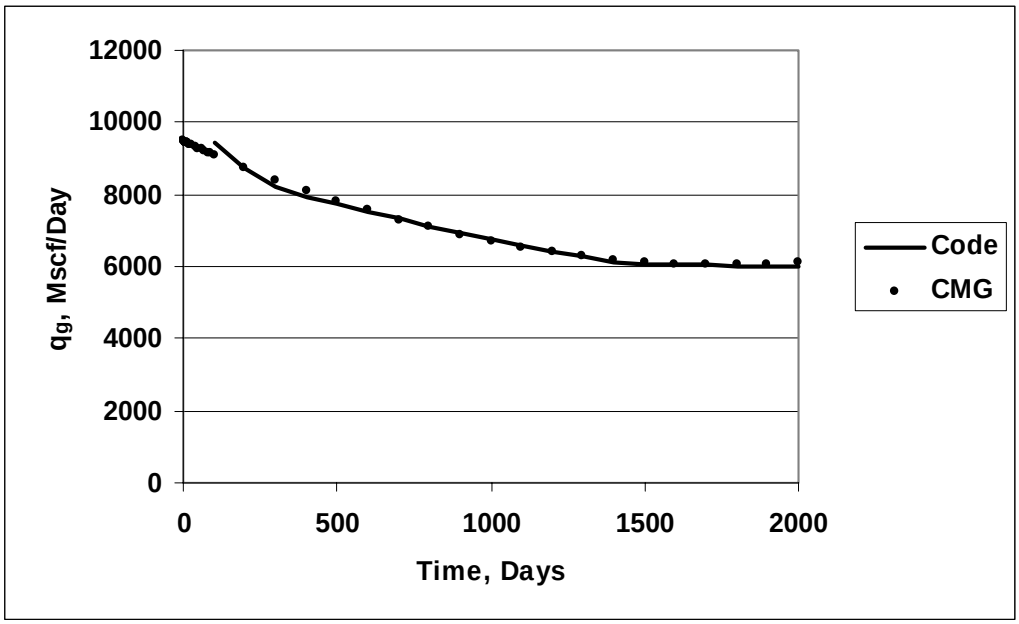


Fig. 4.7– Gas rate plots for well *W-3*

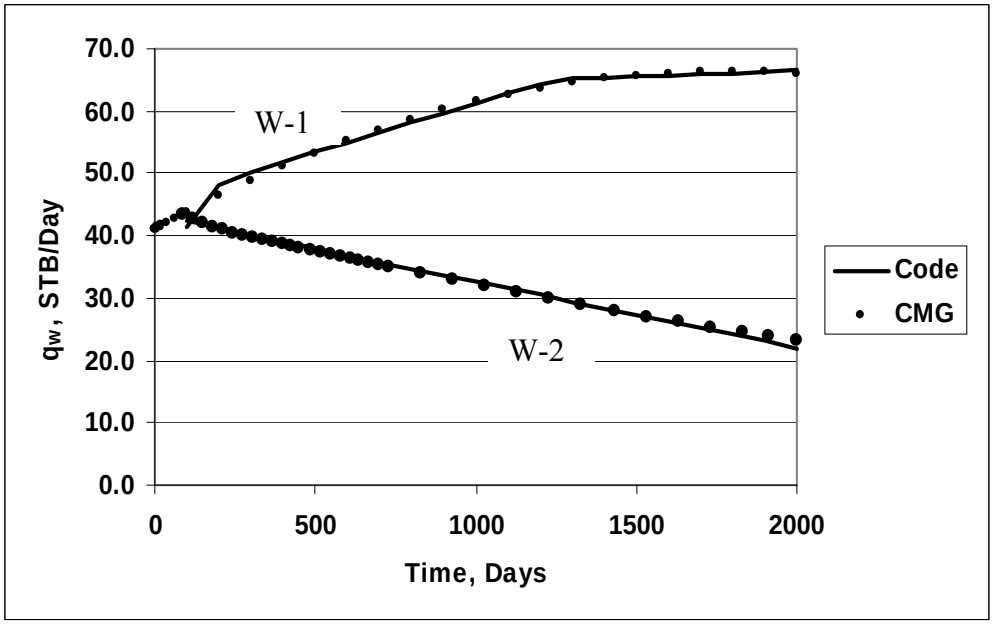


Fig. 4.8–Water rate plots for wells *W-1* and *W-2*

Fig. 4.8 demonstrates the water rates for the two wells with constant P_{wf} and q_g. Well *W-2* shows an almost linear decline in water production rate while the well with constant gas rate (*W-1*) increases water production from 41 STB/Day to 66 STB/Day within the simulated life of the reservoir. Since the other well produces under constant water constraint, the water production well is not included in this graph.

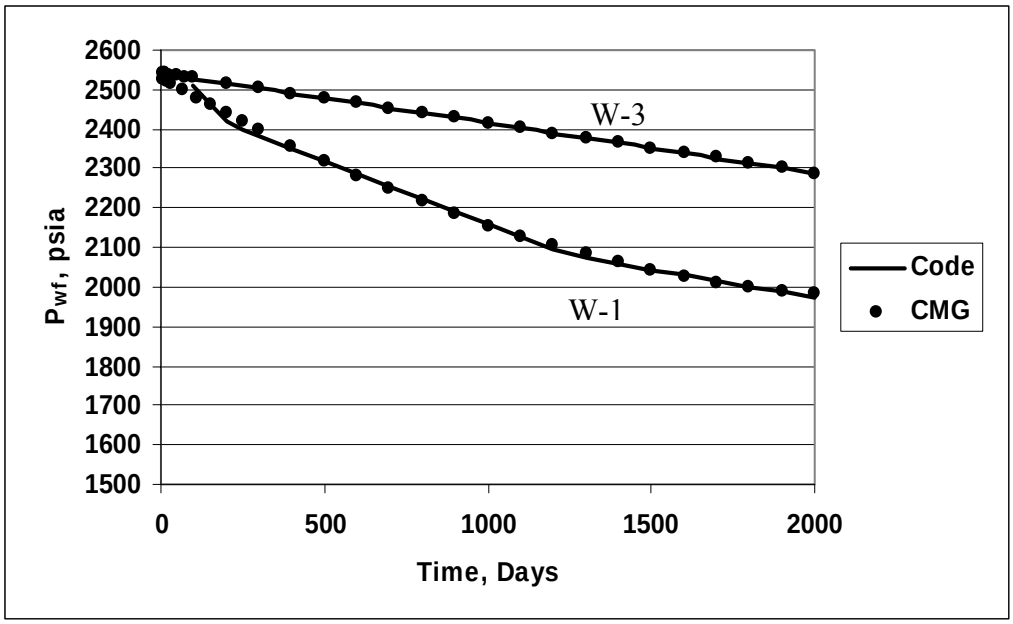


Fig. 4.9– Bottomhole pressure plots for wells *W-1* and *W-3*

Fig. 4.9 shows the P_{wf} for wells *W-1* and *W-3*. Since the first well has constantly produced more gas than the third well (10,000 Mscf/Day), the bottomhole pressure for this well has depleted faster than well *W-3*. **Fig. 4.10** illustrates the cumulative gas and water produced for the well with constant BHP constraint over the life of the reservoir. Cumulative gas and water amounts produced in the whole reservoir are shown in **Figs. 4.11** and **4.12** respectively.

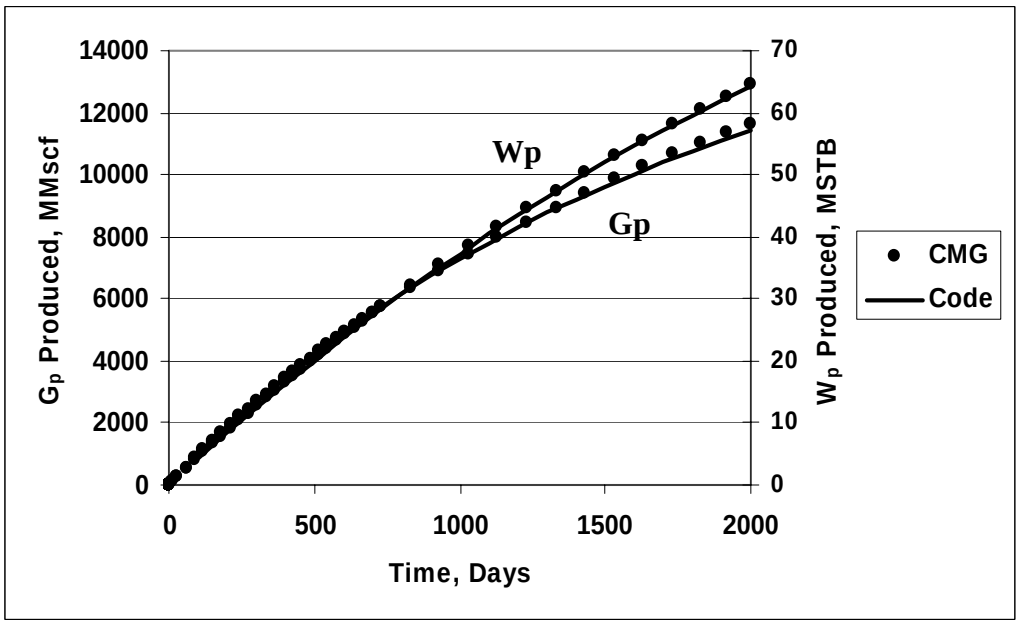


Fig. 4.10– G_p and W_p plots for well *W-2*

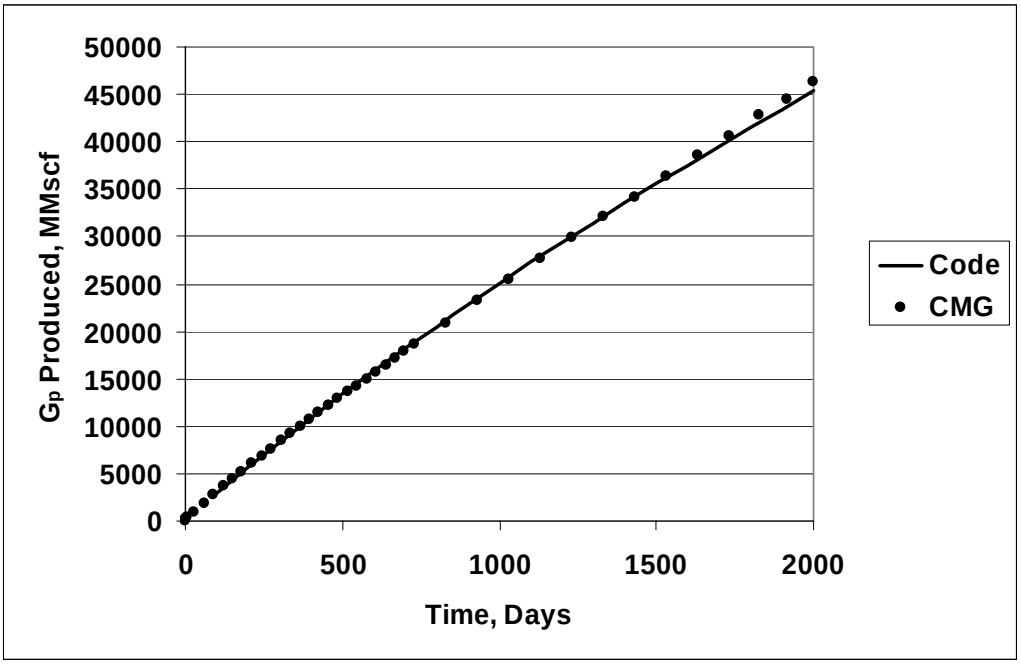


Fig. 4.11– Cumulative gas produced in the field

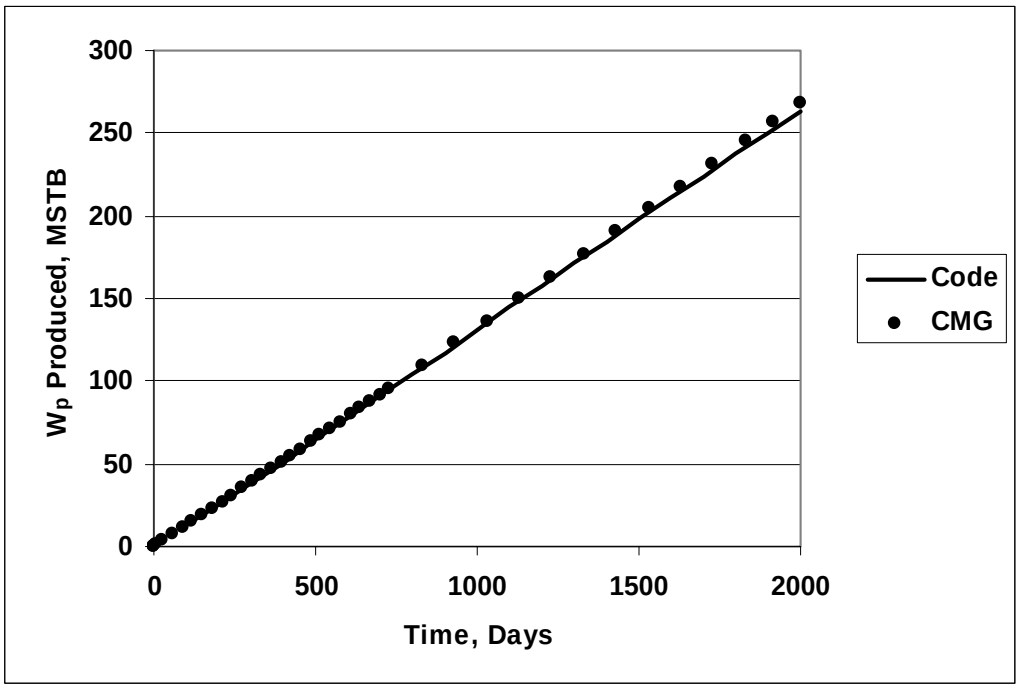


Fig. 4.12– Cumulative water produced in the field

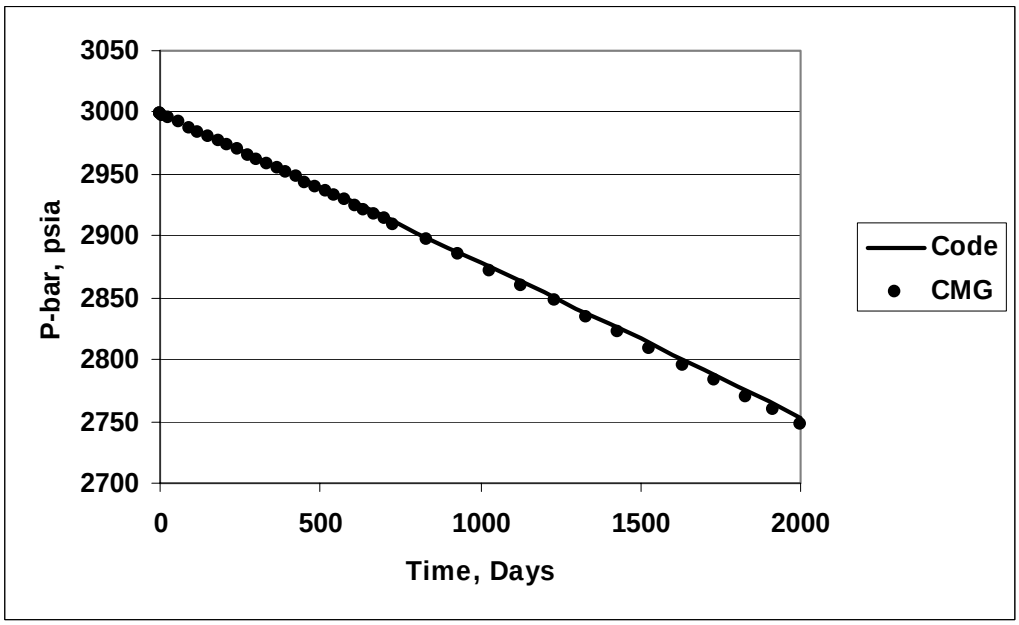


Fig. 4.13– Average reservoir pressure

Cumulative rates are among the most reliable parameters to compare with a commercially available simulator to validate the results and as can be seen from **Figs. 4.11** and **4.12** the results of the code match almost perfectly with those of CMG. **Fig. 4.13** exhibits the declining regime of the average reservoir pressure which has a good match with CMG.

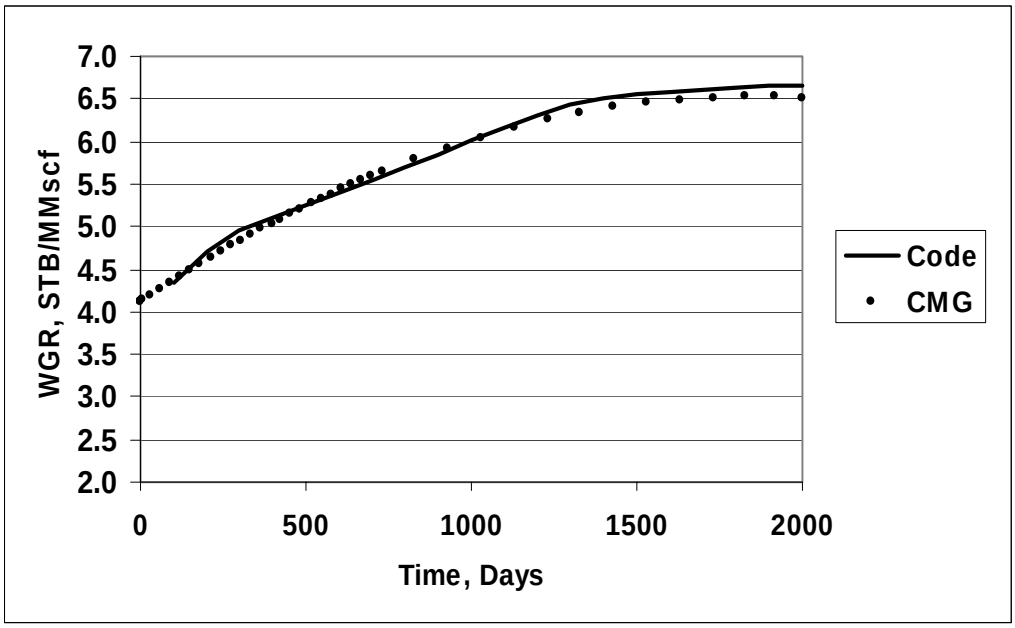


Fig. 4.14– Water/Gas ratio for the field

Fig. 4.14 shows the match of water/gas ratio, WGR, for the whole reservoir. All of above plots illustrate that the results of the code for different parameters are within an acceptable range compared to the same set of results generated by CMG for the same set of reservoir and fluid input data.

4.3 Validation of Plan 2: Injector-Producer Case

In order to assure the developed program is capable of running reliably for all cases it has been designed for; another simulation scheme is run and tested with CMG results. In this case all model and fluid data are similar to those of the first case, but instead of three producer wells, there are one injector and one producer in the model. The constraints and other entries of wells for case one (three producer wells) are tabulated in **Table 4.4**.

Table 4.4–Wells entries for case 2: Injector-Producer

Well ID	X	Y	Z	Well Type	Constraint	Constraint Value	Min. BHP
W-1	10	10	3	Injector	Const. BHP	2,500 psia	1,000 psia
W-2	15	15	8	Producer	Const. Water Rate	40 STB	1,000 psia

Fig. 4.15 demonstrates the comparison of the water injection rate between the two simulators which shows an almost linear behavior over the life of the reservoir. Since the producer well produces with constant water rate constraint of 40 STB/Day, the water rate plot is not made for that. From the other hand, since the injector well injects with constant bottomhole pressure constraint of 2,500 psia, the match of P_{wf} plot with CMG is shown for the producer well along with the gas production rate match on one single graph in **Fig. 4.16**.

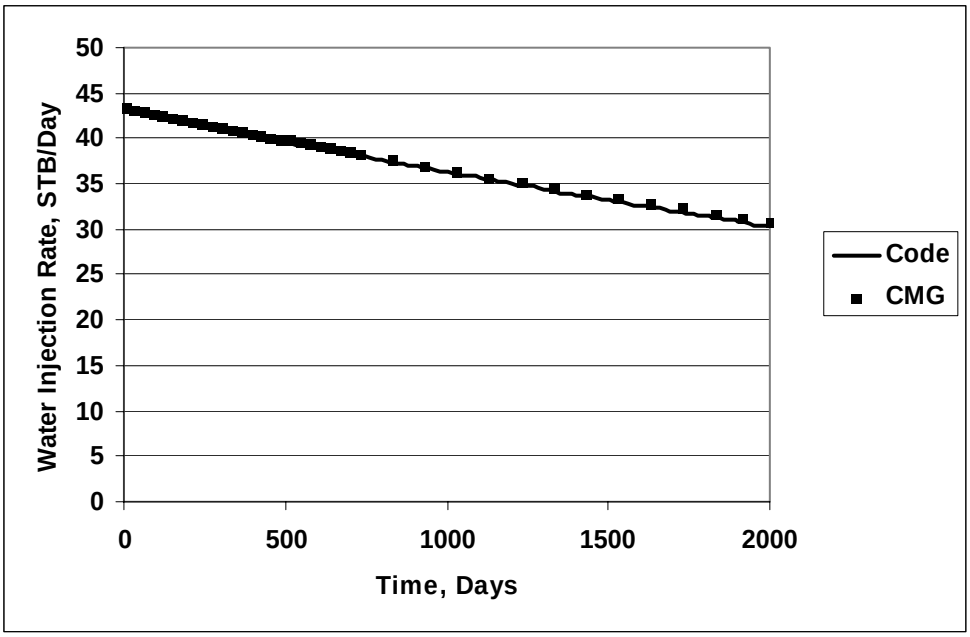


Fig. 4.15– Water injection rate

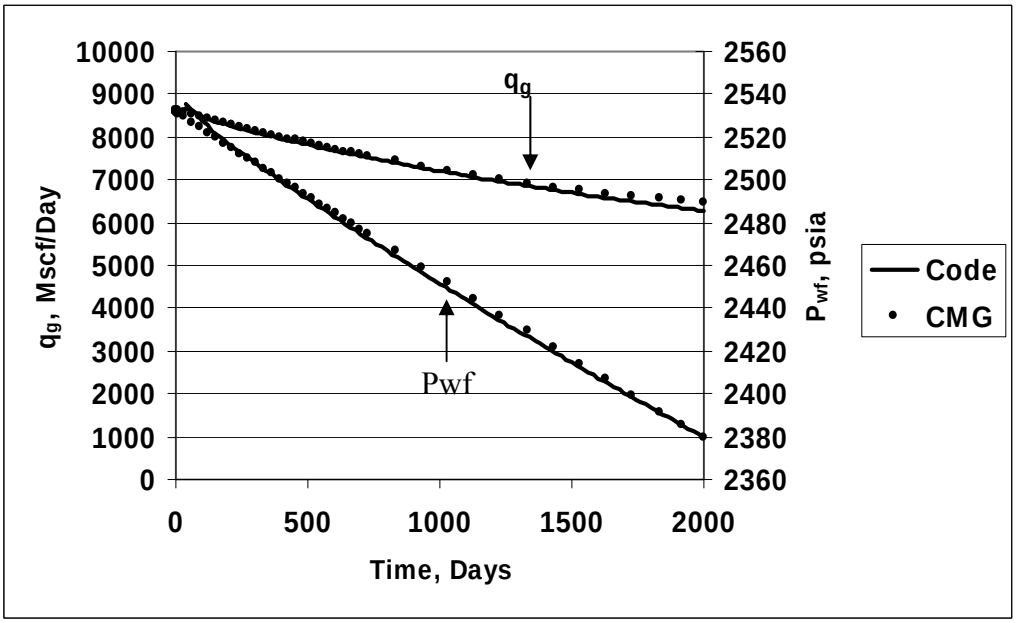


Fig. 4.16– q_g and p_{wf} for the producer

4.4 Gas Volumes and Material Balance Calculations

An estimate of the original gas in place (OGIP or G_i) for volumetric gas reservoirs with mobile water can be obtained from volumetric gas material balance considerations by equating the reservoir pore volume occupied by the gas at initial conditions to that occupied by the gas at some later conditions following gas and water production and the associated pressure reduction. Referring to the tank type model in **Fig. 4.17**, we write the material balance equation as[8,26]:

$$GB_{gi} = (G - G_p)B_g - W_p B_w \quad \text{.....} \quad (4.1)$$

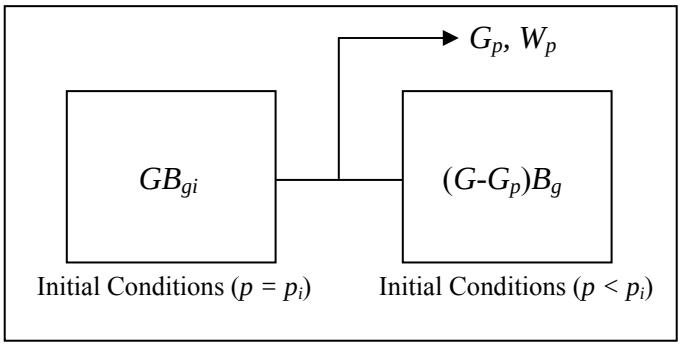


Fig. 4.17– Tank type model for a volumetric dry gas reservoir with water production

$$GB_{gi} = GB_g - G_p B_g - W_p B_w \quad \text{.....} \quad (4.2)$$

$$G_p = G(1 - \frac{B_{gi}}{B_g}) - W_p \frac{B_w}{B_g} \quad \text{.....} \quad (4.3)$$

If we substitute the ratio of the gas formation volume factor evaluated at initial and later conditions, we can write Eq. 4.3 as:

$$G_p = G(1 - \frac{z_i p}{z p_i}) - W_p B_w \frac{p}{z} \quad \text{.....(4.4)}$$

$$G_p = G - G\frac{z_i}{p_i}\frac{p}{z} - W_p B_w \frac{p}{z} \quad \text{.....(4.5)}$$

$$\frac{p}{z} = (G\frac{z_i}{p_i} - W_p B_w) = G - G_p \quad \text{.....(4.6)}$$

$$\frac{p}{z} = \frac{G}{Gz_i / p_i - W_p B_w} - \frac{1}{Gz_i / p_i - W_p B_w} G_p \quad \text{.....(4.7)}$$

This linear relationship is the expression of a constant volume reservoir and assumes that rock and water expansion are negligible and that there is no net movement of gas into or out of the reservoir volume of interest[26, 27] so the reservoir pore volume occupied by gas remains constant over the reservoir's productive life.[8] A material balance plot of *p/z* vs. G_P for a volumetric, depletion drive gas reservoir with mobile water generates a line of slope $-1/(Gz_i / p_i - W_p B_w)$ with an intercept of $G/(Gz_i / p_i - W_p B_w)$ for $G_P = 0$. Extrapolation of the straight line to the G_P axis yields the OGIP. **Fig. 4.18** shows this plot for the first simulation case simulating by the code. Extrapolating the linear relation of suggests *p/z* vs. G_P by $p/z = -0.0253 G_p + 3328.1$ suggests that the original gas in the reservoir is 131.5 Bscf.

This same set of results simulated by CMG is plotted in **Fig. 4.19** and extrapolating *P/Z* vs. G_P suggests the OGIP is 136.1 Bscf which is within an acceptable range from the code's result.

In the real-gas law, in order to solve for the initial volume of gas at standard conditions, we can equate the number of moles of gas at initial conditions to the number of moles at standard conditions and rearrange them[12]:

$$G = \frac{p_i V_{gi}}{z_i T} \frac{z_{sc} T_{sc}}{p_{sc}} \text{ . ……. ……..……. (4.8)}$$

Assuming the pore volume occupied by the gas is constant during the producing life of the reservoir gives:

$$V_{gi} = 43.56 Ah\phi(1 - S_{wi}) \text{ . ……. (4.9)}$$

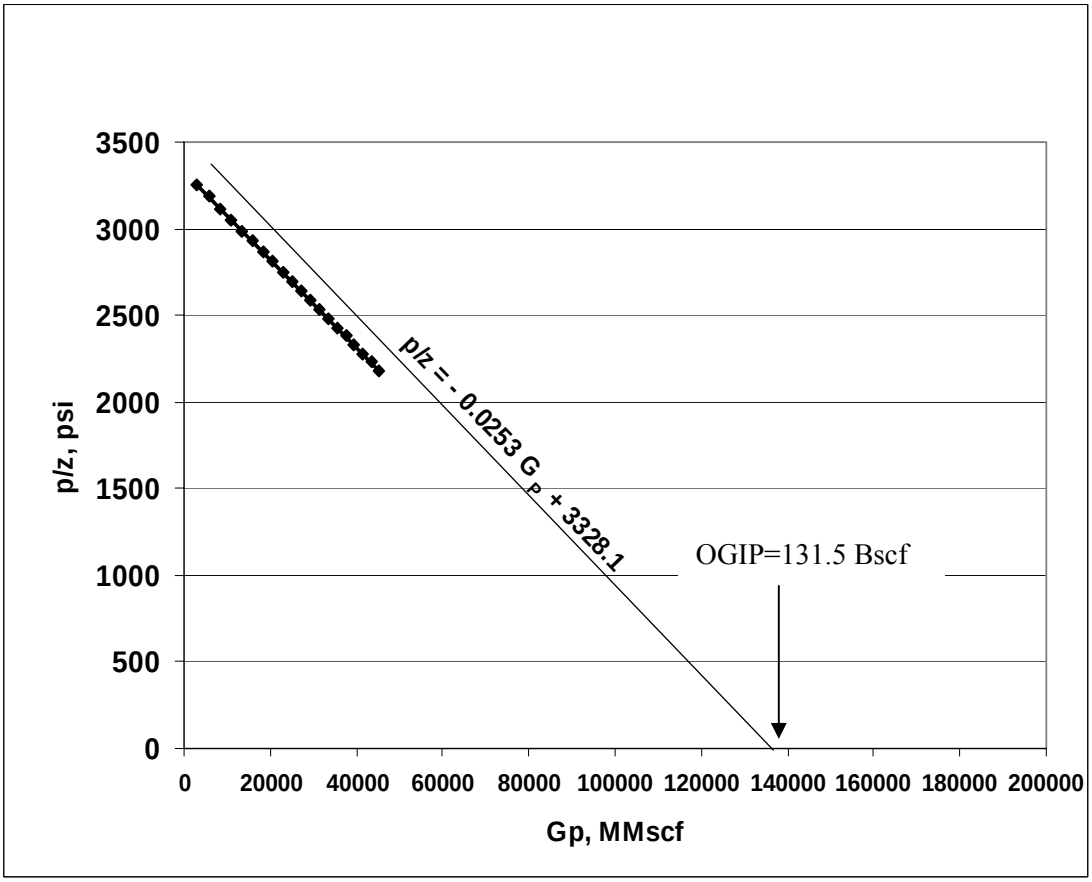


Fig. 4.18– *p/z* versus G_P generated by the code

Substituting Eq. 4.3 into Eq. 4.2 yields

$$G = 43.56AH\phi(1-S_{wi})\frac{p_i z_{sc} T_{sc}}{p_{sc} z_i T} \quad \text{..... (4.10)}$$

If we express the reservoir PV in barrels, Eq. 4.4 becomes

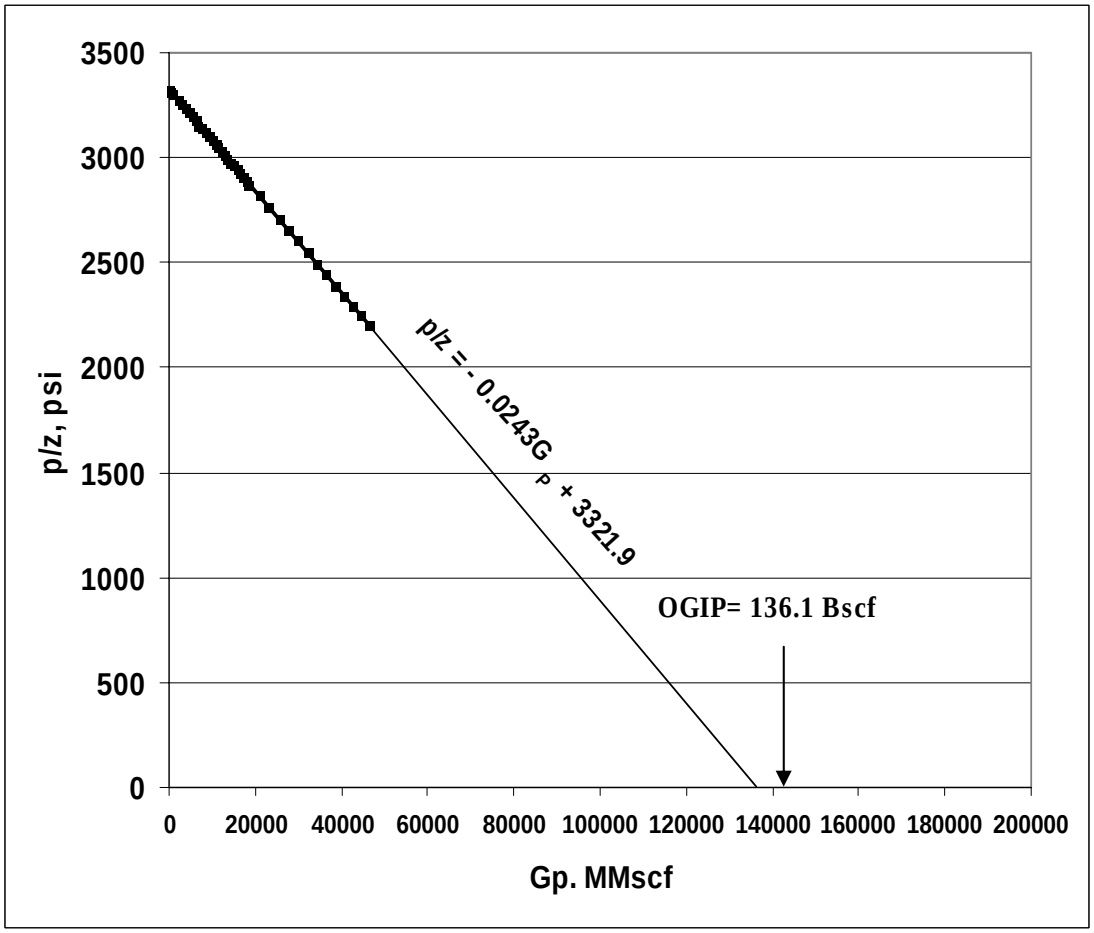


Fig. 4.19– *p/z* versus G_p generated by CMG

$$G = \frac{7758Ah\phi(1-S_{wi})}{B_{gi}} \quad \text{..... (4.11)}$$

This equation is applied in subroutine *Fluids in place* in the code assigning three-dimensional arrays for each parameter and yielded the value of **133.47 Bscf** for the

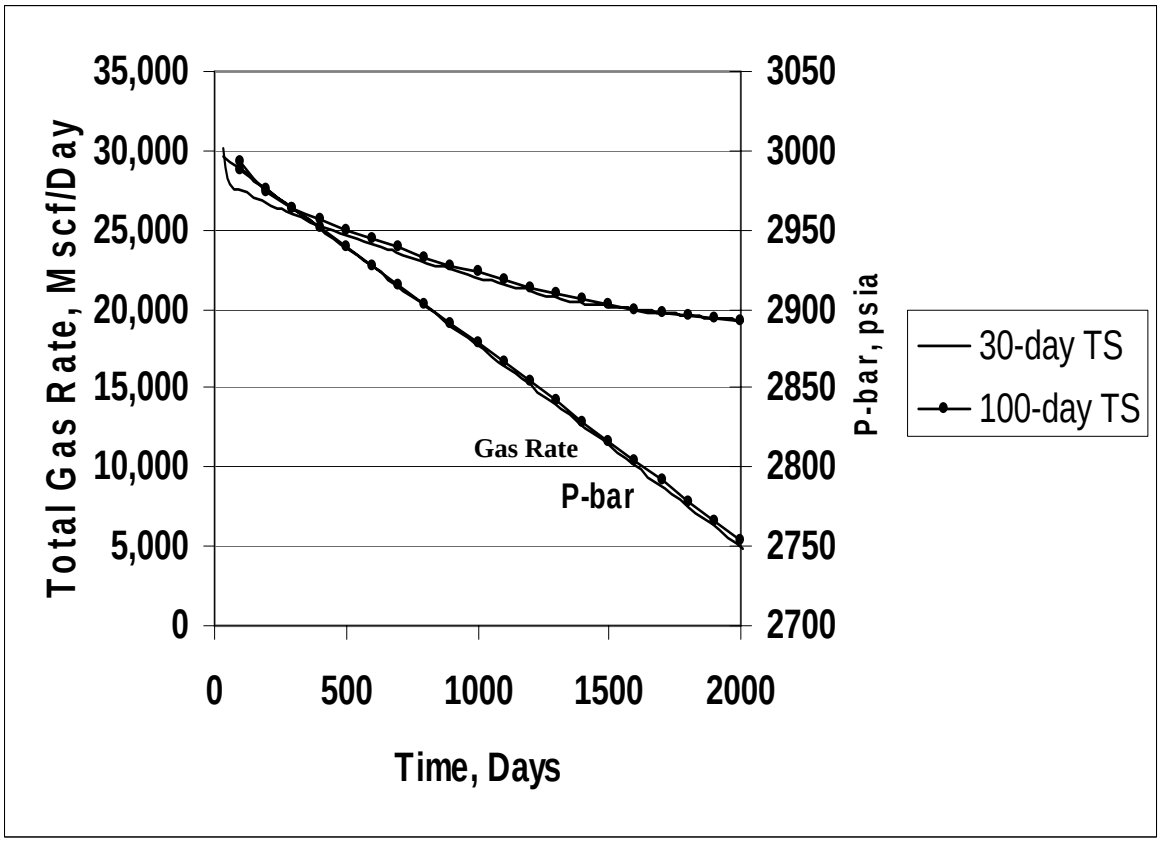


Fig. 4.20– Plots of total q_g using two different timestep sizes

original gas in place which agrees with the values of 131.5 and 136.1 Bscf calculated using the extrapolation of p/z vs. G_P for the code and CMG in **Figs. 4.18** and **4.19** respectively.

4.5 Analysis of the Depletion Schemes

In order to assure the solutions of the developed code converge to the same set of results using different timestep size, scheme one in previous parts, is run with a smaller time step size of 30 days instead of 100 days.

The outcome of this run showed all of the results match quite perfectly with those of the run with the bigger timestep size.

Fig. 4.20 shows the plots of the total gas rate and average reservoir pressure for the reservoir for the two different timestep sizes. As can be seen from this graph, both parameters follow the same trend which shows the material balance is perfectly satisfied in the code.

Another index to show the validity of the code's results is to compare gas and water saturation maps of the code with those of CMG at a few timesteps. These comparisons are made for layer 3 which contains the well with constant gas rate constraint.

Fig. 4.21 represents the gas saturation maps of this layer generated by both the code and CMG at timesteps 2, 10, 20 and 30 using a 50-day timestep size.

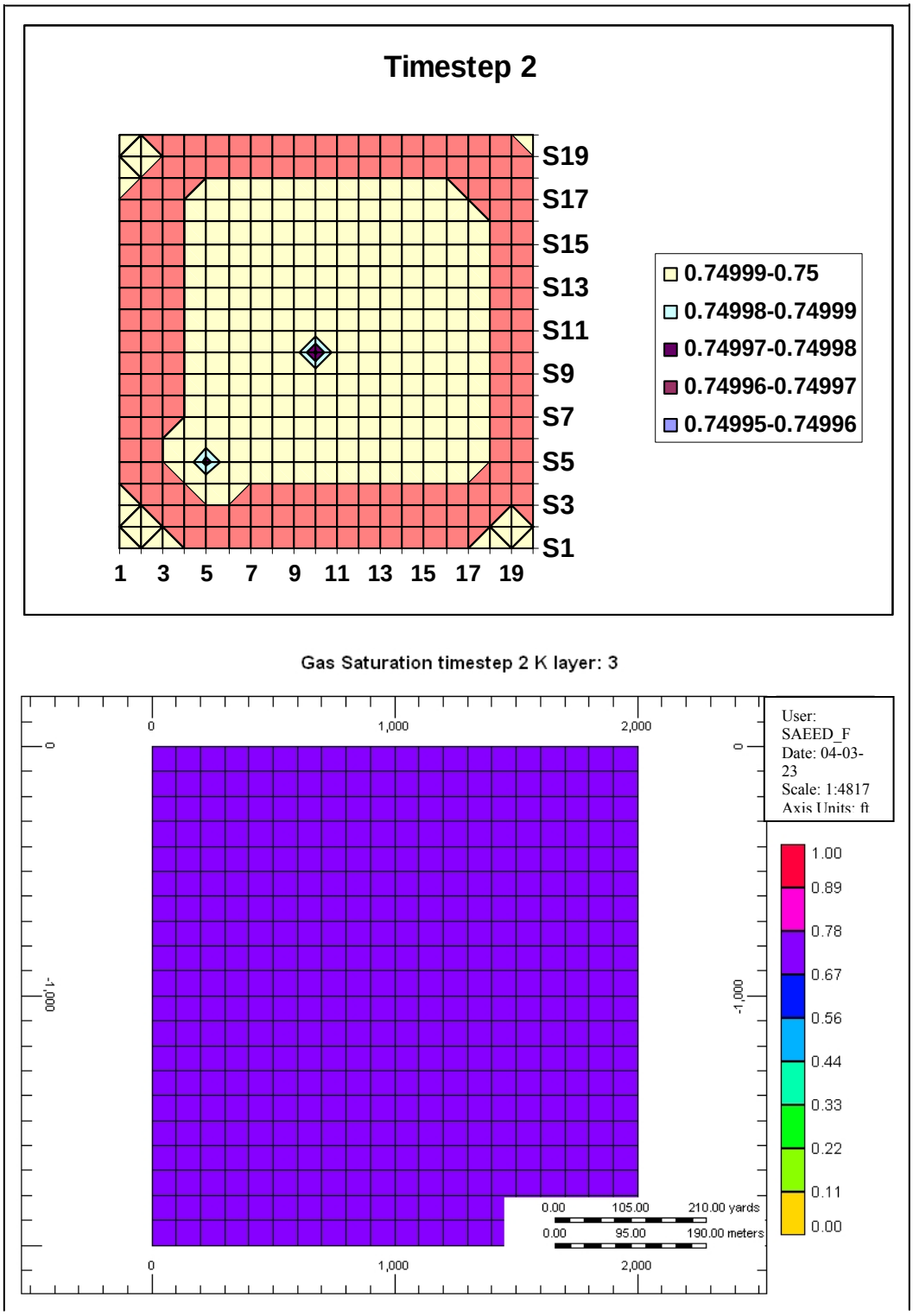


Fig. 4.21– Gas saturation maps of layer 3 at timesteps 2, 10, 20 and 30 generated by both the code and CMG

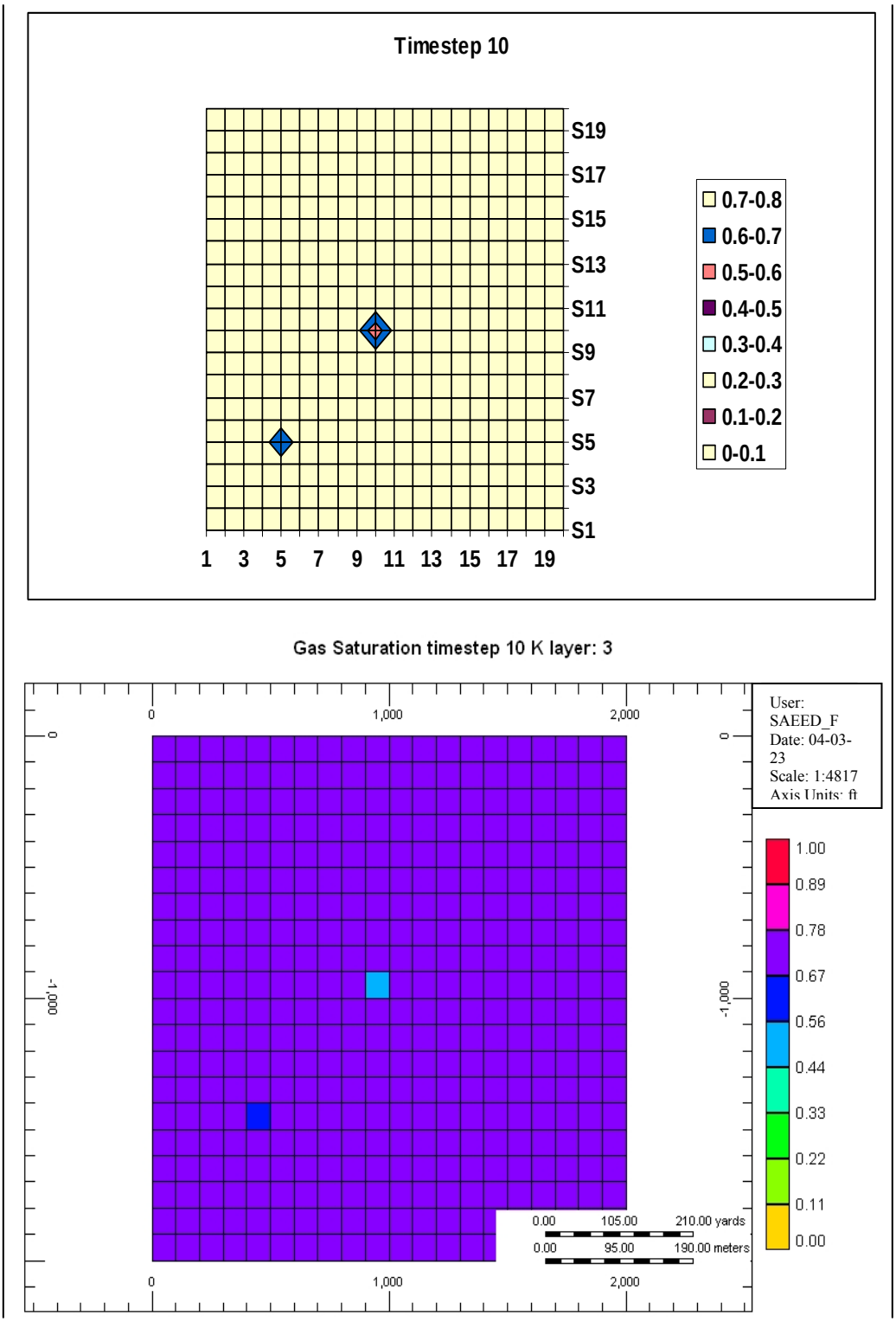


Fig. 4.21– Continued

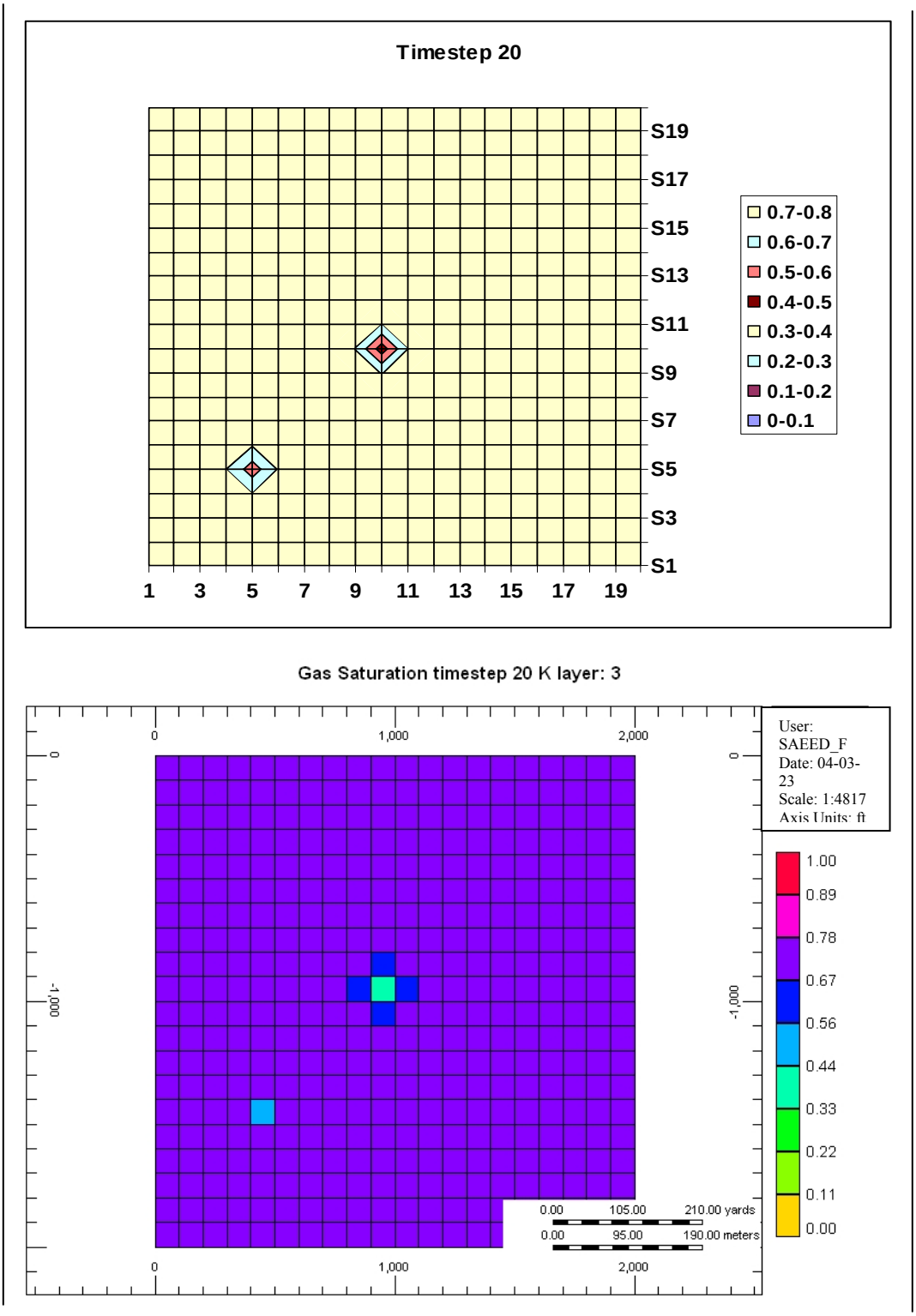


Fig. 4.21– Continued

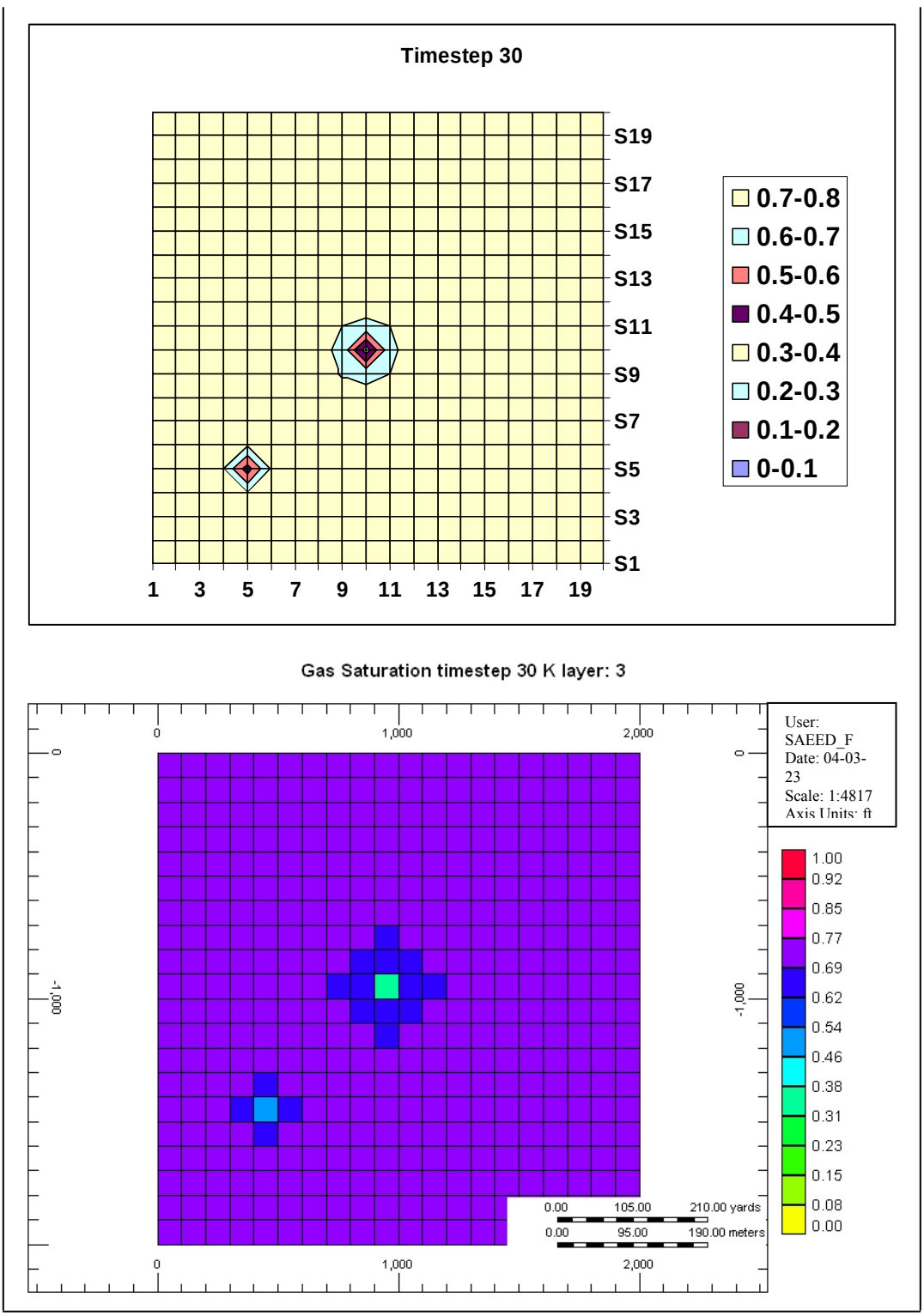


Fig. 4.21– Continued

Fig. 4.22 represents the water saturation maps of layer 3 generated by both the code and CMG at timesteps 2, 10, 20 and 30 using a 50-day timestep size.

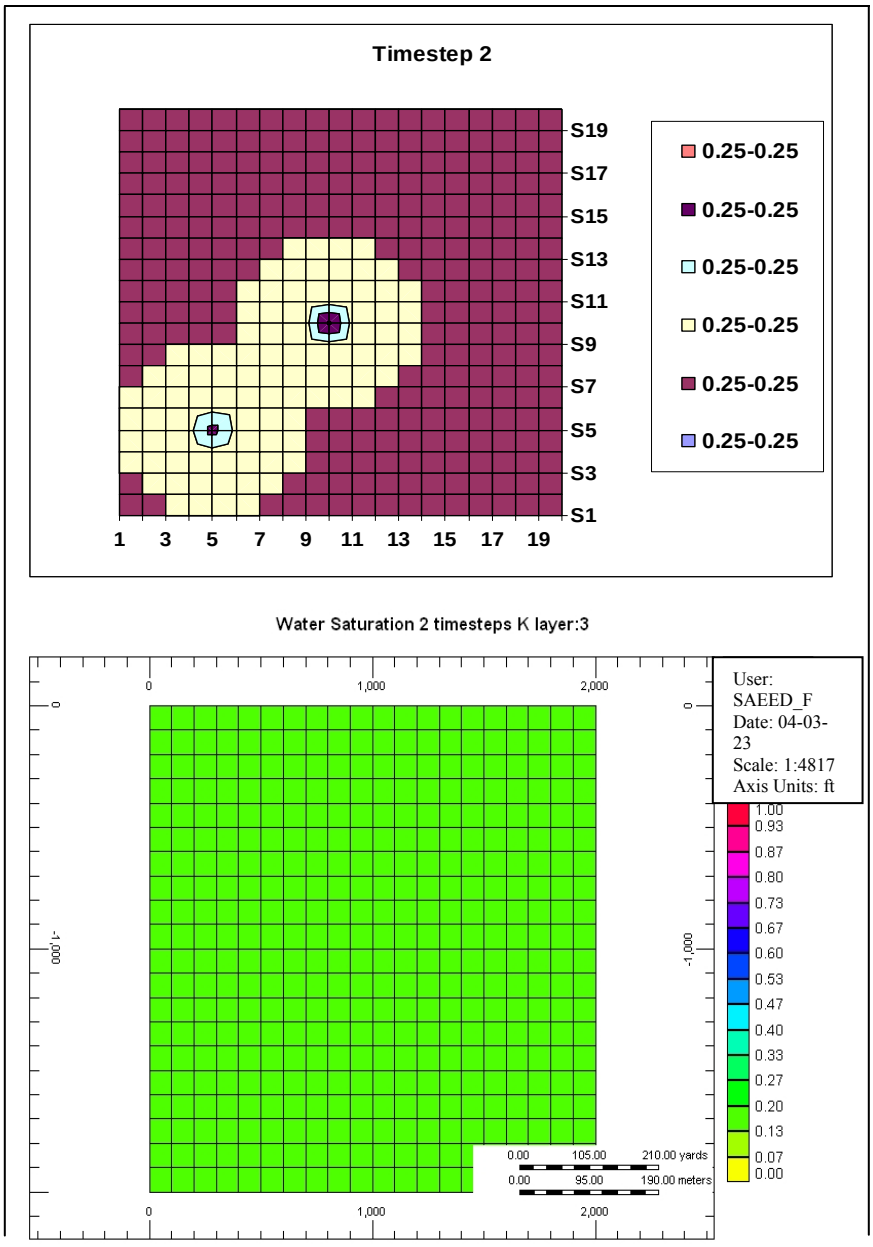
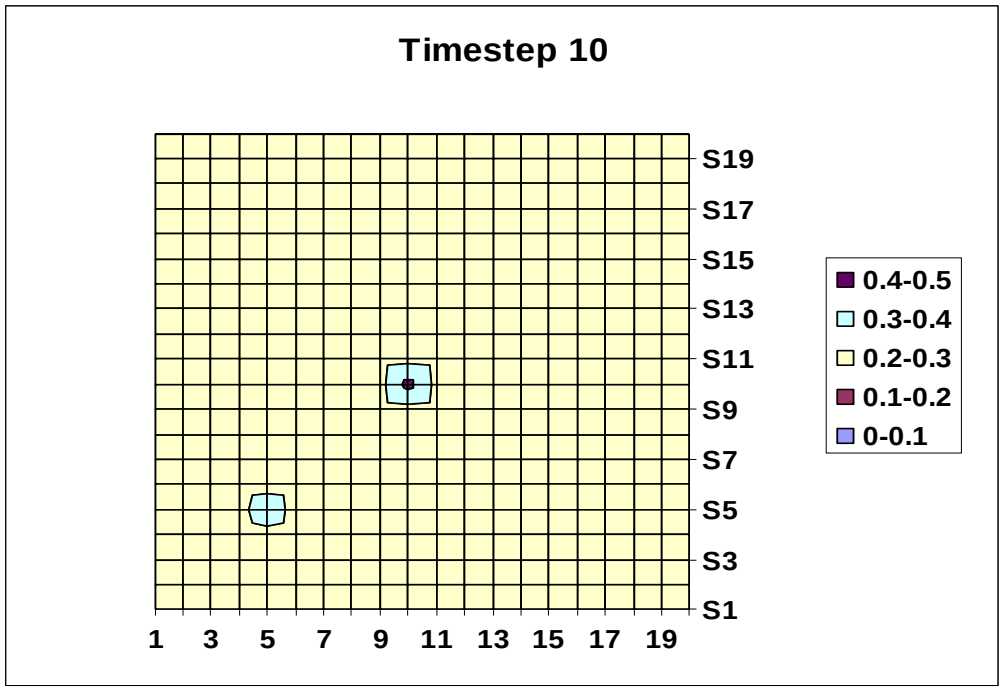


Fig. 4.22– Water saturation maps of layer 3 at timesteps 2, 10, 20 and 30 generated by both the code and CMG



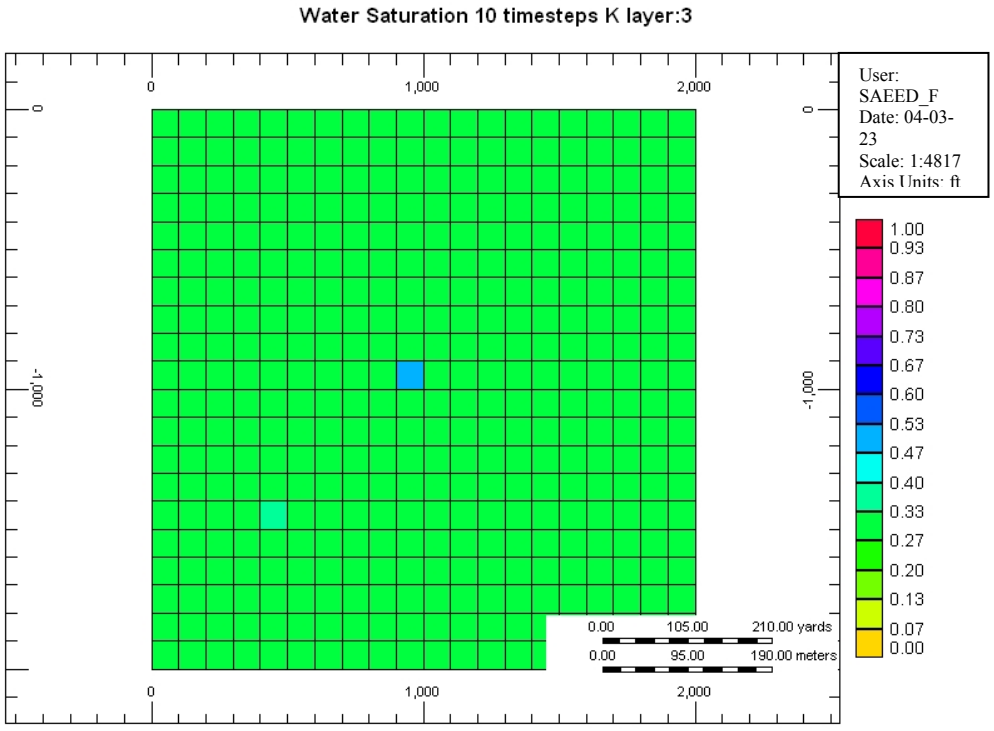


Fig. 4.22– Continued

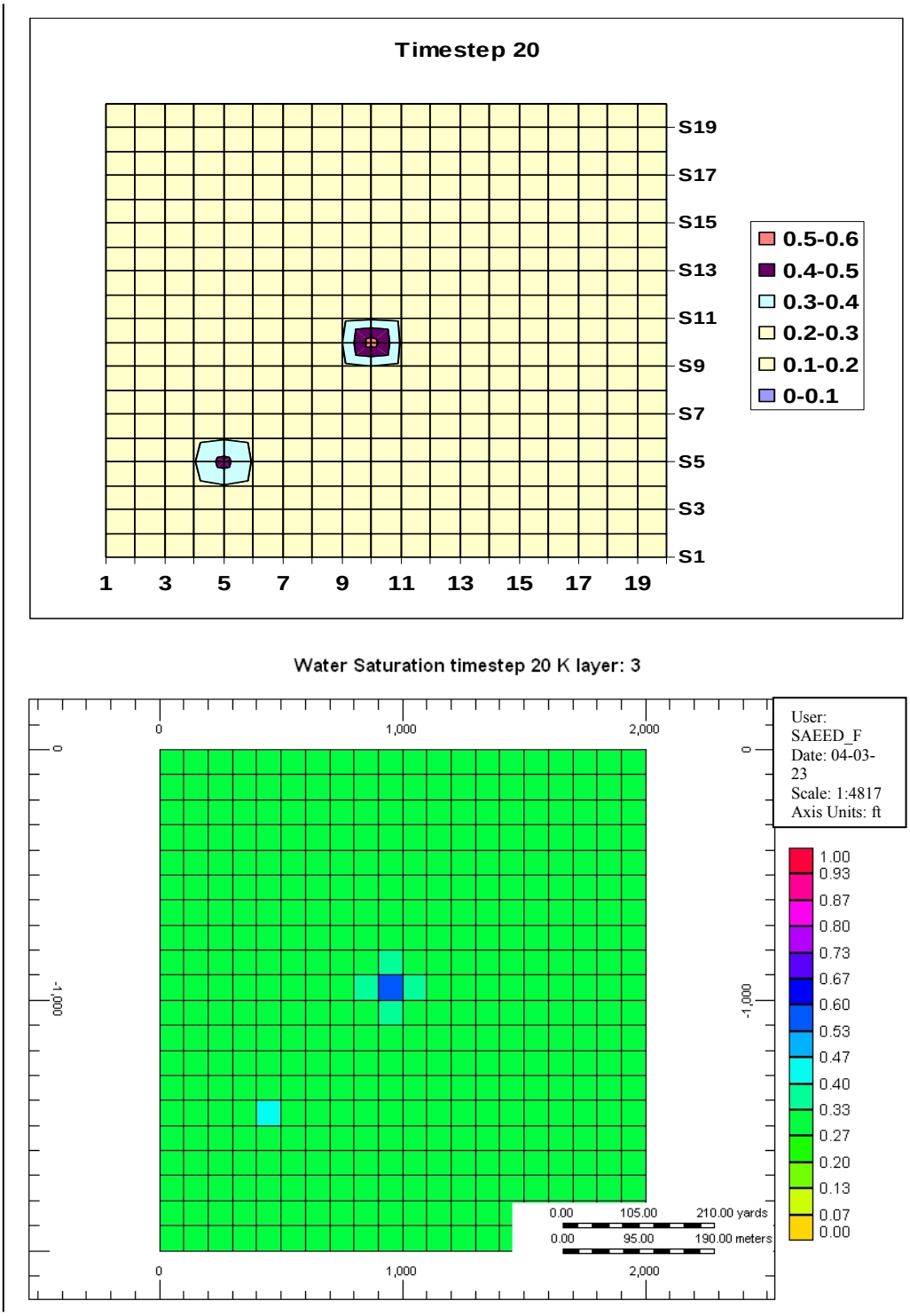


Fig. 4.22– Continued

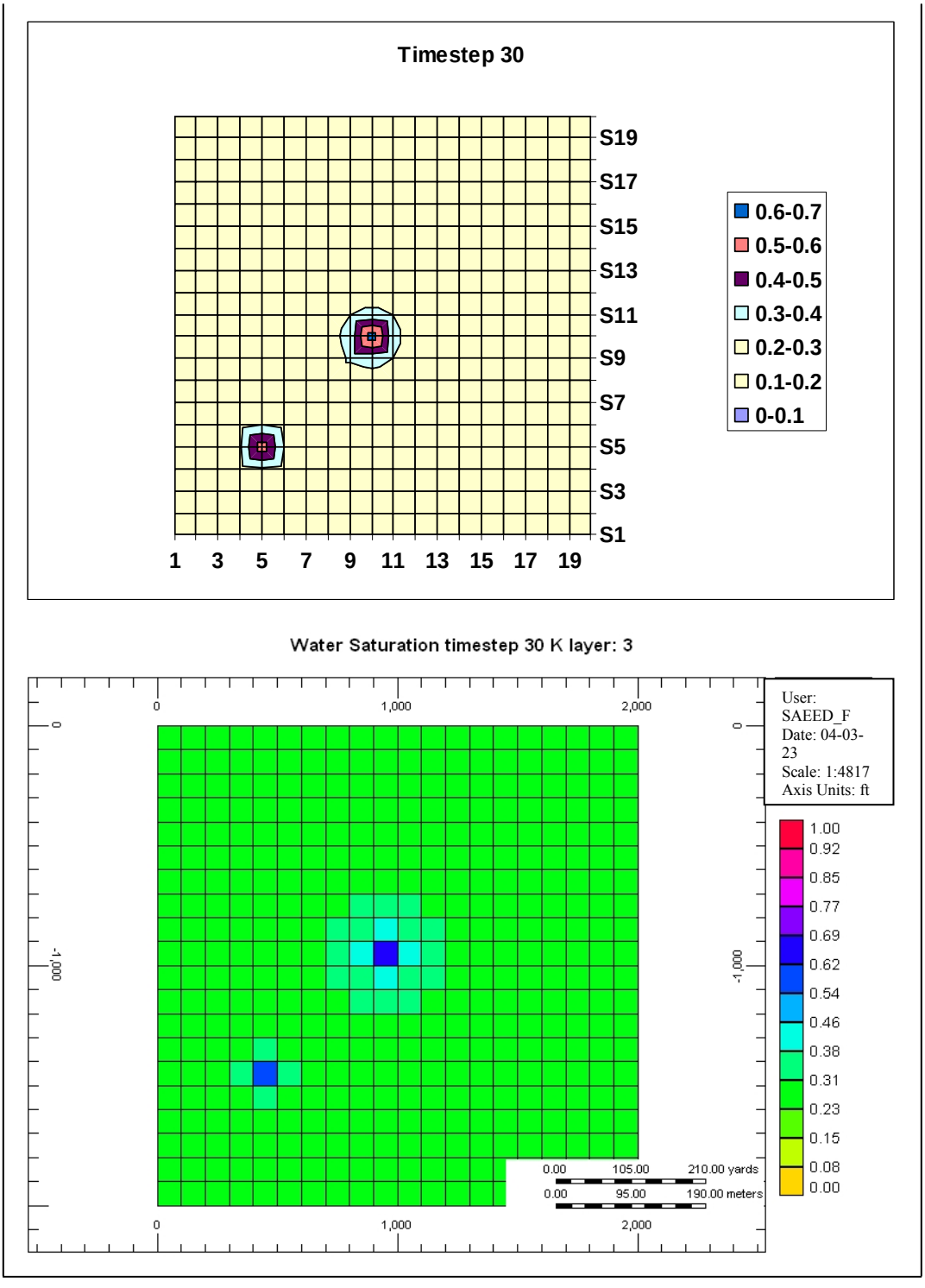


Fig. 4.22– Continued

Another analysis that is often made to evaluate validity of simulators is to test a case where a well undergoes a change in its constraint due to depletion. Such a case occurs when a well with constant gas or water rate constraint depletes P_{wf} up to the specified minimum bottomhole pressure. The well will produce with constant BHP constraint from that point forward.[20] **Fig. 4.23** shows gas rate and bottomhole pressure plots of such a case run by the developed code. In this scheme one producer well is producing with constant gas rate constraint of 10,000 Mscf/Day. After 800 days, since the BHP declines up to the specified minimum value of 2,300 psia, well continues to produce with constant BHP constraint.

Figs. 4.24 and **4.25** exhibit the cell pressures in layer 5 which contains the constant bottomhole pressure well for the first and the last timesteps. These projections represent the smooth depleting pressure of the reservoir around the wells in layer 5 and also the interference of the wells of other layers (layers 3 and 8) in the pressure depletion of layer 5.

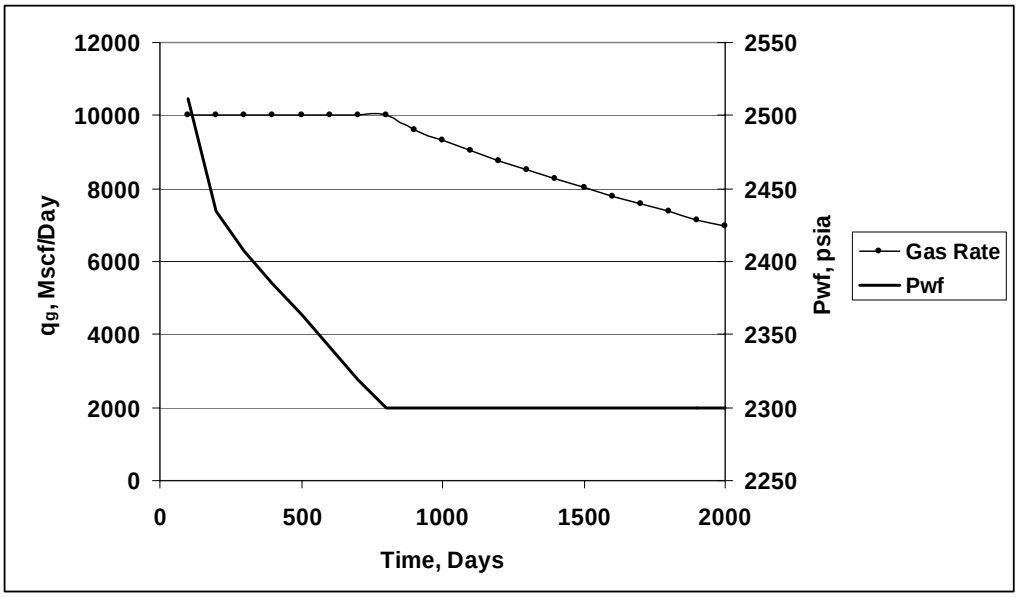


Fig. 4.23– Gas rate and p_{wf} plots for a well with changing constraint

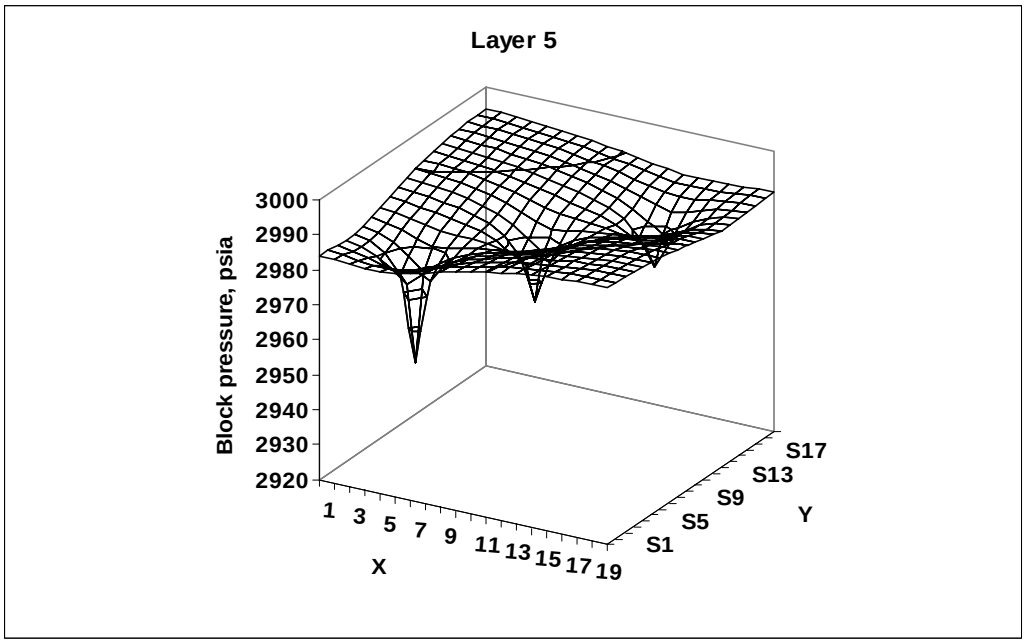


Fig. 4.24– Block pressure projection in layer 5 at the first timestep

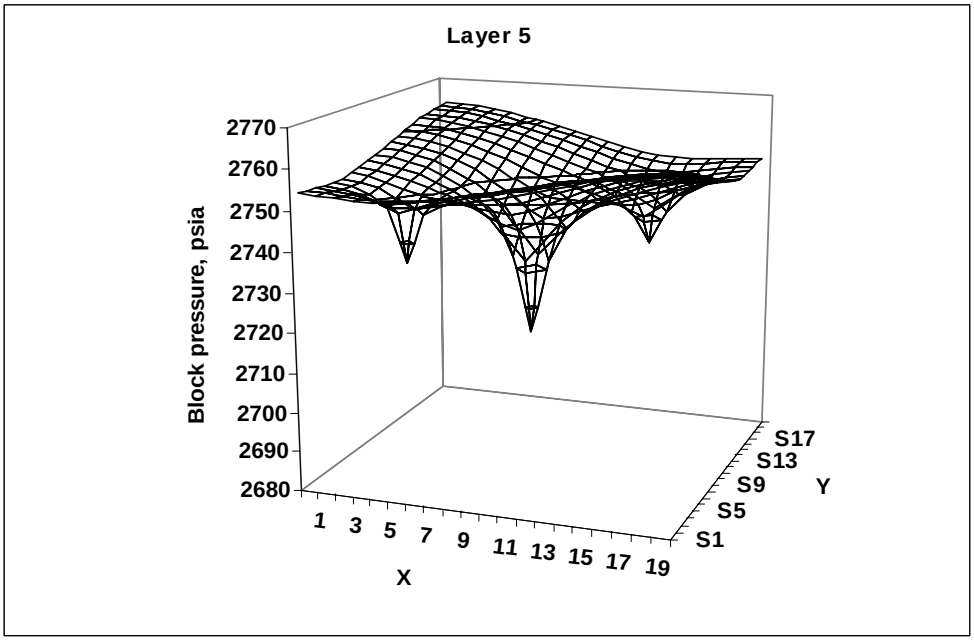


Fig. 4.25– Block pressure projection in layer 5 at the last timestep

In a dry gas reservoir being drained by the same two wells producing at different constant bottomhole pressures, the drainage volumes of both wells will be continuously changing.[20, 28] To illustrate such a case a model with reservoir and fluid properties similar to those of the plan one in previous sections, was run. In this scheme there two constant P_{wf} wells are producing with different bottomhole pressures of 2,200 and 2,800 psia. The wells are located symmetrically in the grid. **Fig. 4.26** exhibits the gas rates for these two wells and **Fig. 4.27** shows the block pressure projection of layer 7 for the 10th timestep for this case. Layer 7 is where the well with higher P_{wf} is located. The other

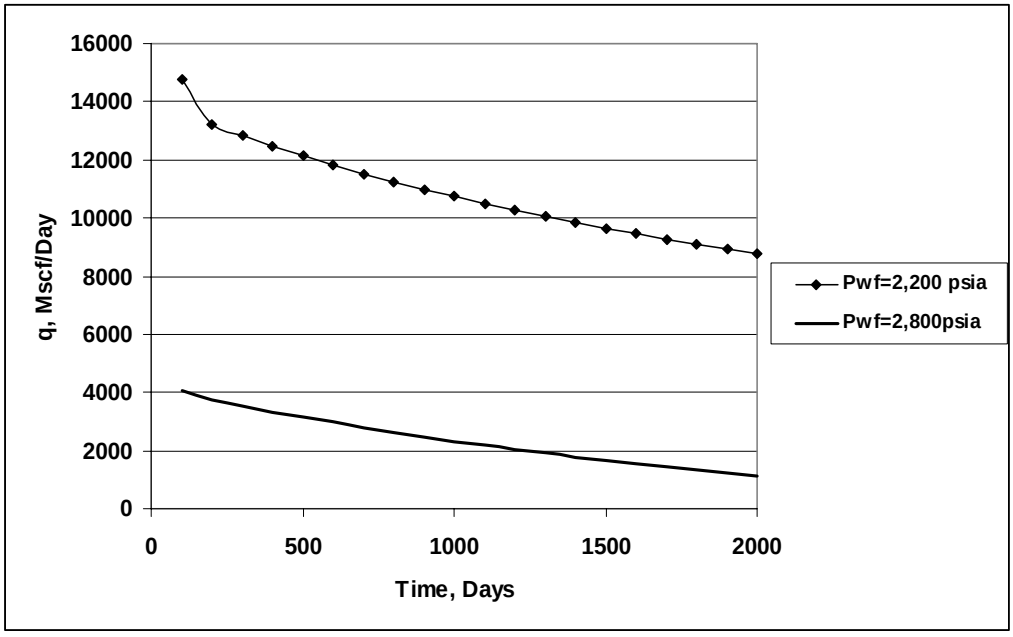


Fig. 4.26– Gas rates for the case of two wells with different p_{wf}

well is located in layer 4. As can be seen **Fig. 4.27** the well with lower P_{wf} continuously captures the production of the well with higher P_{wf} and the latter will stop producing a

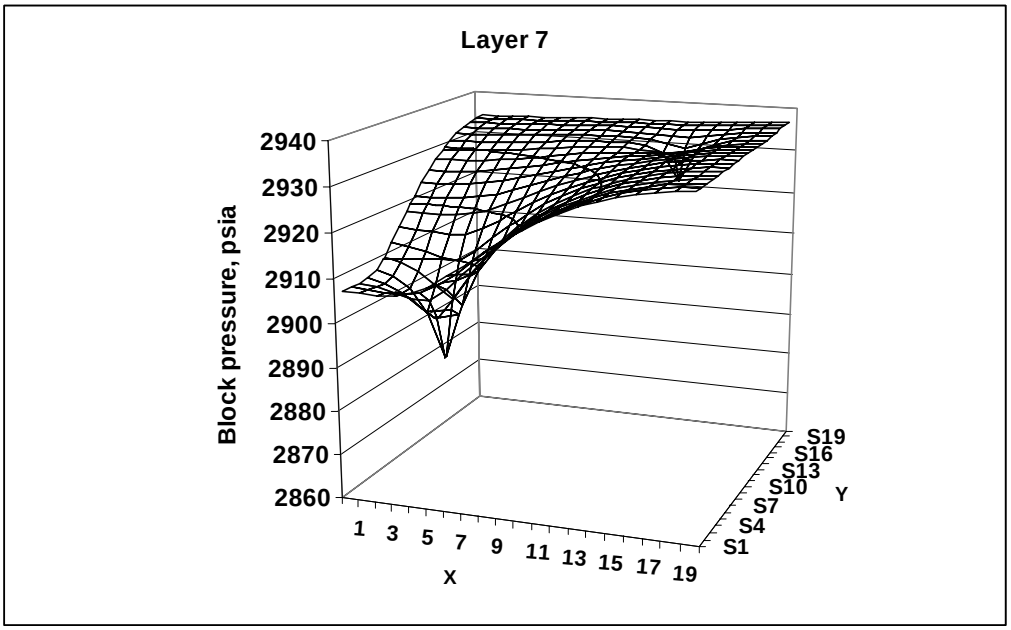


Fig. 4.27– Block pressure projection of layer 7 at the 10th timestep

lot faster than the well with lower P_{wf}.

Fig. 4.28 demonstrates the ratio of the gas rates of the wells in this plan is decreasing because the drainage volume of well 1 (with lower P_{wf}) is increasing at the expense of well 2. Also **Fig. 4.26** indicates well 2 has an accelerated decline as a result of shrinking drainage volume.

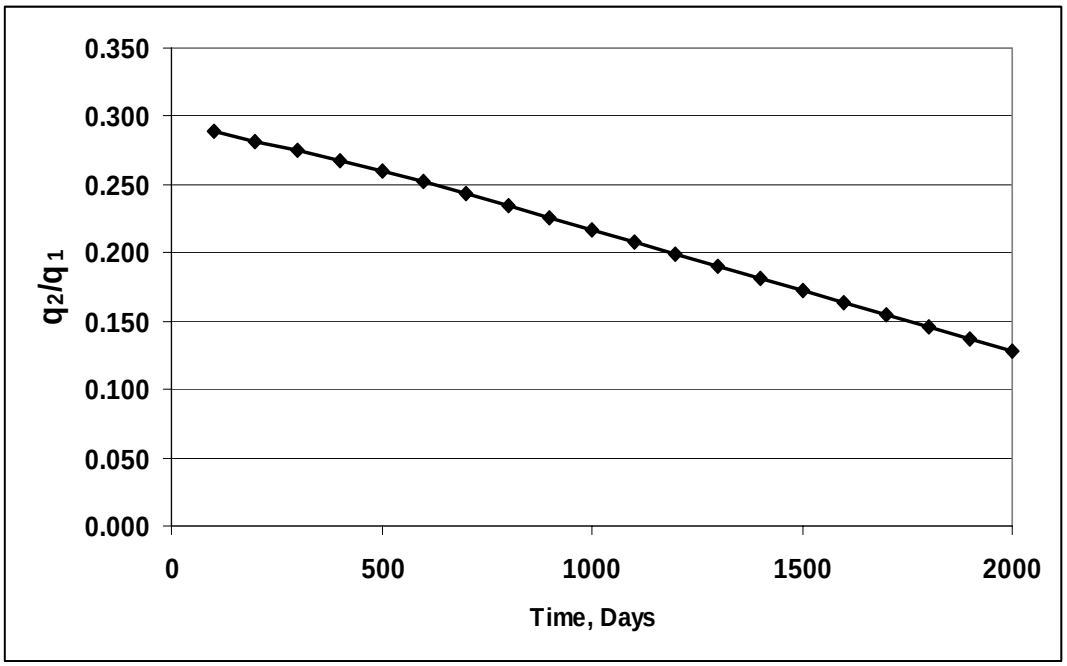


Fig. 4.28– Ratio of the gas rate of well 2 over that of well 1

CHAPTER V

CONCLUSIONS

The following conclusions can be derived from this study.

- The main advantage of the developed 3-D, 2-phase code is that it is specifically designed for volumetric dry gas reservoirs and then when solving problems of these reservoirs, it can be reliably used without a need to deal with more expensive commercial simulators.
- Since there is no gravity or p_c terms in this 2-phase formulation, IMPES approach shows less stability limitations.
- The developed IMPES code is competitive with well-recognized commercially available simulators. The results are reliable for different simulation plans, reservoir and fluid data and well configurations.
- The developed program can later be used as a part of a robust reservoir simulation.
- In volumetric dry gas reservoirs with multiple wells flowing at different flowing bottomhole pressure, the drainage volume of the wells with lower bottomhole pressure increases at the expense of the wells with higher bottomhole pressure. The drainage boundaries of the wells are therefore continuously changing.

REFERENCES

1. Al-Fattah, S.M. and Startzman, R.A.: "Analysis of Worldwide Natural Gas Production," paper SPE 57463 presented at the 1999 SPE Eastern Regional Meeting, Charleston, West Virginia, 20-22 October, 1999.

2. Staggs, H.M. and Herbeck, E.F.: "Reservoir Simulation Models-An Engineering Overview," *JPT* (December 1971) 1428-36.

3. Aziz, K. and Settari, A.: *Petroleum Reservoir Simulation*, Elsevier Science Publishing Co., New York, pp. 12-250, (1999).

4. British Petroleum, Statistical Review of World Energy 2003, 25 February 2004, http://www.bp.com/centres/energy.

5. Fanchi, J.R, *et al*,: *BOAST: A Three-Dimensional, Three-Phase Black Oil Applied Simulation Tool*, United States Department of Energy, Bartlesville, Oklahoma (1982).

6. Poolen, H.K.: "The Wise Use of Reservoir Models," *APEA J*. (1971) **11**, 131-34.

7. CMG Builder Results, Version 2002 User's Guide, Computer Modeling Group Ltd., Calgary, Alberta (2002).

8. Lee, J. and Wattenbarger, R.A.: *Gas Reservoir Engineering*, SPE Published Series, Richardson, Texas, pp. 81-184, (1996).

9. Robinson, J.B. and Ireland, M.M.: "Reserve Prediction Testing in Geopressured Gas Reservoir," SPE paper 16958 presented at the 1987 Annual Technical Conference and Exhibition of the Society of Petroleum Engineers held in Dallas, Texas, 27-30 September, 1987.

10. Cao, H.: "Development of Techniques for General Purpose Simulators," Ph.D. Dissertation, Stanford University, Stanford, California, June 2002.

11. McCain, W.D. Jr.: *The Properties of Petroleum Fluids*, Second Edition, PennWell Publishing Co., Tulsa, Oklahoma, pp. 72-75, (1989).

12. Calvin, C.C. and Dalton, R.L.: *Reservoir Simulation*, SPE Monograph Series, Monograph 13, Richardson, Texas, pp. 58-145, (1990).

13. Odeh, A.S. and Aziz, K.: "Comparison of Solutions to a Three-Dimensional Black-Oil Simulation Problem," *JPT* (January 1981) 13-25.

14. Carter, R.D.: "Performance Predictions for Gas Reservoir Considering Two-Dimensional Unsteady-State Flow," *Society of Petroleum Engineers Journal* (March 1966) 35-43.

15. Wattenbarger, R.A.: *Reservoir Simulation*, class notes for PETE 603, Texas A&M University, College Station, September 2002.

16. Blasingame, T.B.: *Fluid Flow in Porous Media*, class notes for PETE 620, Texas A&M University, College Station, September 2003.

17. Putra, E. and Schechter, D.S.: *Numerical modeling Notes*, NFR Group at Texas A&M University, College Station, Texas, June 2003.

18. Hallam, R. and Kin, M.L.: "Performance of Trinidad Gas reservoirs (Cassia, Immortelle, Flamboyant, Mahogany, Amherstia and Teak)," SPE paper 81010 presented at the 2003 SPE Latin American and Caribbean Petroleum Engineering Conference, Port-of-Spain, Trinidad, West Indies,27-30 April, 2003.

19. Ansah, J.: "Production Rate and Cumulative Production Models for Advanced Decline Curve Analysis of Gas Reservoirs," Ph.D. Dissertation, Texas A&M University, College Station, TX, August 1996.

20. Toh, S.K.: "The Depletion Performance of Heterogeneous Reservoirs," Ph.D. Dissertation, Texas A&M University, College Station, Texas, December 1997.

21. Begland, T.F.: "Depletion Performance of Volumetric High-Pressured Gas Reservoirs," SPE paper 15523 presented at the 1986 Annual Technical Conference and Exhibition of the Society of Petroleum Engineers, New Orleans, Louisiana, 5-8 October, 1986.

22. Qasem, F.H. and Gharbi, R.: "Gas Well Decline Analysis Under Constant –Pressure Conditions, Wellbore Storage, Damage, and Non-Darcy Flow Effects," SPE paper 75526 presented at the 2002 SPE Gas Technology Symposium, Calgary, Canada, 30 April-2 May, 2002.

23. Wattenbarger, R.A. and El-Banbi, A.H.: "Analysis of Linear Flow in Gas well Production," SPE paper 39972 presented at the 2002 SPE Gas Technology Symposium, Calgary, Canada, 15-18 March, 2002.

24. Vinsome, P.K.: "Orthomin, an Iterative Method for Solving Sparse Sets of Simultaneous Linear Equations," paper SPE 5729 presented at the 1976 Fourth Symposium of Numerical Simulation of Reservoir Performance of the Society of Petroleum Engineers of AIME, Los Angeles, California,19-20 February, 1976.

25. Jensen, O.K.: ”An Automatic Timestep Selection Scheme for Reservoir Simulation,” paper SPE 9373 presented at the 1980 SPE Annual Conference and Exhibition, Dallas, Texas, 21-24 September, 1980.

26. Baptise B.J. and Jagai, T.: “P/Z Analysis of a Mature Gas Condensate Field, Offshore Trinidad,” paper SPE 81009 presented at the 2003 SPE Latin American and Caribbean Petroleum Engineering Conference, Port-of-Spain, Trinidad, West Indies, 27-30 April, 2003.

27. Wattenbarger, R.A. and Ibrahim, M.: “Determination of OGIP for Wells in Pseudo-Steady-Old Techniques, New Approaches,” paper SPE 4286 resented at the 2003 SPE Annual Conference and Exhibition, Denver, Colorado, 5-8 October, 2003.

28. Aminian, K. and Ameri, S.: “Gas-Well Production Decline in Multiwell Reservoirs,” SPE 18268 resented at the 1988 SPE Annual Conference and Exhibition, Houston, Texas, 2-5 October, 1988.

APPENDIX

3-D, 2-PHASE VBA CODE LISTING

```
'3-D 2-Phase Simulator For Volumetric Dry Gas Reservoirs
'Simulation Units: Field Units
'Water, Gas FVF: rcf/scf
'Water Rate: STB/Day
'Gas Rate: scf/Day
Option Explicit
Option Base 1
Public nx As Integer, ny As Integer, nz As Integer          'Number of
grid blocks and their sizes
Public Xsize As Double, Ysize As Double, Zsize As Double
Public xD() As Double, yD() As Double, zD() As Double, _
        dx() As Double, dy() As Double, dz() As Double          'Grid
dimensions
Public xThick() As Double, yThick() As Double, zThick() As Double
'Rock properties
Public poro() As Double, perm() As Double, permX() As Double, _
        permY() As Double, permZ() As Double, Cf As Double, _
        TW() As Double, TE() As Double, TN() As Double, TS() As Double,
_
        TT() As Double, TB() As Double, PV() As Double, _
        PV1() As Double, PoreVol As Double, PoreVolTime() As Double
'Well properties
Public rw() As Double, Skin() As Double, Pwf() As Double, _
        ro() As Double, QW() As Double, QG() As Double, _
        QT() As Double, MinBHP() As Double, JMODEL() As Double, _
        nwell As Integer, Wellname() As String, XCoor() As Integer, _
        YCoor() As Integer, ZCoor() As Integer, TypeWell() As String, _
        WellConstraint() As String, TypeWell_read() As String, _
        rw_read() As Double, Skin_read() As Double, QT_read() As
Double, _
```

```
        QW_read() As Double, QG_read() As Double, _
        Pwf_read() As Double, MinBHP_read() As Double, _
        LambdaW() As Double, LambdaG() As Double, LambdaT() As Double
Public BCW As Byte, BCE As Byte, BCN As Byte, BCS As Byte, _
        BCT As Byte, BCB As Byte                              'Boundary
conditions
Public npvt As Integer, nkr As Integer, nkrg As Integer    'Number of
PVT/Kr input data
'Fluid properties
Public Ppvt() As Double, BW() As Double, _
        BG() As Double, UW() As Double, _
        UG() As Double, CW() As Double, _
        CG() As Double, CT As Double
Public BWI() As Double, BGI() As Double, _
        UWI() As Double, UGI() As Double, _
        CWI() As Double, CGI() As Double, _
        CTOT() As Double, BWI1() As Double, _
        BGI1() As Double, dRdP As Double
'Average properties for Matrix coefficients
Public UWavw() As Double, UGavw() As Double, _
        BWavw() As Double, BGavw() As Double, _
        UWave() As Double, UGave() As Double, _
        BWave() As Double, BGave() As Double, _
        UWavs() As Double, UGavs() As Double, _
        BWavs() As Double, BGavs() As Double, _
        UWavn() As Double, UGavn() As Double, _
        BWavn() As Double, BGavn() As Double, _
        UWavb() As Double, UGavb() As Double, _
        BWavb() As Double, BGavb() As Double, _
        UWavt() As Double, UGavt() As Double, _
        BWavt() As Double, BGavt() As Double
'Relative Permeabilities parameters
Public SW() As Double, SG() As Double, SL() As Double, _
        KRW() As Double, KRG() As Double, _
        SWI() As Double, SGI() As Double, SLI() As Double, _
        KRWI() As Double, KRGI() As Double, _
```

```
        KRWUPS As Double, KRGUPS As Double, _
        SWI1() As Double, SGI1() As Double, SLI1() As Double, SWC As
Double
        'KROG() As Double, KROW() As Double,KROWI() As Double,KROGI()
As Double, KROI() As Double
'Matrix Elements
Public aW() As Double, aww() As Double, agw() As Double, _
        ac() As Double, awc() As Double, agc() As Double, _
        aE() As Double, awe() As Double, age() As Double, _
        aN() As Double, awn() As Double, agn() As Double, _
        aSt() As Double, aws() As Double, ags() As Double, _
        aT() As Double, awt() As Double, agt() As Double, _
        aB() As Double, awb() As Double, agb() As Double, _
        MB() As Double, betha() As Double
Public press() As Double, p() As Double, pn() As Double      'Pressure
terms
Public Pinit As Double, PSum As Double                       'Initial
conditions
'To check stability and accuracy of solution and time step control
Public Sat_diff() As Double, Satmax As Double, Check As Boolean, _
        Count As Byte, ncuts As Byte, dt1 As Double, dt As Double, _
        tmax As Double, time As Double
'Fluids in place and cumulative production
Public OWIP() As Double, OGIP() As Double, _
        TotalWIP As Double, TotalGIP As Double, _
        CumWater As Double, CumGas As Double
'Matrix Solver Elements
Public TOL As Double, II As Integer, JJ As Integer, KKK As Integer, _
        IJKM As Long, ITMAX As Double, QI() As Double, AQI() As Double,
_
        AL3() As Double, AL2() As Double, AL1() As Double, _
        AD() As Double, AU1() As Double, AU2() As Double, _
        AU3() As Double, QN() As Double, AQN() As Double, RN() As
Double, _
        DXN() As Double, ADX() As Double, Psim() As Double, IT As Long
'Report variables
```

```
Public rc As Integer, MyCount As Integer


Sub Main()
    Call Read_data
    Call MemAlloc
    Call Wells
    Call Initial
    time = 0
    rc = 0
 Do
        rc = rc + 1
        Call Properties
        Call Rates_con_rate
        Count = 0
        Do
            Call MatrixB
            Call MatrixA
            Call Matrix_Solver
            Call Rates_con_bhp
            Call Saturations
            If Count <= ncuts And Check = False Then
                dt1 = dt1 / (2 + Count)
                Count = Count + 1
            Else
                time = time + dt1
                Call Cum_production
                dt1 = dt
                Exit Do
            End If
        Loop
        Call Update
        Call Report
        Call SgUpdate
Loop Until time >= tmax
Close #2
```

```
End Sub

Sub Initial()
'Initial conditions for simulation
Dim i As Integer, j As Integer, k As Integer
With ThisWorkbook.Sheets("RESULTS"): .Cells.ClearContents: End With
With ThisWorkbook.Sheets("Pressure"): .Cells.ClearContents: End With
With ThisWorkbook.Sheets("Sw"): .Cells.ClearContents: End With
With ThisWorkbook.Sheets("Sg"): .Cells.ClearContents: End With
With ThisWorkbook.Sheets("Sw+Sg"): .Cells.ClearContents: End With
With ThisWorkbook.Sheets("WELLS"): .Cells.ClearContents: End With

PoreVol = 0
CumWater = 0
CumGas = 0

For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            pn(i, j, k) = Pinit
            SWI(i, j, k) = SWC
            SGI(i, j, k) = 1 - SWC
            SWI1(i, j, k) = SWI(i, j, k): SGI1(i, j, k) = SGI(i, j, k)
            PV(i, j, k) = poro(i, j, k) * dx(i, j, k) * dy(i, j, k) *
dz(i, j, k)

        Next
    Next
Next

For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
          PoreVol = PoreVol + PV(i, j, k) / 5615
        Next
    Next
```

```
Next

Call InterpolaPVT
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            BWI1(i, j, k) = BWI(i, j, k)
            BGI1(i, j, k) = BGI(i, j, k)
            OWIP(i, j, k) = 0.1779685 * PV(i, j, k) * SWI(i, j, k) /
BWI(i, j, k) 'Unit conversion of 2.294E-5 acre/ft^2 * coeffifient of
7758
            OGIP(i, j, k) = 0.1779685 * PV(i, j, k) * SGI(i, j, k) /
BGI(i, j, k)
        Next
    Next
Next
Call Fluids_In_Place
Call Matrix_Initial
CumWater = 0#: CumGas = 0#
dt1 = dt: Check = True
End Sub


Sub Properties()

Call InterpolaPVT
Call InterpolaKr
Call Mobilities
Call Comp_Total
Call Trans
Call Avg_PVT

End Sub

Sub Update()
Dim i As Integer, j As Integer, k As Integer
```

```
'Update properties for the new time step
PSum = 0

For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            pn(i, j, k) = p(i, j, k)
            SWI(i, j, k) = SWI1(i, j, k)
            SGI(i, j, k) = SGI1(i, j, k)
            PV(i, j, k) = PV1(i, j, k)
            'PoreVol = PoreVol + PV(i, j, k) / 5615
            PSum = PSum + pn(i, j, k)
        Next
    Next
Next

Call Check_WellConstraints
End Sub


Sub Wells()
Dim i As Integer, j As Integer, k As Integer, m As Integer
Call Calc_dxdydz2

For m = 1 To nwell
            For k = 1 To nz
                If k = ZCoor(m) Then
                    For j = 1 To ny
                        If j = YCoor(m) Then
                            For i = 1 To nx
                                If i = XCoor(m) Then
                                    TypeWell(i, j, k) =
TypeWell_read(m)
                                    rw(i, j, k) = rw_read(m)
                                    Skin(i, j, k) = Skin_read(m)
                                    QW(i, j, k) = QW_read(m) * 5.615
```

```
                                    QG(i, j, k) = QG_read(m) * 1000
                                    Pwf(i, j, k) = Pwf_read(m):
MinBHP(i, j, k) = MinBHP_read(m)
                                End If
                            Next
                        End If
                    Next
                End If
            Next
Next

Call Identify_constraints

For m = 1 To nwell
            For k = 1 To nz
                If k = ZCoor(m) Then
                    For j = 1 To ny
                        If j = YCoor(m) Then
                            For i = 1 To nx
                                If i = XCoor(m) Then
                                    ro(i, j, k) = 0.28 * (((permY(i, j,
k) / permX(i, j, k)) ^ 0.5 * dx(i, j, k) ^ 2 + _
                                                        (permX(i, j, k) /
permY(i, j, k)) ^ 0.5 * dy(i, j, k) ^ 2) ^ 0.5) / _
                                                        ((permY(i, j, k) /
permX(i, j, k)) ^ 0.25 + (permX(i, j, k) / permY(i, j, k)) ^ 0.25)

                                    JMODEL(i, j, k) = 0.039772562 *
(permX(i, j, k) * permY(i, j, k)) ^ 0.5 * dz(i, j, k) / _
                                                            (Log(ro(i, j, k)
/ rw(i, j, k)) + Skin(i, j, k))

                                End If
                            Next
                        End If
                    Next
```

```
                End If
            Next
Next

End Sub


Function Interpolation(x As Double, a() As Double, b() As Double) As
Double
'x, the value of reference for interpolation
'A() the array of reference
'B() the array of values for interpolation
Dim i As Double
Dim A1 As Double, A2 As Double, B1 As Double, B2 As Double
'ReDim A(1 To n) As double, B(1 To n) As double
If a(LBound(a)) > a(UBound(a)) Then
    For i = LBound(a) To UBound(a) - 1
        If x <= a(i) And x > a(i + 1) Then
            A1 = a(i)
            A2 = a(i + 1)
            B1 = b(i)
            B2 = b(i + 1)
        End If
        If x > a(LBound(a)) Then
            A1 = a(LBound(a))
            A2 = a(LBound(a) + 1)
            B1 = b(LBound(a))
            B2 = b(LBound(a) + 1)
        End If
        If x < a(UBound(a)) Then
            A1 = a(UBound(a))
            A2 = a(UBound(a) - 1)
            B1 = b(UBound(a))
            B2 = b(UBound(a) - 1)
        End If
    Next i
```

```
Else
    For i = 1 To UBound(a) - 1
        If x >= a(i) And x <= a(i + 1) Then
            A1 = a(i)
            A2 = a(i + 1)
            B1 = b(i)
            B2 = b(i + 1)
        End If
        If x < a(LBound(a)) Then
            A1 = a(LBound(a))
            A2 = a(LBound(a) + 1)
            B1 = b(LBound(a))
            B2 = b(LBound(a))
        End If
        If x > a(UBound(a)) Then
            A1 = a(UBound(a))
            A2 = a(UBound(a) - 1)
            B1 = b(UBound(a))
            B2 = b(UBound(a))
        End If

    Next i
End If
Interpolation = B1 + (B2 - B1) / (A2 - A1) * (x - A1)

End Function


Function MaxValue(a() As Double) As Double
Dim i As Integer, j As Integer, k As Integer
MaxValue = a(LBound(a, 1), LBound(a, 2), LBound(a, 3))
For k = LBound(a, 3) To UBound(a, 3)
    For j = LBound(a, 2) To UBound(a, 2)
        For i = LBound(a, 1) + 1 To UBound(a, 1)
            If a(i, j, k) > MaxValue Then MaxValue = a(i, j, k)
        Next
```

```
    Next
Next
End Function


Function MinValue(a() As Double) As Double
Dim i As Integer
MinValue = a(UBound(a))
For i = LBound(a) + 1 To UBound(a)
    If a(i) < MinValue Then MinValue = a(i)
Next i
End Function

Function AritAvg(a As Double, b As Double) As Double
AritAvg = (a + b) / 2#
End Function

Function HarmAvg(a As Double, b As Double) As Double
HarmAvg = 2 * a * b / (a + b)
End Function


Sub Calc_dxdydz()
Dim i As Integer, im As Integer, ip As Integer
Dim j As Integer, jm As Integer, jp As Integer
Dim k As Integer, km As Integer, kp As Integer

For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            im = i - 1: ip = i + 1
            jm = j - 1: jp = j + 1
            km = k - 1: kp = k + 1
            If i = 1 Then im = i
            If i = nx Then ip = nx
```

```
            If j = 1 Then jm = j
            If j = ny Then jp = ny

            If k = 1 Then km = k
            If k = nz Then kp = nz

            dx(i, j, k) = (xD(i, j, k) - xD(im, j, k)) / 2 + (xD(ip, j,
k) - xD(i, j, k)) / 2

            If ny = 1 Then
                dy(i, j, k) = yD(i, j, k)
            Else
                dy(i, j, k) = (yD(i, j, k) - yD(i, jm, k)) / 2 + (yD(i,
jp, k) - yD(i, j, k)) / 2
            End If

            If nz = 1 Then
                dz(i, j, k) = zD(i, j, k)
            Else
                dz(i, j, k) = (zD(i, j, k) - zD(i, j, km)) / 2 + (zD(i,
j, kp) - zD(i, j, k)) / 2
            End If
        Next
    Next
Next

End Sub


Sub InterpolaPVT()
Dim i As Integer, j As Integer, k As Integer
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            BWI(i, j, k) = Interpolation(pn(i, j, k), Ppvt(), BW())
            BGI(i, j, k) = Interpolation(pn(i, j, k), Ppvt(), BG())
```

```
            UWI(i, j, k) = Interpolation(pn(i, j, k), Ppvt(), UW())
            UGI(i, j, k) = Interpolation(pn(i, j, k), Ppvt(), UG())
            CWI(i, j, k) = Interpolation(pn(i, j, k), Ppvt(), CW())
            CGI(i, j, k) = Interpolation(pn(i, j, k), Ppvt(), CG())
        Next
    Next
Next
End Sub


Sub InterpolaKr()
Dim i As Integer, j As Integer, k As Integer
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            KRWI(i, j, k) = Interpolation(SWI(i, j, k), SW(), KRW())
            KRGI(i, j, k) = Interpolation(SGI(i, j, k), SG(), KRG())
        Next
    Next
Next
End Sub


Sub Chord_slope()
Dim i As Integer, j As Integer, k As Integer
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            PV1(i, j, k) = PV(i, j, k) * (1 + Cf * (p(i, j, k) - pn(i,
j, k)))
            BWI1(i, j, k) = BWI(i, j, k) * (1 - CWI(i, j, k) * (p(i, j,
k) - pn(i, j, k)))
            BGI1(i, j, k) = BGI(i, j, k) * (1 - CGI(i, j, k) * (p(i, j,
k) - pn(i, j, k)))
        Next
    Next
```

```
Next
End Sub


Sub Saturations()
Dim i As Integer, j As Integer, k As Integer
Dim dp1() As Double, dp2() As Double, dp3() As Double, dp4() As Double,
_
    dp5() As Double, dp6() As Double
ReDim dp1(nx, ny, nz), dp2(nx, ny, nz), dp3(nx, ny, nz), dp4(nx, ny,
nz), dp5(nx, ny, nz), dp6(nx, ny, nz)
Call Chord_slope
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            If i <> 1 Then dp1(i, j, k) = p(i - 1, j, k) - p(i, j, k)
            If i <> nx Then dp2(i, j, k) = p(i + 1, j, k) - p(i, j, k)
            If j <> 1 Then dp3(i, j, k) = p(i, j - 1, k) - p(i, j, k)
            If j <> ny Then dp4(i, j, k) = p(i, j + 1, k) - p(i, j, k)
            If k <> 1 Then dp5(i, j, k) = p(i, j, k - 1) - p(i, j, k)
            If k <> nz Then dp6(i, j, k) = p(i, j, k + 1) - p(i, j, k)

            SWI1(i, j, k) = BWI1(i, j, k) / PV1(i, j, k) * (dt1 *
(aww(i, j, k) * dp1(i, j, k) + _
                            awe(i, j, k) * dp2(i, j, k) + aws(i, j, k)
* dp3(i, j, k) + awn(i, j, k) * dp4(i, j, k) + _
                            awt(i, j, k) * dp5(i, j, k) + awb(i, j, k)
* dp6(i, j, k) - QW(i, j, k)) + PV(i, j, k) * SWI(i, j, k) / BWI(i, j,
k))

            SGI1(i, j, k) = BGI1(i, j, k) / PV1(i, j, k) * (dt1 *
(agw(i, j, k) * dp1(i, j, k) + _
                            age(i, j, k) * dp2(i, j, k) + ags(i, j, k)
* dp3(i, j, k) + agn(i, j, k) * dp4(i, j, k) + _
```

```
                            agt(i, j, k) * dp5(i, j, k) + agb(i, j, k)
* dp6(i, j, k) - QG(i, j, k)) + PV(i, j, k) * SGI(i, j, k) / BGI(i, j,
k))

        Next
    Next
Next
Call Saturations_check
End Sub


Sub Saturations_check()
Dim i As Integer, j As Integer, k As Integer, maxdiff As Double
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            Sat_diff(i, j, k) = Abs(1 - SWI1(i, j, k) - SGI1(i, j, k))
        Next
    Next
Next
maxdiff = MaxValue(Sat_diff())
If maxdiff >= Satmax Then
    Check = False
Else
    Check = True
End If

End Sub


Sub SgUpdate()
Dim i As Integer, j As Integer, k As Integer
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            SGI(i, j, k) = 1 - SWI(i, j, k)
```

```
        Next
    Next
Next
End Sub


Sub Comp_Total()
'Calculate total compressibility
Dim i As Integer, j As Integer, k As Integer
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            CTOT(i, j, k) = Cf + CWI(i, j, k) * SWI(i, j, k) + CGI(i,
j, k) * SGI(i, j, k)
        Next
    Next
Next
End Sub


Sub Mobilities()
Dim i As Integer, j As Integer, k As Integer
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            LambdaW(i, j, k) = KRWI(i, j, k) / (UWI(i, j, k) * BWI(i,
j, k))
            LambdaG(i, j, k) = KRGI(i, j, k) / (UGI(i, j, k) * BGI(i,
j, k))
            LambdaT(i, j, k) = LambdaW(i, j, k) * BWI1(i, j, k) +
LambdaG(i, j, k) * BGI1(i, j, k)
        Next
    Next
Next
End Sub
```

```
Sub Kr_upstream(a As Integer, b As Integer, c As Integer, d As Integer,
e As Integer, f As Integer)
If pn(a, b, c) >= pn(d, e, f) Then
    KRWUPS = KRWI(a, b, c): KRGUPS = KRGI(a, b, c)
Else
    KRWUPS = KRWI(d, e, f): KRGUPS = KRGI(d, e, f)
End If
End Sub


Sub Rates_con_rate()
Dim i As Integer, j As Integer, k As Integer, m As Integer
For m = 1 To nwell
    For k = 1 To nz
     For j = 1 To ny
        For i = 1 To nx
                Select Case TypeWell(i, j, k)

                    Case "PROD"
                           Select Case WellConstraint(i, j, k)
                                  Case "WRate"
                                     QG(i, j, k) = QW(i, j, k) *
LambdaG(i, j, k) / LambdaW(i, j, k)                 'SCF/d
                                     QT(i, j, k) = QW(i, j, k) *
BWI1(i, j, k) + BGI1(i, j, k) * QG(i, j, k)         'rcf/d
                                  Case "GRate"
                                     QW(i, j, k) = QG(i, j, k) *
LambdaW(i, j, k) / LambdaG(i, j, k)                 'SCF/d
                                     QT(i, j, k) = QW(i, j, k) *
BWI1(i, j, k) + BGI1(i, j, k) * QG(i, j, k)         'rcf/d
                                  Case "2Rate"
                                     QT(i, j, k) = QW(i, j, k) *
BWI1(i, j, k) + BGI1(i, j, k) * QG(i, j, k)         'rcf/d
                           End Select

                    Case "WINJ"
```

```
                        Select Case WellConstraint(i, j, k)
                               Case "WRate"
                                  QT(i, j, k) = QW(i, j, k) *
BWI1(i, j, k)      'rcf/d
                               Case "GRate"
                                  QT(i, j, k) = QG(i, j, k) *
BGI1(i, j, k)      'rcf/d
                        End Select

               End Select
          Next
      Next
    Next
Next

End Sub


Sub Rates_con_bhp()
Dim i As Integer, j As Integer, k As Integer, m As Integer
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            Select Case TypeWell(i, j, k)

               Case "PROD"
                   Select Case WellConstraint(i, j, k)
                       Case "Pressure"
                           QW(i, j, k) = JMODEL(i, j, k) * LambdaW(i,
j, k) * (p(i, j, k) - Pwf(i, j, k)) 'SCF/day
                           QG(i, j, k) = JMODEL(i, j, k) * LambdaG(i,
j, k) * (p(i, j, k) - Pwf(i, j, k)) 'SCF/day
                           QT(i, j, k) = QW(i, j, k) * BWI1(i, j, k) +
BGI1(i, j, k) * QG(i, j, k)        'rcf/day
                       Case "WRate", "GRate", "2Rate"
```

```
                        Pwf(i, j, k) = p(i, j, k) - QT(i, j, k) /
(JMODEL(i, j, k) * LambdaT(i, j, k))
                    End Select

                Case "WINJ"
                    Select Case WellConstraint(i, j, k)
                        Case "Pressure"
                            QW(i, j, k) = JMODEL(i, j, k) * LambdaW(i,
j, k) * (p(i, j, k) - Pwf(i, j, k))   'SCF/day
                            QT(i, j, k) = QW(i, j, k) * BWI1(i, j, k)
                        Case "WRate"
                            Pwf(i, j, k) = p(i, j, k) + QT(i, j, k) /
(JMODEL(i, j, k) * LambdaT(i, j, k) * BWI1(i, j, k))
                    End Select

            End Select
        Next
    Next
Next

End Sub


Sub Check_WellConstraints()
'Check whether the minimum constraints are reached, if so change the
constraints
Dim i As Integer, j As Integer, k As Integer
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            Select Case WellConstraint(i, j, k)
                Case "WRate", "GRate", "2Rate"
                    If Pwf(i, j, k) <= MinBHP(i, j, k) Then
                        WellConstraint(i, j, k) = "Pressure"
                        Pwf(i, j, k) = MinBHP(i, j, k)
                    End If
```

```
            End Select
        Next
    Next
Next
End Sub


Sub Cum_production()
Dim i As Integer, j As Integer, k As Integer, m As Integer
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            CumWater = CumWater + QW(i, j, k) * dt1
            CumGas = CumGas + QG(i, j, k) * dt1
        Next
    Next
Next

End Sub


Sub Trans()
'Calculates transmisibilities for each phase
Dim i As Integer, j As Integer, k As Integer
Dim kavw As Double, kave As Double
Dim kavn As Double, kavs As Double
Dim kavt As Double, kavb As Double
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            If i <> 1 Then
                kavw = HarmAvg(permX(i, j, k), permX(i - 1, j, k))
                TW(i, j, k) = 0.00633 * kavw * dy(i, j, k) * dz(i, j,
k) / (xD(i, j, k) - xD(i - 1, j, k))
            End If
            If j <> 1 Then
```

```
                kavs = HarmAvg(permY(i, j, k), permY(i, j - 1, k))
                TS(i, j, k) = 0.00633 * kavs * dx(i, j, k) * dz(i, j,
k) / (yD(i, j, k) - yD(i, j - 1, k))

            End If
            If k <> 1 Then
                kavt = HarmAvg(permZ(i, j, k), permZ(i, j, k - 1))
                TT(i, j, k) = 0.00633 * kavt * dy(i, j, k) * dx(i, j,
k) / (zD(i, j, k) - zD(i, j, k - 1))
            End If
            If i <> nx Then
                kave = HarmAvg(permX(i, j, k), permX(i + 1, j, k))
                TE(i, j, k) = 0.00633 * kave * dy(i, j, k) * dz(i, j,
k) / (xD(i + 1, j, k) - xD(i, j, k))
            End If
            If j <> ny Then
                kavn = HarmAvg(permY(i, j, k), permY(i, j + 1, k))
                TN(i, j, k) = 0.00633 * kavn * dx(i, j, k) * dz(i, j,
k) / (yD(i, j + 1, k) - yD(i, j, k))
            End If
            If k <> nz Then
                kavb = HarmAvg(permZ(i, j, k), permZ(i, j, k + 1))
                TB(i, j, k) = 0.00633 * kavb * dy(i, j, k) * dx(i, j,
k) / (zD(i, j, k + 1) - zD(i, j, k))
            End If

        Next
    Next
Next
End Sub


Sub Avg_PVT()
Dim i As Integer, j As Integer, k As Integer

For k = 1 To nz
```

```
    For j = 1 To ny
        For i = 1 To nx
            If i <> 1 Then
                UWavw(i, j, k) = AritAvg(UWI(i, j, k), UWI(i - 1, j,
k))
                UGavw(i, j, k) = AritAvg(UGI(i, j, k), UGI(i - 1, j,
k))
                BWavw(i, j, k) = AritAvg(BWI(i, j, k), BWI(i - 1, j,
k))
                BGavw(i, j, k) = AritAvg(BGI(i, j, k), BGI(i - 1, j,
k))
            End If
            If j <> 1 Then
                UWavs(i, j, k) = AritAvg(UWI(i, j, k), UWI(i, j - 1,
k))
                UGavs(i, j, k) = AritAvg(UGI(i, j, k), UGI(i, j - 1,
k))
                BWavs(i, j, k) = AritAvg(BWI(i, j, k), BWI(i, j - 1,
k))
                BGavs(i, j, k) = AritAvg(BGI(i, j, k), BGI(i, j - 1,
k))
            End If
            If k <> 1 Then
                UWavt(i, j, k) = AritAvg(UWI(i, j, k), UWI(i, j, k -
1))
                UGavt(i, j, k) = AritAvg(UGI(i, j, k), UGI(i, j, k -
1))
                BWavt(i, j, k) = AritAvg(BWI(i, j, k), BWI(i, j, k -
1))
                BGavt(i, j, k) = AritAvg(BGI(i, j, k), BGI(i, j, k -
1))
            End If
            If i <> nx Then
                UWave(i, j, k) = AritAvg(UWI(i, j, k), UWI(i + 1, j,
k))
```

```
                UGave(i, j, k) = AritAvg(UGI(i, j, k), UGI(i + 1, j,
k))
                BWave(i, j, k) = AritAvg(BWI(i, j, k), BWI(i + 1, j,
k))
                BGave(i, j, k) = AritAvg(BGI(i, j, k), BGI(i + 1, j,
k))
            End If
            If j <> ny Then
                UWavn(i, j, k) = AritAvg(UWI(i, j, k), UWI(i, j + 1,
k))
                UGavn(i, j, k) = AritAvg(UGI(i, j, k), UGI(i, j + 1,
k))
                BWavn(i, j, k) = AritAvg(BWI(i, j, k), BWI(i, j + 1,
k))
                BGavn(i, j, k) = AritAvg(BGI(i, j, k), BGI(i, j + 1,
k))
            End If
            If k <> nz Then
                UWavb(i, j, k) = AritAvg(UWI(i, j, k), UWI(i, j, k +
1))
                UGavb(i, j, k) = AritAvg(UGI(i, j, k), UGI(i, j, k +
1))
                BWavb(i, j, k) = AritAvg(BWI(i, j, k), BWI(i, j, k +
1))
                BGavb(i, j, k) = AritAvg(BGI(i, j, k), BGI(i, j, k +
1))
            End If
        Next
    Next
Next

End Sub


Sub MatrixA()
Dim i As Integer, j As Integer, k As Integer, MH As Integer
```

```
MH = 0#
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            MH = MH + 1

            If i <> 1 Then
                Call Kr_upstream(i, j, k, i - 1, j, k)
                aww(i, j, k) = TW(i, j, k) * KRWUPS / (UWavw(i, j, k) *
BWavw(i, j, k))
                agw(i, j, k) = TW(i, j, k) * KRGUPS / (UGavw(i, j, k) *
BGavw(i, j, k))
                aW(MH) = aww(i, j, k) * BWI1(i, j, k) + agw(i, j, k) *
BGI1(i, j, k)
            End If
            If j <> 1 Then
                Call Kr_upstream(i, j, k, i, j - 1, k)
                aws(i, j, k) = TS(i, j, k) * KRWUPS / (UWavs(i, j, k) *
BWavs(i, j, k))
                ags(i, j, k) = TS(i, j, k) * KRGUPS / (UGavs(i, j, k) *
BGavs(i, j, k))
                aSt(MH) = aws(i, j, k) * BWI1(i, j, k) + ags(i, j, k) *
BGI1(i, j, k)
            End If
            If k <> 1 Then
                Call Kr_upstream(i, j, k, i, j, k - 1)
                awt(i, j, k) = TT(i, j, k) * KRWUPS / (UWavt(i, j, k) *
BWavt(i, j, k))
                agt(i, j, k) = TT(i, j, k) * KRGUPS / (UGavt(i, j, k) *
BGavt(i, j, k))
                aT(MH) = awt(i, j, k) * BWI1(i, j, k) + agt(i, j, k) *
BGI1(i, j, k)
            End If

            If i <> nx Then
                Call Kr_upstream(i, j, k, i + 1, j, k)
```

```
                awe(i, j, k) = TE(i, j, k) * KRWUPS / (UWave(i, j, k) *
BWave(i, j, k))
                age(i, j, k) = TE(i, j, k) * KRGUPS / (UGave(i, j, k) *
BGave(i, j, k))
                aE(MH) = awe(i, j, k) * BWI1(i, j, k) + age(i, j, k) *
BGI1(i, j, k)
            End If
            If j <> ny Then
                Call Kr_upstream(i, j, k, i, j + 1, k)
                awn(i, j, k) = TN(i, j, k) * KRWUPS / (UWavn(i, j, k) *
BWavn(i, j, k))
                agn(i, j, k) = TN(i, j, k) * KRGUPS / (UGavn(i, j, k) *
BGavn(i, j, k))
                aN(MH) = awn(i, j, k) * BWI1(i, j, k) + agn(i, j, k) *
BGI1(i, j, k)
            End If
            If k <> nz Then
                Call Kr_upstream(i, j, k, i, j, k + 1)
                awb(i, j, k) = TB(i, j, k) * KRWUPS / (UWavb(i, j, k) *
BWavb(i, j, k))
                agb(i, j, k) = TB(i, j, k) * KRGUPS / (UGavb(i, j, k) *
BGavb(i, j, k))
                aB(MH) = awb(i, j, k) * BWI1(i, j, k) + agb(i, j, k) *
BGI1(i, j, k)
            End If

            Select Case WellConstraint(i, j, k)
                Case "GRate", "WRate", "2Rate", ""
                    ac(MH) = -aW(MH) - aSt(MH) - aB(MH) - aE(MH) -
aN(MH) - aT(MH) - betha(i, j, k)
                Case "Pressure"
                    ac(MH) = -aW(MH) - aSt(MH) - aB(MH) - aE(MH) -
aN(MH) - aT(MH) - betha(i, j, k) - JMODEL(i, j, k) * LambdaT(i, j, k)
            End Select

If i = 1 Then
```

```
            Select Case BCW
                Case 0
'                    Keep the same values
                Case 1
                    ac(MH) = 1
                    awe(i, j, k) = 0: age(i, j, k) = 0: aE(MH) = 0
                    awn(i, j, k) = 0: agn(i, j, k) = 0: aN(MH) = 0
                    aws(i, j, k) = 0: ags(i, j, k) = 0: aSt(MH) = 0
                    awt(i, j, k) = 0: agt(i, j, k) = 0: aT(MH) = 0
                    awb(i, j, k) = 0: agb(i, j, k) = 0: aB(MH) = 0
            End Select
        End If

        If j = 1 Then
            Select Case BCS
                Case 0
'                    Keep the same values
                Case 1
                    ac(MH) = 1
                    awn(i, j, k) = 0: agn(i, j, k) = 0: aN(MH) = 0
                    aww(i, j, k) = 0: agw(i, j, k) = 0: aW(MH) = 0
                    awe(i, j, k) = 0: age(i, j, k) = 0: aE(MH) = 0
                    awt(i, j, k) = 0: agt(i, j, k) = 0: aT(MH) = 0
                    awb(i, j, k) = 0: agb(i, j, k) = 0: aB(MH) = 0
            End Select
        End If

        If k = 1 Then
            Select Case BCT
                Case 0
'                    Keep the same values
                Case 1
                    ac(MH) = 1
                    awb(i, j, k) = 0: agb(i, j, k) = 0: aB(MH) = 0
                    aww(i, j, k) = 0: agw(i, j, k) = 0: aW(MH) = 0
```

```
                awe(i, j, k) = 0: age(i, j, k) = 0: aE(MH) = 0
                awn(i, j, k) = 0: agn(i, j, k) = 0: aN(MH) = 0
                aws(i, j, k) = 0: ags(i, j, k) = 0: aSt(MH) = 0
            End Select
        End If

        If i = nx Then
            Select Case BCE
                Case 0
'                    Keep the same values
                Case 1
                    ac(MH) = 1
                    aww(i, j, k) = 0: agw(i, j, k) = 0: aW(MH) = 0
                    awn(i, j, k) = 0: agn(i, j, k) = 0: aN(MH) = 0
                    aws(i, j, k) = 0: ags(i, j, k) = 0: aSt(MH) = 0
                    awt(i, j, k) = 0: agt(i, j, k) = 0: aT(MH) = 0
                    awb(i, j, k) = 0: agb(i, j, k) = 0: aB(MH) = 0
            End Select
        End If
        If j = ny Then
            Select Case BCN
                Case 0
'                    Keep the same values
                Case 1
                    ac(MH) = 1
                    aws(i, j, k) = 0: ags(i, j, k) = 0: aSt(MH) = 0
                    aww(i, j, k) = 0: agw(i, j, k) = 0: aW(MH) = 0
                    awe(i, j, k) = 0: age(i, j, k) = 0: aE(MH) = 0
                    awt(i, j, k) = 0: agt(i, j, k) = 0: aT(MH) = 0
                    awb(i, j, k) = 0: agb(i, j, k) = 0: aB(MH) = 0
            End Select
        End If

        If k = nz Then
            Select Case BCB
                Case 0
```

```
'                       Keep the same values
                    Case 1
                        ac(MH) = 1
                        awt(i, j, k) = 0: agt(i, j, k) = 0: aT(MH) = 0
                        aww(i, j, k) = 0: agw(i, j, k) = 0: aW(MH) = 0
                        awe(i, j, k) = 0: age(i, j, k) = 0: aE(MH) = 0
                        awn(i, j, k) = 0: agn(i, j, k) = 0: aN(MH) = 0
                        aws(i, j, k) = 0: ags(i, j, k) = 0: aSt(MH) = 0
                End Select
            End If

        Next i
    Next j
Next k

End Sub


Sub MatrixB()
Dim i As Integer, j As Integer, k As Integer, MH As Integer
MH = 0#
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            MH = MH + 1
            betha(i, j, k) = PV(i, j, k) * CTOT(i, j, k) / dt1
            'Well Constraints
            Select Case WellConstraint(i, j, k)
                Case "WRate", "GRate", "2rate"
                    MB(MH) = -betha(i, j, k) * pn(i, j, k) + QT(i, j,
k)
                Case "Pressure"
                    MB(MH) = -betha(i, j, k) * pn(i, j, k) - JMODEL(i,
j, k) * LambdaT(i, j, k) * Pwf(i, j, k)
                Case ""
                    MB(MH) = -betha(i, j, k) * pn(i, j, k)
```

```
            End Select

            'Boundary conditions
            If i = 1 Then
                Select Case BCW
                    Case 0
'                        Keep the same values
                    Case 1
                        MB(MH) = Pinit
                End Select
            End If
            If j = 1 Then
                Select Case BCS
                    Case 0
'                        Keep the same values
                    Case 1
                        MB(MH) = Pinit
                End Select
            End If
            If k = 1 Then
                Select Case BCT
                    Case 0
'                        Keep the same values
                    Case 1
                        MB(MH) = Pinit
                End Select
            End If

            If i = nx Then
                Select Case BCE
                    Case 0
'                        Keep the same values
                    Case 1
                        MB(MH) = Pinit
                End Select
            End If
```

```
            If j = ny Then
                Select Case BCN
                    Case 0
'                       Keep the same values
                    Case 1
                        MB(MH) = Pinit
                End Select
            End If
            If k = nz Then
                Select Case BCB
                    Case 0
'                       Keep the same values
                    Case 1
                        MB(MH) = Pinit
                End Select
            End If

        Next
    Next
Next

End Sub


Sub MemAlloc()
'Allocate memory for variables after reading data
    ReDim p(nx, ny, nz), pn(nx, ny, nz)
    ReDim BOI(nx, ny, nz), BWI(nx, ny, nz), BGI(nx, ny, nz), RSOI(nx,
ny, nz), BOI1(nx, ny, nz), BWI1(nx, ny, nz), _
          BGI1(nx, ny, nz), RSOI1(nx, ny, nz)
    ReDim UOI(nx, ny, nz), UWI(nx, ny, nz), UGI(nx, ny, nz)
    ReDim COI(nx, ny, nz), CWI(nx, ny, nz), CGI(nx, ny, nz), CTOT(nx,
ny, nz)
    ReDim UOavw(nx, ny, nz), UWavw(nx, ny, nz), UGavw(nx, ny, nz), _
          UOavs(nx, ny, nz), UWavs(nx, ny, nz), UGavs(nx, ny, nz), _
          UOavb(nx, ny, nz), UWavb(nx, ny, nz), UGavb(nx, ny, nz)
```

```
    ReDim BOavw(nx, ny, nz), BWavw(nx, ny, nz), BGavw(nx, ny, nz), _
          BOavs(nx, ny, nz), BWavs(nx, ny, nz), BGavs(nx, ny, nz), _
          BOavb(nx, ny, nz), BWavb(nx, ny, nz), BGavb(nx, ny, nz)
    ReDim UOave(nx, ny, nz), UWave(nx, ny, nz), UGave(nx, ny, nz), _
          UOavn(nx, ny, nz), UWavn(nx, ny, nz), UGavn(nx, ny, nz), _
          UOavt(nx, ny, nz), UWavt(nx, ny, nz), UGavt(nx, ny, nz)
    ReDim BOave(nx, ny, nz), BWave(nx, ny, nz), BGave(nx, ny, nz), _
          BOavn(nx, ny, nz), BWavn(nx, ny, nz), BGavn(nx, ny, nz), _
          BOavt(nx, ny, nz), BWavt(nx, ny, nz), BGavt(nx, ny, nz)
    ReDim RSOavw(nx, ny, nz), RSOave(nx, ny, nz), _
          RSOavs(nx, ny, nz), RSOavn(nx, ny, nz), _
          RSOavb(nx, ny, nz), RSOavt(nx, ny, nz)
    ReDim KRWI(nx, ny, nz), KROWI(nx, ny, nz), KRGI(nx, ny, nz),
KROGI(nx, ny, nz), KROI(nx, ny, nz)
    ReDim SOI(nx, ny, nz), SWI(nx, ny, nz), SGI(nx, ny, nz), SLI(nx,
ny, nz)
    ReDim SOI1(nx, ny, nz), SWI1(nx, ny, nz), SGI1(nx, ny, nz),
SLI1(nx, ny, nz)
    ReDim PV(nx, ny, nz), PV1(nx, ny, nz), OOIP(nx, ny, nz), OWIP(nx,
ny, nz), OGIP(nx, ny, nz)
    ReDim TypeWell(nx, ny, nz), WellConstraint(nx, ny, nz), rw(nx, ny,
nz), Skin(nx, ny, nz), Pwf(nx, ny, nz)
    ReDim MinBHP(nx, ny, nz), QO(nx, ny, nz), QW(nx, ny, nz), QG(nx,
ny, nz)
    ReDim QT(nx, ny, nz), JMODEL(nx, ny, nz), ro(nx, ny, nz), dx(nx,
ny, nz), dy(nx, ny, nz), dz(nx, ny, nz)
    ReDim LambdaO(nx, ny, nz), LambdaW(nx, ny, nz), LambdaG(nx, ny,
nz), LambdaT(nx, ny, nz)
    ReDim TW(nx, ny, nz), TE(nx, ny, nz), TN(nx, ny, nz), TS(nx, ny,
nz), TT(nx, ny, nz), TB(nx, ny, nz), _
          aow(nx, ny, nz), aww(nx, ny, nz), agw(nx, ny, nz), aoe(nx,
ny, nz), awe(nx, ny, nz), age(nx, ny, nz), _
          aon(nx, ny, nz), awn(nx, ny, nz), agn(nx, ny, nz), aos(nx,
ny, nz), aws(nx, ny, nz), ags(nx, ny, nz), _
          aot(nx, ny, nz), awt(nx, ny, nz), agt(nx, ny, nz), aob(nx,
ny, nz), awb(nx, ny, nz), agb(nx, ny, nz)
```

```
    ReDim betha(nx, ny, nz) As Double
    ReDim Sat_diff(nx, ny, nz) As Double

End Sub


Sub Matrix_Solver()
Dim i As Integer, j As Integer, k As Integer, MH As Integer

Call CMAT(aW(), aE(), aSt(), aN(), aT(), aB(), ac(), MB(), TOL, II, JJ,
KKK, IJKM, ITMAX, QI(), AQI(), _
         AL3(), AL2(), AL1(), AD(), AU1(), AU2(), AU3(), QN(), AQN(),
RN(), DXN(), ADX(), Psim(), IT)

'Update pressure at i,j,k coordinates
MH = 0
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            MH = MH + 1
            p(i, j, k) = Psim(MH)
        Next
    Next
Next
End Sub
Sub AllocateMemory_Matrix(ByVal IJKM As Long)
ReDim aW(1 To IJKM)
ReDim aE(1 To IJKM)
ReDim aSt(1 To IJKM)
ReDim aN(1 To IJKM)
ReDim aT(1 To IJKM)
ReDim aB(1 To IJKM)
ReDim ac(1 To IJKM)
ReDim MB(1 To IJKM)
ReDim r(1 To IJKM)
ReDim Psim(1 To IJKM)
```

```
ReDim AL3(1 To IJKM)
ReDim AL2(1 To IJKM)
ReDim AL1(1 To IJKM)
ReDim AD(1 To IJKM)
ReDim AU1(1 To IJKM)
ReDim AU2(1 To IJKM)
ReDim AU3(1 To IJKM)

ReDim QI(1 To 15, 1 To IJKM)
ReDim AQI(1 To 15, 1 To IJKM)
ReDim QN(1 To IJKM)
ReDim AQN(1 To IJKM)

ReDim RN(1 To IJKM)
ReDim DXN(1 To IJKM)
ReDim ADX(1 To IJKM)

End Sub


Sub Matrix_Initial()
'Orthomin Matrix Solver's Parameter
   II = CInt(nx)
   JJ = CInt(ny)
   KKK = CInt(nz)

   IJKM = CInt(II)
   IJKM = IJKM * CInt(JJ)
   IJKM = IJKM * CInt(KKK)
   ITMAX = 50
   Call AllocateMemory_Matrix(IJKM)
   TOL = 0.000001
End Sub
```

```
Sub CMAT(aW() As Double, aE() As Double, aSt() As Double, aN() As
Double, _
         aT() As Double, aB() As Double, ac() As Double, MB() As
Double, _
         TOL As Double, II As Integer, JJ As Integer, KKK As Integer, _
         IJKM As Long, ITMAX As Double, QI() As Double, AQI() As
Double, _
         AL3() As Double, AL2() As Double, AL1() As Double, _
         AD() As Double, AU1() As Double, AU2() As Double, _
         AU3() As Double, QN() As Double, AQN() As Double, RN() As
Double, _
         DXN() As Double, ADX() As Double, Psim() As Double, IT As
Long)


Dim INX As Integer
Dim i As Integer, j As Integer, k As Integer
Dim FAC As Double
Dim TERM As Double
Dim INXY As Integer
Dim IB As Long

'     ORTHOMIN SPARSE MATRIX SOLVER BASED ON PAPER BY P. K. W. VINSOME
'     FOUTH SYMPOSIUM ON RESERVOIR SIMULATION
'     LOS ANGELES, CALIFORNIA     FEBRUARY 19-20,1976

      INX = II
      INXY = II * JJ
      IB = 0
       For k = 1 To KKK
        For j = 1 To JJ
          For i = 1 To II
            IB = IB + 1
            FAC = 1# / ac(IB)
            If i <> 1 Then AL1(IB) = FAC * aW(IB)
            If i <> II Then AU1(IB) = FAC * aE(IB)
```

```
            If j <> 1 Then AL2(IB) = FAC * aSt(IB)
            If j <> JJ Then AU2(IB) = FAC * aN(IB)
            If k <> 1 Then AL3(IB) = FAC * aT(IB)
            If k <> KKK Then AU3(IB) = FAC * aB(IB)
            RN(IB) = FAC * MB(IB)
         Next i
       Next j
      Next k

'     APPROXIMATE LDU FACTORIZATION

      AD(1) = 1#
      For i = 2 To INX
        TERM = 1# - AL1(i) * AD(i - 1) * AU1(i - 1)
        AD(i) = 1# / TERM
      Next i

      For i = INX + 1 To INXY
        TERM = 1# - AL1(i) * AD(i - 1) * AU1(i - 1) _
                  - AL2(i) * AD(i - INX) * AU2(i - INX)
        AD(i) = 1# / TERM
      Next i
      For i = INXY + 1 To IJKM
        TERM = 1# - AL1(i) * AD(i - 1) * AU1(i - 1) _
                  - AL2(i) * AD(i - INX) * AU2(i - INX) _
                  - AL3(i) * AD(i - INXY) * AU3(i - INXY)
        AD(i) = 1# / TERM
      Next
      Call ORTH(AL3(), AL2(), AL1(), AD(), AU1(), AU2(), AU3(), TOL _
                , INX, INXY, IJKM, ITMAX, RN(), DXN(), ADX(), QI() _
                , AQI(), QN(), AQN(), Psim(), IT)


End Sub
```

```
Sub MVEC(AL3() As Double, AL2() As Double, AL1() As Double, AU1() As
Double, AU2() As Double, AU3() As Double, r() As Double, _
         INX As Integer, INXY As Integer, IJKM As Long, c() As Double)

Dim i As Long

      For i = 1 To IJKM
        c(i) = r(i)
      Next i

      For i = 1 To IJKM - 1
        c(i) = c(i) + AU1(i) * r(i + 1)
      Next i
      For i = 1 To IJKM - INX
        c(i) = c(i) + AU2(i) * r(i + INX)
      Next i
      For i = 1 To IJKM - INXY
        c(i) = c(i) + AU3(i) * r(i + INXY)
      Next i
      For i = 2 To IJKM
        c(i) = c(i) + AL1(i) * r(i - 1)
      Next i
      For i = INX + 1 To IJKM
        c(i) = c(i) + AL2(i) * r(i - INX)
      Next i
      For i = INXY + 1 To IJKM
        c(i) = c(i) + AL3(i) * r(i - INXY)
      Next i

   End Sub


Sub ORTH(AL3() As Double, AL2() As Double, AL1() As Double, _
         AD() As Double, AU1() As Double, AU2() As Double, _
         AU3() As Double, TOL, INX As Integer, INXY As Integer, _
```

```
         IJKM As Long, ITMAX As Double, RN() As Double, DXN() As
Double, _
         ADX() As Double, QI() As Double, AQI() As Double, _
         QN() As Double, AQN() As Double, DP() As Double, IT)

Dim Rsq As Double
Dim nmax As Long
Dim N As Long
Dim CONV As Double, CONV1 As Double
Dim NM1 As Long
Dim IB As Long
Dim ITER As Long
Dim i As Long
Dim omega As Double
Dim AI As Double


Dim AQIAQI() As Double

Dim AQIADX As Double
Dim AQNAQN As Double
Dim AQNRN As Double

ReDim AQIAQI(IJKM)

' ======== temp
     nmax = 15
' ======== temp

      CONV1 = TOL * TOL
      If CONV1 > 0.0001 Then CONV1 = 0.0001
      Rsq = 0#
      For IB = 1 To IJKM
        DP(IB) = 0#
        Rsq = Rsq + RN(IB) * RN(IB)
      Next IB
```

```
CONV = CONV1 * Rsq
N = 0
For ITER = 1 To ITMAX
  IT = ITER
  If N = nmax Then N = 0
  N = N + 1
  NM1 = N - 1

  Call MSOLVE(AL3(), AL2(), AL1(), AD(), AU1(), AU2() _
              , AU3(), RN(), INX, INXY, IJKM, DXN())

  Call MVEC(AL3(), AL2(), AL1(), AU1(), AU2(), AU3(), DXN(), _
            INX, INXY, IJKM, ADX())

  If N = 1 Then
    For IB = 1 To IJKM
      QN(IB) = DXN(IB)
      AQN(IB) = ADX(IB)
      QI(1, IB) = QN(IB)
      AQI(1, IB) = AQN(IB)
    Next IB
  Else
    For IB = 1 To IJKM
      QN(IB) = DXN(IB)
    Next IB
    For i = 1 To NM1
      AQIADX = 0#
      For IB = 1 To IJKM
        AQIADX = AQIADX + AQI(i, IB) * ADX(IB)
      Next IB



      AI = AQIADX / AQIAQI(i)
      For IB = 1 To IJKM
        QN(IB) = QN(IB) - AI * QI(i, IB)
```

```
            Next IB
          Next i

          Call MVEC(AL3(), AL2(), AL1(), AU1(), AU2(), _
                    AU3(), QN(), INX, INXY, IJKM, AQN())

          For IB = 1 To IJKM
            QI(N, IB) = QN(IB)
            AQI(N, IB) = AQN(IB)
          Next IB
        End If
        AQNAQN = 0#
        AQNRN = 0#
        For IB = 1 To IJKM
          AQNAQN = AQNAQN + AQN(IB) * AQN(IB)
          AQNRN = AQNRN + AQN(IB) * RN(IB)
        Next IB

        AQIAQI(N) = AQNAQN
        omega = AQNRN / AQNAQN
        Rsq = 0#
        For IB = 1 To IJKM
          DP(IB) = DP(IB) + omega * QN(IB)
          RN(IB) = RN(IB) - omega * AQN(IB)
          Rsq = Rsq + RN(IB) * RN(IB)
        Next IB
         If (Rsq <= CONV) Then GoTo line900
   Next ITER

'   MsgBox " ORTHOMIN DID NOT CONVERGE IN " & ITER & " ITERATIONS"

line900:

End Sub
```

```
Sub MSOLVE(AL3() As Double, AL2() As Double, AL1() As Double, _
           AD() As Double, AU1() As Double, AU2() As Double, _
           AU3() As Double, r() As Double, INX As Integer, _
           INXY As Integer, IJKM As Long, XX() As Double)

Dim i As Long

      XX(1) = AD(1) * r(1)
      For i = 2 To INX
        XX(i) = AD(i) * (r(i) - AL1(i) * XX(i - 1))
      Next i
      For i = INX + 1 To INXY
        XX(i) = AD(i) * (r(i) - AL1(i) * XX(i - 1) - AL2(i) * XX(i -
INX))
      Next i
      For i = INXY + 1 To IJKM
        XX(i) = AD(i) * (r(i) - AL1(i) * XX(i - 1) - AL2(i) * XX(i -
INX) _
               - AL3(i) * XX(i - INXY))
      Next i
      For i = 1 To IJKM
        XX(i) = XX(i) / AD(i)
     Next i
'
'     BACK SUBSTITUTION
'
      For i = IJKM - 1 To IJKM - INX + 1 Step -1
        XX(i) = AD(i) * (XX(i) - AU1(i) * XX(i + 1))
      Next i
      For i = IJKM - INX To IJKM - INXY + 1 Step -1
        XX(i) = AD(i) * (XX(i) - AU1(i) * XX(i + 1) - AU2(i) * XX(i +
INX))
      Next i
      For i = IJKM - INXY To 1 Step -1
        XX(i) = AD(i) * (XX(i) - AU1(i) * XX(i + 1) - AU2(i) * XX(i +
INX) _
```

```
                - AU3(i) * XX(i + INXY))
      Next i

End Sub


Sub Read_data()
Dim InputFile As String
Dim OutputFile As String
Dim Tablename As String, text As String, PR() As Double
Dim i As Integer, j As Integer, k As Integer, convar As String
InputFile = "C:\Documents and Settings\SAEED F\My Documents\My Academic
Career\My Research\Mycodes\3D2PH_2.txt"
OutputFile = "C:\Documents and Settings\SAEED F\Desktop\3D2PH_2.out"
Open InputFile For Input As 1
Open OutputFile For Output As 2
Line Input #1, text
Line Input #1, text
Input #1, nx, ny, nz
ReDim xD(nx, ny, nz), yD(nx, ny, nz), zD(nx, ny, nz), xThick(nx, ny,
nz), yThick(nx, ny, nz), zThick(nx, ny, nz)
ReDim poro(nx, ny, nz), permX(nx, ny, nz), permY(nx, ny, nz), permZ(nx,
ny, nz)

'GRID BLOCKS GENERATION

Input #1, text, convar
Select Case convar
    Case "CONST"
        Input #1, Xsize
        xD(1, 1, 1) = Xsize
        For i = 2 To nx
            xD(i, 1, 1) = xD(i - 1, 1, 1) + Xsize
        Next
        For k = 1 To nz
            For j = 1 To ny
```

```
                For i = 1 To nx
                    xD(i, j, k) = xD(i, 1, 1)
                Next
            Next
        Next

    Case "IVARIABLE"
        For i = 1 To nx
            Input #1, xThick(i, 1, 1)
        Next
        xD(1, 1, 1) = xThick(1, 1, 1)
        For i = 2 To nx
            xD(i, 1, 1) = xD(i - 1, 1, 1) + xThick(i, 1, 1)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    xD(i, j, k) = xD(i, 1, 1)
                Next
            Next
        Next
End Select


Input #1, text, convar
Select Case convar
    Case "CONST"
        Input #1, Ysize
        yD(1, 1, 1) = Ysize
        For j = 2 To ny
            yD(1, j, 1) = yD(1, j - 1, 1) + Ysize
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    yD(i, j, k) = yD(1, j, 1)
```

```
                Next
            Next
        Next

    Case "JVARIABLE"
        For j = 1 To ny
            Input #1, yThick(1, j, 1)
        Next
        yD(1, 1, 1) = yThick(1, 1, 1)
        For j = 2 To ny
            yD(1, j, 1) = yD(1, j - 1, 1) + yThick(1, j, 1)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    yD(i, j, k) = yD(1, j, 1)
                Next
            Next
        Next
End Select


Input #1, text, convar
Select Case convar
    Case "CONST"
        Input #1, Zsize
        zD(1, 1, 1) = Zsize
        zThick(1, 1, 1) = Zsize               'This line is for
printing purposes only
        For k = 2 To nz
            zD(1, 1, k) = zD(1, 1, k - 1) + Zsize
            zThick(1, 1, k) = Zsize           'This line is for
printing purposes only
        Next
        For k = 1 To nz
            For j = 1 To ny
```

```
                For i = 1 To nx
                    zD(i, j, k) = zD(1, 1, k)
                Next
            Next
        Next

    Case "KVARIABLE"
        For k = 1 To nz
            Input #1, zThick(1, 1, k)
        Next
        zD(1, 1, 1) = zThick(1, 1, 1)
        For k = 2 To nz
            zD(1, 1, k) = zD(1, 1, k - 1) + zThick(1, 1, k)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    zD(i, j, k) = zD(1, 1, k)
                Next
            Next
        Next
End Select

'POROSITY ENTRIES
Input #1, text, convar
Select Case convar
    Case "CONST"
        Input #1, poro(1, 1, 1)
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    poro(i, j, k) = poro(1, 1, 1)
                Next
            Next
        Next
    Case "IVARIABLE"
```

```
        For i = 1 To nx
            Input #1, poro(i, 1, 1)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    poro(i, j, k) = poro(i, 1, 1)
                Next
            Next
        Next

    Case "JVARIABLE"
        For j = 1 To ny
            Input #1, poro(1, j, 1)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    poro(i, j, k) = poro(1, j, 1)
                Next
            Next
        Next
    Case "KVARIABLE"
        For k = 1 To nz
            Input #1, poro(1, 1, k)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    poro(i, j, k) = poro(1, 1, k)
                Next
            Next
        Next
End Select

'PERMEABILITY ENTRIES
```

```
'I DIRECTION PERMEABILITY
Input #1, text, convar
Select Case convar
    Case "CONST"
        Input #1, permX(1, 1, 1)
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permX(i, j, k) = permX(1, 1, 1)
                Next
            Next
        Next
    Case "IVARIABLE"
        For i = 1 To nx
            Input #1, permX(i, 1, 1)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permX(i, j, k) = permX(i, 1, 1)
                Next
            Next
        Next

    Case "JVARIABLE"
        For j = 1 To ny
            Input #1, permX(1, j, 1)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permX(i, j, k) = permX(1, j, 1)
                Next
            Next
        Next
```

```
    Case "KVARIABLE"
        For k = 1 To nz
            Input #1, permX(1, 1, k)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permX(i, j, k) = permX(1, 1, k)
                Next
            Next
        Next
End Select

'J DIRECTION PERMEABILITY
Input #1, text, convar
Select Case convar
    Case "CONST"
        Input #1, permY(1, 1, 1)
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permY(i, j, k) = permY(1, 1, 1)
                Next
            Next
        Next
    Case "IVARIABLE"
        For i = 1 To nx
            Input #1, permY(i, 1, 1)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permY(i, j, k) = permY(i, 1, 1)
                Next
            Next
        Next
```

```
    Case "JVARIABLE"
        For j = 1 To ny
            Input #1, permY(1, j, 1)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permY(i, j, k) = permY(1, j, 1)
                Next
            Next
        Next

    Case "KVARIABLE"
        For k = 1 To nz
            Input #1, permY(1, 1, k)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permY(i, j, k) = permY(1, 1, k)
                Next
            Next
        Next
    Case "EQUALI"
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permY(i, j, k) = permX(i, j, k)
                Next
            Next
        Next
End Select

'K DIRECTION PERMEABIITY
Input #1, text, convar
```

```
Select Case convar
    Case "CONST"
        Input #1, permZ(1, 1, 1)
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permZ(i, j, k) = permZ(1, 1, 1)
                Next
            Next
        Next
    Case "IVARIABLE"
        For i = 1 To nx
            Input #1, permZ(i, 1, 1)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permZ(i, j, k) = permZ(i, 1, 1)
                Next
            Next
        Next

    Case "JVARIABLE"
        For j = 1 To ny
            Input #1, permZ(1, j, 1)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permZ(i, j, k) = permZ(1, j, 1)
                Next
            Next
        Next

    Case "KVARIABLE"
        For k = 1 To nz
```

```
            Input #1, permZ(1, 1, k)
        Next
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permZ(i, j, k) = permZ(1, 1, k)
                Next
            Next
        Next
    Case "EQUALI"
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permZ(i, j, k) = permX(i, j, k)
                Next
            Next
        Next
    Case "EQUALJ"
        For k = 1 To nz
            For j = 1 To ny
                For i = 1 To nx
                    permZ(i, j, k) = permY(i, j, k)
                Next
            Next
        Next
End Select

Line Input #1, text
Input #1, Cf

'PVT
Line Input #1, Tablename
Input #1, npvt
Line Input #1, text
ReDim Ppvt(npvt) As Double, BW(npvt) As Double, _
        UW(npvt) As Double, CW(npvt) As Double, BG(npvt) As Double, _
```

```
        UG(npvt) As Double, CG(npvt) As Double
For i = 1 To npvt
    Input #1, Ppvt(i), BW(i), UW(i), CW(i), BG(i), UG(i), CG(i)
Next i

'RELATIVE PERM
Line Input #1, text
Line Input #1, text
Input #1, nkr
Line Input #1, text
ReDim SW(nkr), KRW(nkr)
For i = 1 To nkr
    Input #1, SW(i), KRW(i)
Next i
Line Input #1, text
Input #1, nkrg
Line Input #1, text
ReDim SG(nkrg), KRG(nkrg)
For i = 1 To nkrg
    Input #1, SG(i), KRG(i)
Next
Line Input #1, text
Line Input #1, text
Input #1, SWC
Line Input #1, text
Input #1, Pinit
Line Input #1, text
Line Input #1, text
Input #1, BCW, BCE, BCS, BCN, BCB, BCT
Line Input #1, text
Input #1, dt, tmax, Satmax, ncuts
Line Input #1, text
Input #1, nwell
If nwell <> 0 Then
    ReDim Wellname(nwell), XCoor(nwell), YCoor(nwell), ZCoor(nwell),
TypeWell_read(nwell), _
```

```
            PR(nwell), rw_read(nwell), Skin_read(nwell),
QT_read(nwell), _
            QW_read(nwell), QG_read(nwell), Pwf_read(nwell),
MinBHP_read(nwell)
    Line Input #1, text
    For i = 1 To nwell
        Input #1, Wellname(i), XCoor(i), YCoor(i), ZCoor(i),
rw_read(i), Skin_read(i), TypeWell_read(i)
    Next
    Line Input #1, text
    For i = 1 To nwell
        Input #1, PR(i), QW_read(i), QG_read(i), Pwf_read(i),
MinBHP_read(i)
    Next
End If
Close #1
Print #2, "Simulation Output Results"

End Sub


Sub Identify_constraints()
Dim i As Integer, j As Integer, k As Integer
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            If QW(i, j, k) <> 0# Then WellConstraint(i, j, k) = "WRate"
            If QG(i, j, k) <> 0# Then WellConstraint(i, j, k) = "GRate"
            If QG(i, j, k) <> 0# And QW(i, j, k) <> 0# Then
WellConstraint(i, j, k) = "2Rate"
            If Pwf(i, j, k) <> 0# Then WellConstraint(i, j, k) =
"Pressure"
        Next
    Next
Next
End Sub
```

```
Sub Fluids_In_Place()
Dim i As Integer, j As Integer, k As Integer, m As Integer
TotalWIP = 0#: TotalGIP = 0#
For k = 1 To nz
    For j = 1 To ny
        For i = 1 To nx
            TotalWIP = TotalWIP + OWIP(i, j, k)
            TotalGIP = TotalGIP + OGIP(i, j, k)
        Next
    Next
Next

With ThisWorkbook.Sheets("RESULTS")
    .Cells(1, 1) = "SIMULATION RESULTS"
    .Cells(2, 1) = "Fluids in Place"
    .Cells(4, 1) = "OWIP = " & Round(TotalWIP / (1000000 * 5.615), 1) &
" MMSTB"
    .Cells(5, 1) = "OGIP = " & Round(TotalGIP / 1000000, 1) & " MMSCF"
End With

Print #2, ""
Print #2, "Fluids in place"
Print #2, "OWIP = ", Round(TotalWIP / (1000000 * 5.615), 1) & " MMSTB"
Print #2, "OGIP = ", Round(TotalGIP / 1000000, 1) & " MMSCF"

End Sub


Sub Report()
Call Print_Report
Dim i As Integer, j As Integer, k As Integer, m As Integer

With ThisWorkbook.Sheets("Pressure")
    .Cells(1, 1) = "Pressure at every grid block"
```

```
    .Cells((rc - 1) * (ny * nz + nz + 1) + 2, 1) = "Time = " &
Round(time, 2)
    For k = 1 To nz
    .Cells((rc - 1) * (ny * nz + nz + 1) + (ny + 1) * (k - 1) + 3, 1) =
"k = " & k
        For j = 1 To ny
            For i = 1 To nx
           .Cells((rc - 1) * (ny * nz + nz + 1) + (ny + 1) * (k - 1) +
j + 2, i + 1) = Round(pn(i, j, k), 2)
           Next
       Next
    Next
End With

With ThisWorkbook.Sheets("Sw")
    .Cells(1, 1) = "Water Saturation at every grid block"
    .Cells((rc - 1) * (ny * nz + nz + 1) + 2, 1) = "Time = " &
Round(time, 2)
    For k = 1 To nz
    .Cells((rc - 1) * (ny * nz + nz + 1) + (ny + 1) * (k - 1) + 3, 1) =
"k = " & k
        For j = 1 To ny
            For i = 1 To nx
            .Cells((rc - 1) * (ny * nz + nz + 1) + (ny + 1) * (k - 1) +
j + 2, i + 1) = Round(SWI(i, j, k), 6)
            Next
        Next
    Next
End With

With ThisWorkbook.Sheets("Sg")
    .Cells(1, 1) = "Gas Saturation at every grid block"
    .Cells((rc - 1) * (ny * nz + nz + 1) + 2, 1) = "Time = " &
Round(time, 2)
    For k = 1 To nz
```

```
    .Cells((rc - 1) * (ny * nz + nz + 1) + (ny + 1) * (k - 1) + 3, 1) =
"k = " & k
        For j = 1 To ny
            For i = 1 To nx
            .Cells((rc - 1) * (ny * nz + nz + 1) + (ny + 1) * (k - 1) +
j + 2, i + 1) = Round(SGI(i, j, k), 6)
            Next
        Next
    Next
End With

With ThisWorkbook.Sheets("Sw+Sg")
    .Cells(1, 1) = "Sum of Saturation at every grid block"
    .Cells((rc - 1) * (ny * nz + nz + 1) + 2, 1) = "Time = " &
Round(time, 2)
    For k = 1 To nz
    .Cells((rc - 1) * (ny * nz + nz + 1) + (ny + 1) * (k - 1) + 3, 1) =
"k = " & k
        For j = 1 To ny
            For i = 1 To nx
            .Cells((rc - 1) * (ny * nz + nz + 1) + (ny + 1) * (k - 1) +
j + 2, i + 1) = Round(SWI(i, j, k) + SGI(i, j, k), 6)
            Next
        Next
    Next
End With

If nwell <> 0 Then
    With ThisWorkbook.Sheets("WELLS")
        .Cells(2, 1 + nwell) = "Water Rate, STB/D"
        .Cells(2, 1 + nwell * 2) = "Cum. Water, MSTB"
        .Cells(2, 2 + nwell * 3) = "Gas Rate, Mscf/D"
        .Cells(2, 2 + nwell * 4) = "Cum. Gas, MMscf"
        .Cells(2, 3 + nwell * 5) = "Pwf, psi"
        .Cells(2, 4 + nwell * 6) = "Well Type"
        .Cells(2, 2 + nwell * 8) = "Avg Rsr Prs"
```

```
        .Cells(2, 1) = "Time Step"
        .Cells(3, 1) = "Days"
        .Cells(rc + 4, 1) = Round(time, 2)

    For m = 1 To nwell
        Select Case WellConstraint(XCoor(m), YCoor(m), ZCoor(m))
            Case "GRate"
                            .Cells(3, m + 1 + nwell * 3) = Wellname(m)
& " layer " & ZCoor(m):                                    .Cells(3, m
+ nwell) = Wellname(m) & " layer " & ZCoor(m)
                            .Cells(rc + 4, m + 1 + nwell * 3) =
Round(QG(XCoor(m), YCoor(m), ZCoor(m)) / 1000, 1):
.Cells(rc + 4, m + nwell) = Round(QW(XCoor(m), YCoor(m), ZCoor(m)) /
5.615, 1) ' .Cells(rc + 4, m + nwell) = Round((QW(XCoor(m), YCoor(m),
ZCoor(m)) - (-0.0002 * time ^ 2 + 0.0109 * time + 24)) / 5.615, 1)

                            .Cells(3, m + 1 + nwell * 4) = Wellname(m)
& " layer " & ZCoor(m)
                            .Cells(rc + 4, m + 1 + nwell * 4) =
(.Cells(rc + 4, m + 1 + nwell * 3) * dt1) / 1000 + .Cells(rc + 3, m + 1
+ nwell * 4)

                            .Cells(3, m + nwell * 2) = Wellname(m) & "
layer " & ZCoor(m)
                            .Cells(rc + 4, m + nwell * 2) = (.Cells(rc
+ 4, m + nwell) * dt1) / 1000 + .Cells(rc + 3, m + nwell * 2)

                            .Cells(3, m + 2 + nwell * 5) = Wellname(m)
& " layer " & ZCoor(m)
                            .Cells(rc + 4, m + 2 + nwell * 5) =
Round(Pwf(XCoor(m), YCoor(m), ZCoor(m)), 1)

            Case "WRate"
                            .Cells(3, m + nwell) = Wellname(m) & "
layer " & ZCoor(m)
```

```
                            .Cells(rc + 4, m + nwell) =
Round(QW(XCoor(m), YCoor(m), ZCoor(m)) / 5.615, 1)

                            .Cells(3, m + nwell * 2) = Wellname(m) & "
layer " & ZCoor(m)
                            .Cells(rc + 4, m + nwell * 2) = (.Cells(rc
+ 4, m + nwell) * dt1) / 1000 + .Cells(rc + 3, m + nwell * 2)

                            .Cells(3, m + 1 + nwell * 3) = Wellname(m)
& " layer " & ZCoor(m)
                            .Cells(rc + 4, m + 1 + nwell * 3) =
Round(QG(XCoor(m), YCoor(m), ZCoor(m)) / 1000, 1)

                            .Cells(3, m + 1 + nwell * 4) = Wellname(m)
& " layer " & ZCoor(m)
                            .Cells(rc + 4, m + 1 + nwell * 4) =
(.Cells(rc + 4, m + 1 + nwell * 3) * dt1) / 1000 + .Cells(rc + 3, m + 1
+ nwell * 4)

                            .Cells(3, m + 2 + nwell * 5) = Wellname(m)
& " layer " & ZCoor(m)
                            .Cells(rc + 4, m + 2 + nwell * 5) =
Round(Pwf(XCoor(m), YCoor(m), ZCoor(m)), 1)

            Case "Pressure"
                            .Cells(3, m + 1 + nwell * 3) = Wellname(m)
& " layer " & ZCoor(m):                                    .Cells(3,
m + nwell) = Wellname(m) & " layer " & ZCoor(m)
                            .Cells(rc + 4, m + 1 + nwell * 3) =
Round(QG(XCoor(m), YCoor(m), ZCoor(m)) / 1000, 1):
.Cells(rc + 4, m + nwell) = Round(QW(XCoor(m), YCoor(m), ZCoor(m)) /
5.615, 1) ' .Cells(rc + 4, m + nwell) = Round(QW(XCoor(m), YCoor(m),
ZCoor(m)) / 5.615 - (0.00016 * time ^ 2 - 0.17 * time + 16), 1)

                            .Cells(3, m + 1 + nwell * 4) = Wellname(m)
& " layer " & ZCoor(m)
```

```
                            .Cells(rc + 4, m + 1 + nwell * 4) =
(.Cells(rc + 4, m + 1 + nwell * 3) * dt1) / 1000 + .Cells(rc + 3, m + 1
+ nwell * 4)

                            .Cells(3, m + nwell * 2) = Wellname(m) & "
layer " & ZCoor(m)
                            .Cells(rc + 4, m + nwell * 2) = (.Cells(rc
+ 4, m + nwell) * dt1) / 1000 + .Cells(rc + 3, m + nwell * 2)

                            .Cells(3, m + 2 + nwell * 5) = Wellname(m)
& " layer " & ZCoor(m)
                            .Cells(rc + 4, m + 2 + nwell * 5) =
Round(Pwf(XCoor(m), YCoor(m), ZCoor(m)), 1)
        End Select
    Next


   'Print Well Type & Average Reservoir Pressure
   For m = 1 To nwell
                        .Cells(3, m + 3 + nwell * 6) = Wellname(m) & "
layer " & ZCoor(m)
                        .Cells(rc + 4, m + 3 + nwell * 6) =
WellConstraint(XCoor(m), YCoor(m), ZCoor(m))
   Next

   For m = 1 To 1
                        .Cells(3, m + 4 + nwell * 7) = "psi"
                        .Cells(rc + 4, m + 4 + nwell * 7) = Round(PSum)
/ (nx * ny * nz)
   Next
End With

With ThisWorkbook.Sheets("RESULTS")
        .Cells(8, 1) = "Cumulative Water = " & Round(CumWater / 5615,
2) & " MSTB"
```

```
        .Cells(9, 1) = "Cumulative Gas = " & Round(CumGas / 1000000, 1)
& " MMSCF"
        .Cells(10, 1) = "Pore Volume = " & Round(PoreVol / 1000, 2) & "
MMSTB"
        .Cells(1, 6) = "X-->"
        .Cells(2, 5) = "Z"

        For k = 1 To nz
            For i = 1 To nx
                 .Cells(1 + k, 5 + i) = zThick(1, 1, k)
            Next
        Next
End With

End If

End Sub


Sub Print_Report()
Dim i As Integer, j As Integer, k As Integer, m As Integer
Print #2, ""
Print #2, "Time step:", time
Print #2, ""

'Print pressure at grid blocks
Print #2, "Pressure"
For k = 1 To nz
Print #2, "k= " & k
    For j = 1 To ny
    'Print #2, ""
        For i = 1 To nx
            Print #2, Round(pn(i, j, k), 2),
        Next
    Print #2, ""
    Next
```

```
Next
'Print Water saturation at every grid block
Print #2, ""
Print #2, "Water Saturation"
For k = 1 To nz
Print #2, "k= " & k
    For j = 1 To ny
        For i = 1 To nx
            Print #2, Round(SWI1(i, j, k), 6),
        Next
    Print #2, ""
    Next
Next
'Print Gas Saturation at every grid block
Print #2, ""
Print #2, "Gas Saturation"
For k = 1 To nz
Print #2, "k= " & k
    For j = 1 To ny
        For i = 1 To nx
            Print #2, Round(SGI1(i, j, k), 6),
        Next
    Print #2, ""
    Next
Next
'Print Sum Saturation Verification
Print #2, ""
Print #2, "Verification Sum Sat"
For k = 1 To nz
Print #2, "k= " & k
    For j = 1 To ny
        For i = 1 To nx
            Print #2, Round((SWI1(i, j, k) + SGI1(i, j, k)), 6),
        Next
    Print #2, ""
    Next
```

```
Next

'Print production for wells, if any
If nwell <> 0 Then
Print #2, ""
    Print #2, "Well Name", "Layer", "Qw STB/D", "Qg MSCF/D", "Pwf psi"
    For m = 1 To nwell
        For k = 1 To nz
            If k = ZCoor(m) Then
                For j = 1 To ny
                    If j = YCoor(m) Then
                        For i = 1 To nx
                            If i = XCoor(m) Then
                                Print #2, Wellname(m), ZCoor(m),
Round(QW(i, j, k) / 5.615, 2), Round(QG(i, j, k) / 1000, 2),
Round(Pwf(i, j, k), 2)
                            End If
                        Next
                    End If
                Next
            End If
        Next
    Next
Print #2, ""
Print #2, "Cum Water Production (MSTB)   : ", Round(CumWater / 5615, 2)
Print #2, "Cum Gas Production   (MMSCF)  : ", Round(CumGas / 1000000,
2)
Print #2, ""
End If

End Sub
```

VITA

Name: Saeed Forghany

Permanent Address: No. 12, Sahebazaman St.
Isfahan, P.O. Box: 819 666 9368
IRAN
E-mail: saeed_f@tamu.edu

Education: B.Sc., Mining Engineering
Sahand University of Technology,
Tabriz, IRAN
(Sep. 1996-Sep. 2000)

M.Sc., Petroleum Engineering
Texas A&M University
College Station
Texas, U.S.A.
(Jan. 2002-May 2004)