AN ABSTRACT OF THE THESIS OF

Shannon Miriah Biederman for the degree of Master of Science in Mathematics presented on February 8, 2007.

title: Nonlinear Solvers for a Model Problem of Fluid Flow in the Subsurface

Abstract approved: _____

Malgorzata Peszynska

Water is one of the most biologically and economically important substances on Earth. A significant portion of Earth's water subsists in the subsurface. Our ability to monitor the flow and transport of water and other fluids through this unseen environment is crucial for a myriad of reasons.

One difficulty we encounter when attempting to model such a diverse environment is the nonlinearity of the partial differential equations describing the complex system. In order to overcome this adversity, we explore Newton's method and its variants as feasible numerical solvers. The nonlinearity of the model problem is perturbed to create a linearized one. We then investigate whether this linearized version is a reasonable replacement.

Finite difference methods are used to approximate the partial differential equations. We assess the appropriateness of approximating the analytical Jacobian that arises in Newton's method, with an approximation method, to handle the event when certain derivative information is not available.

Nonlinear Solvers for a Model Problem of Fluid Flow in the Subsurface

by

Shannon Miriah Biederman

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented February 8, 2007
Commencement June 2007

Master of Science thesis of Shannon Miriah Biederman presented on February 8, 2007

APPROVED:

Major Professor, representing Mathematics

Chair of the Department of Mathematics

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Shannon Miriah Biederman, Author

ACKNOWLEDGEMENTS

Academic

I am indebted to Dr. Malgorzata Peszynska for the tremendous amount of time and patience she provided me.

Research in this paper was partially supported by National Science Foundation, grant DMS-0511190 "Model Adaptivity in Porous Media", under the direction of Malgorzata Peszynska (Principal Investigator).

Personal

I wish to thank Ryan Harbert for his editing contributions and understanding, and my parents for all of the support they have given me.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

NONLINEAR SOLVERS FOR A MODEL PROBLEM OF FLUID FLOW IN THE SUBSURFACE

1. INTRODUCTION

1.1. Statement of the Problem

A porous medium is a solid permeated by an interconnected network of pores (voids) filled with a fluid (liquid or gas). The subsurface of our Earth consists of many different forms of porous medium. The fluids that exist simultaneously create a flow system. Since a flow system is a closed system, it satisfies the laws of conservation of mass.

In our desire to chart the behavior of fluid as it traverses through the Earth, we must be able to solve the conservation and constitutive equations that describe this flow. The model problem (one of a slightly compressible, single phase, fluid flow in 1-D) is a nonlinear partial differential equation (PDE). We discretize the original PDE and obtain a discrete system of nonlinear equations. The discretization scheme used in this paper consists of approximating the differential equations with difference equations [8]. This involves a mix of backwards Euler and centered finite differences.

To solve the discrete nonlinear system, we use Newton's method. Newton's method is an effective algorithm that, with a few hypothesis, offers the hope that the solution of the PDE will retain the rate of convergence associated with the discretization scheme applied. In an attempt to decrease computational time, some variants of Newton's method are implemented. In addition, we are interested in how the variants can be related to different modeling choices. In future research, we will explore changing the model adap-

tively depending on what we have learned from perfomance of the nonlinear solvers. In particular, if a certain simple variant of Newton's method works well, this may suggest that a linearized model, instead of the complex nonlinear model, would be sufficient to describe the dynamics of the flow.

1.2. Organization of the Thesis

This paper is organized into eight sections. In Section 2, we introduce the slightly compressible fluid flow model as well as explaining the different equations that govern such a model. In Section 3, we develop the discretization. In Section 4, we describe the various solvers (Newton's method and variants) used to solve the discrete nonlinear system. The focus of Section 5 is to test the convergence of the discretization and of solvers. In Section 6, we present an application where a typical situation is outlined. In Section 7, we discuss another direction leading to Richards equation of unsaturated flow in porous media. A brief summary of the paper and ideas for the direction of continued research are stated in Section 8. The reader will find the applicable code used in this research in the Appendix.

2. MATHEMATICAL BACKGROUND

In this paper, we are interested in solving the partial differential equation that describes flow of a slightly compressible fluid in a porous medium. In general, this model is written in a three dimensional spatial domain (for more details see [6], [5], [3]). We will be concentrating on the one dimensional case. This discussion could be extended to the three dimensional case; after the theory and discretization is understood in this simpler form. The equation we are interested in is the parabolic diffusion equation. This equation describes the density fluctuations of a slightly compressible fluid. It has the following general form:

$$\frac{\partial}{\partial t}(a(u)) - \frac{\partial}{\partial x}(b(u)\frac{\partial u}{\partial x}) = f(x,t), \quad 0 < x < L, \quad t > 0. \tag{2.1}$$

2.1. Description of the Model

Fluid flow of water in the subsurface requires many variables, and many equations, to characterize its movement. In the following sections, the equations that are used to describe the system, in a complete and physical way, are outlined. These are equations for conservation of mass, momentum, and the constitutive equations. The typical variables with units are listed in TABLE(2.1)[4].

2.1.1 Conservation of Mass

First, we discuss the law of conservation of mass. Let us denote by ϕ the porosity of the porous medium (dimensionless), by p the fluid pressure (kg/(ms^2), by ρ the density (kg/m^3), and by $\vec{v}$ the filtration velocity (m/s) [5]. Here, ρ and p are the unknowns of the model. In general, $\rho = \rho(x,y,z,t)$ and $p = p(x,y,z,t)$. Then, in every volume V of porous medium, the time rate of change of mass is $\int_V \frac{\partial}{\partial t}(\phi\rho)dV$. The total mass flux across the

SI base quantities and units		
Base quantity	SI unit & symbol	Common symbol
Time	Second (s)	t
Length, Position	Meter (m)	L,x,y,z
Mass	Kilogram (kg)	M

Common derived units	
Quantity	Formula
Pressure	Newton/m^2
Velocity	m/s
Area	m^2
Volume	m^3
Density	kg/m^3

TABLE 2.1: Quantities with Units

boundary ∂V of V is $\int_{\partial V} \rho\vec{v}\cdot\vec{\eta}dS$ (dS is the surface area). Applying the Gauss divergence theorem to the boundary integral we have:

$$\int_{\partial V} \rho\vec{v}\cdot\vec{\eta}dS = \int_V \nabla\cdot(\rho\vec{v})dV.$$

Conservation theory dictates that the sum of two integrals $\int_V \frac{\partial}{\partial t}(\phi\rho)dV$ and $\int_{\partial V}\rho\vec{v}\cdot\vec{\eta}dS$ must be equal to the total amount of fluid, $\int_V f dV$ [6]. The term f will denote the external sources or sinks, f will be negative for sinks and positive for sources. Thus we have,

$$\int_V \frac{\partial}{\partial t}(\phi\rho)dV + \int_V \nabla\cdot(\rho\vec{v})dV = \int_V f dV. \tag{2.2}$$

The du Bois-Reymond lemma (1879) is a useful way to obtain a differential form of the conservation laws from the original integral formulation, equation (2.2). It states that since equation (2.2) is true in any volume, we may write a differential equation instead of an integral one. This is how the model equation (2.3) is derived.

$$\frac{\partial}{\partial t}(\phi\rho) + \nabla\cdot(\rho\vec{v}) = f. \tag{2.3}$$

2.1.2 Conservation of Momentum

If the flow is slow, it is appropriate to use Darcy's Law [3]. Darcy's Law is an expression of conservation of momentum. It is an empirical relationship describing fluid

flow rate in a porous medium dependent upon the pressure gradient.

Since fluid displacement in porous media depends not only on pressure gradients, but also on external gravity forces, the following formula (Darcy's Law) is used to express this dependence:

$$\vec{v} = \frac{-k}{\mu}(\nabla p - \rho g \nabla D). \tag{2.4}$$

In equation (2.4), k can represent a constant, or a three dimensional real matrix. If k is a matrix then this matrix would need to be symmetric and positive definite. This is discussed further in Section (2.2). Either way, k represents permeability (m^2). In this paper, we are assuming that the medium is isotropic, having the same properties in all directions. Thus, k is a scalar. If we also assume that the medium is homogeneous, we can let k be constant [3]. The other variables are g, the acceleration due to gravity ($\mathrm{m/s}^2$), μ, the dynamic viscosity (kg/(ms)), and D (m), the depth in space (vertical axis directed upward) [5]. We are not concerned with gravity in this paper.

Simplifying equation (2.4) to one spatial dimension in the horizontal direction necessitates that $\nabla D = 0$. We also use $\mu = 1$ for simplicity. This gives,

$$\vec{v} = -k\frac{\partial p}{\partial x}. \tag{2.5}$$

We note that Darcy's Law can be derived from calculations based on Navier-Stokes equations (a set of equations that describe the motion of fluid substances) at the microscopic level.

2.1.3 Constitutive Equation

One property of compressible fluids is that the density of the flow cannot be assumed to be constant. Water is a slightly compressible fluid. The following constitutive equation is an equation of state that is specific for many liquids. For example, for water,

$$\rho(p) = \rho_{ref} e^{c(p - p_{ref})}, \tag{2.6}$$

where ρ_{ref} is the density at the reference pressure p_{ref}. The compressibility coefficient is c ($(\text{ms}^2)/\text{kg}$), and ρ_{ref} is some reference density which we let equal one. Normally, for oil, $c_{oil} \approx 10^{-4}$ $(\text{ms}^2)/\text{kg}$) and for water $c_{H_2O} \approx 10^{-8}$ $(\text{ms}^2)/\text{kg}$) [3]. That is, we may assume that density is only slightly dependent upon pressure. Because the compressibility coefficient of water is so much smaller than that of oil, we say that water is practically incompressible in comparison to oil (or air).

2.1.4 Choosing the Primary Variable

Combining equations (2.3) and (2.4) we obtain,

$$\frac{\partial}{\partial t}(\phi\rho) - \nabla \cdot (\rho\frac{k}{\mu}(\nabla p - \rho g \nabla D)) = f. \tag{2.7}$$

With the simplification introduced in (2.5) ($\nabla D = 0$ and $\mu = 1$), and the reduction to one spatial dimension, equation (2.7) can be reduced to,

$$\frac{\partial}{\partial t}(\phi\rho) - \frac{\partial}{\partial x}(\rho k\frac{\partial p}{\partial x}) = f. \tag{2.8}$$

Additionally, from equation (2.6) we see that the constitutive equation can be written with respect to p ($\rho = \rho(p)$), instead of for ρ, as follows,

$$\frac{\partial}{\partial t}(\phi\rho(p)) - \frac{\partial}{\partial x}(\rho(p)k\frac{\partial p}{\partial x}) = f. \tag{2.9}$$

This allows us the option to rewrite (2.8) as a partial differential equation which is solved for either ρ or p. We will explore three different scenarios; the linear PDE in ρ, the nonlinear PDE in $\rho(p)$, and the linearized PDE for $\rho(p)$.

In the first scenario, the model equation, equation (2.8), is written with respect to ρ and is therefore a linear problem, as will be shown now. To translate equation (2.8) into one which is linear, we have to express $\frac{\partial p}{\partial x}$ using $\frac{\partial \rho}{\partial x}$. First, we use the fact that ρ was defined as the exponential in equation (2.6) to find the analytical translation of the derivative of ρ, $\frac{\partial}{\partial x}\rho = c\rho$. This relationship implies that $\rho(p) = \frac{1}{c}\frac{\partial}{\partial x}\rho(p)$. The conversion proceeds by

first writing $\rho(p)k\frac{\partial p}{\partial x}$ as $k\frac{1}{c}\frac{\partial}{\partial x}\rho(p)\frac{\partial p}{\partial x}$, then using the chain rule to give $\frac{1}{c}k\frac{\partial}{\partial x}\rho(p)$. We can now change $b(u)$ to be equal to $\frac{1}{c}k$ (to use the code notation from equation (2.1)). With this conversion, we can write equation (2.8) as a linear partial differential equation with respect to ρ, as seen below.

$$\frac{\partial}{\partial t}(\phi\rho) - \frac{\partial}{\partial x}(\frac{1}{c}k\frac{\partial}{\partial x}\rho) = f. \tag{2.10}$$

In the natural world, we will most often have to solve a nonlinear PDE. Therefore, in Scenario 2, the nonlinear PDE, with respect to p, is assigned notation as seen in equation (2.9) and (2.6).

Nonlinear equations are, in general, more difficult to solve than their linear counterparts. Because of this, in Scenario 3, we look at the linearization of the nonlinear model equation. This linearization is a modification of Scenario 2.

To linearize (2.9), we first consider a Taylor series expansion of equation (2.6)[4].

$$\rho = \rho_{ref}\{1 + c(p - p_{ref}) + \frac{1}{2!}c^2(p - p_{ref})^2 + \cdots\},$$

which can be truncated to

$$\rho(p) \approx \rho_{ref}(1 + c(p - p_{ref})). \tag{2.11}$$

The final linearized model, a combination of equations (2.9) and (2.11) is,

$$\frac{\partial}{\partial t}(\phi\rho_{ref}(1 + c(p - p_{ref}))) - \frac{\partial}{\partial x}(\rho_{ref}(1 + c(p - p_{ref}))k\frac{\partial p}{\partial x}) = f. \tag{2.12}$$

In summary, equations (2.9), (2.10), and (2.12) present three scenarios all dependent upon which representation of ρ we wish to solve for. The choice depends upon whether or not a nonlinear or linearized version of the model problem is to be solved. In TABLE (2.2), all versions of the model are listed.

Model Variables and terms	Scenario 1 Linear	Scenario 2 Nonlinear	Scenario 3 Linearized
$u(x,t)$	density $\rho(x,t)$	pressure $p(x,t)$	pressure $p(x,t)$
$a(u)$	$\phi\rho$	$\phi\rho_{ref}e^{c(p-p_{ref})}$	$\phi\rho_{ref}(1+c(p-p_{ref}))$
$b(u)$	$\frac{1}{c}k$	$k\rho_{ref}e^{c(p-p_{ref})}$	$k\rho_{ref}(1+c(p-p_{ref}))$

TABLE 2.2: Descriptions of the Model Variables

2.2. Analysis

When modeling the slightly compressible fluid flow case in porous media, we do so using the nonlinear partial differential equation (2.1) and one of its special cases: Scenario 1, 2, or 3 above.

As the theory for nonlinear parabolic equations is relatively difficult, it is easier to study the theory for the linear parabolic equations. There is sufficient theory concerning the well-posedness (existence, uniqueness, and continuous dependency) of linear problems. Fortunately, for the slightly compressible fluid model, a linear version equivalent to the nonlinear one between pressure and density exists, as was seen in Section (2.1.4).

A differential equation is considered to be linear if it is linear in the dependent variable. Consider the following linear initial boundary value problem in one space dimension (where sources or sinks $f(x,t)$ may be present) for ϕ, $k \in \mathbb{R}^+$ (constants).

$$\phi\frac{\partial u}{\partial t} - \frac{\partial}{\partial x}(k\frac{\partial u}{\partial x}) = f(x,t), \quad 0 \le x \le L, \quad t > 0. \tag{2.13}$$

$$u(0,t) = g(t), \quad u(L,t) = h(t), \quad t \ge 0. \tag{2.14}$$

$$u(x,0) = u_0(x), \quad 0 \le x \le L. \tag{2.15}$$

In equation (2.14), the functions $g(t)$ and $h(t)$ represent inhomogeneous boundary conditions, and u_0 represents the inhomogeneous initial condition.

If this linearization was not available, we would have to deal with the theory involving the nonlinear problem separately. Here, it is enough to explore the linear theory and then adopt this theory to our nonlinear case, presuming the nonlinear problem is *well behaved* enough. A problem is generally assumed to be *well behaved* if it does not violate any assumptions needed to successfully apply whatever analysis is being discussed [8]. In the next section, the well-posedness of the linear initial boundary value problem (equations (2.13)-(2.15)) is addressed.

2.2.1 Well–Posedness for the Linear Case with Constant Coefficients

Theorem (2.2.1.1) below discusses the well-posedness of equations (2.13) - (2.15). Although this theorem deals with a solution to the homogeneous boundary condition case, a simple modification to the method used to find $u(x,t)$ could eliminate this discrepancy. It is also assumed that both ϕ, $k > 0$. This is a reasonable physical assumption [6].

Theorem 2.2.1.1 *Suppose that* $u_0(x) \in \mathcal{L}^2(0,L)$ *and* $f(x,t) \in \mathcal{C}[(0,L),(0,T)]$ *with* $u_0(0) = u_0(L) = 0$. *Then* $u(x,t)$ *is a solution of the initial boundary value problem (2.13)-(2.15).*

Besides the obvious importance of the existence of a unique solution to the problem we are solving, there is also the very important idea of continuous dependency. If the problem varies continuosly with the inhomogeneous data, we can assume that any small perturbation introduced via actual data will only produce a small perturbation in the solution. This characteristic, along with the existence of a unique solution, is required if a system is deemed to be well-posed. If we were not able to prove that our problem satisfied the conditions to be well-posed, we would have to find a new mathematical system to describe our natural event [6]. At the same time, we note that some problems are not well-posed. For example, inverse problems are frequently ill-posed. We will not be concerned with ill-posed problems in this paper.

2.2.2 Well–Posedness, General Case

A general nonlinear PDE of the form (2.1) can be analyzed with techniques beyond those discussed in this paper. For the purposes of this paper, we are satisfied with the well–posedness of the linear problem (2.10) and its equivalence, via the smooth function (2.6), to a nonlinear equation (2.9). Clearly, the theory presented above also applies to the linearized model (2.12).

3. CONSTRUCTION OF THE FINITE DIFFERENCE SCHEME

3.1. Discretization

To discretize the continuous diffusion equation first shown in equation (2.1) and find its numerical solution on a computer, we must first decide which method we want to use.

The numerical scheme for (2.1) needs to solve the first order in time and second order in space partial differential equation. One way to do this is to choose a backwards difference operator for the time derivative while using a centered difference operator for the spatial derivative. The backwards difference operator is only a first order accurate scheme, whereas the centered in space is second order accurate. Bacwards difference was chosen over othe schemes due to the unconditionally stable property it has. Other numerical schemes implemented could be those of finite elements or finite volume, but this will not be discussed.

For the finite difference method used in this paper, we use a uniform spatial grid and uniform time-stepping. Let x_0, x_1, $\cdots$, x_M be a uniform partition of $(0, L)$, where M is the number of spatial steps. Let t_0, t_1, $\cdots$, t_N be a uniform partition of $(0, T)$, where N is the number of temporal time steps. With this, we let Δx represent the step size in space ($\Delta x = (x_{right} - x_{left})/M = x_j - x_{j-1}$ for all j), and let Δt represent the step size in time ($\Delta t = (t_{final} - t_{initial})/N = t^n - t^{n-1}$ for all n). In our algorithm we use $x_{right} - x_{left} = L - 0$ where L is the length (L = 1). We also set $t_{final} - t_{initial} = T - 0$ where T is the final time (T = 1).

We approximate $u(x_j, t^n)$ by U_j^n. We find U_j^n using the following finite difference equation with $U_j^0 \approx u(x_j, 0)$ and $f_j^n \approx f(x_j, t^n)$. The terms given by $j = 0$ and $j = M + 1$ are given by the boundary conditions, the terms associated with $n = 0$ are given from the

initial condition.

$$\frac{1}{\Delta t}[a(U_j^n) - a(U_j^{n-1})] + \frac{1}{(\Delta x)^2}[b(u^n_{j-\frac{1}{2}})(U_j^n - U_{j-1}^n) - b(u^n_{j+\frac{1}{2}})(U_{j+1}^n - U_j^n)] = f_j^n \quad (3.1)$$

for $j = 2...M$ and $n = 1, 2, ...$

Now we discuss how to handle the nonlinear terms $b(u^n_{j\pm\frac{1}{2}}) = b(u(x_{j\pm 1/2}, t^n))$ that appear in equation (3.1). For these terms, we apply the arithmetic average, $b(\frac{1}{2}(U_j^n + U_{j\pm1}^n))$. This choice maintains symmetry and results in a positive-definite matrix of the discrete system. We have

$$\frac{1}{\Delta t}[a(U_j^n) - a(U_j^{n-1})] + \frac{1}{(\Delta x)^2}[b(\frac{1}{2}(U_j^n + U_{j-1}^n))(U_j^n - U_{j-1}^n) - b(\frac{1}{2}(U_j^n + U_{j+1}^n))(U_{j+1}^n - U_j^n)] = f_j^n \quad (3.2)$$

for $j = 2...M$ and $n = 1, 2, ...$

Now we state the version of the scheme for the linear problem such as the one in equation (2.13)

$$\frac{\phi}{\Delta t}(U_j^n - U_j^{n-1}) + \frac{k}{(\Delta x)^2}[(U_j^n - U_{j-1}^n) - (U_{j+1}^n - U_j^n)] = f_j^n \quad (3.3)$$

for $j = 2...M$ and $n = 1, 2, ...$

3.1.1 Convergence Theorem

Now we want to discuss the convergence of the discrete scheme (3.2) to the solution of (2.1). Both the continuous and the discrete scheme are nonlinear. Therefore, similar to Section (2.2), we discuss convergence of our numerical scheme (3.3) for the linear parabolic partial differential equation given by equations (2.13)-(2.15).

The following Theorem (3.1.1.1) states that consistency and stability of a scheme, for a linear problem, together imply convergence [8]. This is useful because, for many finite difference schemes, convergence is difficult to prove, whereas consistency and stability are more manageable. A scheme is considered stable if, during some time step, an error is

introduced into the scheme and this error does not grow in size as the scheme progresses. A finite difference operator is consistent if it converges to the continuous operator as both $\Delta x \to 0$ and $\Delta t \to 0$.

Theorem 3.1.1.1 *Suppose that the difference scheme used in (3.1) is consistent to order (p,q) with the partial differential equation (2.13), where p, q are positive integers, and that the scheme is stable. If exact initial data is used then, as $\Delta x \to 0$, $\Delta t \to 0$,*

$$\| U^n - u^n \|_2 = O(\Delta t^p + \Delta x^q), \quad \text{for } 0 \leq t^n \leq T, \tag{3.4}$$

where U_n is the numerical approximation at some time n (t^n) and u_n is the exact value at that time.

In the proof of this theorem for consistency and stability one uses Taylor series and Fourier analysis, respectfully [8].

In order to discuss consistency of a discrete scheme we need the solution of the differential equation to be smooth. In particular it is necessary to construct the Taylor series, where a certain number of derivatives will need to be found. An adequate requirement for (3.3) to be consistent with (2.13) is to make $u \in C^4[(0, L) \times (0, T)]$ where $x \in (0, L)$ and $t \in (0, T)$.

The scheme (3.3) is consistent with order $p = 1$ and $q = 2$, because a simple one-sided difference operator is applied to the time derivative, and a simple second order centered difference scheme is applied to the second order spatial derivative [8]. Therefore, the scheme has a truncation error of $O(\Delta t + (\Delta x)^2)$. The truncation error is the measure of how accurately a difference analog approximates a difference operator [4].

Stability analysis indicates that the scheme is unconditionally stable, meaning that there is no restriction for what Δt is used [8].

In summary, the scheme (3.3) will converge with the rate $O(\Delta t + (\Delta x)^2)$ to the solution of (2.13). To obtain optimal convergence, we will use $\Delta t = (\Delta x)^2$.

3.1.2 Handling the Nonlinear Terms

In the following, we will assume that the theory in Section (3.1.1), which applies to linear problems, can be extended to the scheme (3.2) approximating (2.1), the nonlinear problem.

The difficulties in creating a discretized equation lie in the delicate process of handling the nonlinear terms and solving the nonlinear system. The former is defined in (3.2); we discuss the latter now.

To clean up equation (3.2), and prepare it for the application of Newton's method, we let $c = \frac{\Delta t}{(\Delta x)^2}$. Then equation (3.2) is multiplied by Δt giving,

$$F_j^n = a(U_j^n) - a(U_j^{n-1}) + c(b(\frac{1}{2}(U_j^n + U_{j-1}^n))(U_j^n - U_{j-1}^n) \\ - b(\frac{1}{2}(U_j^n + U_{j+1}^n))(U_{j+1}^n - U_j^n)) - (\Delta t) f_j^n = 0, \quad (3.5)$$

for $j = 2, 3, ...M$ and $n = 1, 2, ...$ The term F_j^n is called the residual.

Now that we have the diffusion equation properly approximated by finite difference operators, we may proceed to the construction of the Newton algorithm that will be used to solve the nonlinear problem for the specified time t^n. We will use the following scheme to represent the κ step of the iterative method,

$$F_j^{n,\kappa} = a(U_j^{n,\kappa}) - a(U_j^{n-1,\kappa}) + c(b(\frac{1}{2}(U_j^{n,\kappa} + U_{j-1}^{n,\kappa})(U_j^{n,\kappa} - U_{j-1}^{n,\kappa}) \\ - b(\frac{1}{2}(U_j^{n,\kappa} + U_{j+1}^{n,\kappa}))(U_{j+1}^{n,\kappa} - U_j^{n,\kappa})) - (\Delta t) f_j^{n,\kappa}, \quad (3.6)$$

where $j = 2, 3, ...M$, $n = 1, 2, ...$ and $\kappa = 0, 1, 2, ...$

In equation (3.5), U_j^n represents the time step that we are solving for, while U_{j-1}^n represents the previous time step (that is known). In equation (3.6), $U_j^{n-1,\kappa}$ is the previously converged iterate.

In the next section, we describe how to solve (3.5) iteratively using equation (3.6) to describe a single iterate. This leads us to the next section, where we discuss how to solve a nonlinear problem.

4. NEWTON'S METHOD

Newton's method is perhaps the best known iterative method used for solving nonlinear equations. The idea is to approximate the graph of a function by its tangent line, then find the root of this tangent line, and repeat.

Consider a system of M equations in the matrix–vector form,

$$\mathbf{F}(\mathbf{u}) = \mathbf{0}$$

where

$$\mathbf{F} = \begin{bmatrix} F_1(\mathbf{u}) \\ F_2(\mathbf{u}) \\ \vdots \\ F_M(\mathbf{u}) \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u_1(x,t) \\ u_2(x,t) \\ \vdots \\ u_M(x,t) \end{bmatrix}$$

Newton's method will require the computation of the Jacobian for this system of nonlinear equations. To find the Jacobian matrix ($\mathbf{DF}$) of $\mathbf{F}$ at a point $\mathbf{u}$ in the domain of $\mathbf{F}$, we must create the square matrix, $M \times M$, whose $(i,j)^{th}$ entry is

$$DF_{i,j}(\mathbf{u}) := \frac{\partial \mathbf{F}_i}{\partial u_j}(\mathbf{u}). \tag{4.1}$$

We now have sufficient notation to write the following set of equations as a template for Newton's method:

$$\mathbf{u}^{(\kappa+1)} = \mathbf{u}^{(\kappa)} + \mathbf{s}^{(\kappa)}, \tag{4.2a}$$

and

$$\mathbf{DF}(\mathbf{u}^{(\kappa)})\mathbf{s}^{(\kappa)} = -\mathbf{F}(\mathbf{u})^{(\kappa)}, \tag{4.2b}$$

where equation (4.2b) is equivalent to

$$\mathbf{s}^{\kappa} = -\mathbf{DF}(\mathbf{u}^{\kappa})^{-1}\mathbf{F}(\mathbf{u}^{\kappa}).$$

Each Newton step corresponds to an iteration κ, where $\kappa = 0, 1, 2, ...$ (stopping when the algorithm has determined convergence has occured). In the above, $\mathbf{u}^{(\kappa)}$ stands for the current iterate, and $\mathbf{u}^{(\kappa+1)}$ is the new iterate. The term $\mathbf{s}$ is commonly called the iteration step.

This can be rewritten in a mathematically equivalent, yet shorter form as,

$$\mathbf{u}^{(\kappa+1)} = \mathbf{u}^{(\kappa)} - \left[\mathbf{DF}(\mathbf{u}^{(\kappa)})\right]^{-1} \mathbf{F}(\mathbf{u}^{(\kappa)}). \tag{4.3}$$

The following is a pseudo-code expressing the Newton algorithm. Details can be found in the Appendix.

do until convergence...

% Newton Step and Error Calculation

$s = DF(1 : M + 1, 1 : M + 1) \setminus F(1 : M + 1)$; % s is the correction to the guess

$u(newguess) = u(guess) - s$;

$u(guess) = u(newguess)$;

end % end Newton loop

4.1. Convergence of Newton's Method

Before we begin to discuss convergence, we need the following definition [8]:

Definition 4.1.0.1 *Let $p \geq 1$. An iterative scheme that produces a sequence U^κ of approximations to $u^* \in \mathbb{R}^n$* ***converges with order*** *p if there exists a constant $C > 0$ and an integer $K \geq 0$ such that*

$$\| u^* - U^{\kappa+1} \| \leq C \| u^* - U^\kappa \|^p$$

whenever $\kappa \geq K$ for some $\| \cdot \| \in \mathbb{R}$. If $p = 1$ we require $0 \leq C < 1$, to ensure the error is decreasing, and we say the scheme converges ***linearly****. If $p = 2$, the scheme converges*

quadratically. *The scheme is said to converge* ***superlinearly*** *if there exists a sequence* $\{C_\kappa\}$ *of positive real numbers such that* $C_\kappa \to 0$ *and*

$$\| u^* - U^{\kappa+1} \| \leq C_\kappa \| u^* - U^\kappa \| .$$

The convergence rate of Newton's method, if it converges at all, is quadratic. Unfortunately, to ensure convergence, Newton's method requires a *good* initial guess. Here, a *good* initial guess is provided from the previous time step.

One of the reasons proving convergence for Newton's method is so difficult, is the necessity for an a priori assumption of the closeness between the initial guess and the zero of the system. The following theorem (4.1.0.1) [9] has the advantage of proving existence of a solution, and convergence to it, under very mild conditions. The theorem also promises quadratic convergence to this approximate solution. To use this theorem, a few standard assumptions must hold at some initial *good* guess. These assumptions are explained below. If they are met, the iteration converges to the root of $\mathbf{F}$ quadratically[9].

Standard Assumptions:

1. $\mathbf{F}(\mathbf{u}) = \mathbf{0}$ has a solution, u^*.
2. The Jacobian $\mathbf{DF}$ is Lipschitz continuous in the domain of $\mathbf{F}$ with Lipschitz constant γ.
3. The Jacobian is nonsingular at the supposed solution, $\mathbf{det}(\mathbf{DF})(u^*) \neq 0$.

Theorem 4.1.0.1 *Let the standard assumptions hold. Then there exists a* $K \geq 0$ *and* $\delta \geq 0$ *such that, if* $u^{(\kappa)} \in \mathcal{B}(\delta)$*, the Newton iterate* $u^{(\kappa)}$ *given by equation (4.3) satisfies*

$$\| e^{\kappa+1} \| \leq K \| e^\kappa \|^2 . \tag{4.4}$$

where $\| e^{\kappa+1} \|$ *is equal to the difference between the new iterate and the exact solution, and* $\| e^\kappa \|$ *is the difference between the current iterate and the exact solution.[9]*

4.2. An Example of Newton's Method

To show how Newton's method works, we look at a simple system of two nonlinear equations. Consider the following two nonlinear equations:

$$\mathbf{F}(x,y) = \begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix} = \begin{bmatrix} x^2 + 4y^2 - 9 \\ 18y - 14x^2 + 45 \end{bmatrix} = \mathbf{0} \tag{4.5}$$

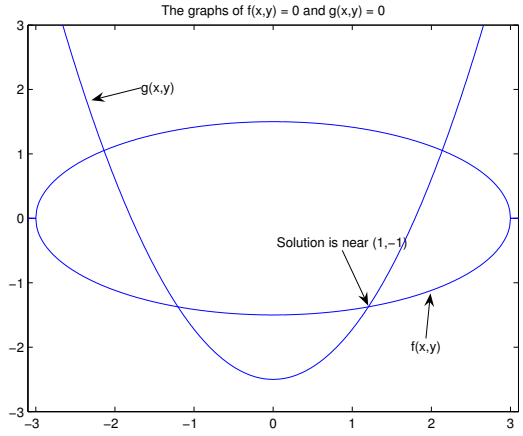

FIGURE 4.1: The Zero Curves of System (4.5)

The equations listed above are called the zero curves of the functions f and g [1]. When the zero curves are graphed simultaneously, the intersections of the curves of f and g correspond to the solutions of the system. Realizing that there is indeed a solution to this system (Figure (4.1)), we see that the first standard assumption has been met. It appears that one solution to this system occurs near to the point $(1, -1)$. Thus, if we use an initial guess of $(1, -1)$, we should converge very quickly to the exact solution. To ensure that this system also satisfies the second standard assumption, we look at the Jacobian.

$$\mathbf{DF} = \begin{bmatrix} 2x & 8y \\ -28x & 18 \end{bmatrix} \tag{4.6}$$

We must ensure that the Jacobian is indeed Lipschitz continuous, with Lipschitz constant γ.

Recall that a function h is Lipschitz continuous if there exists some $C \geq 0$ such that $\| h(x) - h(y) \| \leq C \| x - y \|$ is true for all $x, y \in \mathbb{R}$, in the applicable domain.

To verify that the Jacobian is Lipschitz continuous, let us take $x_1, y_1, x_2, y_2 \in \mathbb{R}$ and check the infinity norm of the difference between these two matrices. Then $\mathbf{DF}(x_1, y_1) - \mathbf{DF}(x_2, y_2)$ is found by matrix subtraction as,

$$\begin{bmatrix} 2x_1 & 8y_1 \\ -28x_1 & 18 \end{bmatrix} - \begin{bmatrix} 2x_2 & 8y_2 \\ -28x_2 & 18 \end{bmatrix} = \begin{bmatrix} 2(x_1 - x_2) & 8(y_1 - y_2) \\ -28(x_1 - x_2) & 0 \end{bmatrix}$$

This is implies that

$$\begin{aligned} \| \mathbf{DF}(x_1, y_1) - \mathbf{DF}(x_2, y_2) \|_\infty &= \max(2|x_1 - x_2| + 8|y_1 - y_2|, 28|x_1 - x_2|) \\ &\leq 28|x_1 - x_2| + 8|y_1 - y_2| \\ &\leq 28\max\left(|x_1 - x_2|, |y_1 - y_2|\right) \\ &\leq 28 \| (x_1, y_1) - (x_2, y_2) \|_\infty . \end{aligned}$$

Thus, for this norm, if we choose, for example $\gamma = 28$, we would satisfy the Lipschitz continuity requirement. If we chose, instead, to check the $\| \cdot \|_1$ norm, we would find that $\gamma = 30$. As long as we let γ be equal to the maximum of the two norms, the Jacobian will be Lipschitz continuous.

As for the third assumption, the determinant of matrix (4.6) is nonsingular as long as $x \neq 0$ and $y \neq \frac{18}{11}$. Because the solution we are looking for is close to $(1, -1)$, all three of the standard assumptions have been met. This means that we can use Newton's method, confident that we will have convergence.

In TABLE (4.1), we show results of Newton's method applied to (4.5). Newton's method stops if the norm of the residual does not exceed $1e - 7$. The exact error was computed by: exact error $= \| \alpha - (x_k, y_k) \|$, where α is the exact solution of (4.5) [1]. The

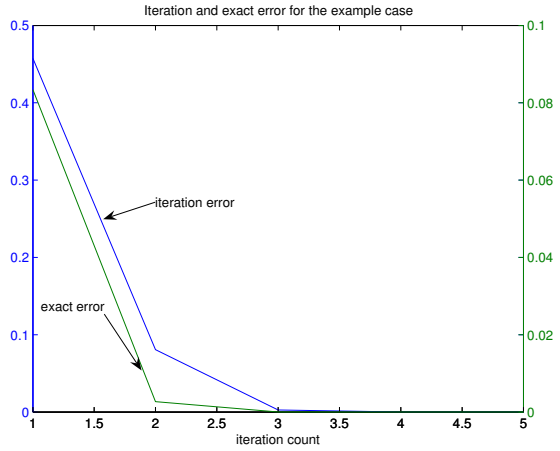

FIGURE 4.2: Iteration Error and Exact Error for the System in (4.5)

iteration error (as defined in equation (4.7)) and the exact error are shown with respect to the iteration count in Figure (4.2). This shows how the error progressed during one time step.

k	x_k	y_k	exact error
0	1.0	-1.0	3.73e-1
1	1.17021276595745	-1.45744680851064	0.08336627427070
2	1.20215882950670	-1.37676032192306	0.00267978768312
3	1.20316580709154	-1.37408348694971	0.00000295270977
4	1.20316696334641	-1.37408053424353	0.00000000000359
5	1.20316696334777	-1.37408053423994	0.00000000000000

TABLE 4.1: Newton Iterates for the System (4.5)

$$\text{iteration error} = \| \mathbf{s} \|_{l^\infty} . \tag{4.7}$$

By using the theory from Theorem (4.1.0.1) we see that quadratic convergence was achieved. The result of the theory is that we should have the relationship $\log(e^{\kappa}/e^{\kappa+1}) \approx$

2. Taking two different errors and comparing, $\log((2.95e-6)/(2.68e-3))/\log((2.68e-3)/(8.34E-2)) = 1.98 \approx 2$, we see the quadratic behaviour has been obtained. Thus, each iterate gains approximately twice as many significant digits as the previous iterate.

4.3. Variants of Newton's Method

The idea behind Newton's method was expressed in the beginning of Section 4. We have covered how the discretization is to be handled. Now we will cover the various forms of Newton's method that will be used in this paper.

4.3.1 Chord Method

The Chord method is one of the simplest modifications of Newton's method. In this method, the Jacobian is only computed once, using the initial guess. The convergence of the chord method is not as fast as Newton's method. Assuming that the initial guess is close enough to the solution, the convergence is only linear [9]. The following equation demonstrates how the initial vector of values is used to compute the Jacobian, whereas a new $\mathbf{s}$ is found for every new iterate κ:

$$\mathbf{s}^{\kappa} = \mathbf{DF}(\mathbf{U}^0)^{-1}\mathbf{F}(\mathbf{U}^{\kappa}). \tag{4.8}$$

4.3.2 Modified Chord Method

One way to utilize the benefits of having fewer operations which the Chord method boasts, without losing convergence properties of Newton's method, is to find a compromise between the two methods. This is why the Modified Chord method is created. The Modified Chord method converges superlinearly, as defined in Section (4.1). This balance between the quadratic rate of Newton's method and the linear rate of the Chord method makes sense, since the Modified Chord method is a balance between the two. Instead of computing the Jacobian only once, the modified chord method computes the Jacobian

every second iterate, or every third iterate, or every κ^{th} iteration. This method is sometimes referred to as the Shamanski (1967) method and can be described with the following transition from the current iterate to the new iterate [9] by:

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{U}^\kappa - \mathbf{DF}(\mathbf{U}^\kappa)^{-1}\mathbf{F}(\mathbf{U}^\kappa), \\ \mathbf{y}_{j+1} &= \mathbf{y}_j - \mathbf{DF}(\mathbf{U}^\kappa)^{-1}\mathbf{F}(y_j) \;\; 1 \le j \le m-1, \\ \mathbf{U}^{\kappa+1} &= \mathbf{y}_m. \end{aligned}$$

When $m = 1$, this is simply Newton's method. If $m = \infty$, then this becomes the Chord method.

4.3.3 Approximate Method

When the Jacobian cannot be computed symbolically, Newton's method may still be used but with a finite difference Jacobian approximation. In this paper we call this approach the Approximation method. This method may result in increased computational time and reduce reliability, when compared with Newton's method. The approximation is given by:

$$DF_{i,j} = \frac{\partial F_i}{\partial u_j} \approx \widetilde{DF}_{i,j}^{\Delta h} = \frac{\mathbf{F}_i(u_j + \Delta h) - \mathbf{F}_i(u_j)}{\Delta h} \tag{4.9}$$

The choice for Δh is consequential in ensuring that this method will achieve a close approximation to the solution. One way to create Δh would be to let Δh stay fixed, at some small value, for every iteration. It is important for Δh to be small so that the corresponding $\widetilde{\mathbf{DF}}$ is close to approximating $\mathbf{DF}$ but not so small as to introduce instabilities. If we would like this method to converge as rapidly as Newton's method, it would be better to let $\Delta h_\kappa = u^{(\kappa-1)} - u^{(\kappa)}$ for $\kappa = 1, 2, 3, ...$ If this is done, then the method is called the Secant method [8]. We chose, for this paper, to save the extra computation of Δh at each iteration and not use the Secant method.

The convergence rate of the Approximate method is superlinear. Using this method to produce the Jacobian of $\mathbf{F}$ creates an almost identical matrix, found by the analytical

method of computing the Jacobian. An example showing the validity of this statement is shown in the next section.

4.4. Comparison of Newton and Variants

If Newton's method has the best convergence rate of all the solvers explained before, when would the other solvers become useful? To help us answer this question, it is helpful to realize that the evaluation of **DF**, and the solving of the resultant linear system (4.2b), is the most time expensive part of all.

One technique we could use to reduce the computational time of computing the Jacobian every iteration, would be to take the computation and factorization of **DF** outside of the main loop. This is what the Chord method does, and although it has a smaller convergence rate, it can be a better method to use when the Jacobian is particularly expensive to compute.

As for the Modified Chord method, it allows the user to benefit both from the quicker convergence of Newton's method and the less expensive computational aspect of the Chord method. Assuming the initial iterate is sufficiently accurate, this method is best for large problems. This is due to the speed at which the algorithm will produce a solution (usually in only a few iterates [9]). Therefore, this method is a good candidate when we have a *good* guess, a large system, and the desire for a higher order convergence rate than the Chord method delivers.

If Δh is small enough, the Approximate method will produce a Jacobian that is very similar to the symbolically produced one. However, theory dictates that Δh should be on the order of $1e-7$or $1e-8$. This may be the users only choice, if they are determined to use Newton's method, to find a solution. In comparison with Newton's method, the Approximation method can add to the computation time as well as allowing for the possi-

bility of error propogation. Also, many more residual function evaluations are required (as seen in equation (4.8)). Although the convergence is only superlinear, it seems that this method of computing the Jacobian produces a scheme that is able to follow the partial differential equation without using derivative information.

5. RESULTS AND EXAMPLES

In this section, we show the application of scheme (3.2) to a generic nonlinear parabolic partial differential equation (2.1). First, we show how the Jacobian is calculated. Next, we confirm validity of the code on a linear case and then on a nonlinear case. Then, we analyze the convergence results and the computational time fluctuations between variants. Lastly, we show how the approximate Jacobian compares to the analytical one for the nonlinear case.

5.1. Calculation of the Jacobian

Here, we show how to calculate the Jacobian matrix **DF** when the residual is given by (3.2). The Jacobian here is a tri-diagonal matrix where DF(i,j) corresponds to the entry in the i^{th} row and the j^{th} column composed as follows:

$$\begin{aligned} DF(j,j) =& a'(u_j^n) + (\frac{1}{2})c(b'(\frac{u_j^n + u_{j-1}^n}{2})(u_j^n - u_{j-1}^n) - b'(\frac{u_j^n + u_{j+1}^n}{2})(u_{j+1}^n - u_j^n) + \cdots \\ & b(\frac{u_j^n + u_{j-1}^n}{2}) + b(\frac{u_j^n + u_{j+1}^n}{2})), \end{aligned} \tag{5.1a}$$

$$DF(j,j-1) = (\frac{1}{2})c(b'(\frac{u_j^n + u_{j-1}^n}{2})(u_j^n - u_{j-1}^n) - b(\frac{u_j^n + u_{j-1}^n}{2}), \tag{5.1b}$$

$$DF(j,j+1) = -(\frac{1}{2})c(b'(\frac{u_j^n + u_{j+1}^n}{2})(u_{j+1}^n - u_j^n) + b(\frac{u_j^n + u_{j+1}^n}{2}). \tag{5.1c}$$

For each iteration, this process will be repeated until the termination criteria has been met.

5.2. Confirm Validity of the Code

Before we can use the code to evaluate the model problem, we have to make sure that the code is working. Therefore, we test the code using various known u's with

corresponding known $a(u)$ and $b(u)$. With this information, the corresponding $f(x,t)$ is created, by first finding $\frac{\partial}{\partial t}(a(u))$ and then subtracting $\frac{\partial}{\partial x}(b(u)\frac{\partial u}{\partial x})$ from that value (see equation (2.1)).

In both of the next two test cases we check first the Approximation error. The Approximation error, $e(\Delta x)$, is given by

$$e(\Delta x) = \max_n \parallel U_j^n - u(x_j, t^n) \parallel_{l^\infty} . \tag{5.2}$$

This is synonymous with the exact error that was used in Section (4.2).

5.2.1 Test Case 1

In this first case we let

$$\begin{aligned} u(x,t) &= sin(\pi x) + x + t, \\ a(u) &= u, \qquad (5.3) \\ b(u) &= 1. \qquad (5.4) \end{aligned}$$

The initial condition is $u(x,0) = sin(\pi x) + x$. With the boundary conditions we have $u(0,0) = 0$. For any time $u(0,t) = t$ and $u(1,0) = 0$ and for any time, $u(1,t) = 1 + t$. This is the Dirichlet type of boundary conditions. Dirichlet boundary conditions occur when the value of the dependent variable is prescribed on the boundary. Thus, the right boundary value should be increasing with time. This formulation leads to the following derivation of $f(x,t)$:

$$f(x,t) = 1 + \pi^2 sin(\pi x), \;\; 0 \leq x \leq 1. \tag{5.5}$$

In TABLE (5.1), and later in TABLE (5.2), the time listed is the computational time taken by the algorithm. The *errortol* is the pre-determined error tolerance that we choose before running the algorithm. While the algorithm is running, the size of **s** is checked against this *errortol*. If the iteration error is still above this error tolerance, the algorithm continues to run. The theory of Newton's method suggests that as the error

tolerance shrinks, the convergence rate of the method will become closer to a quadratic rate. This is not apparent in Test Case 1, because the solution is so smooth and the problem is linear. The fact that the problem is linear should force the convergence to occur after just one or two steps. What we find is that for veritably any *errortol* the method converges after two iterations.

T = 1	$errortol = 1e-3$		$errortol = 1e-5$		$errortol = 1e-7$	
$\Delta t = (\Delta x)^2$	$e(\Delta x)$	time	$e(\Delta x)$	time	$e(\Delta x)$	time
$\Delta x = 1/5$	0.0319	0.015	0.0319	0.015	0.0319	0.015
$\Delta x = 1/10$	0.0083	0.079	0.0083	0.079	0.0083	0.079
$\Delta x = 1/20$	0.0021	0.505	0.0021	.501	.0021	0.504
$\Delta x = 1/40$	0.0005	3.221	.0005	4.360	.0005	4.335
$\Delta x = 1/80$	0.00013	49.40	0.00013	60.35	0.00013	65.40

TABLE 5.1: Test Case 1

5.2.2 Test Case 2

For this nonlinear case we let

$$\begin{aligned} u(x,t) &= xe^t + x, \\ a(u) &= 1 + u^2, \qquad (5.6) \\ b(u) &= 1 + u^3. \qquad (5.7) \end{aligned}$$

These choices for $a(u)$ and $b(u)$ satisfy the conditions that both functions are bounded away from zero. Thus, the corresponding $f(x,t)$ is as follows:

$$f(x,t) = 2(xe^t + x)(xe^t) - 3(xe^t + x)^2(e^t + 1)^2; \qquad (5.8)$$

For this test case, a more drastic array of *errortols* is utilized in hopes of observing more interesting results. The experimental evidence, shown in Table (5.2), shows that

the approximation error for this choice of u is affected very little by the different choices for the error tolerance. However, when the $errortol = 1e0$, the approximation error is smaller for $\Delta x = \frac{1}{5}$ than it is when $errortol = 1e-5$. This demonstrates that, with the tighter error tolerance, we do not necessarily come closer to the exact solution. When a large error tolerance is chosen, like $errortol = 1e0$, the Newton algorithm is not restricted to keeping the size between consecutive iterates small. Instead, the algorithm is allowed to concentrate on solving the nonlinear equation. Notice that the ratio between the approximation errors for $errortol = 1e0$ for $\Delta x = \frac{1}{5}$ and $\Delta x = \frac{1}{10}$ is equal to 3.58, whereas the ratio for when the $errortol = 1e-5$, for the same deltas, is 4.39.

T = 1, dt = $(dx)^2$	$errortol = 1e0$		$errortol = 1e-5$		$errortol = 1e-10$	
	$e(\Delta x)$	time	$e(\Delta x)$	time	$e(\Delta x)$	time
$\Delta x = 1/5$	0.0896	0.009	0.1136	0.022	0.1136	0.032
$\Delta x = 1/10$	0.0250	0.0511	0.0259	0.096	0.0259	0.125
$\Delta x = 1/20$	0.0064	0.328	0.0064	.532	.0064	0.780
$\Delta x = 1/40$	0.0016	3.015	0.0016	4.546	0.0016	6.09
$\Delta x = 1/80$	0.000407	54.980	.00039	67.46	0.00040	80.82

TABLE 5.2: Test Case 2

The number of iteration steps that Newton's method takes to converge, is also affected. For $errortol = 1e0$ it takes only one iteration for each of the Δx values. For $errortol = 1e-5$ and $errortol = 1e-10$, the iteration count increases between one and three for each of the Δx's.

In figure (5.1) all four solvers were run on Test Case 2. To ensure we could see a difference in values between the four, the error tolerance is taken to be large, while the maximum iteration count is small (error tolerance is 1e-1, maximum iteration count is 2). Also, the $\Delta x = 1/10$ while the $\Delta t = 10(\Delta x)^2$. This figure shows that the Approximate

method, with a $\Delta h = 1e-7$ had almost the same approximation error as that of the Newton method. Both were lower than the Chord or Modified Chord methods.

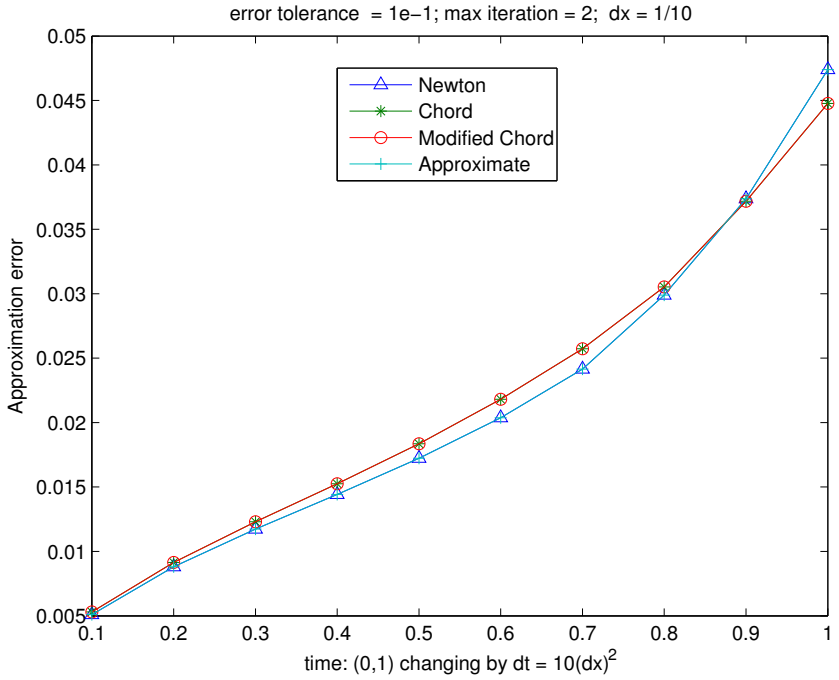

FIGURE 5.1: Comparing the Approximation Error of the Four Solvers, for Test Case 2

Figure (5.2) shows various ways in which we can check convergence, timing, approximation error, iteration error, and residual error for whichever $u(x,t)$ we decide to use. The $u(x,t)$ that is portrayed here is Test Case 2, for $\Delta x = \frac{1}{20}$, $\Delta t = (\Delta x)^2$ and $errortol = 1e-12$. Residual error is found by

$$\text{residual error} = \| \mathbf{F} \|_{l^\infty} . \tag{5.9}$$

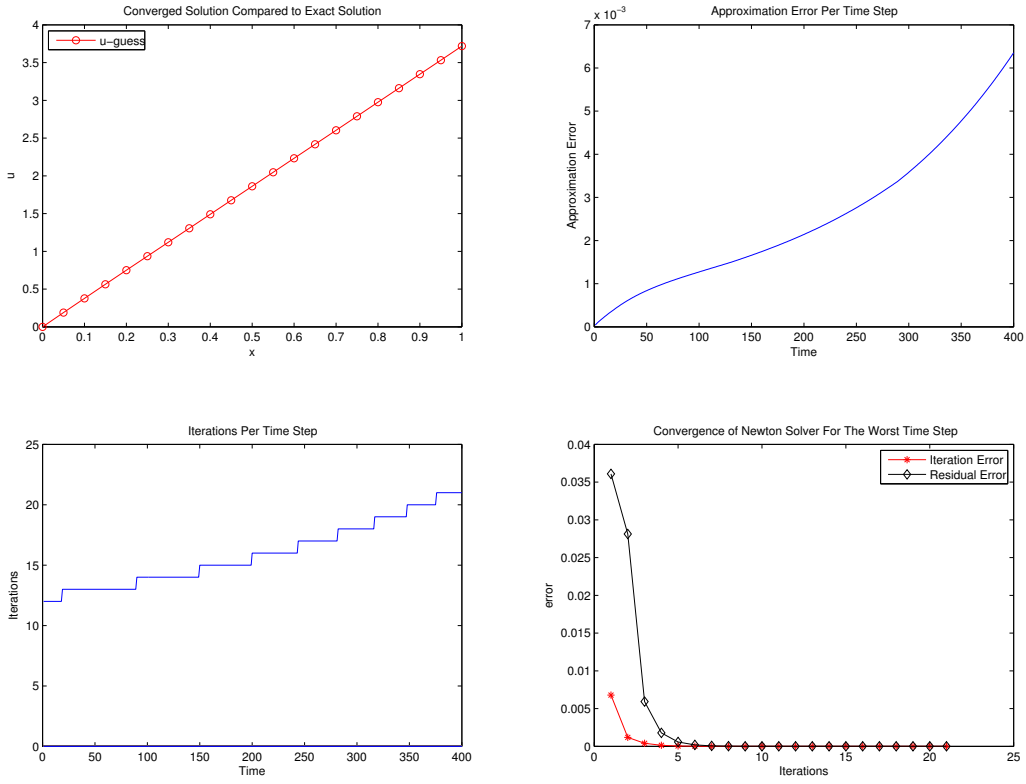

FIGURE 5.2: Newton's Method Used to Solve Test Case 2

5.3. Convergence of the Discretization Scheme: Results

Before we analyze the error, we must mention the fact that, in this paper, we are using absolute convergence criteria. Absolute convergence criteria means that there exists a fixed error tolerance to which the norm of the residual is compared.

If the error tolerance is too small (smaller than the roundoff error of the computer), then it may be impossible to achieve convergence, or the numerical method will take an inordinately long time to reach convergence in one time step. If we set the error tolerance to a small value (away from the roundoff error), with the desire of attaining a higher level of accuracy, we may actually be obtaining the following effect. Since the method was

forced to keep iterating, inside of one time step, until the size between iterates was smaller than the error tolerance, the converged solution may be further from the exact solution than if a less severe error tolerance is specified.

To ensure the proper convergence results in Newton's method, while satisfying a prescribed error tolerance, a *while* loop is utilized to corral multiple termination criteria together. One of the termination factors for the *while* loop is the iteration error, or relative nonlinear residual, while the other is the iteration count. The relative nonlinear residual is computed by taking the maximum difference between the newly computed iterate and the previous iterate. Including the iteration count as part of the termination criteria is important in order to keep the code from running indefinitely. If the error tolerance, combined with very small solution values, supercedes the computer's round off error, the other termination criteria will stop the process.

As long as the standard assumptions (explained in Section (4.1)) are verified, we may assume that as this iteration error decreases, we are converging upon the desired solution. This is true if the initial choice is adequately close to the exact solution [9].

Both TABLE's (5.1) and (5.2) give us enough information to see that this numerical method is producing the correct order of approximation $O((\Delta x)^2$, as discussed in Section 3. The error is essentially squared at each step. To examine, let $e_1(\Delta x_1)$ be the error (equation (5.2)) found with an initial Δx_1 and $e_2(\Delta x_2)$ be the error found at the next iterate. Theory says that $e_1(\Delta x_1) \approx (\Delta x_1)^2$. Using this information, we cut in half the value of Δx_1 so that $\Delta x_2 = \frac{1}{2}\Delta x_1$ to see if our data gives us this result. Now, $e_2(\Delta x_2) = e_2(\frac{1}{2}\Delta x_1)$ should also be $e_2 \approx (\frac{1}{2}\Delta x_1)^2$ using the information from both TABLE's (5.1) and (5.2) we see $\frac{e_1(\Delta x_1)}{e_2(\Delta x_2)} \approx 4$ is satisfied.

5.4. Timings of Solvers

To test the variants of Newton's method we look at how long each variant takes to solve Test Case 2, seen in TABLE (5.3).

Solver	Time	max itc
Newton	0.1237/ 0.5708	4/ 3
Chord	0.0930/ 0.4597	7/ 4
Modified Chord	0.0766/ 0.5519	4/ 4
Approximate	1.7910/ 20.5078	4/ 3

TABLE 5.3: Test Case 2: Solver Ccomputational Ttime (s) for $\Delta x = \frac{1}{20}$ vs. $\frac{1}{40}$, $\Delta t = 10(\Delta x)^2$.

In TABLE (5.3) we compare the time it takes for the four solvers to solve Test Case 2 for $errortol = 1e-7$, and $\Delta h = 1e-7$. Also, *max itc* represents the maximum iteration count over all the time steps. The approximation error (5.2) for all four solvers was 0.0105 and 0.0026, for $\Delta x = 1/20$ and $\Delta x = 1/40$ respectively, showing the proper quadratic convergence ($0.0105 \div 0.0026 \approx 4$). The *maxstep* (showing the time step corresponding to where the maximum approximation error occurs) was at 40 and 160 respectively.

$\Delta x = \frac{1}{100}$	itc	Time	maxerr	maxstep
$\Delta t = \frac{1}{10}$	5	0.1626	0.0177	10
$\Delta t = \frac{1}{100}$	4	1.015	0.0021	100
$\Delta t = \frac{1}{1000}$	3	10.4919	0.000420	1000
$\Delta t = \frac{1}{10000}$	3	198.2244	0.000253	10000

TABLE 5.4: Timing for Newton's Method for a Fixed Δx

T = 1	Newton's Method			Chord Method		
$\Delta t = (\Delta x)^2$	error	time	max itc	error	time	max itc
$\Delta x = \frac{1}{5}$	0.1136	0.0512	4	0.1136	0.0362	8
$\Delta x = \frac{1}{10}$	0.0259	0.1271	4	0.0259	0.1039	5
$\Delta x = \frac{1}{20}$	0.0064	0.7322	3	0.0064	0.0628	4
$\Delta x = \frac{1}{40}$	0.0016	6.1088	3	0.0016	4.2406	3

TABLE 5.5: Newton's Method and Variants for Test Case 2

T = 1	Approximate Method			Modified Chord Method		
$\Delta t = (\Delta x)^2$	error	time	max itc	error	time	max itc
$\Delta x = \frac{1}{5}$	0.1136	0.1268	4	0.1136	0.0247	5
$\Delta x = \frac{1}{10}$	0.0259	1.0408	4	0.0259	0.1127	4
$\Delta x = \frac{1}{20}$	0.0064	13.1262	3	0.0064	0.7176	4
$\Delta x = \frac{1}{40}$	0.0016	202.90	3	0.0016	4.2406	3

TABLE 5.6: Newton's Method and Variants for Test Case 2

From Section (3.1.1), we would expect that the finite difference algorithm would take longer to converge if we kept Δx the same and changed only Δt. This is shown in TABLE (5.4). Thus, as the temporal grid is shrunk, becoming several orders in magnitude smaller than the spatial grid, Newton's method takes a longer amount of time to reduce the iteration error to a value less than the pre-determined tolerance. It takes a little over 1000 times longer to run the numerical method as Δt is reduced by 10000. The error tolerance that is used is $1e-9$.

TABLE's (5.5) and (5.6) display some of the pertinent data one can use to evaluate the usefullness of one variant over another.

5.4.1 Using Approximate Jacobian

As promised in Section 4.4.3, the Jacobian's for both the Newton method and the Approximate method are displayed below in order to show how similar the two are. In the following, the matrices $\widetilde{\mathbf{DF}}$ represents the Approximate Jacobian and $\mathbf{DF}$ represents the analytical Jacobian. The parameters for this computation are $\Delta x = \frac{1}{5}$, $\Delta t = (\Delta x)^2$, *errortol*$= 1e-9$, and $\Delta h = 1e-1$ with the resulting data from Test Case 2.

$$\widetilde{\mathbf{DF}} = \begin{bmatrix} 1.0000 & 0 & 0 & 0 & 0 & 0 \\ -0.8425 & 4.2788 & -4.1996 & 0 & 0 & 0 \\ 0 & -1.2493 & 11.2715 & -11.6361 & 0 & 0 \\ 0 & 0 & -3.9896 & 27.5355 & -26.3835 & 0 \\ 0 & 0 & 0 & -11.3987 & 58.5055 & -50.9901 \\ 0 & 0 & 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{5.10}$$

$$\mathbf{DF} = \begin{bmatrix} 1.0000 & 0 & 0 & 0 & 0 & 0 \\ -0.8448 & 4.6007 & -4.4868 & 0 & 0 & 0 \\ 0 & -1.4118 & 12.1104 & 12.2913 & 0 & 0 \\ 0 & 0 & -4.4413 & 29.1923 & 27.5734 & 0 \\ 0 & 0 & 0 & -12.3004 & 61.3173 & -52.8844 \\ 0 & 0 & 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{5.11}$$

6. APPLICATION

In Section 5, we validated our code. We have shown that our code gives the right order of approximation and the correct order of convergence of Newton's method. For the former we showed this on a test case where the solution $u(x,t)$ is known.

In this section, we experiment with a problem where $u(x,t)$ is not known. This is a typical case in applications.

6.1. Model Application Problem

Let us look at a single phase slightly compressible fluid flow equation of the form of equation (2.1). Consider $a(u)$, $b(u)$ as in TABLE (2.2).

Here, we let $\rho_{ref} = 1$, $p_{ref} = 1$, $k = 10^{-2}$, and $\phi = 10^{-1}$. We use a source

$$f(x,t) = \begin{cases} 1, & 0.4 \leq x \leq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

simulating an injection well.

We solve the problem using Scenarios 1, 2, and 3. In addition to solving the nonlinear and the linearized cases, we also look at what happens when the error tolerance is large in Newton's step when applied to Scenario 2. This case is called the *truncated* nonlinear method.

For the nonlinear problems we used an error tolerance of $1e-12$, which is a fairly strong restriction for the termination criteria. This same error tolerance was used for the linearized case, however, this is unimportant. For the truncated version the error tolerance was allowed to be large, $1e-1$. The results for all three scenarios were compared, to one another, for different values of the compressibility coefficient $c = 1e-0$, $c = 1e-4$, and $c = 1e-8$, recall its use in equation (2.6).

First, we look at a compressibility coefficent of $c = 1e - 0$ (see Figures (6.1)-(6.6)). The difference between the nonlinear and the linearized representations of the diffusion equation is on the order of $1e - 1$.

Next, the compressibility coefficient is set for $c = 1e - 4$, as it would be for oil (see Figures (6.7)-(6.12)). The difference between the nonlinear and the linearized representations of the diffusion equation is on the order of $1e - 7$.

Lastly, the compressibility coefficient is $c = 1e - 8$, as it would be for water (see Figures (6.13)-(6.18)). The difference between the nonlinear and linearized representation is negligible (on the order of $1e - 15$).

For all three compressibility coefficients, the difference between the nonlinear and the truncated methods is drastically smaller than the nonlinear-linearized difference. This information leads us to believe that in this model problem, the truncated problem gives almost the same solution as the nonlinear problem. This prompts us to contemplate why we would iterate Newton to convergence, if we could get away with a truncated method instead. Clearly the linearized method is not a good enough estimate. One idea to make it better would be to take one more term from the Taylor expansion of equation (2.6). Now the linearized method wouldn't be linear anymore, it would be quadratic, but it could give a better approximation to the nonlinear one. This is something that could be challenged in the future.

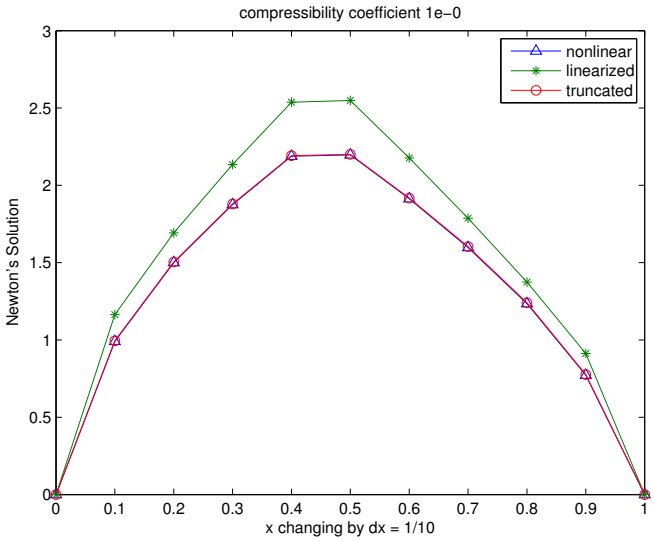

FIGURE 6.1: Compressibility Coefficient 1e-0, $\Delta x = 1/10$ with $\Delta t = 10(\Delta x)^2$

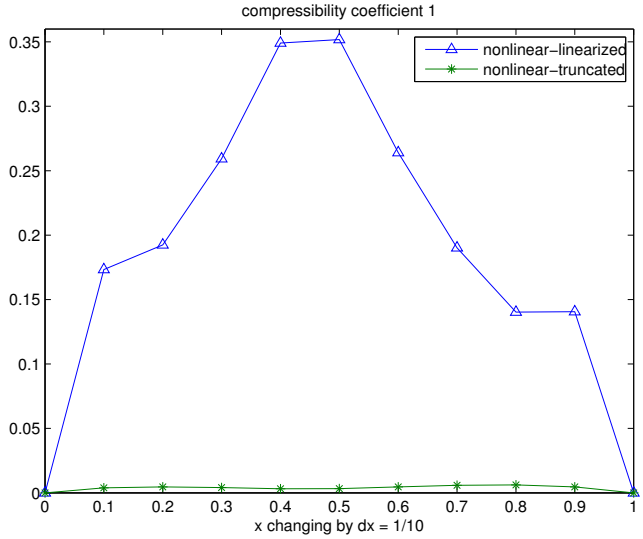

FIGURE 6.2: The Difference Between the Nonlinear-Linearized Case and Nonlinear-Truncated Case: c = 1e-0, $\Delta x = 1/10$ with $\Delta t = 10(\Delta x)^2$

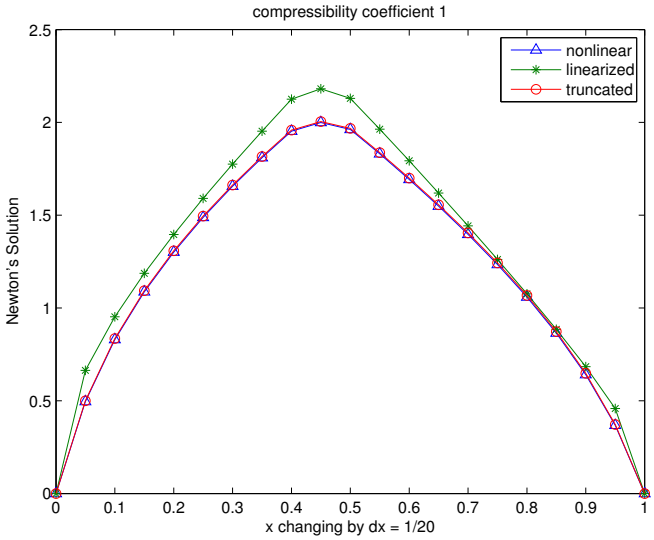

FIGURE 6.3: Compressibility Coefficient 1e-0, $\Delta x = 1/20$ with $\Delta t = 10(\Delta x)^2$

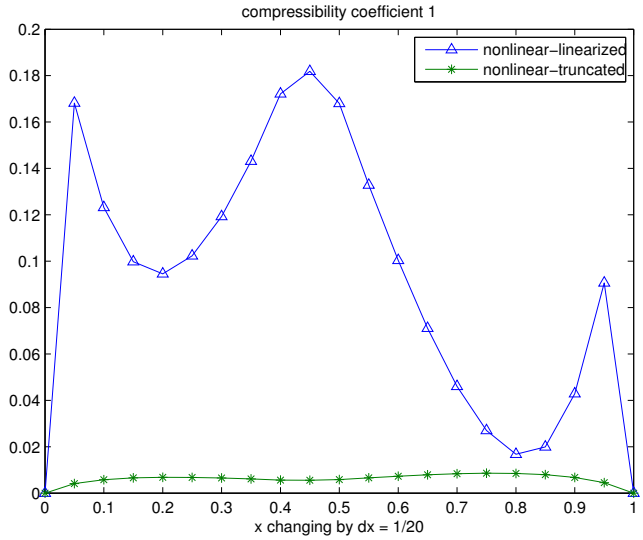

FIGURE 6.4: The Difference Between the Nonlinear-Linearized Case and Nonlinear-Truncated Case: $c = 1e - 0$, $\Delta x = 1/20$ with $\Delta t = 10(\Delta x)^2$

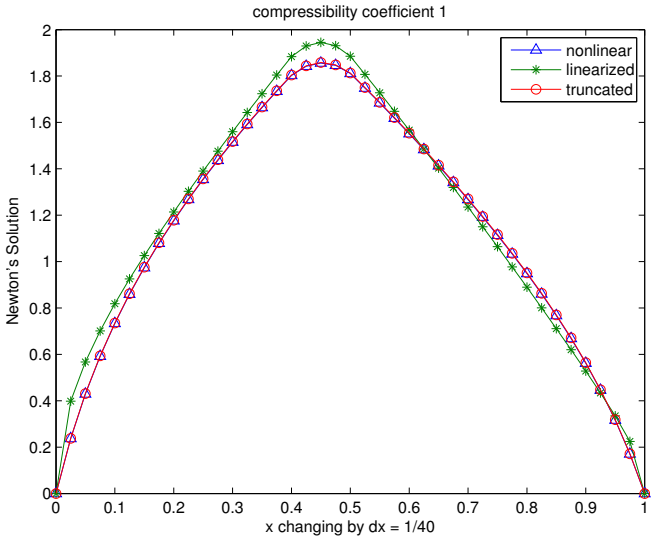

FIGURE 6.5: Compressibility Coefficient 1e-0, $\Delta x = 1/40$ with $\Delta t = 10(\Delta x)^2$

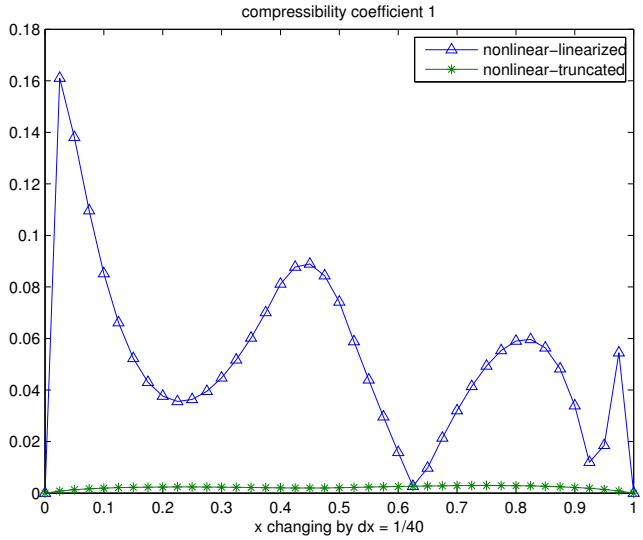

FIGURE 6.6: The Difference Between the Nonlinear-Linearized Case and Nonlinear-Truncated Case: $c = 1e - 0$, $\Delta x = 1/40$ with $\Delta t = 10(\Delta x)^2$

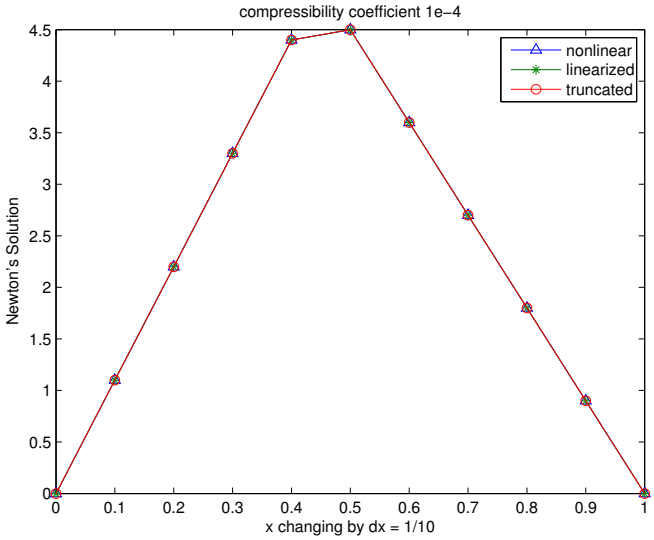

FIGURE 6.7: Compressibility Coefficient 1e-4, $\Delta x = 1/10$ with $\Delta t = 10(\Delta x)^2$

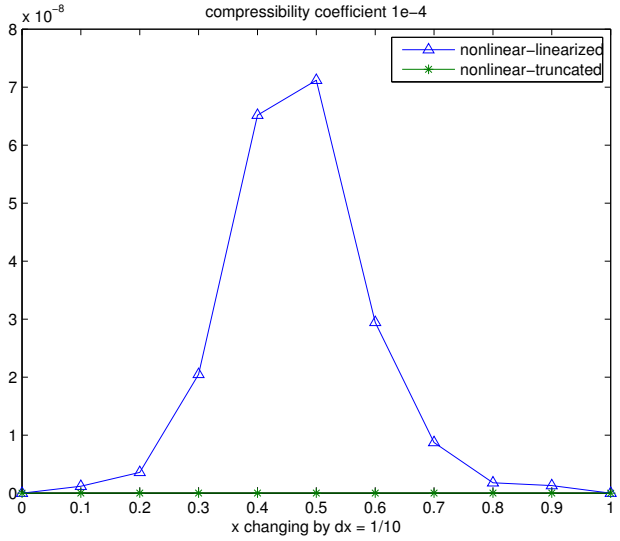

FIGURE 6.8: The Difference Between the Nonlinear-Linearized Case and Nonlinear-Truncated Case: $c = 1e-4$, $\Delta x = 1/10$ with $\Delta t = 10(\Delta x)^2$

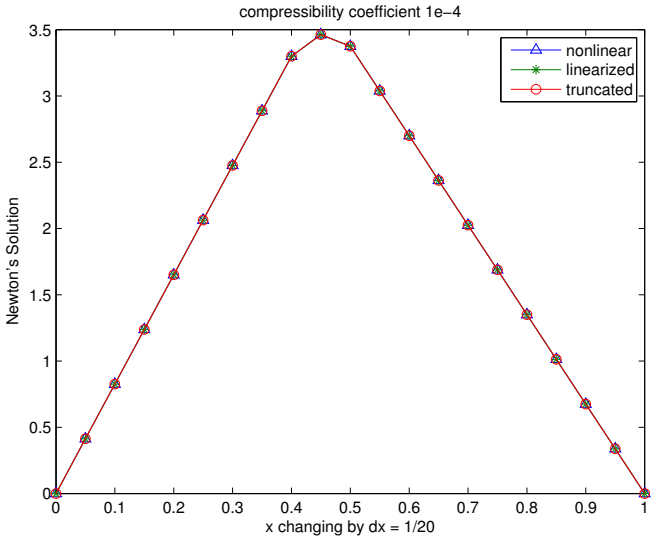

FIGURE 6.9: Compressibility Coefficient 1e-4, $\Delta x = 1/20$ with $\Delta t = 10(\Delta x)^2$

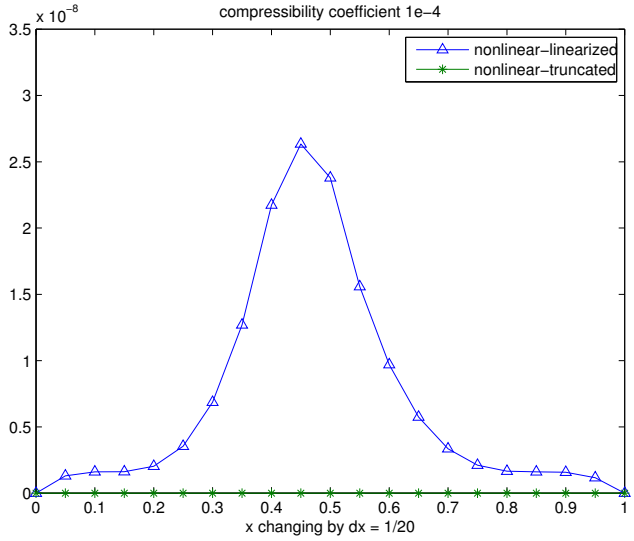

FIGURE 6.10: The Difference Between the Nonlinear-Linearized Case and Nonlinear-Truncated Case: $c = 1e-4$, $\Delta x = 1/20$ with $\Delta t = 10(\Delta x)^2$

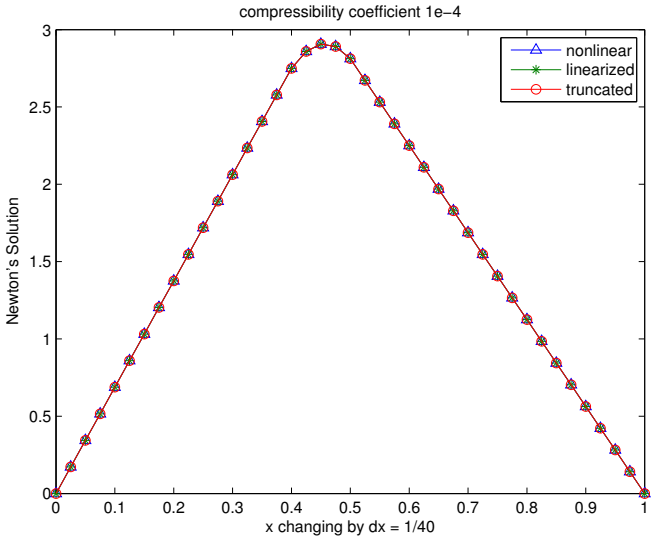

FIGURE 6.11: Compressibility Coefficient 1e-4, $\Delta x = 1/40$ with $\Delta t = 10(\Delta x)^2$

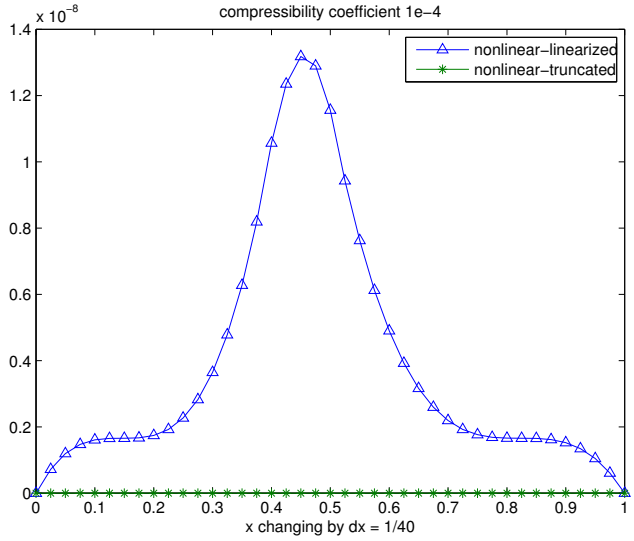

FIGURE 6.12: The Difference Between the Nonlinear-Linearized Case and Nonlinear-Truncated Case: $c = 1e-4$, $\Delta x = 1/40$ with $\Delta t = 10(\Delta x)^2$

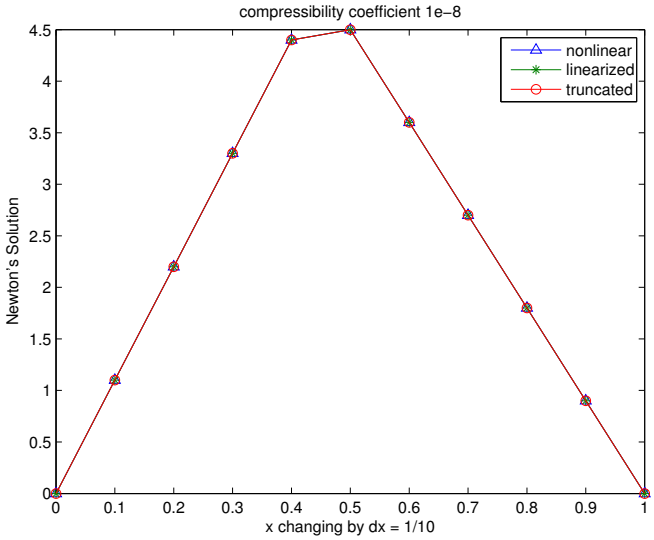

FIGURE 6.13: Compressibility Coefficient 1e-8, $\Delta x = 1/10$ with $\Delta t = 10(\Delta x)^2$

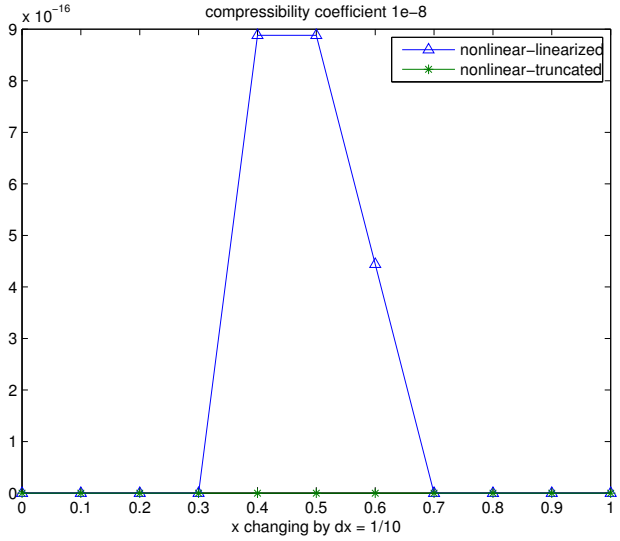

FIGURE 6.14: The Difference Between the Nonlinear-Linearized Case and Nonlinear-Truncated Case: $c = 1e-8$, $\Delta x = 1/10$ with $\Delta t = 10(\Delta x)^2$

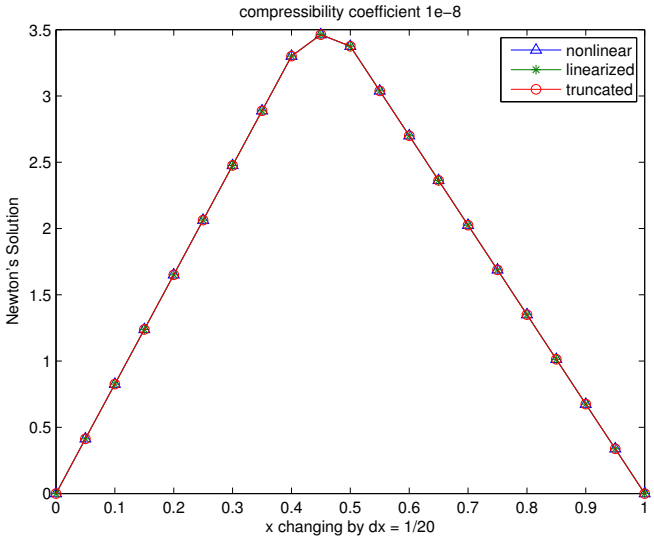

FIGURE 6.15: Compressibility Coefficient 1e-8, $\Delta x = 1/20$ with $\Delta t = 10(\Delta x)^2$

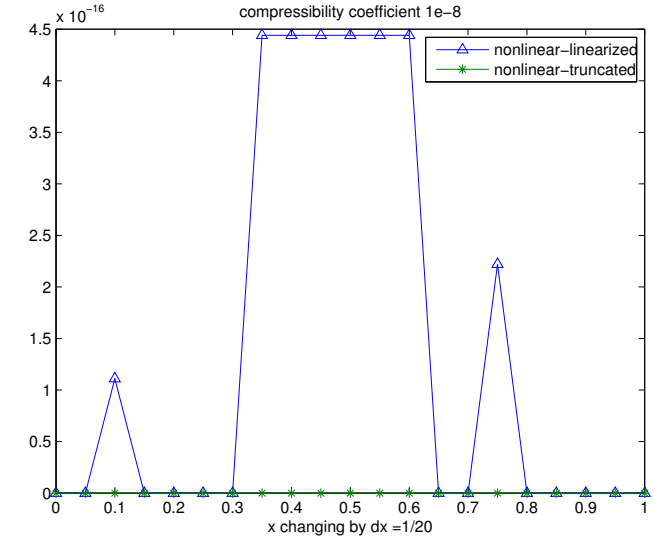

FIGURE 6.16: The Difference Between the Nonlinear-Linearized Case and Nonlinear-Truncated Case: $c = 1e-8$, $\Delta x = 1/20$ with $\Delta t = 10(\Delta x)^2$

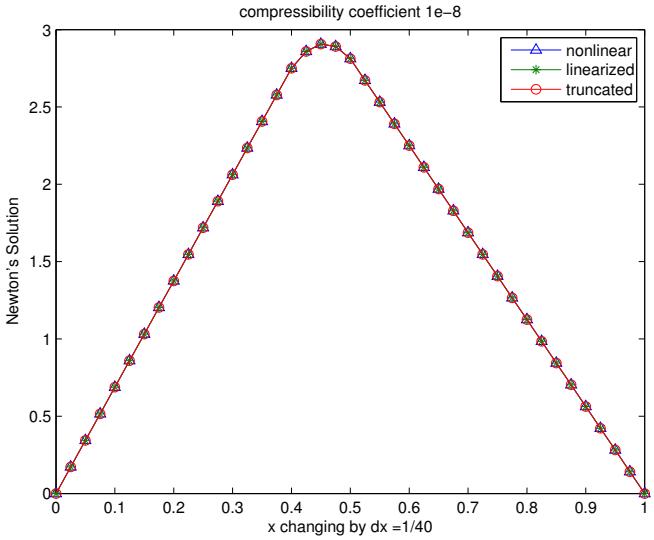

FIGURE 6.17: Compressibility Coefficient 1e-8, $\Delta x = 1/40$ with $\Delta t = 10(\Delta x)^2$

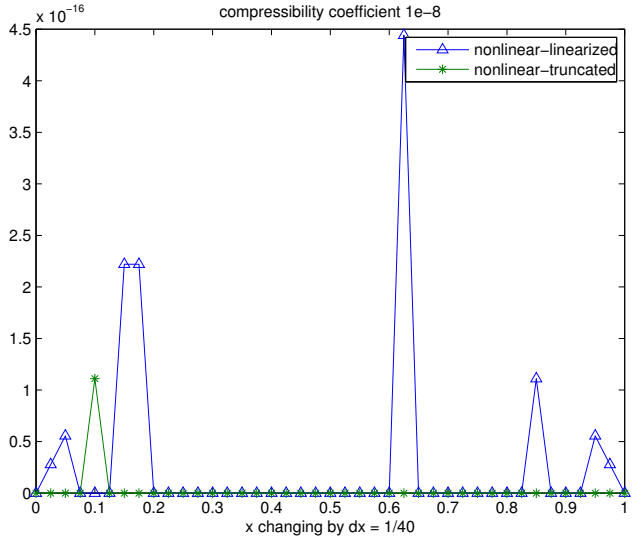

FIGURE 6.18: The Difference Between the Nonlinear-Linearized Case and Nonlinear-Truncated Case: $c = 1e-8$, $\Delta x = 1/40$ with $\Delta t = 10(\Delta x)^2$

7. THE TWO PHASE FLUID FLOW MODEL

In Section (2.1), the conservation equations for mass and momentum were discussed for a single fluid phase, and both nonlinear functions $a(u)$ and $b(u)$ were positive and bounded away from zero.

One application we are particularly interested in, for future research, is the interaction between two fluids. This interaction produces a truly nonlinear degenerate problem where $a(u)$, or $b(u)$, or both, are no longer guaranteed to be bounded away from zero.

We do not yet have a numerical method implemented for this degenerate problem, therefore we only discuss the model.

7.1. Conservation Equations

First, we define some additional conservation equations for two fluids. One phase (water) wets the porous medium more than the other (oil or air) and is called the wetting phase. The other phase is called the nonwetting phase [3]. We will be looking at the water and air case.

7.1.1 Mass Conservation for a Two-Phase System

First, we define *saturation* [5], [3].

Definition 7.1.1.1 *Saturation of fluid a,w is defined as*

$$\begin{aligned} S_a &= \frac{\textit{volume of void space filled with air}}{\textit{volume of void space}}, \\ S_w &= \frac{\textit{volume of void space filled with water}}{\textit{volume of void space}}. \end{aligned} \tag{7.1}$$

Of course, $0 \leq S_a, S_w \leq 1$. We assume that the voids in the porous medium under observation contain only water and air. Because the two fluids fill the voids jointly we

may assume that [4]

$$S_a + S_w = 1. \tag{7.2}$$

Therefore, if we know one value, we know the other. This is the first equation in a series of four.

In Section 2 we discussed that many equations are required to create an environment that is able to handle the number of unknowns found in our model problem. In an attempt to find the correct number of equations to describe this environment, we state two mass conservation equations.

$$Air$$

$$\frac{\partial}{\partial t}(\phi \rho_a S_a) - \nabla(\rho_a k k_a(S_w) \nabla p_a) = f_a \tag{7.3}$$

$$Water$$

$$\frac{\partial}{\partial t}(\phi \rho_w S_w) - \nabla(\rho_w k k_w(S_w) \nabla p_w) = f_w \tag{7.4}$$

Here, k_a and k_w are the relative permeabilities of air and water respectively, which extend Darcy's law to a multiphase case.

We have identified four unknowns, S_a, S_w, p_a, p_w. Thus we need four equations. So far, we have three equations (7.2), (7.3), and (7.5), so we need one more equation. This final equation describes the capillary pressure. Assume the surface tension between the 2-phases is fixed but still dependent on S_w. The pressure difference between the two phases is given by the capillary pressure p_c, dependent upon the saturation fraction of water S_w [4].

$$p_c(S_w) = p_a - p_w. \tag{7.5}$$

Due to the curvature and surface tension of the interface between the two phases, $p_w \leq p_a$, which implies $p_c \geq 0$ [3], [7].

One of the more explicit ways to correlate the two-phase air-water system is by the van-Genuchten soil water retention model, in which

$$p_c(S_e) = \frac{1}{\alpha}(S_e^{-1/m} - 1)^{1/n}, \tag{7.6}$$

and

$$S_e = \frac{S_w - S_{wr}}{1 - S_{wr}}. \tag{7.7}$$

In these correlations, α, n, and m are van-Genuchten (1980) parameters. Usually, the parameter m is defined by $m = 1 - \frac{1}{n}$. The term S_{wr} is the *residual* saturation for water as described in [7]. The term S_e is the effective saturation, defined by equation (7.7), with $S_{wr} \leq S_w \leq 1$. We can define the relative permeability of both the nonwetting and the wetting phases with respect to effective saturation as [7],

$$k_w = S_e^{\frac{1}{2}} \left[1 - \left(1 - S_e^{\frac{1}{m}}\right)^m\right]^2,$$

and

$$k_a = (1 - S_e)^{\frac{1}{3}} \left[1 - S_e^{\frac{1}{m}}\right]^{2m}.$$

The graphs of k_w and k_a with respect to S_e are shown in Figure (7.1). Figure(7.1) shows how there is a very limited domain in which both phases are mobile [7].

See Figure (7.2) for an image of how capillary pressure changes with respect to an increase in water saturation. The physical conditions and parameters are all from [7]. We show graphs for two different sets of parameters. The reason we are interested in seeing the, capillary pressure versus effective saturation, curves for two different sets of parameters is so we can talk about adaptivity. If we were able to solve this degenerate problem we might want options for which nonlinear p_c we input into our code. Having flexibility might enable the numerical solver to be able to solve a system with modified parameters which behaves very similarly to the original, which was more difficult to solve.

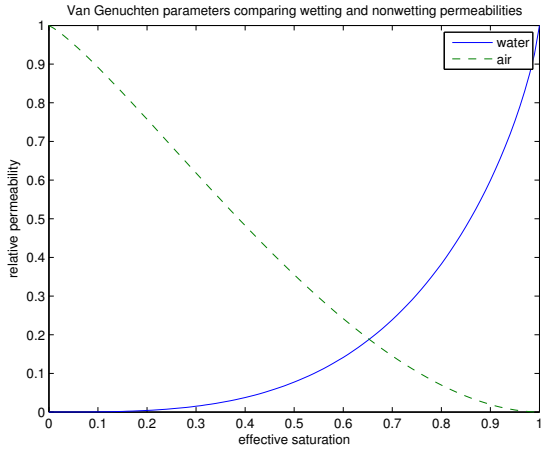

FIGURE 7.1: k_w Versus k_a with Respect to S_e

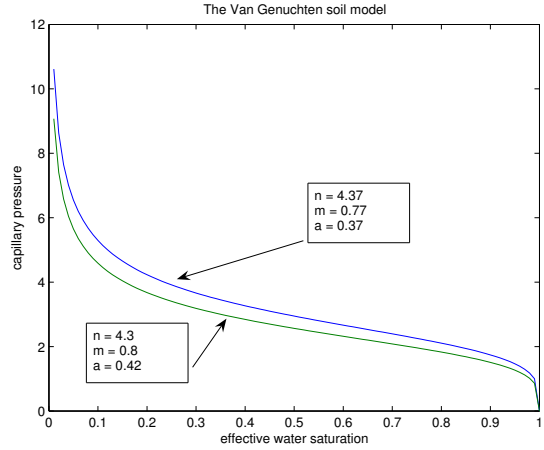

FIGURE 7.2: p_c–S_w Relation

7.1.2 Richards Equation

A partial differential equation that describes the vertical moisture flow, as water pressure changes, for unsaturated flow is Richards equation. Richards equation is a simplification of the full 2-phase equations presented in the previous section. It assumes that the air pressure is a constant,

$$p_a \equiv \text{constant} = 0 \tag{7.8}$$

$$\rho_a \approx 0 \tag{7.9}$$

and water is incompressible,

$$\rho_w \equiv \text{constant} \quad (c_w = 0). \tag{7.10}$$

Using (7.8) we can simplify (7.5) as,

$$\text{First:} - p_w = p_c(S_w) \tag{7.11}$$

$$\text{Implies:} S_w = p_c^{-1}(-p_w) \tag{7.12}$$

With the assumption in equation (7.8), we know that $\nabla p_a = 0$. With this information, the mass conservation equation for water (equation (7.4)) can be rewritten as an equation for S_w or for p_w. In addition, using (7.10) and (7.11), we can simplify equation (7.4) to get:

$$\frac{\partial}{\partial t}(\phi S_w) + \nabla(k k_w p_c' \nabla S_w) = \frac{f_w}{\rho_w}. \tag{7.13}$$

Equation (7.13) is a parabolic nonlinear degenerate equation solved for S_w as the primary unknown. To understand why this PDE is degenerate we look at the relationship between $k_w p_c'$ and S_w, Figure (7.3). We notice the term $k_w p_c$' is not significantly bounded from zero, as theory dictates. This is the reason that equation (7.13) is degenerate. Notice also how values of $k_w p_c$' explode when close to 1. This has to do with how almost vertical the graph in Figure (7.2) gets near 1. In equation (7.14) the term $k_w p_c$' is computed for the van-Genuchten model.

$$k_w p_c' = \left[S_e^{\frac{1}{2}} \left[1 - \left(1 - S_e^{\frac{1}{m}} \right)^m \right]^2 \right] \left[\frac{1}{\alpha n} \left(S_e^{-\frac{1}{m}} - 1 \right)^{(1-n)/n} \left(-\frac{1}{m} S_e^{(-1-m)/m} \right) \right]. \tag{7.14}$$

Equation (7.13) can be rewritten using p_w as the primary unknown as follows:

$$\frac{\partial}{\partial t}(\phi p_c^{-1}(-p_w)) - \nabla(k k_w (p_c^{-1}(-p_w)) \nabla p_w) = \frac{f_w}{\rho_w}. \tag{7.15}$$

However, we note that p_c^{-1} may not exist, which would render (7.15) invalid.

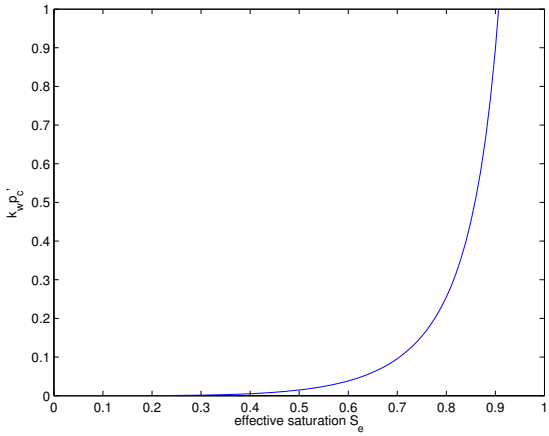

FIGURE 7.3: Nonlinear Diffusion Coefficient in Richards' equation.

8. CONCLUSION

In this paper, we have addressed a 1-D nonlinear parabolic partial differential equation with applications to fluid flow in porous media. We have shown different ways to represent and numerically approximate the model. To solve the problem numerically, we have applied finite differences and explored Newton's method and its variants.

We have seen that behaviour of Newton's method for different discretizations can suggest ways to linearize, truncate, or adapt the nonlinear model.

For continued research, we would like to expand this study and extend it to Richards equation and other numerical methods such as finite elements. Continued research into these areas may lead to savings in computational time.

BIBLIOGRAPHY

1. Kendall Atkinson and Weimin Han. *Elementary Numerical Analysis*. John Wiley and Sons, Inc., New York, USA, 2004.

2. Dietrich Braess. *Finite Elements*. Cambridge University Press, New York, USA, 1997.

3. Zhangxin Chen and Todd Arbogast. Lecture notes from nsf-cbms conference. In *Mathematical and Numerical Treatment of Fluid Flow and Transport in Porous Media*, Las Vegas, USA, 2006. University of Nevada Las Vegas.

4. Zhangxin Chen, Guanren Huan, and Yuanle Ma. *Computational Methods for Multiphase Flows in Porous Media*. Society for Industrial and Applied Mathematics, New York, USA, 2006.

5. Ghislain de Marsily. *Quantitative Hydrogeology*. Academic Press, Inc., Florida, USA, 1986.

6. Ronald B. Guenther and John W. Lee. *Partial Differential Equations of Mathematical Physics and Integral Equations*. Dover Publications, Inc., New York, USA, 1996.

7. Rainer Helmig. *Multiphase Flow and Transport Processes in the Subsurface*. Springer, Berlin, D, 1997.

8. Myron B. Allen III and Eli L. Isaacson. *Numerical Analysis for Applied Science*. John Wiley and Sons, Inc., New York, USA, 1998.

9. C.T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1995.

APPENDIX

A Matlab Code

```
function [x,u_new] = newton(x_left,x_right,T,M,choice,dh,errortol,dt)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Finite Difference approximation of the parabolic initial two-point
%boundary value diffusion problem.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Shannon Biederman, 2/8/07
%Copyright Department of Mathematics, Oregon State University
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


dx = (x_right-x_left)/(M);
if nargin < 8
    dt = 10*(dx)^2;        %% For optimal convergence.
end
c = dt/((dx)^2);           %% Convenient parameter.
t = dt:dt:T; t = t';  N = size(t,1);
x = (x_left:dx:x_right);


%%% Initialization %%%
maxit                   = 100;
error_itc               = zeros(N,maxit);
error_residual          = zeros(N,maxit);
```

```
iterations              = zeros(N);
error_approximation     = zeros(N,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Initialize time-steping, u_old is the previous time step. %%%
u_init = u(x',0);          %% If u(x,t) is given.
u_old = u_init;
%u_old = zeros(M+1,1);     %% If we don't know u(x,t).


tic
%%% Time Loop Begins here %%%
for n = 1:N
    u_guess = u_old;       %% u_guess is a good guess for the Newton itc.
    itc = 0; error = inf ;


    while error > errortol & itc < maxit
        itc = itc+1;
%-------------------------------------------------------------------------
if choice == 1
    method = 'Newton';
    F  = Residual(u_old,u_guess,c,t(n),x,M,dt);
    DF = Jacobian(u_old,u_guess,c,t(n),x,M,dt);


%-------------------------------------------------------------------------
elseif choice==2
    method='Chord Method';
```

```
    F = Residual(u_old,u_guess,c,t(n),x,M,dt);
    if itc == 1
        DF = Jacobian(u_old,u_old,c,t(n),x,M,dt);
    end;

%-------------------------------------------------------------------------
elseif choice == 3
    method = 'Modified Chord';
    if mod(itc,2) == 1,
        F = Residual(u_old,u_guess,c,t(n),x,M,dt);
        DF = Jacobian(u_old,u_guess,c,t(n),x,M,dt);
    else
        F = Residual(u_old,u_guess,c,t(n),x,M,dt);
    end

%-------------------------------------------------------------------------
elseif choice == 4
    method = 'Approximate Method';
    e = zeros(M+1,1);
    u_corrected = zeros(M+1,1);
    F = Residual(u_old,u_guess,c,t(n),x,M,dt);
    for j = 2:M
        e(j-1) = 1;
        u_corrected = u_guess+dh.*e;
        e(j-1) = 0;
        FF = Residual(u_old,u_corrected,c,t(n),x,M,dt);
```

```
        DF(j,j-1) = (1/dh).*(FF(j)-F(j));
    end
    for j = 2:M
        e(j) = 1;
        u_corrected = u_guess+dh.*e;
        e(j) = 0;
        FF = Residual(u_old,u_corrected,c,t(n),x,M,dt);
        DF(j,j) = (1/dh).*(FF(j)-F(j));
    end
    for j = 2:M
        e(j+1) = 1;
        u_corrected = u_guess+dh.*e;
        e(j+1) = 0;
        FF = Residual(u_old,u_corrected,c,t(n),x,M,dt);
        DF(j,j+1) = (1/dh).*(FF(j)-F(j));
    end
        DF(1,1) = 1;       %%Dirichlet conditions.
        DF(M+1,M+1) = 1;
end

%-------------------------------------------------------------------------
%%% Newton Step %%%
s = DF(1:M+1,1:M+1) \ F(1:M);   %%s is the iteration step
u_new_guess = u_guess - s;
u_guess = u_new_guess;

%%% Error Calculation %%%
```

```
error_residual(n,itc) = norm(F,inf);
error_itc(n,itc) = norm(s,inf);
error = error_itc(n,itc);       %%termination criteria

    end                         %%end Newton loop

    iterations(n) = itc;

    u_new = u_guess;            %%Newton's converged solution

%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PLOTTING BEGINS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for k=1:M+1
    uexact(k,n) = u(x(k),t(n));
end;

%%% The first subplot would not be performed if u was unknown. %%%
%-------------------------------------------------------------------------
    subplot(2,2,1)
    plot(x,u_new,'-ro',x,uexact(:,n)','k-*');
    plot(x,u_new,'-ro');
    pause(0.1);
    title('Converged Solution Compared to Exact Solution');
    xlabel('x');ylabel('u');
    legend('u-guess','u',2);
%-------------------------------------------------------------------------
```

```
u_old=u_new;                %%Set new initial condition.

error_approximation (n) = norm((uexact(:,n)-u_new),inf);

end                         %% End time loop.

itc
toc

%-------------------------------------------------------------------------
maxiters = max(iterations);
[maxerr,maxstep] = max(error_approximation)

subplot(2,2,2)
plot(1:N,error_approximation)
title('Approximation Error Per Time Step');
xlabel('Time');ylabel('Approximation Error');

subplot(2,2,3)
plot(1:N,iterations)
title('Iterations Per Time Step');
xlabel('Time');ylabel('Iterations');

subplot(2,2,4)
plot(1:maxiters,(error_itc(maxstep,1:maxiters)),'r*-',...
    1:maxiters,(error_residual(maxstep,1:maxiters)),'kd-');
```

```
title('Convergence of Newton Solver For The Worst Time Step');
legend('Iteration Error','Residual Error');
xlabel('Iterations');ylabel('error');
%-------------------------------------------------------------------------

if error==errortol
    disp('reached error-tolerance')
end

if itc==maxit
    disp('reached maxit')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  FUNCTIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Residual = Residual(u_old,u_guess,c,t,x,M,dt)
         Residual = zeros(M+1,1);
%%% Incorporate the Boundary Conditions %%%
         Residual(1)   = u_guess(1)   -bcleft(t);
         Residual(M+1) = u_guess(M+1) -bcright(t);

      for k=2:M         %%interior nodes
           un(k) = (u_guess(k) + u_guess(k+1))./2;
           up(k) = (u_guess(k) + u_guess(k-1))./2;
           Residual(k) = a(u_guess(k)) - a(u_old(k));
           Residual(k) = Residual(k) + c*b(up(k))*(u_guess(k)-u_guess(k-1));
```

```
          Residual(k) = Residual(k) - c*b(un(k))*(u_guess(k+1)-u_guess(k));
          Residual(k) = Residual(k) - dt * f(x(k),t);
      end
function DF  = Jacobian (u_old,u_guess,c,t,x,M,dt)
         DF = zeros(M+1,M+1);
for k=2:M         %%auxiliary vectors
     un(k)=(u_guess(k)+u_guess(k+1))./2;
     up(k)=(u_guess(k)+u_guess(k-1))./2;
end
for k=2:M         %%interior nodes
     DF(k,k) = ap(u_guess(k));
     DF(k,k) = DF(k,k) + (1/2)*c*(bp(up(k))*(u_guess(k)-u_guess(k-1)) - ...
               bp(un(k))*(u_guess(k+1)-u_guess(k)));
     DF(k,k) = DF(k,k) + c*(b(up(k)) + b(un(k)));
     DF(k,k-1) = c*((1/2)*bp(up(k))*(u_guess(k)-u_guess(k-1)) - b(up(k)));
     DF(k,k+1) =-c*((1/2)*bp(un(k))*(u_guess(k+1)-u_guess(k)) + b(un(k)));
end
%%% Dirichlet conditions %%%
     DF(1,1)=1;
     DF(M+1,M+1)=1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function f = f(x,t)
%%% Test Case 1 %%%
%f = 1 + pi*pi*sin(pi.*x);
%%% Test Case 2 %%%
```

```
f = 2*(x*exp(t)+x)*(x*exp(t))-3*(x*exp(t)+x)^2*(exp(t)+1)^2;

%%% Unknown u, with a(u) and b(u) from Test Case 2 %%%

%if x <= .5 && x >= .4

 %   f = 1;

%else

 %   f = 0;

%end


function u = u(x,t)

%u = sin(pi.*x) + x + t;     %% Test Case 1

u = x.*exp(t)+x;             %% Text Case 2


function bc1 = bcleft(t)

bc1 = u(0,t);

%bc1 = 0;


function bc2 = bcright(t)

bc2 = u(1,t);

%bc2 = 1;


function a=a(u)

%a=u;                         %% Text Case 1

a=1+(u)^2;                   %% Test Case 2

%a = (10^(-1)).*exp(10^(-c).*(u-1));  %% c is input

%%% linearized a(u) %%%

%a = 10^(-1)*(1 + 10^(-c)*(u - 1));
```

```
function b=b(u)
%b=1;                          %% Test Case 1
b=1+(u)^3;                     %% Test Case 2
%b = 10^(-2)*exp(10^(-c)*(u-1));
%%% linearized b(u) %%%
%b = 10^(-2)*(1 + 10^(-c)*(u - 1));


function bp=bp(u)
%bp = 0;                       %% Test Case 1
bp=3.*u^2;                     %% Test Case 2
%bp = (10^(-c))*10^(-2)*exp(10^(-c)*(u-1));;
%bp = (10^(-2))*(10^(-c));


function ap=ap(u)
%ap = 1;                       %% Test Case 1
ap=2.*u;                       %% Test Case 2
%ap = (10^-c)*10^(-1)*exp(10^(-c)*(u-1));
%ap = (10^(-1))*(10^(-c));
```