**NTNU – Trondheim**
Norwegian University of
Science and Technology

# Optimal non-linear solvers

Applications in reservoir simulation

## Svein Morten Drejer

Master of Science in Physics and Mathematics
Submission date: June 2014
Supervisor: Helge Holden, MATH
Co-supervisor: Knut-Andreas Lie, SINTEF

Norwegian University of Science and Technology
Department of Mathematical Sciences

**Abstract**

This thesis starts by giving a general introduction to hydrocarbon reservoirs, before physical relations and mathematical tools used to model the flow of the fluid components in the porous media are presented. The resulting partial differential equations, the black oil equations, describes the dynamics of the hydrocarbon components. Mathematical tools used to solve the PDEs are introduced, in particular the sequential splitting method. This approach results in a set of pressure and saturation equations, to be solved in sequence at each time step of the simulation. The finite volume method (FVM) is presented and used for the discretization of the differential equations, giving a coupled system of non-linear equations. The reordering approach due to Lie et al. [2013] is also presented, which reduces the equations from the FVM to a sequence of non-linear single cell problems. These residual equations can be solved by one-dimensional root finding methods. The theoretical background follows the implementation found in the open source reservoir simulator toolbox provided by the Open Porous Media Initiative (OPM) [OPM, 2014].

The goal of this work is to investigate and implement optimal numerical root finders for solving the singe cell residuals. In particular, the well known general root finding methods due to Brent [1973], Ridders [1979], and the field specific trust region methods due to Jenny et al. [2009], and Wang and Tchelepi [2013] are introduced and tested against the reference Regula Falsi solver used in the OPM . Numerical tests show that our implementation of the trust region method by Jenny et al. [2009] outperforms the reference in terms of CPU time when running the test cases without gravity effects. Further, the tests show that the general methods, namely the Ridders and Brent methods, are less efficient than the reference solver. Finally, numerical evidence is presented indicating that the trust region approach can be improved by providing better initial guesses.

**Sammendrag**

Denne oppgaven starter med en generell introduksjon av olje- og gassreservoir. Så presenteres fysiske relasjoner og matematiske verktøy brukt til å modellere fluidkomponentene i reservoiret. Disse betraktningene leder frem til et sett med partielle differensialligninger som beskriver flyten av hydrokarboner i steinformasjonene. Differensialligningene blir løst ved hjelp av et sett med matematiske verktøy, som også blir presentert her. Den sekvensielle oppsplittingsmetoden deler settet med ligninger i to nye ligningssett kalt trykkligningen og metningsligningen. Metoden løser de partielle differensialligningene sekvensielt, som gir et oppdatert trykk- og metningsfelt etter hvert tidssteg i simuleringen. Både trykk- og metningsligningene blir diskretisert ved hjelp av endelig volum-metoden (FVM), som muliggjør implementasjon og løsning av ligningene på en datamaskin. I utgangspunktet er disse ligningssettene sterkt koblet, slik at en må løse et system av ligninger. Lie et al. [2013] presenterer en reordningsmetode som gjør at en i stedet kan løse en serie én-celle-problemer, altså å finne røtter for en rekke én-dimensjonale ikke-lineære residualer. Teoridelen følger reservoirsimulatorimplementasjonen som blir tilbudt, som åpen kildekode, gjennom prosjektet The Open Porous Media Initiative (OPM) [OPM, 2014].

Målet med denne oppgaven er å undersøke og implementere optimale rotløsere for de ikke-lineære én-celle-problemene. De velkjente metodene introdusert av Brent [1973] og Ridders [1979] blir testet opp mot den eksisterende implementasjonen i OPM . Videre blir metodene presentert i [Jenny et al., 2009] og [Wang and Tchelepi, 2013] introdusert og testet. De numeriske tesetene viser at metoden til Jenny et al. [2009] konvergerer raskere, målt i CPU-tid, enn de øvrige metodene når gravitasjonseffekter ikke simuleres. Videre viser testene at de mer generelle metoden fra [Brent, 1973] og [Ridders, 1979] løser testproblemene saktere enn referanseløseren. Til slutt presenteres numeriske resultater som indikerer at metoden til Jenny et al. [2009] kan gjøres raskere ved å forbedre initialgjetningen til rotløseren.

# Contents

**Abstract** i

**Sammendrag** iii

**Preface** 1

**Introduction** 3

**1 Flow in Porous Media** 5
- 1.1 Introduction to Petroleum Reservoirs . . . 5
  - 1.1.1 Porous media . . . 5
  - 1.1.2 Driving Mechanisms of Production . . . 7
- 1.2 Petroleum Reservoir Modeling . . . 8
  - 1.2.1 The Continuity Equation . . . 9
  - 1.2.2 Fluid Models . . . 12

**2 Numerical Methods** 15
- 2.1 Sequential Splitting . . . 15
  - 2.1.1 The Pressure Equation . . . 17
  - 2.1.2 The Transport Equation . . . 19
  - 2.1.3 Mathematical Model . . . 20
- 2.2 The Finite-Volume Method . . . 22
- 2.3 Pressure Solver . . . 23
- 2.4 Transport Solver . . . 25
  - 2.4.1 Reordering . . . 30
  - 2.4.2 Root Finders . . . 33

**3 Numerical Results** 49
- 3.1 Test Procedure . . . 49
  - 3.1.1 The OPM Package . . . 50
- 3.2 Test Cases . . . 50
  - 3.2.1 Case A: Quarter Five Spot . . . 51

3.2.2 Case B: Tarbert 2D . . . . . 61
3.2.3 Case C: Upper Ness 2D . . . . . 65
3.2.4 Case D: Three-Dimensional Domain . . . . . 68
3.3 Convergence Tests . . . . . 71

**4 Discussion** **75**
4.1 Two-Dimensional Domains . . . . . 75
4.1.1 Case A - Large Grid Cells . . . . . 75
4.1.2 Case A - Small Grid Cells . . . . . 80
4.1.3 Case B and C . . . . . 81
4.2 Three-Dimensional Domain . . . . . 84
4.3 Convergence Tests . . . . . 85
4.4 Initialized Precise Trust Region . . . . . 87
4.4.1 Implementation . . . . . 87
4.4.2 Numerical Results . . . . . 87

**5 Conclusion** **93**
5.1 Further Work . . . . . 94

**Appendices** **95**

**A Test Drivers** **97**

# Preface

This report is the final obligatory thesis for my masters degree in Industrial Mathematics at the Norwegian University of Science and Technology (NTNU), course code TMA4910. The work was started in January 2014 and was handed in on the 19th of June 2014 and should represent 100% of this semesters total work load. The project has been done in collaboration with Chief Scientist Knut-Andreas Lie at the Department of Applied Mathematics at SINTEF in Oslo and professor Helge Holden at the Department of Mathematics at NTNU, Trondheim.

Svein Morten Drejer
Oslo/Trondheim
June 19th, 2014

# Introduction

Petroleum products are a major source of energy in the modern world and also see extensive use in other parts of industry e.g., in the production of plastics. These resources are usually found in subsurface petroleum reservoirs, located in porous rock, clay, and sand formations under the surface of the earth. The production of oil and gas requires large investments in hardware and man-hours, making the development of a new petroleum field a risky decision in terms of economics. Thus it is interesting to know as much as possible about the economic potential of a discovered reservoir in the exploration phase. Further, controlling the parameters of a reservoir during the production phase is a demanding and important task, since decisions can have a large impact on the hydrocarbon recovery percentage from the field.

A reservoir engineer will analyze experimental data gathered from field samples in order to predict the characteristics of a subsurface hydrocarbon reservoir, laying the groundworks for an economic analysis of the proposed development. A *reservoir simulator* is a useful tool for the reservoir engineer, allowing her to test different production scenarios, removing some uncertainty from the economic decisions. Modern computers allows larger and larger simulations to be performed, even on fairly standard desktop computers, and with good speed and accuracy.

Reservoir simulators are designed around a reservoir model, where the characteristics of the rock formations and fluids are described using physical insight and mathematical tools. Still, a full scale realistic simulation of fluid behavior on the pore scale of the rock is impractical. Instead, a discrete approximation of the reservoir is made, with rock formation data such as permeability and porosity averaged over larger portions of the reservoir. Further, the fluid parameters are also averaged over *control volumes* in the computational domain. These parameters include the pressure and saturation of the different fluid phases. Conservation principles are then applied to the fluids, resulting in systems of *partial differential equations* describing the dynamics of the fluids in the porous media. Only simple theoretical models have known closed form algebraic solutions, so in practice the sys-

tems of equations must be solved by numerical methods. These techniques uses mathematical tools to develop discrete approximations of the equations, giving problems that can be implemented and solved on computers.

One such computer implementation is provided by The Open Porous Media Initiative (OPM), developed as collaboration project between a number of industrial players and research institutions, see OPM [2014]. OPM is an open source library seeking to supply researchers with a broad selection of efficient reservoir simulation tools in an accessible format. The OPM library implements a range of numerical methods for solving the flow equations arising from the reservoir modeling. On such approach is to use the *sequential splitting scheme*, i.e., splitting the flow equations into one equation for the *pressure* of the fluid phases and a separate set of equations for the phase *saturations*, often called the *transport equations*. The most straight forward way of solving the transport equation involves solving a large non-linear system of equations where all cell saturation are solved for simultaneously. It is possible to *reorder* this set of equations based on the flow field found in the pressure solver such that one can solve a series of single cell or smaller coupled problems, thus reducing the computational effort. The single cell problems are on the form "find $x$ such that $f(x) = 0,\ f\colon \mathbb{R} \to \mathbb{R}$", that is, *root finding problems*. Currently the OPM library solves the single cell problems using a modified version of the *Regula Falsi method*. Several other root finding algorithms are known, both classic methods like the Newton-Raphson method, and more advanced methods like the Brent method. This project focuses on testing numerical methods for solving the single cell equations. A range of root finders are tested, among them Newton-Raphson-like methods with update heuristics, like the *Trust Region methods*, which we will study two examples of, namely the method due to Jenny et al. [2009], and the the more recent method due to Wang and Tchelepi [2013]. The root finders are implemented in the OPM framework and tested against the current Regula Falsi solver.

Chapter 1 gives a general introduction to petroleum reservoirs, before conservation principles and the conservation equation are introduced. Next, Chapter 2 starts by presenting the sequential splitting scheme and the finite volume discretization of the flow equations. The reordering approach is presented in Section 2.4.1, before the root finding algorithms used to solve the residual equations are discussed in Section 2.4.2. Then, Chapter 3 presents numerical results for a range of test cases in order to compare the new methods with each other and the existing solver in the OPM library. Finally, Chapter 4 discusses the numerical results, before the conclusions of the work are shown in Chapter 5.

# Chapter 1

# Flow in Porous Media

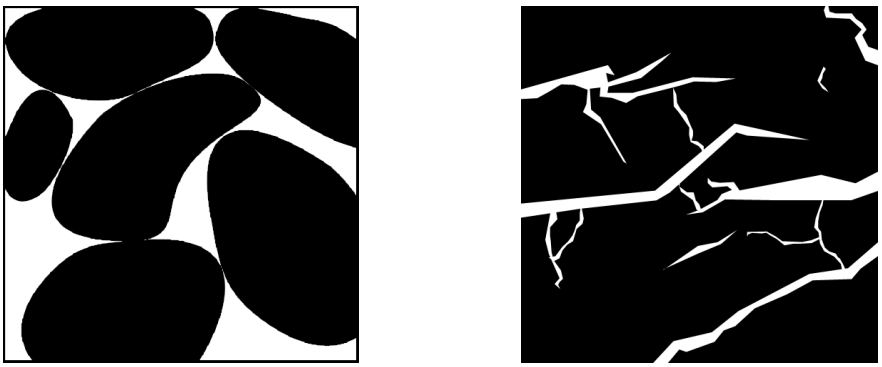
This chapter introduces petroleum reservoirs and the basic mathematical tools used to model fluid flow in porous media. We start out by giving a brief overview of porous media and petroleum reservoirs in Section 1.1, before the mathematical model for fluid flow is developed from conservation principles and constitutive relations in Section 1.2.

## 1.1 Introduction to Petroleum Reservoirs

### 1.1.1 Porous media

The term *porous media* encompasses a wide range of physical media containing void space, quantified by the *porosity* $\phi = \frac{\text{volume of void space in V}}{|V|}$, where $V$ is a connected region in the media at hand and $|V|$ is the volume of said region. Here the term *void space* is interpreted as areas of the material matrix not occupied by the material itself, that is, areas where for example fluids can reside. We also use the term *pore space* for these volumes. The total available *pore volume* in a rock sample is measured by the quantity $|V|\phi$, where $\phi$ is assumed to be a constant value for given regions of the media. Many seemingly solid everyday materials contain void space on a microscopic scale. Examples include wood, fabric, and, maybe more interesting, geological objects like rock and clay. Even "solid" rocks can contain a non-trivial void space, and it is in these cracks and crevices the hydrocarbon components in a petroleum reservoir are trapped. Void space in solid rock can be caused by either space between mineral grains, fractures, solution cavities in carbonate rock, or gas vesicles in volcanic rock [Jain, 2013]. Figure 1.1 illustrates the void space for the mineral grain and fracture type pore volumes.

For so-called *immiscible* (non-mixing) fluids the volume available for hy-



(a) Mineral grains (b) Fractures

Figure 1.1: Illustration of volume between mineral grains and fracture voids in rocks. White color indicates void space where fluids can reside. Black color indicates mineral structures.

drocarbons, the *hydrocarbon pore volume*, is limited by residual water in the pore space, called the *irreducible water saturation* $S_{\text{wc}}$ of the rock [Dake, 1978]. This water cannot be displaced by the hydrocarbon components, effectively reducing the available pore volume $\phi$ with a factor $S_{\text{wc}}$. Thus the hydrocarbon pore volume becomes $|V|\phi(1 - S_{wc})$. In the following it is assumed that the porosity is adjusted according to the irreducible water saturation, allowing us to use $|V|\phi$ for the hydrocarbon pore volume.

The porosity is obviously an essential parameter for a petroleum reservoir in that it limits the amount of space available for fluid components. What the porosity does not tell us about is the ease with which fluids flow through the formation. A rock with completely isolated pore spaces could in theory have a very high porosity, but without fluid flow between pore spaces oil and gas extraction would be impossible. Thus, the *permeability* $\boldsymbol{K}$ of the rock is introduced [Jain, 2013]. $\boldsymbol{K}$ measures the degree of interconnectivity between the pore spaces. A high permeability indicates that it is easy for fluids to pass through the rock. As here, $\boldsymbol{K}$ is often given as a tensor since the media in which fluid flows can be anisotropic. That is, the permeability is directional dependent and varies between the different spatial directions. Table 1.1 shows a few typical absolute value permeability ranges, along with a classification and examples of rock types with the relevant properties. The table is modified from Bear [1972].

The permeable regions where hydrocarbons flow are of little use if the valuable components escape to the surface. Laymen often think of oil and gas reservoirs as underground "pools" of fluids. The reality is not that far off, except that the geometry is inverted; light petroleum components es-

Table 1.1: Typical permeability ranges for petroleum reservoir rock formations. Modified from source: Table 5.5.1 in [Bear, 1972, p. 136].

| **Permeability** | **Rocks** | Range of $\log_{10}(K)[mD]$ |
|---|---|---|
| Pervious | Fracture rock | $10^8$ to $10^4$ |
| Semipervious | Oil Rock | $10^4$ to 10 |
| | Sandstone | 10 to 1 |
| Impervious | | 1 to 0.1 |
| | Dolomite | 0.1 to $10^{-3}$ |
| | Granite | $10^{-3}$ to $10^{-5}$ |

cape towards the surface and are trapped in an upside down pool by low-permeability geological formations, or in some cases by special hydrological phenomena [Jain, 2013]. Light components such as natural gas, if present, are found in the top layer, while the heavier oil is found just above the water aquifer in the bottom of the region.

### 1.1.2 Driving Mechanisms of Production

Petroleum components are harvested by drilling wells with perforations in the reservoir region, where pressure differences in the fluid drives it towards the surface. The natural pressure of the reservoir is often sufficient to drive the initial production. Continued production of hydrocarbon is driven by one or more of four mechanisms; solution gas drive, gas cap drive, natural water drive, and compaction drive [Dake, 1978]. Removal of fluids from the reservoir causes a pressure drop. When the pressure is lowered compressible components expand and push the fluid components out of the rock formations. This is the cause of the three first drivers. In particular, gas drive is caused by expansion of oil and gas in solution. A lowering of pressure causes these components to precipitate and expand the volume of fluids, causing an evacuation of the rock formation. A gas cap or an aquifer, if present, will also expand under lowered pressure, again pushing down (or up in the case of water) on the oil strata and forcing it out of the reservoir region. The last driving mechanism, compaction drive, is caused by a collapse in the rock formation following the removal of supporting fluids. The collapse of the rock matrix forces remaining fluid out of the void space. All of these processes are part of the *primary recovery* of the oil field. Primary recovery usually accounts for no more than 15% of the oil in place [Tzimas et al., 2005].

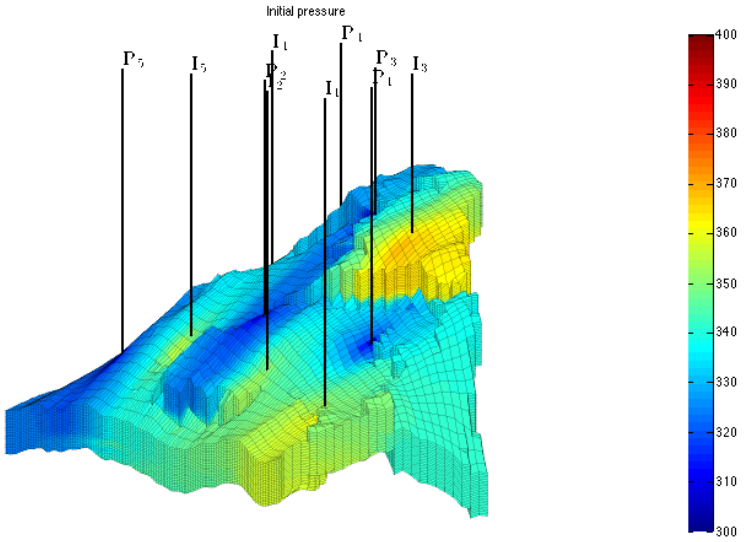



Figure 1.2: Example of a stratigraphic grid model of the Saigup field with wells and initial pressure data [Norsk Regnesentral, 2003].

After the natural pressure drive of the reservoir weakens so called *secondary recovery* is used. These techniques expend energy to increase the production potential of the reservoir. The most common secondary recovery technique is water injection, but other fluid types are also used. In the North Sea the primary and secondary oil recovery ranges between 45% and 55% of original oil in place [Green and Willhite, 2003].

The last category of the so called *enhanced oil recovery* techniques, *tertiary recovery*, seeks to alter the fluid and rock properties in the reservoir to improve the flow. These techniques are usually employed towards the end of the lifetime of a field, and are known to give an extra 5% to 15% of production [Tzimas et al., 2005]. It is worth noting that in modern petroleum reservoirs all three levels of recovery techniques are used in every part of the lifecycle of a field according to need, contrary to the hierarchical naming convention.

## 1.2 Petroleum Reservoir Modeling

An oil reservoir is a complex and extensive structure. To run fluid simulations on the full scale model we need to identify and store the important properties of the rock formations together with information about the fluids contained

within the hydrocarbon pore volume. These data points are gathered from the field by core samples, fluid samples, and seismic and electromagnetic geological exploration. The data gathered from field studies are compiled into a reservoir model containing all relevant parameters about the physical reservoir. Parameters like the permeability tensor $\boldsymbol{K}$, porosity $\phi$, phase saturation $S_l$ for phase $l$, and pressure $p$ are averaged and assigned to blocks representing subdomains of the model. This discrete version of the reservoir is closely connected to the discretized domain used when solving the fluid equations, as discussed in Section 2.1. These static parameters represent the geological model, which (at least in our discussion) does not change throughout the lifetime of the field. The reservoir model also includes any injection or production wells and relevant well equations. An example of a grid on a rock formation is shown in Figure 1.2. Here, the wells are shown as black lines and the pressure in each cell is indicated with color. This example uses a typical *stratigraphic* grid, which allows for a semi-structured grid while retaining the layered nature of the rock formations. It is on such discrete versions of the domain we will develop the flow equations. The reservoir model also includes a *fluid model*, a set of principles and equations chosen to model the hydrocarbon and water components present in the rock formations. Finally, the external interfaces of the reservoir are described. These include production and injection wells, and any fluxes across the outer boundaries of the reservoir, although *no-flow boundaries* are usually assumed. We start by deriving a *continuity equation* from the principle of *mass conservation*.

### 1.2.1 The Continuity Equation

Conservation of mass is an important concept in fluid dynamics. It effectively states that mass can be neither created nor destroyed. This implies that the amount of mass in a closed system is constant. Here "closed" is taken to mean closed to mass and energy transfer, since thermodynamical processes also cause mass transfer according to the principle of *mass-energy equivalence*. Even for thermodynamically open systems the conservation of mass is a relatively good approximation at reasonable energy levels. The continuity equation follows from conservation of mass by considering a *control volume* $V \subset \mathbb{R}^d$, $d \in \{1, 2, 3\}$, over which we track mass exchange, see Figure 1.3 for an example in two dimensions ($d = 2$). For a material with density $\rho$ we can compute the mass $m$ in the control volume at time $t$ by a volume integral of $\rho(\boldsymbol{x}, t)$ over $V$, where $\boldsymbol{x} \in \mathbb{R}^d$ is a point in $V$:

$$m = \int_V \rho(\boldsymbol{x}, t) \, \mathrm{d}V.$$

If the concentration of some quantity in $V$ is measured by $\varphi$ we can do a similar integral and compute the amount in the control volume at time $t$ by

$$\varphi_V(t) = \int_V \varphi(\boldsymbol{x}, t)\rho(\boldsymbol{x}, t)\ \mathrm{d}V.$$

This assumes that the conserved quantity is chemically inert, i.e., that there is no mass transfer between the conserved material and the other components in the control volume, and that no sources are present. We now open the

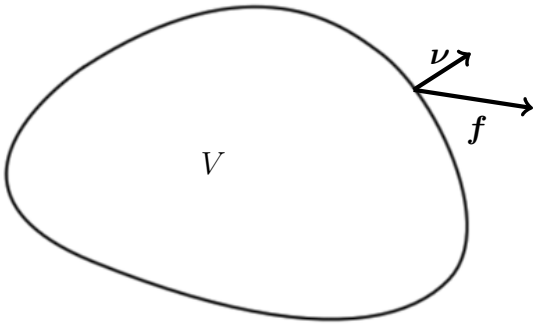



Figure 1.3: A control volume $V \subset \mathbb{R}^2$ with boundary $\partial V$, unit surface normal $\boldsymbol{\nu}$ and a flux $\boldsymbol{f}$.

boundary $\partial V$ of $V$ and start tracking the mass transfer out of the control volume. The movement across $\partial V$ sets up a flux $\boldsymbol{f}$. Letting $\mathrm{d}\boldsymbol{v}$ be an infinitesimal part of $\partial V$ with an outward facing unit normal $\boldsymbol{\nu}$ we can compute the mass transfer by

$$\int_{\partial V} \boldsymbol{f} \cdot \boldsymbol{\nu}\ \mathrm{d}\boldsymbol{v},$$

where $\boldsymbol{f} \cdot \boldsymbol{\nu}$ is the transport of the preserved quantity across $\mathrm{d}\boldsymbol{v}$. Further, the change in the concentration of the preserved quantity within $V$ is measured by the temporal derivative of $\varphi_V(t)$:

$$\frac{\partial \varphi_V(t)}{\partial t} = \text{change in } \varphi_V(t) \text{ during } \mathrm{d}t.$$

Sources, either negative (sinks) or positive (sources), are introduced through a source term $q(\boldsymbol{x}, t)$. Integrating over the control volume gives the total source term $q_{\text{tot}} = \int_V q(\boldsymbol{x}, t)\ \mathrm{d}V$. By convention, $q > 0$ is treated as an injection into the control volume. We now arrive at the complete conservation

principle as applied to the control volume $V$:

$$\frac{\partial \varphi_V(t)}{\partial t} = \int_V q(x,t)\ \mathrm{d}V - \int_{\partial V} \boldsymbol{f} \cdot \boldsymbol{\nu}\ \mathrm{d}v. \tag{1.2.1}$$

We now use the *divergence theorem*, see e.g. [Weber and Arfekn, 2003, p. 68-69], to relate the boundary flux to the divergence inside the control volume:

$$\int_{\partial V} \boldsymbol{f} \cdot \boldsymbol{\nu}\ \mathrm{d}v = \int_V \nabla \cdot \boldsymbol{f}\ \mathrm{d}V. \tag{1.2.2}$$

The boundary flux term now transforms directly to a control volume formulation, allowing us to gather the terms in the same integral, giving

$$\int_V \left[ \frac{\partial}{\partial t} \left[ \varphi(x,t) \rho(x,t) \right] + \nabla \cdot \boldsymbol{f} - q(x,t) \right]\ \mathrm{d}V = 0.$$

This works under the assumption of sufficient smoothness of the flux (for the divergence theorem) and the concentration and density (for the time derivative to move inside the integral), and by the linearity of the integral operation. Finally we arrive at the *continuity equation* for the quantity of concentration $\varphi$:

$$\frac{\partial}{\partial t} \left[ \varphi(x,t) \rho(x,t) \right] + \nabla \cdot \boldsymbol{f} = q(x,t). \tag{1.2.3}$$

Here we have used the fact that the control volume $V$ is chosen arbitrarily, which implies that we can drop the integral sign without destroying the equality. If needed the source term can be modified to track mass transfer between components. The mass conservation relation follows from Equation (1.2.3) by setting $\varphi = 1$ and defining the mass flux $\boldsymbol{f} = \rho \boldsymbol{u}$, where $\boldsymbol{u}$ is the fluid velocity. Using a subscript for the time derivative we get the mass continuity equation:

$$\rho_t + \nabla \cdot (\rho \boldsymbol{u}) = q. \tag{1.2.4}$$

So far we have assumed that the fluid phases can fill the control volume $V$ completely. As mentioned before, the porosity $\phi$ of the rock formations in the reservoir measures the available pore space. The porosity can be introduced into the continuity equation by letting it scale the total mass in the control volume, giving

$$(\phi \rho)_t + \nabla \cdot \boldsymbol{f} = q. \tag{1.2.5}$$

Here the temporal and spatial arguments are dropped for brevity. Equation (1.2.5) only models a single fluid *phase*. A more advanced fluid description is introduced in the next section.

### 1.2.2 Fluid Models

An important part of the reservoir model is the fluid description. The crude oil usually contains dissolved gas and the presence of a water phase is also common. A standard approach to fluid modeling is to use a compositional model where each hydrocarbon component, or at least a pseudo-component combining several chemical species, is subject to a mass balance equation. The fluid is described using three phases; the water, liquid and gas phase. In addition we introduce the mass fractions $C_{kg}$ and $C_{ko}$, that is, the mass fraction of component $k$ present in the gas and oil phase, respectively. Now the conditions $\sum_{k=1}^{n_c} C_{k\alpha} = 1, \alpha = \{g, o\}$, hold for a system with $n_c$ components. The mass-balance equations become

$$(\phi(C_{kg}\rho_g S_g + C_{ko}\rho_o S_o))_t + \nabla \cdot (C_{kg}\boldsymbol{f}_g + C_{ko}\boldsymbol{f}_o) = q_k,$$

for all components $k$, in addition to a standard continuity equation for water. The compositional fluid model will not be pursued further here.

At surface pressure and temperature the fluids from the reservoir separate into three phases; oil, gas, and water. The *black-oil model* assumes that this holds in the reservoir too. Three pseudo-phases are assumed; a liquid phase, a gaseous phase, and a water phase. The black-oil model includes gas in solution through the *solution gas-oil ratio*:

$$R_{so} = \frac{\text{volume of gas evolved from oil at std. conditions}}{\text{volume of oil at std. conditions}}.$$

The solution gas-oil ratio $R_{so}$ is used to modify the density of the oil phase in order to account for the dissolved gas. Assuming three fluid phases implies three conservation laws, one for each phase. We model each of these phases by defining the *phase saturation* $S_l$ of phase $l \in \{w, g, o\}$, denoting water, gas, and oil, respectively. The saturation of a phase measures the ratio of the amount of fluid of the given phase to the available hydrocarbon pore volume in the control volume $V$. The restriction

$$\sum_l S_l = 1, \tag{1.2.6}$$

called the *saturation constraint*, is rather obvious since we assume that the phases fill the entire available pore volume. In the two phase case the restriction becomes $S_w + S_o = 1$. Introducing the phase saturation into Equation (1.2.5) produces the phase continuity equation

$$(\phi S_l \rho_l)_t + \nabla \cdot \boldsymbol{f}_l = q_l. \tag{1.2.7}$$

Since the oil can contain gas in solution we need to modify the densities accordingly. Introducing the oil and gas density at standard condition, $\rho_o^s$ and $\rho_g^s$, respectively, the liquid oil density $\rho_o^l$ and the gaseous oil density $\rho_o^g$, we can express the reservoir density of oil as

$$\rho_o = \frac{\rho_o^s + \rho_g^s R_{\text{so}}}{B_o} = \rho_o^l + \rho_o^g,$$

where $B_o$ is the *formation volume factor*, the ratio of volume of oil at reservoir conditions to the volume of oil at standard (surface) conditions. That is,

$$B_o = \frac{\text{volume of oil at reservoir conditions}}{\text{volume of oil at std. conditions}}.$$

This gives the following set of black oil equations which we will use, where gas in solution is taken into account:

$$(\phi S_w \rho_w)_t + \nabla \cdot \boldsymbol{f}_w = q_w, \tag{1.2.8a}$$

$$(\phi S_o \rho_o^l)_t + \nabla \cdot \boldsymbol{f}_o = q_o, \tag{1.2.8b}$$

$$(\phi S_g \rho_g + \phi S_o \rho_o^g)_t + \nabla \cdot \boldsymbol{f}_l = q_g. \tag{1.2.8c}$$

# Chapter 2

# Numerical Methods

In practice we want to use the black oil model equations from Section 1.2.2 to predict fluid flow in the reservoirs. Closed form algebraic solutions are only available for the simplest problems, e.g. the Buckley-Leverett problem [Buckley and Leverett, 1942]. For real life reservoirs we need to use *numerical methods* to solve the system of equations. Different solution procedures have been proposed, and seen extensive use, throughout the years. Examples include, but are not limited to, the *simultaneous solution method* [Aziz and Settari, 1979; Molenaar, 1995], the *IMPES method* [Fagin, 1966; Coats, 2000; Aziz and Settari, 1979], and the *sequential implicit method*, also called the *sequential splitting method* or the *sequential solution method* [MacDonald, 1970; Spillette et al., 1973; Aziz and Settari, 1979; Aarnes et al., 2007, chap. 5]. The latter method will be presented and used here.

The *sequential splitting method* is presented in Section 2.1 before we introduce the *finite volume method* in Section 2.2 which we use to develop discrete fluid flow equations for the pressure and saturation in Sections 2.3 and 2.4, respectively. We conclude the chapter by presenting a number of numerical root finders used to solved the residual transport equations resulting from the discretization of the transport equation from Section 2.4.

## 2.1 Sequential Splitting

The black-oil equations, Equation (1.2.8), are coupled through the saturation constraint, Equation (1.2.6). To solve this coupled set of equations we want to rewrite the system to a form with a single unknown. To this end we introduce two tools; a per-phase version of *Darcy's law*,

$$\boldsymbol{u}_l = -\boldsymbol{K}\lambda_l(\nabla p_l - \rho_l \boldsymbol{g}), \tag{2.1.1}$$

and the *capillary pressure*

$$p_{cow} = p_o - p_w. \tag{2.1.2}$$

Darcy's law, first described by Darcy [1856], is a semi-empirical law relating pressure, gravity effects, and flow velocity of fluids in a porous medium. This formulation follows the velocity $\boldsymbol{u}_l$ and pressure $p_l$ of phase $l$. In Darcy's law, $\boldsymbol{K}$ is the absolute permeability tensor, and $\lambda_l$ the *mobility*, defined by

$$\lambda_l = \frac{k_{rl}}{\mu_l}. \tag{2.1.3}$$

Here the *relative permeability* for phase $l$, $k_{rl}$ is used. The relative permeability is modeled heuristically according to the properties of the fluid components in the reservoir. In the following $k_{rl} = S_l^2$ will be used. We also define the total mobility $\lambda = \sum_l \lambda_l$. Together, the absolute and relative permeability define the parameter $k_l = \boldsymbol{K} k_{rl}$, the permeability of phase $l$. This number quantifies the ease with which each phase moves through the rock formation. Here we will limit the discussion to a two-phase, immiscible, incompressible black-oil model. Thus we drop the gas equation and the $R_{\text{so}}$ part of the oil equation in Equation (1.2.8).

Together with boundary conditions, the multi-phase continuity equation and Darcy's law model the dynamics of the fluids in a reservoir through a coupled system of partial differential equations. Additional effects like compressibility can be accounted for within this framework, see e.g., [Aziz and Settari, 1979]. The sequential splitting method works by decoupling the system of equations into a pressure equation and a saturation equation, also called the transport equation. The decoupling is done by using the saturation constraint from Equation (1.2.6) together with the Darcy law in Equation (2.1.1) and the capillary pressure defined in Equation (2.1.2). These relations allow us to eliminate the oil variables $S_o$ and $p_o$ from the continuity equation and Darcy's law, giving two non-linear PDEs with the water saturation $S_w$ and water pressure $p_w$ as primary variables. Having obtained separate equations for the pressure and transport we can solve the two equations sequentially with separate implicit methods suited for each type of problem. We start out with an initial saturation in the reservoir, which is fed into the implicit pressure solver. This produces an updated velocity field $\boldsymbol{u}$. The transport solver uses this updated $\boldsymbol{u}$ to compute new saturations, after which the process is restarted. At each invocation of the transport solver (resp. pressure solver) the flux field (resp. saturation field) is assumed known. That is, the values are evaluated at the previous time step, making them explicit in nature. The primary unknowns in the equations are

evaluated at the current time step, making them implicit. This makes the sequential splitting method semi-implicit. Algorithm 1 shows pseudo code for the sequential splitting method. One assumes that this splitting introduces only small errors for incompressible reservoir simulations [Aziz and Settari, 1979, chap. 5.6]. In the next two sections we develop the pressure and transport equations in more detail.

**Algorithm 1:** Pseudo code implementing the sequential splitting scheme, see Section 2.1

**Data**: $s_0$, $t_{end}$, $\Delta t$, reservoir grid and parameters
**Result**: s

```
Initialize saturation field;
s = s_0;
Solve for initial pressure;
p = PRESSURE-SOLVER(s_0);
t = 0;
while time t is less than t_end do
    Solve transport equation with pressure assumed constant;
    s = TRANSPORT-SOLVER(s,p,Δt);
    Solve pressure equation with saturation assumed
    constant;
    p = PRESSURE-SOLVER(s);
    Advance time step;
    t = t + Δt;
end
```

### 2.1.1 The Pressure Equation

The derivation of the pressure equation loosely follows the notation and procedure from Aarnes et al. [2007] and Lie and Mallison [2013], and starts by assuming that the porosity $\varphi$ and density $\rho$ are constant in time, that is, incompressibility of rock formations and fluids. Now, by Equation (1.2.5), we obtain

$$\nabla \cdot (\rho_l \boldsymbol{u}_l) = q_l,$$

since the temporal derivative vanishes. The flux is defined to be a mass flux such that $\boldsymbol{f}_l = \rho_l \boldsymbol{u}_l$, with $\boldsymbol{u}_l$ being the velocity of the fluid. Note that the equation is taken to be per phase $l \in \{w, o\}$. Dividing by the density and substituting the velocity using the Darcy law in Equation (2.1.1) yields the

pressure equation for a single phase:

$$\nabla \cdot (-\boldsymbol{K}\lambda_l(\nabla p_l - \rho_l \boldsymbol{g})) = \frac{q_l}{\rho_l}.$$

Now we define the global velocity $\boldsymbol{u} = \boldsymbol{u}_w + \boldsymbol{u}_o$, giving an equation relating the water and oil pressure:

$$\nabla \cdot \boldsymbol{u} = -\nabla \cdot (\boldsymbol{K}\left[\lambda_w(\nabla p_w - \rho_w \boldsymbol{g}) + \lambda_o(\nabla p_o - \rho_o \boldsymbol{g})\right]) = q', \tag{2.1.4}$$

with a modified source term

$$q' = \frac{q_w\rho_o + q_o\rho_w}{\rho_w\rho_o}.$$

We still have both the oil and water pressure as unknowns. Following Chavent and Jaffre [1982] we define a saturation dependent complementary pressure $p_c$ by

$$p_c(s_w) = \int_{s_{\text{wc}}}^{s_w} f_w(s)\frac{\partial p_{\text{cow}}}{\partial s_w}(s)\ \mathrm{d}s. \tag{2.1.5}$$

Here, $s_{\text{wc}}$ denotes the irreducible water saturation discussed in Section 1.1 and we have defined the *fractional flow function* for phase $l$ by

$$f_l = \frac{\lambda_l}{\lambda}. \tag{2.1.6}$$

We note that in the two phase case the fractional flow function becomes

$$f_l = \frac{k_{rl}}{k_{rl} + Mk_{rn}}, \tag{2.1.7}$$

where $n$ indicates the second phase and the *viscosity ratio* $M$ is defined by

$$M = \frac{\mu_l}{\mu_n} > 0. \tag{2.1.8}$$

Figure 2.1 shows $f_w$ under the influence of different viscosity ratios. Note that $M < 1$ increases the $f_w$-value on the left hand side, while $M > 1$ lowers the $f_w$-values in the same region. Even moderate deviations from $M = 1$ causes significant changes in $f_w$. The complementary pressure equation in Equation (2.1.5) takes care of the saturation dependency of the capillary pressure, giving a looser coupling between the pressure and transport equation [Aarnes et al., 2007]. Taking the gradient of $p_c$ yields

$$\begin{aligned}\nabla p_c = \nabla \int_{s_{\text{wc}}}^{s_w} f_w(s)\frac{\partial p_{\text{cow}}}{\partial s_w}(s)\ \mathrm{d}s &= \left[f_w\frac{\partial p_{\text{cow}}}{\partial s_w}\right](s_w) - \left[f_w\frac{\partial p_{\text{cow}}}{\partial s_w}\right](s_{\text{wc}}) \\ &= \left[f_w\frac{\partial p_{\text{cow}}}{\partial s_w}\right](s_w) = f_w\nabla p_{\text{cow}},\end{aligned} \tag{2.1.9}$$

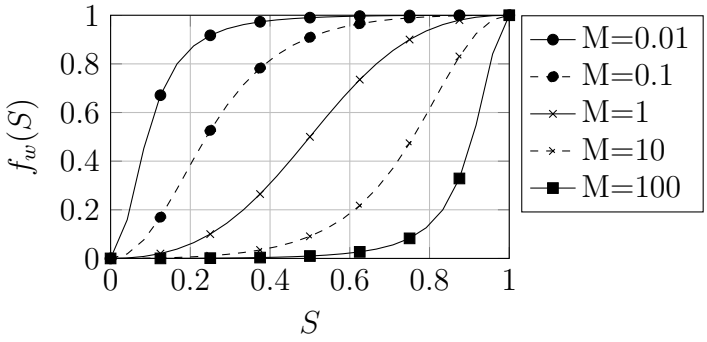



Figure 2.1: The fractional water flow function $f_w$, Equation (2.1.6), with quadratic $k_{rl}$ and viscosity ratio $M$, Equation (2.1.8).

by the fundamental theorem of calculus and the fact that $f_w(s_{\text{wc}}) = 0$. The purpose of $p_c$ is to define a *global pressure* $p$ by

$$p = p_o - p_c \tag{2.1.10}$$

in order to rewrite the pressure equation to be dependent on the global pressure and saturation only. Gathering the gradient pressure terms in Equation (2.1.4), and since $p_{\text{cow}} = p_o - p_w$, we get

$$\lambda_w \nabla p_w + \lambda_o \nabla p_o = \lambda_w(\nabla p_o - \nabla p_{\text{cow}}) + \lambda_o \nabla p_o = \lambda \nabla p_o - \lambda_w p_{\text{cow}}.$$

Using the relation from Equation (2.1.9) and the global pressure definition in Equation (2.1.10) we are able to express the gradients in terms of the global pressure only:

$$\lambda_w \nabla p_w + \lambda_o \nabla p_o = \lambda \nabla p_o - \lambda_w p_{\text{cow}} = \lambda \nabla p_o - \lambda_w \frac{\nabla p_c}{f_w} = \lambda(\nabla p_o - \nabla p_c) = \nabla p.$$

Inserting this relation into Equation (2.1.4) gives the global pressure equation, an *elliptic* equation for the global pressure $p$:

$$-\nabla \cdot (\boldsymbol{K}\left[\lambda \nabla p - (\lambda_w \rho_w + \lambda_o \rho_o)\boldsymbol{g}\right]) = q'. \tag{2.1.11}$$

### 2.1.2 The Transport Equation

Having found a pressure equation we need to complete the model by introducing the transport equation. We start out with the phase continuity equations in the black oil model as stated in Equation (1.2.8), but drop the gas terms. These equations already contain the time derivative of the saturation, but the flow velocity term must be removed in order to have a single

unknown. We do this by using Darcy's law from Equation (2.1.1) and the capillary pressure defined in Equation (2.1.2), as in Aarnes et al. [2007], to obtain

$$\boldsymbol{K}\nabla p_{cow} = \boldsymbol{K}(\nabla p_o - \nabla p_w) = (\boldsymbol{K}\rho_o \boldsymbol{g} - \frac{\boldsymbol{u}_o}{\lambda_o}) - (\boldsymbol{K}\rho_w \boldsymbol{g} - \frac{\boldsymbol{u}_w}{\lambda_w})$$

Inserting the total velocity $\boldsymbol{u} = \boldsymbol{u}_w + \boldsymbol{u}_o$ for $\boldsymbol{u}_o$ and multiplying by the mobilities, we get

$$\lambda_o\lambda_w \boldsymbol{K}\nabla p_{cow} = (\boldsymbol{K}\lambda_o\lambda_w\rho_o\boldsymbol{g} - \lambda_w\boldsymbol{u} + \lambda_w\boldsymbol{u}_w) - (\boldsymbol{K}\lambda_o\lambda_w\rho_w\boldsymbol{g} - \lambda_o\boldsymbol{u}_w).$$

Gathering terms and dividing by the total mobility $\lambda$ yields the following expression for the water velocity vector $\boldsymbol{u}_w$:

$$\boldsymbol{u}_w = f_w\boldsymbol{u} + \boldsymbol{K}\lambda_o f_w \nabla p_{cow} + \boldsymbol{K}\lambda_o f_w \boldsymbol{g}(\rho_w - \rho_o).$$

Here we have used the fractional flow function $f_w$, see Equation (2.1.6). Inserting this relation into the continuity equation, Equation (1.2.7), and assuming constant porosity and density gives the following equation for the water saturation, and in extension the oil saturation (by the saturation constraint in Equation (1.2.6)):

$$\phi\frac{\partial S_w}{\partial t} + \nabla\cdot(f_w[\boldsymbol{u} + \boldsymbol{K}\lambda_o\nabla p_{cow} + \boldsymbol{K}\lambda_o\boldsymbol{g}(\rho_w - \rho_o)]) = \frac{q_w}{\rho_w}. \tag{2.1.12}$$

This equation has both hyperbolic and parabolic properties [Aziz and Settari, 1979]. The coupled pressure and transport equations are solved using the procedure outlined in Algorithm 1.

### 2.1.3 Mathematical Model

The black oil model, and in extension the pressure and transport equations, describes the spatial and temporal variation of the properties of the fluids in the reservoir. We solve these equations on the spatial domain $\Omega \subset \mathbb{R}^d$, where $d \in \{2, 3\}$, from time $t = 0$ to the final time $t = T$, giving the domain $\Omega^+ := \Omega \times [0, T]$ for the partial differential equations, as sketched in Figure 2.2. The boundaries of this domain are denoted by $\partial\Omega^+ := \partial\Omega \times [0, T]$. To have a well posed problem, we need *initial* and *boundary conditions*. That is, we have to know the initial value at time $t = 0$ of all variables and how the equations behave at the boundaries $\partial\Omega^+$ of the domain $\Omega^+$. The initial condition $S_w(\boldsymbol{x}, 0) = S_w^0(\boldsymbol{x})$, $S_0 \colon \Omega \to [0, 1]$, allows us to compute the corresponding initial pressure field $p(\boldsymbol{x}, 0)$ by the pressure equation in

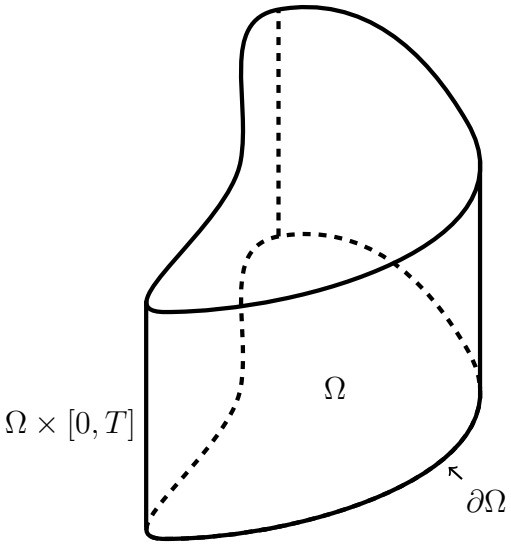



Figure 2.2: The spatial and temporal domain $\Omega^+ = \Omega \times [0, T]$ for the partial differential equations in the black oil model, Equation (1.2.8). The spatial domain has border $\partial\Omega$.

Equation (2.1.11), giving a complete initial condition. Common boundary conditions for reservoir simulations include flow rate (Neumann type) and pressure (Dirichlet type) conditions. The Dirichlet and Neumann part of the boundary denoted by $\partial\Omega_D$ and $\partial\Omega_N$, respectively. Note that $\partial\Omega_D \cap \partial\Omega_N = \emptyset$. The pressure boundary condition becomes

$$p(\boldsymbol{x}) = p_D(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \partial\Omega_D$$

where $\partial\Omega_D \subset \partial\Omega$ and $p_D \colon \partial\Omega_D \to \mathbb{R}_+$ is some scalar pressure function. Rate conditions can be specified as

$$\boldsymbol{v} \cdot \boldsymbol{\nu} = [-\boldsymbol{K}\lambda(\nabla p - \rho \boldsymbol{g})](\boldsymbol{x}) \cdot \boldsymbol{\nu} = Q_{\partial\Omega_N}(\boldsymbol{x}),\ \forall\ \boldsymbol{x} \in \partial\Omega_N,$$

using the unit surface normal $\boldsymbol{\nu}$ of $\partial\Omega$ and Darcy's law, see Equation (2.1.1). The magnitude of the rate at the Neumann part of the boundary, $\partial\Omega_N \subset \partial\Omega$, is defined by the function $Q_{\partial\Omega_N} \colon \partial\Omega_N \to \mathbb{R}$. The default rate boundary condition is a *no-flow* condition, i.e., $\boldsymbol{v} \cdot \boldsymbol{\nu} = Q_{\partial\Omega_N} = 0$, indicating that no fluid particles will cross the domain boundary.

The transport and pressure equation over the domain $\Omega^+$, along with

boundary conditions, combines to the following problem:

$$\begin{aligned}
\phi S_w(\boldsymbol{x},t)_t + \nabla\cdot(f_w\alpha(\boldsymbol{x},t)) &= q_w(\boldsymbol{x},t)\rho_w^{-1}, \quad (\boldsymbol{x},t)\in\Omega^+,\\
-\nabla\cdot(\boldsymbol{K}(\boldsymbol{x})\left[\lambda\nabla p(\boldsymbol{x},t) - (\lambda_w\rho_w+\lambda_o\rho_o)\boldsymbol{g}\right]) &= q(\boldsymbol{x},t), \quad (\boldsymbol{x},t)\in\Omega^+,\\
S_w(\boldsymbol{x},0) &= S_0(\boldsymbol{x}), \quad \boldsymbol{x}\in\Omega,\\
[-\boldsymbol{K}\lambda(\nabla p-\rho\boldsymbol{g})](\boldsymbol{x},t)\cdot\boldsymbol{\nu} &= Q(\boldsymbol{x},t), \quad (\boldsymbol{x},t)\in\partial\Omega^+,
\end{aligned}$$

where $\alpha(\boldsymbol{x},t) = [\boldsymbol{u}(\boldsymbol{x},t) + \boldsymbol{K}(\boldsymbol{x})(\lambda_o\nabla p_{cow} + \lambda_o\boldsymbol{g}(\rho_w-\rho_o))]$ and we have used Neumann boundary conditions. To solve these equations we will use the *finite volume method*, or FVM, as presented in the following section.

## 2.2 The Finite-Volume Method

The finite volume method is a discretization technique for solving differential equations. It is well suited for elliptic, parabolic, and hyperbolic equations, and is a natural choice for conservation laws because of the control volume formulation of the method and the fact that it lends itself to implementation on a wide range of grid types, including unstructured grids. The idea behind the method is to express a balance over each control volume, making the FVM *conservative* in the sense that the numerical flux is conserved between neighboring control volumes. In other words, the conservation of quantities over any group of control volumes is exact [Patankar, 1980]. Another strength is the natural and intuitive formulation of the method.

The finite volume method is defined over discrete control volumes of the domain. We proceed by using the domain $\Omega$ from Figure 2.2. We let $\mathcal{T}$ be a mesh on $\Omega$ such that $\bigcup_{V\in\mathcal{T}} V = \Omega$, where $V$ is a control volume. The finite volume method expresses an integral flux balance for each such control volume $V$. In general the control volumes can be of any shape, but a usual choice is to let every $V$ be a polygonal convex subset of $\Omega$ such that $V\cap K = \emptyset,\ \forall(V,K)\in\mathcal{T}\times\mathcal{T}, V\neq K$ [Eymard et al., 2003]. The collection of sides $s$ of the polygon $V$ is denoted $E_V$. Note that the term "polygonal" is used for both polygonal two-dimensional control volumes with $d=2$ and polyhedral three-dimensional control volumes with $d=3$. Figure 2.3 shows an example of a polygonal mesh on the two dimensional domain $\Omega$. Notice that the mesh coverage of the domain is only partial due to the straight edges of the grid cells. The error introduced by this discrepancy is assumed to be negligible in the theoretical setup. In practice the domain, i.e., the reservoir, consists of grid cells taken from the geological model of the rock formations. Such a grid is typically of a polyhedral type, removing the partial coverage problem altogether. The precise formulation of the FVM is introduced by applying it to the pressure and transport equations.

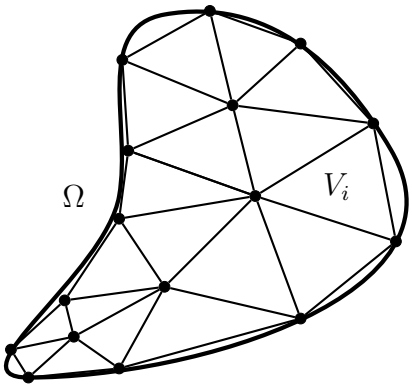



Figure 2.3: The domain Ω, see Figure 2.2, with a grid $\mathcal{T}$ consisting of triangular control volumes $V_i$.

## 2.3 Pressure Solver

The pressure equation in (2.1.11) is solved by the FVM method. We start by integrating over a grid cell $V$:

$$\int_V -\nabla \cdot (\boldsymbol{K}\left[\lambda \nabla p - (\lambda_w \rho_w + \lambda_o \rho_o)\boldsymbol{g}\right])\, \mathrm{d}V = \int_V q'\, \mathrm{d}V.$$

The left hand side integral is split into two parts, allowing us to isolate the pressure. Using the divergence theorem, and assuming that $\boldsymbol{K}\lambda\nabla p$ is smooth, we obtain

$$-\int_{\partial V} (\boldsymbol{K}\lambda\nabla p)\cdot \boldsymbol{\nu}\, \mathrm{d}v = \int_{\partial V} (\boldsymbol{K}(\lambda_w\rho_w + \lambda_o\rho_o)\boldsymbol{g})\cdot\boldsymbol{\nu}\, \mathrm{d}v + \int_V q'\, \mathrm{d}V. \tag{2.3.1}$$

Exploiting the polygonal geometry of the grid cells we can write

$$-\int_{\partial V} (\boldsymbol{K}\lambda\nabla p)\cdot\boldsymbol{\nu}\, \mathrm{d}v = -\sum_{s\in E_V}\int_s (\boldsymbol{K}\lambda\nabla p)\cdot\boldsymbol{\nu}_s\, \mathrm{d}v.$$

Thus, our task reduces to approximating the integral $\int_s (\lambda \boldsymbol{K}\nabla p)\cdot\boldsymbol{\nu}_s\, \mathrm{d}v$ on each edge $s$ of the cell. To this end we introduce a one-sided transmissibility $t_V^s$ defined by

$$t_V^s = \frac{\boldsymbol{\nu}_s \boldsymbol{K}_V \Delta \boldsymbol{c}_V^s}{\|\Delta \boldsymbol{c}_V^s\|^2},$$

where $\boldsymbol{\nu}_s$ is the surface normal of $s$ with magnitude equal to $m(s)$, $\boldsymbol{K}_V$ is the permeability tensor for the current cell, and $\Delta \boldsymbol{c}_V^s = \boldsymbol{c}_s - \boldsymbol{c}_V$ is the face-to-cell

centroid difference vector. Here $\boldsymbol{c}_V$ is the *centroid* of cell $V$ while $\boldsymbol{c}_s$ is the centroid of face $s$. Further, the function $m\colon \mathcal{T} \to \mathbb{R}_+$ is the $d$-dimensional *Lebesgue-measure*, which computes the "size" of the control volume, see e.g. [Eymard et al., 2003]. When $d = 2$ this function gives the *area* of the control volume $V$, while $d = 3$ gives the *volume*. We will also use the function $m\colon E \to \mathbb{R}_+$, the $d-1$-dimensional Lebesgue measure to be used on edges $s$ of $V$. No confusion should arise from this double definition of $m$ since the correct version should be apparent from the context. Now we can express the integral on the edge $s$ connecting $V$ and $K$ using the *two-point flux approximation* scheme, the TPFA scheme, expressed as

$$\int_s (\boldsymbol{K}\nabla p_s)\cdot\boldsymbol{\nu}\,\mathrm{d}v = (p_V - p_K)\left(\frac{1}{t_V^s}+\frac{1}{t_K^s}\right)^{-1} = \frac{t_V^s t_K^s}{t_V^s + t_K^s}(p_V - p_K).$$

A mobility weighted version becomes

$$\int_s (\lambda\boldsymbol{K}\nabla p_s)\cdot\boldsymbol{\nu}\,\mathrm{d}v = (p_V - p_K)\left(\frac{1}{\lambda_V t_V^s}+\frac{1}{\lambda_K t_K^s}\right)^{-1},$$

where $\lambda_V$ is the total mobility in $V$. This result is inserted into Equation (2.3.1), such that we obtain

$$-\sum_{s\in E_V}(p_V - p_K)\left(\frac{1}{\lambda_V t_V^s}+\frac{1}{\lambda_K t_K^s}\right)^{-1} = \int_{\partial V}(\boldsymbol{K}(\lambda_w\rho_w+\lambda_o\rho_o)\boldsymbol{g})\cdot\boldsymbol{\nu}\,\mathrm{d}v + \int_V q'\,\mathrm{d}V.$$

The right hand side of Equation (2.3.1) is approximated in a similar manner. The integral of the gravity term over the boundary is approximated by the following relation:

$$\int_{\partial V}(\boldsymbol{K}(\lambda_w\rho_w+\lambda_o\rho_o)\boldsymbol{g})\cdot\boldsymbol{\nu}\,\mathrm{d}v = \sum_{s\in E_V}\boldsymbol{g}[\Delta\boldsymbol{c}_V^s\omega_V + \Delta\boldsymbol{c}_K^s\omega_K]\left(\frac{1}{\lambda_V t_V^s}+\frac{1}{\lambda_K t_K^s}\right)^{-1},$$

where $\omega_V = \frac{\lambda_{wV}\rho_w+\lambda_{oV}\rho_o}{\lambda_V}$. The source term $q'$ is simply integrated over the control volume and expressed as a discrete value $q'_V$ for each $V$. This results in the following linear system to be solved for the pressure in each control volume $V$:

$$-\sum_{s\in E_V}(p_V - p_K)T_s = \boldsymbol{g}[\Delta\boldsymbol{c}_V^s\omega_V + \Delta\boldsymbol{c}_K^s\omega_K]T_s + q'_V,\ \forall V\in\mathcal{T}.$$

Here we have defined the *mobility weighted transmissibility* $T_s$ by

$$T_s = \left(\frac{1}{\lambda_V t_V^s}+\frac{1}{\lambda_K t_K^s}\right)^{-1},$$

where $K$ is the unique neighbor cell to $V$ such that $\partial V \cap \partial K = s$.

The next section introduces the finite volume method applied to the transport solver. Since the method essentially expresses a balance equation over the control volume at hand we will need to know the fluid fluxes across the boundary $\partial V$. One of the assumptions of the sequential splitting method is that these face fluxes can be computed based on the pressure field from the current iteration. The face fluxes $F_s$ for face $s$ are computed by

$$F_s = T_s(p_V - p_K + F_s^g),\ \forall s \in E, (V,K) \in \mathcal{T} \times \mathcal{T} : \partial V \cap \partial K = s, \quad (2.3.2)$$

where the gravity flux $F_s^g$ is defined as

$$F_s^g = (\Delta \boldsymbol{c}_V^s + \Delta \boldsymbol{c}_K^s)\boldsymbol{g}.$$

## 2.4 Transport Solver

The OPM code assumes that the transport problem can be solved in two steps by splitting Equation (2.1.12) into a buoyant and a viscous-capillary equation. That is, first

$$\phi \partial_t S_w + \nabla \cdot (f_w[\boldsymbol{u} + \lambda_o \boldsymbol{K} \nabla p_{cow}]) = q_w(\boldsymbol{x}, t)\rho_w^{-1}, \quad (\boldsymbol{x}, t) \in \Omega^+ \quad (2.4.1)$$

is solved for the saturation influenced by viscous and capillary forces, and sources before

$$\phi \partial_t S_w + \nabla \cdot (f_w \lambda_o \boldsymbol{K} \boldsymbol{g}(\rho_w - \rho_o)) = 0, \quad (\boldsymbol{x}, t) \in \Omega^+ \quad (2.4.2)$$

is solved for the gravity influenced saturation. The variables $\boldsymbol{x}$ and $t$ are dropped for brevity. We start by integrating the viscous-capillary transport equation from Equation (2.4.1) over each control volume $V \in \mathcal{T}$:

$$\int_V \phi \partial_t S_w(\boldsymbol{x}, t) + \nabla \cdot (f_w[\boldsymbol{u} + \lambda_o \boldsymbol{K} \nabla p_{cow}]) - \frac{q_w(\boldsymbol{x}, t)}{\rho_w}\ \mathrm{d}V = 0,\ \forall V \in \mathcal{T}.$$

This gives

$$\phi_V \frac{\partial}{\partial t} \int_V S_w\ \mathrm{d}V + \int_{\partial V} (f_w[\boldsymbol{u} + \lambda_o \boldsymbol{K} \nabla p_{cow}]) \cdot \boldsymbol{\nu}\ \mathrm{d}v - \int_V \frac{q_w}{\rho_w}\ \mathrm{d}V = 0,\ \forall V \in \mathcal{T} \quad (2.4.3)$$

by the divergence theorem and under the assumptions that $S_w$ is sufficiently smooth and that the porosity $\phi$ is a given constant $\phi_V$ for each grid cell. We now express the cell averaged water saturation $S_V$ for cell $V$ as

$$S_V = \frac{1}{m(V)} \int_V S_w\ \mathrm{d}V. \quad (2.4.4)$$

The number $S_V$ will be used as a representation of the saturation in the cell and is one of the primary variables in the final system of equations. Now the source term is integrated over $V$, giving a discrete source

$$q_V = \int_V q_w \, \mathrm{d}V. \tag{2.4.5}$$

This leaves only the treatment of the boundary integral term. Letting $s \in E_V$ be the edges of $V$ and $\boldsymbol{\nu}_s$ be the outward facing unit normal of the edge $s$, we can express the boundary integral as

$$\int_{\partial V} (f_w[\boldsymbol{u} + \lambda_o \boldsymbol{K} \nabla p_{cow}]) \cdot \boldsymbol{\nu} \, \mathrm{d}v = \sum_{s \in E_V} \left[ \int_s (f_w[\boldsymbol{u} + \lambda_o \boldsymbol{K} \nabla p_{cow}]) \cdot \boldsymbol{\nu}_s \, \mathrm{d}v \right],$$

since $\partial V = \bigcup_{s \in E_V} \bar{s}$. Here $\bar{s}$ is the *closure* of side $s$. The pressure solver handles each edge integral, see Section 2.3, but a few comments are in order here regardless. The *upwind method* will be used to compute the interface fluxes. That is, on each edge $s$ shared by two control volumes, say $V$ and $K$, a scalar approximation $F_s$ of the flux is chosen such that the information is gathered in the cell the flow is coming from. This flux was calculated by the pressure solver, and is shown in Equation (2.3.2). The fluxes over $\partial V$ can be categorized as either incoming or outgoing fluxes. The set of edges with incoming fluxes for cell $V$ is denoted $E_V^+$, while the set of edges with outgoing fluxes is denoted $E_V^-$. The fractional flow value for the incoming fluxes are independent of the local cell saturation $S_V$ and distinct for each edge, and will be denoted by $f_s$. This allows us to denote the incoming flow as

$$Q_V^+ = \sum_{s \in E_V^+} f_s F_s$$

and the outgoing flow as

$$Q_V^- = \sum_{s \in E_V^-} f_w(S_V) F_s = f_w(S_V) \sum_{s \in E_V^-} F_s = f_w(S_V) F_V^-,$$

where $F_V^-$ is the total outgoing flux. Note that because of the upwind method only the flow out of cell $V$ is influenced by the local saturation $S_V$. Summing the flow terms over all edges of $V$ yields

$$\sum_{s \in E_V} \left[ \int_s (f_w[\boldsymbol{u} + \lambda_o \boldsymbol{K} \nabla p_{cow}]) \cdot \boldsymbol{\nu}_s \, \mathrm{d}v \right] = f_w(S_V) F_V^- + Q_V^+.$$

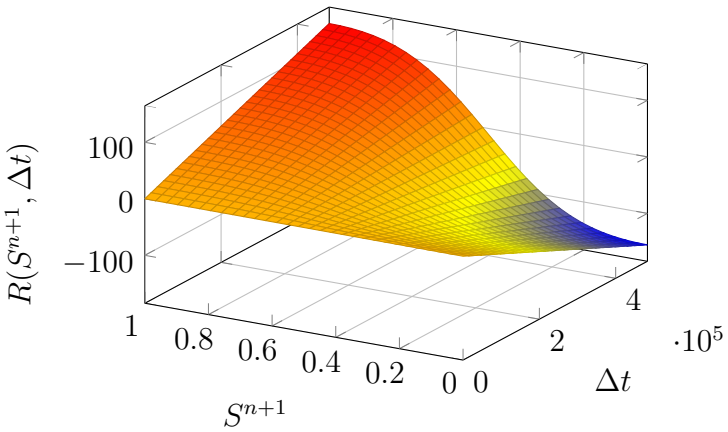



Figure 2.4: An example of the single cell residual in Equation (2.4.7) as a function of $\Delta t$ and $S^{n+1}$.

Inserting this into Equation (2.4.3), using (2.4.4) and (2.4.5) and dividing by the cell "volume" $m(V)$ and the porosity $\phi_V$ we obtain

$$\frac{\partial S_V}{\partial t} + \frac{1}{m(V)\phi_V}\left[f_w(S_V)F_V^- + Q_V^+\right] - \frac{q_V}{\rho_w \phi_V} = 0, \ \forall V \in \mathcal{T} \tag{2.4.6}$$

By averaging values over the control volume and using the upwind method we have arrived at a semi-discretized version of the transport equation. Now we must choose a technique for resolving the time derivative in the first term of the Equation (2.4.6). We approximate the derivative by

$$\frac{\partial S_V}{\partial t} = \frac{S_V^{n+1} - S_V^n}{\Delta t} + \mathcal{O}(\Delta t),$$

where the superscript $n$ denotes the current *time level* corresponding to the chosen *time step* $\Delta t$. That is, the current time is $t = n\Delta t$, where $n \in [0, 1, 2, \ldots, n_{\max}]$ and $n_{\max} = \lceil \frac{T}{\Delta t} \rceil$. Now we can choose between an *explicit* and an *implicit* scheme by setting the time level of the other terms in the equation. Explicit difference schemes put severe restrictions on the time step $\Delta t$, e.g. through a CFL condition, as first described in Courant et al. [1928], and becomes unstable for time steps exceeding this limit. Implicit schemes are much more robust and are known to give unconditional stability, see e.g. [Aziz and Settari, 1979]. We want to exploit the extra stability of the implicit scheme, and thus choose to evaluate the other $S_V$-dependent terms at the new time level, that is, $f_w = f_w(S_V^{n+1})$.

One remark is in order here. In writing out Equation (2.4.6) we have made a few shortcuts by skipping the dependent variables of the various

terms. The cell saturation is obviously time dependent, but the interface fluxes $F_s$ also have a saturation dependency. In a pure implicit approach these saturation values should also be taken at the new time level $n+1$, but the assumption of known interface fluxes implies $F_s = F_s(S_V^n)$, that is, the fluxes are evaluated at the current time level. This is an explicit approach. This mixing of implicit and explicit terms gives rise to the semi-implicit nature of the sequential splitting method (a similar approach is used in the IMPES method). Inserting the time derivative approximation and using the implicit scheme we arrive at the residual form of the discrete viscous-capillary transport equation, plotted for a single cell $V$ as a function of $\Delta t$ and $S_V^{n+1}$ in Figure 2.4:

$$R(S_i^{n+1}; S_i^n, \Delta t) \coloneqq S_V^{n+1} - S_V^n - \frac{\Delta t}{m(V)\phi_V}\left[f_w(S_V^{n+1})F_V^- + Q_V^+\right] - \frac{q_V \Delta t}{\rho_w \phi_V} \tag{2.4.7}$$

$$= 0, \ \forall V \in \mathcal{T}. \tag{2.4.8}$$

A similar approach is used on Equation (2.4.2), the gravity transport equation. The OPM code assumes that the grid for this problem is aligned in vertical columns, which holds for the stratigraphical grids often used in reservoir simulation packages, as discussed in Section 1.1. Further it assumes that the gravity effects are only influencing the saturation in cells above or below a cell, allowing solution of the transport equations on a per column basis. The gravity terms on the interface to neighboring cells are approximated using the transmissibility and a centroid difference, as was the case for the viscous-capillary equation. These boundary fluxes are gathered in an edge flux variable, say $F_s^g$ for each edge $s \in E_V$, and are constant throughout a simulation since they only depend on permeabilities, constant densities, and the grid configuration. Note that the flux on edges in the $x$-$z$ and $y$-$z$ planes are zero, since the cells are assumed to be vertically aligned and the gravitational influence only works in the vertical direction. The FVM requires the mobilities $\lambda_l$ to be evaluated on each interface edge $s$, a task again accomplished by the upwind method. Since gravity causes the lightest phase to move upwards the mobility for this phase must be taken from the cell below the current edge. Likewise the mobility for the heavy phase is gathered from the cell above the current edge. Figure 2.5 illustrates this for a heavy water phase and a light oil phase. Denoting the top face of cell $V$ as $s_t$ and the

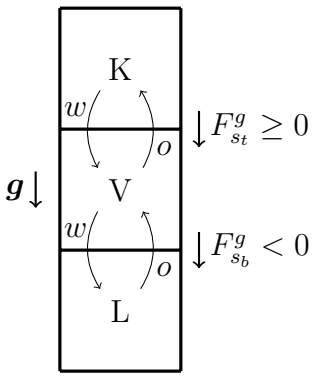



Figure 2.5: The flow of a light phase $o$ and a heavy phase $w$ with vertically aligned cells $K, V, L$ and interface edges $s_t$ and $s_b$.

bottom face as $s_b$, we arrive at the following residual equation to be solved:

$$\begin{aligned} R_g(S_i^{n+1}) &:= S_V^{n+1} - S_V^n \\ &- \frac{\Delta t}{m(V)\phi_V}\left[\frac{\lambda_w(S_K^n)\lambda_o(S_V^{n+1})}{\lambda_w(S_K^n)+\lambda_o(S_V^{n+1})}F^g_{s_t} + \frac{\lambda_w(S_V^{n+1})\lambda_o(S_L^n)}{\lambda_w(S_V^{n+1})+\lambda_o(S_L^n)}F^g_{s_b}\right] \\ &= 0, \ (K,L) \in \mathcal{T}\times\mathcal{T} : K \cap V = s_t, L \cap V = s_b, \ \forall V \in \mathcal{T}. \quad (2.4.9) \end{aligned}$$

Here the phase mobilities $\lambda_l$ are evaluated explicitly in the neighboring cells using the cell saturation $S_K^n$ and $S_L^n$ according to the configuration in Figure 2.5. That is, $\lambda_w(S_K^n)$ and $\lambda_o(S_L^n)$ are known a priori when solving Equation 2.4.9. The single cell gravity residual is shown in Figure 2.6.

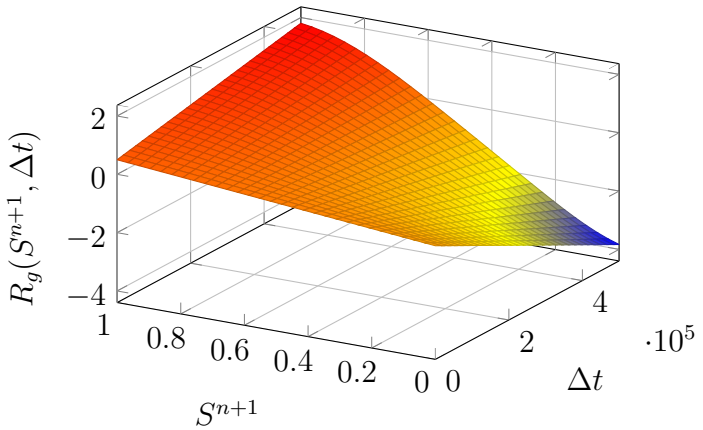



Figure 2.6: An example of the single cell gravity residual in Equation (2.4.9) as a function of $\Delta t$ and $S^{n+1}$.

We now want to solve Equations (2.4.7) and (2.4.9) by finding roots of

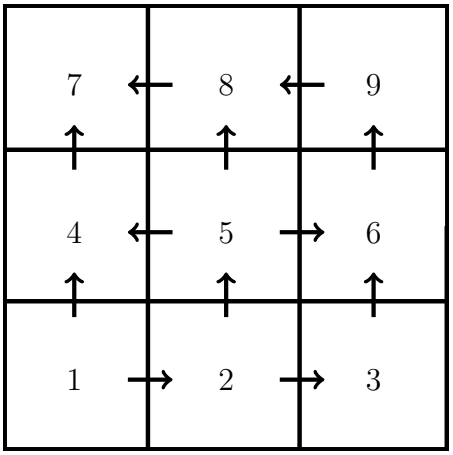
the residuals $R(S_V^{n+1})$ and $R_g(S_V^{n+1})$. Existence of solutions of these residual equations is hard to prove with rigor. Despite this a solution is assumed to exist for well-posed reservoir simulation residuals for every time step $\Delta t$ [Younis et al., 2010]. Further, if brackets $[a, b]_R$ and $[c, d]_{R_g}$ can be found according to Definition 1 we know that a solution exists by Theorem 2.1 and the continuity of the residuals.

### 2.4.1 Reordering

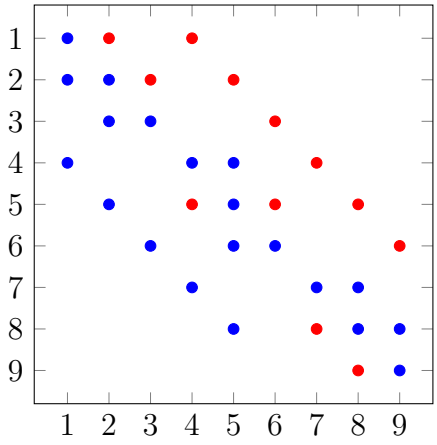
The upwind method is used for the discretization of the flow equations. This choice of discretization ensures that the state of a cell $i$ is only affected by the state in the upwind neighboring cells $U(i)$, creating a well defined domain of dependence for each cell. The upwind direction is based on the fluid flux on the border between neighboring cells. An example of a discretization with interface flux directions is shown in Figure 2.7 along with the sparsity pattern resulting from a standard numbering of the cells in the grid. The state of each cell is influenced by the neighboring cells according to Equation (2.4.6), giving the sparsity pattern in Figure 2.7b. In each row $i$ in Figure 2.7b the red dots mark neighboring cells $j$ where the interface flux goes from $i$ to $j$. That is, cell $j$ is *downwind* relative to cell $i$ under the given flux field. Using the upwind method, the state in cell $i$ is invariant under the state of downwind cells, i.e., red cells $j$ in row $i$ in the figure. Therefore the coefficient corresponding to the red dots in the sparsity pattern can be set to zero in the system of equations, effectively reducing the computational complexity of the problem. Note, however, that we still must use a full matrix solve since there are non-zero values on the super-diagonal. This can be amended by *reordering* the cells in the domain according to the flow direction, as described in the following.

The approach can be motivated by viewing the domain as a directed graph with the cells as nodes and the interface fluxes determining the edge directions between nodes. In computer science, a topological sort is an algorithm designed to order the nodes in a directed graph according to the direction of the interconnecting edges. The sorting algorithm provides a list of nodes such that all edges from every node points to nodes with a higher ordering in the list. In fluid flow terms this approach provides an ordering of the cells according to the domain of influence for each cell, as defined using the upwind method on the interface terms. A cell early in the ordering is independent of the subsequent cells in the list, allowing the state of each cell in the ordering to be computed sequentially. In other words, the new numbering gives a lower triangular matrix which indicates that the system of equations for the cell saturations can be solved sequentially by a forward sub-

stitution. Figure 2.8 shows the cell numbering and sparsity pattern resulting from a topological sort of the cells from the example domain in Figure 2.7. Note the lower triangular structure of the matrix after setting the coefficient of downwind cells to zero, that is, cells marked with red dots.

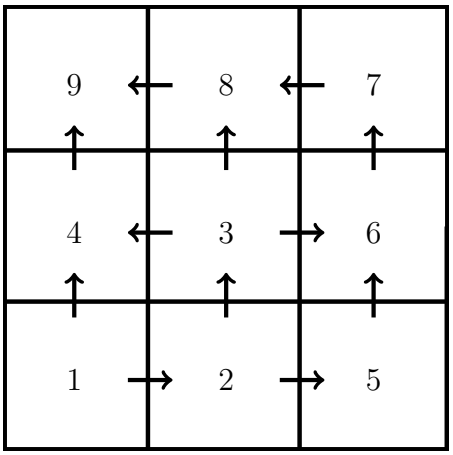



(a) Cell numbering and fluid flow direction.

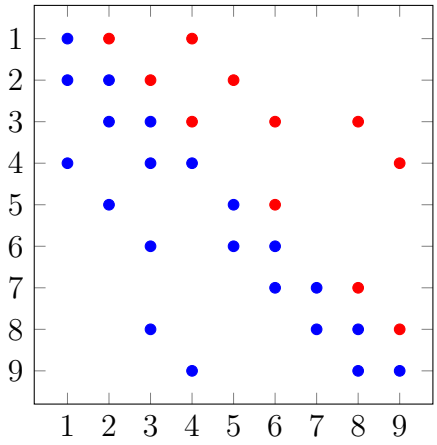



(b) Sparsity pattern. Downwind cells are shown using red markers.

Figure 2.7: Domain with natural numbering




(a) Cell numbering and fluid flow direction.




(b) Sparsity pattern. Downwind cells are shown using red markers.

Figure 2.8: Domain with topological numbering

The topological ordering can always be generated provided that the graph is *cycle free*. That is, after leaving a node along an edge that node will never

be revisited. This node structure results from a circulation free flux field $\boldsymbol{v}$. For incompressible flow, see Natvig et al. [2006], and cases with negligible or no gravity and capillary forces, see Kwok and Tchelepi [2007]; Lie et al. [2013], zero circulation is typical, at least with careful choice of numerical methods [Natvig et al., 2006; Lie et al., 2013]. When introducing significant gravity and capillary effects, see Kwok and Tchelepi [2007], or compressible flow, see Lie et al. [2013], circulation can occur in the velocity field. On the discrete domain circulation appear as cycles or *strongly connected components* in the graph. A strongly connected component is a group of nodes such that every node is reachable from every other node. These types of problems are not unusual in practice and thus must be dealt with, since these groups represents irreducible blocks in the system of equations, reintroducing the need for a full matrix solve. One possible solution is to redefine the strongly connected component as a single pseudo node in the topological ordering, and solving this region as a separate problem using e.g. a modified Newton method or Gauss-Seidel iterations. Another approach is to cycle through the cells in each strongly connected region until the solution in each cell in the component converges. This approach is described in more detail by Lie et al. [2013]. Cycles can be found in linear time $\mathcal{O}(n)$ by either Tarjan's algorithm or by using a double depth first search, where $n$ is the number of cells in the discretization [Natvig and Lie, 2008]. Algorithm 2 outlines the reordering procedure used to solve the transport equation, see Equation (2.1.12).

**Algorithm 2:** Pseudo code showing the reordering procedure for solving a transport problem, as in Equation (2.1.12)

**Data**: Saturations $S_V$ in all cells $V$, fluxes $F_s$ on all faces $s$.
**Result**: Updated saturations $S_V$

```
generate a topological ordering T_order of (pseudo) cells V_order based on
the face fluxes F_s;
foreach (pseudo) cell V_order in T_order do
    if V_order contains multiple cells V from T then
        solve the non-linear system for S_V using a vector procedure,
        e.g. Gauss-Seidel iterations;
    else
        solve the single cell problem for S_V using a scalar root finder,
        e.g. Regula Falsi;
    end
end
return updated cell saturations S_V
```

### 2.4.2 Root Finders

The reordering approach breaks the large system of equations into smaller subproblems. The single cell problems involves solving a univariate equation for the saturation in each cell $V$, namely Equation (2.4.7). That is, we want to find the *root* of the residual $R$, the number $S^{n+1}$ such that $R(S^{n+1}) = 0$. The literature contains a long list of numerical root finding algorithms for such problems, a few of which will be tested here for the single cell solver.

#### 2.4.2.1 The Bisection Method

The bisection method is a simple and robust *bracketing method*. That is, the method works over a *bracket* of the function $f$ on the real line, using the following definition

**Definition 1.** *A bracket* $[a, b]_f$ *for* $f : \mathbb{R} \to \mathbb{R}$ *is a closed subset of* $\mathbb{R}$ *such that* $[a, b]_f = \{x \in \mathbb{R} : a \leq x \leq b, a < b, f(a)f(b) < 0\}$.

Now we state the *intermediate value theorem*, which will help guarantee the existence of a root in a given bracket:

**Theorem 2.1.** ***Intermediate Value Theorem:*** *Let* $f\colon [a, b] \to \mathbb{R}$ *be a continuous function on the closed interval* $[a, b] = \{x \in \mathbb{R} : a \leq x \leq b\}$. *Then for every value* $y$, $f(a) < y < f(b)$, *there exists a number* $c \in (a, b)$ *such that* $f(c) = y$.

The proof of this theorem can be found in for example [Binmore, 1977]. Using a bracket $[a, b]_f$ in Theorem 2.1 quickly leads to the following corollary:

**Corollary 2.2.** *Let* $f\colon \mathbb{R} \to \mathbb{R}$ *be a continuous function with a* bracket $[a, b]$. *Then there exists at least one number* $r \in (a, b)$ *such that* $f(r) = 0$.

The corollary follows since the end points of the bracket are such that $f(a) < 0, f(b) > 0$ or, conversely, $f(a) > 0, f(b) < 0$, thereby bracketing the value $y = 0$ from the statement of Theorem 2.1.

Now, starting with a bracket $[a, c]_f$ the next iterate $b$ is found by halving the interval (hence the name of the method):

$$b = \frac{a + c}{2}.$$

A new starting bracket is chosen from the points $a, b, c$ according to Definition 1. The new bracket is named $[a, c]_f$ and the process is restarted. This iteration continues until a precise root is found, or the bracket becomes smaller than some tolerance $\epsilon$. By Corollary 2.2, and the fact that

the size of the bracket always decreases, the bisection method is guaranteed to converge. This is true in general for bracketing methods, although the convergence speed will vary. A more formal convergence theorem with proof can be found in Kincaid and Cheney [2002].

#### 2.4.2.2 Regula Falsi

Another bracketing method is the *regula falsi*, also called the *false position method*. Again we start with a bracket $[a, c]_f$. The method computes a secant line between the two end points of the bracket, as illustrated in Figure 2.9, using the following equation:

$$y(x) = \frac{f(c) - f(a)}{c - a}(x - a) + f(a).$$

In the following we will use the notation $f_x \coloneqq f(x)$ to simplify the equations. Since the secant line $y(x)$ is a continuous function the bracket $[a, b]_f$ can be used as a bracket for $y$ as well, giving $[a, b]_y$. This implies, by Corollary 2.2, that $\exists b \in (a, b) : y(b) = 0$. The root $b$ is found by

$$b = a - f_a \frac{c - a}{f_c - f_a}, \tag{2.4.10}$$

the *regula falsi step*. Now $b$ is used to update the bracket $[a, c]_f$ according to Definition 1. This iteration is continued until a root is found or the bracket size falls below some tolerance $\epsilon$. Note that efficient implementations of the regula falsi method requires only one function evaluation every iteration, since only one new point in the bracket is computed. A common problem pitfall this procedure is that for certain function shapes, for instance for convex or concave functions, only on side of the bracket will be updated, potentially leading to slow convergence. Dowell and Jarratt [1972] presents a modification to the regula falsi method, named the Pegasus method, where the end point retention problem is removed. This is done by first computing the regula falsi update as usual, and then reducing the function value of a retained end point by a factor $\gamma$, defined by

$$\gamma = \frac{f_o}{f_o + f_n}.$$

Here, $f_n = f(b)$ is the function value at the regula falsi update, and $f_o$ is the function value of the non-retained value on the opposite side of the bracket. The modified method has an order of convergence of around 1.64 [Dowell and Jarratt, 1972]. Algorithm 3 shows pseudo code for the OPM implementation of the Regula Falsi method with the Pegasus modification.

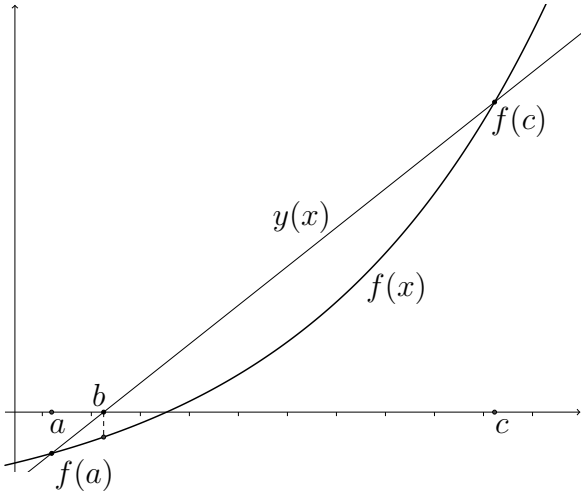



Figure 2.9: The secant line $y(x)$ is computed from the function $f(x)$ and a bracket $[a, c]_f$. $y(x) = 0$ is solved for the new update $b$ and used in the regula falsi method.

#### 2.4.2.3 Ridders' Method

Ridders' method is another bracketing scheme, introduced by Ridders [1979]. Again a bracket $[a, c]_f$ is chosen. A function $h(x; \alpha)$ is defined by

$$h(x; \alpha) = f(x)\mathrm{e}^{\alpha x}.$$

Computing the midpoint $b$ of the bracket we want to find an $\alpha \in \mathbb{R}$ such that

$$h(c; \alpha) - 2h(b; \alpha) + h(a; \alpha) = 0.$$

Inserting $h(x; \alpha)$ gives the following equation in $\alpha$:

$$\mathrm{e}^{\alpha c} f_c - 2\mathrm{e}^{\alpha b} f_b + \mathrm{e}^{\alpha a} f_a = 0.$$

Multiplying this equation by $\mathrm{e}^{-\alpha a}$ gives

$$\mathrm{e}^{\alpha(c-a)} f_c - 2\mathrm{e}^{\alpha(b-a)} f_b + \mathrm{e}^{\alpha(a-a)} f_a = \mathrm{e}^{\alpha 2\delta} f_c - 2\mathrm{e}^{\alpha\delta} f_b + f_a = 0,$$

since $c - a = 2(b - a)$ and $\delta := b - a$. Thus we get a second order equation in $\mathrm{e}^{\alpha\delta}$. The solution of this equation can be found by

$$\mathrm{e}^{\alpha\delta} = \frac{f_b \pm \sqrt{f_b^2 - f_c f_a}}{f_c}. \tag{2.4.11}$$

**Algorithm 3:** Pseudo code implementing the Regula Falsi root finder, see Section 2.4.2.2. The algorithm is modified with the *Pegasus method*, due to Dowell and Jarratt [1972].

**Data**: Initial guess $x_i$, a bracket $[x_0, x_1]_f$ for the function $f(x)$, a tolerance $\epsilon$, and the iteration limit $n_{\max}$
**Result**: An approximate root of $f(x)$

$f_0 \coloneqq f(x_0)$;
$f_1 \coloneqq f(x_1)$;
**if** $x_i$ *is a root* **then return** $x_0$;
**else** Form a new bracket $[x_0, x_1]_f$ from $x_0$,$x_1$,$x_i$;
**while** *not converged* **and** *iterations less than* $n_{max}$ **do**
    **if** $[x_0, x_1]$ *does not bracket the root* **then** handle the bracket error;
    $x_n \coloneqq \frac{x_1 f_0 - x_0 f_1}{f_0 - f_1}$;
    $f_n \coloneqq f(x_n)$;
    **if** $|f_n| < \epsilon$ **then return** $x_n$;
    **if** $f_n f_0 < 0$ **then**
        $x_0 \coloneqq x_1$;
        $f_0 \coloneqq f_1$;
    **else**
        $\gamma \coloneqq \frac{f_1}{f_1 + f_n}$;
        $f_0 \coloneqq \gamma f_0$;
    **end**
    $x_1 = x_n$;
    $f_1 = f_n$;
**end**
**return** the root approximation $\frac{x_0 + x_1}{2}$

We now need to know under which restrictions this equation has a solution. Since $e^x \geq 1,\ \forall x \in \mathbb{R}$, we need the right hand side positive in order for the equation to have a solution. Definition 1 implies that $f_b^2 - f_c f_a \geq f_b^2 \geq 0$ and thus the square root always yields a real number. Since the square root is a monotonic and increasing function, this implies that $\sqrt{f_b^2 - f_c f_a} \geq |f_b|$. Thus,

$$f_b + \sqrt{f_b^2 - f_c f_a} \geq 0,$$
$$f_b - \sqrt{f_b^2 - f_c f_a} \leq 0,$$

implying that the sign of the right hand side of Equation (2.4.11) is completely controlled by $\operatorname{sgn} f_c$. The solution $\mathrm{e}^{\alpha\delta}$ is then found by

$$\mathrm{e}^{\alpha\delta} = \frac{f_b + \operatorname{sgn} f_c \sqrt{f_b^2 - f_c f_a}}{f_c} \coloneqq \sigma_\alpha. \tag{2.4.12}$$

Now we find $\alpha$ by

$$\alpha = \frac{\ln \sigma_\alpha}{\delta}.$$

Ridder's method proceeds by applying the Regula Falsi to $h(x)$ on the bracket $[b, c]_h$, using the regula falsi step in Equation (2.4.10). This computes a new point $d$ by

$$d = b - h(b)\frac{c - b}{h(c) - h(b)}.$$

Inserting the definition for $h(x)$ gives

$$d = b - \mathrm{e}^{\alpha b} f_b \frac{c - b}{\mathrm{e}^{\alpha c} f_c - \mathrm{e}^{\alpha b} f_b} = b - \frac{\delta f_b}{\mathrm{e}^{\alpha c - b} f_c - f_b}.$$

By Equation (2.4.12) $\mathrm{e}^{\alpha\delta} f_c$ is given by

$$\mathrm{e}^{\alpha\delta} f_c = f_b + \operatorname{sgn} f_c \sqrt{f_b^2 - f_c f_a},$$

and because $\delta = c - b$, we get

$$d = b - \frac{\delta f_b}{f_b + \operatorname{sgn} f_c \sqrt{f_b^2 - f_c f_a} - f_b} = b - \frac{\delta f_b}{\operatorname{sgn} f_c \sqrt{f_b^2 - f_c f_a}}.$$

Now, by Definition 1, $\operatorname{sgn} f_c = -\operatorname{sgn} f_a$. Using $\operatorname{sgn} x \coloneqq \frac{x}{\sqrt{x^2}}$ we arrive at Ridders' method:

$$d = b + \frac{\delta f_b}{\frac{f_a}{f_a^2}\sqrt{f_b^2 - f_c f_a}} = b + \frac{\delta \frac{f_b}{f_a}}{\sqrt{\left(\frac{f_b}{f_a}\right)^2 - \frac{f_c}{f_a}}}. \tag{2.4.13}$$

The final step involves selecting the smallest new starting bracket $[a, c]_f$ from the points $\{a, b, c\}$ in combination with point $d$, keeping with Definition 1. Now the process is restarted, and continues until a root is found, or the size of the bracket falls below a tolerance $\epsilon$. Algorithm 4 shows the OPM implementation of Ridders' method.

**Algorithm 4:** Pseudo code implementing Ridders' method, see Section 2.4.2.3.

**Data**: Initial guess $x_i$, a highly unlikely answer $x_{\text{invalid}}$, a bracket $[x_0, x_1]_f$ for the function $f(x)$, a tolerance $\epsilon$, and the iteration limit $n_{\max}$

**Result**: An approximate root of $f(x)$

**if** $x_0$,$x_1$, *or* $x_i$ *is a root* **then return** the root;
**else** Form a new bracket $[x_0, x_1]_f$ from $x_0$,$x_1$,$x_i$;
$f_0 \coloneqq f(x_0)$;
$f_1 \coloneqq f(x_1)$;
$x_r \coloneqq x_{\text{invalid}}$;
**while** *not converged* **and** *iterations less than* $n_{max}$ **do**
  $x_m \coloneqq \frac{x_0+x_1}{2}$;
  $f_m \coloneqq f(x_m)$;
  $s \coloneqq \sqrt{f_m^2 - f_0 * f_1}$;
  **if** $s$ *is zero* **then return** $x_r$;
  **if** $f_0 \geq f_1$ **then**
    $x_r \coloneqq x_m + (x_m - x_0)\frac{f_m}{s}$;
  **else**
    $x_r \coloneqq x_m - (x_m - x_0)\frac{f_m}{s}$;
  **end**
  **if** $x_r$ *is converged under* $\epsilon$ **then return** $x_r$;
  Form a new bracket $[x_0, x_1]_f$ from $x_0$, $x_1$, $x_m$, and $x_r$;
**end**
Error: The iteration limit $n_{\max}$ is exceeded;

#### 2.4.2.4 Newton's Method

Unlike the bisection method, Regula Falsi, and Ridders' method, Newton's method is an *open* method, meaning that it does not restrict the search to a closed interval. This important feature allows the iterates to take on any value $x \in \mathbb{R}$, opening up the possibility for divergence of the solution. The upside is that the method has quadratic local convergence, in contrast to the super-linear convergence of the previously mentioned methods [Kincaid and Cheney, 2002]. Newton's method does not exhibit global convergence properties.

To derive Newton's method for solving $f(x) = 0,\ x \in \mathbb{R}, f\colon \mathbb{R} \to \mathbb{R}$, we start with a Taylor expansion of $f(x)$ around an initial guess $x_0$:

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!}f''(x_0) + \mathcal{O}((x - x_0)^3).$$

Evaluating this function at the root, say $x_r$, gives:

$$f(x_r) = f(x_0) + \Delta x_0 f'(x_0) + \frac{\Delta x_0^2}{2!} f''(x_0) + \mathcal{O}(\Delta x_0^3) = 0,$$

where $\Delta x_0 = x - x_0$. Dropping all higher order terms in $\Delta x_0$ leads to the following approximate equation:

$$0 = f(x_r) \approx f(x_0) + \Delta x_0 f'(x_0).$$

This relation implies that

$$\Delta x_0 \approx -\frac{f(x_0)}{f'(x_0)} \implies x_r \approx x_0 - \frac{f(x_0)}{f'(x_0)} := x_1.$$

for $f'(x_0) \neq 0$. Here $x_1$ is an updated guess for the root $x_r$. Iterating this equation leads to Newton's method for a univariate equation:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{2.4.14}$$

Newton's method can also be derived from a geometric argument. The tangent $y(x; x_n)$ to a curve $f(x)$ at a point $x_n$ is given as

$$y(x; x_n) = f'(x_n)(x - x_n) + f(x_n).$$

As long as $f'(x_n) \neq 0$ this tangent will cross the $x$-axis, i.e. we can find a root $x_{r_n}$ such that $y(x_{r_n}; x_n) = 0$. The solution to this equation is given as

$$x_{r_n} = x_n - \frac{f(x_n)}{f'(x_n)},$$

which when setting the new iterate $x_{n+1} = x_{r_n}$ is equivalent to Equation (2.4.14). Figure 2.10 illustrates the geometric interpretation and one step of Newton's method.

**The Secant Method** The *secant method* is a derivative free version of Newton's method obtained by approximating $f'(x_n)$ as

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

Inserted into Equation (2.4.14) this yields

$$x_{n+1} = \ell_s(x_n, x_{n-1}) = x_n - f(x_n)\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} =: \ell_s(x_n, x_{n-1}). \tag{2.4.15}$$

Note that two initial guesses are required to start the method and that we have defined a secant method function $\ell_s$ for later use. The secant method has a super-linear convergence rate whereas Newton's method converges quadratically [Kincaid and Cheney, 2002].

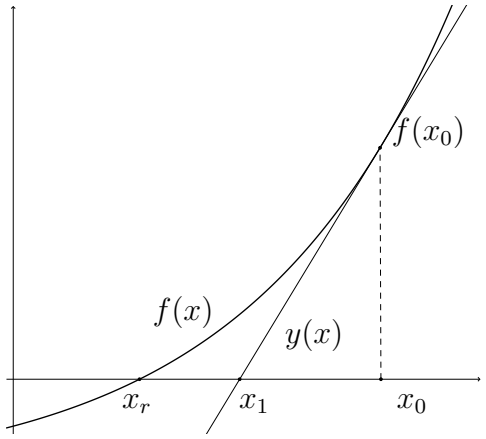



Figure 2.10: One step of Newton's method used to approximate the root $x_r$ of the function $f(x)$. The new iterate $x_1$ is found by computing the root of the tangent line $y(x)$ to $f(x)$ at the initial guess $x_0$.

#### 2.4.2.5 Brent's Method

Brent's method, due to Brent [1973], combines the bisection method, see Section 2.4.2.1, the secant method, see Section 2.4.2.4 and *inverse quadratic interpolation* and switches between the methods using a suitable heuristic. Brent's method is inspired by the older Dekker's method, see [Dekker, 1969]. We begin with a short presentation of inverse quadratic interpolation.

Linear interpolation is used in for example the secant method (Section 2.4.2.4) to approximate the function $f'(x)$ at two points $a, b$. Quadratic interpolation approximates $f(x)$ as a quadratic function, based on *three* points $a, b, c$, possibly leading to complex roots. Similarly, inverse quadratic interpolation approximates $f^{-1}(y)$ by three points $f(a), f(b), f(c)$, that is

$$f_*^{-1}(y) = \sum_{i=1}^{3} f^{-1}(f_i) \prod_{\substack{j=1 \\ j \neq i}}^{3} \frac{y - f_j}{f_i - f_j}. \tag{2.4.16}$$

Here $f_*^{-1}(y)$ denotes the interpolated function. Note that $f_i \coloneqq f(x_i)$, where $(x_1, x_2, x_3) \coloneqq (a, b, c, )$.The interpolated root is found by inserting $f(x_r) = 0$ into Equation (2.4.16). Since, by definition, $f^{-1}(f(x_r)) = x_r$ this gives an approximation for the root $x_r$ by the following equation (note the definition of an inverse quadratic interpolation function $\ell_{\text{iqi}}$):

$$x_r \approx \sum_{i=1}^{3} x_i \prod_{\substack{j=1 \\ j \neq i}}^{3} \frac{f_j}{f_i - f_j} =: \ell_{\text{iqi}}(x_1, x_2, x_3). \tag{2.4.17}$$

Brent's method starts with two points $a_k, b_k$ such that $f(a_k)f(b_k) < 0$ where $b_k$ is the current solution guess and $|f(a_k)| > |f(b_k)|$. At the initial step $k = 0$, we define $b_{-1} \coloneqq a_0$. A candidate update $s$ is found by

$$s = \begin{cases} \ell_{\text{iqi}}(a_k, b_k, b_{k-1}), & \text{if } f(a_k) \neq f(b_{k-1}) \text{ and } f(a_k) \neq f(b_{k-1}). \\ \ell_{\text{s}}(a_k, b_k), & \text{otherwise.} \end{cases} \tag{2.4.18}$$

The function $\ell_{\text{s}}$ implements the *secant method* defined in Equation (2.4.15). If $s \notin [\frac{3a_k+b_k}{4}, b_k]$ a bisection step $s = \frac{a_k+b_k}{2}$ is used in this iteration. On the other hand, if $s \in [\frac{3a_k+b_k}{4}, b_k]$ we define a number $\Delta$ such that

$$\Delta = \begin{cases} b_k - b_{k-1}, & \text{if bisection was used in the previous iteration} \\ b_{k-1} - b_{k-2}, & \text{if interpolation was used in the previous iteration} \end{cases} \tag{2.4.19}$$

Now, if $|s - b_k| \geq \frac{1}{2}\Delta$ or $|\Delta| \geq \delta$, for some tolerance $\delta > 0$, we fall back to the bisection method, such that $s = \frac{a_k+b_k}{2}$. If $f(a_k)f(s) < 0$, $b_{k+1} = s$ and $a_{k+1} = a_k$. If $f(a_k)f(s) \geq 0$, then $a_{k+1} = s$ and $b_{k+1} = b_k$. The final step is to ensure $|f(a_{k+1})| > |f(b_{k+1})|$ by swapping $a_{k+1}$ and $b_{k+1}$, if necessary. This iteration continues until the interval size is below a given tolerance $\epsilon$ or a root is found. Brent's method is shown in pseudo code in Algorithm 5 following the implementation in the OPM code.

**Algorithm 5:** Pseudo code implementing Brent's method, see Section 2.4.2.5.

**Data**: Initial guess $x_i$, function $f(x)$, bracket $[x_0, x_1]_f$, tolerance $\epsilon$, iteration limit $n_{\max}$
**Result**: An approximate root of $f(x)$

**if** $x_0$,$x_1$, *or* $x_i$ *is a root* **then return** the root;
**else** form a new bracket $[x_0, x_1]_f$ from $x_0$,$x_1$,$x_i$;
$f_0 \coloneqq f(x_0)$;
$f_1 \coloneqq f(x_1)$;
**while** *not converged* **and** *iterations less than* $n_{max}$ **do**
    use inverse quadratic interpolation to find an update $x_n$;
    **if** *the interpolation failed* **then** use the secant method to find $x_n$;
    **if** *iterate* $x_n$ *converged to slowly* **then** do a bisection step on $x_n$;
    form a new bracket from points $x_0, x_1, x_n$;
**end**
**if** *point* $x_1$ *or* $x_n$ *is a converged root* **then return** $x_1$ or $x_n$;
**else** the iteration limit $n_{\max}$ is exceeded;

#### 2.4.2.6 Trust Regions

As mentioned in Section 2.4.2.4, Newton's method can diverge for bad initial guesses. Despite this shortcoming we would like to exploit the nice convergence properties of the method. Several modifications have been proposed, among others the *Appleyard Heuristic* and the *Modified Appleyard Heuristic*, see e.g., [Younis, 2011]. These methods seek to scale the Newton update $\frac{f(x_n)}{f'(x_n)}$ to stop the method from diverging or using too many iterations to converge. Since we want to solve Equation (2.4.7) for the saturation $S_V^{n+1}$ we already have a well defined region of allowable values, namely $S_V^{n+1} \in [0, 1]$. This fact obviously follows from the physics of the problem and allows us to limit the Newton updates to this interval, keeping the iterates from diverging. This is one example of an imposed heuristic. Another approach, used in optimization, is to define a region within which the iterative technique is trusted to compute valid results, a so called *trust region*. Jenny et al. [2009] applies this to reservoir simulation residuals by identifying regions where the Newton method converges reliably. This can also be viewed as a globalization technique for Newton's method. In order to present the update scaling choices for the trust region methods we first introduce the *dimensionless flux function*.

**The Dimensionless Flux Function** Wang and Tchelepi [2013] defines a *dimensionless water flux function* by

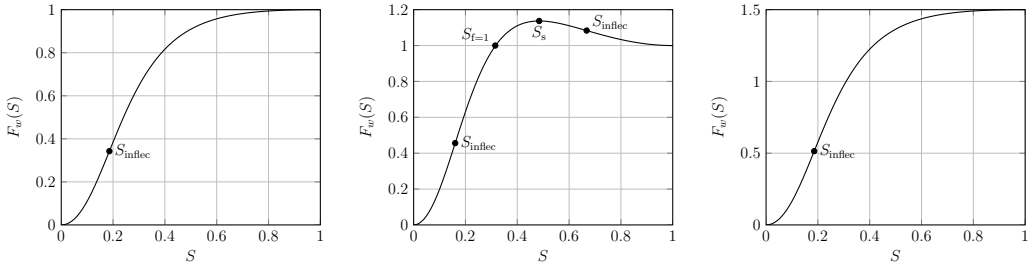
$$F_w = \frac{u_w}{u} = f_w + \frac{Kg\lambda_o f_w(\rho_w - \rho_o)\nabla h}{u_t} + \frac{f_w K \lambda_o \nabla p_c}{u_t}. \tag{2.4.20}$$

Here the permeability and gravity are assumed to be scalar. This equation states the water flux as a product of three terms; the viscosity terms, the buoyancy term, and the capillarity term, in that order. The idea is that this function is the main contribution to the non-linearity of the transport residual, and that this fact can be used to develop efficient update heuristics for Newton's method [Jenny et al., 2009]. Because the transport equation has been split into two equations, see Section 2.4, we operate with a slightly modified set of flux functions, the effects of which is most apparent in the buoyancy term. The flux functions for the three regimes are shown in Figure 2.11. It is apparent from the figure that the different regimes have qualitative differences, which are exploited in two different trust region schemes, presented in the two following sections. The discussion that follows warrants two definitions, as follows:

**Definition 2.** *An* inflection point $x_{inflec}$ *of a function* $f : \mathbb{R} \to \mathbb{R}$ *is a point where the sign of the derivative of* $f$ *changes sign or, equivalently, where the second derivative of* $f$ *is zero. That is,*

$$\frac{\partial^2 f}{\partial x^2}(x_{inflec}) = 0.$$

**Definition 3.** *A* sonic point $x_s$ *of a flux function* $f$ *is a point such that* $f(x_s) = 1$.




(a) Viscosity (b) Buoyancy (c) Capillarity

Figure 2.11: Water flux function $F_w$ dominated by viscosity, buoyancy and capillarity. $S_{\text{inflec}}$ denotes inflection points and $S_{\text{s}}$ sonic points.

**Jenny et al. Trust Region** Jenny et al. [2009] presents an update heuristic for the Newton method in the viscosity dominated case, with the flux function shown in Figure 2.11a. This function is s-shaped and its qualitative features are governed by the fractional flow function $f_w$. A closer inspection of this function shows that the domain $[0, 1]$ can be split into two subsets such that $F_w$ is concave on one and convex on the other. This is convenient, since Newton's method is known to converge regardless of starting point on convex or concave functions, see e.g. Morris [1983]. A smooth function making a transition from a concave to a convex region, or vice versa, must cross the so called *inflection point*, here denoted $S_{\text{inflec}}$. The inflection point is exactly the point where the second derivative of the function at hand changes sign. The *Jenny Trust Region* method, or the JTR, introduced in Jenny et al. [2009], tries to exploit the convergence guarantee of the Newton-Raphson method on convex or concave regions by first doing a regular Newton solve and then restricting the saturation such that no update leaps between the two regions. That is, if a saturation update $\Delta S$ and the initial saturation $S$ is such that $(S - S_{\text{inflec}})(S + \Delta S - S_{\text{inflec}}) < 0$, then the new saturation is set to $S_{\text{inflec}}$. The inequality $f_w''(S) f_w''(S + \Delta S) < 0$ also holds if the inflection point has

been crossed since the sign of the second derivative $f_w''(S)$ changes at $S_{\text{inflec}}$. This test is useful if the inflection point is not known *a priori*. In that case Jenny et al. [2009] proposes to instead cut back the saturation update by some heuristic, for example setting the new saturation to $S + \frac{\Delta S}{2}$. We call this the *approximate JTR scheme*. The *precise JTR scheme* is summarized in Algorithm 6.

**Algorithm 6:** Pseudo code implementing the JTR method, see Section 2.4.2.6.

**Data**: Initial guess $x_0$, function $f(x)$, tolerance $\epsilon$
**Result**: A approximate root $x_r$

$x_n = x_0$;
**while** $f(x_n) > \epsilon$ **and** $\Delta x_n > \epsilon$ **do**
  $\Delta x_n \coloneqq \frac{f(x_n)}{f'(x_n)}$;
  **if** $x_n + \Delta x_n$ *has crossed an inflection point* $x_{inflec}$ **then**
    $\Delta x_n \coloneqq x_{\text{inflec}} - x_n$;
  **end**
  $x_n \coloneqq x_n + \Delta x_n$;
**end**
$x_r \coloneqq x_n$;
**return** the root $x_r$

When solving the single cell residual in Equation (2.4.7) the inflection point can be computed a priori when the fractional flow function $f_w$ is known. This follows since the second derivative of the residual becomes

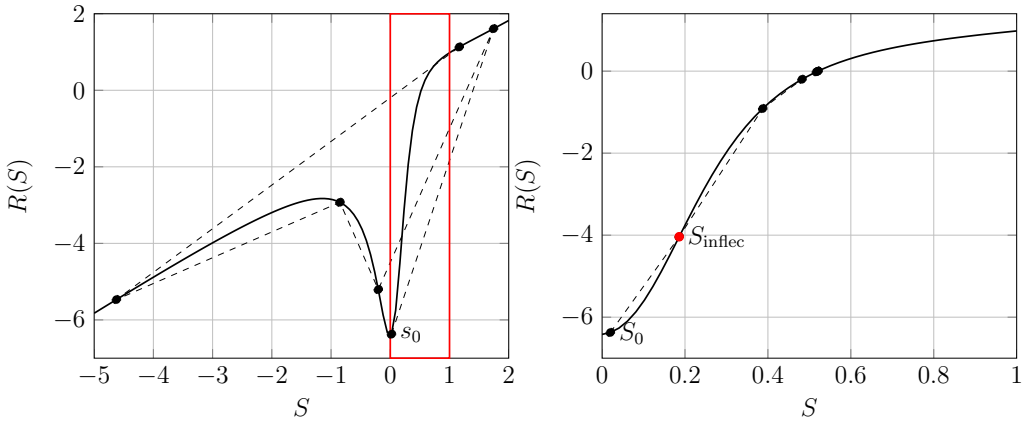
$$\begin{aligned}\partial^2_{S_V^{n+1}} R &= \partial_{S_V^{n+1}} \left[1 - \frac{\Delta t}{m(V)\phi_V} \partial_{S_V^{n+1}} f_w\right] \\ &= -\frac{\Delta t}{m(V)\phi_V} \partial^2_{S_V^{n+1}} f_w, \end{aligned} \tag{2.4.21}$$

which holds because all the terms besides the fractional flow function terms are constants (or first order) in $S_V^{n+1}$. Now, because the second derivative is zero at the inflection point $S_{\text{inflec}}$, Equation (2.4.21) implies that $S_{\text{inflec}}$ can be found by solving

$$\frac{\partial^2 f_w}{\partial {S_w}^2}(S_w) = 0, \tag{2.4.22}$$

where $f_w = f_w(S)$. Recalling that $f_w$ was defined in Equation (2.1.6) as the ratio of the water mobility $\lambda_w$ and the total mobility $\lambda$ we observe that in our case Equation (2.4.22) is uniquely defined by the fluid model chosen at

at the beginning of the simulation. Thus the inflection points are constant and equal for every cell residual throughout the simulation. With favorable definitions of the relative permeabilities $k_{rl}$ Equation (2.4.22) can be solved algebraically. When using quadratic relative permeabilities, as here, a cubic equation must be solved. The current implementation of the precise JTR method uses that approach. In the more general case a simple numerical root finder can be used to find $S_{\text{inflec}}$. It is important to note that these considerations are valid only for the viscosity dominated transport residual in Equation (2.4.7). The inflection points of the gravity residual, Equation (2.4.9), are dependent on the mobility in neighbor cells, calling for a new algebraic or numerical solution for $S_{\text{inflec}}$ before every call to the transport solver. This makes the precise trust region scheme a less attractive alternative for solving the gravity residual. We note that with quadratic permeabilities the gravity residual inflection points are given by a simple quadratic equation, leaving a possibility for an efficient algorithm.




(a) Regular Newton (b) JTR Newton

Figure 2.12: Newton iterations showing the JTR scheme converging unlike the regular Newton updates on the residual in Equation (2.4.7) with $\Delta t = 20000\,\text{s}$, $\phi_V = 0.5$, $m(V) = 1\,\text{m}^3$, $F_V^- = 0.16\,\text{m}^3/\text{s}$, and $Q_V^+ = -0.16\,\text{m}^3/\text{s}$, $\frac{\mu_w}{\mu_o} = 10$, and using initial guess $S_0 = 0.02$. $S_{\text{inflec}}$ is the residual inflection point. The red rectangle marks the range of valid saturations.

Figure 2.12a shows an example of a situation where the Newton method failed to converge and attained values outside the allowable range $[0, 1]$. Figure 2.12b shows the JTR method applied to the same problem. We observe that the method converges in a few iterations. Note especially the first step as compared to the first step in Figure 2.12a. The pure Newton method computes and accepts a solution outside domain. When using the JTR, the

algorithm detects that the solution update has crossed the inflection point and the solution is cut back to the inflection point $S_{\text{inflec}}$. The next update falls within the same concave region as the actual root, and the method converges in a few iterations.

**Wang-Tchelepi Trust Region** Jenny et al. [2009] only considered the viscosity dominated flux function, that is, all capillary and buoyant effects were removed from the flow equations. Building on the JTR method, Wang and Tchelepi [2013] present another trust region scheme in order to take the gravity and capillary forces into account. The strength of their new approach relative to the JTR method is based on the way the interface flux is computed. Jenny et al. [2009] uses a simple TPFA scheme with upwinding based on the phase velocity. The resulting residual is smooth and monotonic, with non-linearities caused by the fractional flow function $f_w$. In contrast, Wang and Tchelepi [2013] includes the buoyancy term, as in Equation (2.4.20), which can cause the water flux function to grow larger than one, i.e. $\exists S : F_w(S) > 1$. When this happens the flow of water is larger than the hydrocarbon pore volume and a back flow of the other phase, in our case oil, is implied by mass conservation. This is called *counter current flow*. A phase based upwind method is then employed to evaluate these fluxes on the cell interfaces in a conservative manner. In practice this means that when the water flux crosses the *unit flux point*, denoted $S_{F_w=1}$, the evaluation of the phase mobilities in the gravity flux term suddenly switches to the other cell, possibly causing a discontinuity in the resulting residual function. The residual is convex or concave on both sides of $S_{F_w=1}$, but the discontinuity can cause convergence problems for the ordinary Newton updates. To amend this, Wang and Tchelepi [2013] presents a scheme where the unit flux point $S_{F_w=1}$ is handled in the same manner as the inflection points was by the JTR method; any solution updates crossing the inflection points or unit flux points are chopped back. This restricts the Newton updates to the regions where they are trusted to converge. Algorithm 7 presents the pseudo code for the *Wang-Tchelepi trust region*, or WTR, method.

The preceding discussion indicates that the WTR method will not present any advantage over the simpler JTR when used to solve the residuals in Equations (2.4.7) and (2.4.9). This is caused by the gravity splitting scheme described in Section 2.4 and the way the phase based upwind method is applied. Specifically, the upwind method used by Wang and Tchelepi [2013] evaluates the dimensionless fractional flow function from Equation (2.4.20) as a function of the updated cell saturation $S_V^{n+1}$. Thus, when the unit flux point is crossed the upwind method makes the interface flux a function of

**Algorithm 7:** Pseudo code implementing the WTR method in a Newton iteration, see Section 10.

**Data**: $x_0$,$x_{\text{inflec}}$,$x_{f=1}$,$f(x)$,$\epsilon$,$n_{\max}$
**Result**: $x_n$

```
Initial guess;
x_n = x_0;
while not converged and iterations less than n_max do
    x_{n+1} := x_n − f(x_n)/f'(x_n);
    if x_{n+1} > 1 then x_{n+1} := 1;
    else if x_{n+1} < 0 then x_{n+1} := 0;
    if x_{n+1}, x_n has crossed the unit flux point point x_{f=1} then
        x_{n+1} := x_{f=1};
    end
    if x_{n+1}, x_n has crossed an inflection point x_{inflec} then
        x_{n+1} := x_inflec;
    end
    x_n := x_{n+1};
end
return the root approximation x_n
```

the oil mobility $\lambda_o$ in the neighbor cell, producing the discontinuity at $S_{F_w=1}$ in the residual as shown in Figure 2.13. In contrast, the numerical method presented in Section 2.4 evaluates the upwind method based on a flux field generated in the pressure solver. During the transport step this flux field is constant with respect to $S_V^{n+1}$, rendering the unit flux point check in the WTR method superfluous.

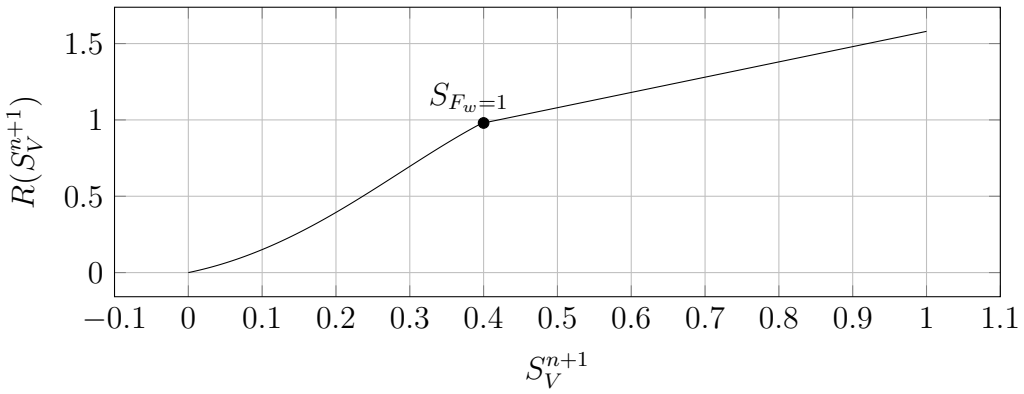



Figure 2.13: A transport equation residual with a discontinuity at $S_{F_w=1}$ caused by the upwind method as described in Wang and Tchelepi [2013].

# Chapter 3

# Numerical Results

Reservoir simulation packages are large and complex programs with IO functionality for well specifications, grid parameters, fluid definitions, etc., along with a host of utility functions. In order to test the numerical methods described in Chapter 2 we use the open source simulator supplied by the Open Porous Media initiative, or OPM , see [OPM, 2014]. Section 3.1 starts this chapter with a brief overview of the hardware and software used to test the efficiency of the new root finder implementations, as presented in Section 2.4.2. Section 3.2 presents a number of tests cases and the corresponding numerical results from running the cases with the different root finder implementations.

## 3.1 Test Procedure

In order to test the efficiency of the single cell solvers the OPM library was installed on a server with Intel® Xeon® X7542 CPUs running at 2.67 GHz with a 18432 kB cache size. The server has 252 GB of available ram. The homogeneous tests have been run using the C++ driver program included in Listing A.1. For the inhomogeneous tests the code in Listing A.2 was used. Finally, the homogeneous 3D tests were run using the code in Listing A.3. Further, all test cases were checked against the reference Regula Falsi solver, see Section 2.4.2.2, to ensure that the correct solution is found. The iteration count for each solver was reported along with the solution updates to determine the convergence speed versus iteration for each method. Finally, the total CPU time is reported to check the overall performance of each root finder. In the following we will test these methods for solving the single cell residual: Regula Falsi (RF), Section 2.4.2.2, Ridders (R), Section 2.4.2.3, Brent (B), Section 2.4.2.5, and the Approximate (TR*) and Precise (TR) Trust Region methods, Section 2.4.2.6. The method name abbreviations in

the parentheses will be used in the following.

### 3.1.1 The OPM Package

The OPM package provides a range of modules for grid handling, polymer injection, upscaling methods, and more. The `opm-core` module contains basic grid and well handling, and IO utilities, along with pressure and transport solvers for the porous media fluid flow problems described in Section 1.2.2. In fact, the OPM package implements the exact numerical methods described in Section 2.1 through the `Opm::IncompTpfa` pressure solver and the `Opm::TransportSolverTwophaseReorder` transport solver classes using the Regula Falsi Method for the single cell problems resulting from the reordering procedure, see Section 2.4.1. Listings A.1, A.2, and A.3 show the code for the driver programs implementing the sequential splitting scheme using the OPM library. The class `Opm::TransportSolverTwophaseReorder` implements the functionality for solving the residual equations in Equation (2.4.7) and (2.4.9), and is instantiated with the static properties of the simulation, such as the grid specification and fluid model. At each iteration of the sequential splitting method, as outlined in Algorithm 1, a new saturation field is computed by calling the method `solve(...)` on a `Opm::TransportSolverTwophaseReorder` object. The arguments to the solve method includes the time step $\Delta t$, and an instance of the fluid state class `TwophaseState` containing the saturation, flux, and other state information for every cell $V \in \mathcal{T}$. The `solve` method proceeds to compute the ordering of the flux graph based on the flux values obtained by solving the pressure equation. This ordering leads to a set of pseudo cells consisting of one or more regular cells, as described in Section 2.4.1. The new saturation field is finally obtained by iterating over all the pseudo cells and solving each subproblem by either the single cell root finders or the multi cell solver, depending on the number of grid cells in each pseudo cell. The single cell root finders are the main focus of this work. The transport equation residuals are implemented by two structs, Equation (2.4.7) in the struct `Residual` and Equation (2.4.9) in the struct `GravityResidual`. These structs are supplied with an ()-operator taking the update saturation $S_V^{n+1}$ as an argument and returning the residual value.

## 3.2 Test Cases

The following test cases are chosen to highlight different properties of the root finders. We start with the classical two-dimensional quarter five spot

problem as case A, Section 3.2.1. Next, two inhomogeneous, two-dimensional problems are tested in case B and case C, Section 3.2.2 and 3.2.3, respectively. Case D is a three-dimensional homogeneous block with gravity effects, see Section 3.2.4. All tests are run with tolerance $\epsilon = 1 \times 10^{-9}$. We have assumed that the numerical methods are stable, so convergence of the single cell problems indicates that the solutions are identical to the reference solver.

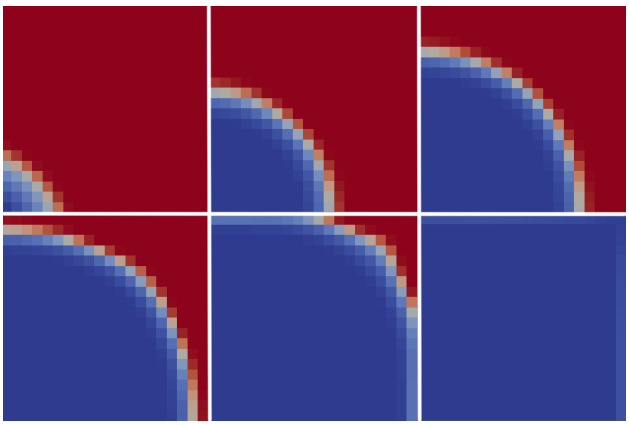


Figure 3.1: Water saturation profile when solving the Q5 problem on a 20x20 grid with $t_{\text{end}} = 300\,\text{d}$ and $\Delta t = 60\,\text{d}$. The region has a homogeneous permeability of 10 mD. Blue is water, red is oil.

### 3.2.1 Case A: Quarter Five Spot

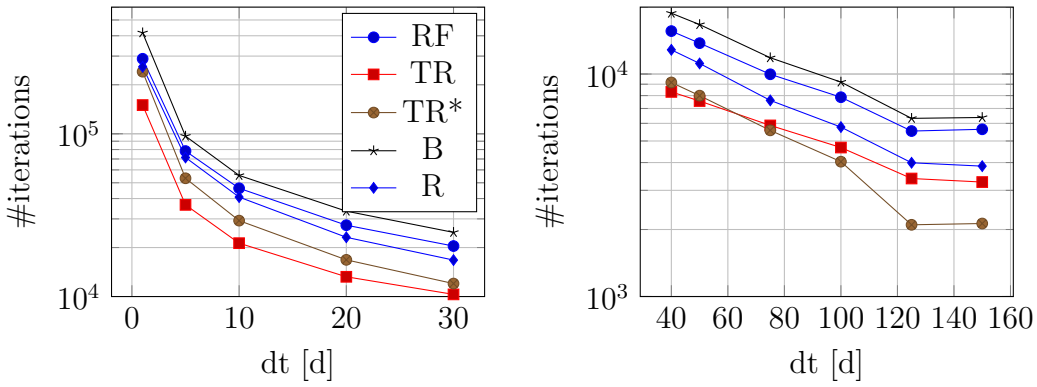
The *quarter five spot*, abbreviated Q5, is a quadratic 2D domain with a source in one corner and a sink in the opposite corner along the diagonal, while the rest of the boundary has no-flow conditions. The fluid density is set to $1000\,\text{kg/m}^3$ for water and $800\,\text{kg/m}^3$ for oil, porosity to 0.5 , and the formation has homogeneous permeability 10 mD. There are no gravity effects in this test. The simulation is run using different time steps $\Delta t$ until the final time $t_{\text{end}} = 300\,\text{d}$. The viscosity is varied through three cases; $\mu_w = 1\,\text{cP}$ and $\mu_o = 1\,\text{cP}$, $\mu_w = 1\,\text{cP}$ and $\mu_o = 10\,\text{cP}$, and $\mu_w = 10\,\text{cP}$ and $\mu_o = 1\,\text{cP}$. This gives viscosity ratios $M$ of 0.1, 1, and 10, respectively. Relative permeabilities are assumed to be quadratic functions of the saturation, i.e., $k_{rl} = S_l^2$. Two sub-cases are tested with different cell sizes. The first case is set up with $120\,\text{m} \times 120\,\text{m} \times 10\,\text{m}$ cells on a $20 \times 20$ grid with a source and a sink of magnitude $180\,\text{m}^3/\text{s}$. Next the problem is scaled down to cell dimensions $10\,\text{m} \times 10\,\text{m} \times 10\,\text{m}$ and a weaker source of magnitude $4\,\text{m}^3/\text{s}$. Figure 3.1 shows an example of an advancing saturation profile from a simulation of the homogeneous Q5 problem. Because of the uniform permeability and the

quadratic geometry the saturation profile is symmetric around the diagonal between the source and sink.

#### 3.2.1.1 Large Cells

The total iteration count spent when solving the Q5 problem with 120 m × 120 m cells is presented in Figure 3.2 for all root finders. We include the viscosity ratio because of the significant influence it has on the shape of $f_w$, as evidenced by Figure 2.1. The trend in the data is that Brent's method uses the highest number of iterations, while the Precise Trust Region scheme needs significantly fewer iterations than all other methods. The data also indicates that the average iteration count is lower and the differences between the methods are smaller for larger values of $M$. The corresponding total CPU running times are shown in Figure 3.3. The general impression from the plots is that the methods have similar performance, i.e., CPU time within the same order of magnitude. Further, none of the methods are consistently better than the others, neither with varying time steps and constant viscosity ratio $M$ nor the other way around. All in all the data are fairly inconclusive.

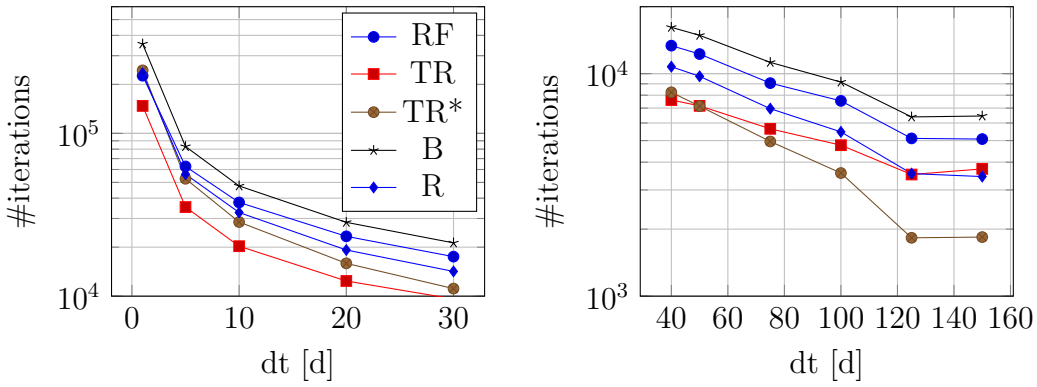
A comparison of the iteration count and total CPU time results shown in Figure 3.2 and 3.3, respectively, highlights the differences in computational complexity for the different root finders. That is, even with the quite significant variation in total amount of iterations, the CPU time is comparable for all root finders over the range of tested parameters. For instance, the Trust Region methods, which are essentially Newton methods, converge fast in terms of number of iterations due to the quadratic convergence of the Newton-Raphson scheme. But, since each iteration requires two function evaluations, one for the function itself and one for the derivative, the reduction in number of iterations is balanced by a relatively high computational complexity in each iteration. Similarly, the Brent method has "only" superlinear convergence and thus uses a very high number of iterations, but since it requires just one function evaluation per iteration, and has a fast implementation in general, the total CPU time spent is again normalized. Similar considerations can be used to explain some of the discrepancy between the the iteration count and CPU time results for the other root finders. Still, the iteration differences are so significant that we should expect more pronounced variations in execution time between the root finders. Worse, the iteration counts show a consistent dependency on the time step $\Delta t$, while the CPU times do not. We hypothesize that the reason for the inconclusive results is that the time spent by the pressure solver dominates the total run time, effectively hiding the transport solver results. The choice of pressure




(a) $M = 0.1$



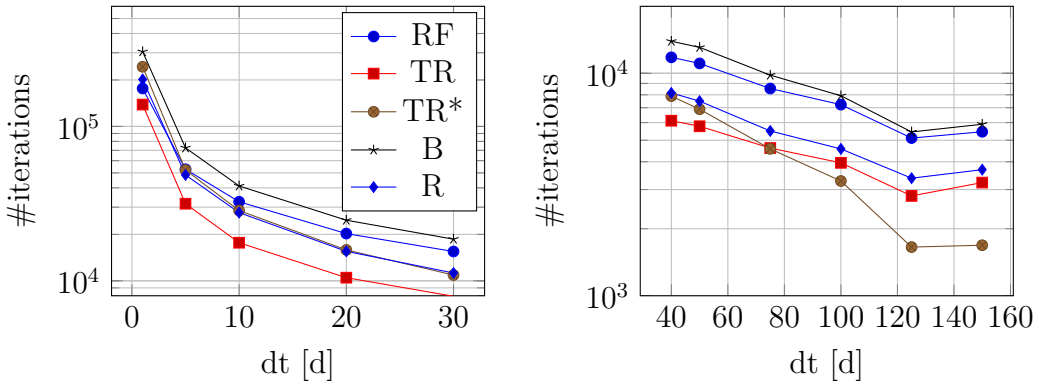

(b) $M = 0.1$




(c) $M = 1$




(d) $M = 1$

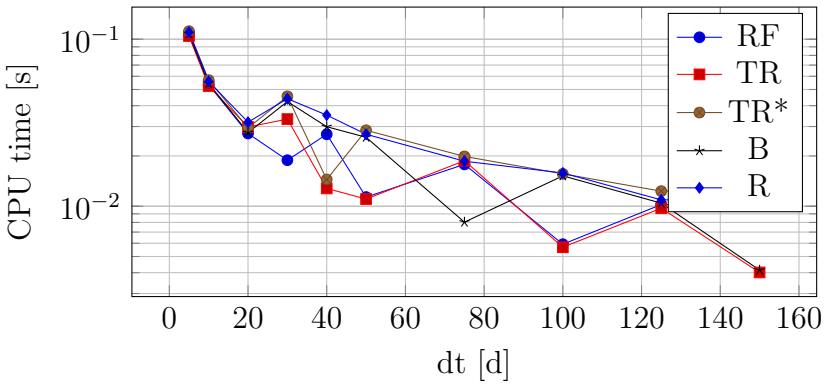



(e) $M = 10$

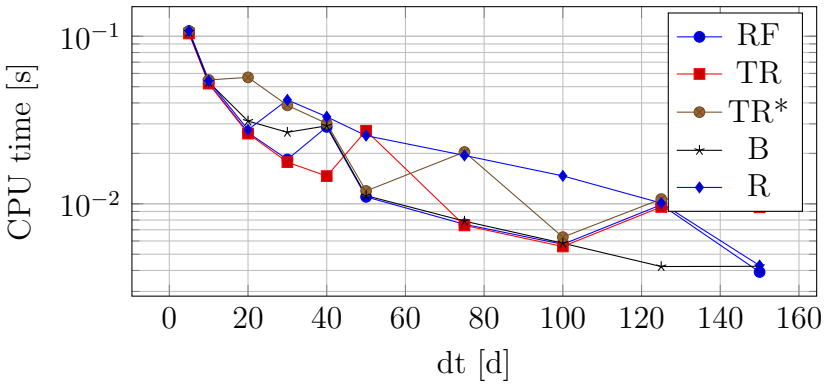



(f) $M = 10$

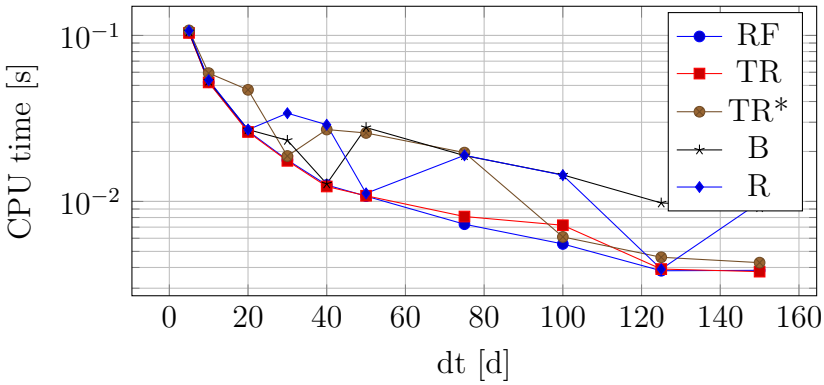
Figure 3.2: #iterations used to solve a modified Q5 problem, Section 3.2.1, for varying root finders, $\Delta t$, and $M$, with 120 m $\times$ 120 m cells.




(a) $M = 0.1$




(b) $M = 1$



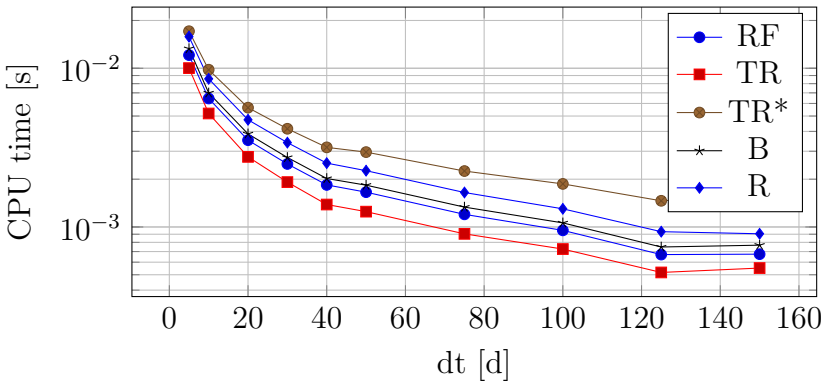

(c) $M = 10$

Figure 3.3: CPU time used to solve the Q5 problem, Section 3.2.1, for varying root finders, $\Delta t$, and $M$, with 120 m $\times$ 120 m cells.

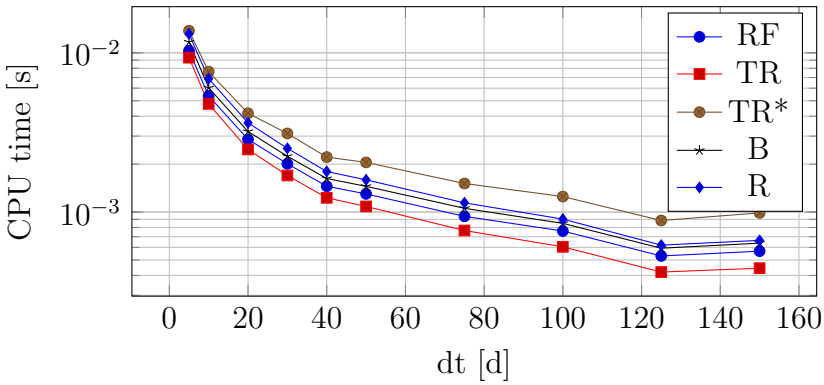
solver is independent of the transport solver, as discussed in Section 2.1, so a more efficient pressure solver could be chosen. In that case the transport solver might influence the total CPU time to a larger extent. Because of the independence of the two solvers it is still interesting to investigate the efficiency of the transport solver using different root finders. The plots in Figure 3.4 show timing results for just the transport step. It is clearly the case that these data correspond better with the iteration counts in Figure 3.2 than the total CPU times do. Now the Precise Trust Region method outperforms the other methods for all time steps, while the Regula Falsi method is the second best. The Brent method is slightly faster than the Ridders method, while the Approximate Trust Region method has the highest CPU time. As mentioned earlier, these results can be explained by the convergence properties of the individual methods, but these new results, without the influence of the pressure solver, follow the theory even closer. Note especially the low performance of the Approximate Trust Region method, and the good performance of the (modified) Regula Falsi approach. The Approximate Trust Region method uses two extra evaluations of the second derivative of the residual to identify trusted Newton-Raphson updates, bringing the total number of function evaluations per iteration up to four. Thus, each iteration of the method is slow compared to the other methods. The improved convergence rate of 1.64 for the modified Regula Falsi method, together with the single function evaluation per iteration, makes this approach better than the more complicated Brent and Ridders algorithms. Still, the local quadratic convergence of the Precise Trust Region scheme makes this the most efficient method for all tested time steps and viscosity ratios. We also note that the average CPU time used by the transport solver is lower for large $M$, for all root finders.

#### 3.2.1.2 Small Cells

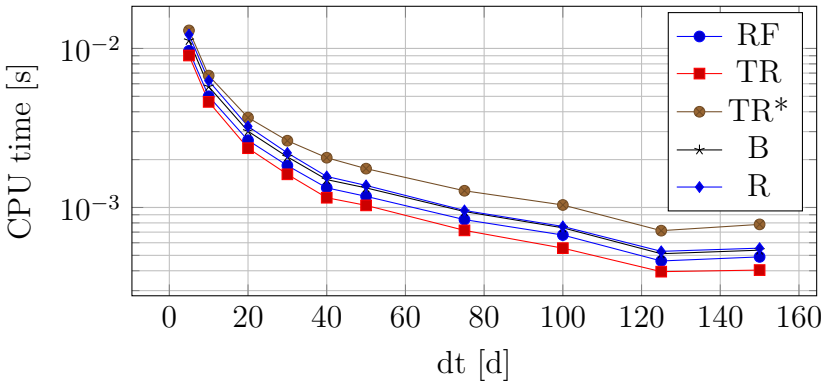
The total number of iterations spent by each root finder when solving the Q5 problem with cell dimensions $10\,\mathrm{m} \times 10\,\mathrm{m}$ is shown as a function of the time step $\Delta t$ and the viscosity ratio $M$ in Figure 3.5. The Brent method is the least efficient in terms of iteration count, while the Regula Falsi method is a close second. The Ridders algorithm needs more iterations than the two Trust Region schemes for the majority of time steps. For $\Delta t \lessapprox 60$ the Precise Trust Region scheme is the most efficient with the Approximate Trust Region scheme a close second, while $\Delta t \gtrapprox 60$ favours the approximate approach. Qualitatively these observations are consistent for all tested viscosity ratios, discounting edge effects. Note however that the actual difference between the number of iterations spent by the various root finders is influenced by $M$.




(a) $M = 0.1$




(b) $M = 1$



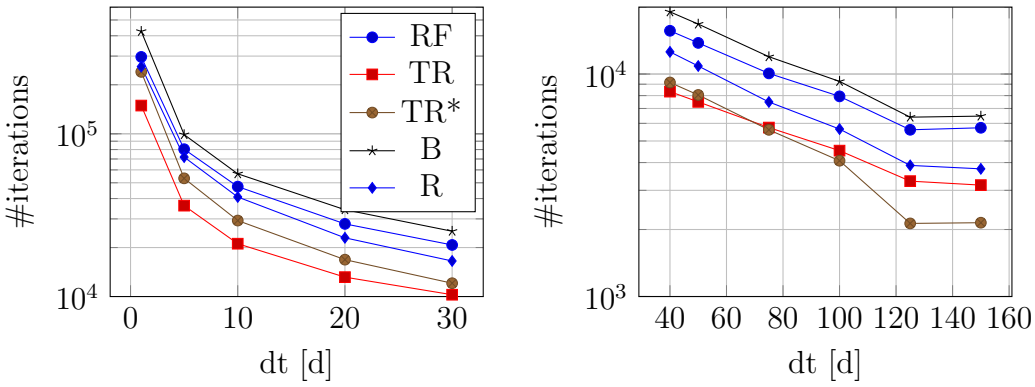

(c) $M = 10$

Figure 3.4: CPU time used by the transport solver when solving case A, Section 3.2.1, for varying root finders, $\Delta t$, and $M$, with 120 m $\times$ 120 m cells.

The ordering of the methods in terms of iteration count is about as expected based on theoretical convergence rates. The Trust Region schemes benefits from the quadratic local convergence of the underlying Newton-Raphson method, while the other procedures with only superlinear convergence need more iterations to obtain the desired precision. The advantage of the trust region schemes over the other methods seems quite significant. Note that the flatlining of all methods for $\Delta t = 150\,\mathrm{d}$ is easily explained by the way time step overshoots are handled, since the number of time steps for the simulation is set to $N := \lfloor \frac{t_{\mathrm{end}}}{\Delta t} \rfloor$. Thus, $\Delta t = 150\,\mathrm{d}$ and $120\,\mathrm{d}$ actually result in the same number of iterations of the sequential splitting method, while the time step factor in the residual is slightly different.

The total CPU time spent when solving the Q5 problem is shown in Figure 3.6 for all root finders and different viscosity ratios. Starting with Figure 3.6a for $M = 0.1$ we see that the CPU times for all methods are practically indistinguishable up to $\Delta t \approx 20\,\mathrm{d}$. At this point the trust region methods become slower than the rest. This holds until $\Delta t \approx 50\,\mathrm{d}$ where the Regula Falsi method also raises its CPU time. Finally, the Brent method slows down somewhere between $\Delta t = 50\,\mathrm{d}$ and $75\,\mathrm{d}$. The Ridders method has the best performance for most time steps. The timing results with $M = 1$ are shown in Figure 3.6b. The results in this case are less clear cut than with $M = 0.1$, with the ordering of the methods changing between time steps. Finally, Figure 3.6c shows the results with viscosity ratio $M = 10$. Here the Trust Region methods are the most efficient methods after the other algorithms slow down between $\Delta t = 20\,\mathrm{d}$ and $40\,\mathrm{d}$. The Approximate Trust Region methods slows down at around $\Delta t = 100\,\mathrm{d}$. Note the end effects for the last time step size, as mentioned earlier. A general observation is that all methods seems to have similar performance for the smallest time steps, up until around $\Delta t \approx 20\,\mathrm{d}$.

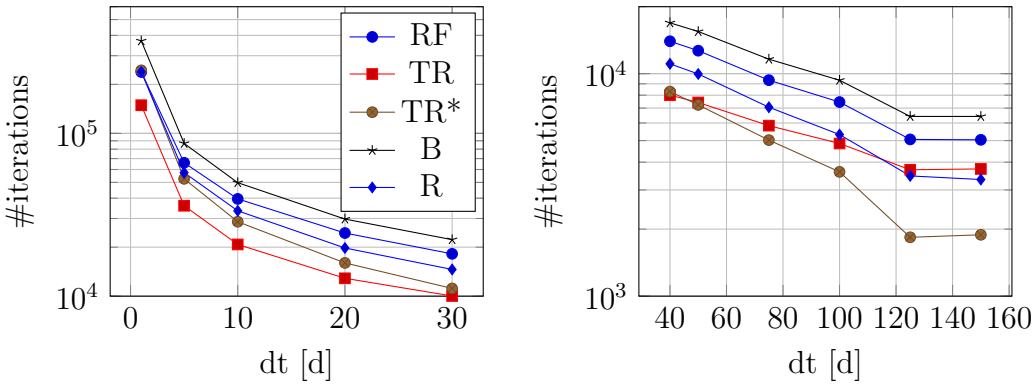
Again the reported total CPU times show little agreement with the iteration counts reported in Figure 3.5. The same reasoning applies here concerning the dominance of the pressure solver, so we report the transport solver CPU time in Figure 3.7. As for the case with large cells, the omission of the pressure time overhead makes the CPU time correspond very well with the iteration count. Again the Precise Trust Region method is the fastest method, while the Regula Falsi, the Ridders, and the Brent methods follow. The differences between the latter three methods are smaller, but the Regula Falsi algorithm consistently outperforms the other two root finders. Finally, the Approximate Trust Region method is the slowest approach. The trend with large $M$ giving a smaller average CPU time still holds.




(a) $M = 0.1$



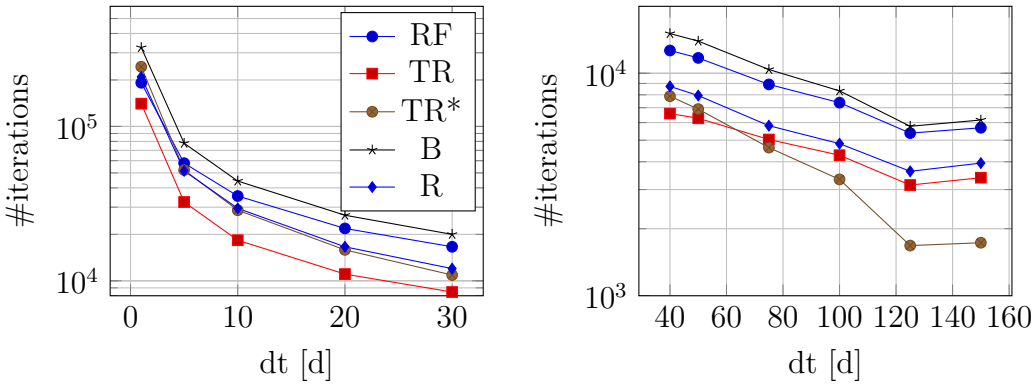

(b) $M = 0.1$




(c) $M = 1$




(d) $M = 1$

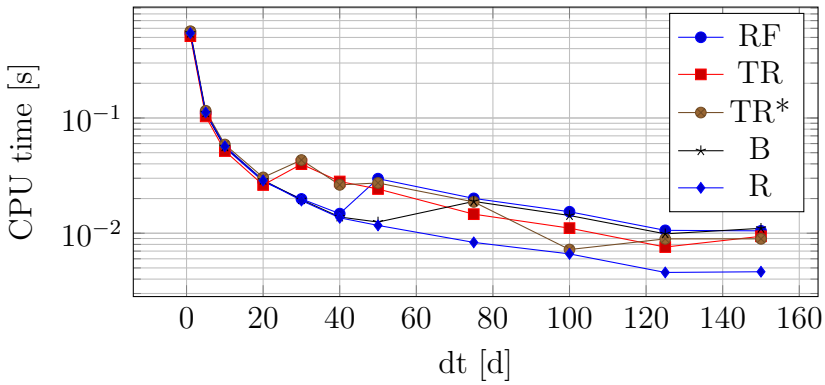



(e) $M = 10$

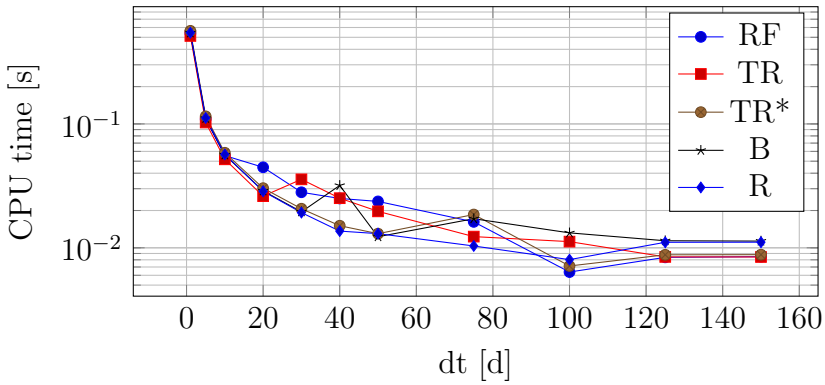



(f) $M = 10$

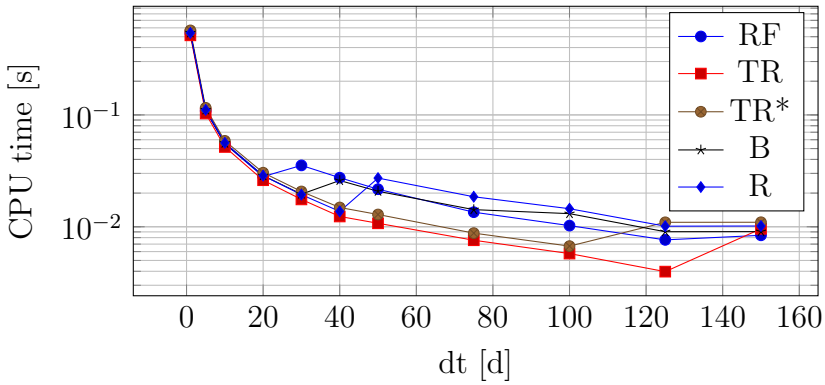
Figure 3.5: #iterations used to solve the Q5 problem, Section 3.2.1, for varying root finders, $\Delta t$, and $M$, with $10\,\mathrm{m} \times 10\,\mathrm{m}$ cells.




(a) $M = 0.1$

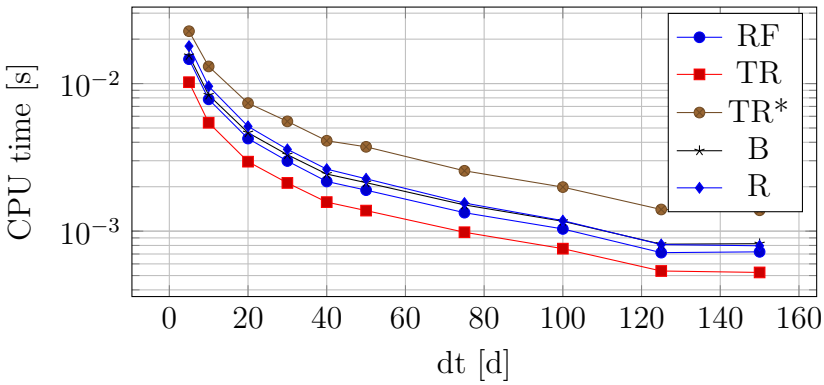



(b) $M = 1$

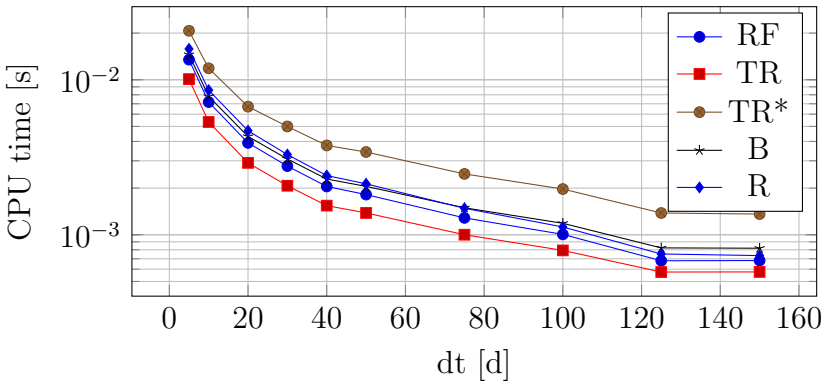



(c) $M = 10$

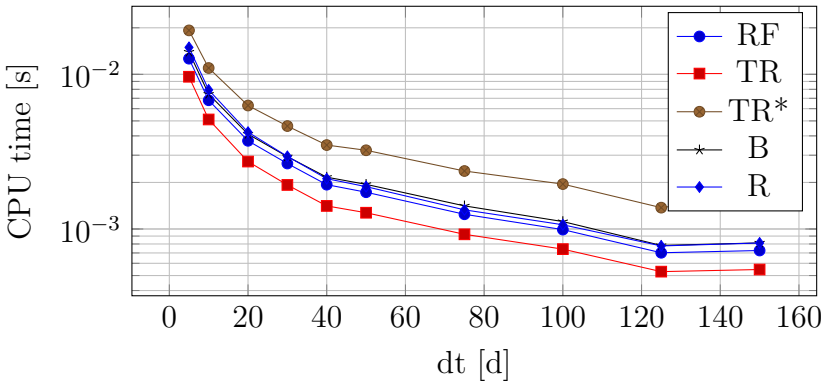
Figure 3.6: CPU time used to solve the Q5 problem, Section 3.2.1, for varying root finders, $\Delta t$, and $M$, with $10\,\mathrm{m} \times 10\,\mathrm{m}$ cells.




(a) $M = 0.1$




(b) $M = 1$

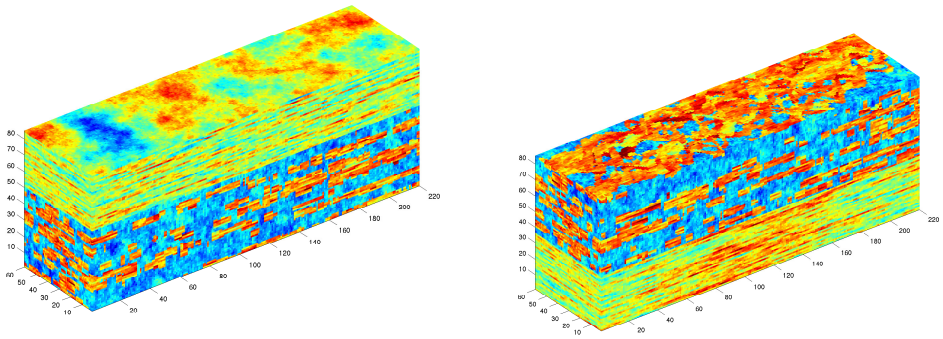



(c) $M = 10$

Figure 3.7: CPU time used by the transport solver, alternative run, when solving case A, Section 3.2.1, for varying root finders, $\Delta t$, and $M$, with $10\,\mathrm{m} \times 10\,\mathrm{m}$ cells.

### 3.2.2 Case B: Tarbert 2D

We now test a case with a more realistic inhomogeneous permeability distribution taken from the second SPE10 data set [SPE10, 2000]. The data set consists of scalar permeabilities in the $x$-, $y$- and $z$-directions on a three dimensional grid with $60 \times 220 \times 85$ cells, with the top 35 layers being part of the Tarbert formation and the bottom 50 the Upper Ness formation.The fine scale permeability grid cells are $20\,\text{ft} \times 10\,\text{ft} \times 2\,\text{ft}$ in size. Figure 3.8 shows the logarithm of the x-direction permeabilities for the entire domain, from which the first layer of the Tarbert formation is chosen for the numerical tests in this case. This layer is shown in Figure 3.9 along with a snapshot of the saturation front. Since a layer has $220 \times 60 = 13200$ cells, versus the 400 cells in case A, we expect the iteration count and CPU times to be significantly larger in the present case. We have chosen to use a $60 \times 220$ grid with a source and a sink of magnitude $50\,\text{m}^3/\text{s}$ in each opposite corner, without gravity effects. Otherwise the boundary has no flow conditions. The cells in the grid have dimensions $10\,\text{m} \times 10\,\text{m} \times 10\,\text{m}$, while the fluid density is set to $1000\,\text{kg}/\text{m}^3$ for water and $800\,\text{kg}/\text{m}^3$ for oil, and the porosity to $0.5$. The viscosity ratio $M$ is tested in the three cases $M = 0.1$, $M = 1$, and $M = 10$, as before. Finally, the relative permeabilities are still assumed to be quadratic functions of the saturation.

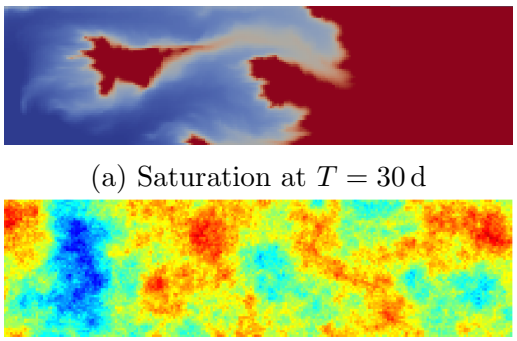


(a) Top view (Tarbert) (b) Bottom view (Upper Ness)

Figure 3.8: Inhomogeneous permeability data from the second SPE10 data set [SPE10, 2000]. The top 35 layers are part of the Tarbert formation. The lower 50 are part of the Upper Ness formation. The model dimensions are $1200\,\text{ft} \times 2200\,\text{ft} \times 170\,\text{ft}$ with $60 \times 220 \times 85$ cells.

Figure 3.10 shows the total iteration count used when solving case B. As expected, the iteration count is orders of magnitude larger than for case

A, with a range from around $9 \times 10^4$ up to around $1.5 \times 10^7$ iterations. Again the lowest iteration count is obtained by the Trust Region schemes, with an advantage to the Precise Trust Region method. The other methods also follow the pattern observed in case A, with the Brent method using the highest number of iterations, and the Regula Falsi and Ridders methods slightly fewer. We note that for $M = 1$ and $M = 10$, $\Delta t \lessapprox 30\,\mathrm{d}$ gives the Regula Falsi method a slight advantage, while the Ridders method is better for larger time steps. With $M = 0.1$ the Ridders method is more efficient than the Regula Falsi algorithm for all $\Delta t$.

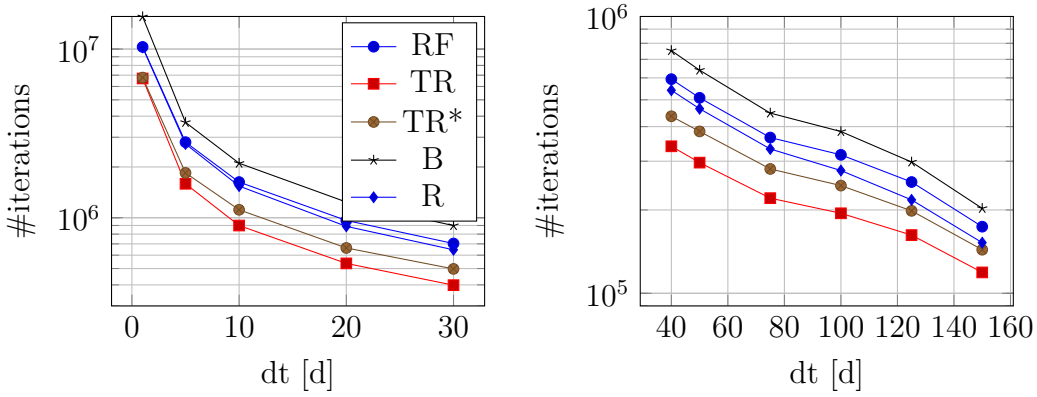


(a) Saturation at $T = 30\,\mathrm{d}$

(b) Permeability

Figure 3.9: Saturation profile (top) of water (blue) when running case B on the first layer of the Tarbert formation from the SPE10 permeability data set. The logarithm of the $x$-direction permeabilities are also shown (bottom).

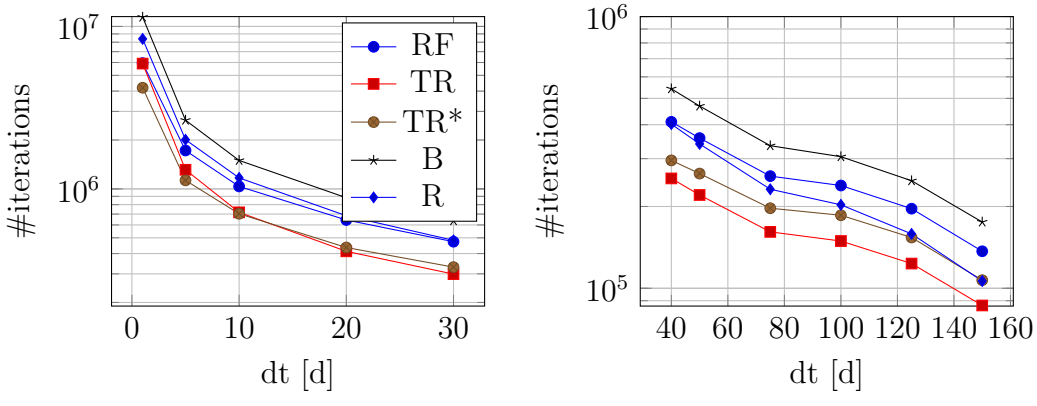
Now, using the same reasoning as before with regards to the pressure solver dominance, we show timing results for just the transport step in Figure 3.11. The order of magnitude of the run times varies between $10^{-2}$ and 1, much larger than for the smaller test case A, as predicted. The timing results also follow the iteration count results closely, with the Precise Trust Region method as the fastest root finder. We still see the effect of the high per-iteration computational complexity of the Approximate Trust Region scheme with this method as the slowest despite the good iteration count. Yet again the average run time is reduced for larger $M$.




(a) $M = 0.1$



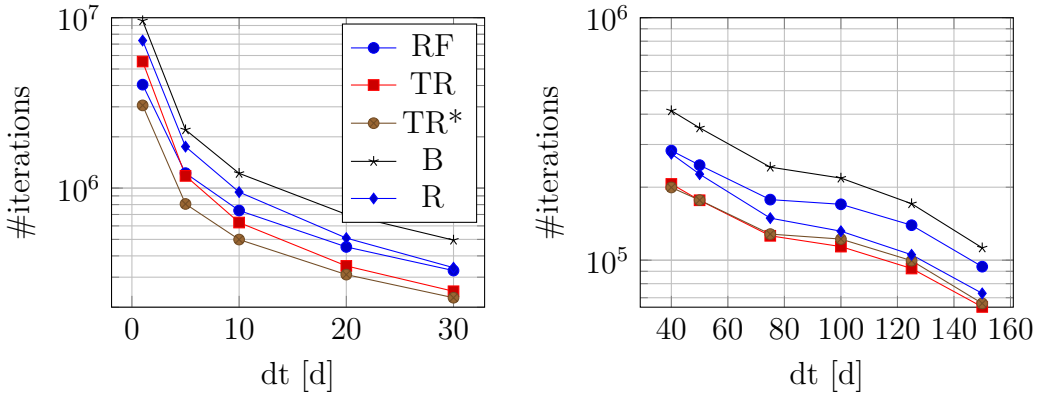

(b) $M = 0.1$




(c) $M = 1$




(d) $M = 1$

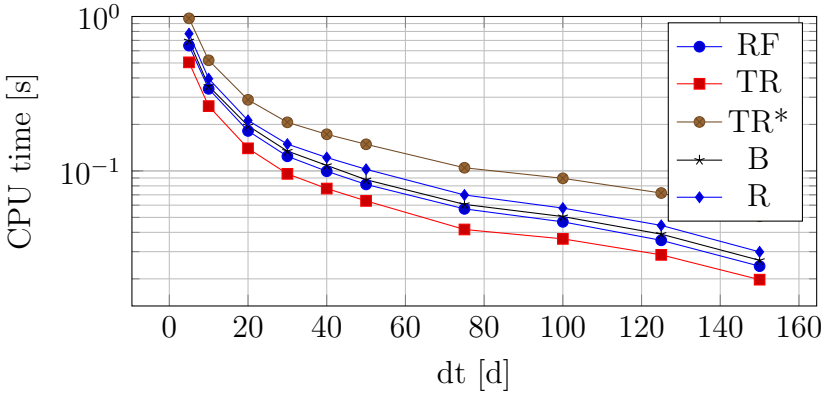



(e) $M = 10$

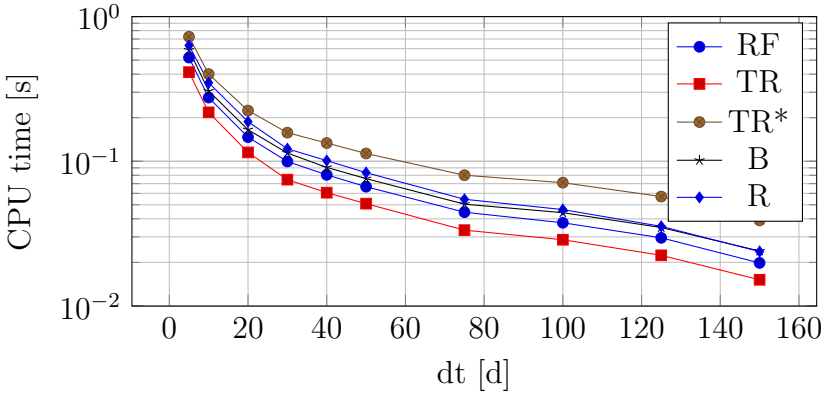



(f) $M = 10$

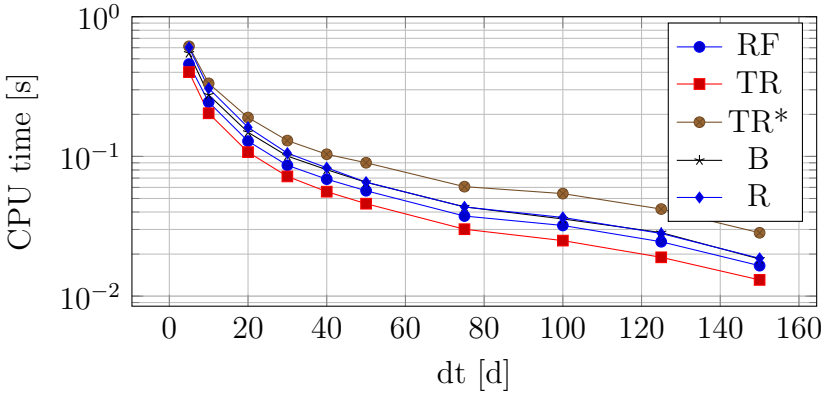
Figure 3.10: #iterations used to solve case B, Section 3.2.2, for varying root finders, time steps and viscosity ratios.




(a) $M = 0.1$




(b) $M = 1$

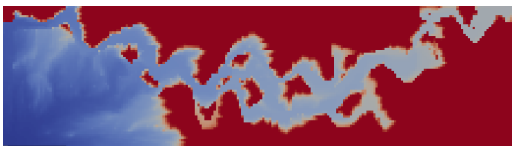



(c) $M = 10$

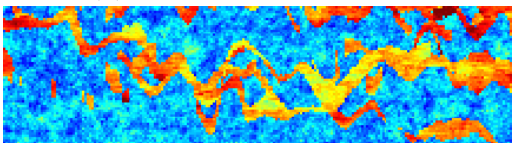
Figure 3.11: CPU time used by the transport solver when solving case B, Section 3.2.2, for varying root finders, time steps and viscosity ratios.

### 3.2.3 Case C: Upper Ness 2D

The problem setup for this case is exactly as in case B but with permeability data from the Upper Ness formation. Figure 3.8 indicates that the Upper Ness formation has more severe local permeability variations compared to the Tarbert formation. Upper Ness is a fluvial formation, with channels of high permeability rock, seen as red and yellow tones in Figures 3.8 and 3.12b. The latter figure shows layer 36 in the dataset which will be used with $60 \times 220$ grid cells in the layer, with cell dimensions $10\,\mathrm{m} \times 10\,\mathrm{m} \times 10\,\mathrm{m}$. Figure 3.12a shows a saturation profile at time 30 d with the source in the bottom left hand cell. The high permeability channels are clearly seen as regions with water saturated cells with blue coloring.



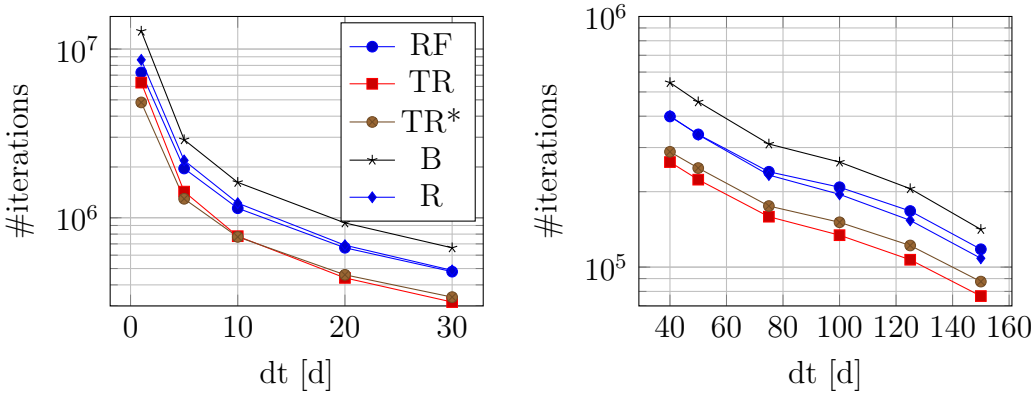
(a) Saturation at $T = 30\,\mathrm{d}$



(b) Permeability

Figure 3.12: Saturation profile (top) when running case C on the first layer of the Upperness formation, that is, layer 36 from the SPE10 permeability data set. The logarithm of the $x$-direction permeabilities are also shown (bottom).

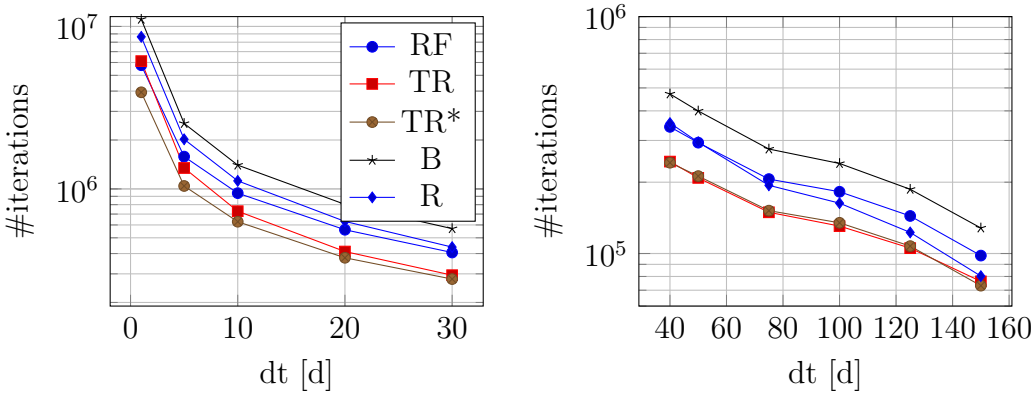
Figure 3.13 shows that the iteration count follows the same pattern as before, with the trust region methods outperforming the other methods for all tested time step sizes and viscosity ratios. For $\Delta t \gtrapprox 40\,\mathrm{d}$ the Ridders method uses fewer iterations than the Regula Falsi method for all viscosity ratios, with a more pronounced difference for larger $M$ values. As before we report the transport solver CPU times, see Figure 3.14. The Precise Trust Region method is again the best performer, with the Regula Falsi method as the second most efficient method, although the Ridders and Brent methods are not much slower. The Approxmiate Trust Region scheme is still slow. As for the other cases the simulation time is reduced for larger $M$ values.




(a) $M = 0.1$



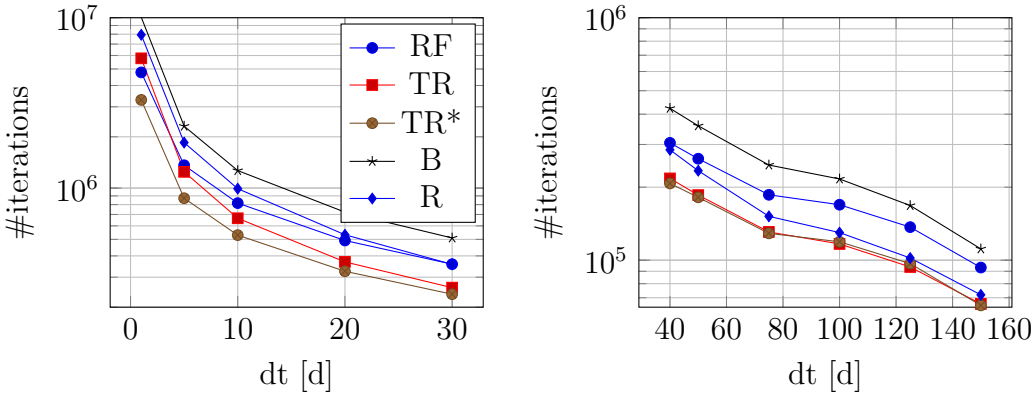

(b) $M = 0.1$




(c) $M = 1$




(d) $M = 1$

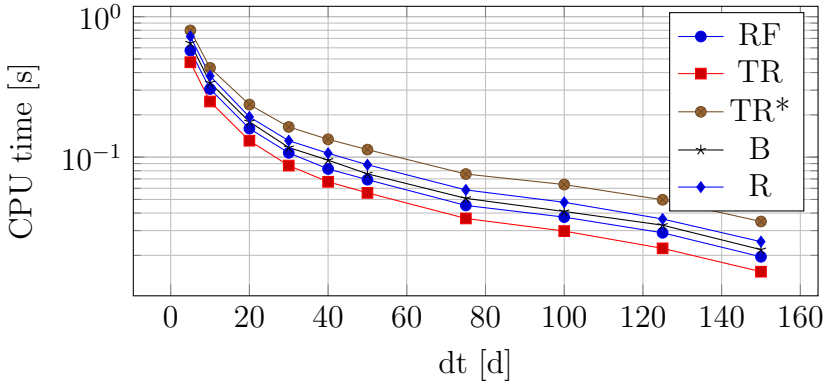



(e) $M = 10$

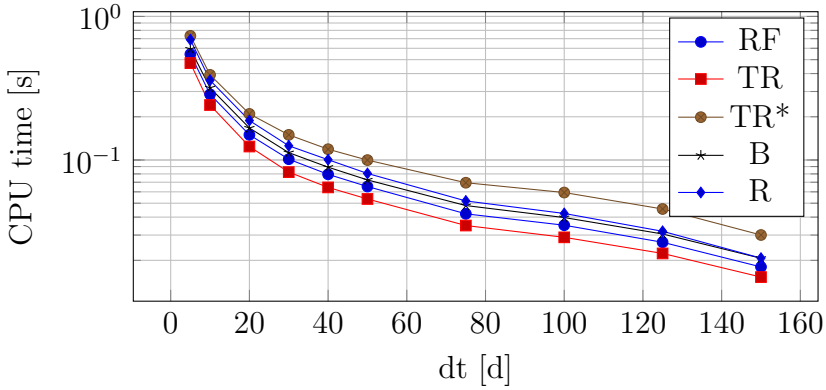



(f) $M = 10$

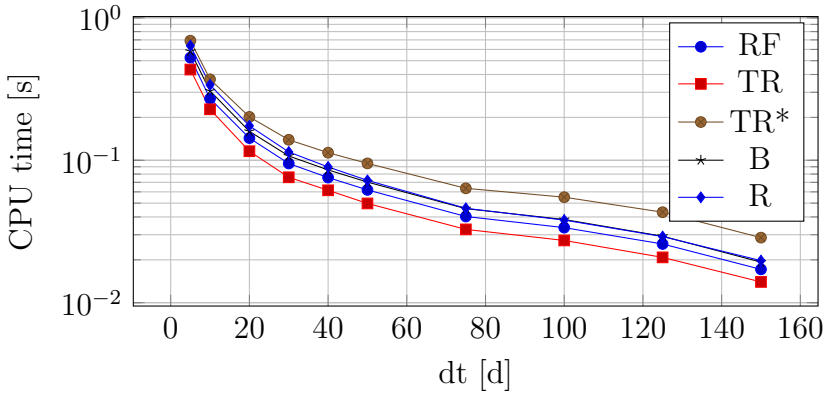
Figure 3.13: #iterations used to solve case C, Section 3.2.3, for varying root finders, time steps and viscosity ratios.




(a) $M = 0.1$




(b) $M = 1$




(c) $M = 10$

Figure 3.14: CPU time used by the transport solver when solving case C, Section 3.2.3, for varying root finders, time steps and viscosity ratios.

### 3.2.4 Case D: Three-Dimensional Domain

We now want to test the root finders on the gravity residual in Equation (2.4.9). The domain is made up of $10 \times 60 \times 10$ grid cells of dimension $10\,\mathrm{m} \times 10\,\mathrm{m} \times 10\,\mathrm{m}$, with permeability $10\,\mathrm{mD}$ for the whole domain. A source of magnitude $5\,\mathrm{m}^3/\mathrm{s}$ is placed in the first grid cell, while a sink of equal magnitude is placed in the last. Otherwise the boundaries have no-flow conditions. Additionally, the top half of the domain is initialized with fully water saturated cells, while the bottom half is saturated by oil. Since the oil is lighter than the water, with a density of $\rho_o = 400\,\mathrm{kg}/\mathrm{m}^3$ versus the water density at $\rho_w = 1200\,\mathrm{kg}/\mathrm{m}^3$, the oil will rise towards the top of the domain. The simulation is run with $t_{\text{end}} = 400\,\mathrm{d}$, and different time steps and viscosity ratios. Figure 3.15 shows the saturation after 200 d.

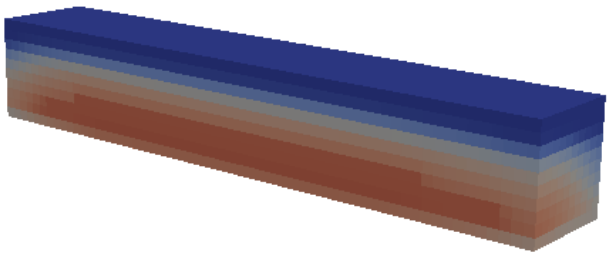
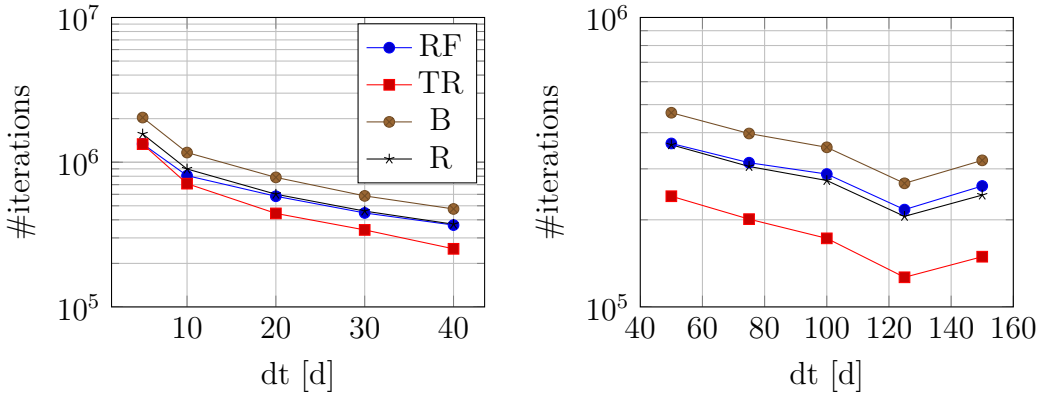


Figure 3.15: Saturation profile after 200 d for case D. Blue is water, red oil.

Figure 3.16 shows the combined iteration counts for the regular transport solver and the gravity column solver. Here the Precise Trust Region method performs worse than before for small time steps, relative to the other methods. This effect is especially significant for small time steps with $M = 1$, as shown in Figure 3.16c. The other results are qualitatively as before, with the Precise Trust Region method using fewer iterations than the Regula Falsi method, the Ridders method, and the Brent method, in that order. We note that the Approximate Trust Region method failed for all time steps.

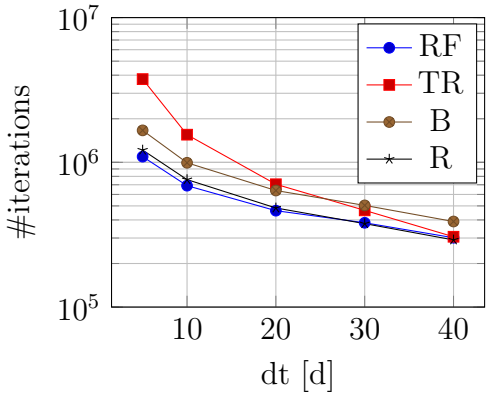
The CPU time without the pressure solver time is reported in Figure 3.17, i.e., the time spent by the transport solver including the gravity column solver. Here both the viscosity dominated residual in Equation (2.4.7) and the gravity dominated residual in Equation (2.4.9) are solved. The plots show the Precise Trust Region method as the fastest solver overall, followed by the Regula Falsi method, the Ridders method, and the Brent method. The increase in iterations used by the Precise Trust Region approach for small $\Delta t$ and $M = 1$ shows up as higher running times in Figure 3.17b.



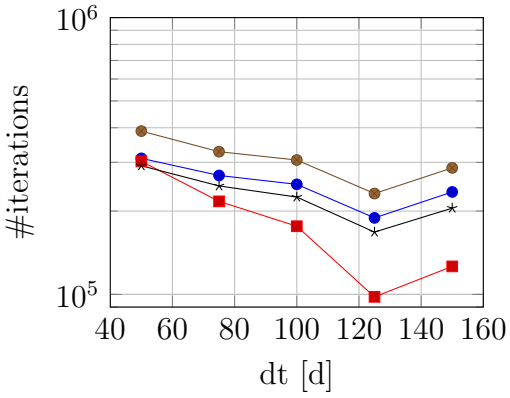

(a) $M = 0.1$



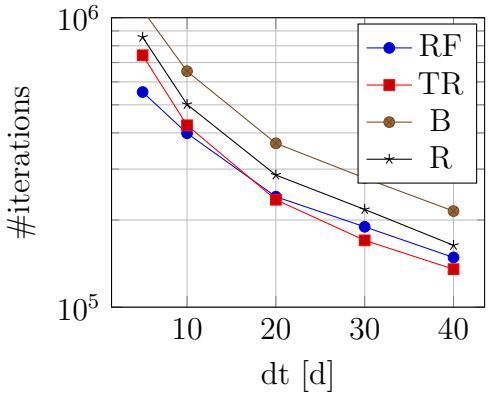

(b) $M = 0.1$



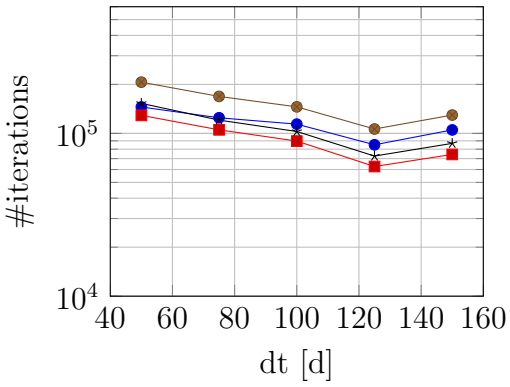

(c) $M = 1$




(d) $M = 1$

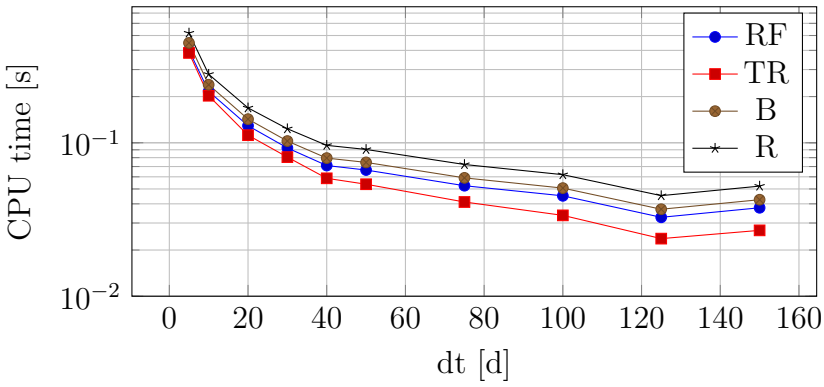



(e) $M = 10$

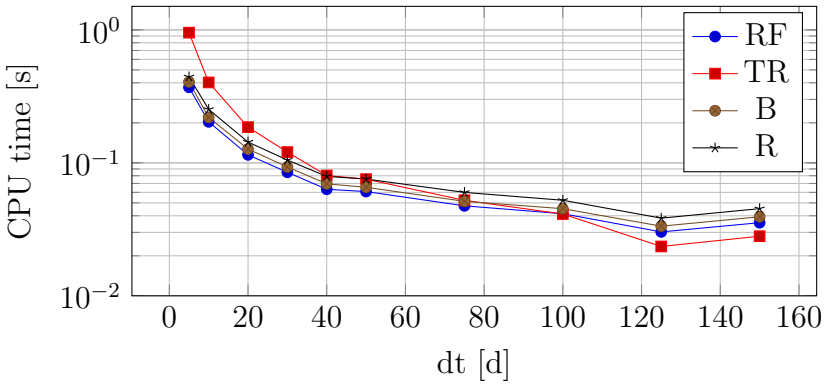



(f) $M = 10$

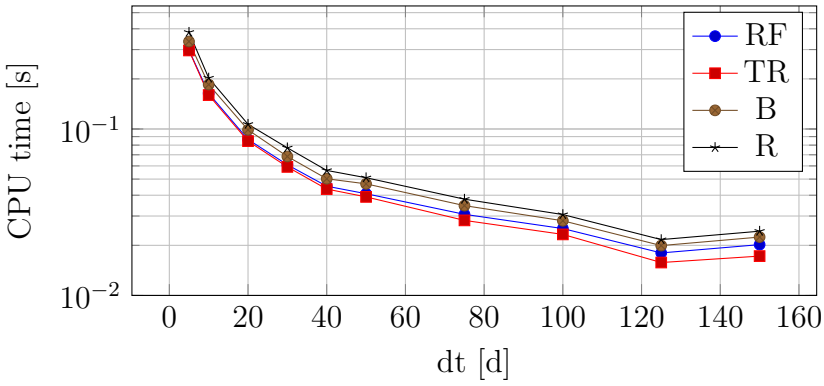
Figure 3.16: #iterations used to solve case D, Section 3.2.4, for varying root finders, time steps and viscosity ratios.




(a) $M = 0.1$




(b) $M = 1$




(c) $M = 10$

Figure 3.17: CPU time used by the transport solver when solving case D, Section 3.2.4, for varying root finders, time steps and viscosity ratios.

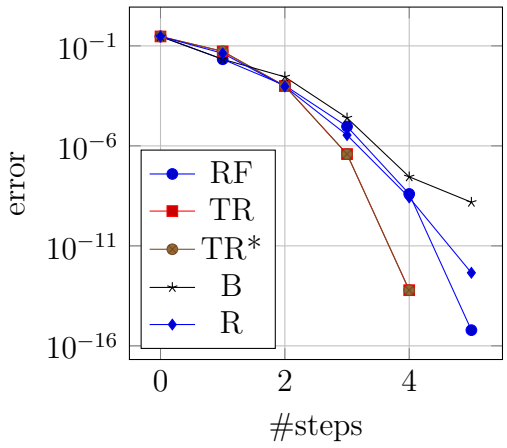
## 3.3 Convergence Tests

We now try to test the efficiency of the different root finders in a more controlled environment. One measure of the efficiency of numerical equation solvers is the *convergence rate*. This number measures the error reduction per iteration of the method, that is, a high convergence rate gives a larger reduction in the residual error per iteration, hence requiring fewer iterations for convergence. We note that the viscosity dominated single cell residual in Equation (2.4.7) is determined by five parameters; the initial saturation in the cell, $S_V^n$, the time step to pore volume ratio $\tau = \frac{\Delta t}{m(V)\phi_V}$, the flux out of the cell, say $q_o$, the flux into the cell, say $q_i$, and finally the viscosity ratio $M$ as defined in Equation (2.1.8). Thus, $R = R(S; S_0, \tau, M, q_o, q_o)$, where $S \coloneqq S^{n+1}$ and $S_0 \coloneqq S^n$. Varying these parameters through a range of typical values and running the root finders from Section 2.4.2 on the resulting single cell problems can help in understanding the solver properties. Note that the cell saturation from the previous time step is used as initial guess for the root finders, as is the case in the reference Regula Falsi solver in the OPM package. Figures 3.18 through 3.20 shows a number of different convergence and residual plots where the solvers are run until a tolerance of $1 \times 10^{-9}$ is obtained. We keep $M = 1$ and $\tau = 6\,\mathrm{s/m^3}$ constant.

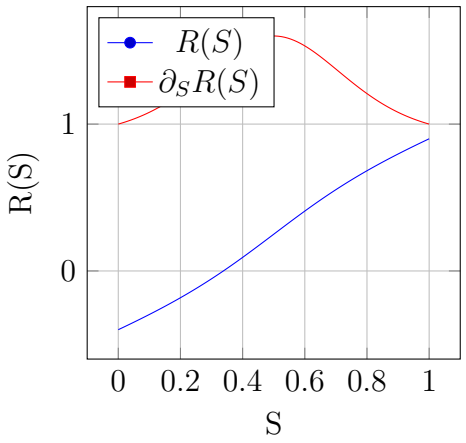
Figure 3.18 is obtained with incoming flux $q_i = -0.05\,\mathrm{m^3/s}$ and outgoing flux $q_o = 0.05\,\mathrm{m^3/s}$. This gives fairly linear residuals, and the initial guess $S_0$ is close to the root for $S_0 = 0.5$ and 0.9. The number of iterations for all tested root finders decreases when $S_0$ is increased. We also note that the Newton-like methods converge faster than the other methods. This plot indicates that for small flux values the initial guess strongly influences the residual bringing the root close to $S_0$, with a stronger effect for larger $S_0$.

Setting the incoming flux to $-0.35\,\mathrm{m^3/s}$ and the outgoing flux to $0.35\,\mathrm{m^3/s}$ we obtain Figure 3.19. The residual plots show a stronger non-linear influence, a fact reflected in the overall increase in the amount of spent iterations. $S_0$ still influences the residual, but to a lesser extent than with the lower flux values. Again we observe that large initial guesses generally leads to a lower iteration count. The Newton-like method are, together with the Ridders method, the best performers.

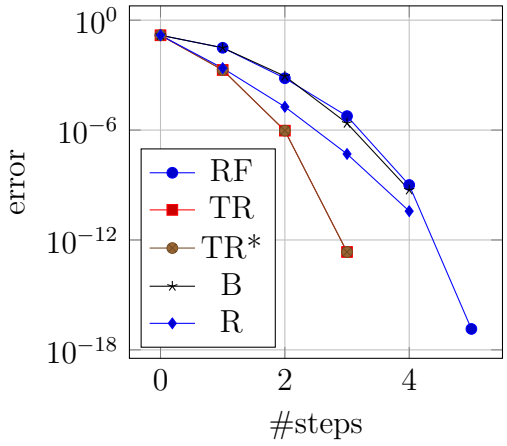
Moving on to even larger flux values, we set the incoming flux $q_i$ to $-0.5\,\mathrm{m^3/s}$ and the outgoing flux $q_o$ to $0.8\,\mathrm{m^3/s}$. Figure 3.20 shows the resulting plots. The trend from the previous plots continues in that the initial guess has less influence on the root position. The performance of the root finders also seems to be relatively unaffected by the initial guess, except the Newton-like methods. They show a significant performance boost for $S_0 = 0.5$, where the root is very close to the initial guess.



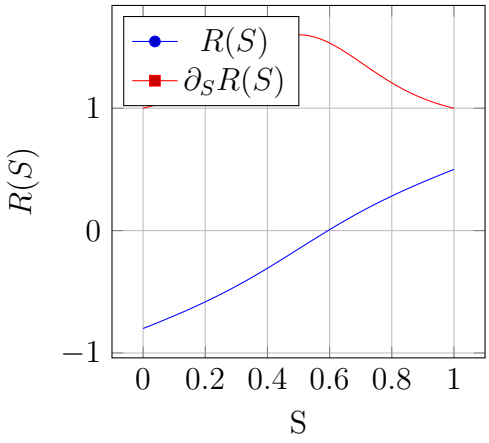

(a) $S_0 = 0.1$



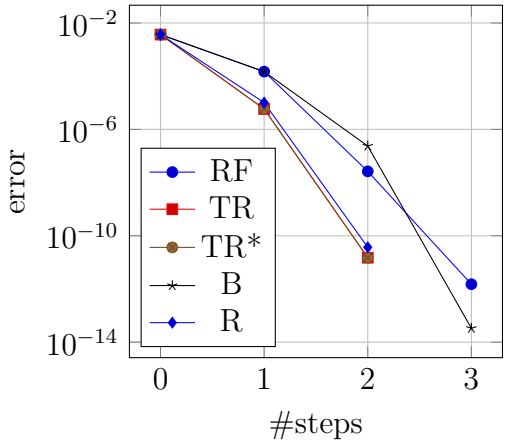

(b) $S_0 = 0.1$



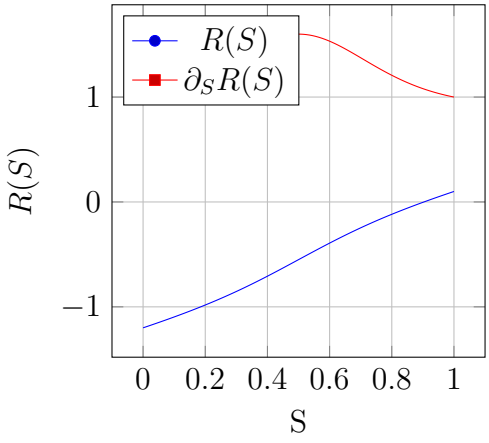

(c) $S_0 = 0.5$




(d) $S_0 = 0.5$

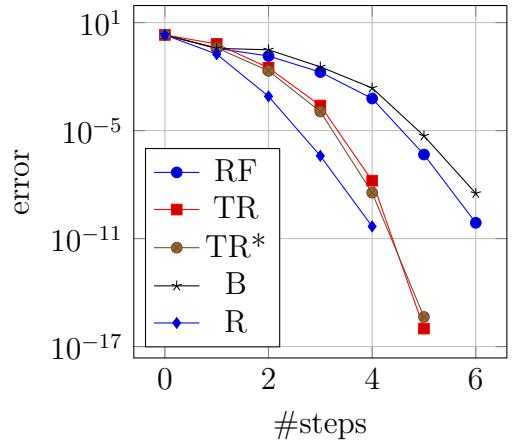



(e) $S_0 = 0.9$

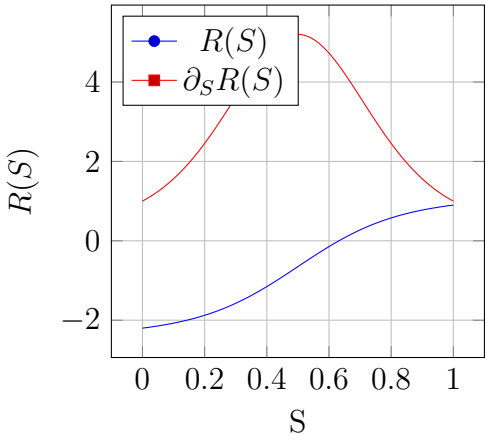



(f) $S_0 = 0.9$

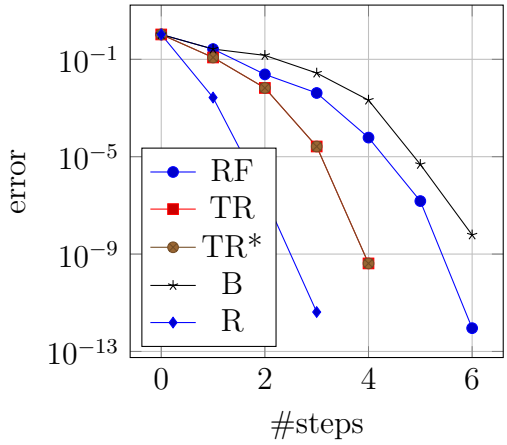
Figure 3.18: Convergence history and residuals with parameters $M = 1$, $\tau = 6\,{}^{\mathrm{s}}/_{\mathrm{m}^3}$, $q_i = -0.05\,{}^{\mathrm{m}^3}/_{\mathrm{s}}$, $q_o = 0.05\,{}^{\mathrm{m}^3}/_{\mathrm{s}}$



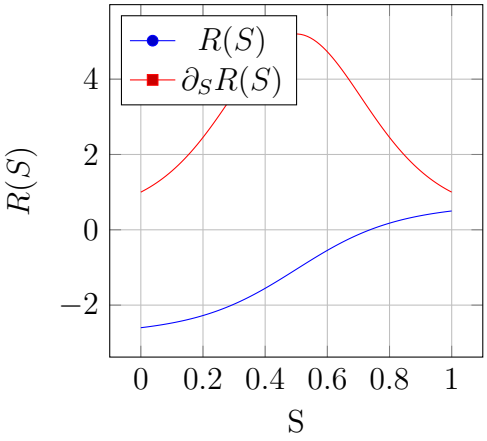

(a) $S_0 = 0.1$



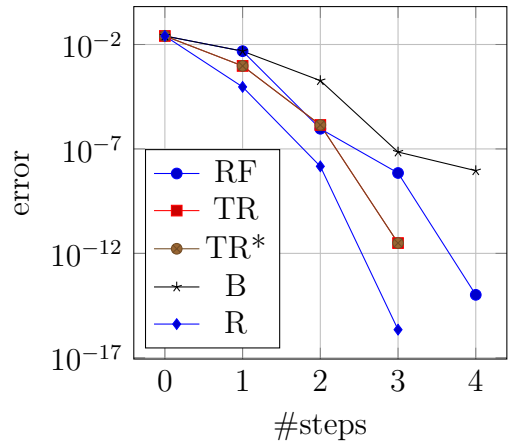

(b) $S_0 = 0.1$



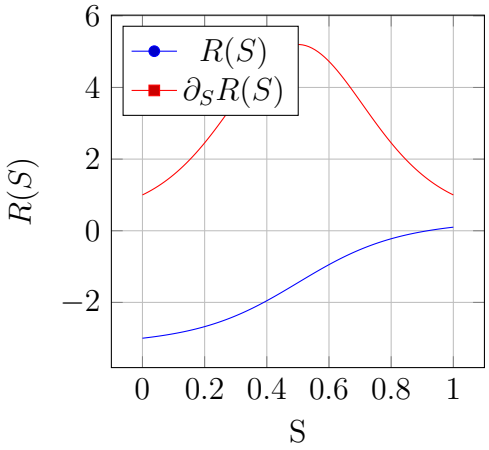

(c) $S_0 = 0.5$




(d) $S_0 = 0.5$

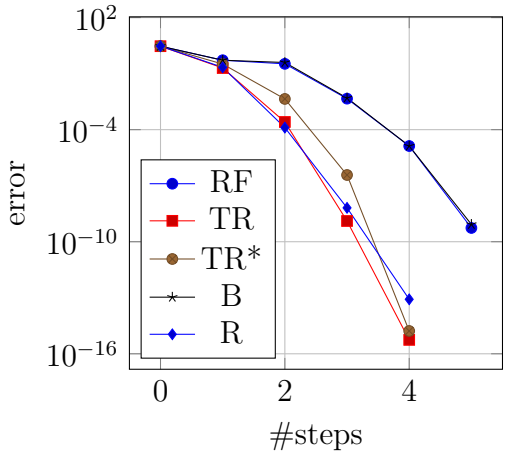



(e) $S_0 = 0.9$

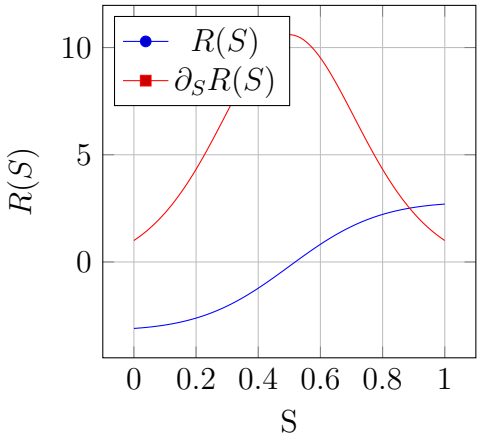



(f) $S_0 = 0.9$

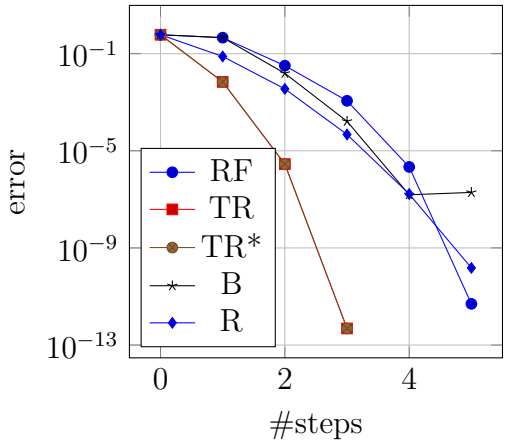
Figure 3.19: Convergence history and residuals with parameters $M = 1$, $\tau = 6\,\mathrm{s/m^3}$, $q_i = -0.35\,\mathrm{m^3/s}$, $q_o = 0.35\,\mathrm{m^3/s}$



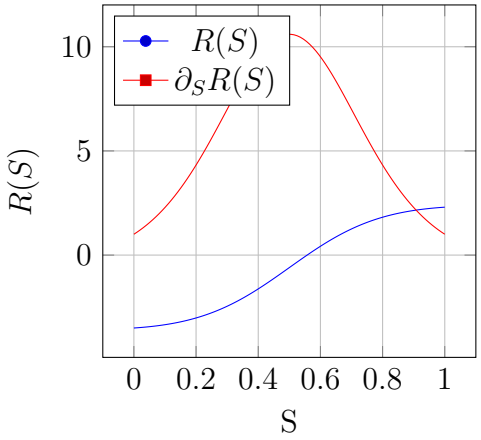

(a) $S_0 = 0.1$



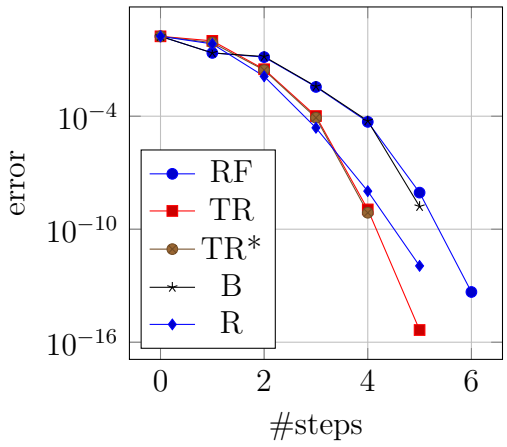

(b) $S_0 = 0.1$



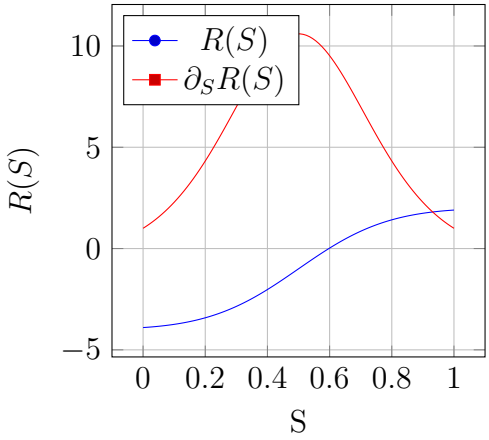

(c) $S_0 = 0.5$




(d) $S_0 = 0.5$




(e) $S_0 = 0.9$




(f) $S_0 = 0.9$

Figure 3.20: Convergence history and residuals with parameters $M = 1$, $\tau = 6\,^{\mathrm{s}}/_{\mathrm{m}^3}$, $q_i = -0.5\,^{\mathrm{m}^3}/_{\mathrm{s}}$, $q_o = 0.8\,^{\mathrm{m}^3}/_{\mathrm{s}}$

# Chapter 4

# Discussion

## 4.1 Two-Dimensional Domains

The results from Sections 3.2.1, 3.2.2, and 3.2.3 all show very similar results, with the Precise Trust Region methods as the fastest method both in terms of number of iterations and the CPU time spent by the transport solver. The following sections seeks to explain these results.

### 4.1.1 Case A - Large Grid Cells

Figure 4.1 shows a sorted distribution of all the converged cell saturations $S_V^{n+1}$ for all time steps $\Delta t$ and the usual $M$ values for case A, Section 3.2.1. The mean $\mu_S$ and standard deviation $\sigma_S$ of the distributions, shown in Table 4.1, indicates that the viscosity ratio $M$ has significant impact on the converged saturation distribution. It seems that large $M$ values give a larger average water saturation in the cells, while smaller $M$ give smaller average saturation values. The trend in the shape of the distribution is even clearer, with larger values of $M$ pushing the distribution to the right and leaving a larger fraction of saturations near zero. These observations are explained by again noting that small $M$ gives values of the fractional water flow function $f_w$ closer to one, while larger $M$ keeps $f_w$ close to zero, as shown in Figure 2.1. The function $f_w$ measures the water flow, and scales the outgoing flux in the transport residual, Equation (4.3.1). Thus, large $f_w$ values gives a large flow of water out of the cell, while small values gives a small flow. A large outflow of water will lead to smaller saturation values in the cell, and vice versa. Thus, a large $M$ gives a small outflow of water, since it corresponds to a small $f_w$, leaving more water in the cell. This is exactly what we observe in Figure 4.1 and Table 4.1. Recalling the definition in Equation (2.1.8) of $M$ as the ratio of water viscosity to oil viscosity, the physical interpretation

of a large $M$ is that water flows easier than oil, which corresponds with these observations. Since the injected amount of water is independent of $M$ this discussion also explains the smearing of saturation values shown in Figure 4.1.

Table 4.1: Standard deviation $\sigma$ and mean $\mu$ of the converged cell saturations $S^{n+1}$ and initial guess error $e_{S_0} = |S^{n+1} - S^n|$ for the Q5 problem, Section 3.2.1. The problem was solved with $\Delta t = 50\,\mathrm{d}$, cell size $120\,\mathrm{m} \times 120\,\mathrm{m}$, and viscosity ratio $M$. Correlation coefficients $\rho_{i,e}$ between the iteration count and $e_{S_0}$ are also shown., where $\rho'_{i,e}$ denotes results without data points where $e_{S_0} \leq \epsilon$

| $M$ | $\mu_S$ | $\sigma_S$ | $\mu_e$ | $\sigma_e$ | $\rho_{i,e}$ | $\rho'_{i,e}$ |
|---|---|---|---|---|---|---|
| 0.1 | 0.3549 | 0.1832 | 0.0814 | 0.0677 | 0.7103 | 0.6293 |
| 1 | 0.4815 | 0.3104 | 0.1219 | 0.1242 | 0.7750 | 0.6908 |
| 10 | 0.5224 | 0.4210 | 0.1474 | 0.2192 | 0.8258 | 0.7822 |

Figure 4.2 shows (truncated) initial guess errors $e_{S_0}$ and iteration counts for the Precise Trust Region method when using the old cell saturation as a starting point. The error is defined by

$$e_{S_0} = |S^{n+1} - S^n|.$$

Data points in the figure are shown in the order determined by the reordering procedure, starting from the left at time zero. Note that all time steps are shown, such that every block of 400 data points contains the single cell problems in one time step $\Delta t$. This causes the periodic nature of the plots. We also stress that the initial guess errors are truncated at the machine precision $10^{-16}$, giving a solid line at this point, and that a tolerance of $\epsilon = 10^{-9}$ is used. Thus, values of $e_{S_0}$ below $\epsilon$ gives convergence in only one iteration.

The plots indicate that there is some correlation between $e_{S_0}$ and the number of iterations, with large initial guess errors generally giving a higher iteration count. A basic statistical analysis gives the correlations coefficients $\rho_{i,e}$ shown in Table 4.1, indicating a strong positive correlation between the error and the amount of iterations for all viscosity ratios. Since $e_{S_0} \leq \epsilon$ gives convergence in one iteration the correlation seems higher than it actually is. Table 4.1 also shows the correlation coefficients, denoted $\rho'_{i,e}$, for only the data points with initial guess errors above $\epsilon$. As predicted, the coefficients are smaller, but the data still supports the conclusion that there is positive correlation between $e_{S_0}$ and the number of iterations used by the Precise Trust Region method.

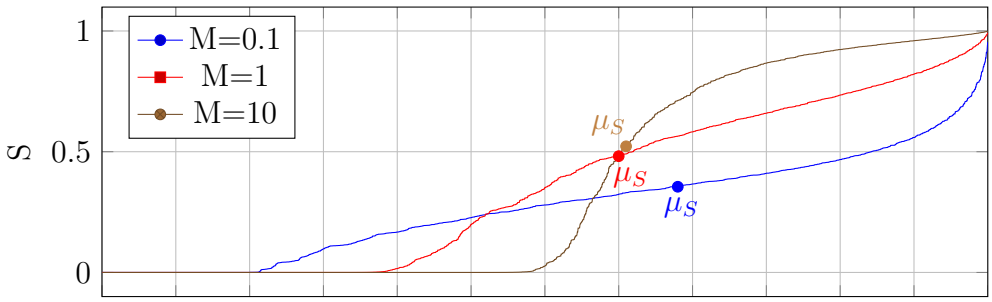
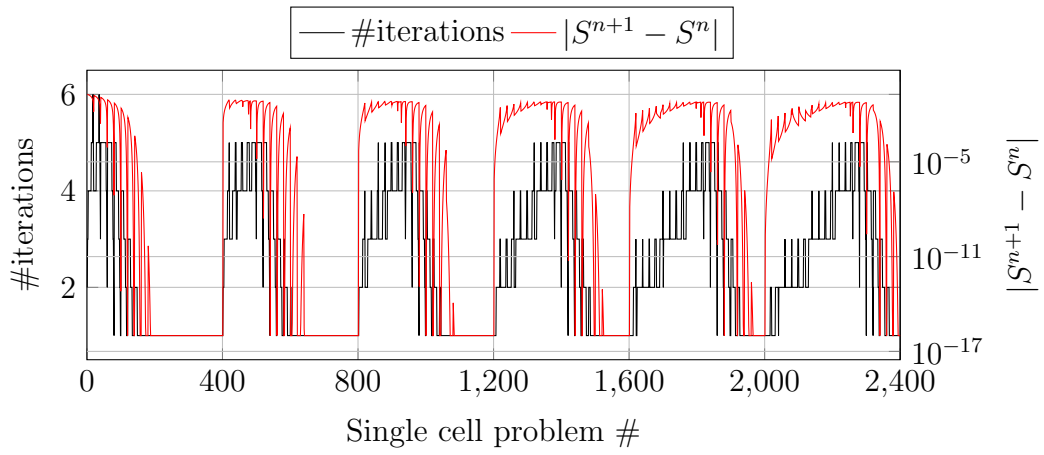
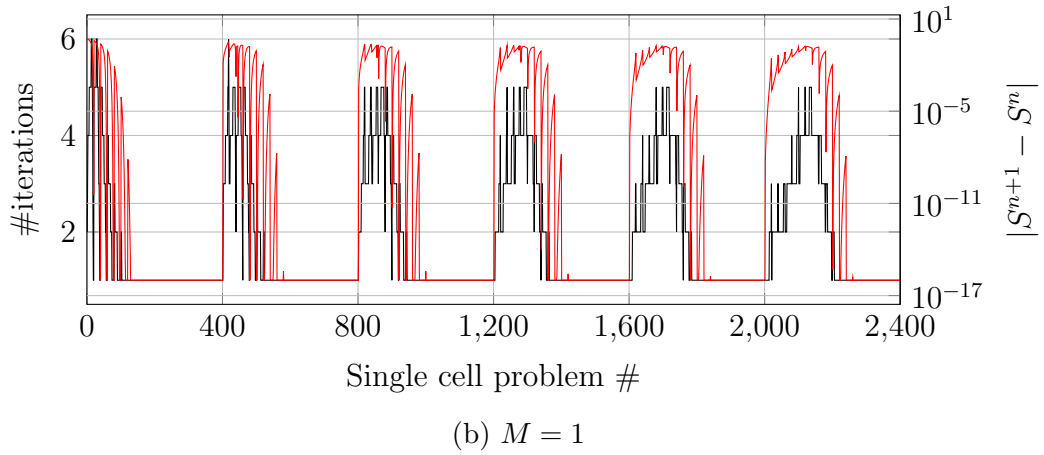



Figure 4.1: Distribution of converged $S^{n+1}$ for the Q5 problem, Section 3.2.1, with $\Delta t = 50\,\mathrm{d}$, $120\,\mathrm{m} \times 120\,\mathrm{m}$ cells, and mean values $\mu_S$, see Table 4.1.

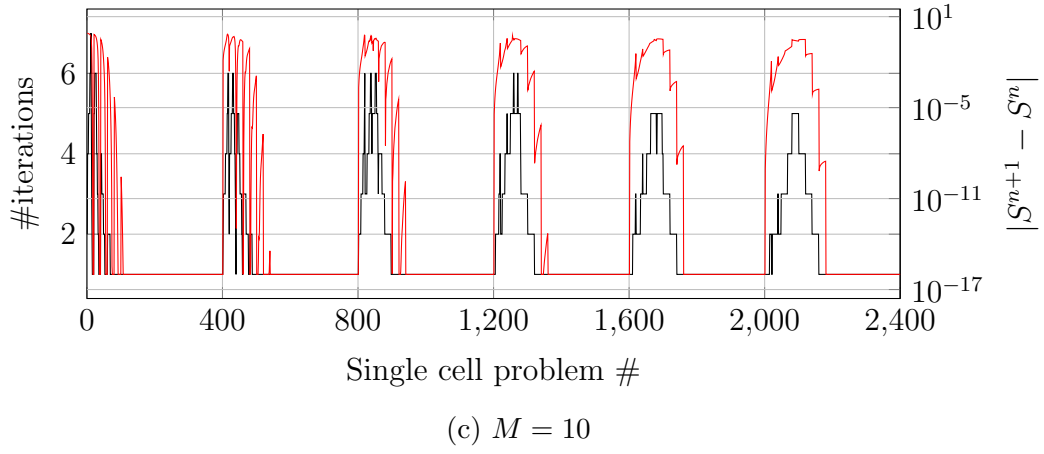
Another feature in Figure 4.2 is that the there is a spike in the iteration data in each 400 cell time step block. In the first time step, i.e., for single cell problems 1 through 400 in the figure, the spike is located at the beginning of the region. For larger time step the spike is located further right relative to the start of the block, indicating that the iteration count has a spatial dependency. Due to the nature of the reordering scheme the single cell problems close to the source are encountered first, and thus shows up early in each time step block in the figure. As the simulation progresses, the saturation front moves towards the sink, as shown in Figure 3.1. This figure, along with Figure 4.2 hints that the high iteration single cell problems are encountered along the saturation front. Figure 4.3 shows this more clearly with a plot of the iteration count per cell over the entire domain for all time steps. This result seems reasonable since this is exactly where the saturation changes are large. We also observe that the iteration spikes are sharper for larger $M$, and more smeared out for smaller $M$, indicating that a relatively high number of iterations are needed over a larger part of the domain when $M$ is small. These observations are in good agreement with the earlier discussion of the saturation distribution in Figure 4.1. There we stated that large $M$ gives smaller flow, giving a more defined saturation front, and vice versa. Figure 4.2 also supports the observations made from Figure 3.4, namely that the average CPU time decreased with increasing $M$. It is clear from Figure 4.2 that the iteration count is lower for larger $M$ values, giving lower CPU times.




(a) $M = 0.1$

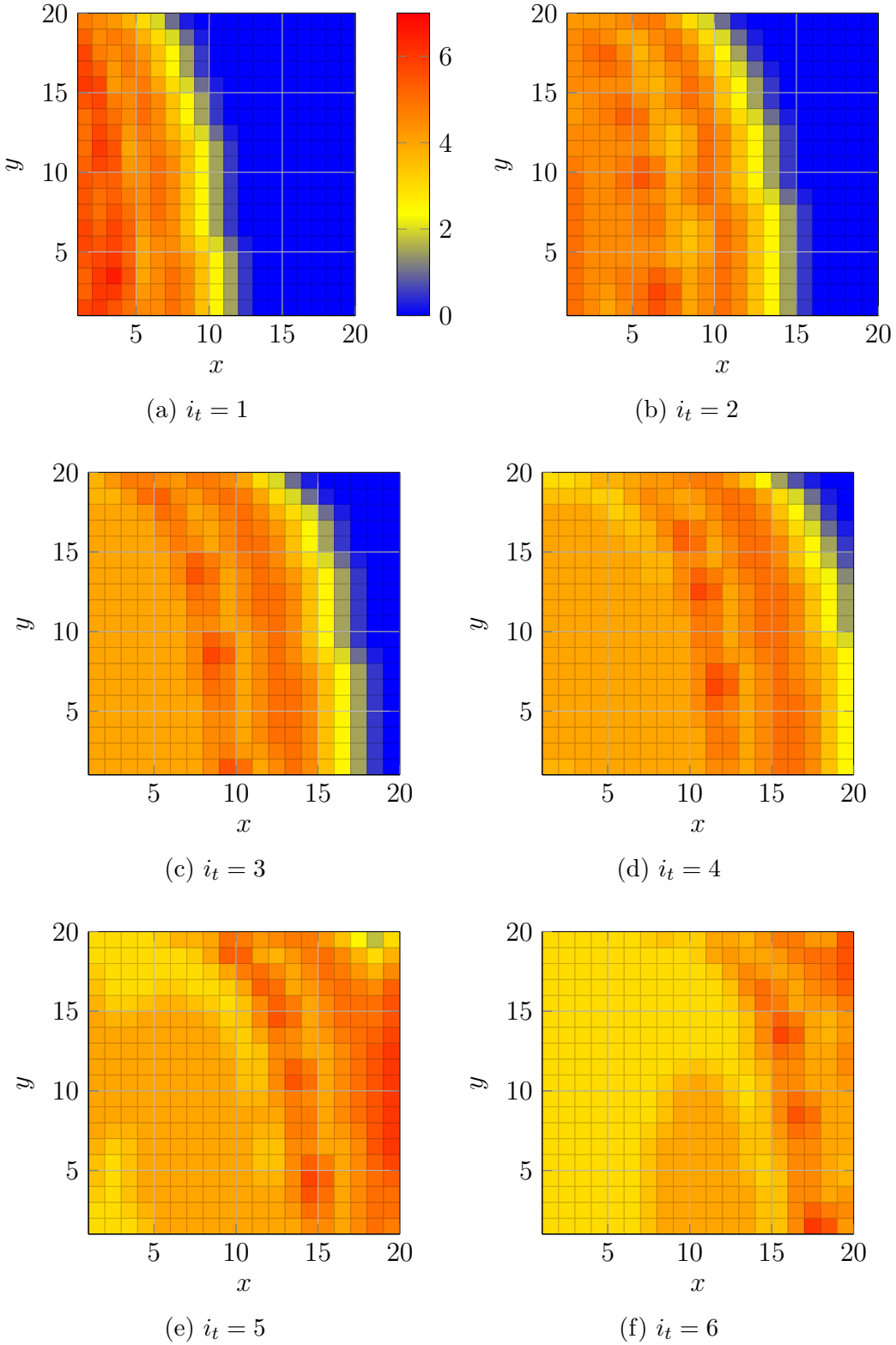



(b) $M = 1$




(c) $M = 10$

Figure 4.2: Initial guess error $|S^{n+1} - S^n|$ and #iterations spent by the precise trust region method to converge for all single cell problems as sorted by the reordering method in case A, see Section 3.2.1. 120 m × 120 m cells are used.




(a) $i_t = 1$ (b) $i_t = 2$

(c) $i_t = 3$ (d) $i_t = 4$

(e) $i_t = 5$ (f) $i_t = 6$

Figure 4.3: Number of iterations spent in each cell of the computational domain for each time step $i_t$ of case A with $\Delta t = 50\,\mathrm{d}$. Cell dimensions $120\,\mathrm{m} \times 120\,\mathrm{m} \times 10\,\mathrm{m}$ are used.

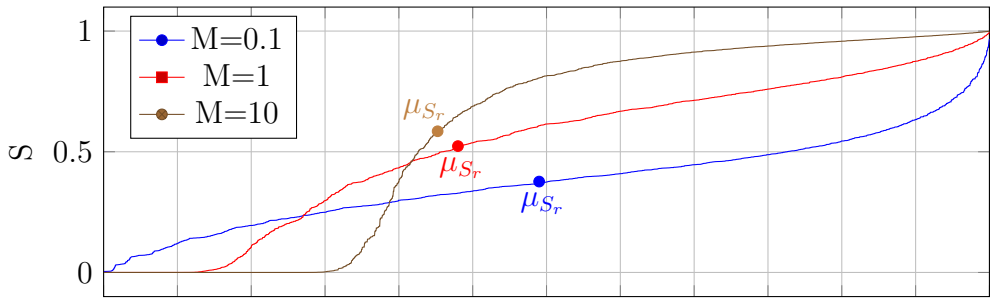



Figure 4.4: Sorted cell saturations, 2400 in total, from all time steps for the solution of the Q5 problem in Section 3.2.1 with $\Delta t = 50\,\mathrm{d}$. Values for viscosity ratios $M = 0.1$, $M = 1$, and $M = 10$ are reported with corresponding mean values $\mu_{S_r}$ at 0.3765, 0.5234, and 0.5849, respectively.

### 4.1.2 Case A - Small Grid Cells

The sorted converged cell saturation values are shown for the case with cell dimension $10\,\mathrm{m} \times 10\,\mathrm{m}$ in Figure 4.4. The overall trend in the data is the same as for the $120\,\mathrm{m} \times 120\,\mathrm{m}$ cell case, but with slightly larger mean saturation values, as shown in Table 4.2. That is, we still observe the skewing of saturation values to the right for large $M$. The difference is that in the present case a large percentage of the saturation values are non-zero. When the cell size was reduced from 120 m to 10 m the source strength was also reduced. But, since the permeability is unchanged and the adjusted source strength is larger relative to the cell size, the water flow reaches further into the domain with the smaller cells. The effect is that a larger portion of the domain contains water, and this happens relatively early in the simulation, as observed in Figure 4.4.

Figure 4.5 shows the iteration count and initial guess errors $e_{S_0}$ as for the case with cell dimensions $120\,\mathrm{m} \times 120\,\mathrm{m}$. As just stated, the average saturation values are larger with smaller grid cells. This fact has a clear impact on the iteration and initial guess data. The figure shows that the values are more uniform across each time step block for all viscosity ratios, although the spread is tighter for larger $M$ in this case to. Still, as indicated by the correlation coefficient in Table 4.2, high $e_{S_0}$ and high iteration counts are strongly correlated. We also observe the same iteration spike advancing with the saturation front, although less pronounced in this case because of the smeared saturation front. Since more cells are reached by the flow we observe a larger portion of the domain with significant flow, giving a more uniform distribution of iterations in the domain. The spatial dependency of the iteration count is shown more clearly in Figure 4.6, with a plot of the

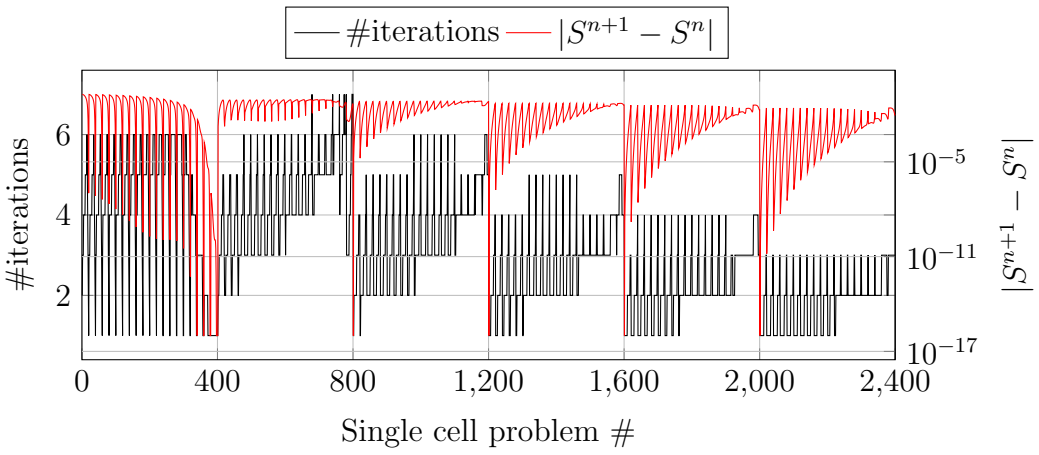
iterations spent in each cell over the whole domain for all time steps.

Table 4.2: Standard deviation $\sigma$ and mean $\mu$ of converged $S^{n+1}$ and initial guess error $e_{S_0} = |S^{n+1} - S^n|$ and correlation coefficients $\rho_{i,e}$ for the iteration count and $e_{S_0}$ for the Q5 problem, Section 3.2.1 with $\Delta t = 50\,\mathrm{d}$, cell size $10\,\mathrm{m} \times 10\,\mathrm{m}$, and viscosity ratio $M$. Correlation for data with truncated values removed is denoted $\rho'_{i,e}$

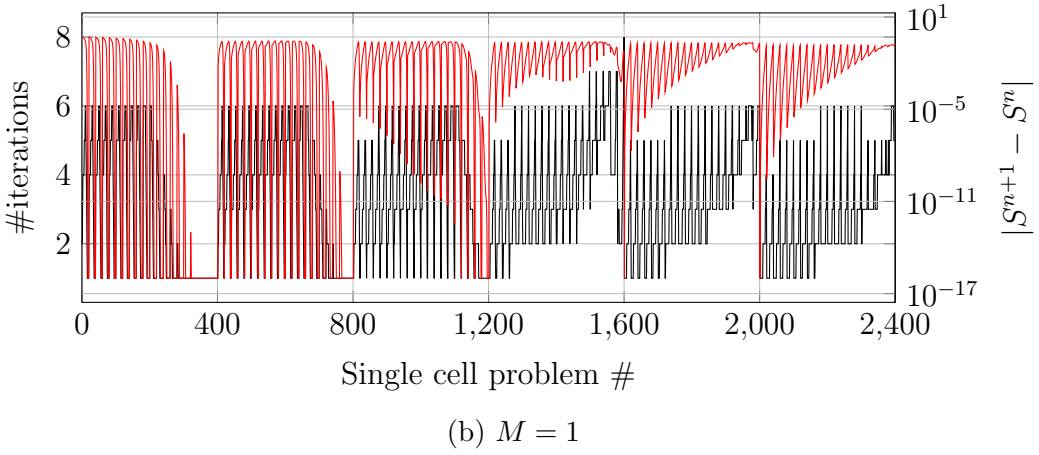
| $M$ | $\mu_S$ | $\sigma_S$ | $\mu_e$ | $\sigma_e$ | $\rho_{i,e}$ | $\rho'_{i,e}$ |
|---|---|---|---|---|---|---|
| 0.1 | 0.3765 | 0.1811 | 0.0849 | 0.0711 | 0.7103 | 0.6830 |
| 1 | 0.5234 | 0.2982 | 0.1266 | 0.1275 | 0.7750 | 0.7674 |
| 10 | 0.5849 | 0.4092 | 0.1548 | 0.2233 | 0.8258 | 0.8570 |

### 4.1.3 Case B and C

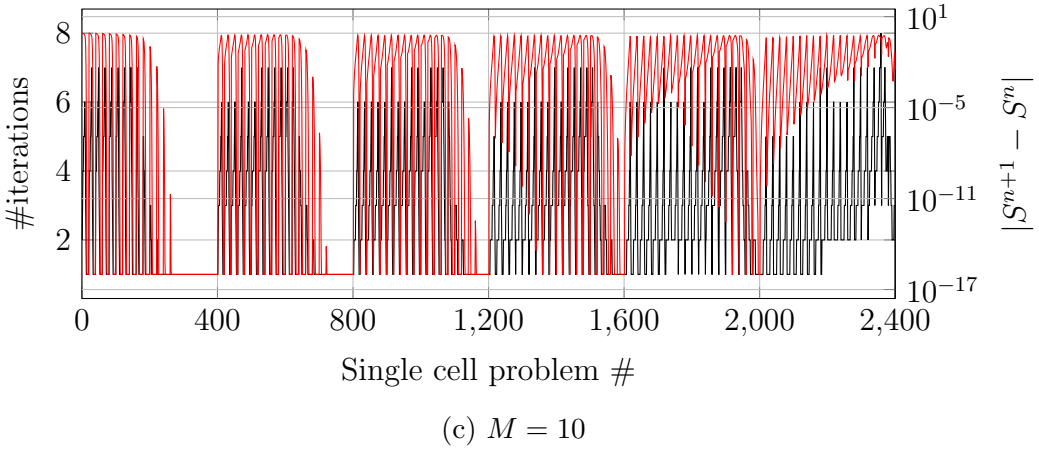
The two cases with inhomogeneous permeabilities gathered from the SPE10 data set were included to investigate the impact of permeabilities on the root finder efficiency. The numerical results presented in Sections 3.2.2 and 3.2.3 shows little difference between the two cases for all tested parameter and root finder combinations. The one exception is that the reduction in CPU time with the viscosity ratio $M$ is much less pronounced for case C, as shown in Figure 3.14. This is caused by the sharp permeability gradients, since the fluid fills the high-permeability zones quickly even with the lowered flow caused by high $M$ values. Thus, increasing $M$ has little impact on the spread of the permeability front causing a comparable amount of high flow cells for all viscosity ratios. This effect is not observed in case B, since the permeabilities have fewer sharp fronts causing water saturation build up, so low viscosity ratios still causes a smeared saturation front resulting in a larger region with significant flow, and a higher computational load.




(a) $M = 0.1$

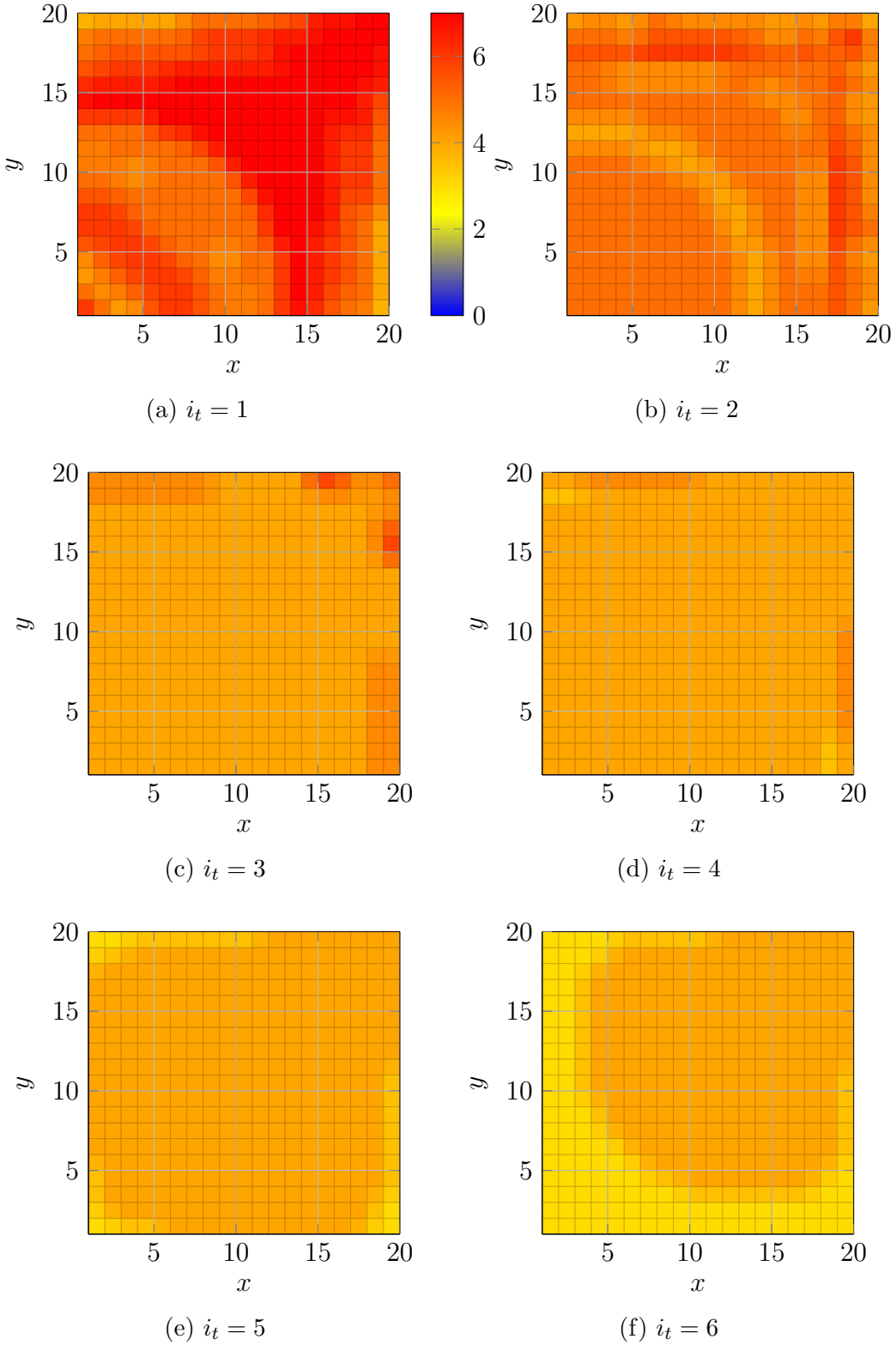



(b) $M = 1$




(c) $M = 10$

Figure 4.5: Initial guess error $|S^{n+1} - S^n|$ and #iterations spent by the precise trust region method to converge for all single cell problems as sorted by the reordering method in case A, see Section 3.2.1. 10 m × 10 m cells are used.




(a) $i_t = 1$ (b) $i_t = 2$

(c) $i_t = 3$ (d) $i_t = 4$

(e) $i_t = 5$ (f) $i_t = 6$

Figure 4.6: Number of iterations spent in each cell of the computational domain for each time step $i_t$ of case A with $\Delta t = 50\,\mathrm{d}$. Cell dimensions $10\,\mathrm{m} \times 10\,\mathrm{m} \times 10\,\mathrm{m}$ are used.

## 4.2 Three-Dimensional Domain

Our implementation of the Precise Trust Region method computes the inflection point from the fractional water flow function $f_w$ as described in Section 2.4.2.6, which is the wrong inflection point for the gravity residual in Equation 2.4.9. Since the Precise Trust Region method computes the inflection point for the viscous residual it chops back to the wrong saturation value. In practice this causes the method to miss the trusted region of the residual, giving worse convergence properties than when used correctly on the viscous transport residual in Equation 2.4.7. With the wrong inflection point the Precise Trust Region works similarly to the Appleyard and Modified Appleyard Heuristics briefly mentioned in Section 2.4.2.6, giving good convergence but wasting a significant number of iterations because of too imprecise update chops. Despite this the Precise Trust Region method seems to perform adequately in terms of CPU time except for the smallest time steps, as observed in Figure 3.16. These results can be explained by remembering that, as previously discussed, small viscosity ratio values increases the significance of the viscous flow. That is, with $M = 0.1$ the viscous flow dominates the gravity induced flow. This causes the Precise Trust Region method to perform almost as god as with pure viscous flow, since the majority of the computational effort is spent on the viscous residual, where the Precise Trust Region method performs well. With the larger viscosity ratios the gravity flow becomes more significant, which explains the increase in the number of iterations for larger $M$ and small time steps in Figure 3.16. For large time steps this effect is not seen as clearly because the number of extra iterations per time step caused by the wrong trust regions is fairly low. With smaller $\Delta t$ the extra iterations caused by the bad trust region identification is compounded over each time step, causing a larger number of extra iterations. This is seen as a performance drop both in terms of iterations and CPU time for small time steps, the latter shown in Figure 3.17. Still, despite the obvious problem with wrong trust regions, the Precise Trust Region method performs adequately on the three-dimensional problem with gravity.

## 4.3 Convergence Tests

We restate the viscosity residual from Equation (2.4.7) with simplified notation:

$$R(S^{n+1}; S^n, q_i, q_o, M, \tau) = S^{n+1} - S^n + \tau \left(q_o f_w(S^{n+1}) + q_i\right) = 0 \qquad (4.3.1)$$

Here $\tau$ is defined by

$$\tau = \frac{\Delta t}{m(V)\phi_V}.$$

The flow induced by the pressure gradient is governed by the last term of Equation (4.3.1), i.e., $\tau\left(q_o f_w(S^{n+1}) + q_i\right)$, through the flux values $q_o$ and $q_i$. We call this the *flow term* of the transport equation. The observations from the convergence tests in Section 3.3 indicates that the cell saturation from the previous transport step can be a determining factor for the root placement under certain circumstances. In other cases the previous cell saturation has less influence. The tests showed that the trust region methods are highly dependent on a good initial guess. This is also supported by the theory, in that the Newton method has quadratic convergence only locally. It is of interest to know *a priori* under which circumstances the old saturation value is a good starting guess for the root finders. We now seek to determine under which circumstances a good initial guess can be obtained.

Since $S^n \in [0, 1]$ we have a well defined range for the size of the two first terms in Equation (4.3.1). This implies that the old cell saturation $S^n$ is significant when $\tau\left(q_o f_w(S^{n+1}) + q_i\right) \approx 1$, in the sense that the size of the flow term is of the same order of magnitude as the number 1. Of course, $S^{n+1} = S^n$ when $\tau\left(q_o f_w(S^{n+1}) + q_i\right) = 0$. Likewise, if the flow term is much larger than 1, the solution $S^{n+1}$ is completely dominated by the fractional flow function $f_w$ and the flux terms $q_i$ and $q_o$. These facts explains the observations made based on the convergence plots in Section 3.3. The size of the flow term in Equation (4.3.1) is determined by the incoming and outgoing fluxes, and the factor $\tau$. The fluxes measure the magnitude of flow in and out of the current cell, while $\tau$ gives the time-volume scale of the flow. That is, $\tau$ measures the number of seconds the fluxes $q_i$ and $q_o$ are allowed to move across the cell boundaries per volume unit of the cell. In practice this leads to small cells being drained or filled faster than larger cells, and a larger flow in each iteration for large time steps. The idea is that different combinations of the flux and time scale magnitudes causes the flux term in the transport equation to have varying significance on the root placement.

We start by treating the case $q_i = 0$. Now, Equation (4.3.1) becomes

$$R(S^{n+1}) = S^{n+1} - S^n + \tau q_o f_w(S^{n+1}) = 0$$

If $\tau q_o \gg 1$ the equation reduces to $f_w(S^{n+1}) = 0$, which will be positive and much larger than zero for all but the smallest $S^{n+1}$. As $S^{n+1}$ goes to zero, the influence of $S^{n+1}$ becomes more significant, pulling the root to the right. The root remains close to zero in this case. With $\tau q_o$ of the same order of magnitude as $S^n$ the root will be significantly influenced by $S^n$.

Now, if $q_o = 0$ the residual in Equation (4.3.1) reduces to

$$R(S^{n+1}) = S^{n+1} - S^n + \tau q_i = 0 \ \Rightarrow S^{n+1} = S^n - \tau q_i$$

By definition $q_i \leq 0$, so this requires $0 \leq -\tau q_i \leq 1 - S^n$. The lower bound is always satisfied since $\tau \geq 0$. Again $S^n$ is significant in determining the root placement.

The time scale $\tau$ is always greater than zero, and usually large when compared to the saturation values when working on a reservoir scale, since the value is given with unit seconds. With $q_0 f_w$ not very small we expect the flux term to dominate the root placement. With small $S^{n+1}$, $f_w(S^{n+1})$ becomes close to zero. This causes the flux term dominance to be reduced close to $S^{n+1} = 0$, in effect reducing the flux term dominance. This effect is pronounced with $M < 1$, since $f_w$ is small over a larger range in that case, as shown in Figure 2.1. In this case we expect the root to be close to zero if $\tau q_i \ll 1$. Otherwise the root is given by

$$\frac{q_o}{q_i} f_w(S^{n+1}) + 1 \approx 0, \tag{4.3.2}$$

which, with quadratic relative permeabilities $k_{rl}$, becomes the second order equation

$$S_\pm = \frac{2M \pm \sqrt{4M^2 - 4M(1 + M + r_q)}}{2(1 + M + r_q)} = \frac{M \pm \sqrt{-M(1 + r_q)}}{1 + M + r_q},$$

where $r_q$ is defined by

$$r_q \coloneqq \frac{q_o}{q_i}.$$

The special case $1+M+r_q = 0$ implies $S = 1/2$. In the following we therefore assume $1 + M + r_q \neq 0$. In order for the quadratic formula to yield a solution we need $r_q \leq -1$. We define a number $\alpha = |1 + r_q|/M$, such that

$$S_\pm = \frac{1 \pm \sqrt{\alpha}}{1 - \alpha},$$

and $\alpha \geq 0$. The degenerate case $\alpha = 0$ gives $S_\pm = 1$. Otherwise, with $\alpha > 0$ we see that

$$S_+ = \frac{1 + \sqrt{\alpha}}{1 - \alpha}.$$

The saturation $S_+$ is clearly greater than unity for $0 < \alpha < 1$, further $\alpha = 1$ corresponds to $M+1+r_q = 0$, so $S_+ = 1/2$ as stated above. Last, with $\alpha > 1$ we get a positive numerator and a negative denominator, yielding a negative $S_+$, and invalid saturation solutions. We thus only consider the solution

$$S = S_- = \frac{1 - \sqrt{\alpha}}{1 - \alpha}. \tag{4.3.3}$$

The case $\alpha = 1$ still gives $S = 1/2$. With $0 < \alpha \neq 1$ we get $0 < S < 1$ since $1 - \sqrt{\alpha} < 1 - \alpha$, which holds because $\sqrt{x} < x$ for $x > 1$ and $\sqrt{x} > x$ for $0 < x < 1$. This lengthy discussion has provided a simple formula for an approximate solution to the transport equation, the point being that it can be used as to generate a better initial guess than the old cell saturation $S^n$ provides, in certain cases. The idea is to these updates initial guesses to improve the convergence speed for the Precise Trust Region method.

## 4.4 Initialized Precise Trust Region

The results from the previous sections show that the Precise Trust Region method is the fastest solver overall, especially for cases without gravity influence. Further, the numerical results and theory suggests that improved initial guesses can speed up the method even more, and that this is of special interest in high flow regions where the old cell saturation often is far from the true root. The next subsection describes a numerical scheme based on the considerations in Section 4.3 and some practical tests to determine parameters sizes.

### 4.4.1 Implementation

The Initialized Precise Trust Region is implemented by using the precomputed residual equation parameters $q_o$, $q_i$, $\tau$, $M$, and $S_0$ to identify regions where the old cell saturation $S_0$ provides a bad approximation to the root $S_r$. The proposed initializer, shown as pseudo code in Algorithm 8, is executed before each call to the Precise Trust Region method to give a better initial saturation guess. Note that the current implementation tries to use the initial guess approximation on every single cell problem.

### 4.4.2 Numerical Results

The initialization technique presented in the previous section, here denoted by the letters IN, was tested on cases A through C from Sections 3.2.1, 3.2.2,

**Algorithm 8:** Pseudo code implementing the initialization method proposed in Section 4.4.

**Data**: Old cell saturation $S_0$, outgoing flux $q_o$, incoming flux $q_i$, viscosity ratio $M$, time step to pore volume ratio $\tau$

**Result**: Improved root approximation $S_r$

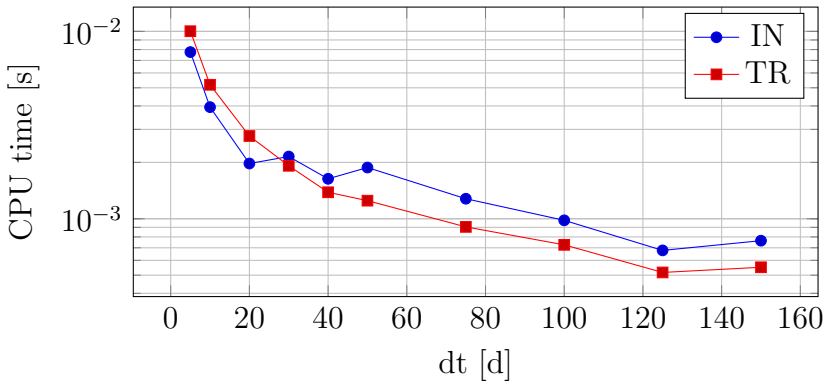
```
r_q := q_o / q_i;
α := |r_q + 1| / M;
S_r := (1 − √α) / (1 − α);
return S_r
```

and 3.2.3, respectively. Only the best root finder from previous tests is shown for each case, namely the Precise Trust Region method. We omit the three-dimensional test because of the erroneous trust region identification used by the Precise Trust Region method for the gravity residual. The CPU time results from case A are shown in Figure4.7 large and small grid cells. The plots show that the IN method outperforms the Precise Trust Region scheme only for low time step sizes. Here the computational overhead of using the initializer on every single cell residual is so low, and the impact of bad initial guesses so high, that the method works good even without a classification of bad initial guesses. For the $\Delta t$ the transport solver is very fast, such that the overhead of computing the initial guess approximations becomes large enough to dominate the overall results.

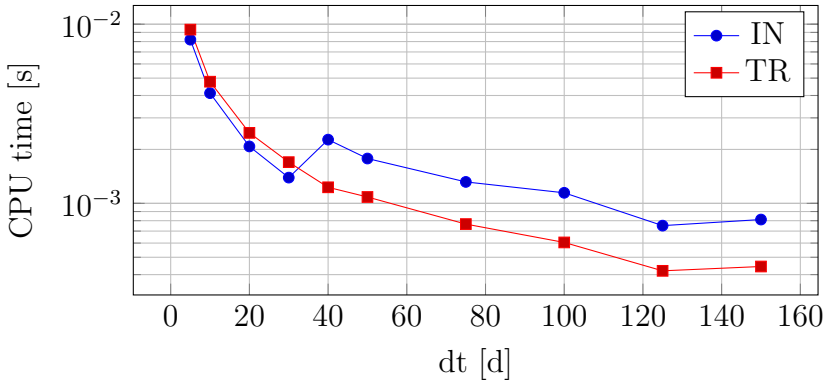
Figure 4.8 shows the results from using the new technique on case B. Here we see that the IN method improves with increasing $\tau$, the time scale factor introduced in Section 4.3. Larger $\tau$ gives a larger flow which, as discussed, should give a worse initial guess. The method also performs worse for larger $M$, which corresponds with the flow size observations since large $M$ corresponds with smaller flow. For the two lowest viscosity ratios the time steps size also has an effect.

Finally, Figure 4.9 shows that the IN method performs on par with or slightly worse than the Precise Trust Region method on case C. This results from the large number of zero flow cells, i.e., the cells outside the high permeability zones. Here the initial guess approximation provided by the IN approach is not so good. This is balanced by large flows in the high permeable regions, making the two methods perform similarly.

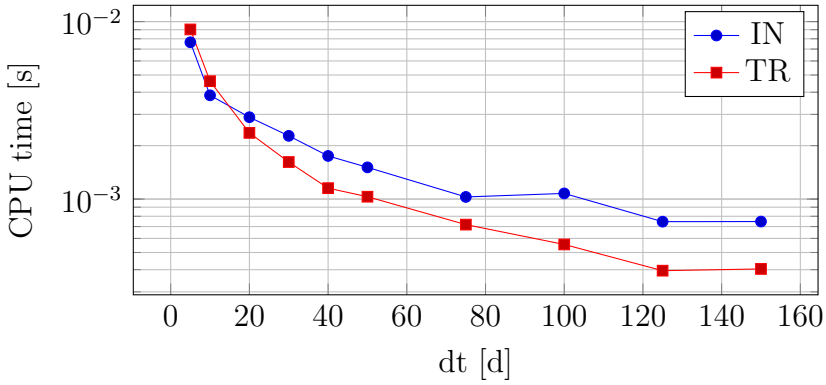
The results presented here indicates that the IN approach can have a positive impact on the transport solver CPU time in special cases. Further work is required to classify regions where the method is useful, potentially reducing the number of unnecessary or bad invocations of the approximation.




(a) $M = 0.1$

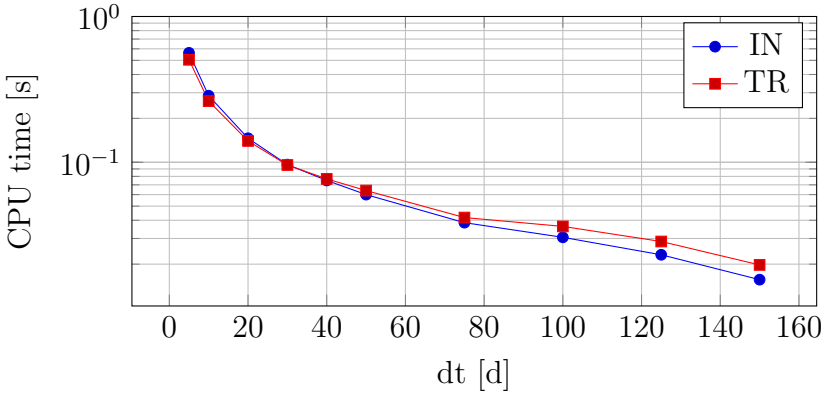



(b) $M = 1$

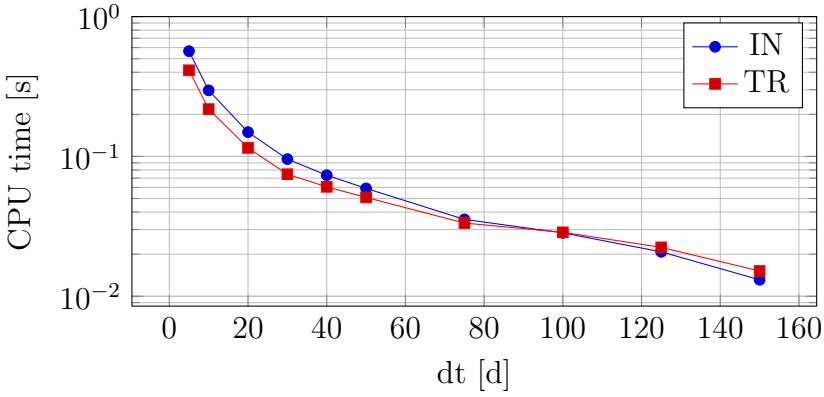



(c) $M = 10$

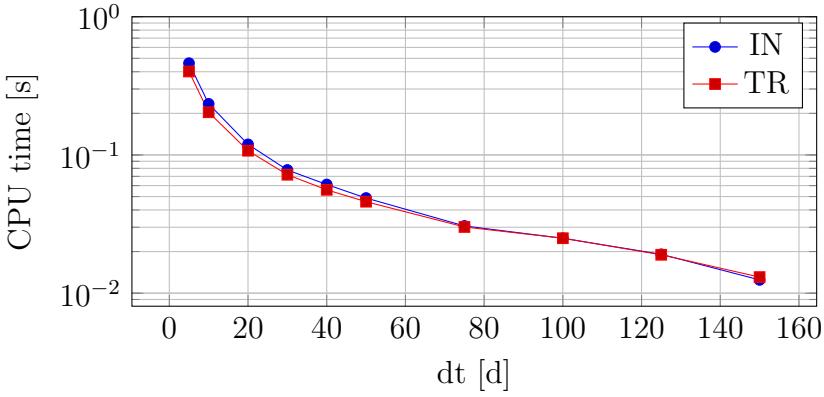
Figure 4.7: CPU time used by the transport solver when solving case A, Section 3.2.1, for varying root finders, time steps and viscosity ratios. Note that 120 m × 120 m cells are used here.




(a) $M = 0.1$

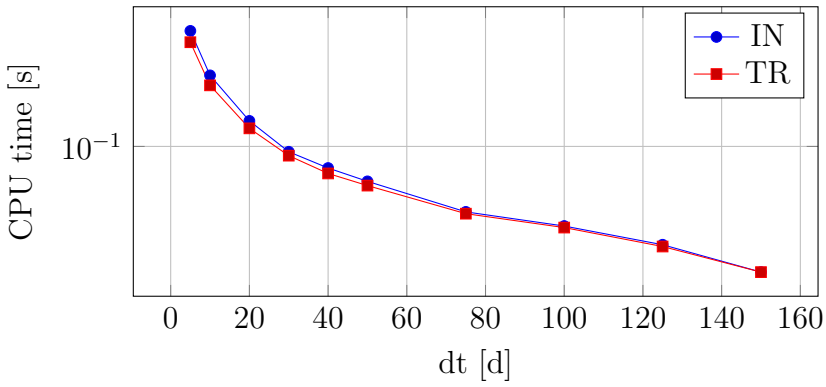



(b) $M = 1$

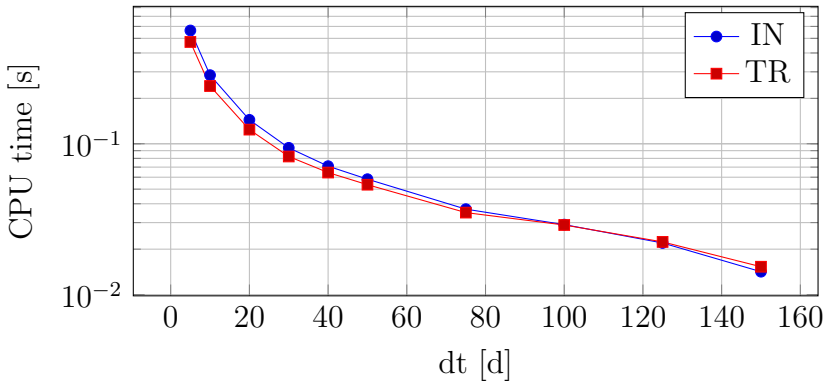



(c) $M = 10$

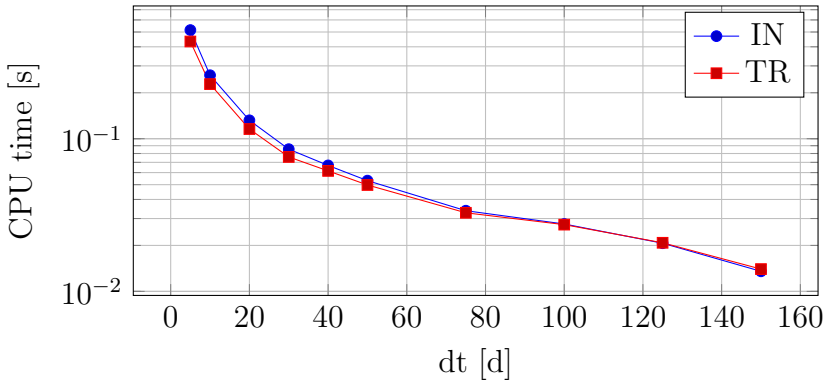
Figure 4.8: CPU time used by the transport solver when solving case B, Section 3.2.2, for varying root finders, time steps and viscosity ratios.




(a) $M = 0.1$




(b) $M = 1$




(c) $M = 10$

Figure 4.9: CPU time used by the transport solver when solving case C, Section 3.2.3, for varying root finders, time steps and viscosity ratios.

# Chapter 5

# Conclusion

This work has given a general introduction to reservoir simulation and the physical models and mathematical tools used to simulate flow phenomena in porous media. Special attention was given to the numerical methods used in the reservoir simulator provided by the Open Porous Media initiative, which has been thoroughly covered herein. In particular, the sequential splitting method was covered, along with the finite volume method and the reordering approach due to Lie et al. [2013]. Several numerical root finders were introduced to solve the single cell problems resulting from the reordering technique.

Numerical tests have been performed to test the new root finder implementations. The tests show that the pressure solver, i.e., the two-point flux approximation scheme, dominates the simulation run time for the tested cases. Further, the tests show that the Brent and Ridders methods perform on par with or slightly worse than the reference Regula Falsi root finder for the tested cases.

Our version of the Approximate Trust Region method is shown to perform worse than the other root finders for all tests cases, and it fails to converge for the three-dimensional case with gravity. The method fails because the trust region identification is wrong for the gravity dominated residual.

Further, the bracketing methods, namely the Brent, Ridders, and Regula Falsi methods are shown to converge fast with little dependence on the provided initial guess. This result follows from the global nature of bracketing schemes. Likewise, in situations where the initial guess is good, these methods fails to take full advantage of this. Further, the Regula Falsi method with the Pegasus method is found to be the best method among the bracketing schemes.

The Precise Trust Region Scheme is shown to be the fastest method for all tests without gravity effects. For viscosity dominated flow the method

performs on par with the bracketing schemes, due to the small impact of gravitational forces on the total flow in the tested problem. Finally, we have shown that the CPU time used by the transport solver with the Precise Trust Region root finder can be improved by providing better initial guesses, as in the IN scheme presented in Section 4.4.

## 5.1 Further Work

Further improvements of the speed of the transport solver is of interest in the reservoir simulation industry. This work indicates that the solution of the transport equation can be done even faster. In particular, the initial guess initialization scheme, IN, can be improved by providing better heuristics for regions with high flow, such that the initial guess approximation proposed in Equation (4.3.3) is only employed in regions where the extra computational overhead is worth the cost.

The Precise Trust Region method presented here can also be improved by exploring techniques to improve the classification of trust region for the gravity residual.

# Appendices

# Appendix A

# Test Drivers

```
(...) // Includes are omitted for brevity
int main (int argc, char ** argv)
try
{
  int nx = 20, ny = 20, nz = 1, layer = 0;
  int nxperm = 60, nyperm = 220;
  int nprint = 100;
  double xpos = 0, ypos = 0;
  double dxperm = 365.76, dyperm = 670.56;
  double dx = 10.0, dy = 10.0, dz = 10.0;
  double perm_mD = 10;
  double muw = 1, muo = 1;
  double time_step_days = 0.1, comp_length_days = 2;
  double srcVol = 0.2, sinkVol = -srcVol;
  double grav_x = 0, grav_y = 0, grav_z = 0;
  bool verbose = false, is_inhom_perm = false;
  Opm::RootFinderType solver_type = Opm::RegulaFalsiType;
  std::string perm_file_name = "spe_perm.dat";
  std::string execName = boost::filesystem::path(std::string(
    argv[0])).stem().string();

  using namespace Opm;

  if(argc > 1)
  parseArguments(argc, argv, muw, muo, verbose, time_step_days,
    comp_length_days, dx, dy, dz, nx, ny, nz, solver_type,
    printIterations, nprint, print_points_file_name,
    perm_file_name, layer, xpos, ypos, perm_mD, is_inhom_perm,
    srcVol, sinkVol, grav_x, grav_y, grav_z);

  std::vector<double> perm;

  GridManager grid_manager(nx, ny, nz, dx, dy, dz);
```

```
29  const UnstructuredGrid& grid = *grid_manager.c_grid();
    int num_cells = grid.number_of_cells;
31
    int num_phases = 2;
33  using namespace Opm::unit;
    using namespace Opm::prefix;
35  std::vector<double> density(num_phases, 1000.0); density[1] =
      800.0;
    double visc_arr[] = {muw*centi*Poise, muo*centi*Poise};
37  std::vector<double> viscosity(visc_arr, visc_arr + sizeof(
      visc_arr)/sizeof(double));
    double porosity = 0.5;
39  double permeability = perm_mD*milli*darcy;
    SaturationPropsBasic::RelPermFunc rel_perm_func =
      SaturationPropsBasic::Quadratic;
41
    IncompPropertiesBasic props(num_phases, rel_perm_func, density
      , viscosity, porosity, permeability, grid.dimensions,
      num_cells);
43  IncompPropertiesShadow shadow_props(props);

45  const double grav_arr [] = {grav_x, grav_y, grav_z};
    const double *grav = &grav_arr[0];
47  std::vector<double> omega;

49  double injectedFluidAbsolute = srcVol; // m^3
    double poreVolume = dz*dx*dy*porosity/(nx*ny);
51  double injectedFluidPoreVol = injectedFluidAbsolute/poreVolume
      ;

53  std::vector<double> src(num_cells, 0.0);
    src[0] = injectedFluidPoreVol;
55  src[num_cells-1] = -injectedFluidPoreVol;

57  FlowBCManager bcs;

59  LinearSolverUmfpack linsolver;
    IncompPropertiesInterface * prop_pointer;
61  prop_pointer = (IncompPropertiesInterface *) &props;
    IncompTpfa psolver(grid, *prop_pointer, linsolver, grav, NULL,
       src, bcs.c_bcs());
63
    WellState well_state;
65
    std::vector<double> porevol;
67  Opm::computePorevolume(grid, props.porosity(), porevol);

69  const double tolerance = 1e-9;
    const int max_iterations = 50;
```

```
71    Opm::TransportSolverTwophaseReorder transport_solver(grid, *
        prop_pointer, grav, tolerance, max_iterations, solver_type,
        verbose);

73    const double comp_length = comp_length_days*day;
      const double dt = time_step_days*day;
75    const int num_time_steps = comp_length/dt;

77    TwophaseState state; state.init(grid, 2);

79    std::vector<int> allcells(num_cells);
      for (int cell = 0; cell < num_cells; ++cell)
81      allcells[cell] = cell;
      state.setFirstSat(allcells, *prop_pointer, TwophaseState::
        MinSat);
83
      for (int i = 0; i < num_time_steps; ++i) {
85      psolver.solve(dt, state, well_state);
        transport_solver.solve(&porevol[0], &src[0], dt, state);
87    }
  }
89  catch (const std::exception &e) {
      std::cerr << "Program threw an exception: " << e.what() << "\n
        ";
91    throw;
  }
```

Listing A.1: The C++ program used to run the homogeneous 2D numerical tests in Section 3.2.1.

```
(...) // Includes are omitted for brevity
int main (int argc, char ** argv)
try
{
  const int NPRINT = 100;
  int nprint = NPRINT, nx = 20, ny = 20, nz = 1;
  int xpos = 0, ypos = 0, zpos = 0;

  double xpos_double = 0.0, ypos_double = 0.0;
  double dx = 10.0, dy = 10.0, dz = 10.0;
  double muw = 1, muo = 1;
  double time_step_days = 0.1, comp_length_days = 2;
  double srcVol = 0.2, sinkVol = -srcVol;
  double tol = 1e-9;
  double denswater = 1000.0, densoil = 800.0;

  bool verbose = false, solve_gravity_column = false,
    useInitialGuessApproximation = false;

  Opm::RootFinderType solver_type = Opm::RegulaFalsiType;

  string perm_file_name = "spe_perm.dat", print_points_file_name
    = "print_points.dat";
  string execName = boost::filesystem::path(std::string(argv[0])
    ).stem().string();

  using namespace Opm;

  double ddummy; bool bdummy;
  if(argc > 1)
    parseArguments(argc, argv, muw, muo, verbose, time_step_days
    , comp_length_days, dx, dy, dz, nx, ny, nz, solver_type,
    time_pressure_solver, printVTU, printIterations, nprint,
    print_points_file_name, perm_file_name, zpos, xpos_double,
    ypos_double, ddummy, bdummy, srcVol, sinkVol, ddummy, ddummy,
     ddummy, tol, bdummy, bdummy, useInitialGuessApproximation,
    printFluxValues,denswater,densoil);
  xpos = (int)xpos_double;
  ypos = (int)ypos_double;

  std::vector<double> perm;
  buildPermData(perm_file_name,perm,xpos,nx,ypos,ny,zpos,nz,
    verbose);

  GridManager grid_manager(nx, ny, nz, dx, dy, dz);
  const UnstructuredGrid& grid = *grid_manager.c_grid();
  int num_cells = grid.number_of_cells;

```

```
    int num_phases = 2;
40  using namespace Opm::unit;
    using namespace Opm::prefix;
42  std::vector<double> density(num_phases, denswater);
    density[1] = densoil;
44  double visc_arr[] = {muw*centi*Poise, muo*centi*Poise};
    std::vector<double> viscosity(visc_arr, visc_arr + sizeof(
      visc_arr)/sizeof(double));
46  double porosity = 0.5;
    SaturationPropsBasic::RelPermFunc rel_perm_func =
      SaturationPropsBasic::Quadratic;
48
    IncompPropertiesBasic props(num_phases, rel_perm_func, density
      , viscosity, porosity, 1*milli*darcy, grid.dimensions,
      num_cells);
50  IncompPropertiesShadow shadow_props(props);

52  const double grav_arr [] = {0.0,0.0,0.0};
    const double *grav = &grav_arr[0];
54  solve_gravity_column = false;
    std::vector<double> omega;
56
    std::vector<double> porevol;
58  Opm::computePorevolume(grid, props.porosity(), porevol);

60  double injectedFluidAbsolute = srcVol;
    double injectedFluidPoreVol = injectedFluidAbsolute/porevol
      [0];
62  std::vector<double> src(num_cells, 0.0);
    src[0] = injectedFluidPoreVol;
64  src[num_cells-1] = -injectedFluidPoreVol;

66  FlowBCManager bcs;

68  LinearSolverUmfpack linsolver;
    IncompPropertiesInterface * prop_pointer;
70  prop_pointer = (IncompPropertiesInterface *) &shadow_props.
      usePermeability(&perm[0]);

72  IncompTpfa psolver(grid, *prop_pointer, linsolver, grav, NULL,
       src, bcs.c_bcs());

74  WellState well_state;

76  const double tolerance = tol;
    const int max_iterations = 50;
78  Opm::TransportSolverTwophaseReorder transport_solver(grid, *
      prop_pointer, grav, tolerance, max_iterations, solver_type,
      verbose, useInitialGuessApproximation, printFluxValues);
```

```
80    const double comp_length = comp_length_days*day;
      const double dt = time_step_days*day;
82    const int num_time_steps = comp_length/dt;

84    TwophaseState state;
      state.init(grid, 2);
86
      std::vector<int> allcells(num_cells);
88    for (int cell = 0; cell < num_cells; ++cell) {
        allcells[cell] = cell;
90    }
      state.setFirstSat(allcells, *prop_pointer, TwophaseState::
       MinSat);
92
      std::vector<int>::iterator it = print_points.begin();
94
      for (int i = 0; i < num_time_steps; ++i) {
96      psolver.solve(dt, state, well_state);
        transport_solver.solve(&porevol[0], &src[0], dt, state);
98    }
}
100 catch (const std::exception &e) {
      std::cerr << "Program threw an exception: " << e.what() << "\n
       ";
102   throw;
}
```

Listing A.2: The C++ program used to run the inhomgeneous numerical tests in Sections 3.2.2 and 3.2.3.

```
(...) // Includes are omitted for brevity
int main (int argc, char ** argv)
try
{
  int nx = 20, ny = 20, nz = 1,xpos = 0, ypos = 0, zpos = 0;
  const int NPRINT = 100;
  int nprint = NPRINT;
  double xpos_double = 0.0, ypos_double = 0.0;
  double dx = 10.0, dy = 10.0, dz = 10.0;
  double muw = 1, muo = 1;
  double time_step_days = 0.1, comp_length_days = 2;
  double srcVol = 0.2, sinkVol = -srcVol;
  double grav_x = 0,grav_y = 0,grav_z = 0;
  double tol = 1e-9;
  bool verbose = false, printIterations = false,
    solve_gravity_column = false;
  Opm::RootFinderType solver_type = Opm::RegulaFalsiType;
  string perm_file_name = "spe_perm.dat";
  string print_points_file_name = "print_points.dat";
  string execName = boost::filesystem::path(std::string(argv[0])
    ).stem().string();

  using namespace Opm;

  double ddummy; bool bdummy;
  if(argc > 1)
  parseArguments(argc, argv, muw, muo, verbose, time_step_days,
    comp_length_days, dx, dy, dz, nx, ny, nz, solver_type,
    printIterations, nprint, print_points_file_name,
    perm_file_name, zpos, xpos_double, ypos_double, ddummy,bdummy
    ,srcVol, sinkVol, grav_x, grav_y, grav_z, tol, bdummy, bdummy
    );
  xpos = (int)xpos_double;
  ypos = (int)ypos_double;

  std::vector<double> perm;
  buildPermData(perm_file_name,perm,xpos,nx,ypos,ny,zpos,nz,
    verbose);

  GridManager grid_manager(nx, ny, nz, dx, dy, dz);
  const UnstructuredGrid& grid = *grid_manager.c_grid();
  int num_cells = grid.number_of_cells;

  int num_phases = 2;
  using namespace Opm::unit;
  using namespace Opm::prefix;
  std::vector<double> density(num_phases, 1000.0);
  density[1] = 800.0;
```

```
41  double visc_arr[] = {muw*centi*Poise, muo*centi*Poise};
    std::vector<double> viscosity(visc_arr, visc_arr + sizeof(
      visc_arr)/sizeof(double));
43  double porosity = 0.5;
    SaturationPropsBasic::RelPermFunc rel_perm_func =
      SaturationPropsBasic::Quadratic;
45
    IncompPropertiesBasic props(num_phases, rel_perm_func, density
      , viscosity,porosity, 1*milli*darcy, grid.dimensions,
      num_cells);
47  IncompPropertiesShadow shadow_props(props);

49  const double grav_arr [] = {grav_x, grav_y, grav_z};
    const double *grav = &grav_arr[0];
51  solve_gravity_column = ( fabs(density[1]-density[0]) > 0.0 )
     && ( fabs(grav_x)+fabs(grav_y)+fabs(grav_z) > 0.0 );
    std::vector<double> omega;
53
    double injectedFluidAbsolute = srcVol;
55  double poreVolume = dz*dx*dy*porosity/(nx*ny*nz);
    double injectedFluidPoreVol = injectedFluidAbsolute/poreVolume
      ;
57  std::vector<double> src(num_cells, 0.0);
    src[0] = injectedFluidPoreVol;
59  src[num_cells-1] = -injectedFluidPoreVol;

61  FlowBCManager bcs;

63  LinearSolverUmfpack linsolver;
    IncompPropertiesInterface * prop_pointer;
65  prop_pointer = (IncompPropertiesInterface *) &shadow_props.
      usePermeability(&perm[0]);
    IncompTpfa psolver(grid, *prop_pointer, linsolver, grav, NULL,
       src, bcs.c_bcs());
67
    WellState well_state;
69
    std::vector<double> porevol;
71  Opm::computePorevolume(grid, props.porosity(), porevol);

73  const double tolerance = tol;
    const int max_iterations = 50;
75  Opm::TransportSolverTwophaseReorder transport_solver(grid, *
      prop_pointer, grav, tolerance, max_iterations, solver_type,
      verbose);

77  const double comp_length = comp_length_days*day;
    const double dt = time_step_days*day;
79  const int num_time_steps = comp_length/dt;
```

```
81    TwophaseState state; state.init(grid, 2);

83    std::vector<int> allcells(num_cells);
      for (int cell = 0; cell < num_cells; ++cell) {
85      allcells[cell] = cell;
      }
87    state.setFirstSat(allcells, *prop_pointer, TwophaseState::
       MinSat);

89    std::vector<int>::iterator it = print_points.begin();
      for (int i = 0; i < num_time_steps; ++i) {
91      psolver.solve(dt, state, well_state);
        transport_solver.solve(&porevol[0], &src[0], dt, state);
93      if(solve_gravity_column) transport_solver.solveGravity(&
       porevol[0], dt, state);
      }
95  }
  catch (const std::exception &e) {
97    std::cerr<<"Program threw an exception: "<<e.what()<<"\n";
      throw;
99  }
```

Listing A.3: The C++ program used to run the 3D numerical tests in Section 3.2.4.

# Bibliography


Aarnes, J. E., Gimse, T., and Lie, K.-A. (2007). An introduction to the numerics of flow in porous media using matlab. *Geometrical Modeling, Numerical Simulation, and Optimization: Industrial Mathematics at SINTEF*, pages 265–306.

Aziz, K. and Settari, A. (1979). *Petroleium Reservoir Simulation*. Applied Science Publishers, Essex, UK.

Bear, J. (1972). *Dynamics of Fluids in Porous Media*. Dover Publications.

Binmore, K. G. (1977). *Mathematical Analysis: A Straightforward Approach*. Cambridge University Press, Cambridge, England.

Brent, R. (1973). *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, NJ.

Buckley, S. and Leverett, M. (1942). Mechanism of fluid displacement in sands. *Transactions of the AIME*, 146(01):107–116.

Chavent, G. and Jaffre, J. (1982). *Mathematical Models and Finite Elements for Reservoir Simulation*, volume 17 of *Studies in Mathematics and Its Applications*. North Holland.

Coats, K. (2000). A note on impes and some impes-based simulation models. *SPE Journal*, 5(03):245–251.

Courant, R., Friedrichs, K., and Lewy, H. (1928). Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100(1):32–74.

Dake, L. P. (1978). *Fundamentals of Reservoir Engineering*, volume 8 of *Developments in Petroleum Science*. Elsevier, The Hague, Netherlands, 17 edition.

Darcy, H. (1856). *Les Fontaines Publiques de la Ville de Dijon*. Dalmont, Paris.

Dekker, T. (1969). Finding a zero by means of successive linear interpolation. In Dejon, B. and Henrici, P., editors, *Constructive Aspects of the Fundamental Theorem of Algebra*, pages 37–48. Wiley-Interscience.

Dowell, M. and Jarratt, P. (1972). The Pegasus method for computing the root of an equation. *BIT Numerical Mathematics*, 12(4):503–508.

Eymard, R., Gallouët, T., and Herbin, R. (2003). *Finite Volume Methods*, volume 7 of *Handbook of Numerical Analysis*. North Holland.

Fagin, R. (1966). A new approach to the two-dimensional multiphase reservoir simulator. *SPE Journal*, 6(02):175–182.

Green, D. and Willhite, G. (2003). *Enhanced Oil Recovery*, volume 6 of *SPE Textbook Series*. Society of Petroleum Engineers, Richardson, Texas, USA.

Jain, S. (2013). Ch. 9 - hydrological property of rocks. In *Fundamentals of Physical Geology*, pages 215–218. Springer India, New Delhi, Delhi, India.

Jenny, P., Tchelepi, H. A., and Lee, S. (2009). Unconditionally convergent nonlinear solver for hyperbolic conservation laws with s-shaped flux functions. *Journal of Computational Physics*, 228(20):7497–7512.

Kincaid, D. and Cheney, W. (2002). Ch. 3 - solution of nonlinear equations. In *Numerical Analysis - Mathematics of Scientific Computing*, pages 74–138. Brooks Cole, Pacific Grove, California, 3 edition.

Kwok, F. and Tchelepi, H. A. (2007). Potential-based reduced newton algortihm for nonlinear multiphase flow in porous media. *Journal of Computational Physics*, 227:706–727.

Lie, K.-A. and Mallison, B. T. (2013). Mathematical models for oil reservoir simulation. *Encyclopedia of Applied and Computational Mathematics*.

Lie, K.-A., Nilsen, H. M., Rasmussen, A. F., and Raynaud, X. (2013). Fast simulation of polymer injection in heavy-oil reservoirs based on topological sorting and sequential splitting. *SPE Journal*.

MacDonald, R. (1970). Methods for numerical simulation of water and gas coning. *SPE Journal*, 10(04):425–436.

Molenaar, J. (1995). Multigrid methods for fully implicit oil reservoir simulation. Proceedings Copper Mountain Conference on Multigrid Methods.

Morris, J. (1983). *Computational Methods in Elementary Numerical Analysis*. Wiley, New York.

Natvig, J. R. and Lie, K.-A. (2008). Fast computation of multiphase flow in porous media by implicit discontinuous galerkin schemes with optimal ordering of elements. *Journal of Computational Physics*, 227(24):10108–10124.

Natvig, J. R., Lie, K.-A., and Eikemo, B. (2006). Fast solvers for flow in porous media based on discontinuous galerkin methods and optimal reordering. Copenhagen, Denmark. Eds., P.J. Binning et al.

Norsk Regnesentral (2003). SAIGUP Study.

OPM (2014). The open porous media initiative.

Patankar, S. V. (1980). *Numerical Heat Transfer and Fluid Flow*. Series in Computational Methods in Mechanics and Thermal Sciences. McGraw-Hill.

Ridders, C. (1979). A new algorithm for computing a single root of a real continuous function. *IEEE Transactions on Circuits and Systems*, 26(11):979–980.

SPE10 (2000). SPE comparative solution project.

Spillette, A., Hillestad, J., and Stone, H. (1973). A high-stability sequential solution approach to reservoir simulation. In *Fall Meeting of the Society of Petroleum Engineers of AIME*, Dallas, Texas, USA. Society of Petroleum Engineers, Society of Petroleum Engineers.

Tzimas, E., Georgakaki, A., Garcia, C., and S.D., P. (2005). Enhanced oil recovery using carbon dioxide in the european energy system. *European Comission - Joint Research Centre*.

Wang, X. and Tchelepi, H. A. (2013). Trust-region based solver for nonlinear transport in heterogeneous porous media. *Journal of Computational Physics*, 253:114–137.

Weber, H.-J. and Arfekn, G. B. (2003). *Essential Mathematical Methods for Physicists*. Academic Press, San Diego, USA.

Younis, R. M. (2011). *Modern Advances in Software and Solution Algorithms for Reservoir Simulation*. PhD thesis, Stanford University, Stanford, CA.

Younis, R. M., Tchelepi, H. A., and Aziz, K. (2010). Adaptively localized continuation-newton; reservoir simulation nonlinear solvers that converge all the time. *SPE Journal*, 15(2):526–544.