Improved Iterative Methods for NAPL Transport Through Porous Media

An honors thesis for the Department of Mathematics

Victor Minden

Tufts University, 2012

Contents

Chapter 1

Introduction

Modeling fluid flow through a porous medium is a fundamental problem of interest in many areas of science and engineering. Petroleum engineers are concerned with strategies for hydrocarbon recovery and reservoir development [4]. Packed-bed reactors in chemical engineering are filled with a catalyst particulate through which gas reactants are pumped and must be carefully monitored [18]. In environmental engineering and contaminant hydrology, researchers develop methods to predict the flow of pollutants in groundwater [7]. The common denominator is the flow of some fluid – typically a gas or liquid – through a domain that only admits flow through some regions (the pores or void spaces) and not others (the solid matrix), such as soil, a tube packed with sand, or swiss cheese. We concern ourselves in this thesis with contaminant transport and the modeling of groundwater flow in the subsurface.

Industrial and agricultural contamination of groundwater due to run-off, spills, or leakage is a widespread problem, which causes great ecological concern and negatively impacts water quality [9]. A particularly troublesome class of contaminants is Non-Aqueous Phase Liquids (NAPLs) such as pesticides or gasoline. As their name implies, such contaminants do not mix with water and, thus, tend to exist as a distinct fluid phase when introduced to a subsurface soil system [7].

Once a contaminant is introduced to an aquifer such as Figure 1, the pore space at a given point can be saturated with water, the NAPL, air, or some mix of the three. If an air phase is present, then the overall system is one of three-phase flow, and we refer to the region as *unsaturated*, otherwise it is *saturated* [9].

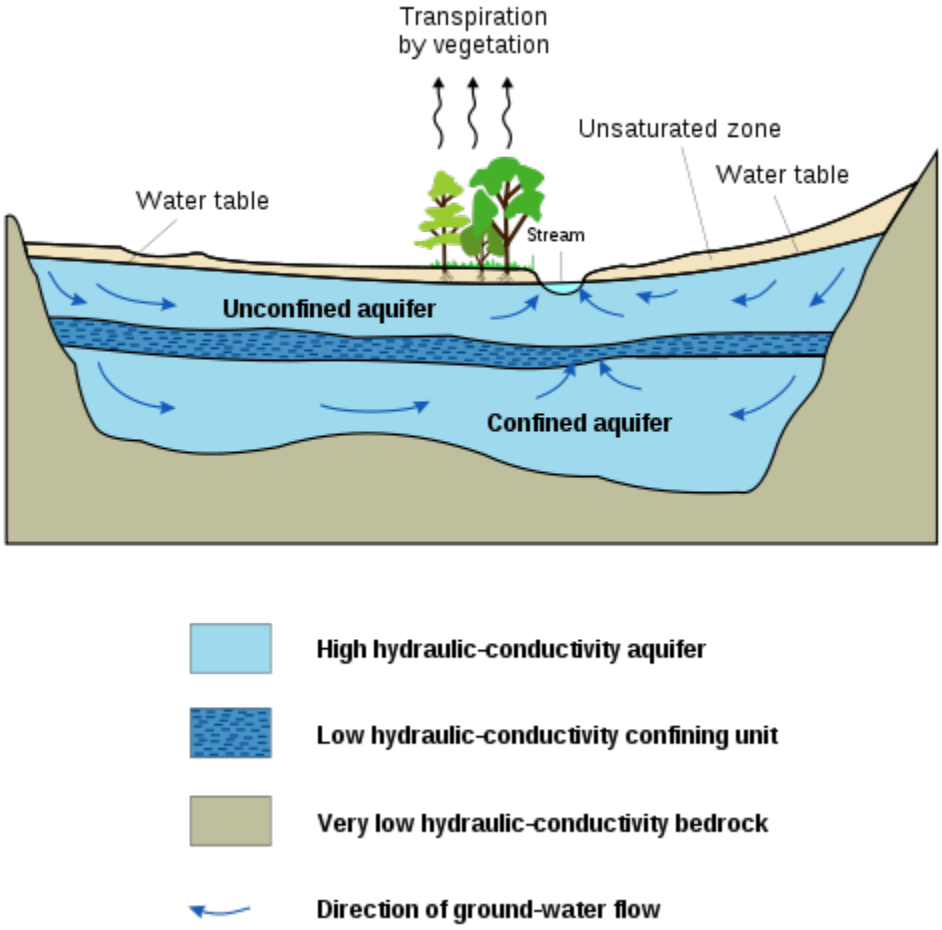


Figure 1.1: A typical aquifer cross-section. Image © Hans Hillewaert / CC-BY-SA-3.0

In either case, the contaminant is subject to the fluid dynamics of the subsurface regime wherein pressure differences and complicated physical interactions between the medium and the fluid phases drive the fluids from one place to another.

If a pollutant is known to have been released into the subsurface, a mathematical model for the way it moves through the soil and interacts physically with the fluids already present there is an invaluable tool to aid containment and ultimate remediation. However, models for physical properties of the soil, such as porosity, cannot be expected to have a simple functional representation – rather, such data is available from measurement or statistical simulation [13]. Even if all the relevant material and fluid properties did have analytic functional forms, the system of partial differential equations that models subsurface flow is complicated, nonlinear, and does not, in general, admit analytical solution [7]. Thus, we turn to numerical

methods for the solution of the system.

In [1], Abriola et al. develop a numerical model and solution scheme (VALOR) for the problem of NAPL transport in two dimensions, using the Implicit Pressure - Explicit Saturation (IMPES) scheme. They later extend the VALOR software to solve three-dimensional problems. In this thesis, we take VALOR and IMPES as the initial model and modify the numerical methods used, with the goal of improving program runtime and efficiency. As such, the primary original contribution of this research is the study of the iterative methods used as applied to the final numerical models obtained.

For completeness, we first include a thorough derivation of both the continuous model for multiphase fluid flow through the subsurface based on first principles, as well as the resulting finite-difference discretization. While this discrete system is nonlinear, a linear system can be obtained through the method of Picard linearization [25] and, thus, the bulk of the numerical problem becomes the solution of a large number of linear systems and the method of generating them from the nonlinear model. We look at the theory of the generalized minimum residual (GMRES) method [30], an iterative technique used by VALOR for solving linear systems, as well as the theory of the Picard linearization process itself.

We look at several changes to the VALOR solution scheme. The GMRES method has different tolerances to detemine when to stop the iterative process, which we tune by hand to obtain better performance. Additionally, a well-known method of improving the convergence of iterative techniques such as GMRES is *preconditioning*, or modification of the system to one with the same solution but a different matrix with better numerical properties [19]. We look at both the method of Jacobi preconditioning originally used by VALOR and an alternative, more complicated preconditioning method – algebraic multigrid [32]. In addition, we introduce another error check to the Picard linearization step wherein we monitor the residual error from the linearization step to ensure that the linear approximation does not deviate substantially from the nonlinear solution. We use this check to investigate an alternative Picard linearization structure to resolve the pressure nonlinearities. All modifications are tested on an example problem that exhibits realistic structure of the physical domain over 5 days.

Chapter 2

Derivation of Models

Here, we first derive a system of partial differential equations describing the pressures and saturations of multiphase fluid flow with an organic contaminant. We then derive a discrete analog of the system for the special case of two-phase flow, where we assume the flow is saturated and, thus, there is no air phase present. While the full multiphase model offers a more complete picture of the physics at hand, it is an extension of the two-phase model and, thus, the numerical methods which we consider could be generalized to multiphase flow without much difficulty. We note that the derivations in this section follow closely those of Abriola et al. in [1].

2.1 Terminology

When a NAPL contaminant, or "organic liquid", is present, the subsurface fluid is made up of three distinct phases: air, water, and the organic liquid itself. As such, we will index fluid properties with a subscript denoting the fluid phase to which they refer: a for air, w for water, and o for the organic liquid. As the equations are symmetric with respect to the fluid phase labeling, we will in general refer simply to the $\alpha-$fluid, where $\alpha = a, w, o$. We will use bold-face font for vector or tensor quantities.

2.2 Physical Model

2.2.1 Base Model

We begin with the principle of conservation of mass, a core tenet of physics that states that the mass of a closed system remains constant over time. In the context of fluid dynamics, this simply means that any change in the amount of a fluid present at one point of the system must be due to either

(i) movement of the fluid between the current point and another point in the system, or

(ii) external sources or sinks (e.g., pumping or injection),

where here we neglect internal sources or sinks due to chemical or biological processes, as well as interphase mass transfer[1].

When expressing conservation of mass for our system, it is important to realize that fluid flow in the subsurface is governed in large part by the permeability of the soil itself. For example, a subsurface region that is completely occupied by the soil medium cannot accommodate any additional volume, so fluid flow is impossible. While, in the ideal case, we would take the exact mathematical description of the distribution of void space on a microscopic level and simulate fluid flow on this domain, this is completely impractical due to a lack of knowledge of the physical configuration of these pores and the intractable nature of such a computation [9]. As such, we will use a macroscopic model and look at simply the fraction of the soil volume that can actually be occupied by a fluid on average, i.e., the *soil porosity*,

$$\phi \equiv \frac{\text{volume of void space}}{\text{volume of soil}}.$$

We will use this continuum approach throughout our model.

We can now formalize the conservation of mass equation for a given phase as

$$\frac{\partial}{\partial t}[\phi S_\alpha \rho_\alpha] = -\nabla \cdot [\rho_\alpha \phi S_\alpha \mathbf{v}_\alpha] + q_\alpha, \tag{2.1}$$

where ρ_α is the density of the $\alpha-$fluid, $\mathbf{v}_\alpha$ is the $\alpha-$fluid velocity, S_α is the $\alpha-$fluid saturation,

$$S_\alpha \equiv \frac{\text{volume of } \alpha\text{-fluid}}{\text{volume of void space}},$$

[1] Valid as, in our case we are interested in time scales and organic contaminants for which these effects are not large [1].

and q_α is the net effect of external sources/sinks on the $\alpha-$fluid.

To convert Equation (2.1) to an equation in terms of pressure rather than velocity, we will use a modified form of Darcy's Law that relates the flux, $F \equiv \phi S_\alpha \mathbf{v}_\alpha$, to the pressure gradient,

$$F \equiv \phi S_\alpha \mathbf{v}_\alpha = -\frac{\boldsymbol{\kappa} k_{r\alpha}}{\mu_\alpha}[\nabla P_\alpha - \gamma_\alpha \nabla z]. \tag{2.2}$$

Above, we have introduced a few new variables. $\boldsymbol{\kappa}$ is the soil intrinsic permeability tensor, which describes from a macroscopic viewpoint the geometrical configuration of the void space [8]. The relative permeability, $k_{r\alpha}$, gives the permeability "seen" by fluid phase α,

$$k_{r\alpha} \equiv \frac{|\ \boldsymbol{\kappa} \text{ effective for } \alpha\text{-fluid}|}{|\boldsymbol{\kappa} \text{ actual}|}.$$

Additionally, μ_α is the dynamic viscosity (which describes a fluids resistance to flow), P_α is the fluid pressure, and $\gamma_\alpha = \rho_\alpha g$ is the specific weight (where g is the acceleration due to gravity). In a simplified sense, Equation (2.2) says that the fluid movement is caused by pressure differences and gravity, which is intuitive – some potential difference (described by the gradient of the pressure) causes fluid to flow at a given rate (described by the flux), but some fluids flow more freely than others so we need to account for that (via the viscosity). We substitute Equation (2.2) into Equation (2.1) to obtain

$$\frac{\partial}{\partial t}[\phi S_\alpha \rho_\alpha] = \nabla \cdot [\boldsymbol{\lambda}_\alpha (\nabla P_\alpha - \gamma_\alpha \nabla z)] + q_\alpha, \tag{2.3}$$

where

$$\boldsymbol{\lambda}_\alpha \equiv \frac{\rho_\alpha \boldsymbol{\kappa} k_{r\alpha}}{\mu_\alpha}$$

is the transmissibility of the $\alpha-$fluid.

2.2.2 Simplifying Assumptions

We make a number of simplifying assumptions to make solution of Equation (2.3) more tractable. In particular, we will assume the following

- The system is isothermal, i.e., temperature gradients do not impact fluid flow. This assumption is not always valid[2], but, in the subsurface regime that we model, the temperature gradients are not large

[2]For example, the density of a fluid changes with temperature [8].

enough to modify the flow significantly [1].

- The dependence of fluid viscosity on pressure is negligible:

$$\mu_\alpha \approx \text{constant}.$$

Abriola et al. note [1] that this assumption is valid for the liquid phases as the actual dependence is negligible over the range of pressures of interest, but caution that further study is needed for the air phase.

- The soil matrix structure is static, i.e., the porosity does not change during the course of the fluid imbibition and drainage:

$$\frac{\partial \phi}{\partial t} \approx 0.$$

This assumption is really one about the interplay between the pressures of interest and the poroelasticity of the material [10] – we are assuming that the pressures in our system at any point are not strong enough to increase the pore size there during imbibition, and that the depths aren't great enough that the force exerted by the soil column above isn't great enough to compress the pores during drainage. Abriola et al. assert this to be a fair assumption in [1].

- Gas phase compressibility follows the Ideal Gas Law:

$$\rho_a = \frac{P_a M_a}{RT},$$

where M_a is the gas phase molecular weight, T is temperature, and R is the universal gas constant.

- The liquid compressibility,

$$c_\alpha \equiv -\frac{1}{V_\alpha}\frac{\partial \rho_\alpha}{\partial P_\alpha},$$

is constant over the range of interest (where, here, V_α is volume). Under this assumption, we see that

$$\rho_\alpha \approx \rho_\alpha^*[1 + c_\alpha(P_\alpha - P_\alpha^*)]$$

by integrating and linearizing around a reference density and pressure, ρ_α^* and P_α^*.

- The capillary pressure at an α, β-fluid interface,

$$P_{c\alpha\beta} \equiv P_\alpha - P_\beta,$$

can be determined as a function of the saturations:

$$P_{c\alpha\beta} = f(S_\alpha, S_\beta). \tag{2.4}$$

We note that, in the literature, the capillary pressure-saturation relationship is in fact hysteretic, but we will neglect this in favor of a functional form [1].

- The relative permeability, $k_{r\alpha}$, can be determined as a function of the saturation[3],

$$k_{r\alpha} = g(S_\alpha). \tag{2.5}$$

- The fluid saturation accounts totally for the pore volume (continuity),

$$S_o + S_w + S_a = 1.$$

We relegate a description of the functional forms used to describe the capillary pressure - saturation and relative permeability - saturation relationships, to Appendix A. Suffice it to say that the relationships are nonlinear and contribute significantly to the difficulty of this problem.

2.3 Numerical Discretization

To convert the physical model of Equation (2.3) to a numerical model, there are a number of decisions which must be made that impact the ultimate system to be solved. First, there are a few different fundamental methods typically used to discretize a system – finite elements, finite differences, etc. – each with its own pros and cons. Second, these methods can differ in how they treat the time-stepping procedure, i.e., implicitly, explicitly, or some combination of the two.

Before a discretization model is chosen, it is important to consider the impact that the model will have on the overall solution scheme. Here, we will briefly review the theory of finite differences, then apply them to our system of PDEs.

[3]Again, known to actually be hysteretic, partly due to poroelasticity.

2.3.1 Finite Difference Schemes

The method of finite differences is a popular scheme for discretizing a differential equation due to its simplicity and intuitive nature. We look first at the discretization of a simple ordinary differential equation, then extend the theory to partial differential equations.

Ordinary Differential Equations

Given a differential operator, L, a real-valued function of one variable $u(x)$, and a real-valued forcing function $f(x)$, we aim, in general, to generate an approximate solution to the equation,

$$Lu(x) = f(x), \quad \text{for } \; x \in \Omega,$$

subject to some initial or boundary conditions. For simplicity, we will assume Ω is bounded.

The finite difference discretization scheme comes out of Taylor's Theorem, which says loosely that, for a function u which is k-times differentiable at the point x, u can be approximated around x by the Taylor polynomial

$$u(x+h) \approx u(x) + u'(x)h + \frac{u''(x)h^2}{2!} + ... + \frac{u^{(k)}(x)h}{k!}.$$

In particular, the first-order Taylor polynomial approximation gives

$$u(x+h) = u(x) + u'(x)h + O(h^2),$$

which we can rearrange and truncate to yield the first-order forward difference approximation of $u'(x)$,

$$\begin{aligned} u'(x) &= \frac{u(x+h) - u(x) + O(h^2)}{h} = \frac{u(x+h) - u(x)}{h} + O(h), \\ &\approx \frac{u(x+h) - u(x)}{h} \equiv \Delta^+ u(x). \end{aligned} \tag{2.6}$$

We note that the truncation error here is $O(h)$. We can similarly derive the first-order backward and central difference approximations, with truncation error $O(h)$ and $O(h^2)$ respectively:

$$u'(x) \approx \frac{u(x) - u(x-h)}{h} \equiv \Delta^- u(x), \tag{2.7}$$

$$u'(x) \approx \frac{u(x+h/2) - u(x-h/2)}{h} \equiv \Delta^0 u(x). \tag{2.8}$$

In the finite difference scheme, we discretize Ω with N points, yielding a discrete domain Ω_N, and approximate all differential operators according to schemes such as Equations (2.6), (2.7), or (2.8). As an example, if we let

$$\begin{aligned} Lu(x) \equiv \frac{du(x)}{dx} &= f(x), \quad x \in \Omega = [0,1], \\ u(0) &= 0, \end{aligned} \tag{2.9}$$

then we can discretize Equation (2.9) according to Equation (2.6) with N points by defining

$$\begin{aligned} x_i &\equiv \frac{i}{N}, \quad i = 1...N \\ h &\equiv x_{i+1} - x_i = \frac{1}{N}, \\ u_i &\equiv u(x_i) \\ f_i &\equiv f(x_i), \end{aligned}$$

which yields

$$\begin{aligned} \frac{du(x_i)}{dx} &= \frac{u_{i+1} - u_i}{h} = f_i, \\ u_0 &= 0. \end{aligned} \tag{2.10}$$

We can rearrange Equation (2.10) to yield an explicit formula for u_{i+1} in terms of u_i and f_i,

$$u_{i+1} = u_i + f_i h. \tag{2.11}$$

This is easily solved given the boundary condition.

Partial Differential Equations

The theory of finite differences for partial differential equations does not differ substantially from that for ordinary differential equations, though each new dimension of Ω must be discretized independently. For example, for a function $u(x, y)$ defined on $\Omega = (0, 1] \times (0, 1]$ with $N_x N_y$ points, we can define

$$\begin{aligned} x_i &\equiv \frac{i}{N_x} \\ y_j &\equiv \frac{j}{N_y} \\ u_{i,j} &\equiv u(x_i, y_j), \end{aligned}$$

and then proceed as above. For example, consider

$$\begin{aligned} Lu(x,t) \equiv \frac{\partial u(x,t)}{\partial t} - \frac{\partial u(x,t)}{\partial x} &= 0, \quad (x,t) \in \Omega = (0,1) \times (0,t_0], \\ u(x,0) &= u_0(x), \\ u(0,t) &= 0. \end{aligned} \tag{2.12}$$

We can discretize Equation (2.12) using Equation (2.6) for both x and t by defining

$$\begin{aligned} x_i &\equiv \frac{i}{N}, \quad i = 1...N \\ t_k &\equiv \frac{k}{K} t_0, \quad k = 1...K \\ h_x &\equiv x_{i+1} - x_i = \frac{1}{N}, \\ h_t &\equiv t_{k+1} - t_k = \frac{t_0}{K}, \\ u_{i,k} &\equiv u(x_i, t_k) \\ f_i &\equiv f(x_i), \end{aligned}$$

which yields

$$\frac{u_{i,k+1} - u_{i,k}}{h_t} + \frac{u_{i+1,k} - u_{i,k}}{h_x} = 0.$$

Rearranging gives

$$u_{i,k+1} = u_{i,k} - \frac{h_t}{h_x}(u_{i+1,k} - u_{i,k}).$$

This, again, can be solved easily by utilizing the initial and boundary conditions.

2.3.2 Explicit Methods vs Implicit Methods

As we mentioned above, numerical methods that use a time-stepping procedure can be broadly classified into either explicit methods or implicit methods. Similar to the way we discretized x and y in Section 2.3.1, solving a time-dependent differential equation requires discretization of t for approximation of time derivatives.

Assume that we are looking for a solution to an equation of the form

$$Fu(\mathbf{x}, t) = 0,$$

where $\mathbf{x}$ is a vector of the spatial variables and F is a differential operator[4]. In an explicit method, the system of equations that result from the discretization process gives an explicit equation for $u(\mathbf{x}, t + \Delta t)$ in terms of the values from previous timesteps,

$$u(\mathbf{x}, t + \Delta t) = f[u(\mathbf{x}, 0 : t), \mathbf{x}, 0 : t].$$

Here we define a function dependent on $0 : t$ to be one that is dependent on potentially any timestep in the range $[0, t]$. Equation (2.11) is an example of an explicit scheme.

In contrast, in an implicit method, the relationship between $u(\mathbf{x}, t + \Delta t)$ and $u(\mathbf{x}, 0 : t)$ is a nontrivial implicit equation which must be solved with each time step to yield $u(\mathbf{x}, t + \Delta t)$:

$$g[u(\mathbf{x}, t + \Delta t), u(\mathbf{x}, 0 : t), \mathbf{x}, 0 : t] = 0.$$

In general, implicit time-stepping methods are computationally more expensive than explicit methods because implicit equations are inherently more difficult to solve than explicit equations. In the case of finite difference discretizations of linear PDEs[5], implicit time-stepping methods require the solution of at least one linear system of equations, $Ax = b$, at each time-step, whereas explicit methods do not. The trade-off, however, comes in the form of *numerical stability*, which refers to how the error in an approximate solution accumulates in successive time-steps. Explicit time-stepping methods can face severe stability problems where the error grows in an unbounded way. Avoiding this sort of error growth requires adherence to what is known as an equation's Courant-Friedrichs-Lewy (CFL) condition, which gives a bound relating the temporal discretization size and the spatial discretization size [24]. This CFL bound usually forces a smaller time step or larger spatial step than would otherwise be desireable.

In contaminant hydrology, the CFL conditions associated with fully explicit time-stepping methods are generally too restrictive to be of practical use, which has made fully implicit methods much more broadly used [14]. In petroleum engineering, however, a commonly used scheme is neither fully implicit nor fully explicit: the Implicit Pressure - Explicit Saturation (IMPES) method [1].

[4]Here, we have incorporated any forcing terms into F.

[5]As our system is nonlinear, we will later discuss linearization.

2.3.3 The IMPES Scheme

The IMPES solution scheme is a hybrid method in reservoir simulation which updates the pressure variables and saturation variables of Equation (2.3) separately, using an implicit method for the pressure update and an explicit method for the saturation update. This is obtained by solving for new pressures based on the saturations from the previous iteration, and then updating the saturation based on these new pressures.

The key advantage of the IMPES method is that it is more stable than a fully explicit method, but less costly than a fully implicit method. As the saturation update is the only explicit part of the procedure, the previously restrictive CFL condition becomes looser and allows larger time steps for the same spatial discretization size, which is desireable.

The fundamental assumption required for the IMPES method is that the capillary pressures change only negligibly with time, i.e.,

$$\frac{\partial P_{c\alpha\beta}}{\partial t} \approx 0.$$

By definition of $P_{c\alpha\beta}$, this implies:

$$\frac{\partial P_w}{\partial t} = \frac{\partial P_o}{\partial t} = \frac{\partial P_a}{\partial t}, \tag{2.13}$$

which we can use to form an equation with only the pressure variables [1].

By applying the product rule to the left-hand side of Equation (2.3), we obtain

$$\begin{aligned}
\frac{\partial}{\partial t}\left[\phi S_w \rho_w\right] = \left[\phi S_w \frac{\partial \rho_w}{\partial P_w} + \rho_w S_w \frac{\partial \phi}{\partial P_w}\right] \frac{\partial P_w}{\partial t} + \phi \rho_w \frac{\partial S_w}{\partial t} &= \nabla \cdot [\boldsymbol{\lambda}_w (\nabla P_w - \gamma_w \nabla z)] + q_w \\
\frac{\partial}{\partial t}\left[\phi S_a \rho_a\right] = \left[\phi S_a \frac{\partial \rho_a}{\partial P_a} + \rho_a S_a \frac{\partial \phi}{\partial P_a}\right] \frac{\partial P_a}{\partial t} + \phi \rho_a \frac{\partial S_a}{\partial t} &= \nabla \cdot [\boldsymbol{\lambda}_a (\nabla P_a - \gamma_a \nabla z)] + q_a \\
\frac{\partial}{\partial t}\left[\phi S_o \rho_o\right] = \left[\phi S_o \frac{\partial \rho_o}{\partial P_o} + \rho_o S_o \frac{\partial \phi}{\partial P_o}\right] \frac{\partial P_o}{\partial t} + \phi \rho_o \frac{\partial S_o}{\partial t} &= \nabla \cdot [\boldsymbol{\lambda}_o (\nabla P_o - \gamma_o \nabla z)] + q_o.
\end{aligned}$$

Using the definition of capillary pressure and our assumption in Equation (2.13), we can write all the pressures in terms of the water pressure and associated capillary pressure for that phase interface, as below:

$$\begin{aligned}
(\phi S_w \frac{\partial \rho_w}{\partial P_w} + \rho_w S_w \frac{\partial \phi}{\partial P_w}) \frac{\partial P_w}{\partial t} + \phi \rho_w \frac{\partial S_w}{\partial t} &= \nabla \cdot [\boldsymbol{\lambda}_w (\nabla P_w - \gamma_w \nabla z)] + q_w \\
(\phi S_a \frac{\partial \rho_a}{\partial P_a} + \rho_a S_a \frac{\partial \phi}{\partial P_a}) \frac{\partial P_w}{\partial t} + \phi \rho_a \frac{\partial S_a}{\partial t} &= \nabla \cdot [\boldsymbol{\lambda}_a (\nabla P_w + \nabla P_{cal} - \gamma_a \nabla z)] + q_a \\
(\phi S_o \frac{\partial \rho_o}{\partial P_o} + \rho_o S_o \frac{\partial \phi}{\partial P_o}) \frac{\partial P_w}{\partial t} + \phi \rho_o \frac{\partial S_o}{\partial t} &= \nabla \cdot [\boldsymbol{\lambda}_o (\nabla P_w + \nabla P_{cow} - \gamma_o \nabla z)] + q_o.
\end{aligned} \tag{2.14}$$

In the above, P_{cal} is the capillary pressure at the air-liquid interface, which depends on whether the organic liquid is present or not[6]. We can rewrite the above to obtain the IMPES saturation equations,

$$\begin{aligned}
\phi\rho_w\frac{\partial S_w}{\partial t} &= \nabla\cdot[\boldsymbol{\lambda}_w(\nabla P_w-\gamma_w\nabla z)]-(\phi S_w\frac{\partial\rho_w}{\partial P_w}+\rho_w S_w\frac{\partial\phi}{\partial P_w})\frac{\partial P_w}{\partial t}+q_w\\
\phi\rho_a\frac{\partial S_a}{\partial t} &= \nabla\cdot[\boldsymbol{\lambda}_a(\nabla P_w+\nabla P_{cal}-\gamma_a\nabla z)]-(\phi S_a\frac{\partial\rho_a}{\partial P_a}+\rho_a S_a\frac{\partial\phi}{\partial P_a})\frac{\partial P_w}{\partial t}+q_a\\
\phi\rho_o\frac{\partial S_o}{\partial t} &= \nabla\cdot[\boldsymbol{\lambda}_o(\nabla P_w+\nabla P_{cow}-\gamma_o\nabla z)]-(\phi S_o\frac{\partial\rho_o}{\partial P_o}+\rho_o S_o\frac{\partial\phi}{\partial P_o})\frac{\partial P_w}{\partial t}+q_o.
\end{aligned}\tag{2.15}$$

By careful manipulation of the the saturation equations, we can eliminate the saturation derivatives (note that $S_o+S_a+S_w=1$, so $\frac{\partial}{\partial t}(S_o+S_a+S_w)=0$). Defining

$$\begin{aligned}
d_{\alpha 1} &\equiv \phi S_\alpha\frac{\partial\rho_\alpha}{\partial P_\alpha}+\rho_\alpha S_\alpha\frac{\partial\phi}{\partial P_\alpha}\\
d_{\alpha 2} &\equiv \phi\rho_\alpha,
\end{aligned}$$

we have

$$\begin{aligned}
(d_{w2}d_{a2}d_{o1}+d_{o2}d_{a2}d_{w1}+d_{w2}d_{o2}d_{a1})\frac{\partial P_w}{\partial t} = \\
d_{o2}d_{w2}\nabla\cdot[\boldsymbol{\lambda}_a(\nabla P_w+\nabla P_{cal}-\gamma_a\nabla z)]+d_{o2}d_{w2}q_a\\
+d_{a2}d_{w2}\nabla\cdot[\boldsymbol{\lambda}_o(\nabla P_w+\nabla P_{cow}-\gamma_o\nabla z)]+d_{a2}d_{w2}q_o\\
+d_{o2}d_{a2}\nabla\cdot[\boldsymbol{\lambda}_w(\nabla P_w-\gamma_w\nabla z)]+d_{o2}d_{a2}q_w.
\end{aligned}$$

This is the IMPES pressure equation. We again point out that the treatment here of the capillary pressures is important, and direct the reader to Appendix A for more information.

2.3.4 Final Model

We now make the transition to a numerical model of two-phase flow consisting of water and the organic phase liquid. We note that this implies the saturation continuity equation changes to

$$S_w+S_o=1,$$

[6]We direct the reader to Section 4.2.3.3 of [1], though we note that this term does not appear in the ultimate two-phase model we will consider.

and that there is only one capillary pressure in the system, P_{cow}. With these changes, however, the IMPES derivation above is still valid and we can modify Equation (2.15) and Equation (2.16) to obtain

$$\begin{aligned}
\phi\rho_w\frac{\partial S_w}{\partial t} &= \nabla\cdot[\boldsymbol{\lambda}_w(\nabla P_w-\gamma_w\nabla z)]-(\phi S_w\frac{\partial\rho_w}{\partial P_w}+\rho_w S_w\frac{\partial\phi}{\partial P_w})\frac{\partial P_w}{\partial t}+q_w\\
\phi\rho_o\frac{\partial S_o}{\partial t} &= \nabla\cdot[\boldsymbol{\lambda}_o(\nabla P_w+\nabla P_{cow}-\gamma_o\nabla z)]-(\phi S_o\frac{\partial\rho_o}{\partial P_o}+\rho_o S_o\frac{\partial\phi}{\partial P_o})\frac{\partial P_w}{\partial t}+q_o,
\end{aligned}\tag{2.16}$$

and

$$\begin{aligned}
(d_{o2}d_{w1}+d_{w2}d_{o1})\frac{\partial P_w}{\partial t} =&\\
&d_{w2}\nabla\cdot[\boldsymbol{\lambda}_o(\nabla P_w+\nabla P_{cow}-\gamma_o\nabla z)]+d_{w2}q_o\\
&+d_{o2}\nabla\cdot[\boldsymbol{\lambda}_w(\nabla P_w-\gamma_w\nabla z)]+d_{o2}q_w.
\end{aligned}\tag{2.17}$$

To discretize Equation (2.17), we will use a central difference for all spatial discretizations and a backwards difference for time – this gives an implicit formulation appropriate for IMPES. First, we explicitly write out the differential operators:

$$\begin{aligned}
(d_{o2}d_{w1}+d_{w2}d_{o1})\frac{\partial P_w}{\partial t} =&\\
&d_{w2}\left(\frac{\partial}{\partial x}[\boldsymbol{\lambda}_o(\frac{\partial P_w}{\partial x}+\frac{\partial P_{cow}}{\partial x})]+\frac{\partial}{\partial y}[\boldsymbol{\lambda}_o(\frac{\partial P_w}{\partial y}+\frac{\partial P_{cow}}{\partial y})]+\frac{\partial}{\partial z}[\boldsymbol{\lambda}_o(\frac{\partial P_w}{\partial z}+\frac{\partial P_{cow}}{\partial z}+\gamma_o)]+q_o\right)\\
&+d_{o2}\left(\frac{\partial}{\partial x}[\boldsymbol{\lambda}_w\frac{\partial P_w}{\partial x}]+\frac{\partial}{\partial y}[\boldsymbol{\lambda}_w\frac{\partial P_w}{\partial y}]+\frac{\partial}{\partial z}[\boldsymbol{\lambda}_w(\frac{\partial P_w}{\partial z}+\gamma_w)]+q_w\right).
\end{aligned}$$

Next, we will substitute the central difference approximation Δ^0 for all spatial derivatives. For the temporal derivative, we will use a backward difference. Below, we assume a uniform discretization within each dimension, though we allow each dimension a different step-size. We define

$$P_{\alpha,i,j,l,t}=P_\alpha(x_i,y_j,z_l,t_t),$$

for some suitable discretization of the space in rectangular coordinates, and write

$$
\begin{aligned}
(d_{o2}d_{w1} + d_{w2}d_{o1})_{i,j,l,t}\Delta_t^- P_{w,i,j,l,t} = \\
& d_{w2,i,j,l,t}(\Delta_x^0[\lambda_{o,i,j,l}(\Delta_x^0 P_{w,i,j,l,t} + \Delta_x^0 P_{cow,i,j,l})] \\
+ \quad & \Delta_y^0[\lambda_{o,i,j,l}(\Delta_y^0 P_{w,i,j,l,t} + \Delta_y^0 P_{cow,i,j,l})] \\
+ \quad & \Delta_z^0[\lambda_{o,i,j,l}(\Delta_z^0 P_{w,i,j,l,t} + \Delta_z^0 P_{cow,i,j,l} - \Delta_z^0 \gamma_{o,i,j,l})] + q_{o,i,j,l,t}) \\
\\
+ \quad & d_{o2,i,j,l,t}(\Delta_x^0[\lambda_{w,i,j,l,t}\Delta_x^0 P_{w,i,j,l,t}] \\
+ \quad & \Delta_y^0[\lambda_{w,i,j,l,t}\Delta_y^0 P_{w,i,j,l,t}] \\
+ \quad & \Delta_z^0[\lambda_{w,i,j,l,t}\Delta_z^0 P_{w,i,j,l,t} - \Delta_z^0 \gamma_{w,i,j,l,t}] + q_{w,i,j,l,t}).
\end{aligned}
$$

Expanding the finite difference operators above, we obtain Equation (2.21), a giant system of equations that gives $\mathbf{P}_t \equiv \{P_{w,i,j,l,t}\}$ implicitly in terms of $\mathbf{P}_{t-1}$[7]. Because the difference $\mathbf{P}_t - \mathbf{P}_{t-1}$ may be small relative to the summands themselves, we will define this change in pressure as $\delta\mathbf{P}_t$ and reformulate Equation (2.21) in terms of this. If we additionally gather like terms, we obtain the final discrete pressure equation, Equation (2.22) (with coefficients as defined in Equation (2.23) and Equation (2.24) and right hand side as defined in Equation (2.25)). Under an appropriate vectorization of the space and linearization scheme, we can write this as a matrix inverse problem,

$$A_t \delta\mathbf{P}_t = \mathbf{b}_t, \tag{2.18}$$

where $A_t \in \mathbb{R}^{N\times N}$, $\delta\mathbf{P}_t, \mathbf{b}_t \in \mathbb{R}^N$.

Equations (2.22) through (2.25) give an implicit discretization of the pressure equations, but we still need to discretize the saturation system in Equation (2.16). The first thing to note is that from our assumption of continuity, we only need to update the organic saturation S_o – the water saturation can be found as $S_w = 1 - S_o$. We can then obtain an expression for updating S_o by following a similar method as for the pressure, yielding Equation (2.26). This gives a saturation update of the form

$$\mathbf{S}_t = \mathbf{S}_{t-1} + B\mathbf{P}_t,$$

[7]Because of their length, some of the equations mentioned on this page are found on later pages.

where B is some coefficient matrix. These equations should be all we need for the IMPES method, but we note that there are a few problems with the formulation of Equation (2.22) and Equation (2.26).

First, as written these require the evaluation of λ_α and γ_α at half-integer indices, and we have not as of yet defined what we mean by this. In a seemingly obvious approach, we would simply define the value at the half-integer index to be the average of the neighboring values, i.e.,

$$\lambda_{\alpha,i+\frac{1}{2},j,l,t} = \frac{\lambda_{\alpha,i,j,l,t} + \lambda_{\alpha,i+1,j,l,t}}{2}. \tag{2.19}$$

However, as Abriola et al. note [1], this is not physically realistic when applied to the transmissibilities. Instead, we calculate the transmissibilties using an upstream weighting, defined as

$$\lambda_{\alpha,i+\frac{1}{2},j,l,t} = \begin{cases} \lambda_{\alpha,i,j,l,t} & \text{if flow is from } i \text{ to } i+1, \\ \lambda_{\alpha,i+1,j,l,t} & \text{if flow is from } i+1 \text{ to } i. \end{cases} \tag{2.20}$$

This promotes physically realistic results. For γ, however, we see no such physical limitations and use the simple average.

The next problem we note with Equation (2.22) is that it is nonlinear, with functions of pressure and saturations multiplying other functions of pressure and saturations. Thus, while it does provide a finite difference approximation of Equation (2.17), we must resolve the nonlinearities or it can face accuracy problems. In particular, we see that the coefficients $d_{\alpha i}$ are linear in the pressures and saturations, so their product is quadratic. Similarly, λ_α and γ_α are functions of the pressure, so evaluating them at time t as Equation (2.22) suggests is impractical, since the pressure at time t is exactly that for which we are trying to solve. We will explore these problems more in Chapter 3, where we look at the Picard linearization method and lagging certain variables to obtain a good approximation of the equation we want to solve.

Finally, we mention that while we've assumed analytically by use of the IMPES method that $\frac{\partial P_{c\alpha\beta}}{\partial t} \approx 0$, it is possible that more accurate results could be obtained by tacking on correction terms to Equation (2.25) to compensate for the capillary pressure time derivatives,

$$\tilde{b}_{i,j,l,t} = b_{i,j,l,t} - \frac{d_{w2}d_{o1}}{h_t}(P_{cow,i,j,l,t} - P_{cow,i,j,l,t-1}).$$

Such a correction will also require lagging variables in the linearization process, as mentioned above.

$$
\begin{aligned}
&\frac{(d_{o2}d_{w1}+d_{w2}d_{o1})_{i,j,l,t}}{\Delta t}(P_{w,i,j,l,t}-P_{w,i,j,l,t-1}) = \\
&d_{w2,i,j,l,t}\left(\frac{\lambda_{o,i+\frac{1}{2},j,l,t}}{h_x}\left[\frac{P_{w,i+1,j,l,t}-P_{w,i,j,l,t}}{h_x}+\frac{P_{cow,i+1,j,l,t}-P_{cow,i,j,l,t}}{h_x}\right]\right. \\
-\;&\frac{\lambda_{o,i-\frac{1}{2},j,l,t}}{h_x}\left[\frac{P_{w,i,j,l,t}-P_{w,i-1,j,l,t}}{h_x}+\frac{P_{cow,i,j,l,t}-P_{cow,i-1,j,l,t}}{h_x}\right] \\
+\;&\frac{\lambda_{o,i,j+\frac{1}{2},l,t}}{h_y}\left[\frac{P_{w,i,j+1,l,t}-P_{w,i,j,l,t}}{h_y}+\frac{P_{cow,i,j+1,l,t}-P_{cow,i,j,l,t}}{h_y}\right] \\
-\;&\frac{\lambda_{o,i,j-\frac{1}{2},l,t}}{h_y}\left[\frac{P_{w,i,j,l,t}-P_{w,i,j-1,l,t}}{h_y}+\frac{P_{cow,i,j,l,t}-P_{cow,i,j-1,l,t}}{h_y}\right] \\
+\;&\frac{\lambda_{o,i,j,l+\frac{1}{2},t}}{h_z}\left[\frac{P_{w,i,j,l+1,t}-P_{w,i,j,l,t}}{h_z}+\frac{P_{cow,i,j,l+1,t}-P_{cow,i,j,l,t}}{h_z}-\gamma_{o,i,j,l+\frac{1}{2},t}\right] \\
-\;&\left.\frac{\lambda_{o,i,j,l-\frac{1}{2},t}}{h_z}\left[\frac{P_{w,i,j,l,t}-P_{w,i,j,l-1,t}}{h_z}+\frac{P_{cow,i,j,l,t}-P_{cow,i,j,l-1,t}}{h_z}-\gamma_{o,i,j,l-\frac{1}{2},t}\right]\right) \\
+\;&d_{o2,i,j,l,t}\left(\frac{\lambda_{w,i+\frac{1}{2},j,l,t}}{h_x}\left[\frac{P_{w,i+1,j,l,t}-P_{w,i,j,l,t}}{h_x}\right]\right. \\
-\;&\frac{\lambda_{w,i-\frac{1}{2},j,l,t}}{h_x}\left[\frac{P_{w,i,j,l,t}-P_{w,i-1,j,l,t}}{h_x}\right] \\
+\;&\frac{\lambda_{w,i,j+\frac{1}{2},l,t}}{h_y}\left[\frac{P_{w,i,j+1,l,t}-P_{w,i,j,l,t}}{h_y}\right] \\
-\;&\frac{\lambda_{w,i,j-\frac{1}{2},l,t}}{h_y}\left[\frac{P_{w,i,j,l,t}-P_{w,i,j-1,l,t}}{h_y}\right] \\
+\;&\frac{\lambda_{w,i,j,l+\frac{1}{2},t}}{h_z}\left[\frac{P_{w,i,j,l+1,t}-P_{w,i,j,l,t}}{h_z}-\gamma_{w,i,j,l+\frac{1}{2},t}\right] \\
-\;&\left.\frac{\lambda_{w,i,j,l-\frac{1}{2},t}}{h_z}\left[\frac{P_{w,i,j,l,t}-P_{w,i,j,l-1,t}}{h_z}-\gamma_{w,i,j,l-\frac{1}{2},t}\right]\right) \\
+\;&d_{w2,i,j,l,t}q_{o,i,j,l,t}+d_{o2,i,j,l,t}q_{w,i,j,l,t}.
\end{aligned}
\tag{2.21}
$$

$$
\begin{aligned}
a_{0,i,j,l,t}\delta P_{w,i,j,l,t}+a_{1,i,j,l,t}\delta P_{w,i-1,j,l,t}+a_{2,i,j,l,t}\delta P_{w,i+1,j,l,t}+a_{3,i,j,l,t}\delta P_{w,i,j-1,l,t} \\
+a_{4,i,j,l,t}\delta P_{w,i,j+1,l,t}+a_{5,i,j,l,t}\delta P_{w,i,j,l-1,t}+a_{6,i,j,l,t}\delta P_{w,i,j,l+1,t} \\
=b_{i,j,l,t}
\end{aligned}
\tag{2.22}
$$

$$
\begin{aligned}
a_{0,i,j,l,t} = \frac{(d_{o2}d_{w1} + d_{w2}d_{o1})_{i,j,l,t}}{\Delta t} \quad &+ \quad \frac{1}{h_x^2}(d_{w2,i,j,l,t}(\lambda_{o,i+\frac{1}{2},j,l,t} + \lambda_{o,i-\frac{1}{2},j,l,t}) \\
&+ \quad d_{o2,i,j,l,t}(\lambda_{w,i+\frac{1}{2},j,l,t} + \lambda_{w,i-\frac{1}{2},j,l,t})) \\
&+ \quad \frac{1}{h_y^2}(d_{w2,i,j,l,t}(\lambda_{o,i,j+\frac{1}{2},l,t} + \lambda_{o,i,j-\frac{1}{2},l,t}) \\
&+ \quad d_{o2,i,j,l,t}(\lambda_{w,i,j+\frac{1}{2},l,t} + \lambda_{w,i,j-\frac{1}{2},l,t})) \\
&+ \quad \frac{1}{h_z^2}(d_{w2,i,j,l,t}(\lambda_{o,i,j,l+\frac{1}{2},t} + \lambda_{o,i,j,l-\frac{1}{2},t}) \\
&+ \quad d_{o2,i,j,l,t}(\lambda_{w,i,j,l+\frac{1}{2},t} + \lambda_{w,i,j,l-\frac{1}{2},t})) \qquad (2.23)
\end{aligned}
$$

$$
\begin{aligned}
a_{1,i,j,l,t} &= -\frac{1}{h_x^2}[d_{w2,i,j,l,t}\lambda_{o,i-\frac{1}{2},j,l,t} \quad + \quad d_{o2,i,j,l,t}\lambda_{w,i-\frac{1}{2},j,l,t}] \\
a_{2,i,j,l,t} &= -\frac{1}{h_x^2}[d_{w2,i,j,l,t}\lambda_{o,i+\frac{1}{2},j,l,t} \quad + \quad d_{o2,i,j,l,t}\lambda_{w,i+\frac{1}{2},j,l,t}] \\
a_{3,i,j,l,t} &= -\frac{1}{h_y^2}[d_{w2,i,j,l,t}\lambda_{o,i,j-\frac{1}{2},l,t} \quad + \quad d_{o2,i,j,l,t}\lambda_{w,i,j-\frac{1}{2},l,t}] \\
a_{4,i,j,l,t} &= -\frac{1}{h_y^2}[d_{w2,i,j,l,t}\lambda_{o,i,j+\frac{1}{2},l,t} \quad + \quad d_{o2,i,j,l,t}\lambda_{w,i,j+\frac{1}{2},l,t}] \\
a_{5,i,j,l,t} &= -\frac{1}{h_z^2}[d_{w2,i,j,l,t}\lambda_{o,i,j,l-\frac{1}{2},t} \quad + \quad d_{o2,i,j,l,t}\lambda_{w,i,j,l-\frac{1}{2},t}] \\
a_{6,i,j,l,t} &= -\frac{1}{h_z^2}[d_{w2,i,j,l,t}\lambda_{o,i,j,l+\frac{1}{2},t} \quad + \quad d_{o2,i,j,l,t}\lambda_{w,i,j,l+\frac{1}{2},t}+] \qquad (2.24)
\end{aligned}
$$

$$
\begin{aligned}
b_{i,j,l,t} = d_{w2,i,j,l,t} \Bigg[& \frac{1}{h_x^2}(\lambda_{o,i+\frac{1}{2},j,l,t}(P_{w,i+1,j,l,t-1} - P_{w,i,j,l,t-1} + P_{cow,i+1,j,l,t-1} - P_{cow,i,j,l,t-1}) \\
& -\lambda_{o,i-\frac{1}{2},j,l,t}(P_{w,i,j,l,t-1} - P_{w,i-1,j,l,t-1} + P_{cow,i,j,l,t-1} - P_{cow,i-1,j,l,t-1})) \\
& + \frac{1}{h_y^2}(\lambda_{o,i,j+\frac{1}{2},l,t}(P_{w,i,j+1,l,t-1} - P_{w,i,j,l,t-1} + P_{cow,i,j+1,l,t-1} - P_{cow,i,j,l,t-1}) \\
& -\lambda_{o,i,j-\frac{1}{2},l,t}(P_{w,i,j,l,t-1} - P_{w,i,j-1,l,t-1} + P_{cow,i,j,l,t-1} - P_{cow,i,j-1,l,t-1})) \\
& + \frac{1}{h_z^2}(\lambda_{o,i,j,l+\frac{1}{2},t}(P_{w,i,j,l+1,t-1} - P_{w,i,j,l,t-1} + P_{cow,i,j,l+1,t-1} - P_{cow,i,j,l,t-1}) \\
& -\lambda_{o,i,j,l-\frac{1}{2},t}(P_{w,i,j,l,t-1} - P_{w,i,j,l-1,t-1} + P_{cow,i,j,l,t-1} - P_{cow,i,j,l-1,t-1})) \\
& - \frac{1}{h_z}(\lambda_{o,i,j,l+\frac{1}{2},t}\gamma_{o,i,j,l+\frac{1}{2},t-1} - \lambda_{o,i,j,l-\frac{1}{2},t}\gamma_{o,i,j,l-\frac{1}{2},t-1}) + q_{o,i,j,l,t} \Bigg] \\
& + d_{o2,i,j,l,t} \Bigg[\frac{1}{h_x^2}(\lambda_{w,i+\frac{1}{2},j,l,t}(P_{w,i+1,j,l,t-1} - P_{w,i,j,l,t-1}) \\
& -\lambda_{w,i-\frac{1}{2},j,l,t}(P_{w,i,j,l,t-1} - P_{w,i-1,j,l,t-1})) \\
& + \frac{1}{h_y^2}(\lambda_{w,i,j+\frac{1}{2},l,t}(P_{w,i,j+1,l,t-1} - P_{w,i,j,l,t-1}) \\
& -\lambda_{w,i,j-\frac{1}{2},l,t}(P_{w,i,j,l,t-1} - P_{w,i,j-1,l,t-1})) \\
& + \frac{1}{h_z^2}(\lambda_{w,i,j,l+\frac{1}{2},t}(P_{w,i,j,l+1,t-1} - P_{w,i,j,l,t-1}) \\
& -\lambda_{w,i,j,l-\frac{1}{2},t}(P_{w,i,j,l,t-1} - P_{w,i,j,l-1,t-1})) \\
& - \frac{1}{h_z}(\lambda_{w,i,j,l+\frac{1}{2},t}\gamma_{w,i,j,l+\frac{1}{2},t-1} - \lambda_{w,i,j,l-\frac{1}{2},t}\gamma_{w,i,j,l-\frac{1}{2},t-1}) + q_{w,i,j,l,t} \Bigg]
\end{aligned} \tag{2.25}
$$

$$
\begin{aligned}
S_{o,i,j,l,t} = \; & S_{o,i,j,l,t-1} + \frac{\Delta t}{d_{o2,i,j,l,t}} \Bigg(\frac{\lambda_{o,i+\frac{1}{2},j,l,t}}{h_x} \left[\frac{P_{w,i+1,j,l,t} - P_{w,i,j,l,t}}{h_x} + \frac{P_{cow,i+1,j,l,t} - P_{cow,i,j,l,t}}{h_x} \right] \\
& - \frac{\lambda_{o,i-\frac{1}{2},j,l,t}}{h_x} \left[\frac{P_{w,i,j,l,t} - P_{w,i-1,j,l,t}}{h_x} + \frac{P_{cow,i,j,l,t} - P_{cow,i-1,j,l,t}}{h_x} \right] \\
& + \frac{\lambda_{o,i,j+\frac{1}{2},l,t}}{h_y} \left[\frac{P_{w,i,j+1,l,t} - P_{w,i,j,l,t}}{h_y} + \frac{P_{cow,i,j+1,l,t} - P_{cow,i,j,l,t}}{h_y} \right] \\
& - \frac{\lambda_{o,i,j-\frac{1}{2},l,t}}{h_y} \left[\frac{P_{w,i,j,l,t} - P_{w,i,j-1,l,t}}{h_y} + \frac{P_{cow,i,j,l,t} - P_{cow,i,j-1,l,t}}{h_y} \right] \\
& + \frac{\lambda_{o,i,j,l+\frac{1}{2},t}}{h_z} \left[\frac{P_{w,i,j,l+1,t} - P_{w,i,j,l,t}}{h_z} + \frac{P_{cow,i,j,l+1,t} - P_{cow,i,j,l,t}}{h_z} - \gamma_{o,i,j,l+\frac{1}{2},t} \right] \\
& - \frac{\lambda_{o,i,j,l-\frac{1}{2},t}}{h_z} \left[\frac{P_{w,i,j,l,t} - P_{w,i,j,l-1,t}}{h_z} + \frac{P_{cow,i,j,l,t} - P_{cow,i,j,l-1,t}}{h_z} - \gamma_{o,i,j,l-\frac{1}{2},t} \right] \\
& - \frac{d_{o1,i,j,l,t}}{\Delta t}\delta P_{i,j,l,t} + q_{o,i,j,l,t} \Bigg)
\end{aligned} \tag{2.26}
$$

Chapter 3

Solution Scheme

The separation of Equation (2.3) into Equations (2.16) and (2.17) in the IMPES method allows us to easily solve the nonlinear system in an iterative fashion: first, we solve the implicit pressure equation, then we update the saturation equations explicitly using the new pressures.

To solve the finite-difference pressure equation, Equation (2.21), we need to solve a linear system of the form $Ax = b$ for x. This is a problem that is well-studied (see, for example, [31]). While general direct methods for solving the system exist (consider the QR and LU decompositions, etc.), they are not sufficient for our needs for two reasons:

1. Direct methods are expensive, and

2. Direct methods do not, in general preserve matrix sparsity.

Regarding (1), most direct methods for general matrices cost $O(n^3)$, which is not acceptable in our case (as any suitable discretization of the subsurface over which we will be computing will be generating more than 10^4 unknowns). With respect to (2), each point in our grid is connected to at most six others (neighboring nodes left and right, up and down, forward and backward), so the resulting matrices are sparse, i.e., have very few nonzeros relative to the potential number of nonzeros. Knowing that the underlying structure has only $O(n)$ nonzeros, it is inconvenient[1] to store the potentially full matrices that result from naively applying direct methods.

[1]Sometimes impossible, depending on available memory and the size of the problem.

Two alternatives to general direct methods are iterative methods and adaptations to direct methods to preserve sparsity. To preserve sparsity, there exist methods that take advantage of the small number of nonzeros by using reordering techniques, typically in an LU decomposition. However, for 3D problems such as those in which we are interested, comparable or better performance can be obtained with iterative methods, which develop improved approximations to the solution with successive iterations [29]. The inherent benefit in such approximation methods is that often the cost to compute a "good enough" solution (i.e., one that is accurate to discretization error) is much less than that for an exact solution.

3.1 GMRES

Solving $Ax = b$ in an iterative fashion requires first generating an initial guess, x_0, of the solution, x, then improving the guess via some iterative algorithm. To ease the following discussion, we will adopt some mathematical formalism here. In particular, let $A \in \mathbb{R}^{n \times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, and $\epsilon \in \mathbb{R}^+$. In an iterative method then, our goal is to find an $\hat{x} \in \mathbb{R}^n$ such that the norm[2] of the residual is sufficiently small, i.e.,

$$||r|| \equiv ||A\hat{x} - b|| < \epsilon.$$

A popular iterative method (and the method we will use) for finding an approximation $\hat{x}$ is the *generalized minimal residual method*, or GMRES [30]. While, in many formulations, the pressure matrix is symmetric and, thus, methods to take advantage of this symmetry (such as the conjugate gradient method[3]) are employed, the coefficients $d_{\alpha 2}$ which appear in Equation (2.24) depend on the current node's porosity and fluid density (which in turn depends on the pressure at that node), precluding symmetry. This causes the problem to require a more general iterative method such as GMRES.

The GMRES method, in concept, is simple: at each iteration, k, we seek $x_k \in K_k$ to minimize the k^{th} residual norm, $||r_k||$. Here, K_k is the k^{th} Krylov subspace of the problem, defined as

$$K_k(A, r_0) \equiv \text{span}\left\{r_0, Ar_0, A^2 r_0, ..., A^{k-1} r_0\right\},$$

where $r_0 \equiv b - Ax_0$ for some initial guess x_0. In our algorithms, we will choose $x_0 = 0$ and thus $r_0 = b$. We

[2] Whenever not specified, the norm used should be assumed to be the Euclidean 2-norm $|| \cdot ||_2$

[3] Assuming the matrix is additionally positive-definite.

see that, by definition, this means that the approximations we find are of the form

$$A^{-1}b \approx x_k = p_{k-1}(A)b, \tag{3.1}$$

where p_{k-1} is a polynomial of degree $k-1$ [29].

To explicity construct x_k, we use a process known as Arnoldi iteration to form an orthonormal basis of $K_k(A, r_0)$. Our goal is to find the x_k that is the minimizer of $||Ax - b||$:

$$x_k = \underset{x \in K_k}{\operatorname{argmin}} ||Ax - b||. \tag{3.2}$$

Given an orthonormal basis $\{q_i\}$ for K_k, we can assert the existence of a $y \in \mathbb{R}^k$ such that $x_k = Q_k y$, where here Q_k is the $n \times k$ matrix

$$Q_k = [q_1|q_2|...|q_k]\,.$$

Such a Q_k is exactly what the Arnoldi algorithm (Algorithm 1) gives us. Beginning with $q_1 = b/||b||$, we use the modified Gram-Schmidt process to form our orthonormal basis by, at each iteration k, generating a vector $\tilde{q}_k \in K_k$ via multiplication by A to generate a new direction, which is orthogonalized against each of the previously generated vectors in turn to construct $q_k \in K_k \setminus K_{k-1}$. In matrix form, the Arnoldi algorithm

Algorithm 1 The Arnoldi Algorithm [19]

Given q_1 with $||q_1|| = 1$

for $j = 1, 2, ...$ **do**

  $\tilde{q}_{j+1} \leftarrow Aq_j$

  for $i = 1, 2, ...,$ j **do**

    $h_{i,j} \leftarrow \langle \tilde{q}_{j+1}, q_i \rangle$

    $\tilde{q}_{j+1} \leftarrow \tilde{q}_{j+1} - h_{ij} q_i$

  end for

  $h_{j+1,j} = ||\tilde{q}_{j+1}||$

  $q_{j+1} = \tilde{q}_{j+1}/h_{j+1,j}$

end for

may be written as

$$AQ_k = Q_{k+1} H_{k+1,k},$$

where $H_{k+1} \in \mathbb{R}^{k+1 \times k}$ is the matrix whose top $k \times k$ block is made up of $\{h_{i,j}\}$ (according to Algorithm 1) and whose last row is zero except for the element $h_{k+1,k}$.

Hence, with Arnoldi in hand, we can reframe the problem of Equation (3.2) as finding

$$\begin{aligned} y_k &= \underset{y \in \mathbb{R}^k}{\operatorname{argmin}} ||AQ_k y - b|| = \underset{y \in \mathbb{R}^k}{\operatorname{argmin}} ||Q_{k+1} H_{k+1,k} y - b|| \\ &= \underset{y \in \mathbb{R}^k}{\operatorname{argmin}} ||Q_{k+1}(H_{k+1,k} y - \beta e_1)|| = \underset{y \in \mathbb{R}^k}{\operatorname{argmin}} ||H_{k+1,k} y - \beta e_1||, \\ x_k &= Q_k y_k, \end{aligned} \tag{3.3}$$

where e_1 is the vector with a 1 in the first component and zeros everywhere else, $\beta = ||b||$, and the last equality follows from the columns of Q_{k+1} being orthonormal. Vector y_k can now be found cheaply with a QR factorization of $H_{k+1,k}$, and x_k extracted from y_k. Because of the form of $H_{k+1,k}$, information regarding its QR factorization can be saved from iteration to iteration via Givens rotations to reduce the cost of the algorithm. The GMRES algorithm can be seen in Algorithm 2, where we terminate when the desired accuracy is reached (or some maximum iteration count is exceeded). We note that the residual norm $||Ax - b||$ can be checked implicitly.

3.1.1 Restarts

As the size of the matrices involved increases with each iteration, one common modification to GMRES is to incorporate a 'restart' after some number of iterations, where we take the current estimate x_k and throw away our information about the Krylov subspace, then set $x_0 = x_k$ and begin the Arnoldi iteration from the beginning (this variant is known as GMRES(k)). This puts a limit on the number of basis vectors that must be stored and orthogonalized against, but it also unfortunately can cause the iteration to stagnate, and GMRES(k) is not guaranteed to converge [29]. We choose to use GMRES(k) for our problem in order to cut down on costs, but note that, depending on the number of iterations before restart and number of total iterations, standard GMRES could potentially be a better choice.

Algorithm 2 The GMRES Algorithm [31]

$q_1 \leftarrow b/||b||$

for $k = 1, 2, ..., N$ **do**

$\tilde{q}_{k+1} \leftarrow Aq_k$

for $i = 1, 2, ..., k$ **do**

$h_{i,k} \leftarrow \langle \tilde{q}_{k+1}, q_i \rangle$

$\tilde{q}_{k+1} \leftarrow \tilde{q}_{k+1} - h_{ik}q_i$

end for

$h_{k+1,k} = ||\tilde{q}_{k+1}||$

$q_{k+1} = \tilde{q}_{k+1}/h_{k+1,k}$

Check $||Ax - b||$

end for

$y_k \leftarrow \underset{y \in \mathbb{R}^k}{\operatorname{argmin}} ||H_{k+1,k}y - \beta e_1||$

$x \leftarrow Q_k y_k$

3.2 Convergence and Preconditioning of GMRES

3.2.1 Convergence

An important factor regarding the GMRES Algorithm, or, indeed, any iterative algorithm, is its rate of convergence – how many iterations do we need to perform to get the residual norm down to the desired error tolerance?

From Equation (3.1), the residual r_k in standard GMRES satisfies

$$||r_k|| = \min_{p_k} ||p_k(A)r_0||,$$

with the restriction that $p_k(0) = 1$. We see this by noting that Equation (3.1) implies

$$r_k \equiv b - Ax_k = b - Ap_{k-1}(A)b = [I - Ap_{k-1}(A)]\, b = p_k(A)b$$

where, for the general case, b can be replaced by r_0. Since the identity naturally arises at each iteration, we see that the constant value of the polynomial p_k must be 1, which explains our restriction above.

Following the derivation of Greenbaum [19], we assume that A is diagonalizable as $A = V\Lambda V^{-1}$ (where V is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues) and see that

$$||r_k|| = \min_{p_k} ||p_k(A)r_0|| = \min_{p_k} ||Vp_k(\Lambda)V^{-1}r_0|| \leq \kappa(V) \min_{p_k} ||p_k(\Lambda)|| \cdot ||r_0||$$

and thus, assuming $||r_0|| \neq 0$, that

$$\frac{||r_k||}{||r_0||} \leq \kappa(V) \min_{p_k} \max_i |p_k(\lambda_i)|, \tag{3.4}$$

where $\kappa(V)$ is the 2-norm condition number of V.

As we have mentioned before, in our case the matrix A is not symmetric. However, it is *close* to being symmetric, positive-definite (SPD). In particular, if we look at a subset[4] of the eigenvalues for a particular problem (Figure 3.1) we see that the imaginary parts are small (in this case, zero, though again we stress this is only a subset) and the real parts are positive. We note that we obtain estimates of the extremal eigenvalues of A by using those of the Hessenberg matrix H inside GMRES [5].

In the case that the eigenvalues are all positive real (A is SPD), $\kappa(V) = 1$ and the optimization problem of Equation (3.4) reduces to the problem of finding the minimax polynomial on the real interval $[\lambda_{min}, \lambda_{max}]$. In approximation theory, it is well known that this corresponds to the k^{th} shifted, scaled Chebyshev polynomial, and it follow from this that

$$\frac{||r_k||}{||r_0||} \leq 2 \left(\frac{\sqrt{\lambda_{max}/\lambda_{min}} - 1}{\sqrt{\lambda_{max}/\lambda_{min}} + 1} \right)^k. \tag{3.5}$$

For a detailed proof of this, we direct the reader to [19].

In Equation (3.5), we see a nice bound on the convergence rate of GMRES in terms of the largest and smallest eigenvalues for an SPD matrix. Heuristically, we can[5] argue that, because A is close to SPD, the derived bound still provides some useful intuition behind how GMRES will behave. In particular, we see that the wider the spectrum of A is, the more GMRES iterations it will take to bring the residual down to a given tolerance. This difference can be dramatic. Consider, the case of a small spectrum, $\lambda_{max} = 4$ and $\lambda_{min} = 1$. Then we see in the SPD case that with each iteration the residual norm decreases by at least a

[4]It is infeasible to calculate all the eigenvalues for a given discretization for problems of the size with which we are dealing.

[5]Though perhaps we shouldn't.

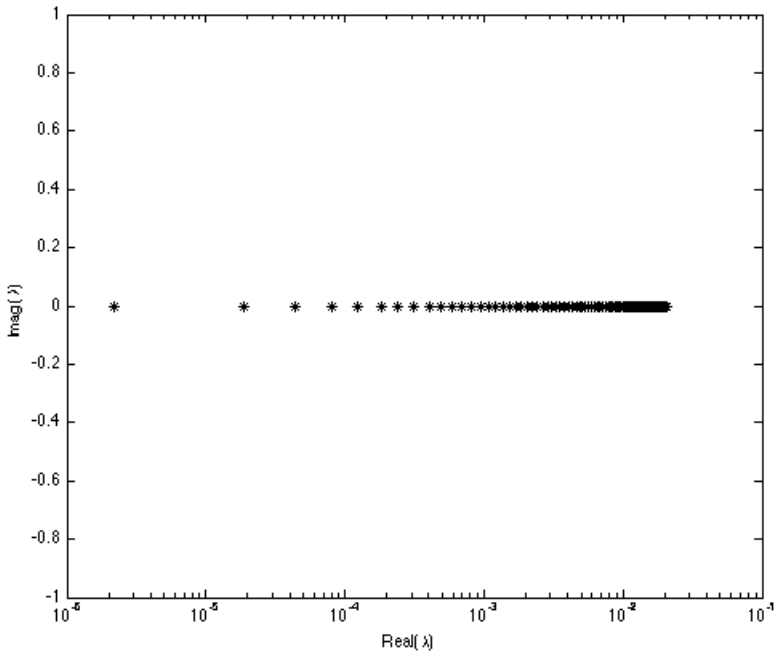


Figure 3.1: The first 100 eigenvalues estimated by GMRES for a typical problem, plotted in the complex plane. Note that the real axis is plotted on a logarithmic scale.

factor of 3. In contrast, if a spectrum is very wide, e.g., $\lambda_{max} = 10^6$, $\lambda_{min} = 1$, then the residual is only guaranteed to decrease by a factor of about 1.005 – the iteration is almost useless.

3.2.2 Preconditioning

As the convergence of GMRES depends on the spectrum of the matrix with which we are iterating, an essential tool in numerical linear algebra is to replace A in the iteration with a "preconditioned" version of A, which has a nicer spectrum. For right preconditioning, we seek a matrix M such that AM^{-1} is closer to normal than A with more clustered eigenvalues, then we use AM^{-1} in our iterative method instead of A, solving the system

$$AM^{-1}y = b,$$

and then, finally, computing $x = M^{-1}y$. The end result is an iterative process that, if we've chosen M well, converges more quickly than simply iterating on A.

Choosing the correct preconditioner is a difficult problem and, in most cases, it seems that preconditioner choice is more of an art than a science. For our purposes, we will look at two difference choices of preconditioner: the Jacobi (or "diagonal") preconditioner, and an Algebraic Multigrid (AMG) preconditioner. The former is one of the simplest forms of preconditioning and is very cheap to construct and apply, whereas the latter is based on more modern ideas and can be expensive.

Jacobi

The Jacobi preconditioner is a simple, yet effective one for many problems of interest. For Jacobi preconditioning, we choose $M = D \equiv \text{diag}(A)$, such that the matrix AM^{-1} is simply A with each element a_{ij} divided by the element on the main diagonal[6] of that column, δ_{jj} [19]. Jacobi has the advantage of being very cheap to construct and use, given that the entries of D can easily be extracted from A and multiplication by a diagonal matrix can be done in linear time, but it often faces problems due to a lack of robustness – its simplicity is its downfall, and there are many more complicated preconditioners which often beat Jacobi in terms of convergence rate and overall time-to-solution.

[6]Here we note that we require the main diagonal to be non-zero, which is true for our matrices of interest.

AMG

A more complicated method for preconditioning involves the use of what are known as algebraic multigrid (AMG) methods. Originally, multigrid methods arose from a geometric observation: during Jacobi- or Gauss-Seidel-like iteration on some elliptic differential equation discretized on a grid, the high-frequency error is damped much more quickly than the low-frequency (or "smooth") error [32].

In a discretized setting, it is well-known that the concept of frequency is relative to the discretization. For example, if we consider the function $\cos(\pi x)$ on the discrete domain $\Omega_1 = \{0, 1, ..., 10\}$, then we see in Figure 3.2 (top) that we would consider it to be highly oscillatory[7]. However, if we consider the same function on the finer domain $\Omega_2 = \{0, 1/10, ..., 10\}$, we see in Figure 3.2 (bottom) that, relative to this grid, the function is much smoother.

Such observation gave rise to the idea of geometric multigrid: given a problem discretized on a fine grid Ω^h, improve the solution by *restricting* the error and residual to a coarser grid (in many cases, this coarser grid is Ω^{2h} = "every other point in Ω^h"), solving there, then *interpolating* a correction term back to the fine grid. Via this method, the portion of the error in the solution that was smooth relative to Ω^h should have been sufficiently damped (as it is resolved well on Ω^{2h}) and the remaining error in the solution should be oscillatory relative to Ω^h, which is quickly damped by (weighted-) Jacobi or Gauss-Seidel iteration on Ω^h [32]. This would be considered a an example of a "two-grid" scheme, but one strength of multigrid is its recursive nature – we could simply continue to restrict things in our initial guess, to Ω^{4h}, Ω^{8h} and so on.

In geometric multigrid as we've discussed so far, the key idea has been taking advantage of the phyiscal grid inherent in the problem and different observations regarding the spatial sampling frequency. The success of geometric multigrid, however, has prompted the application of multigrid-type methods to general linear systems, even those in which *no underlying grid exists* – this is AMG.

Geometric multigrid has two basic aspects: the relaxation scheme, which smooths high-frequency error, and the coarse-grid correction scheme. For AMG, once we have a relaxation scheme, we face two main questions [11]:

- What does it mean for error to be low-frequency when no grid exists?

[7] In fact, it is the highest frequency representable on this grid.

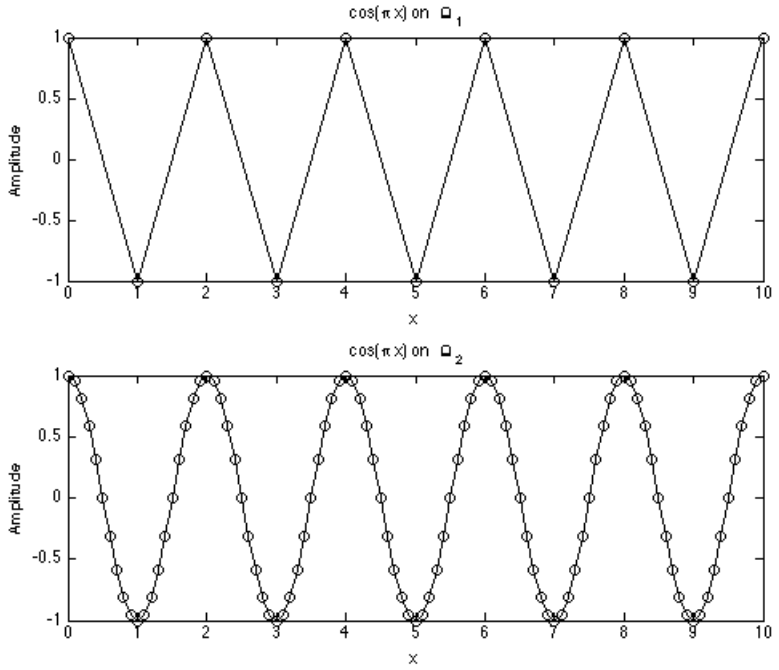


Figure 3.2: The function $\cos(\pi x)$ on two different discrete domains. Relative to Ω_1 (top), the function is very oscillatory, but relative to Ω_2 (bottom), it looks much less oscillatory.

- How do we generate a "coarse grid" from a general matrix A?

A simple solution for the first question is to simply define low-frequency (algebraically smooth) error as that error which the relaxation scheme does not effectively damp [32]. With regards to the second question, we can think about the matrix A as being a weighted adjacency matrix and consider the nodes of its associated graph F to make up the fine "grid". The problem then reduces to finding a subset, $C \subset F$, such that the the solution on C can be used to approximate the solution on F and the high-frequency error appears smoother on C [11]. There exist algorithms for finding C, which can be more or less complicated, but what they have in common is that they use the information in the elements of A to find the coarse grid. As such, AMG is very flexible, but this comes at the cost of an additional setup phase in which A is analyzed to determine the grid hierarchy and necessary interpolation/restriction operators [32].

Now that we've given a (very) basic outline of AMG, its use as a preconditioner is not difficult to understand. Given a Krylov method such as GMRES, our preconditioner can be considered a general operator, and in AMG-preconditioned GMRES we let the preconditioning algorithm be the actual application of multigrid for a certain number of cycles. For example, we can consider the case of a multigrid V-cycle with grids Ω_{N-1} to Ω_0. Application of AMG as a preconditioner in GMRES involves solving systems of the form $Mz = r$ (or equivalently, computing $z = M^{-1}r$) [19], which then corresponds to restricting r to the coarsest level Ω_0 (pre-smoothing before each restriction), solving on the coarse grid, then interpolating corrections to the solution of $Ae = r$ back to the finest grid Ω_{N-1} (post-smoothing before each interpolation). We note that the theory of using a multilevel preconditioner does not differ significantly from applying Jacobi or Gauss-Seidel as a preconditioner itself [32]. For a more in-depth treatment (including parallelization), we direct the reader to [20].

3.3 Picard Linearization

We have looked at some of the theory behind solving linear systems numerically, but, as we mentioned in Section 2.3, our system of PDEs has some nonlinearities that should not be ignored. There are two main linearization strategies used in solving groundwater flow equations: Picard iteration, and Newton iteration [25]. As their names imply, each involves employing a nested iteration within the time-stepping

loop. Each method has its pros and cons. In particular, the Picard scheme is inexpensive, but exhibits only linear convergence of the inner iteration, whereas Newton iteration converges quadraticaly but requires computation of a Jacobian, which is expensive [28]. In essence, with Picard we can perform more inner iterations of a cheaper method, whereas with Newton we would perform fewer inner iterations of a more expensive method [25].

We will use a Picard iteration for our equations, due to its simplicity and cheap computational cost per iteration. Picard iteration is a fixed-point method for solving nonlinear systems of the form

$$x = F(x), \tag{3.6}$$

where F is some system nonlinear in $x \in \mathbb{R}^n$. Picard iteration involves simply taking an initial guess x_0 and then obtaining better approximations by applying F repeatedly,

$$\begin{aligned} x^{(0)} &= x_0 \\ x^{(k)} &= F(x^{(k-1)}). \end{aligned}$$

Given that Equation (3.6) does, in fact, have a solution and that F follows some loose assumptions with regard to Lipschitz continuity (we direct the reader to [22]), the Picard iteration will converge. We define the nonlinear residual norm as

$$||r_{NL}^{(k)}|| \equiv ||x^{(k)} - F(x^{(k)})||.$$

In our model, the equation we need to linearize comes from Equation (2.18), where we note that because both the matrix A_t and the vector $\mathbf{b}_t$ depend on coefficients that are functions of the pressure and saturation, we can write A_t as $A_t = A(\delta\mathbf{P}_t, \mathbf{S}_t)$, and $\mathbf{b}_t$ as $\mathbf{b} = \mathbf{b}(\delta\mathbf{P}_t, \mathbf{S}_t)$. Then the fixed-point equation of interest is

$$\delta\mathbf{P}_t = A(\delta\mathbf{P}_t, \mathbf{S}_t)^{-1}\mathbf{b}(\delta\mathbf{P}_t, \mathbf{S}_t).$$

The natural initial guess for the change in pressure between successive timesteps is $\delta\mathbf{P}_t = 0$, and with this the corresponding Picard iteration is

$$\begin{aligned} \delta\mathbf{P}_t^{(0)} &= 0 \\ \delta\mathbf{P}_t^{(k)} &= A(\delta\mathbf{P}_t^{(k-1)})^{-1}\mathbf{b}(\delta\mathbf{P}_t^{(k-1)}). \end{aligned} \tag{3.7}$$

Even in Equation (3.7), there are still choices that can be made. In paticular, when choosing how to update A and $\mathbf{b}$ in the Picard iteration, we note that they are dependent not only on $\delta\mathbf{P}$ but also on $\mathbf{S}$. The purpose of the iteration is to resolve the nonlinearities of the equation, and so it is popular to treat linear terms or terms that are less strongly nonlinear as "frozen" and not update them with each iteration,k.

For our purposes we will look at two different program structures for the nonlinear updates. The first, following Abriola et al. [1], simply involves updating every dependence at each Picard iteration following Algorithm 3 . Because this resolves nonlinearities in both the pressure and the saturation at each iteration, we will refer to this as Parallel Picard iteration. We note that the convergence criterion is some tolerance on the nonlinear residual, or a maximum number of iterations, among other choices.

Algorithm 3 Parallel Picard iteration

for $t = 1, 2, ..., t_{max}$ **do**

$\delta\mathbf{P}_t^{(0)} = 0$

$\mathbf{S}_t^{(0)} = \mathbf{S}_{t-1}$

for $k = 1, 2, ..., k_{max}$ **do**

//Picard iteration

$\delta\mathbf{P}_t^{(k)} = A^{-1}(\delta\mathbf{P}_t^{(k-1)}, \mathbf{S}_t^{(k-1)})\mathbf{b}(\delta\mathbf{P}_t^{(k-1)}, \mathbf{S}_t^{(k-1)})$

$\mathbf{P}_t^{(k)} = \mathbf{P}_{t-1} + \delta\mathbf{P}_t^{(k)}$

$\mathbf{S}_t^{(k)} = \mathbf{S}_{t-1} + B(\delta\mathbf{P}_t^{(k)}, \mathbf{S}_t^{(k-1)})\mathbf{P}_t^{(k)}$

if converged **then**

break

end if

end for

end for

The second method we will use freezes the saturations for the Picard iteration process and calculates the saturation update after the fact. This saturation can then be used to perform an additional linearization of the pressure equations. If the saturation and this linearization sufficiently agree (ie, the nonlinear residual is small), then we can move on to the next timestep. Otherwise, we will use this new saturation and perform

the entire Picard iteration process again. This sort of additional "sanity check" can be thought of as a predictor-corrector loop, where we guess the saturation will not change drastically with the pressure, solve the pressure, then update the saturation and see if this new saturation affects the pressure substantially or not. We refer to this as Predictor-Corrector Picard iteration, and it can be seen in Algorithm 4.

The motivation for Algorithm 4 is mathematical. In the parallel scheme of Algorithm 3, we note that the pressure nonlinearities are never resolved to our tolerance before we use them to update the saturations. In contrast, pulling the saturation out of the Picard loop allows us to, for a given saturation guess, completely resolve the pressure before we move on to update the saturation again. The predictor-corrector step then provides an extra check to ensure that the new saturation and calculated pressure agree – if they don't, we perform the same steps again. This makes sense, because it restricts the Picard iteration itself to the implicit portion of the solution scheme, i.e., that for which we can actually track the residual.

3.4 Adaptive Time-stepping

Due to the hybrid implicit/explicit nature of IMPES, the resultant numerical algorithm faces some restrictions on the maximum timestep, as mentioned previously. In [1], however, Abriola et al. point out that these restrictions are a function of not only the discretization in space, but also the soil and fluid properties such as the capillary pressure gradient. As these variables change over the course of the simulation, it is reasonable to intelligently vary the timestep size so as to take as large a timestep as possible at each iteration, thus reducing the overall number of solves required.

We use the same heuristic as in [1] for Parallel Picard iteration, adjusting the timestep as a function of the number of Picard iterations required for the previous solve. In particular, given some specified initial timestep Δt_0, we define the next time step as

$$\Delta t_t = \begin{cases} 0.75\Delta t_{t-1} & \text{if more than 20 Picard iterations required at time } t-1, \\ 1.05\Delta t_{t-1} & \text{if 3 or fewer Picard iterations required at time } t-1, \\ \Delta t_{t-1} & \text{else.} \end{cases}$$

Algorithm 4 Predictor-Corrector Picard iteration

for $t = 1, 2, ..., t_{max}$ **do**

$\mathbf{S}_t^{(0)} = \mathbf{S}_{t-1}$

for $p = 1, 2, ..., p_{max}$ **do**

//Predictor-Corrector

$\delta\mathbf{P}_t^{(0,p)} = 0$

for $k = 1, 2, ..., k_{max}$ **do**

//Picard iteration

$\delta\mathbf{P}_t^{(k,p)} = A^{-1}(\delta\mathbf{P}_t^{(k-1,p)}, \mathbf{S}_t^{(p-1)})\mathbf{b}(\delta\mathbf{P}_t^{(k-1,p)}, \mathbf{S}_t^{(p-1)})$

$\mathbf{P}_t^{(k,p)} = \mathbf{P}_{t-1} + \delta\mathbf{P}_t^{(k,p)}$

if Picard converged **then**

break

end if

end for

$\mathbf{S}_t^{(p)} = \mathbf{S}_{t-1} + B(\delta\mathbf{P}_t^{(p)}, \mathbf{S}_t^{(p-1)})\mathbf{P}_t^{(p)}$

if Predictor-Corrector converged **then**

break

end if

end for

end for

Chapter 4

Results

In this chapter, all simulations were run on a single node of the Tufts Linux Research Cluster with a 2.8GHz Intel® Xeon® processor and 16GB of RAM. For these numerical experiments, we use a test problem with a completely flat water table and fluid parameters as given in Appendix B. Our timestep parameters are $\Delta t_0 = 60s$, $\Delta t_{min} = 1 \times 10^{-4}s$, and $\Delta t_{max} = 86400s$. In our spatial discretization, we use $l = 141$ points in the vertical (z) direction, $m = 65$ points in one horizontal direction (x), and $n = 21$ in the other (y). The spatial step sizes are $h_z = 0.05$ m, $h_x = 0.25$ m, and $h_y = 0.5$ m. We enforce no flow through the top and bottom boundaries, and constant pressure / saturation at the horizontal boundaries.Material property values for the capillary pressure and saturation calculations were generated from simulation with the Transition Probability Geostatistical Software Library (T-PROGS) [13].

4.1 Original Fortran and C Port

The original VALOR library of Abriola et al. outlined in [1] was written in Fortran using sparse linear algebra routines from the SLATEC Common Mathematical Library. The implementation uses Jacobi for left preconditioning of GMRES and follows the Parallel Picard iteration as described in Algorithm 3.

Because the SLATEC library is no longer well-documented nor supported, we will move away from it in this thesis in favor of the Portable, Extensible Toolkit for Scientific Computation (PETSc), a C/C++ library which includes BLAS- and LAPACK-type subroutines such as are in SLATEC, and also provides a

front-end to several other modern scientific computing libraries [6].

We port the pressure matrix assembly and solution via GMRES to the PETSc library, using the same algorithms and parameters. In particular, we use a restart parameter $k = 10$ and run GMRES for 30 iterations at each time step (i.e., restart three times). The Parallel Picard iteration is terminated when the maximum change in saturation over all nodes is less than 1×10^{-4}. Note that as part of the porting process the sparse matrix format was changed from the SLATEC-preferred coordinate format to the PETSc-preferred Compressed Sparse Row (CSR) format [5].

In Figures 4.1 and 4.2, we see views for the solution after 5 days for both Fortran and C performing the same calculations. Qualititatively, the solutions have the same basic profile – we see high concentrations and low concentrations in the same place for both implementations, which gives credence to the idea that these are comparable solutions. Numerically, the 2-norm of the difference between the solutions is on the order of 1×10^{-4}, which is sufficiently small. We note that while the solutions are not exactly the same, this discrepancy can be expected as we are using different libraries which perform the computations in a different order.

A fortunate benefit of the switch from Fortran to C for the pressure solve is the greater efficiency of the PETSc library for solving the sparse system – even with the same parameters, we see in Figure 4.3 that the C implementation is about 1.7x faster overall than the Fortran implementation, with all of that speed-up coming from the pressure solve.

There are two other preliminary changes we would like to make to the original implementation for our following numerical experiments. First, because we will later be changing the flat 30 iterations to tolerances on the residual, we would like to switch from left preconditioning to right preconditioning. The reason for this is that left preconditioning will change the residual norm that we are actually minimizing during the GMRES alogrithm, while right preconditioning will not [29].

Second, we would like to add an additional error criterion to our Picard iteration. In particular, whereas the original VALOR source code uses only the max-norm over saturation as described above, we would like to monitor the max-norm over the pressure, and the nonlinear residual as well. Since the pressure solution is changing with each linearization as well as the saturation solution, enforcing the pressure solution to

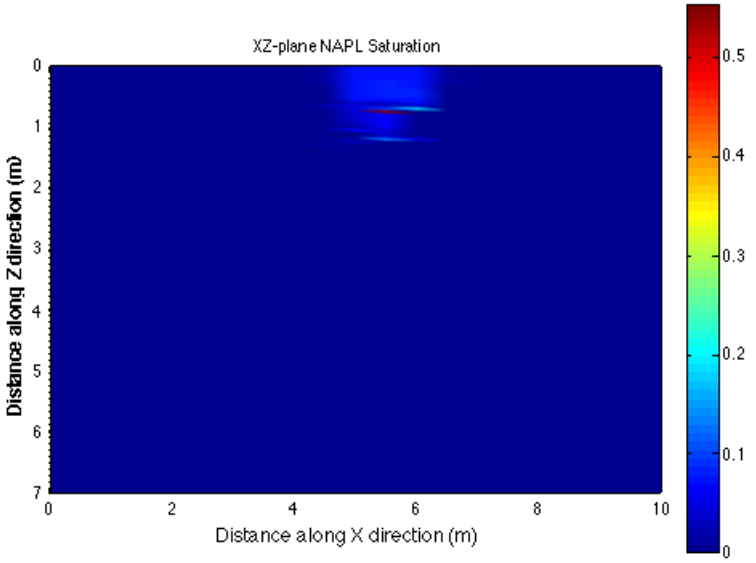
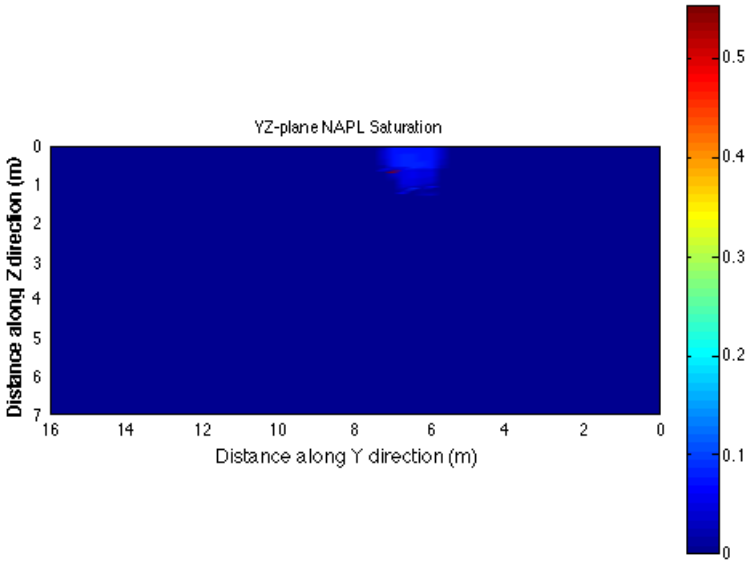


Figure 4.1: NAPL Saturation after 5 days for Fortran implementation.



have converged before we advance the timestep is good practice. We use a scaled L_2 norm to monitor the nonlinear residual, defined as $||A(\delta \mathbf{P}_t^k)\delta \mathbf{P}_t^k - b(\delta \mathbf{P}_t k)||_2/(mnl)$. We note that scaling by the total number of grid points allows for some normalization with respect to grid size. We restrict this to 1×10^{-4} in accordance with our other tolerances.

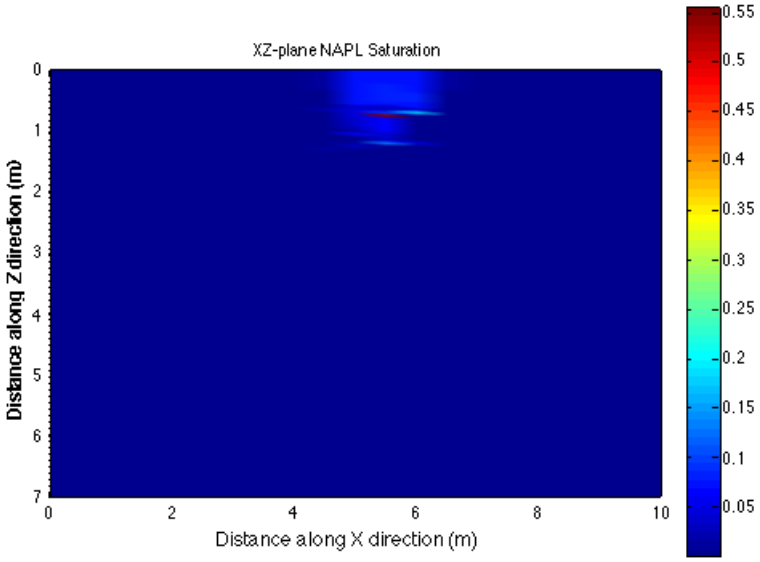


Figure 4.2: NAPL Saturation after 5 days for C implementation.

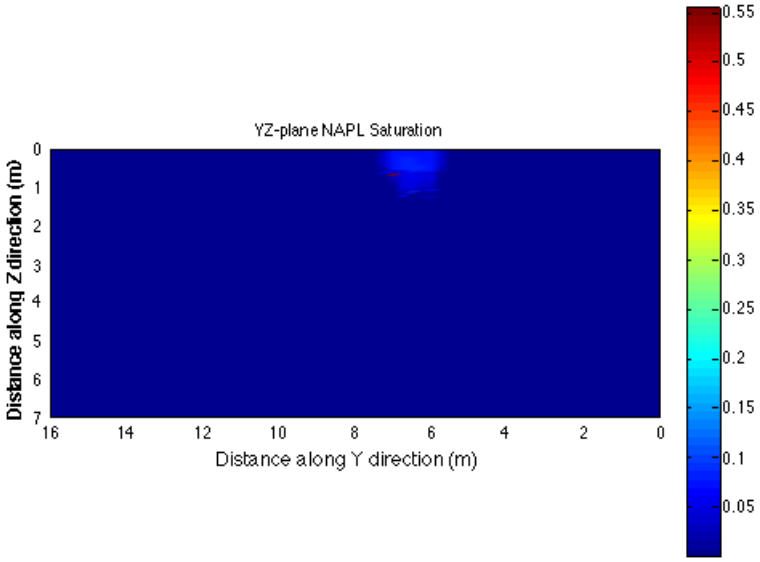


Figure 4.4 shows the comparative runtimes for right and left preconditioning, and for right preconditioning with the new error criterion. We note that right preconditioning takes a small performance hit with these parameters, but that the additional error criterion does not increase the runtime noticeably.

For all the remaining simulations, we will use right-preconditioned GMRES in the C implementation

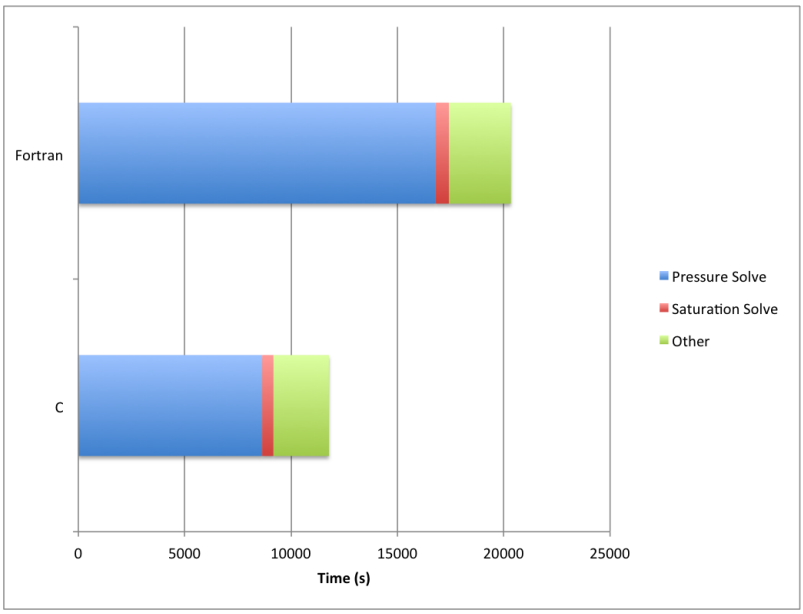


Figure 4.3: Comparative runtime for the C and Fortran implementations. We break the total time down into three main components: the pressure solve, the saturation solve, and the general program overhead (file I/O, matrix/vector assembly, etc.).

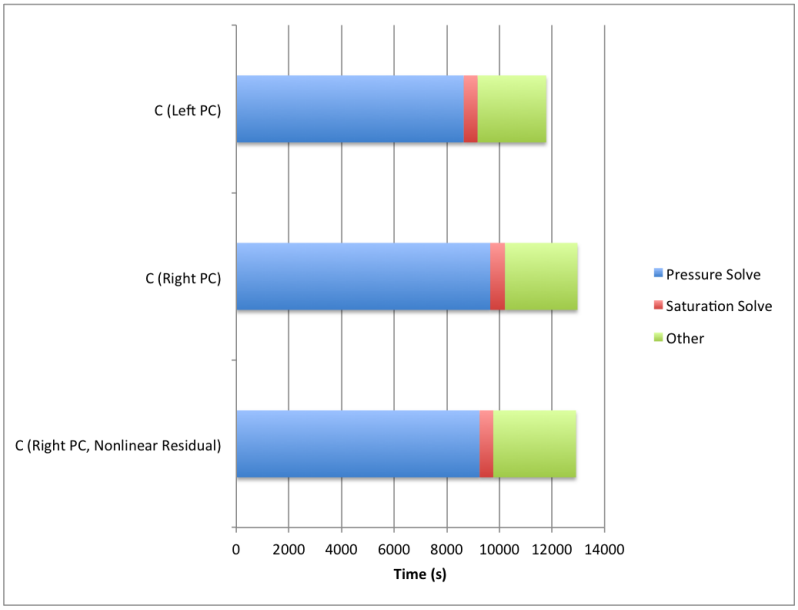


Figure 4.4: Comparative runtime for left preconditioning, right preconditioning, and right preconditioning with a tolerance on the nonlinear residual norm.

with the two-norm error as described above.

4.2 Varying GMRES Tolerances

While above we focused on using a set number of iterations of GMRES for each linear solve, an alternate practice that we will adopt is using a tolerance on the residual as described in Section 3.1. This makes sense because our ultimate goal is not to enforce a given number of iterations of GMRES–it's to find an approximate solution of the linear system. Using the Euclidean two-norm, we will put tolerances on both the absolute residual norm,

$$R_{abs} \equiv ||b - Ax||,$$

and the relative residual norm,

$$R_{rel} \equiv ||b - Ax||/||b||.$$

The first is a metric for the size of the residual in an absolute sense, and the second gives us the relative size of the residual compared to the size of the right-hand side, which is also the ratio that gives the reduction in the size of the residual as compared to the initial guess $x_0 = 0$.

In addition to the standard residual checks above, we also implement a "stagnation" tolerance due to a simple observation. In the initial implementation, the GMRES solver tends to perform well initially for any given linearization and then the reduction in residual between successive iterations approaches zero (examples can be seen in Figure 4.5.). To avoid wasting cycles on iterations that aren't really helping, we force GMRES to quit when the reduction in residual is less than 1% in sucessive iterations. This returns us to the linearization step of the IMPES algorithm, in hopes that the system resulting from the next linearization will be better conditioned, or that progress can be made from a smaller improvement.

In Figure 4.6, we see the runtime breakdown for different tolerance choices. It is interesting to note that the stagnation tolerance makes a marked improvement in the pressure solve time, but beyond that the choice of tolerance does not drastically impact the program run time. A comparison of the solutions shows that the 2-norm difference when adding the stagnation tolerance to GMRES is on the order of 1×10^{-3} and the max-norm difference is on the order of 1×10^{-4}. This is greater deviation then we saw in the transition from

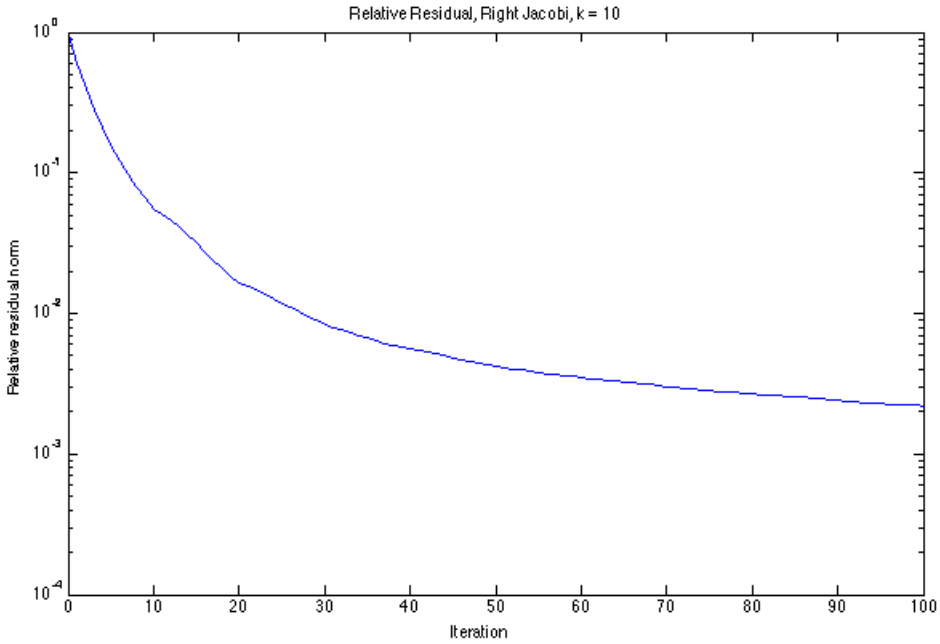
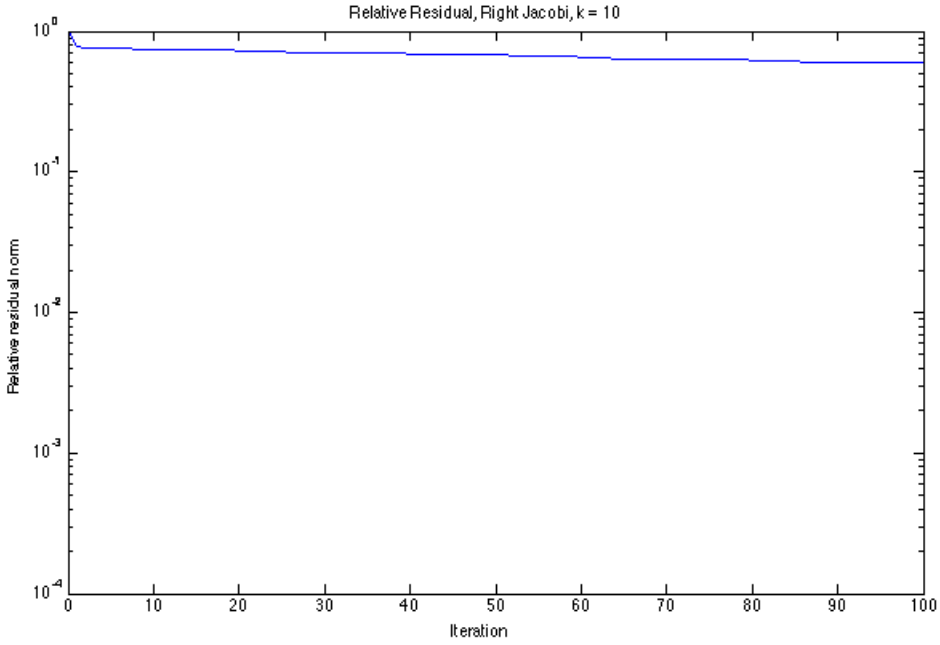


Figure 4.5: Plots of the relative residual in GMRES with Jacobi preconditioning as a function of iteration index for example timesteps. Above, we see an iteration that starts off functional then slowly stagnates. Below, the algorithm is effectively stagnant after even just the first few iterations.



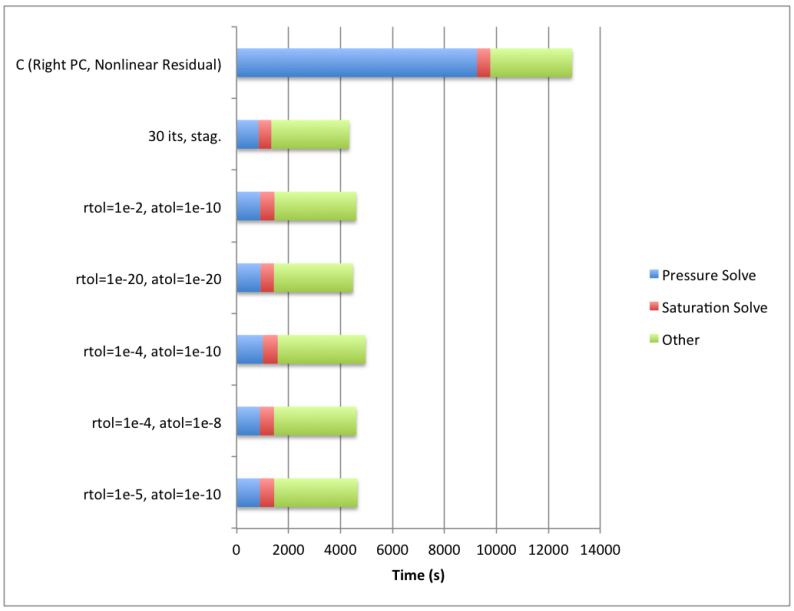


Figure 4.6: Comparative runtime for different choices of tolerances. Top, we see the C version with nonlinear residual tolerance from Figure 4.4. Below that is the same setup with the stagnation tolerance implemented. The remaining runs use the stagnation tolerance with various relative residual tolerances (rtol) and absolute residual tolerances (atol).

Fortran to C, but not by much. The parameters which showed the least deviation were rtol = 1×10^{-4}, atol = 1×10^{-8}, so we will use those in the following simulations, since the computational cost does not differ substantially from the best-case parameters[1].

4.3 Varying Restart Parameter

Another parameter which is notoriously difficult to choose is the restart parameter k of restarted GMRES. One would expect that a larger restart parameter would imply better convergence at the cost of more work (as we are discarding less information), but this is not necessarily the case [15]. In this section, we look at the effect of hand-tuning k on our test problem.

Figure 4.7 shows the results for values of k between 5 and 50, using rtol=1×10^{-4} and atol = 1×10^{-8} as described above. The difference between the worst choice (k=50, about 4875 seconds of runtime) and the

[1]Basically, running for as many iterations as needed until GMRES stagnates.

best choice (k=30, about 4267 seconds of runtime) is 608 seconds, which gives an idea of the overall range of variation. The improvement of the best choice over the previous method (k=10, about 4608 seconds of runtime) is 341 seconds. This is not as substantial a performance increase as we saw tuning other parameters, but does represent a 1.08x boost that was repeatable in multiple trials.

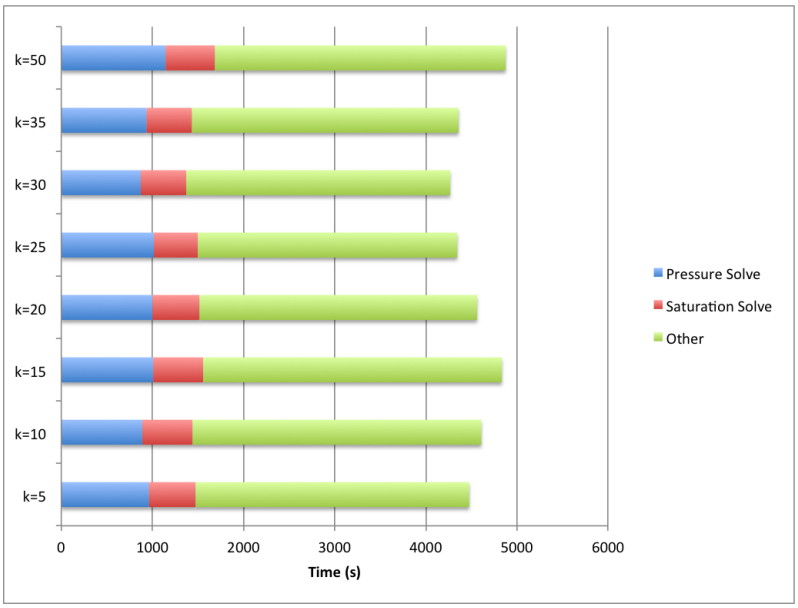


Figure 4.7: Comparative runtime for different choices of GMRES restart parameter k.

4.4 Alternative Choice of Preconditioner

In Section 3.2.2, we discussed the importance of preconditioning for the GMRES algorithm. Until now, our simulations have all been using a simple Jacobi preconditioner, but now we look at using AMG for this problem. Using the BoomerAMG preconditioner from the Hypre library [16], we explore two different methods. In the first, we perform the AMG setup phase (in which the coarse grid operators are determined from the matrix) at each matrix solve separately. However, as we've previously mentioned, the AMG setup phase is costly. Thus, our second method involves *amortizing* the cost of the setup over multiple linear solves.

For a given timestep, the preconditioner is constructed for the first Picard linearization, but is not updated with the pressures and saturation. If the initial guess is close to the true result, the old preconditioner should

still be effective, though it will perhaps require a few more iterations (either GMRES or linearizations, or both). If the cost of these additional iterations is less than the cost of the setup, there is a net gain.

Monitoring the GMRES residual at the same two timesteps we saw in Figure 4.5, Figure 4.8 shows the strength of AMG. GMRES on the AMG-preconditioned problem does not stagnate and exhibits a much better convergence rate than on the Jacobi-preconditioned problem. However, we must expect this to come at some cost.

In Figure 4.9, we see the runtime results of applying these two AMG methods. For the non-amortized case, we choose a strength threshold of 0.75, which is a parameter that determines which inter-node connections are "strong" in the coarsening process. For the amortized case, as it is more promising than the first, we explore the parameter space for the strength threshold.

We see that the non-amortized implementation of AMG performs very poorly, taking even more time to run than the original Fortran code. This illustrates the problem with AMG we previously discussed – the setup phase can be expensive, and we are constructing many different linear systems in the course of time-evolving the pressures and saturations.

The amortized implementations show varying performance. All the choices of strength threshold perform better than the non-amortized case with respect to the pressure solve itself, but we see that AMG with a threshold of 0.25 does not beat the non-amortized case overall. This can be simply explained by the adaptive timestepping – the amortized, low-threshold implementation does not solve the system well, and thus causes the timestep to be decreased. This results in more total pressure solves, saturation solves, and matrix/vector assemblies.

The amortized implementations with strength thresholds of 0.75 and 0.5 perform better, but still do not see as great of a computational gain as the best-case Jacobi parameters. We note that experiments with incomplete LU and Cholesky[2] preconditioners did not yield results significantly better than the amortized AMG case.

[2]Not technically applicable, but $A \approx$ SPD.

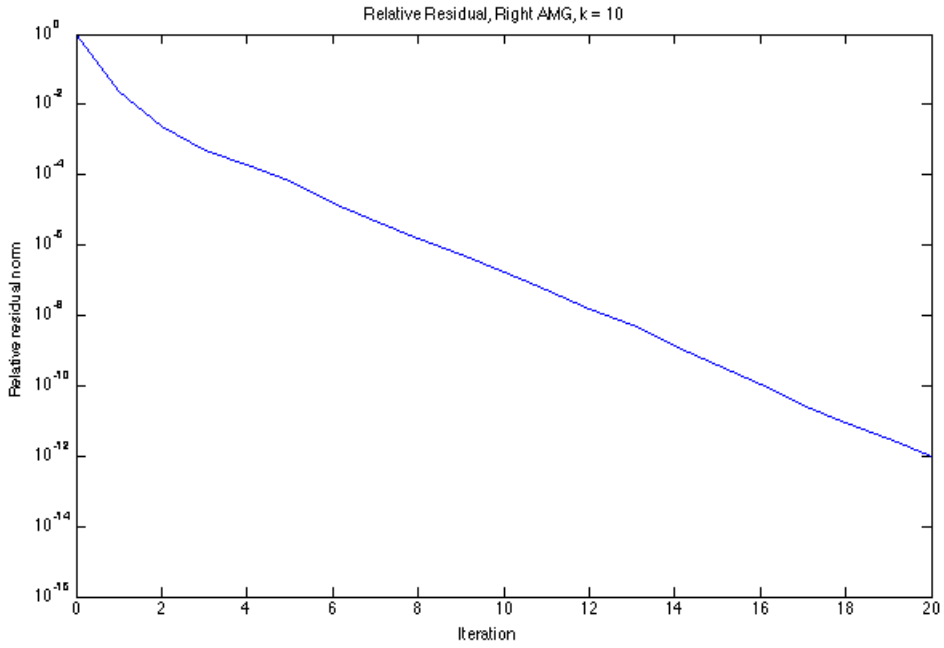
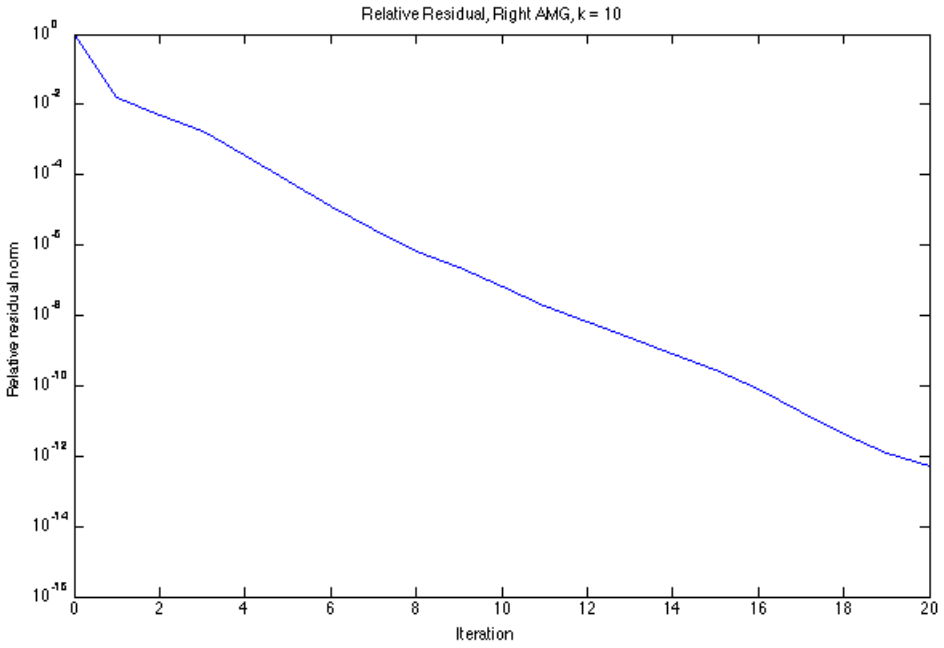


Figure 4.8: Plots of the relative residual in GMRES with AMG preconditioning as a function of iteration index for example timesteps. The strength threshold for BoomerAMG was set to 0.75.



4.5 Alternative Linearization Structure

The final alteration we consider is the restructuring of the fixed-point iteration described in Section 3.3. Using the same tolerances as before (rtol = 1×10^{-4}, atol = 1×10^{-8}) and the Jacobi preconditioner, we

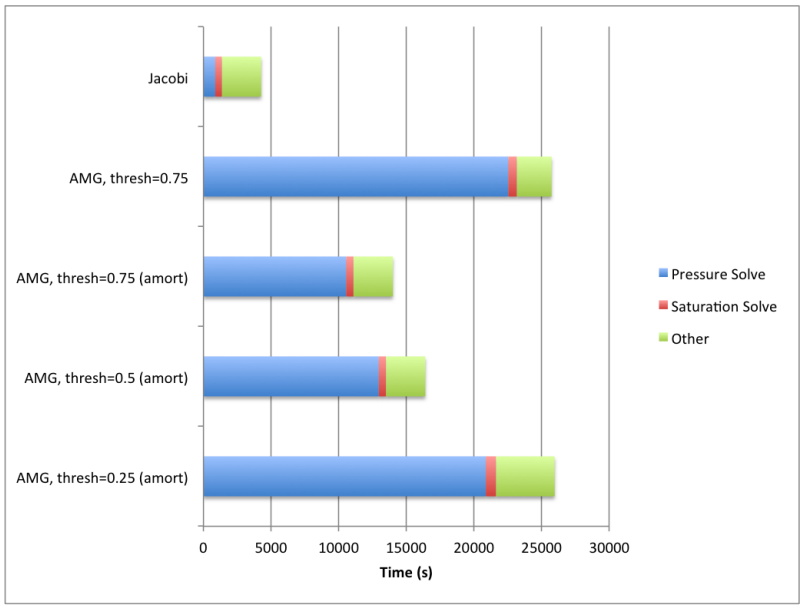


Figure 4.9: Comparative runtime for AMG implementations. Compare to Figure 4.5 for Jacobi. Note that the vertical and horizontal limits are different between these and the Jacobi plots.

run the test problem using both the Parallel Picard iteration and the Predictor-Corrector Picard iteration structure. For the predictor-corrector loop, we use as our convergence criterion the same nonlinear residual check that we introduced for the original structure.

In Figure 4.10, we see that the new linearization structure results in a greater runtime, both for the pressure solution and saturation solution. It is understandable that we might see this result – we can perform more Picard iterations now per saturation solve, so we might expect the number of pressure solves to increase in general. Indeed, profiling reveals that the new linearization structure averages about 8 Picard iterations (and thus linear solves) total per time step, whereas the previous method averages about 6. The profiling also reveals that the number of timesteps for the new structuring is greater than those previously (possible because of the adaptive timestepping) – 4136 versus 3451. This is rather substantial, so we look further at the time behavior of the solution.

Figure 4.11 shows the net simulated time as a function of the iteration for each of the two cases. We see that while both methods increase and decrease the time step at the same time in the simulation (i.e., horizontal cross-sections hit points on the curves which have roughly the same derivative), the iteration at

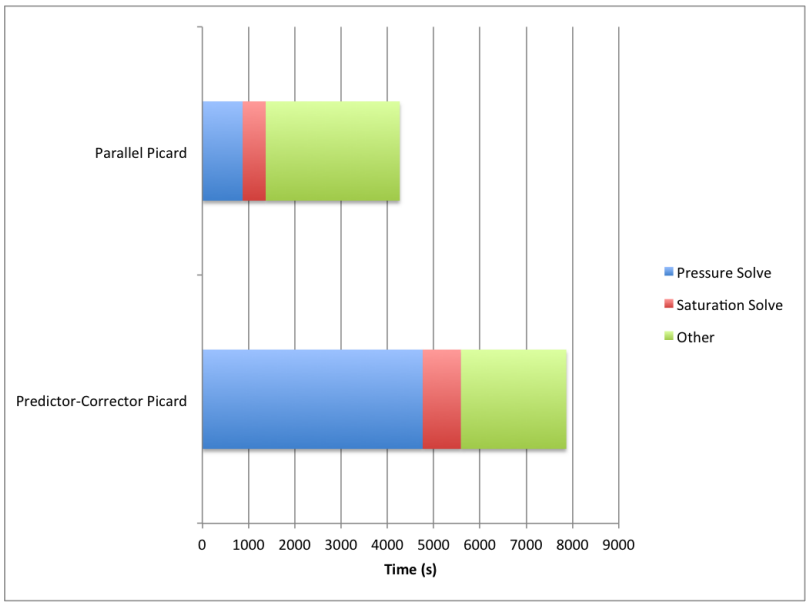


Figure 4.10: Comparative runtime results with the Parallel Picard iteration structure and Predictor-Corrector Picard iteration structure.

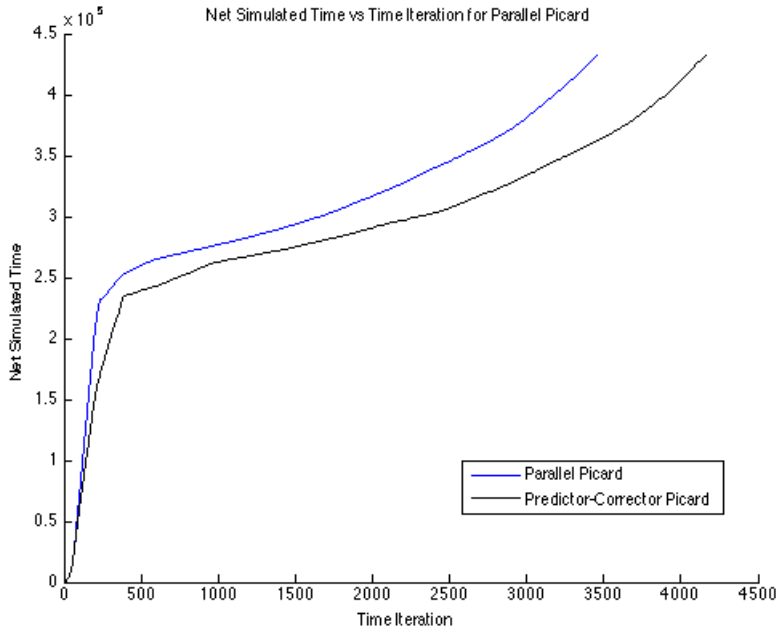


Figure 4.11: Simulated time vs timestep for the two different Picard structures. We see that the adaptive time-stepping behaves differently for the two cases – the new structure lags the old.

which these events occur is not the same for both. In fact, the new predictor-corrector linearization starts off performing the same as the previous Picard structure, then begins to lag it. This difference in behavior implies that the predictor-corrector requires more linearizations than the previous structure at certain time steps, and thus the step-size does not increase there, which causes the predictor-corrector to fall behind.

We note that the max norm difference between the solution for the new structure and the old is on the order of 1×10^{-2}, much larger than we've seen previously. We discuss this more in Section 4.6.

4.6 Discussion

4.6.1 Summary of Results

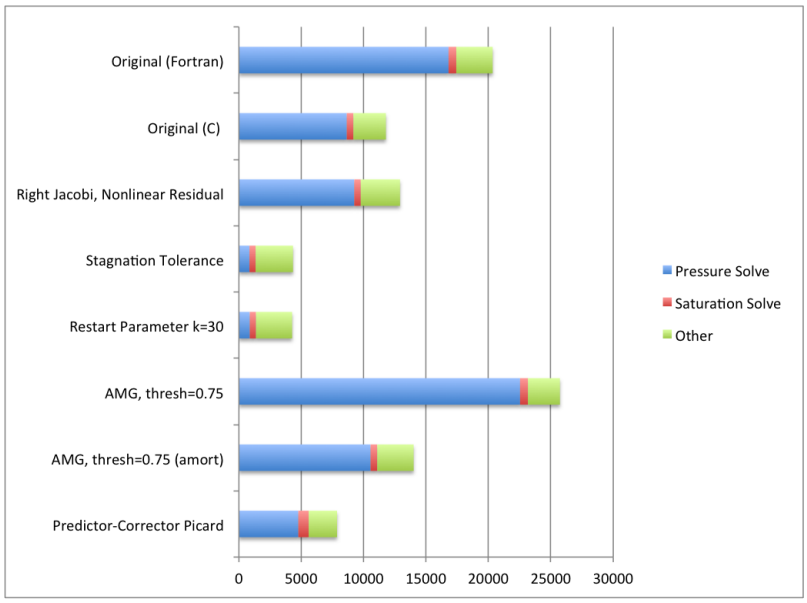


Figure 4.12: Comparative runtimes for each change explored in this thesis. Note that the cases where the stagnation tolerance was introduced and restart parameter was modified both use right Jacobi and monitor the nonlinear residual, as does the implementation using the Predictor-Corrector loop. The AMG implementations also use GMRES(30) and the stagnation tolerance, though stagnation was never an issue with AMG.

In Figure 4.12, we see the comparative runtimes plotted together for the original test problem. From the top, the first 5 bars show roughly the "successful" changes to the solution algorithm and how much each

contributed to lessening the runtime. The remaining plots, consisting of the AMG cases and the alternative linearization scheme, did not offer computational gain in the form of runtime improvement, for different reasons that we explore below.

For verification of the results obtained with our successful changes, we use an alternative test problem with the same fluid and block parameters as the first but with different material parameters as generated by T-PROGS. The max-norm difference of the solutions is about 7×10^{-4}, which is not large. The comparative runtimes can be seen in Figure 4.13, where we observe about a 6.25x speed-up.

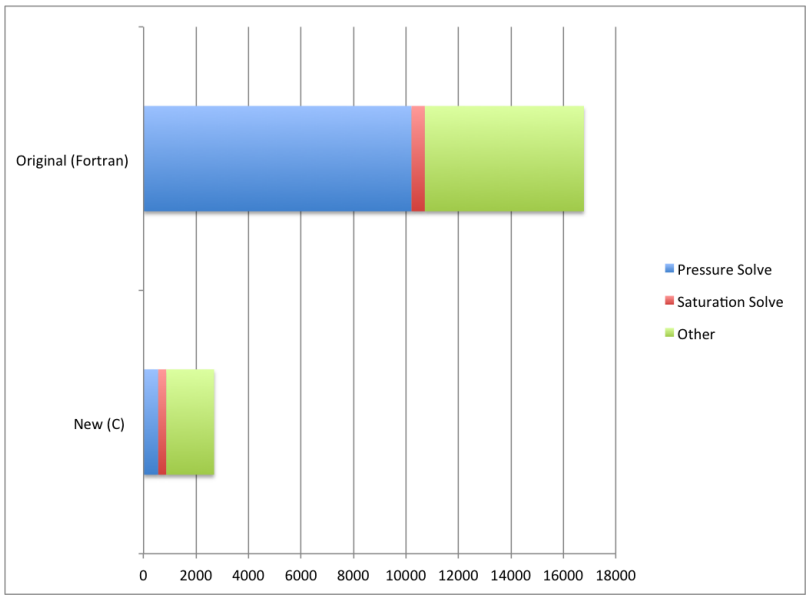


Figure 4.13: Comparative runtimes for the second test problem. We look at the original Fortran program and the C implementation using right Jacobi, rtol = 1×10^{-4}, atol = 1×10^{-8}, the stagnation criterion, and k = 30.

4.6.2 AMG vs Jacobi

We observed in the Results section that, even when amortizing AMG builds across multiple linear solves, Jacobi still outmatches AMG by a wide margin. To more deeply understand this, we will look at the eigenvalues of the preconditioned matrices for both the Jacobi and AMG cases, in accordance with the theory outlined in Section 3.2.1.

Figure 4.14 shows the eigenvalues estimated from GMRES for a given timestep on a log plot for both types

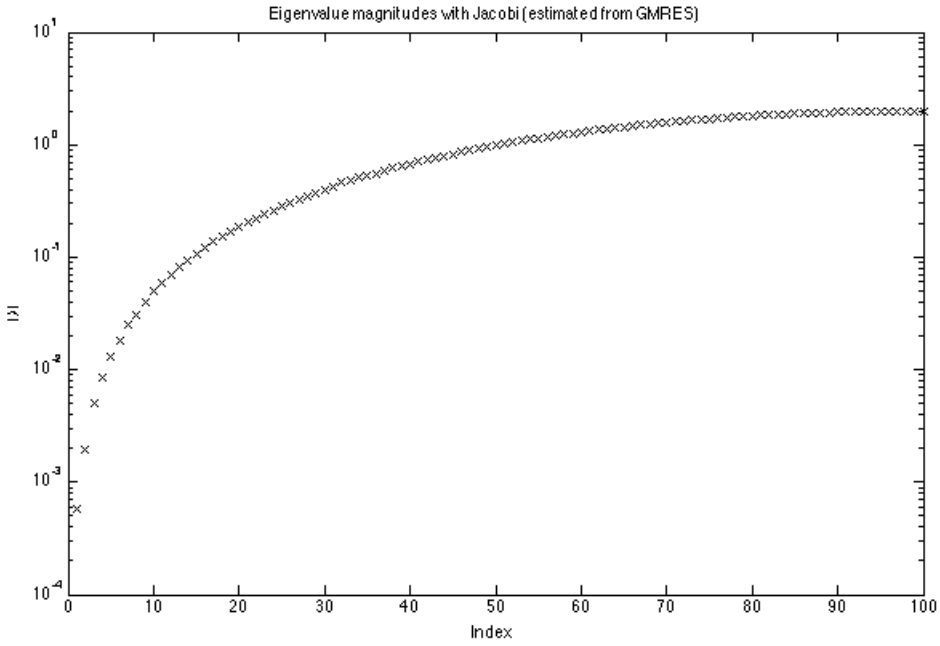
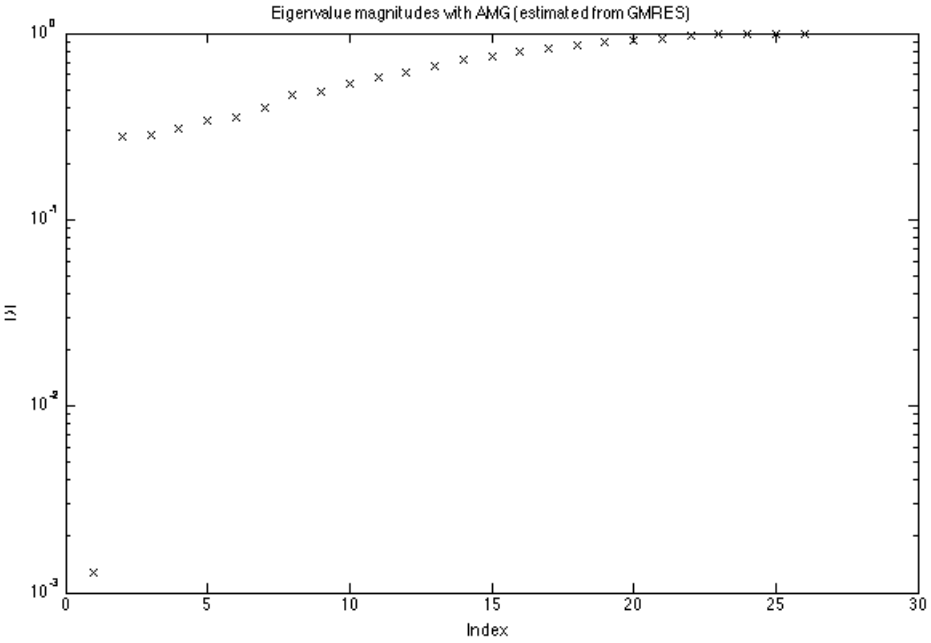


Figure 4.14: Eigenvalue magnitudes obtained for both the Jacobi and AMG preconditioners for a given timestep. We note that the number of eigenvalues obtained for AMG is less than the 100 obtained for Jacobi beacuse the AMG residual reaches machine precision in much fewer than 100 iterations.



of preconditioning. We note that in both cases, the eigenvalues obtained were positive real, which reinforces our belief that A is close to SPD. Looking at the plots, we see that, while the estimated conditioning of the Jacobi matrix is obviously inferior to that of the AMG matrix, the Jacobi conditioning isn't *too* bad.

In particular, we can use Equation (3.5) to get a rough estimate on the number of iterations it would require to obtain a relative reduction in residual of 1×10^{-4}, given our data. Solving for k, we see that achieving such a reduction for Jacobi would require about 314 iterations in the SPD case (estimating $\lambda_{max} \approx 2$ and $\lambda_{min} \approx 5 \times 10^{-4}$), where the same reduction for AMG would require about 9 iterations (estimating $\lambda_{max} \approx 1$ and $\lambda_{min} \approx 3 \times 10^{-1}$). While this is a large difference, the cost of an AMG iteration is much higher than that of a Jacobi iteration, and we have to take the AMG setup costs into account. If we assume that the cost of assembly can be amortized across the AMG iterations, in order for AMG to be more cost-efficient we would need the amortized cost of assembly plus the cost of an iteration of AMG to be less than $314/9 \approx 35$ times as expensive as the cost of a Jacobi iteration. Because the cost of AMG depends on the structure of the matrix A [32], it is not possible to give an exact statistic here on how expensive we expect the cost to be on average during our timestepping, but we can assert that this is the range where it is feasible that we might see Jacobi outperform AMG.

Given the above, while AMG provides a nice convergence rate and steady behavior throughout the simulation, it seems that for our application it is advisable to just stick with Jacobi due to the sheer number of linearizations and subsequent solves required.

4.6.3 Picard Restructuring

As we mentioned above, the Predictor-Corrector Picard iteration yielded results that varied numerically from the previous results by an "uncomfortable" amount (max-norm difference on the order of 1×10^{-2}). Visual inspection of the solution cross sections after 5 days (Figure 4.15) shows that the solution does not differ substantially in trends or overall character, but this is not the most rigorous measure. Because ground truth[3] is not available for these simulations, it is difficult to say whether the new structure is providing spurious results or not. It would be advisable to validate the solution against laboratory experiments to determine

[3]So to speak.

how it compares to the previous solution.

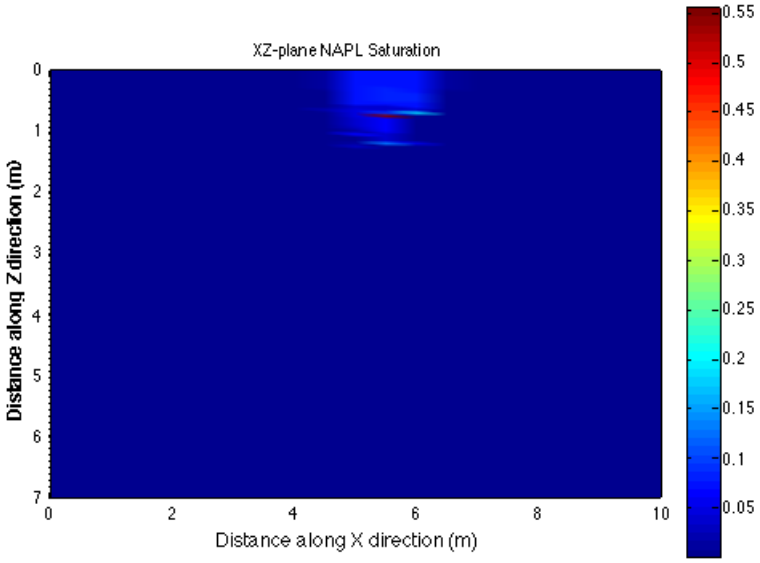
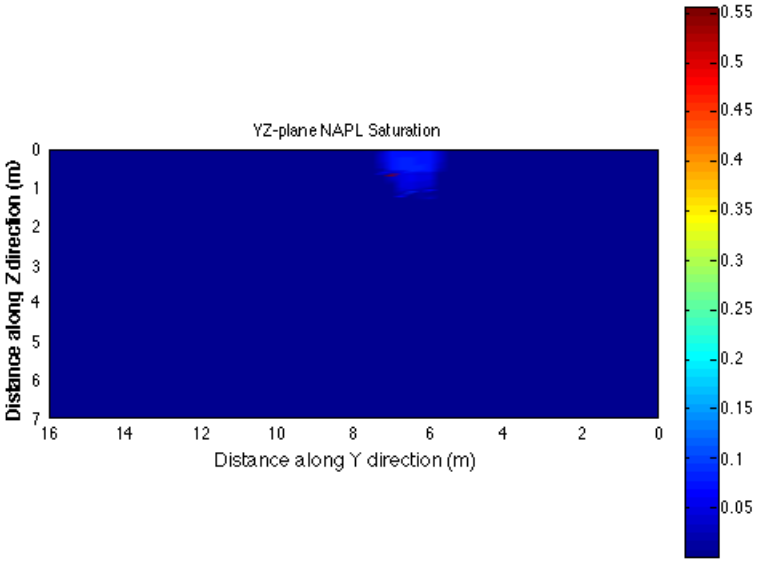


Figure 4.15: Cross-sections of the solution obtained after 5-days of Predictor-Corrector Picard iteration on our initial test problem. Compare to Figure 4.1.



We note additionally that the original adaptive timestepping scheme was developed for the original Parallel Picard iteration, and thus it mostly likely needs modification to be optimal with the alternative

algorithm structure. It is possible that a more fine-tuned method for changing the step-size could make the new linearization scheme comparable to the old in terms of computational cost and runtime.

Chapter 5

Conclusion

In this thesis, we first derived an expression for modeling contaminant transport in the subsurface, using conservation of mass and Darcy's law to obtain a system of partial differential equations relating the time evolution of the phase saturations and the phase pressures. After a brief review of finite-difference theory, we used the IMPES method to break the system into a discrete analogue with an implicit and explicit part – the pressure and saturation, respectively.

We used the GMRES(k) method and Picard linearization to solve the nonlinear IMPES equations, under a variety of configurations. Compared to the original implementation of Abriola et al. in [1], we observed the following results with a test problem generated to simulate real conditions:

- Porting the matrix assembly and pressure solve to C with the PETSc library improved runtime over the original Fortran version by about 1.7x. Modifying the code to use right-preconditioning and monitor the nonlinear residual from the Picard linearization brought this gain down to about 1.6x.
- Monitoring the relative residual and terminating the GMRES iteration when stagnation was evident yielded a further runtime improvement of just under 3x.
- Choice of residual tolerances for GMRES did not improve runtime by any substantial amount, but proved to affect accuracy when compared to the original Fortran solution (i.e., some choices caused solutions that agreed more with the Fortran output than others did).

- Varying the GMRES restart parameter showed a small, but repeatable improvement of 1.08x between the best-case ($k = 30$) and the previous parameter choice ($k = 10$).

- Use of an algebraic multigrid preconditioner proved to greatly improve GMRES efficacy, but setup cost made the method impractical. Attempts to improve runtime by preconditioner re-use were somewhat successful, but did not beat the Jacobi preconditioning runtimes.

- An alternative structure to the Picard linearization algorithm to separate the linearization of the pressure from the saturation solution proved slower than the previous structure, and additionally caused the final saturations to differ from the Fortran solution by as much as 1×10^{-2}.

Attempts to use AMG as a preconditioner for GMRES, as mentioned, suffered due to the sheer number of linearizations required during the timestepping process combined with the cost of assembling the preconditioner. This is unfortunate, as the AMG preconditioner proved very effective in actually solving the linear systems. Because of this fact, it would be interesting to look more at the use of AMG, perhaps in a different context. For example, in [32], a streamline method for applying multigrid to IMPES equations is discussed, which could potentially be used in our formulation.

The alternative linearization structure investigated in this thesis, as well, could warrant further investigation. Due to the lack of a knowledge of the true solution to the system, it is unclear whether the difference in solution in the new linearization as compared to the original structure is error, improvement, or simply "difference". Aziz and Settari imply in [4] that perhaps the nonlinear dependence of pressure on saturation is much more important and difficult to resolve than the self-dependence of pressure, which could account for the discrepancy, but further work is needed.

In total, we observed a 4.75x speed-up over the implementation in [1] for one test-problem, and a 6.25x speed-up for another with similar parameters. Both of these tests used right-preconditioned GMRES with Jacobi and tolerances as detailed in this thesis. The primary source of speed-up was clearly the implementation of the stagnation tolerance, which is a simple modification to the original algorithm that proves quite effective.

Appendix A

Calculating Capillary Pressure and Relative Permeability

A.1 Functional Forms for Hysteretic Relationship

In multiphase flow, where the available pore volume is occupied by two or more distinct, immiscible fluids, properly modeling the relation between capillary pressure of the fluids and their migration is critical to obtaining accurate simulations [1]. In a canonical paper [12], Brooks and Corey note the hysteretic nature of the capillary pressure / saturation and relative permeability / saturation relationships, concluding that, while hysteresis is an important phenomenon in modeling imbibition and drainage, a functional form can suffice in many cases. They present such a form for both the capillary pressure and relative permeability for two-phase flow.

In two-phase flow, one phase can be considered as the "wetting" phase, and the other the "non-wetting" phase, depending on how they interact in a capillary tube [12]. In our model, the wetting phase is water, and the non-wetting phase is the organic liquid. From empirical observation, Brooks and Corey noted that the log of the effective (or "normalized") saturation of the wetting phase is approximately a linear function of the log of the capillary pressure between the phases, i.e.,

$$\bar{S} \equiv \frac{S - S_r}{S_m - S_r} \approx \left[\frac{P_b}{P_c}\right]^{\lambda}. \tag{A.1}$$

Above, S is the wetting phase saturation, S_r is the residual wetting phase saturation, a constant dependent on the fluid and medium which describes the fluid volume that cannot be reduced hydraulically, and S_m is the maximum wetting phase saturation (which we assume to be identically one in our model). P_b is a media-dependent constant called the bubbling pressure, and the parameter λ is an empirical fitting parameter known as the pore size distibution index [1]. We obtain the functional form for the capillary pressure between the fluid phases, Equation (A.2), by rearranging Equation (A.1), yielding

$$P_c = P_b \bar{S}^{-1/\lambda}. \tag{A.2}$$

In [27], Parker, Lenhard, and Kuppusamy propose that once the above relationship is calculated for a given wetting fluid and media, it can be transformed to the correct relationship for a different wetting fluid via a scaling factor based on the fluid interfacial tensions. Following Abriola et al. in [1], we take the air/water capillary pressure as the datum and obtain the NAPL/water capillary pressure by scaling Equation (A.2) by the dimensionless parameter $\beta_{ow} = \sigma_{aw}/\sigma_{ow}$, where $\sigma_{\alpha\beta}$ is the $\alpha - \beta$ fluid pair interfacial tension.

The Brooks-Corey model of [12] also relates the relative permeability and normalized saturation using the same parameters as above, for both the wetting and non-wetting fluids. The relative permeability of the wetting fluid is given by

$$k_{rw} \approx \bar{S}^{(2+3\lambda)/\lambda} = \left[\frac{P_b}{P_c}\right]^{2+3\lambda}, \tag{A.3}$$

and, for the non-wetting fluid, we have

$$k_{rnw} \approx (1 - \bar{S})^2 (1 - \bar{S}^{(2+\lambda)/\lambda}) = \left(1 - \left[\frac{P_b}{P_c}\right]^{\lambda}\right)^2 \left(1 - \left[\frac{P_b}{P_c}\right]^{2+\lambda}\right). \tag{A.4}$$

A.2 Parker-Lenhard Model

In multiphase flow contaminant problems, another important dynamic for accurate transport modeling is the entrapment of the organic phase liquid during imbibition (wherein a volume of the contaminant becomes isolated by the water phase, bypassing the pores) [21]. In [26], Parker and Lenhard give a model for estimating the effect of the entrapped organic phase on the apparent saturation of the water phase.

In particular, they consider an empirical result by Land [23] in which the normalized residual organic saturation (corresponding to trapped organic saturation at zero capillary pressure) , $\bar{S}_{or}$, is given by

$$\bar{S}_{or} = \frac{1 - \bar{S}_w^{min}}{1 + R(1 - \bar{S}_w^{min})} \tag{A.5}$$

$$R = \frac{1}{\bar{S}_{or}^{max}} - 1, \tag{A.6}$$

where $\bar{S}_w^{min}$ is the minimum normalized water saturation that has occurred at the location since oil was introduced and $\bar{S}_{or}^{max}$ is the maximum normalized residual organic saturation,

$$\bar{S}_{or}^{max} \equiv \frac{S_{or}}{1 - S_{wr}}. \tag{A.7}$$

They note that this is the maximum normalized entrapped organic saturation, achieved if free organic liquid is continuously present during complete water imbibition, and then use simple linear interpolation to calculate the actual normalized entrapped organic saturation, $\bar{S}_{ot}$, as a function of $\bar{S}_w$,

$$\bar{S}_{ot} = \min\left\{\bar{S}_{or}\left(\frac{\bar{S}_w - \bar{S}_w^{min}}{1 - \bar{S}_w^{min}}\right), \bar{S}_o\right\}. \tag{A.8}$$

Here, we take the minimum of the calculated value with the total normalized organic saturation because we note total entrapped liquid cannot exceed the total present liquid [26].

Thus, during water imbibition, we use the Parker-Lenhard model to correct the normalized saturation for entrapment and define the apparent water saturation as

$$\bar{S}_w^{apprt} \equiv \min\left\{\bar{S}_w + \bar{S}_{ot}, 1\right\}, \tag{A.9}$$

where again we restrict the saturation to its theoretical maximum, one. This apparent saturation is then used to calculate the capillary pressure.

Appendix B

Test Problem Parameters

The following are the parameters used in the test problem for our numerical experiments.

- Gravity constant $g = 9.80665 m/s^2$
- 15 NAPL source nodes (1 L/day spill rate for 2.5 days)
- Compressibilities: $C_w = 4.4 \times 10^{-10}$/Pa, $C_o = 0$
- Reference Densities: $\rho_w^* = 999.032$ kg/m^3, $\rho_o^* = 1625$ kg/m^3
- Reference Pressures: $P_w^* = P_o^* = 1.013 \times 10^5$ Pa
- Viscosities: $\mu_w = 1.121 \times 10^{-3}$ Pa $\cdot$ s, $\mu_o = 8.9 \times 10^{-4}$ Pa $\cdot$ s
- System Temperature: 15° C
- Interfacial Tensions: $\sigma_{aw} = 72.75$ dyn/cm, $\sigma_{ow} = 47.8$ dyn/cm

Bibliography

[1] L. Abriola, K. Rathfelder, M. Maiza, and S. Yadav, *VALOR Code Version 1.0: A PC Code for Simulating Immiscible Contaminant Transport in Subsurface Systems*, Tech. Rep. EPRI TR-101018, The University of Michigan, Ann Arbor, Michigan, September 1992.

[2] L. M. Abriola and G. F. Pinder, *A multiphase approach to the modeling of porous media contamination by organic compounds: 1. equation development*, Water Resour. Res., 21 (1985), pp. 11–18.

[3] ——, *A multiphase approach to the modeling of porous media contamination by organic compounds: 2. numerical simulation*, Water Resour. Res., 21 (1985), pp. 19–26.

[4] K. Aziz and A. Settari, *Petroleum reservoir simulation*, Applied Science Publishers, 1979.

[5] S. Balay, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc users manual*, Tech. Rep. ANL-95/11 - Revision 3.2, Argonne National Laboratory, 2011.

[6] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Web page*, 2011. http://www.mcs.anl.gov/petsc.

[7] J. Bear, *Dynamics of fluids in porous media*, Dover, 1988.

[8] J. Bear and Y. Bachmat, *Introduction to modeling of transport phenomena in porous media*, Theory and applications of transport in porous media, Kluwer Academic Publishers, 1990.

[9] J. Bear and A. Verruijt, *Modeling groundwater flow and pollution: with computer programs for sample cases*, Theory and applications of transport in porous media, D. Reidel Pub. Co., 1988.

[10] M. Biot, *General theory of three-dimensional consolidation*, Journal of Applied Physics, 12 (1941), pp. 155–164.

[11] W. Briggs, V. Henson, and S. McCormick, *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics, 2000.

[12] R. H. Brooks and A. T. Corey, *Hydraulic Properties of Porous Media*, Hydrology Papers, Colorado State University, (1964).

[13] S. F. Carle, *T-PROGS: Transition Probability Geostatistical Software and User's Guide*, University of California, 1999.

[14] Z. Chen, G. Huan, and Y. Ma, *Computational methods for multiphase flows in porous media*, Computational science and engineering, Society for Industrial and Applied Mathematics, 2006.

[15] M. Embree, *The tortoise and the hare restart GMRES*, SIAM Review, 45 (2003), pp. 259–266.

[16] R. Falgout, A. Cleary, J. Jones, E. Chow, V. Henson, C. Baldwin, P. Brown, P. Vassilevski, and U. M. Yang, *HYPRE user's manual*, Tech. Rep. Version 2.0, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 2006.

[17] R. Falgout and U. Yang, *HYPRE : A library of high performance preconditioners*, in Computational Science - ICCS 2002, P. Sloot, A. Hoekstra, C. Tan, and J. Dongarra, eds., vol. 2331 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, pp. 632–641.

[18] H. Fogler, *Elements of chemical reaction engineering*, International series in the physical and chemical engineering sciences, Prentice-Hall, 1992.

[19] A. Greenbaum, *Iterative methods for solving linear systems*, Frontiers in applied mathematics, Society for Industrial and Applied Mathematics, 1997.

[20] V. E. Henson and U. M. Yang, *Boomeramg: a parallel algebraic multigrid solver and preconditioner*, Applied Numerical Mathematics, 41 (2000), pp. 155–177.

[21] J. J. Kaluarachchi and J. C. Parker, *Multiphase flow with a simplified model for oil entrapment*, Transport in Porous Media, 7 (1992), pp. 1–14. 10.1007/BF00617314.

[22] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, no. 16 in Frontiers in Applied Mathematics, SIAM, 1995.

[23] C. Land, *Calculation of imbibition relative permeability for two- and three-phase flow from rock properties*, Society of Petroleum Engineers Journal, 8 (1968), pp. 149–156.

[24] R. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*, Classics in Applied Mathematics, Society for Industrial and Applied Mathematics, 2007.

[25] S. Mehl, *Use of Picard and Newton iteration for solving nonlinear ground water flow equations*, Ground Water, 44 (2006), pp. 792–796.

[26] J. C. Parker and R. J. Lenhard, *A model for hysteretic constitutive relations governing multiphase flow: 1. saturation-pressure relations*, Water Resour. Res., 23 (1987), pp. 2187–2196.

[27] J. C. Parker, R. J. Lenhard, and T. Kuppusamy, *A parametric model for constitutive properties governing multiphase flow in porous media*, Water Resour. Res., 23 (1987), pp. 618–624.

[28] M. Putti and C. Paniconi, *Picard and Newton linearization for the coupled model for saltwater intrusion in aquifers*, Advances in Water Resources, 18 (1995), pp. 159 – 170.

[29] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, 2003.

[30] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[31] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM: Society for Industrial and Applied Mathematics, June 1997.

[32] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, 2001.