Nonlinear Multigrid for Efficient Reservoir Simulation

Master's Thesis

Supervisor: Allan P. Engsig-Karup

External supervisor: Mark Wakefield

August 31st, 2012

Max la Cour Christensen

Klaus Langgren Eskildsen

Department of Informatics and Mathematical Modelling

Technical University of Denmark

IMM-M.Sc-2012-103

Technical University of Denmark
Department of Informatics and Mathematical Modelling
Asmussens Alle, building 305, 2800 Kongens Lyngby, Denmark
Phone: +45 45253351, Fax: +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Preface

Together with a supplementary source code booklet, this report constitutes our M.Sc. thesis. The work has been carried out in a collaboration between DTU Informatics at the Technical University of Denmark and Schlumberger at Abingdon Technology Centre in Oxfordshire, England. Three months have been spent at Schlumberger's main center for reservoir simulation software development in Abingdon and three months have been spent at DTU Informatics in Denmark. The work presented in the thesis was carried out from February 2012 to August 2012.

First of all, we would like to thank our supervisor Allan P. Engsig-Karup for always taking the time to answer our questions and for his high level of involvement in the project. Also, we would like to thank him for allowing us the freedom to pursue our own ideas.

During our three months in Abingdon, we were under the excellent guidance of our external supervisor Mark Wakefield. We would like to thank Mark for giving us the opportunity of collaborating with the industry and for investing so much time in the project.

A special thanks to Morten Rode Kristensen for his many valuable inputs and for answering a great deal of questions.

Finally, we wish to thank our families and friends for their patience and continuous support.

Technical University of Denmark,
Kongens Lyngby,
August, 2012

Max la Cour Christensen (s072245)   Klaus Langgren Eskildsen (s072272)

Summary

The subject of this thesis is a thorough investigation of the application of nonlinear multigrid techniques, specifically the Full Approximation Scheme (FAS), for simulation of subsurface multiphase porous media flow. The main motivation for addressing this topic is a need for higher resolution and efficient simulations leading to better decision making in the production of oil and gas. Higher resolution simulations require efficient utilization of many-core parallel architectures. Current numerical methods employed in industrial reservoir simulators are memory intensive and not readily scalable on large-scale distributed systems and modern many-core architectures such as GPUs or Intel MICs.

In a first step, we investigate alternative numerical methods to establish algorithmic performance in serial computations. The nonlinear multigrid technique FAS uses local linearization, which allows for local components suitable for parallel implementation. Furthermore, FAS is a memory lean algorithm. To our knowledge, very little work is published on FAS for reservoir simulation. Molenaar, [36], considers the application of FAS on a simple 2D immiscible two-phase no gravity homogeneous example. To our knowledge, FAS has not been applied successfully to more complicated heterogeneous reservoir problems.

Two reservoir simulators have been implemented in C++ in serial. The first simulator is based on conventional techniques with global linearization in Newton's method and state-of-the-art choice of methods for the linear solver. The second simulator is based on the nonlinear multigrid method FAS. Both simulators solve the same system of PDEs governing 3D three-phase flow of oil, water and gas in a subsurface porous medium taking into account gravitational effects. The same discretization techniques are used for both simulators. For spatial discretization, the Finite Volume Method is used and for temporal integration, the backward Euler method is used. This enables fair comparisons between the conventional methods and FAS.

The two reservoir simulators have been tested extensively to compare the nonlinear multigrid approach FAS with the conventional techniques applied in modern reservoir simulation. It has been demonstrated that, without loss of robustness, FAS outperforms the conventional techniques in terms of algorithmic and numerical efficiency for the model equations considered. Furthermore, memory comparisons have been carried out, which show that FAS provides a significant memory reduction in comparison with conventional techniques. This memory reduction is an attractive feature, which enables higher resolution simulation for the beforementioned modern many-core architectures.

Resumé på dansk

Emnet for denne afhandling er et grundigt studie i anvendelsen af ikke-lineære multigrid teknikker, herunder Full Approximation Scheme (FAS) til simulering af undergrunds flerfase strømninger i et porøst medium. Den primære motivation for at undersøge dette emne er et behov for effektiv simulering i højere opløsning, hvilket vil muliggøre bedre beslutningstagning i produktionen af olie og gas. Simulering i højere opløsning kræver effektiv anvendelse af parallelle beregnings-arkitekturer med mange kerner. De nuværende numeriske metoder, som bliver anvendt i industriel reservoir simulering, har et højt hukommelsesforbrug og er ikke umiddelbare skalerbare på stor-skala distribuerede systemer og moderne mange-kerne arkitekturer såsom grafikkort eller Intel MICs.

Som et første skridt undersøger vi alternative numeriske metoder for at fastslå den algoritmiske ydelse ved almindelige sekventielle beregninger. Den ikke-lineære multigrid teknik FAS er baseret på lokal linearisering, hvilket tillader brugen af lokale komponenter, som er velegnet til parallel implementering. Yderligere er FAS en algoritme, der ikke bruger meget hukommelse. Så vidt vi ved, er der kun publiceret meget lidt om FAS for reservoir simulering. Molenaar, [36], betragter anvendelsen af FAS på et simpelt 2D ikke blandbart to-fase homogent eksempel uden hensyntagen til tyngdekraften. Så vidt vi ved, er FAS ikke blevet anvendt med succes på heterogene reservoir problemer.

To reservoir simulatorer er blevet implementeret i C++ i sekventiel. Den første simulator er baseret på konventionelle teknikker med global linearisering i Newton's metode samt nyeste metodevalg i den lineære løser. Den anden simulator er baseret på ikke-lineær multigrid metoden FAS. Begge simulatorer løser det samme system af partielle differentialligninger, hvilket beskriver 3D tre-fase strømning af olie, vand og gas i et porøst medium i undergrunden under hensyntagen til tyngdekraften. De samme diskretiseringsteknikker er brugt for begge simulatorer. Finite Volume metoden er anvendt til rumlig diskretisering og backward Euler metoden er anvendt til tidslig diskretisering. Dette sikrer et fair sammenligningsgrundlag mellem de konventionelle metoder og FAS.

De to reservoir simulatorer er blevet grundigt testet for at sammenligne den ikke-lineære multigrid tilgang med de konventionelle metoder anvendt i reservoir simulering. Det er blevet vist, at uden tab af robusthed udkonkurrerer FAS de konventionelle teknikker både med hensyn til algoritmisk og numerisk effektivitet for de anvendte modelligninger. Yderligere er der blevet foretaget sammenligninger af hukommelsesforbrug, der viser, at brugen af FAS giver et signifikant mindre hukommelsesforbrug i sammenligning med de konventionelle teknikker. Det lavere hukommelsesforbrug er en attraktiv egenskab, som tillader simuleringer med højere opløsning på de førnævnte moderne mange-kerne arkitekturer.

Contents

CHAPTER 1

Introduction

The purpose of this introductory chapter is to motivate the work presented in this thesis and state the objectives of the research. The motivation is founded by a literature review providing an overview of the current status of strategies employed in reservoir simulation.

Section 1.1 motivates the need for considering alternative methods in reservoir simulation. This is followed by a literature review in section 1.2. A short introduction to modern hardware is given in section 1.3. In section 1.4, the objectives of the thesis are listed. The novel contributions are described in section 1.5. A thesis outline is given in section 1.6.

Both authors have contributed equally to all of the work presented in this thesis.

1.1 Motivation

The world energy outlook from the International Energy Agency estimates that due to rising incomes and population, the global energy demand increases by one-third from 2010 to 2035 [7, International Energy Agency]. The passenger vehicle fleet alone is estimated to double to 1.7 billion in 2035, driving the oil demand to new heights. In Denmark, oil production in 2010 in the North Sea experienced a 6 percent decline compared to 2009. This reflects the trend of oil production in Denmark, which has continued with a downwards rate of 3-9 percent per year since 2005 [14, Danish Energy Agency, p. 19]. Figure 1.1 shows the oil and gas production in Denmark since the first oil field, Dan, started producing in 1972.

This decline is partly due to aging fields and is also a problem on a global basis. The most easily accessible oil has already been produced. To keep up with current demands

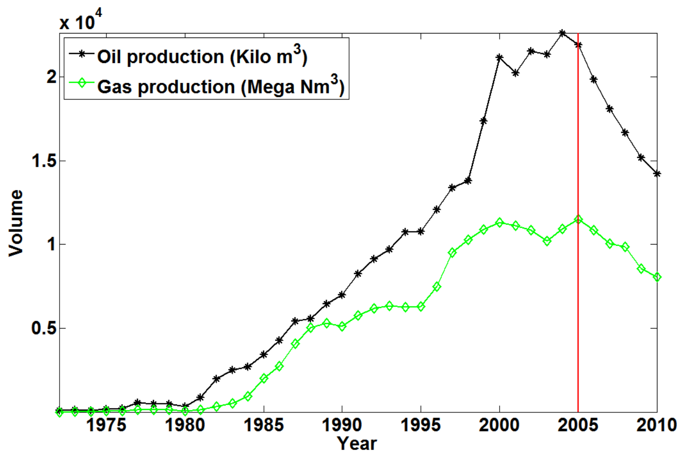


Figure 1.1: *Oil and gas production in Denmark. Numbers are from the Danish Energy Agency, www. ens. dk.*

and to possibly increase production to meet the future requirements, new and better oil recovery techniques are a necessity.

New enhanced oil recovery techniques are applied more and more to both existing and new fields. Oil prices have been steadily increasing and with recent social and political events occuring in several Middle Eastern and North African economies, oil prices are driven even higher, [44, World Energy Outlook 2011, p. 1]. Sustained high oil prices allow enhanced oil recovery techniques to become economically competitive [44, World Energy Outlook 2011, p. 25].

These enhanced oil recovery techniques, such as gas injection (CO_2, natural gas, nitrogen), chemical injection (surfactants, polymers, alkaline, low-salinity) and thermal methods (in-situ combustion, hot water injection, steam injection), all require elaborate mathematical simulations to optimize the oil recovery. Decisions such as placements of new wells and injection patterns are very dependent on reliable and quick simulation strategies.

Many oil and gas reservoirs have large amounts of seismic, geological and dynamic reservoir data available. This vast amount of data provides high resolution geological models. However, for conventional reservoir simulators to run in practical times, upscaling of these high resolution geological models from data points at a density of e.g. 25-50 meters to 250 meters is required. Simulators using upscaled reservoir properties often fail to accurately predict oil recovery [16, 17, 18, Dogru]. To fully utilize the seismic data and capture the flow of components in the reservoirs accurately, simulators must accommodate giga-cell scale models.

Actual field case experiences described in [16, Dogru] show how an increase in model resolution from 53,000 cells to 1.4 million cells revealed trapped oil. Based on these results, a decision was made to change the location of a new well. Consequently, a new horizontal well was drilled and oil was found as predicted by the simulator. Based on the simulator results, four new wells were drilled in 2003 and 2004 in similar locations to produce trapped oil. All wells found oil and are still producing today.

For large reservoirs, more geological data exists than can be used in the mathematical models. With more and more new technologies being implemented recently, such as deep-well electro-magnetic surveys, new borehole gravimetric surveys, new seismic methods, use of geochemistry and new sensor technology, even middle-sized reservoirs will have giga-cell geological models describing heterogeneity more accurately, [17, Dogru].

As demonstrated in [16, Dogru], even increasing model resolution without more geological data available is beneficial for accurately simulating advancing water fronts and water breakthrough at wells. Experiments in [16, Dogru] describe how a coarse model miscalculates water arrival by a few years, whereas the fine model agrees with the observations of the well. Evidently, increasing resolution of the mathematical models is beneficial both when additional geological data exists and when it does not.

Current industrial simulation tools have advanced to running mega-cell scale models on parallel hardware in practical times. These tools are often based on conventional simulation techniques predating the parallel hardware that has become a necessary part of scientific computing. As a consequence, industrial simulation tools often cannot utilize the performance capacity in modern parallel architectures. For giga-cell models to become a reality, new methods with more parallelizable algorithms for modern and emerging architectures are needed.

1.2 Literature review

In this literature review, we give a brief overview of the methods used in relation to our subject. We review both the conventional methods, which we aim at comparing against and the work already done on the nonlinear multigrid method FAS for reservoir simulation.

A petroleum reservoir is a porous medium containing hydrocarbons. The purpose of reservoir simulation is to predict future performance of a reservoir in order to find ways of optimizing the recovery of those hydrocarbons. Fluid flow in reservoirs is normally described mathematically with a system of PDEs governing subsurface porous media flow, [6, Aziz] and [12, Chen]. The system of PDEs cannot be solved analytically and hence must be solved with numerical techniques. Different numerical techniques, e.g. finite difference, finite volume or finite element methods, can be applied to solve the system of PDEs. In [12, Chen] the finite element method is employed, whereas [6, Aziz] describes the use of the finite difference and finite volume methods for reservoir simulation.

Common for all of the methods is a discretization of the continuous system of equations to a discrete system of equations. The discretization yields a set of equations whose solution approximates the solution of the continuous system of equations at discrete points or volumes in the reservoir. The system of equations is highly nonlinear and stiff. An approximation to the solution of the nonlinear system of equations can be found with an iterative method.

With conventional techniques, it is common to use a global linearization in a Newton-type method to solve the strongly nonlinear system of equations arising from the spatial and temporal discretization of the governing system of PDEs, [6, Aziz, p. 48]. The global linearization in e.g. Newton's method results in very large linear systems, which means the linear solver component often constitutes more than 70% of the computation time in reservoir simulators. Iterative linear solvers depend on effective preconditioners, which can be hard to parallelize to the extent required by many-core simulations [38, Saad, p.393]. Additionally, the memory requirement to store the sparse Jacobian for the linear systems is significant. To our knowledge, very little research has been published on matrix-free methods for reservoir simulation. The complexity of designing an effective matrix-free preconditioner might be the limiting factor.

The linear solver methods employed nowadays to solve the linear system in each newton iteration are iterative methods. These include the combination of ORTHOMIN with nested factorization as preconditioner used by the established commercial simulator ECLIPSE from Schlumberger, [19, Durlofsky, p. 40]. Wallis introduced the Constrained Pressure Residual preconditioning (CPR) in [45, 46]. CPR preconditioning is developed specifically for reservoir simulation and targets the individual elliptic and hyperbolic parts of the system of equations effectively in two stages. The first stage of the preconditioner deals with the pressure system and resolves global coupling and low frequency errors. As a consequence, the second stage only needs to deal with the remaining high frequency errors, which can be dealt with effectively by well-known Incomplete LU factorizations, [11, Cao].

In [11, Cao] it is demonstrated how a simulator with a linear solver based on CPR preconditioning using Algebraic Multigrid (AMG) to solve the first stage pressure system outperforms ECLIPSE. AMG is a black-box type of multigrid method that works without knowledge of the underlying partial differential equation. It is based solely on the information provided in the coefficients of the linear system. For more information on AMG, see [43, Stüben, p. 413]. The work presented in [11, Cao] is the basis of the linear solver in the next generation reservoir simulator INTERSECT[1]. INTERSECT is the result of combined research and development from Schlumberger and Chevron since 2000. It was released in 2009 and recently Total joined the collaboration for further development.

As [11, Cao] and [22, Fung] describe, a linear solver based on CPR-AMG preconditioning is extremely effective in terms of algorithmic efficiency (convergence rate). However, there are still challenges to overcome in implementing a near-ideal scalable AMG solver. Also the second stage of CPR preconditioning often includes some variant of Incomplete

[1)] www.slb.com/intersect

LU factorization, which again is hard to parallelize. As a result, the linear solver is still some way from near-ideal scalability.

In [22, Fung], a preconditioning method is described which is coined Line Solve Power Series (LSPS). It is a generalization of the z-line Neumann series method used to approximate inverses. With this approach, they demonstrate linear strong scalability as both single-stage preconditioner and two-stage CPR (LSPS as both pressure solve and full system solve in CPR) preconditioner. Despite the better numerical efficiency and scalability of CPR-LSPS compared to CPR-AMG, they also demonstrate that algorithmic efficiency is still better with CPR-AMG. This implies that if AMG could be implemented in a near-ideal scalable fashion, the CPR-AMG combination would be a very powerful strategy.

Much research in reservoir simulation has been done on optimizing the methods used for solving the linear systems resulting from global linearization in e.g. Newton's method. An alternative approach is to deal directly with the nonlinear system at hand. This is possible with nonlinear multigrid. In [28, Henson] and [43, Trottenberg], overviews of the two possible variations of nonlinear multigrid are given. One is based on global linearization in Newton's method and linear multigrid, which means it does not deal differently with the nonlinear system compared to any of the beforementioned methods. The second variation is called the Full Approximation Scheme (FAS). FAS is a nonlinear solver method, where the linearization happens on a local basis and therefore the method becomes very interesting with modern many-core architectures. Modern many-core architectures such as Graphics Processing Units (GPUs) or Intel Many Integrated Core (MIC) can obtain very high floprates at relatively low cost given the methods are "local enough" (parallelizable). Furthermore, the bandwidth of GPUs is high, [31, Keckler], which is beneficial for memory bound applications.

Interestingly, very little work has been published on the use of FAS for reservoir simulation. Molenaar, [36], demonstrates convergence rates of the two variations of nonlinear multigrid on a simple 2D immiscible two-phase no gravity homogeneous example. In this work, it is found that nonlinear multigrid provides fast, grid independent convergence behaviour and optimal complexity, meaning the time needed per time step per grid point is independent of the number of grid points.

1.3 Modern many-core architectures

Due to high power consumption and heat issues, a bottleneck has been reached for increasing clock frequencies in modern processors. To accomodate the growing need for more computational power, multi-core processors were introduced and are now the standard in both personal laptops and in clusters at universities and industry. The well-established multi-core systems have complex scheduling procedures and memory pipelining, which enable highly optimized execution on each core, [31, Keckler].

Meanwhile, NVIDIA and ATI grew big on selling graphics cards for computer games. This has resulted in a spin-off into scientific computing. With the support of double-precision arithmetics and more user-friendly programming paradigms for GPUs, the scientific computing field has widely adopted these PCI-accelerators as a means to speed up computations. However, many existing and well-functioning algorithms on CPUs need changing to give real world applications any significant speedups. This in turn means that legacy codes cannot be adjusted with simple means to achieve near-optimal performance. The big difference between CPUs and GPUs is that GPU chips have significantly less physical space allocated with various control units. Instead, more physical space is dedicated to a larger number of cores, the so-called Single Instruction Multiple Data (SIMD) units, [31, Keckler].

With GPUs disrupting the architectural foundation of scientific computing, Intel is now responding with the launch of MICs (Many Integrated Core), specifically the Knights Corner. Similarly to GPUs, the Knights Corner is a PCI-accelerator. It has more than 50 cores per chip and is based on a more traditional approach with Multiple Instruction Multiple Data (MIMD) and vector design. Intel's intention with MICs is to allow an easier transition from legacy codes to massively parallel architectures by enabling the use of known parallel languages[2].

Since many legacy codes have not been written for modern many-core systems, a simple recompilation to the MIC architecture is unlikely to provide good utilization of the performance capabilities. Common for both GPUs and MICs is the need to revisit "old" algorithms and evaluate and likely change these algorithms if they do not have sufficient parallel capabilities. It is no longer sufficient to only consider pure serial algorithmic performance without considering if a given algorithm is parallelizable and preferably massively parallelizable.

Currently, many-core architectures are attached through the PCI. The trend seems to go in the direction of more hybrid architectures. Many scientific and engineering applications are memory bound, meaning they depend on high-bandwidth access to memory. Architecturally this means that memory has to be placed close to the processing units which complicates and increases cost for extending the capacity of memory spaces. This suggests more memory lean algorithms would be preferable on emerging many-core architectures. Prototypes of hybrid memory cubes have proven to be promising and perhaps these will play a big role in the next generation many-core architures, [23, Farber].

No matter what the architectural evolution might be, a physical limitation in terms of power consumption and heat has been reached for the transistor based chips. The current way forward is many-core architectures and this approach will require adapting algorithms implemented in most applications.

2) www.intel.com

1.4 Objectives

The main objective of this thesis is to investigate feasibility of numerical methods for advanced reservoir simulation. We propose the use of nonlinear multigrid techniques to solve the equations governing the flow of components in reservoirs. These techniques have proven highly efficient on modern many-core architectures in other fields, [21, Engsig-Karup]. Specifically, we propose the Full Approximation Scheme (FAS).

Components of the nonlinear multigrid solver are local and therefore appropriate for efficient and scalable implementation on modern, many-core architectures such as GPUs or Intel MICs. Furthermore, by using FAS we avoid having to assemble the Jacobian on the finest grid, which results in major memory savings.

We have found that only little work has been published on this topic. We want to investigate if it is possible to apply FAS to more complicated and realistic reservoir models than described by [36, Molenaar]. If FAS can be successfully applied, it should have superior algorithmic abilities in the sense that optimal complexity is attainable, namely that the number of arithmetic operations needed to solve a problem is proportional to the number N of unknowns in the problem considered [43, Trottenberg p. 20]. Furthermore, FAS has memory requirements that are lower than conventional techniques which may provide a basis for larger simulations and more efficient simulators.

The objectives of this work are

- Develop and implement a sequential 3D immiscible three-phase reservoir simulator based on FAS in C++.
- Study its effectiveness and robustness under various heterogeneous cases.
- Compare with current reservoir simulation strategies based on global linearization, e.g. Newton's method. This will be accomplished through implementation of such a simulator in C++.

This work will contribute to and advance the knowledge of FAS applied to reservoir simulation.

1.5 Novel contributions

The novel contributions of this work are the application and study of nonlinear multigrid techniques, specifically FAS, on mathematically challenging reservoir model equations. These challenges include high nonlinearity and heterogeneity. To our knowledge, similar studies do not appear in any published litterature.

Furthermore, comparisons are carried out of a FAS based reservoir simulator with a reservoir simulator based on global linearization in Newton's method. The global linearization technique used for comparison is implemented with state-of-the-art choice of methods for the linear solver, also applied in commercial simulators. We demonstrate improved algorithmic and numerical efficiency using FAS for the given model equations and problems considered.

Lastly, the work in this thesis contribute to the knowledge of nonlinear multigrid techniques applied to complicated engineering problems. Specifcally, we recommend that further studies in the application of FAS based techniques for reservoir simulations are considered.

The work presented in this thesis has been selected for presentation at three conferences. Appendix G contains the abstracts submitted and a poster presented at one of the conferences.

1.6 Thesis outline

Chapter 2 presents the differential equations governing immiscible multiphase flow of fluids in a porous medium, which are the equations considered throughout this thesis. The spatial and temporal discretization of these equations are thoroughly described in chapter 3.

The methods used to solve this discretized system of differential equations are presented in chapter 4. The result is a simulator based on global linearization with Newton's method. The chapter covers theory, implementation, verification and a small performance study.

Chapter 5 is dedicated to the introduction of multigrid methods, starting with a simple linear multigrid method and moving on to the nonlinear Full Approximation Scheme method. A simulator based on this nonlinear method is implemented, verified and tested.

The implementation of the FAS simulator presented in chapter 5 is a working prototype. Some of the strategies applied in the simulator leave room for improvement. Chapter 6 suggests alternative strategies and studies whether or not these alterations provide improvements. The chapter follows the actual work process, meaning that one alternative at the time is proposed, implemented and evaluated. If it provides an improvement, it is used as the standard from that point onwards.

After the final modifications at the end of chapter 6, an extensive performance study of the two simulators is carried out in chapter 7. Finally, chapter 8 presents considerations for a parallel implementation, which however, is not carried out due to time constraints. Chapter 9 concludes the work done in the thesis.

Appendix A specifies the input parameters kept fixed for all the tests carried out in the thesis, and Appendix B lists the hardware used for these test. The properties describing the discretized reservoir are introduced in Appendix C, and the derivation of the derivatives of the Jacobian matrix for the system of differential equations is presented in Appendix D. Some of the test cases used in the thesis consider smooth heterogeneous permeability fields. Appendix E describes how to generate such a permeability field. Appendix F contains additional verification plots for the FAS simulator using more grid levels.

Finally, Appendix G lists the conferences, for which we have been selected to give a presentation based on the work done in this thesis.

Chapter 2

Model equations

Following the approach described in [12, Chen, p. 10] we derive the equation governing single-phase flow of fluids in a porous medium. Specifically, the equation is given by the conservation of mass and momentum. The momentum is governed by Darcy's law, which was first derived empirically in 1856 [15, Darcy]. Darcy's law describes a linear relationship between the fluid velocity and the pressure gradient. Later, the extension to the multiphase flow model, which forms the basis for this thesis, is presented.

We assume that the mass flux due to diffusion, meaning the mass flux caused by differences in concentrations is negligible compared to mass flux due to advection and that the fluid cannot go through solid material. Based on these assumptions, we derive the differential form of the governing equation using a differential volume.

Before deriving the model equations we establish a convention for notation used throughout this thesis. The following notation applies

$$a : \textit{scalar} \qquad \mathbf{a} : \textit{vector} \qquad \mathbf{A} : \textit{matrix}$$

Consider the rectangular cuboid in Figure 2.1 with volume $\Delta x \Delta y \Delta z$.

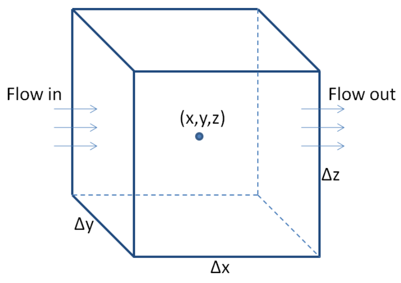


Figure 2.1: *Differential volume.*

We denote in the following the porosity of the porous medium: ϕ (the fraction of volume available for the fluid taking into account rock compressibility), the density of the fluid per unit volume: ρ, the Darcy velocity: $\mathbf{v} = (v_x, v_y, v_z)$ and the external sources/sinks (wells): w. The porosity is described in more detail in Appendix C.7.

The rectangular cuboid in Figure 2.1 has faces parallel to the coordinate axes and its center is given by (x, y, z). We only consider flow perpendicular to the faces. Now the mass flux in the x-direction is the mass inflow at the surface at $x - \frac{\Delta x}{2}$ and the mass outflow at the surface at $x + \frac{\Delta x}{2}$. These mass flows are given by

$$\text{In}: \quad (\rho v_x)_{x-\frac{\Delta x}{2},y,z}\Delta y \Delta z, \qquad \text{Out}: \quad (\rho v_x)_{x+\frac{\Delta x}{2},y,z}\Delta y \Delta z, \tag{2.1}$$

where ρv is the mass flow per unit area per unit time. Similarly, these mass flows in the y- and z-directions are given by:

$$\text{In}: \quad (\rho v_y)_{x,y-\frac{\Delta y}{2},z}\Delta x \Delta z, \qquad \text{Out}: \quad (\rho v_y)_{x,y+\frac{\Delta y}{2},z}\Delta x \Delta z, \tag{2.2}$$

in the y-direction and

$$\text{In}: \quad (\rho v_z)_{x,y,z-\frac{\Delta z}{2}}\Delta x \Delta y, \qquad \text{Out}: \quad (\rho v_z)_{x,y,z+\frac{\Delta z}{2}}\Delta x \Delta y, \tag{2.3}$$

in the z-direction.

Having defined the mass flows, we continue with mass accumulation due to compressibility. Mass accumulation per unit time is

$$\frac{\partial(\phi\rho)}{\partial t}\Delta x \Delta y \Delta z \tag{2.4}$$

and the removal (or addition) of mass in the cuboid from external sinks/sources with strength w (mass per unit volume per unit time) is

$$-w\Delta x \Delta y \Delta z \tag{2.5}$$

Using that the difference between mass inflow and mass outflow equals the sum of mass accumulation within the cuboid, the following equation holds:

$$\begin{aligned}
&\left[(\rho v_x)_{x-\frac{\Delta x}{2},y,z} - (\rho v_x)_{x+\frac{\Delta x}{2},y,z}\right]\Delta y \Delta z \\
+&\left[(\rho v_y)_{x,y-\frac{\Delta y}{2},z} - (\rho v_y)_{x,y+\frac{\Delta y}{2},z}\right]\Delta x \Delta z \\
+&\left[(\rho v_z)_{x,y,z-\frac{\Delta z}{2}} - (\rho v_z)_{x,y,z+\frac{\Delta z}{2}}\right]\Delta x \Delta y \\
=&\left[\frac{\partial(\phi\rho)}{\partial t} - w\right]\Delta x \Delta y \Delta z
\end{aligned} \tag{2.6}$$

Dividing by $\Delta x \Delta y \Delta z$ yields

$$
\begin{aligned}
&-\frac{(\rho v_x)_{x+\frac{\Delta x}{2},y,z} - (\rho v_x)_{x-\frac{\Delta x}{2},y,z}}{\Delta x} \\
&-\frac{(\rho v_y)_{x,y+\frac{\Delta y}{2},z} - (\rho v_y)_{x,y-\frac{\Delta y}{2},z}}{\Delta y} \\
&-\frac{(\rho v_z)_{x,y,z+\frac{\Delta z}{2}} - (\rho v_z)_{x,y,z-\frac{\Delta z}{2}}}{\Delta z} \\
&=\frac{\partial(\phi\rho)}{\partial t} - w
\end{aligned}
\tag{2.7}
$$

and letting $\Delta x, \Delta y, \Delta z \to 0$, the differential form of the mass conservation equation is obtained:

$$\frac{\partial(\phi\rho)}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = w, \tag{2.8}$$

where the divergence operator is

$$\nabla \cdot \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \tag{2.9}$$

2.1 Immiscible multiphase flows

In multiphase models we operate with different components and phases. The convention here is that the subscript c denotes a component and the superscript α denotes a phase.

The derivation of the equations governing multiphase flows is similar to the description above. The only difference is the accumulation term, where saturations (or molar densities) of individual phases are included. Using this approach, the differential form of the mass conservation for each component can be formulated as a conservation law

$$\frac{\partial(\phi m_c)}{\partial t} + \nabla \cdot \mathbf{f}^\alpha = \sigma_c, \tag{2.10}$$

where m_c is the component molar density, σ_c is the source term and $\mathbf{f}^\alpha = b^\alpha \mathbf{v}^\alpha$ is the flux with b^α being the phase molar density and α denoting the phase, [42, Trangenstein]. The distinction between component molar densities and phase molar densities can be confusing. The phase molar density is a function of pressure as described in Appendix C.2, whereas the component molar density is considered a primary variable.

Note that since the components are immiscible, phases essentially equal components. In the oil business, a system of immiscible fluids with oil, water and gas is called dead oil, dry gas, meaning oil does not vaporize into the gas phase and gas does not dissolve in the oil phase. For more complicated fluid models, the oil component can vaporize into the gas phase and the gas component can dissolve in the oil phase.

A system consisting of 3 components/phases (oil, water and gas) is then modelled with the following system of differential equations:

$$\begin{aligned} \frac{\partial(\phi m_o)}{\partial t} + \nabla \cdot \mathbf{f}^o = \sigma_o \\ \frac{\partial(\phi m_g)}{\partial t} + \nabla \cdot \mathbf{f}^g = \sigma_g \\ \frac{\partial(\phi m_w)}{\partial t} + \nabla \cdot \mathbf{f}^w = \sigma_w, \end{aligned} \tag{2.11}$$

The component molar densities m_o, m_g, m_w and the pressure p are the primary variables of the system. We do not consider the effects of capillary pressure. This is a simplification.

The phase velocities are given by Darcy's law:

$$\mathbf{v}^\alpha = -\mathbf{K}\frac{k_r^\alpha}{\mu^\alpha}(\nabla p - \rho^\alpha g \nabla z), \tag{2.12}$$

where $\mathbf{K}$ is the absolute permeability, μ^α is the viscosity, k_r^α is the relative permeability, ρ^α is the density and g is the gravitational acceleration. The phase velocities in equation (2.12) take into account the coordinate system in Figure 2.2, which we are using throughout this thesis.

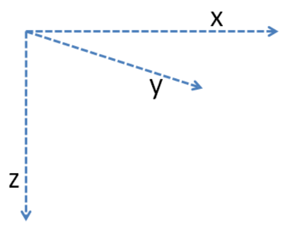


Figure 2.2: *Coordinate system*

The absolute permeability $\mathbf{K}$ describes the capacity of the porous medium to transport the fluids through its interconnected pores. It can vary over the domain in both the x-,y- and z-direction. It can be defined as the following diagonal matrix.

$$\mathbf{K} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix} \tag{2.13}$$

The diagonal matrix in (2.13) is for anisotropic permeability fields when $k_x \neq k_y \neq k_z$ as opposed to isotropic permeability fields when $k_x = k_y = k_z$. For simplicity we assume isotropic permeability fields.

The viscosity μ^α of a fluid is a measure of its resistance to flow. See Appendix C.4 for details. The relative permeability k_r^α describes how the different fluids flow in the presence of each other. It is a dimensionless term devised to adapt Darcy's law (2.12) to multiphase flow conditions. See Appendix C.6 for details. The mass density ρ^α of a material is defined as its mass per unit volume. See Appendix C.5 for details.

Volume balance constraint

The system is closed with a "volume balance" type constraint that is constant. For this we use the saturation constraint.

$$\sum_\alpha S^\alpha = 1, \tag{2.14}$$

Initial and boundary conditions

The initial molar densities are given by initial saturations as described in Appendix C.3.

The boundary conditions are no flow conditions, meaning

$$\mathbf{n} \cdot \mathbf{f}^\alpha(t, x, y, z) = 0 \quad (x, y, z) \in \partial\Omega, \tag{2.15}$$

where t is time, the spatial domain Ω represents the whole reservoir, $\partial\Omega$ is the boundary of the domain and $\mathbf{n}$ is an outward pointing normal vector. No flow conditions correspond to Neumann type conditions.

Chapter 3

Discretization

With the model equations being introduced we are ready to discretize the system of differential equations in (2.11). Instead of discretizing each of these three differential equations we consider discretization of the general form, (2.10), in space and time.

The method of lines is used to solve the system of partial differential equations. Spatial discretization is done using the Finite Volume Method (FVM), which is explained in section 3.1. The actual discretization is carried out in section 3.2. Temporal discretization is performed with a backward Euler scheme as described in section 3.3.

3.1 The Finite Volume Method

For spatial discretization we use the FVM, since this method is commonly used when considering time-dependent conservation laws.

The following convention is used throughout this thesis. Subscripts of the type i and j refer to an index position in a 3-dimensional grid, whereas the subscript ij refers to something being evaluated across the interface between cell i and cell j. The superscripts n and $n+1$ are used for indicating time steps.

Let the domain of the considered 3-dimensional oil reservior be denoted $\Omega \in \mathbb{R}^3$. In the FVM the domain Ω is represented by N non-overlapping grid cells or control volumes. Let the ith grid cell be denoted Ω_i, $i \in \mathcal{C} = \{1, ..., N\}$. The domain Ω can then be expressed as

$$\Omega = \bigcup_{i \in \mathcal{C}} \Omega_i \tag{3.1}$$

The volume of Ω_i is denoted V_i and the surface of Ω_i, which is denoted $\partial\Omega_i$, has the total area A_i.

The ith grid cell Ω_i is centered around the point $(x, y, x)_{(i)} = (x_i, y_i, z_i)$ as illustrated in Figure 3.1.

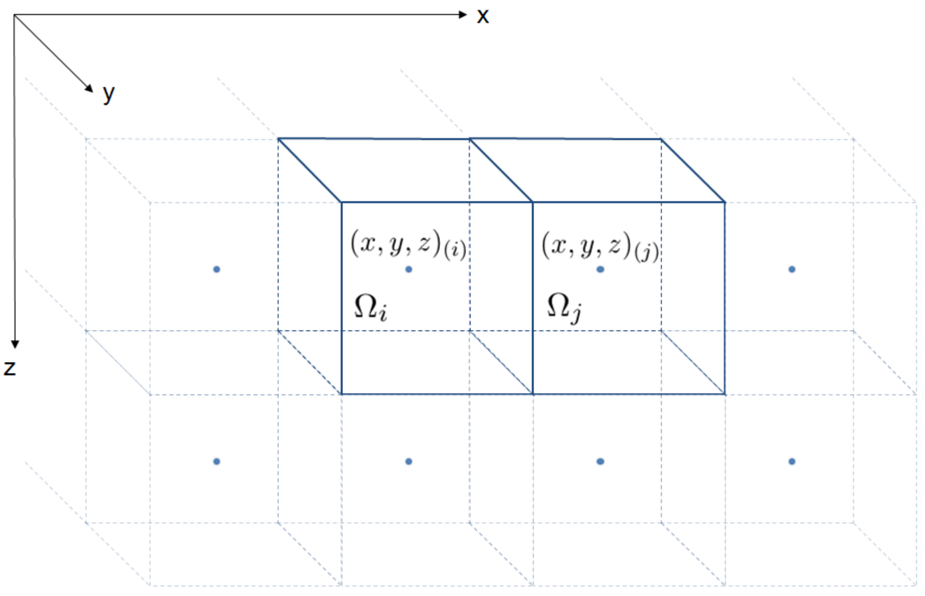


Figure 3.1: *The grid cells Ω_i and Ω_j in the discretized reservior.*

This means that the volume of the ith grid cell Ω_i is defined as

$$V_i = \Delta x_i \Delta y_i \Delta z_i, \quad i \in \mathcal{C} \tag{3.2}$$

where $\Delta x_i, \Delta y_i, \Delta z_i$ are the lengths of grid cell i in each of the three dimensions, respectively. In this thesis, we consider only regularly structured grids, meaning that two grid cells Ω_i and Ω_j, $i, j \in \mathcal{C}$, $i \neq j$ have the same dimensions, i.e.

$$\Delta x_i = \Delta x_j, \quad \Delta y_i = \Delta y_j, \quad \Delta z_i = \Delta z_j, \quad V_i = V_j \quad \forall\, i, j \in \mathcal{C},\ i \neq j \tag{3.3}$$

Thus we skip the index and use Δx, Δy, Δz and V. Note that Δx, Δy, Δz and V are positive values. Furthermore, due to (3.3) we know that two cells have the same total area, i.e. $A_i = A_j$, $i, j \in \mathcal{C}$, $i \neq j$.

The approach in the FVM is to approximate the solution to our problem locally over each grid cell. This is done using cell averages, which according to [33, LeVeque, p.98] are defined as

$$u_i = \frac{1}{V} \int_{\Omega_i} u \, dV, \quad i \in \mathcal{C} \tag{3.4}$$

Each grid cell can have up to 6 neighbouring cells. The cells located at the boundary of the domain has less. Let the set $\mathcal{N}^{(i)} \subset \mathcal{C}$ denote the set of indices for the neighbouring cells to cell Ω_i. We can then define the interface between two adjacent cells as

$$\tau_{ij} = \Omega_i \cap \Omega_j, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)} \tag{3.5}$$

The subscript ij means that τ_{ij} is derived from information from the ith cell Ω_i and one of its neighbouring cells Ω_j. This notation will be used throughout the thesis for various calculations made for interfaces.

3.2 Spatial discretization

Consider the conservation law in differential form in (2.10)

$$\frac{\partial(\phi m_c)}{\partial t} + \nabla \cdot \mathbf{f}^\alpha = \sigma_c. \tag{3.6}$$

By integrating locally over the ith grid cell and dividing by the volume Ω_i we rewrite equation (3.6) into

$$\frac{\partial}{\partial t}\frac{1}{V}\int_{\Omega_i} \phi m_{c,i}\, dV = -\frac{1}{V}\int_{\Omega_i} \nabla \cdot \mathbf{f}_i^\alpha\, dV + \frac{1}{V}\int_{\Omega_i} \sigma_{c,i}\, dV, \quad i \in \mathcal{C} \tag{3.7}$$

where the subscript c, i refers to component c at cell i. Using the definition of a cell average (3.4) gives us

$$\frac{\partial(\phi m_{c,i})}{\partial t} = -\frac{1}{V}\int_{\Omega_i} \nabla \cdot \mathbf{f}_i^\alpha\, dV + \sigma_{c,i}, \quad i \in \mathcal{C} \tag{3.8}$$

Instead of explicitly considering the porosity ϕ in the accumulation term, the convention in the oil-business is to use the pore volume $V_p = \phi V$. Thus we multiply (3.8) with the volume of the ith grid cell leading to

$$\frac{\partial(V_p m_{c,i})}{\partial t} = -\int_{\Omega_i} \nabla \cdot \mathbf{f}_i^\alpha\, dV + s_{c,i}, \quad i \in \mathcal{C}, \tag{3.9}$$

where $s_{c,i} = \sigma_{c,i} V$ describes the average value of the source over the control volume.

By using the Divergence Theorem we can evaluate the flux term over the surface area of the grid cell

$$\frac{\partial(V_p m_{c,i})}{\partial t} = -\int_{\partial\Omega_i} \mathbf{n}_i \cdot \mathbf{f}_i^\alpha\, dA + s_{c,i}, \quad i \in \mathcal{C}, \tag{3.10}$$

where A is the total surface area of Ω_i and $\mathbf{n}_i$ is the outward pointing normal field of $\partial\Omega_i$.

Since the considered grid is regularly structured, the normal vectors, which are outward pointing unit vectors to each cell in question, are defined as shown in Figure 3.2 below.

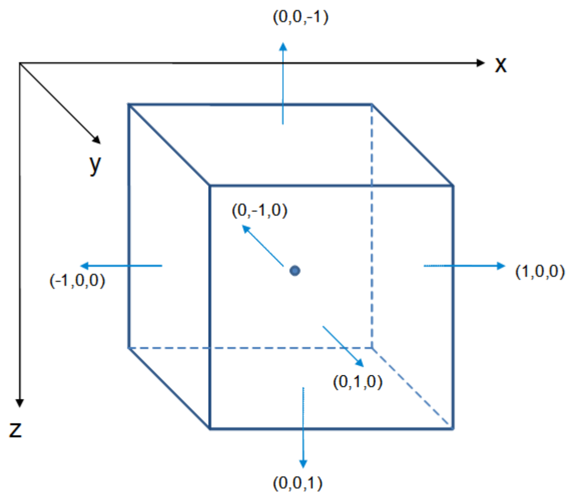


Figure 3.2: *The normal vectors for a grid cell.*

We now look at the flux over a single interface of the ith grid cell and sum over all the interfaces τ_{ij}. This leads to

$$\frac{\partial(V_p m_{c,i})}{\partial t} = -\sum_{j\in\mathcal{N}(i)} \int_{\tau_{ij}} \mathbf{n}_{ij} \cdot \mathbf{f}^{\alpha}_{ij}\, dA_{ij} + s_{c,i}, \quad i \in \mathcal{C}, \tag{3.11}$$

where $\mathbf{f}^{\alpha}_{ij}$ is the flux across interface τ_{ij}, A_{ij} is the surface area of interface τ_{ij}, and $\mathbf{n}_{ij}$ is the normal vector for interface τ_{ij}. This normal vector points outward as illustrated in Figure 3.2.

By applying the midpoint rule to the surface integrals in (3.11), this equation can be rewritten into

$$\frac{\partial(V_p m_{c,i})}{\partial t} = -\sum_{j\in\mathcal{N}(i)} \mathbf{n}_{ij} \cdot \mathbf{f}^{\alpha}_{ij} A_{ij} + s_{c,i}, \quad i \in \mathcal{C} \tag{3.12}$$

We do not know the flux across interface τ_{ij}. Thus, we have to reconstruct it.

Flux reconstruction

In chapter 2 we defined the flux function as

$$\mathbf{f}^{\alpha} = b^{\alpha}\mathbf{v}^{\alpha} = -b^{\alpha}\mathbf{K}\frac{k_r^{\alpha}}{\mu^{\alpha}}(\nabla p - \rho^{\alpha} g \nabla z), \tag{3.13}$$

We collect all the properties dependent on either the pressure p or the mass m_c in a single scalar called the mobility, which is denoted λ. As seen in the dependencies table C.1 in Appendix C the mobility therefore consists of b^{α}, k_r^{α} and μ^{α}, meaning

$$\lambda^{\alpha} = \frac{b^{\alpha} k_r^{\alpha}}{\mu^{\alpha}} \tag{3.14}$$

Note that λ^α is positive since b^α, k_r^α and μ^α are all positive scalar values. The absolute permeability term $\mathbf{K}$ is static, since it does not depend on p or m_c. All static terms are later collected in a single scalar value T, called the transmissibility.

Inserting (3.14) into (3.13) yields

$$\mathbf{f}^\alpha = -\mathbf{K}\lambda^\alpha(\nabla p - \rho^\alpha g \nabla z), \tag{3.15}$$

In (3.12) we consider this flux across interface τ_{ij}, meaning that we have to determine the normal flux of each face

$$\mathbf{n}_{ij} \cdot \mathbf{f}_{ij}^\alpha = -\mathbf{n}_{ij} \cdot \left(\mathbf{K}\lambda^\alpha(\nabla p - \rho^\alpha g \nabla z)\right)_{ij}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)} \tag{3.16}$$

By using the definition of $\mathbf{K}$ in (2.13) and that the mobility λ_{ij} is a scalar we can rewrite (3.16) such that

$$\mathbf{n}_{ij} \cdot \mathbf{f}_{ij}^\alpha = -k_{ij}\lambda_{ij}^\alpha \mathbf{n}_{ij} \cdot (\nabla p - \rho^\alpha g \nabla z)_{ij}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)}, \tag{3.17}$$

where the definition of k_{ij} for regularly structured grids is

$$k_{ij} = \begin{cases} \dfrac{2k_{x,i}k_{x,j}}{k_{x,i} + k_{x,j}} & \text{if} \quad \tau_{ij} \perp x\text{-direction} \\ \dfrac{2k_{y,i}k_{y,j}}{k_{y,i} + k_{y,j}} & \text{if} \quad \tau_{ij} \perp y\text{-direction} \\ \dfrac{2k_{z,i}k_{z,j}}{k_{z,i} + k_{z,j}} & \text{if} \quad \tau_{ij} \perp z\text{-direction} \end{cases} \tag{3.18}$$

The notation in (3.18) uses the assumption that the flow is parallel to either the x-, y- or z-direction, such that τ_{ij} is perpendicular to the flow direction.

Due to stability reasons, we reconstruct the flux across the interface τ_{ij} using an upwind method. Without upwinding, the numerical solution may display oscillations, overshoots or undershoots (e.g., saturations less than zero or greater than one), or converge to an incorrect solution, [6, Aziz, p.163]. An upwind flux is a type of flux, where information is obtained by looking in the directions from which we expect this information to come, [34, LeVeque, p.72].

The upwind idea is illustrated by the example in Figure 3.3.

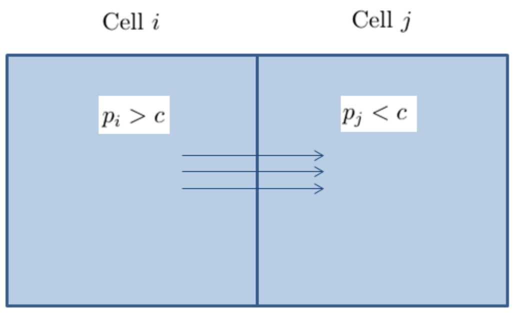


Figure 3.3: *Upwind illustration with two cells i and j.*

The example considers the cells i and j, which are filled with the same fluid material. However, the pressure in cell i is higher than the pressure in cell j, and therefore material is moving from cell i, called the upwind cell, to cell j.

The idea is that, depending on the flow direction, we reconstruct the flux across the interface using information from the upwind cell. In this case this information is the mobility λ^α.

Direction of flow

In (3.17) it is not pressure alone, but also the normal vector $\mathbf{n}_{ij}$ and the bracket in the phase velocity $\mathbf{v}^\alpha$ that determines the direction of the flow. For interface τ_{ij} this is

$$\mathbf{n}_{ij} \cdot (\nabla p - \rho^\alpha g \nabla z)_{ij} = \mathbf{n}_{ij} \cdot \nabla p_{ij} - \rho^\alpha_{ij} g \mathbf{n}_{ij} \cdot \nabla z_{ij}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)} \tag{3.19}$$

The derivative of the pressure at the interface is approximated locally using a finite difference scheme

$$\mathbf{n}_{ij} \cdot \nabla p_{ij} = \mathbf{n}_{ij} \cdot \begin{bmatrix} \dfrac{\partial p_{ij}}{\partial x} \\ \dfrac{\partial p_{ij}}{\partial y} \\ \dfrac{\partial p_{ij}}{\partial z} \end{bmatrix} \approx \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_{ij} \cdot \begin{bmatrix} \dfrac{p_j - p_i}{x_j - x_i} \\ \dfrac{p_j - p_i}{y_j - y_i} \\ \dfrac{p_j - p_i}{z_j - z_i} \end{bmatrix}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)}, \tag{3.20}$$

where p_i is the pressure of the ith grid cell.

In the rightmost vector in (3.20), the denominators of the fractions are the differences between the coordinates for the cell center of cell i and the cell center of cell j in either the x-, y- or z-direction. This difference can be either positive or negative depending on the location of j.

Since $\mathbf{n}_{ij}$ is the outward pointing normal vector of interface τ_{ij} only one of the elements in $\mathbf{n}_{ij}$ is nonzero, yielding

$$\mathbf{n}_{ij} \cdot \nabla p_{ij} \approx n_{ij} \left(\frac{p_j - p_i}{h_j - h_i} \right), \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)}, \tag{3.21}$$

where n_{ij} is either 1 or -1 depending on the interface considered, and $h_j - h_i$ is defined as

$$h_j - h_i = n_{ij}|h_j - h_i| = n_{ij}\Delta h = \begin{cases} n_{ij}|x_j - x_i| = n_{ij}\Delta x, & \text{if} \quad \tau_{ij} \perp x\text{-direction} \\ n_{ij}|y_j - y_i| = n_{ij}\Delta y, & \text{if} \quad \tau_{ij} \perp y\text{-direction} \\ n_{ij}|z_j - z_i| = n_{ij}\Delta z, & \text{if} \quad \tau_{ij} \perp z\text{-direction} \end{cases}, \tag{3.22}$$

for $i \in \mathcal{C}$ and $j \in \mathcal{N}^{(i)}$. Using (3.22) we rewrite (3.21) into

$$\mathbf{n}_{ij} \cdot \nabla p_{ij} \approx n_{ij} \left(\frac{p_j - p_i}{n_{ij} \Delta h} \right) = \frac{\Delta p_{ij}}{\Delta h}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)}, \tag{3.23}$$

where $\Delta h = |h_j - h_i|$ and $\Delta p_{ij} = p_j - p_i$.

The mass density ρ^{α}_{ij} for phase α is computed as a saturation weighted average between the mass densities at cell i and j, as explained in Appendix C.5. The reason for using this average density is that it ensures symmetry, such that it is the same regardless of whether the flow is going from cell i to cell j or from cell j to cell i.

The dot product between $\mathbf{n}_{ij}$ and the gradient ∇z_{ij} ensures that the gravitational part of (3.19) is only taken into consideration when the flow is along the z-direction. We approximate ∇z_{ij} using a finite difference approach similar to the approximation of the pressure gradient described above. The result is

$$\mathbf{n}_{ij} \cdot \nabla z_{ij} = \mathbf{n}_{ij} \cdot \begin{bmatrix} \dfrac{\partial z_{ij}}{\partial x} \\ \dfrac{\partial z_{ij}}{\partial y} \\ \dfrac{\partial z_{ij}}{\partial z} \end{bmatrix} \approx \mathbf{n}_{ij} \cdot \begin{bmatrix} \dfrac{z_j - z_i}{x_j - x_i} \\ \dfrac{z_j - z_i}{y_j - y_i} \\ \dfrac{z_j - z_i}{z_j - z_i} \end{bmatrix} = \frac{\Delta z_{ij}}{\Delta h}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)}, \tag{3.24}$$

where $\Delta z_{ij} = z_j - z_i$, and z_i is the z-coordinate of the cell center of the ith grid cell. Note that even though we consider a regularly structured grid, Δz_{ij} and Δz might be different since Δz_{ij} can be negative.

Furthermore, note that $\Delta z_{ij} = 0$ if the flow is along either the x- or y-direction, since in such cases $z_i = z_j$. This means that $\mathbf{n}_{ij} \cdot \nabla z_{ij}$ only gives a contribution if the flow is along the z-direction.

Inserting (3.23) and (3.24) in (3.19) yields

$$\mathbf{n}_{ij} \cdot (\nabla p - \rho^{\alpha} g \nabla z)_{ij} \approx \frac{1}{\Delta h} (\Delta p - \rho^{\alpha} g \Delta z)_{ij}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)} \tag{3.25}$$

We use the bracket on the right-hand side of (3.25) to determine the direction of the flow across interface τ_{ij}.

Mobility upwinding

Having determined the direction of the flow, we know the upwind cell. Using upwinding the mobility becomes

$$\lambda_{ij} = \begin{cases} \lambda_i, & \text{if} \quad (\Delta p - \rho^{\alpha} g \Delta z)_{ij} < 0 \\ \lambda_j & \text{otherwise} \end{cases}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)} \tag{3.26}$$

Flux over the entire cell

We can now derive an expression for the flux over the entire grid cell. Substituting (3.25) into (3.17) yields

$$\mathbf{n}_{ij} \cdot \mathbf{f}_{ij}^{\alpha} = -\frac{k_{ij}}{\Delta h} \lambda_{ij}^{\alpha} (\Delta p - \rho^{\alpha} g \Delta z)_{ij}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)} \tag{3.27}$$

Inserting (3.27) into the semidiscrete differential equation in (3.12) leads to

$$\begin{aligned} \frac{\partial (V_p m_{c,i})}{\partial t} &= - \sum_{j \in \mathcal{N}^{(i)}} \mathbf{n}_{ij} \cdot \mathbf{f}_{ij}^{\alpha} A_{ij} + s_{c,i} \\ &= - \sum_{j \in \mathcal{N}^{(i)}} -\frac{k_{ij}}{\Delta h} \lambda_{ij}^{\alpha} (\Delta p - \rho^{\alpha} g \Delta z)_{ij} A_{ij} + s_{c,i}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)} \end{aligned} \tag{3.28}$$

As earlier mentioned, we collect the properties dependent only on the location of the interface and the geometry of the grid cell. For this purpose we define the transmissibility at interface τ_{ij} as

$$T_{ij} = \frac{A_{ij} k_{ij}}{\Delta h}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)} \tag{3.29}$$

Using this definition, equation (3.28) can be written as the following semidiscrete system of differential equations

$$\begin{aligned} \frac{\partial (V_p m_{c,i})}{\partial t} &= \sum_{j \in \mathcal{N}^{(i)}} T_{ij} \lambda_{ij}^{\alpha} (\Delta p - \rho^{\alpha} g \Delta z)_{ij} + s_{c,i} \\ &= f_i^{\alpha} + s_{c,i}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)}, \end{aligned} \tag{3.30}$$

where the flux over an entire grid cell i is given by

$$f_i^{\alpha} = \sum_{j \in \mathcal{N}^{(i)}} T_{ij} \lambda_{ij}^{\alpha} (\Delta p - \rho^{\alpha} g \Delta z)_{ij}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)} \tag{3.31}$$

This completes the spatial discretization of the differential equations. The next step is temporal integration.

3.3 Temporal integration

The system of differential equations in (2.11) is considered a stiff system, since the system includes terms that can lead to rapid variation in the solution. The gas phase is considerably more mobile than the other phases due to differences in viscosities (cf. Table C.2 and Table C.3). Furthermore, stiffness arises due to differences in how pressure and saturations vary. Saturation changes are slow, whereas pressure changes spread more rapidly across the field. This will be illustrated in plots of the solution later.

Explicit methods are generally quite inefficient on stiff problems, since the time step has to be very small for the method to stay in its stability region. We therefore use an implicit method, namely the backward Euler method, as implicit methods are usually more stable. This means that larger step sizes can be considered, [35, LeVeque, p.170], and instead chosen based on accuracy requirements only.

The backward Euler method is used to march forward in time and it is based on the Taylor expansion of the function y around the value t

$$\begin{aligned} y(t-\Delta t) &= y(t) - y'(t)\Delta t + O(\Delta t^2) \Leftrightarrow \\ y'(t)\Delta t &= y(t) - y(t-\Delta t) + O(\Delta t^2), \end{aligned} \tag{3.32}$$

By truncating terms of quadratic and higher order we end up with the first order approximation to $y'(t)$

$$y'(t) \approx \frac{y(t) - y(t-\Delta t)}{\Delta t} \tag{3.33}$$

This means that if we want to approximate the differential equation

$$y'(t) = f(t, y(t)), \tag{3.34}$$

where $f(t, y(t))$ in general is some nonlinear function, we get

$$y'(t) \approx \frac{y(t) - y(t-\Delta t)}{\Delta t} = f(t, y(t)) \Leftrightarrow \tag{3.35}$$

$$y^{n+1} = y^n + \Delta t f(t^{n+1}, y^{n+1}). \tag{3.36}$$

This equation must be solved for y^{n+1}. In practice this is done by approximating the solution using some iterative method. One of the most popular approaches is to use Newton's method, also known as Newton-Raphsons method. Because the backward Euler method provides an equation that has to be solved for y^{n+1} it is called an implicit method, [35, p.120].

By using the backward Euler method on the semidiscrete system of equations in (3.30) we obtain

$$\frac{V_p m_{c,i}^{n+1} - V_p m_{c,i}^n}{\Delta t} = f_i^{\alpha,n+1} + s_{c,i}^{n+1}, \tag{3.37}$$

which can be rewritten into

$$V_p m_{c,i}^{n+1} = V_p m_{c,i}^n + \Delta t \left(f_i^{\alpha,n+1} + s_{c,i}^{n+1} \right) \tag{3.38}$$

This is the fully discretized system of mass conservation equations, which we are using throughout the thesis as the basis for multiphase reservoir simulation.

Chapter 4

Applying Newton's method

This chapter covers the theory necessary to develop our first multiphase reservoir simulator. One of the main objectives of this thesis is to compare a reservoir simulator based on global linearization with a reservoir simulator based on FAS. In order to do this fairly, we implement a simulator based on global linearization as well. This chapter describes all the considerations involved in implementing such a simulator. In fact, the simulator based on FAS will use all of the same routines and therefore it is a good starting point before considering the FAS method based on local linearization.

Section 4.1 introduces Newton's method in general terms for nonlinear system of equations and section 4.2 describes how we apply Newton's method to the reservoir model. This is followed by section 4.3, where the implementation is described in detail. Section 4.4 verifies our first simulator against the commercial simulator ECLIPSE. An important issue in simulators is time stepping. The three different time step selection methods implemented in section 4.3.6 are compared in section 4.5. Lastly, a small performance study is carried out in section 4.6.

4.1 Newton's method

As shown in section 3 choosing an implicit method such as backward Euler

$$y^{n+1} = y^n + \Delta t f(t^{n+1}, y^{n+1}) \tag{4.1}$$

leads to an equation (or in our case a system of equations) that needs to be solved to find the solution at the next time step y^{n+1}.

Newton's method is a root-finding method that successively approximates a root of a given function, [20, Eldén, p. 69]. Since it is a root-finding method, equation (4.1) has to be formulated correspondingly

$$\mathcal{F}(y^{n+1}) = y^n + \Delta t f(t^{n+1}, y^{n+1}) - y^{n+1} = 0 \tag{4.2}$$

Newton's method is derived from the Taylor series of $\mathcal{F}(x)$ around a point $x = x^{[0]} + h$

$$\mathcal{F}(x^{[0]} + h) = \mathcal{F}(x^{[0]}) + h\mathcal{F}'(x^{[0]}) + \dots, \tag{4.3}$$

where we neglect higher order terms and set $\mathcal{F}(x^{[0]} + h) = 0$, thus

$$\mathcal{F}(x^{[0]} + h) \approx \mathcal{F}(x^{[0]}) + h\mathcal{F}'(x^{[0]}) = 0 \tag{4.4}$$

This expression is then used to estimate the offset $h \equiv h^{[0]}$ needed to come closer to the root starting from an initial guess $x^{[0]}$

$$h^{[0]} = -\frac{\mathcal{F}(x^{[0]})}{\mathcal{F}'(x^{[0]})} \tag{4.5}$$

Letting $x^{[1]} = x^{[0]} + h^{[0]}$ and continuing the process of estimating offsets $h^{[k]}$, a root of a given function can iteratively be found by

$$x^{[k+1]} = x^{[k]} - \frac{\mathcal{F}(x^{[k]})}{\mathcal{F}'(x^{[k]})} \tag{4.6}$$

for $k = 0, 1, 2, ...$

It should be noted that if the starting guess $x^{[0]}$ is not sufficiently close to the exact solution, Newton's method may not converge.

Nonlinear system of equations

Newton's method readily extends to nonlinear system of equations, [35, Leveque, p. 38]. For 2 nonlinear equations with 2 unknowns we seek a better approximation $\mathbf{x}^{[k+1]} = \mathbf{x}^{[k]} + \mathbf{h}^{[k]}$. Using Taylor series of $\mathcal{F}(\mathbf{x})$ gives

$$\begin{aligned} \mathcal{F}_1(\mathbf{x}^{[k]} + \mathbf{h}) &= \mathcal{F}_1(\mathbf{x}^{[k]}) + \frac{\partial \mathcal{F}_1}{\partial x_1}(\mathbf{x}^{[k]})h_1 + \frac{\partial \mathcal{F}_1}{\partial x_2}(\mathbf{x}^{[k]})h_2 + \dots \\ \mathcal{F}_2(\mathbf{x}^{[k]} + \mathbf{h}) &= \mathcal{F}_2(\mathbf{x}^{[k]}) + \frac{\partial \mathcal{F}_2}{\partial x_1}(\mathbf{x}^{[k]})h_1 + \frac{\partial \mathcal{F}_2}{\partial x_2}(\mathbf{x}^{[k]})h_2 + \dots \end{aligned} \tag{4.7}$$

In matrix-vector notation this becomes

$$\begin{bmatrix} \mathcal{F}_1(\mathbf{x}^{[k]} + \mathbf{h}) \\ \mathcal{F}_2(\mathbf{x}^{[k]} + \mathbf{h}) \end{bmatrix} = \begin{bmatrix} \mathcal{F}_1(\mathbf{x}^{[k]}) \\ \mathcal{F}_2(\mathbf{x}^{[k]}) \end{bmatrix} + \begin{bmatrix} \frac{\partial \mathcal{F}_1}{\partial x_1}(\mathbf{x}^{[k]}) & \frac{\partial \mathcal{F}_1}{\partial x_2}(\mathbf{x}^{[k]}) \\ \frac{\partial \mathcal{F}_2}{\partial x_1}(\mathbf{x}^{[k]}) & \frac{\partial \mathcal{F}_2}{\partial x_2}(\mathbf{x}^{[k]}) \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + \dots, \tag{4.8}$$

The matrix above is the Jacobian matrix $\mathbf{J}$ leading to the following equation

$$\mathcal{F}(\mathbf{x}^{[k]}+\mathbf{h}) = \mathcal{F}(\mathbf{x}^{[k]}) + \mathbf{J}(\mathbf{x}^{[k]})\mathbf{h} + \dots \tag{4.9}$$

Neglecting higher order terms and setting $\mathcal{F}(\mathbf{x}^{[k]}+\mathbf{h}) = \mathbf{0}$ we obtain

$$\mathcal{F}(\mathbf{x}^{[k]}+\mathbf{h}) \approx \mathcal{F}(\mathbf{x}^{[k]}) + \mathbf{J}(\mathbf{x}^{[k]})\mathbf{h} = \mathbf{0}, \tag{4.10}$$

where the step $\mathbf{h}^{[k]}$ can be found by solving the following linear system of equations.

$$\mathbf{J}(\mathbf{x}^{[k]})\mathbf{h}^{[k]} = -\mathcal{F}(\mathbf{x}^{[k]}) \tag{4.11}$$

By adding the step to the previous solution: $\mathbf{x}^{[k+1]} = \mathbf{x}^{[k]} + \mathbf{h}^{[k]}$, and performing the same procedure for this newly found solution, the method iteratively finds better and better approximations to the solution of the nonlinear system of equations. However, as mentioned in the scalar case, Newton's method is not guaranteed to converge.

4.2 Application to model equations

For the discretized model equations presented in equation (3.38) and the saturation constraint in equation (2.14), we can write up the following residuals:

$$\begin{aligned}
r_{o,i}(m_o,p) &= \Delta t\,(s_{o,i}+f_i^o)^{n+1} + (V_p m_{o,i})^n - (V_p m_{o,i})^{n+1}\\
r_{g,i}(m_g,p) &= \Delta t\,(s_{g,i}+f_i^g)^{n+1} + (V_p m_{g,i})^n - (V_p m_{g,i})^{n+1}\\
r_{w,i}(m_w,p) &= \Delta t\,(s_{w,i}+f_i^w)^{n+1} + (V_p m_{w,i})^n - (V_p m_{w,i})^{n+1}\\
r_{vb,i}(m_o,m_g,m_w,p) &= S^o + S^g + S^w - 1,
\end{aligned} \tag{4.12}$$

where r_{vb} denote the residual of the volume balance constraint, which we in this work have chosen to be the saturation constraint.

This is a nonlinear system of equations, which can be solved using Newton's method. Following the approach in section 4.1, we formulate our nonlinear system of equations as a root-problem

$$\mathbf{r}(m_o,m_g,m_w,p) = \mathbf{r}(\mathbf{x}) = \mathbf{0} \tag{4.13}$$

and seek a better approximation to the root or solution

$$\mathbf{r}(\mathbf{x}+\mathbf{h}) \approx \mathbf{r} + \frac{\partial \mathbf{r}}{\partial \mathbf{x}}\mathbf{h} = \mathbf{r} + \mathbf{J}\mathbf{h} = \mathbf{0} \tag{4.14}$$

by solving for $\mathbf{h}$ in the linear system of equations $\mathbf{J}\mathbf{h} = -\mathbf{r}$, where $\mathbf{J}$ is the Jacobian for the nonlinear system of equations and $\mathbf{r}$ is the residuals.

For the choice of spatial discretization described in section 3.2, we have the 7-point stencil in Figure 4.1.

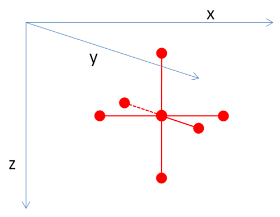


Figure 4.1: *Seven-point stencil.*

With a $3 \times 3 \times 3$ grid, where the 3 layers of the z-dimension are shown side by side, the indexing of the cells and a demonstration of how the seven-point stencil is applied to the middle point of the grid, is displayed in Figure 4.2.

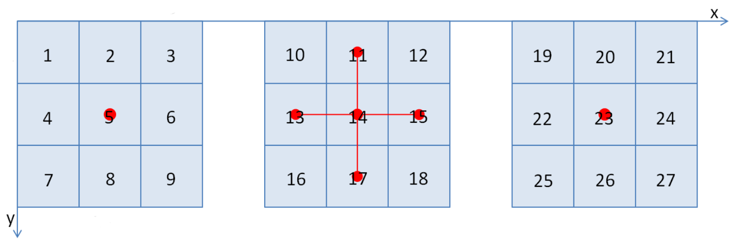


Figure 4.2: *Indexing of a $3 \times 3 \times 3$ grid. The seven-point stencil from Figure 4.1 is applied to the point in the middle of the cube.*

For this grid, indexing and stencil, the Jacobian will have the sparsity structure displayed in Table 4.1 below.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	x	x		x						x																	
2	x	x	x		x						x																
3		x	x			x						x															
4	x			x	x		x						x														
5		x		x	x	x		x						x													
6			x		x	x			x						x												
7				x			x	x								x											
8					x		x	x	x								x										
9						x		x	x									x									
10	x									x	x		x						x								
11		x								x	x	x		x						x							
12			x								x	x			x						x						
13				x						x			x	x		x						x					
14					x						x		x	x	x		x						x				
15						x						x		x	x			x						x			
16							x						x			x	x								x		
17								x						x		x	x	x								x	
18									x						x		x	x									x
19										x									x	x		x					
20											x								x	x	x		x				
21												x								x	x			x			
22													x						x			x	x		x		
23														x						x		x	x	x		x	
24															x						x		x	x			x
25																x						x			x	x	
26																	x						x		x	x	x
27																		x						x		x	x

Table 4.1: *Sparsity structure of the Jacobian.*

For the considered system of equations, each cross in Table 4.1 represents a 4×4 block matrix. On the diagonal, the block matrices are given by $\frac{\partial \mathbf{r_i}}{\partial \mathbf{x_i}}$, meaning the derivative of the residuals for cell i wrt. itself: cell i. This means, the block matrices include flow, accumulation and the volume balance constraint.

However, the block matrices on the off-diagonal are given by $\frac{\partial \mathbf{r_i}}{\partial \mathbf{x_j}}$, meaning the derivative of the residuals for cell i wrt. a neighbour cell j. This means that the block matrices only include flow as it is only the flow that have dependencies on neighbour cells. For more details on the calculation of the derivatives of the residuals, see Appendix D.

The diagonal and off-diagonal block matrices are given in equation (4.15).

$$\frac{\partial \mathbf{r}_i}{\partial \mathbf{x}_i} = \overbrace{\begin{bmatrix} \frac{\partial r_{o,i}}{\partial m_{o,i}} & 0 & 0 & \frac{\partial r_{o,i}}{\partial p_i} \\ 0 & \frac{\partial r_{g,i}}{\partial m_{g,i}} & 0 & \frac{\partial r_{g,i}}{\partial p_i} \\ 0 & 0 & \frac{\partial r_{w,i}}{\partial m_{w,i}} & \frac{\partial r_{w,i}}{\partial p_i} \\ \frac{\partial r_{vb,i}}{\partial m_{o,i}} & \frac{\partial r_{vb,i}}{\partial m_{g,i}} & \frac{\partial r_{vb,i}}{\partial m_{w,i}} & \frac{\partial r_{vb,i}}{\partial p_i} \end{bmatrix}}^{\text{Diagonal block matrix}}, \quad \frac{\partial \mathbf{r}_i}{\partial \mathbf{x}_j} = \overbrace{\begin{bmatrix} \frac{\partial r_{o,i}}{\partial m_{o,j}} & 0 & 0 & \frac{\partial r_{o,i}}{\partial p_j} \\ 0 & \frac{\partial r_{g,i}}{\partial m_{g,j}} & 0 & \frac{\partial r_{g,i}}{\partial p_j} \\ 0 & 0 & \frac{\partial r_{w,i}}{\partial m_{w,j}} & \frac{\partial r_{w,i}}{\partial p_j} \\ 0 & 0 & 0 & 0 \end{bmatrix}}^{\text{Off-diagonal block matrix}} \tag{4.15}$$

where the residual vector $\mathbf{r}$ and solution vector $\mathbf{x}$ corresponding to the Jacobian in Table 4.1 are given in (4.16).

$$\begin{aligned} \mathbf{r} &= \begin{bmatrix} r_{o,1} & r_{g,1} & r_{w,1} & r_{vb,1} & \dots & r_{o,27} & r_{g,27} & r_{w,27} & r_{vb,27} \end{bmatrix}^T \\ \mathbf{x} &= \begin{bmatrix} m_{o,1} & m_{g,1} & m_{w,1} & p_1 & \dots & m_{o,27} & m_{g,27} & m_{w,27} & p_{27} \end{bmatrix}^T \end{aligned} \tag{4.16}$$

4.3 Implementation

Having presented the basic theory of discretizing a continouos system of equations to a discrete nonlinear system of equations and the methods that are employed to solve these nonlinear systems, we have the components necessary to create our first implementation of a reservoir simulator. The most outer loop in the reservoir simulator controls the time integration. For each iteration, it calls Newton's method with an updated time step size Δt. Algorithm 4.1 describes this procedure.

Algorithm 4.1 Main loop - Input: $\mathbf{x}^0$, Δt^0

```
1: while time < finaltime do
2:     $\mathbf{x}^{n+1}$ = newton($\mathbf{x}^n$, $\Delta t$)
3:     Update time step size $\Delta t$ with some heuristic
4:     time = time + $\Delta t$
5:     $n = n + 1$
6: end while
```

To reach the requested final time, the time step size is chopped to $\Delta t = \text{finaltime} - \text{time}$ if $\text{time} + \Delta t > \text{finaltime}$. Newton's method is implemented in accordance with Algorithm 4.2. The value k_{max} is the maximum number of newton iterations allowed per time step.

Algorithm 4.2 Newton's method - $\mathbf{x}^{n+1} = \text{newton}(\mathbf{x}^n,\Delta t)$

```
1: while stop = false or $k < k_{\max}$ do
2:     Compute properties
3:     Compute residuals $\mathbf{r}$
4:     Set up Jacobian $\mathbf{J}$
5:     if stop criteria is met and $k > 0$ then
6:         stop = true
7:     end if
8:     Solve for $\mathbf{h}$ in $\mathbf{Jh} = -\mathbf{r}$
9:     Update solution: $\mathbf{x} = \mathbf{x} + \mathbf{h}$
10:    $k = k + 1$
11: end while
```

Notice that Newton's method is implemented such that we always do at least one newton iteration. This is done to ensure that the solution is updated in each time step. The individual steps of Newton's method is explained in the following.

4.3.1 Properties

Before computing the residual and corresponding Jacobian, we need to update the properties for each cell. Given current values for the primary variables: molar densities and pressure, we compute formation volume factors, viscosities, phase molar densities, mass densities, saturations, relative permeabilities and mobilities. These variables are all necessary to set up the residual and Jacobian for the linear solve. For more details on how to compute these properties, see Appendix C.

4.3.2 Residuals

The first three equations in our residual in equation (4.12) consist of a source term, a flow term and two accumulation terms. Equation (4.17) describes these. The source term is kept to zero in all tests, meaning no well models are considered. Well models in reservoir simulation can be implemented in many ways. See [6, Aziz, p. 220] for an introduction. For simplicity and to get a working reservoir simulator up and running as a first step, we have chosen not to include wells.

$$r_{c,i} = \Delta t \overbrace{s_i^{\alpha,n+1}}^{\text{Source}} + \Delta t \overbrace{f_i^{\alpha,n+1}}^{\text{Flow}} + \overbrace{(V_p m_c)_i^n}^{\text{Accumulation 1}} - \overbrace{(V_p m_c)_i^{n+1}}^{\text{Accumulation 2}} \tag{4.17}$$

The pore volume in the accumulation term $V_p = V\phi$ is computed as described in Appendix C.7. The flow is computed on an edge based basis. When initializing, we set up a map of the edges with their corresponding transmissibilities as displayed in Table 4.2.

Edge	**Cell** i	**Cell** j	**T**
1	1	2	$T_{1,2}$
2	1	4	$T_{1,4}$
3	1	10	$T_{1,10}$
4	2	3	$T_{2,3}$
5	2	5	$T_{2,5}$
6	2	11	$T_{2,11}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
54	26	27	$T_{26,27}$

Table 4.2: *Edge map associated with the* $3 \times 3 \times 3$ *grid in Figure 4.2.*

The transmissibilities are computed once in the initialization as described in equation (3.29). By looping over all edges and summing the contributions in the residual (same approach is used for the Jacobian) for the respective cells, we end up with the cell based residuals in (4.12) as needed. There are several advantages in using an edge based approach.

- Due to mass conservation, redundant computations are avoided as we can exploit the fact that the flow from cell i to cell j equals the negated flow from cell j to cell i: $f_{ij} = -f_{ji}$
- No-flow boundary conditions are treated by the absence of edges out of the domain from cells on the boundary.
- Fewer memory reads as a result of avoiding redundant computations.

The fourth equation in the residual is given by the volume balance equation in (4.18)

$$r_{vb} = S^o + S^g + S^w - 1 \tag{4.18}$$

The contribution to the residual from this equation along with the accumulation terms and the source term are computed on a cell based basis.

4.3.3 Jacobian

Similar to the flow in the residual, the derivatives of the flow part of the residual are computed by traversing the edge map. For the ith cell, it involves computing the block matrices in (4.15). The derivatives used to set up the Jacobian are all analytic expressions. A detailed derivation of the derivatives is given in Appendix D.

The Jacobian with the sparsity structure displayed in Table 4.1 contains $N \times N$ block matrices all of size 4×4. From (4.15), we can see that each diagonal block matrix has 10 non-zero entries and each off-diagonal block matrix has 6 non-zero entries. For a cubic reservoir of a typical $N = 10^6$ cells, this results in $N_{edges} = 3 \cdot N - 3 \cdot 100 = 2,999,700$ edges and $10 \cdot N + N_{edges} \cdot 2 \cdot 6 = 45,996,400$ non-zero entries in the Jacobian. For this size of reservoir, the fraction of non-zero entries compared to total elements in the Jacobian is roughly $2.9 \cdot 10^{-6}$. Evidently, a sparse linear solver is a necessity for efficient simulations.

The choice of sparse linear solver is discussed in the following section. In accordance with our choice of linear solver, we are using triplets or coordinate format (COO), namely

```
(row index, column index, value)
```

as sparse format to store the Jacobian.

When assembling the Jacobian on an edge based basis, all the entries in the diagonal block matrices need to be accessed multiple times in order to accumulate contributions from itself and all neighbour cells. Meanwhile, all the entries in the off-diagonal block matrices are only written to once. As a result, we employ a system, where the first $10N$ triplets are for entries in the diagonal block matrices and the remaining $N_{edges} \cdot 2 \cdot 6$ triplets are for entries in the off-diagonal block matrices. Figure 4.3 illustrates the system.

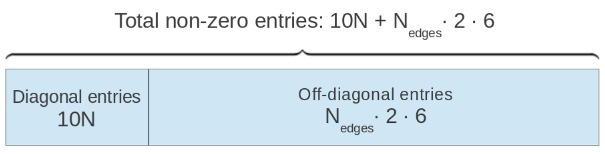

Total non-zero entries: 10N + N_edges · 2 · 6
Diagonal entries 10N
Off-diagonal entries N_edges · 2 · 6

Figure 4.3: *Sparse data structure for Jacobian.*

By employing this system, the entries for the diagonal block matrices are easily found, which makes it possible to accumulate the appropriate contributions. A counter is used to write the entries for the off-diagonal block matrices. Because the edge map is traversed from start to end in every iteration, it is only necessary to set up the row and column indices once for the Jacobian.

4.3.4 Linear solver

When solving a linear system of equations, there are two classes of methods: direct methods and iterative methods. In the absence of rounding errors, direct methods such as Gaussian elimination find an exact solution to a linear system of equations by a finite sequence of operations. For Gaussian elimination these operations include multiplying, adding and switching rows. In contrast, iterative methods use an initial guess of the solution to iteratively generate better and better approximations to the solution.

For large systems in the order of millions of grid cells, direct methods for sparse linear systems are insufficient and outperformed by iterative solution techniques. Implementing an efficient iterative or direct linear solver for reservoir simulations is a challenging task on its own, and as a result we have chosen to utilize an off-the-shelf linear solver. To increase certainty on retrieving reliable results for the highly complex linear systems arising from strong heterogeneous reservoir models, we are using a library that implements a direct method to solve the sparse linear system of equations. The library that we apply is MUMPS[1] (MUltifrontal Massively Parallel sparse direct Solver), [3, 4, Amestoy]. MUMPS is considered one of the best sparse direct linear solvers on the market [25, 27, 37]. It is easy to use and it has many options for improving performance by tweaking and using prior knowledge of the properties of the linear system of equations. The results given later in this chapter in section 4.4 and 4.6 are all for the sequential version of MUMPS that only need BLAS[2] (Basic Linear Algebra Subprograms) to run.

We realize that using a direct linear solver instead of an iterative linear solver prohibits simulating larger systems. However, the primary focus in this work is nonlinear multigrid, where the goal is to reduce the dependence on the linear solver component for the systems arising from global linearization from Newton's method. Furthermore, while developing it is good to know that errors do not come from an iterative linear solver that fails to converge properly.

4.3.5 Stop criteria

The stop criterion we use in Newton's method is kept simple. It computes the 2-norm of the residuals over the number of grid cells N and compares with a user-given tolerance τ. Equation (4.19) describes this mathematically.

$$\frac{||\mathbf{r}_o||_2}{N} < \tau_o \quad \text{AND} \quad \frac{||\mathbf{r}_g||_2}{N} < \tau_g \quad \text{AND} \quad \frac{||\mathbf{r}_w||_2}{N} < \tau_w \quad \text{AND} \quad \frac{||\mathbf{r}_{vb}||_2}{N} < \tau_{vb} \tag{4.19}$$

Since the values for the primary variables pressure and molar densities have different magnitudes, the tolerances τ have to be set accordingly. Typical values for the tolerances that give reliable and efficient simulations are $\tau_o = \tau_g = \tau_w = 10^{-4}$ and $\tau_{vb} = 10^{-6}$. Remember that pressure is linked to the volume balance constraint and as a consequence τ_{vb} has to be stricter.

In addition to comparing with given tolerances, Newton's method is also stopped if the number of newton iterations is larger than a given maximum number of newton iterations. In our simulator, the default number of allowed newton iterations is 10. More sophisticated stop criteria can be implemented, but for simplicity and to be able

[1] http://graal.ens-lyon.fr/MUMPS/.
[2] http://www.netlib.org/blas/.

to directly control the size of the residual values, we keep to the above described stop criterion.

4.3.6 Time stepping

As a consequence of using a fully implicit scheme, it is unconditionally stable, meaning that we in theory can take time steps of arbitrary size. However, in practice there are certain limitations given by the following considerations [6, Aziz, p. 207].

- Large time steps lead to large discretization errors. This follows from the derivation of equation (3.33).
- When solving the nonlinear system of equations iteratively, the number of newton iterations increases as the size of the time step increases, since the solution from the previous time step is used as initial guess. The iterative process may not converge, if the size of the time step is too large.

To obtain an effective simulator, a good balance between the size of the time step and the number of newton iterations per time step has to be found. Most simulators use a combination of various time step selection methods. Some popular choices are

- Limit on maximum change in key variables.
- Limit on time truncation error.

The time step selection method that utilizes the maximum change in key variables is widely used, since it is fairly easy to implement. More simple heuristic methods that consider the number of newton iterations for the previous time step can also be employed. In this work, we have implemented and tested three different time step selection methods. The three different methods are explained in the following. In section 4.5, a comparison study of the three methods is carried out.

The first two time step selection methods are based on knowledge from previous time steps. To begin with we implemented the naive method in Algorithm 4.3. It simply increases the time step size with 30% when the number of newton iterations in the previous time step was less than 2/3 of the maximum allowed number of newton iterations and chops the time step size in half otherwise.

Algorithm 4.3 Naive heuristic

```
1: if $k < \frac{2}{3}k_{\max}$ then
2:     $\Delta t = 1.3 \cdot \Delta t$
3: else
4:     $\Delta t = 0.5 \cdot \Delta t$
5: end if
6: $\Delta t = \min(\Delta t, \Delta t_{\max})$
```

The increase and decrease factors: 30% and 1/2 are chosen by parameter tuning for different problems. All parameters in the methods described in this section are chosen on a heuristic basis. As it will be described later, the performance of this naive approach is reasonable, but it leaves room for methods with more complexity for controlling time step sizes. We found the naive approach to be overly aggressive in increasing time step sizes in some situations and too slow at increasing time step sizes in other situations. Furthermore, we found that the naive approach too often chopped the time step size in half unnecessarily.

We attempt to overcome these deficiences by including knowledge from more than one previous time step. The approach is described in Algorithm 4.4. If the number of newton iterations from the previous iteration is less than half of the maximum allowed number of newton iterations, a count of successful time steps is incremented and 10% for each successful time step is added to the time step size. On the other hand, if the number of newton iterations from the previous time step is greater or equal to 2/3 of the maximum allowed number of newton iterations then the count of successful time steps is reset to zero and the time step size is halfed. Lastly, if $\frac{1}{2}k_{\max} \leq k < \frac{2}{3}k_{\max}$ the time step size is kept, but the count of successful time steps is reset to zero.

Algorithm 4.4 Improved heuristic

```
 1: if $k < \frac{1}{2}k_{\max}$ then
 2:     SuccessRuns = SuccessRuns + 1
 3:     $\Delta t = \Delta t \cdot (1.0 + 0.1 \cdot \text{SuccessRuns})$
 4: else if $k \geq \frac{2}{3}k_{\max}$ then
 5:     SuccessRuns = 0
 6:     $\Delta t = 0.5 \cdot \Delta t$
 7: else
 8:     SuccessRuns = 0
 9: end if
10: $\Delta t = \min(\Delta t, \Delta t_{\max})$
```

The third time step selection method we implemented is based on more mathematical terms. It uses the maximum change in key variables as described in [6, Aziz, p. 207] and [26, Grabowski]. The formula is given in equation (4.20)

$$\Delta t^{n+1} = \Delta t^n \min_{i \in \Omega} \frac{(1+w)\eta}{\delta_i + w\eta}, \tag{4.20}$$

where $w \in [0;1]$ is a tuning parameter, η is a desired change and δ_i is the change of the value of a key variable over the previous time step n for a grid block i in the domain Ω. The formula in equation (4.20) has to be applied to all key variables as outlined in Algorithm 4.5.

Evidently, this approach requires more computational work, however this is negligible compared to the other components of a simulator.

Algorithm 4.5 Maximum change in key variables

```
1: $\text{frac} = 10^{16}$
2: for $i = 1..N$ do
3:     $\text{frac} = \min\left(\text{frac}, \dfrac{(1+w)\cdot\eta_{S^o}}{|S_i^o - S_{i,\text{old}}^o| + w\eta_{S^o}}\right)$
4:     $\text{frac} = \min\left(\text{frac}, \dfrac{(1+w)\cdot\eta_{S^g}}{|S_i^g - S_{i,\text{old}}^g| + w\eta_{S^g}}\right)$
5:     $\text{frac} = \min\left(\text{frac}, \dfrac{(1+w)\cdot\eta_{S^w}}{|S_i^w - S_{i,\text{old}}^w| + w\eta_{S^w}}\right)$
6:     $\text{frac} = \min\left(\text{frac}, \dfrac{(1+w)\cdot\eta_{p}}{|p_i - p_{i,\text{old}}| + w\eta_{p}}\right)$
7: end for
8: $\Delta t = \min(\Delta t \cdot \text{frac}, \Delta t_{\max})$
```

As mentioned before, a comparison of the three time step selection methods is carried out in section 4.5.

4.4 Verification

Complex reservoir models are hard to verify, since no comparable analytic solutions exist. In this work, we verify our implementation by comparing solutions with the solutions from the most established commercial reservoir simulator on the market: ECLIPSE from Schlumberger[3]. For this comparison, we use the 2012.1 version of ECLIPSE. Given the input values in Appendix A and the grid cell sizes $\Delta x = 20$, $\Delta y = 10$, $\Delta z = 2$ meters, we verify our implementation using three different test cases.

Note that porosities are kept constant throughout the domain. This is primarily done to keep focus on how permeabilities affect the robustness and the performance of the simulator as we expect the simulator to be more sensitive towards heterogeneous permeability fields rather than heterogeneous porosity fields. This is due to the fact that permeabilities explicitly control the flow of fluids, whereas the porosities are part of the accumulation term.

The three test cases are:

- The simplest possible case with a homogeneous permeability field. Problem size: 1x1x3.
- More complicated case with a smooth heterogeneous permeability field. Problem size: 18x18x18.

3) www.slb.com.

- Even harder case with a very heterogeneous permeability field derived from the SPE10 comparative solution project[4]. Problem size: 10x10x10.

For all three test cases, the initial saturations, the permeability field and the solution are presented. All test cases are what we call "gravity inversion", where the components water and gas switch place due to the gravity and their given densities. The initial pressure for all test cases is $p = 250$ bar.

4.4.1 Simple homogeneous permeability field

The first test case consists of three vertical grid cells, where the top cell has a water saturation of one, the middle cell has an oil saturation of one and the bottom cell has a gas saturation of one. Because of the gravity and the given densities of oil, water and gas, we expect the water to end up in the bottom cell and gas to end up in the top cell.

The saturations are initialized as depicted in Figure 4.4.

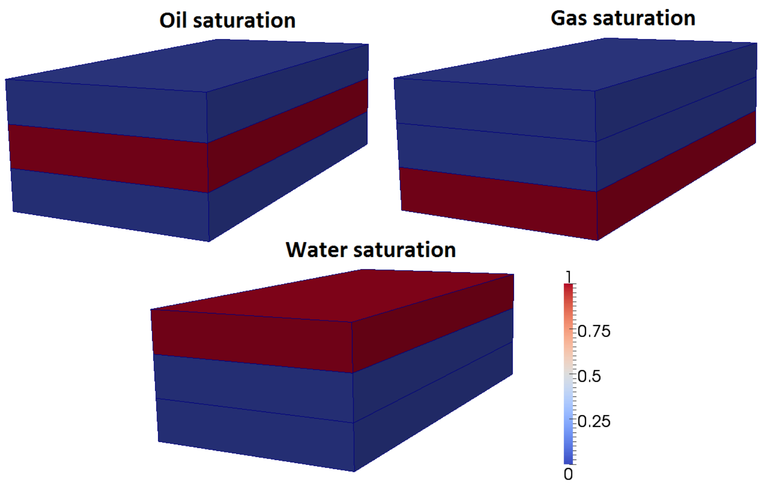


Figure 4.4: *Initial saturation values.*

Given these initial saturations, we initialize the primary variables molar densities using formula (C.9). The absolute permeability is 100 millidarcy (mD) in all three cells. Note that we are using isotropic permeability fields. We allow 10 newton iterations per time step. The solution is displayed in Figure 4.5.

As Figure 4.5 shows, the solution is the same as the solution obtained from ECLIPSE. The simulation is carried out with a maximum time step of 0.1 days to ensure that we take exactly the same time steps as ECLIPSE.

ECLIPSE has a more sophisticated time step selection procedure and uses a type of damped Newton's method [47, Younis] with better convergence properties. This allows

[4] http://www.spe.org/web/csp/.

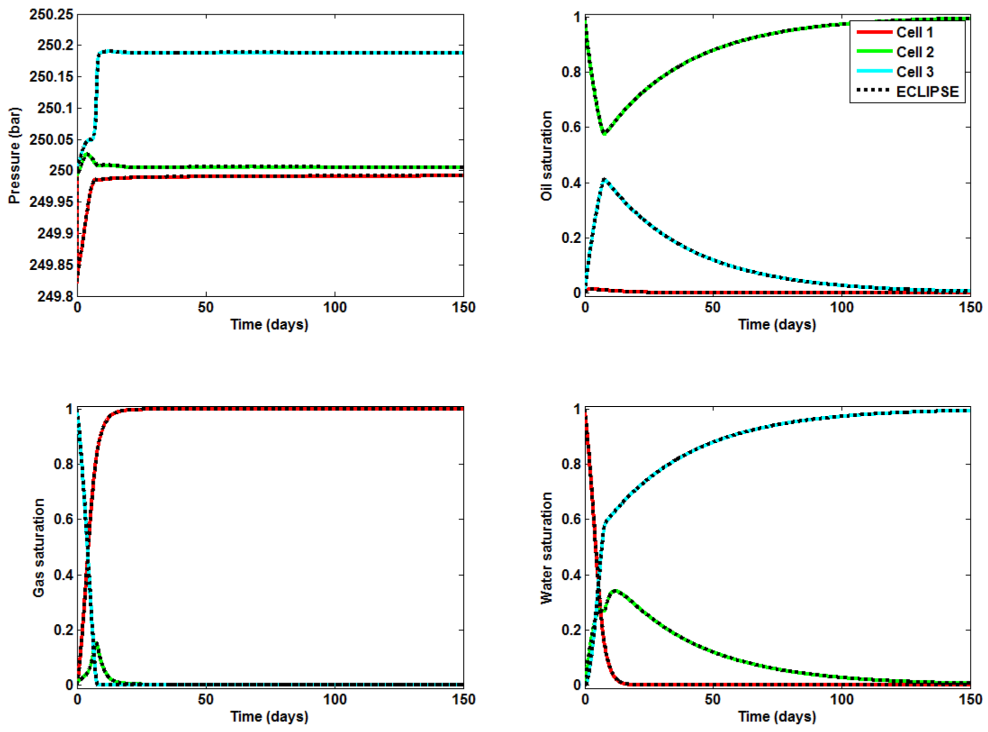


Figure 4.5: *Verification of a* $1 \times 1 \times 3$ *problem, 150 days of simulation,* $\Delta t_{\max} = 0.1$ *and initial time step length of 0.1 days.*

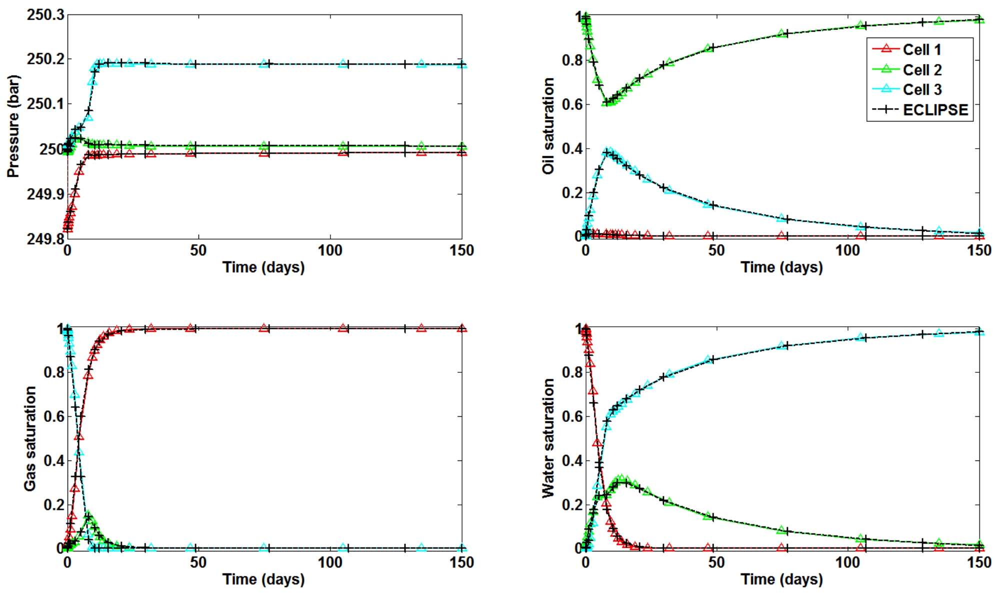


Figure 4.6: *Verification of a* $1 \times 1 \times 3$ *problem, 150 days of simulation,* $\Delta t_{\max} = 30$ *and initial time step length of 0.1 days.*

ECLIPSE to take longer and more adaptive time steps than we are able to with the time step selection method in Algorithm 4.4 and standard Newton's method. However, if we permit ECLIPSE and our simulator to take time steps of up to one month, the results are still very much alike as depicted in Figure 4.6. The dots indicate the data points. ECLIPSE uses 17 time steps and our simulator uses 24 time steps. Despite the low number of data points, the difference in the solutions is hardly visible.

4.4.2 Smooth heterogeneous permeability field

The second test case is using a smooth heterogeneous permeability field, which is generated using trigonometric formulas and a random number generator. We have implemented a function that given any problem size will generate a fixed pattern permeability field. See Appendix E for details. This approach will allow for more comparable simulations across various problem sizes. The generated permeability fields have higher variation in the z-direction than in the x- and y-directions. This mimics what we expect from real reservoirs, where different rock types are layered on top of each other. We can control the fixed pattern by setting a seed for the random number generator.

The initial saturations are similar to the previous test case. As Figure 4.7 shows, the top 6 layers are water, the middle 6 layers are oil and the bottom 6 layers are gas.

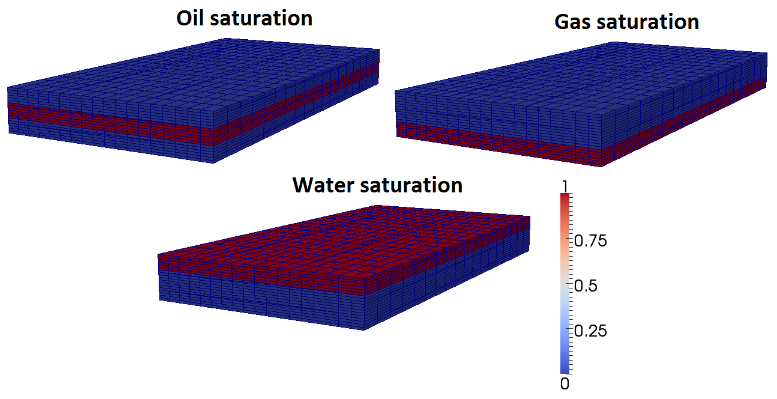


Figure 4.7: *Initial saturation values.*

The permeability field is displayed in Figure 4.8. The permeabilities range from 200 to 600 mD.

10 newton iterations are allowed per time step. The solution is shown in Figure 4.9. The results obtained from ECLIPSE are similar.

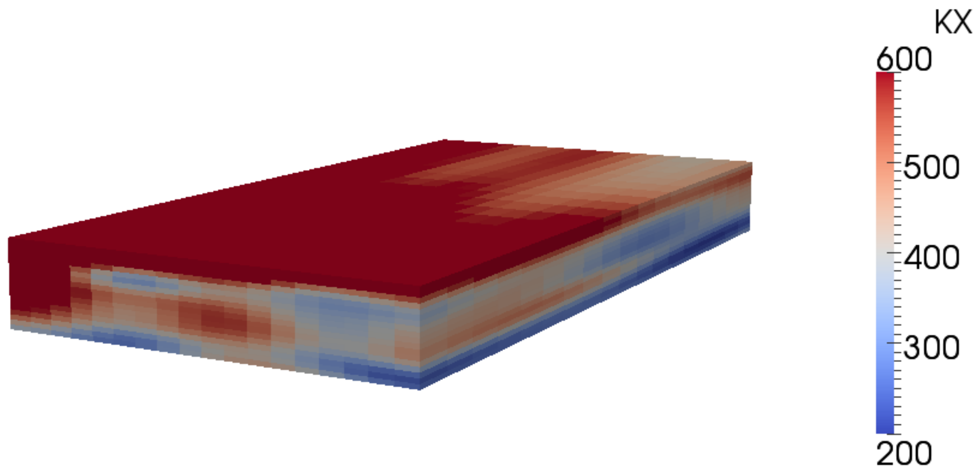


Figure 4.8: *Smooth heterogeneous permeability field.*

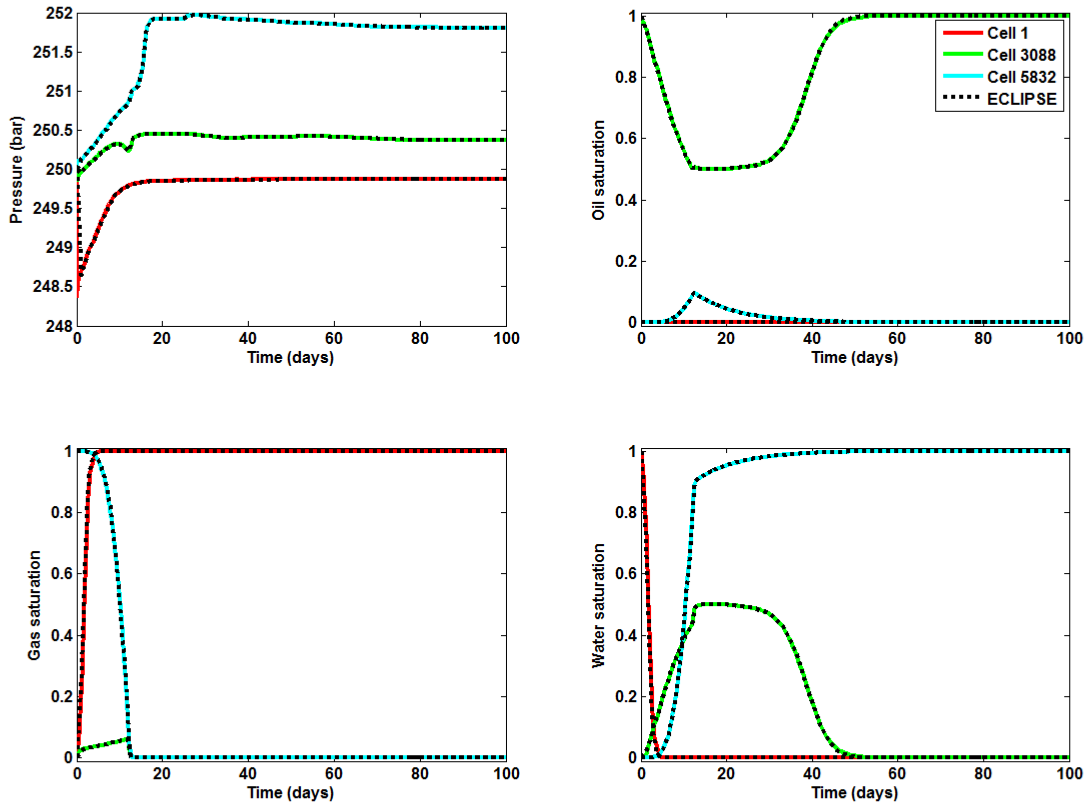


Figure 4.9: *Verification of a* $18 \times 18 \times 18$ *problem, 100 days of simulation,* $\Delta t_{\max} = 30$ *and initial time step length of 0.1 days.*

4.4.3 SPE10 derived heterogeneous permeability field

The last test case is using a permeability field derived from the SPE10 comparative solution project. SPE10 is designed to compare upscaling approaches [13], but it is also widely used as a benchmark for large-scale highly heterogeneous reservoir simulation. Compared to geological models of real reservoirs, the SPE10 is less realistic, however, as a stress test for validation and performance evaluations of a simulator, it is relevant. The second SPE10 case models a $1200 \times 2200 \times 170$ ft reservoir with 60x220x85 $= 1,122,000$ grid cells, but we are only using a small segment of 10x10x10 $= 1,000$ grid cells and scaling the permeabilities between 0.1 and 1,000 mD.

Again, the problem is initialized with water in the top, oil in the middle, and gas in the bottom as depicted in Figure 4.10.

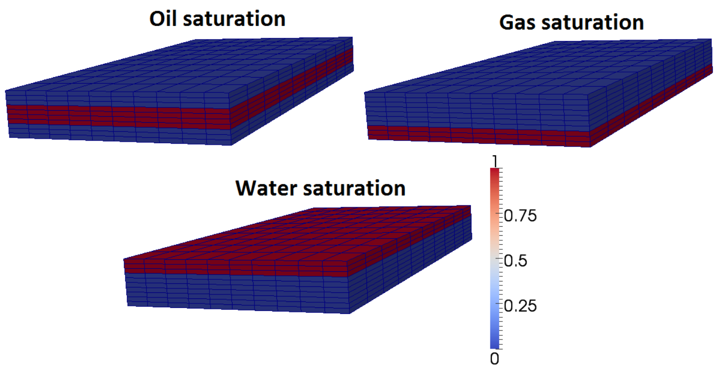


Figure 4.10: *Initial saturation values.*

The permeability field is displayed in Figure 4.11. As it can be seen, it is highly heterogeneous with large and small permeability values scattered around the domain.

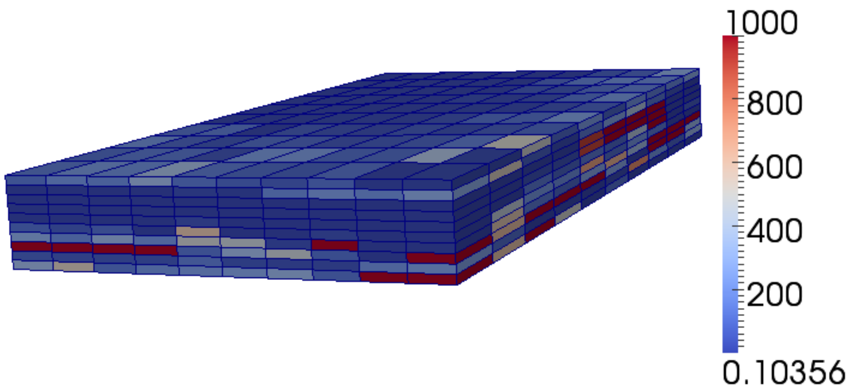


Figure 4.11: *Permeability field.*

Since some of the permeability values are very small (≈ 0.1 mD), it now takes more than 40 years to reach equilibrium. 10 newton iterations per time step are allowed. The solution is displayed in Figure 4.12 and, as it can be seen, ECLIPSE provides similar results.

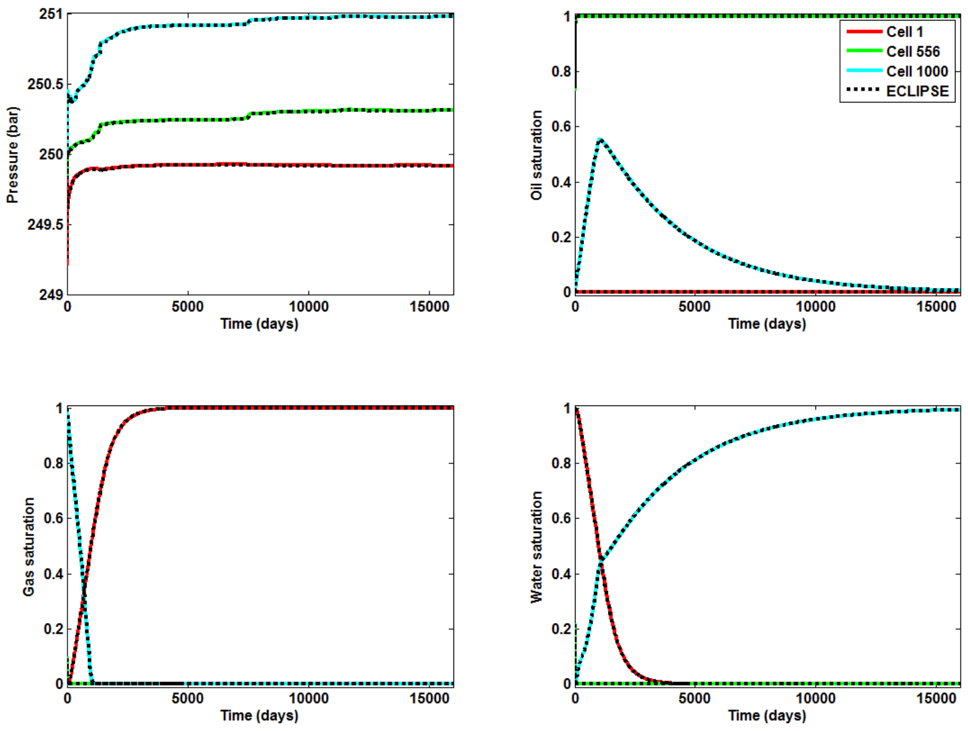


Figure 4.12: *Verification of a* $10 \times 10 \times 10$ *problem, 16000 days* $\approx$ *44 years of simulation,* $\Delta t_{\max} = 30$ *and initial time step length of 0.1 days.*

4.5 Time stepping study

The performance of the three time step selection methods described in section 4.3.6 are studied on the three test cases introduced in the verification, section 4.4. The hardware used to run the tests is hardware specifications 2 in appendix B. Table 4.3 contains the results for the naive approach described in algorithm 4.3, the improved approach in algorithm 4.4 and the maximum change in key variables approach in algorithm 4.5.

	Naive	Improved	Max change
Test case 1	26 steps	24 steps	24 steps
Test case 2	108 steps/1092 secs	110 steps/963 secs	141 steps/1102 secs
Test case 3	944 steps/275 secs	965 steps/265 secs	809 steps/291 secs

Table 4.3: *Performance of time step selection methods.*

Test case 1 is too small to measure any meaningful timings. We observe that the naive approach uses 2 additional time steps compared to the two other methods. The improved approach is better than the two other methods for test case 2 and test case 3 in terms of computation time. From extensive testing, this is the general behaviour we experience. The length of the time steps used by each of the three methods for the test cases are distributed as depicted in Figure 4.13.

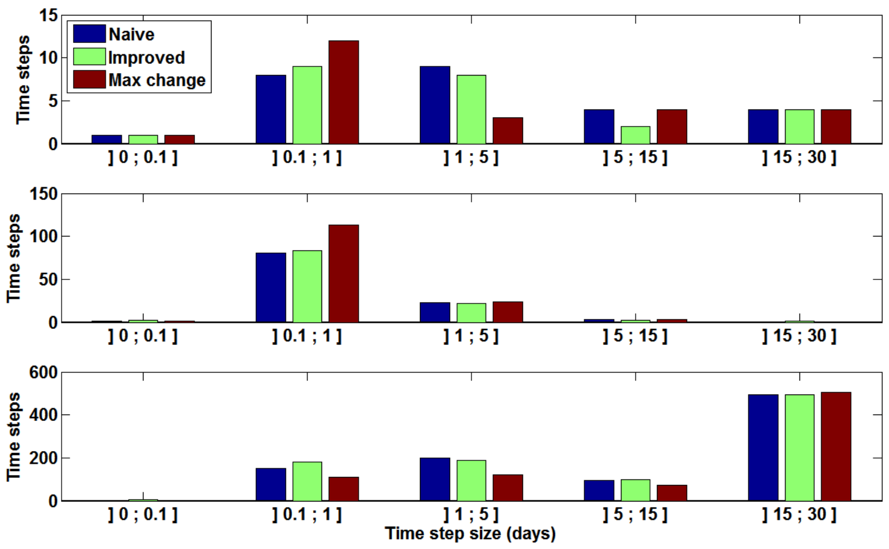


Figure 4.13: *Distribution of time step sizes for each method for the test cases. The top plot is for test case 1, the middle plot is for test case 2 and the bottom plot is for test case 3.*

The distribution of time step sizes for the three time stepping methods appears quite similar. However, the method based on maximum change in key variables seems to take more of the shorter steps than the other two methods for test cases 1 and 2. For the very heterogeneous test case 3, the method based on maximum change in key variables takes more of the longer steps than the other two methods.

The maximum change in key variables method is very sensitive to changes in its parameters. After some testing and tuning, we found the following parameters to give reasonable results:

- $\eta_{S^o} = 0.1$
- $\eta_{S^g} = 0.1$
- $\eta_{S^w} = 0.1$
- $\eta_p = 5$

These values are to some extent problem specific and, as a result, it can be hard to find optimal parameter values. If the η values are set too small, too many time steps are carried out. However, if the η values are set too large, it attempts to take too long time steps, resulting in Newton's method failing to converge for many time steps.

Overall, the improved method works well for all problems we have encountered. As a consequence, we use this method in the rest of this thesis.

4.6 Performance study

This section contains a small performance study of the simulator using Newton's method to solve the nonlinear system of equations. In Chapter 7, a more comprehensive performance study is carried out that includes all of the improvements described in Chapter 6.

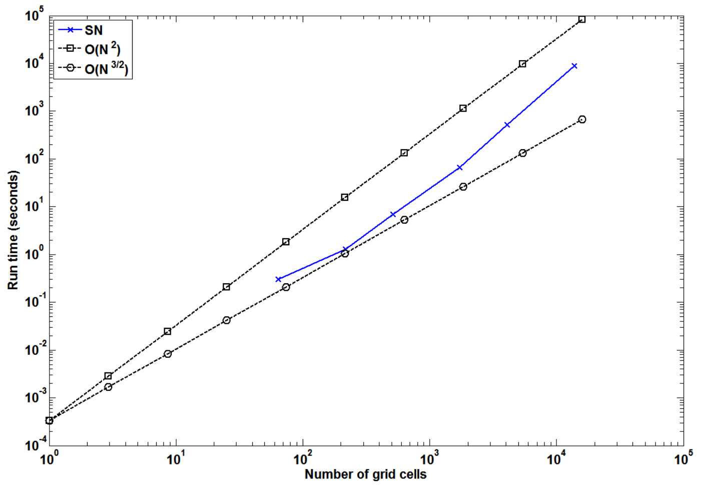


Figure 4.14: *Run time as a function of number of grid cells.*

Figure 4.14 shows the run time of the simulator for different problem sizes with a fixed reservoir size (480m x 240m x 48m). The permeability fields are generated using the technique described in Appendix E. Simulations are running 150 days with the parameters specified in Appendix A, initial time step size of 0.1 days and maximum time steps size of 30 days. Hardware specifications 2 in Appendix B is used. 10 newton iterations are allowed for these simulations.

The run time is increasing dramatically as the number of grid cells increases. This can be seen by comparing the $O(N^{3/2})$ and $O(N^2)$ lines with the scaling of the simulator. This increase is expected, since we are using the sparse direct solver MUMPS to solve the linear systems in each newton iteration. The reason is that the complexity of factorizing a band matrix is dependent on the bandwidth of the matrix. The larger the bandwith is, the higher is the complexity, [24, p.152]. The example in section 4.2, leading to the Jacobian matrix in Table 4.1, illustrates that the bandwidth of the Jacobian matrix for the reservoir problem is dependent on the problem size. This implies that the complexity increases as the problem size increases, which leads to poor scalability.

Profiling the code with the tool *callgrind* in Valgrind[5], the percentage of total computation time for MUMPS for different problem sizes is measured and displayed in Figure 4.15. Again the reservoir size is fixed.

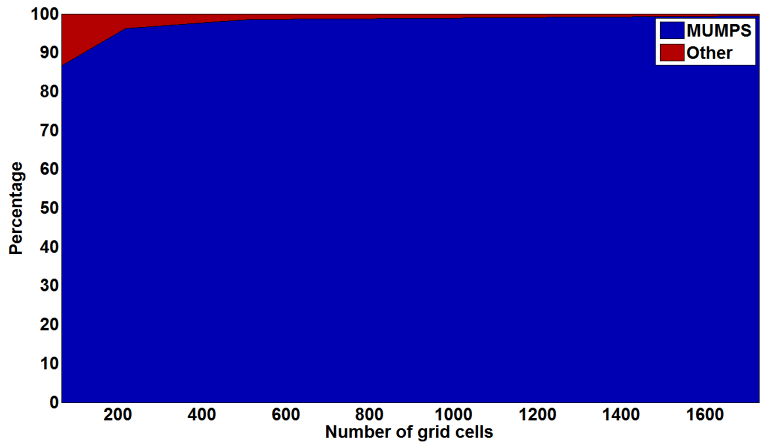


Figure 4.15: *Percentage of total computation time for the sparse direct solver MUMPS for different number of grid cells.*

MUMPS quickly takes up 99% of the computation time even for small problems. Consequently, effort has been put into replacing MUMPS with an iterative linear solver. This is explained in detail in section 6.3.

4.7 Summary

A working simulator has been implemented, verified and studied. As most other reservoir simulators, it is based on global linearization in Newton's method to solve the nonlinear system of equations arising at each time step. The linear solver is a direct linear solver from a library and as demonstrated in section 4.6, it scales poorly for larger 3D problems and consequently it makes up nearly all computation time in the reservoir simulator. As described in the objectives, section 1.4, we aim at decreasing the dependence on the linear solver. This is accomplished through implementation of a FAS-based simulator as described in the following chapter.

[5] http://valgrind.org/.

Chapter 5

Nonlinear multigrid

In chapter 4, we used Newton's method to solve the discretized system of differential equations in (3.38). We are now going to solve this system of differential equations using a nonlinear geometric multigrid method called the Full Approximation Scheme (FAS). As it will be outlined later, Newton's method is an essential part of the FAS algorithm.

Before approaching the theory behind the nonlinear FAS method in section 5.2, an introduction to the basic concepts of multigrid methods is provided in section 5.1. This introduction is based on linear multigrid methods. Furthermore, this section presents a naive Matlab implementation of the FAS method for solving a simple nonlinear problem, with the purpose of getting familiar with the method before applying it to the more advanced reservoir problem. In section 5.3, the first version of our FAS based reservoir simulator is described. The implementation is done in C++. Section 5.4 holds a verification study of the FAS simulator presented in 5.3, and finally section 5.5 contains a small performance study of the algorithm.

5.1 The multigrid idea

A multigrid method operates on a set of $M+1$ grids $G^0, G^1, ..., G^M$, all approximating the same domain Ω. These grids have different mesh sizes h, such that $h_0 < h_1 < ... < h_M$. This means that G^0 has the finest mesh resolution and G^M the coarsest, [8, Brandt, p.337]. Given a differential equation we want to approximate the solution at the finest grid by doing most of the computational work at the coarsest grid.

The two main principles in a multigrid method is error smoothing and coarse grid correction.

5.1.1 Error smoothing

Assume that we want to solve the linear system

$$\mathbf{A}\mathbf{u} = \mathbf{f} \tag{5.1}$$

By $\mathbf{v}$ we denote an approximation to the exact solution $\mathbf{u}$ and by

$$\mathbf{e} = \mathbf{u} - \mathbf{v} \tag{5.2}$$

we denote the error. If we apply an iterative relaxation method to $\mathbf{v}$, e.g. the Jacobi method or the Gauss-Seidel method (these methods are described below), then it is possible in only a few iterations to remove the high frequency contributions to the error. This means that the error of the approximation becomes smooth quickly, [43, Trottenberg, p.15].

In order to illustrate this statement we consider the linear 2-dimensional Poisson equation

$$\begin{aligned} -\nabla^2 u(x,y) &= f(x,y) \qquad \in \Omega \\ u(x,y) &= 0 \qquad\qquad \in \partial\Omega, \end{aligned} \tag{5.3}$$

where we assume that the analytical solution is given by

$$u(x,y) = (x - x^2)(y - y^2) \tag{5.4}$$

This gives us the righthand-side

$$f(x,y) = 2((x - x^2) + (y - y^2)) \tag{5.5}$$

We define our initial guess, $\mathbf{v}$, to the solution, $\mathbf{u}$, as the exact solution minus a random noise signal ω

$$v(x,y) = u(x,y) - \omega \tag{5.6}$$

This means that the error, $\mathbf{e}$, defined in (5.2) becomes this random noise signal. In this way, we are able to study how well our relaxation method eliminates the error.

If we discretize (5.3) using a central finite difference approximation and use the notation $u_{i,j} = u_{i,j}(x,y)$ we end up with the following equation

$$-\left(\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2}\right) = f_{i,j}, \tag{5.7}$$

which by isolating $u_{i,j}$ can be rewritten into

$$u_{i,j} = \frac{1}{4}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} + h^2 f_{i,j}\right), \tag{5.8}$$

We can apply this scheme iteratively to our initial solution $\mathbf{v}$ and thereby approximate the exact solution $\mathbf{u}$. If we traverse all cells in our grid before updating any of the values in the solution, such that we operate with a $\mathbf{u}$ and a $\mathbf{u}_{old}$, we reach the Jacobi relaxation method

$$u_{i,j}^{k+1} = \frac{1}{4}\left(u_{i-1,j}^{k} + u_{i+1,j}^{k} + u_{i,j-1}^{k} + u_{i,j+1}^{k} + h^2 f_{i,j}\right), \tag{5.9}$$

where k is an iteration number. By applying the Jacobi method to the initial guess given in (5.6) and comparing the output data to the exact solution, we can study the error $\mathbf{e}$ as a function of the number of Jacobi smoothings. The result is seen in Figure 5.1.

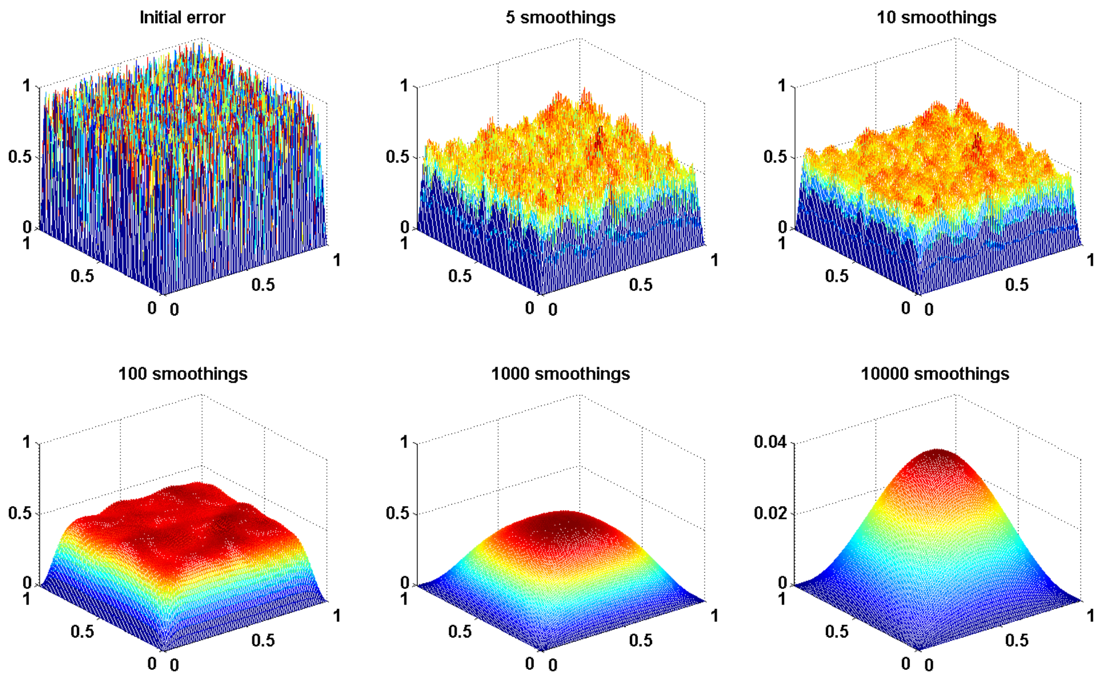


Figure 5.1: *Initial error and error after different number of Jacobi smoothings.*

The top left plot of this figure shows the initial error, which resembles white noise. The remaining five plots illustrate the error after respectively 5, 10, 100, 1000 and 10000 Jacobi smoothings. It is seen that the Jacobi method slowly eliminates the error, and somewhere between 10 and 100 iterations the error becomes smooth. However, even after 10000 iterations the error is still quite large, meaning that the Jacobi method has a slow convergence rate.

Instead of applying the stencil in (5.8) to all the cells in the grid before making an update, we could update the solution continuously. This leads to the Gauss-Seidel relaxation method given by

$$u_{i,j}^{k+1} = \frac{1}{4}\left(u_{i-1,j}^{k+1} + u_{i+1,j}^{k} + u_{i,j-1}^{k+1} + u_{i,j+1}^{k} + h^2 f_{i,j}\right), \tag{5.10}$$

Repeating the experiment in Figure 5.1 using the Gauss-Seidel method leads to the results in Figure 5.2.

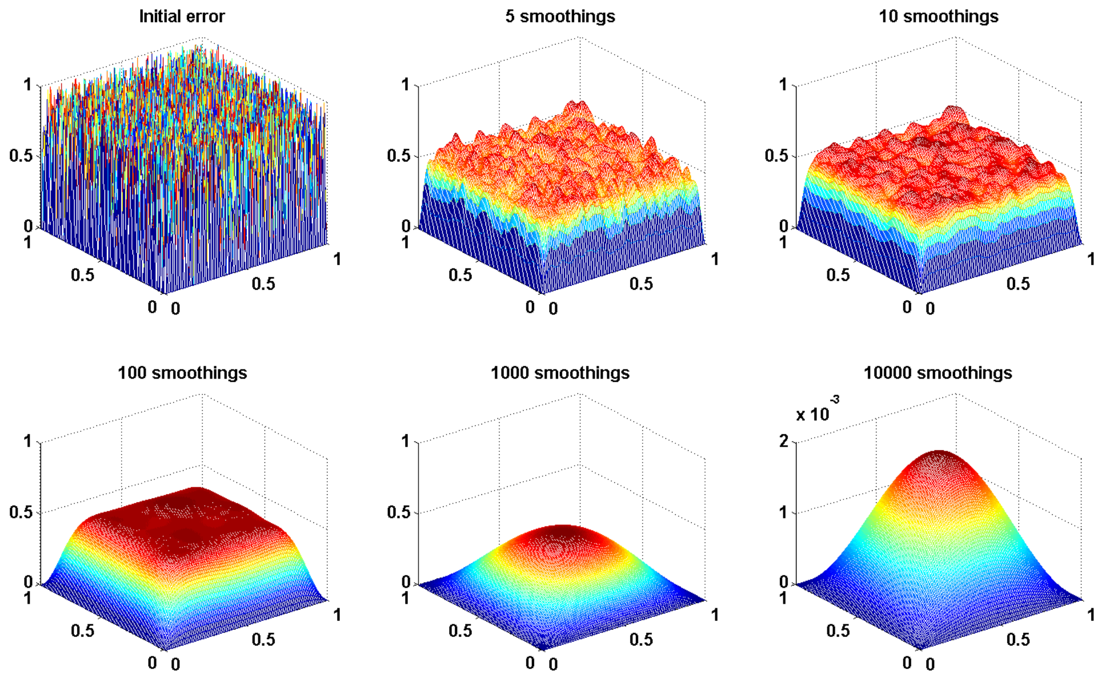


Figure 5.2: *Initial error and error after different number of Gauss-Seidel smoothings.*

The figure shows that after only 5-10 smoothings the error is relatively smooth, and after 10000 smoothings the Gauss-Seidel smoother has reduced the error to a magnitude of 10^{-3}. This indicates that the Gauss-Seidel method has a higher convergence rate than the Jacobi method. Theory also confirms that the Gauss-Seidel method in general has better smoothing properties than the Jacobi method, [43, Trottenberg, p.29].

Furthermore, the Gauss-Seidel method uses less memory than the Jacobi method, because the Gauss-Seidel method does not operate with a $\mathbf{u}_{old}$ solution vector. On the other hand, the Gauss-Seidel method is not as straightforward to parallelize as the Jacobi method. This will be addressed in more detail in chapter 8.

5.1.2 Coarse grid correction

Consider a quantity which is smooth on a certain grid. According to [43, Trottenberg, p.16], it is possible to approximate this quantity on a coarser grid without any essential loss of information.

In the perspective of solving the system $\mathbf{Au} = \mathbf{f}$, this means that quantities, such as the error $\mathbf{e}_h$ and the approximate solution $\mathbf{v}_h$ at a certain grid level with mesh size h, can be approximated relatively well as $\mathbf{e}_H$ and $\mathbf{v}_H$ at a coarser grid level with mesh size $H > h$, as long as these quantities are smooth.

This gives the opportunity to do computations at a coarser and less computationally expensive grid.

As mentioned, it is essential that the considered quantities are smooth. We illustrate this importance with an example.

Once again we consider the equation given in (5.3), and its solution $\mathbf{u}$ in (5.4). The initial guess $\mathbf{v}$ is given as in (5.6) with the exception of the random noise signal ω being added instead of subtracted. The exact solution $\mathbf{u}$ and the initial guess $\mathbf{v}$ are plotted on a uniform cell-centered grid with 128×128 grid cells and mesh size h. Furthermore, we plot $\mathbf{v}$ after 10 Gauss-Seidel iterations. These plots are seen in the three top plots of Figure 5.4.

From these plots it is seen that the exact solution is very smooth and the initial guess is not at all smooth because of the random noise. However, after 10 Gauss-Seidel iterations the initial guess has become smooth.

We interpolate (restrict) each of these quantities $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{v}_{GS}$ to a coarser grid with only 64×64 grid cells and mesh size $H = 2h$. The restriction stencil, which is applied, is the four-point average stencil, [43, Trottenberg, p.69], given by

$$\begin{aligned} I_h^{2h} v_h(x,y) =& \frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}_h^{2h} v_h(x,y) = \\ & \frac{1}{4}\left[v_h\left(x - \frac{h}{2}, y - \frac{h}{2}\right) + v_h\left(x - \frac{h}{2}, y + \frac{h}{2}\right) \right. \\ & \left. + v_h\left(x + \frac{h}{2}, y - \frac{h}{2}\right) + v_h\left(x + \frac{h}{2}, y + \frac{h}{2}\right)\right], \end{aligned} \tag{5.11}$$

meaning that in a cell-centered 4×4 fine grid we end up with the 2×2 coarse grid seen in Figure 5.3.

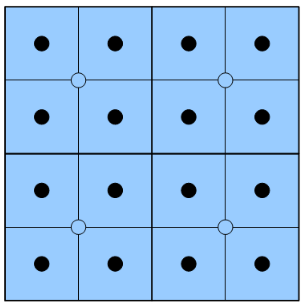


Figure 5.3: *Cell-centered grid cells in fine 4×4 and coarse 2×2 grid, where (●) indicate fine grid cells and (○) indicate coarse grid cells.*

The middle three plots in Figure 5.4 shows the restricted quantities $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{v}_{GS}$. These plots illustrate that information is lost if a quantity is not smooth before restricting it. If the quantity is smooth, then the essential information is preserved.

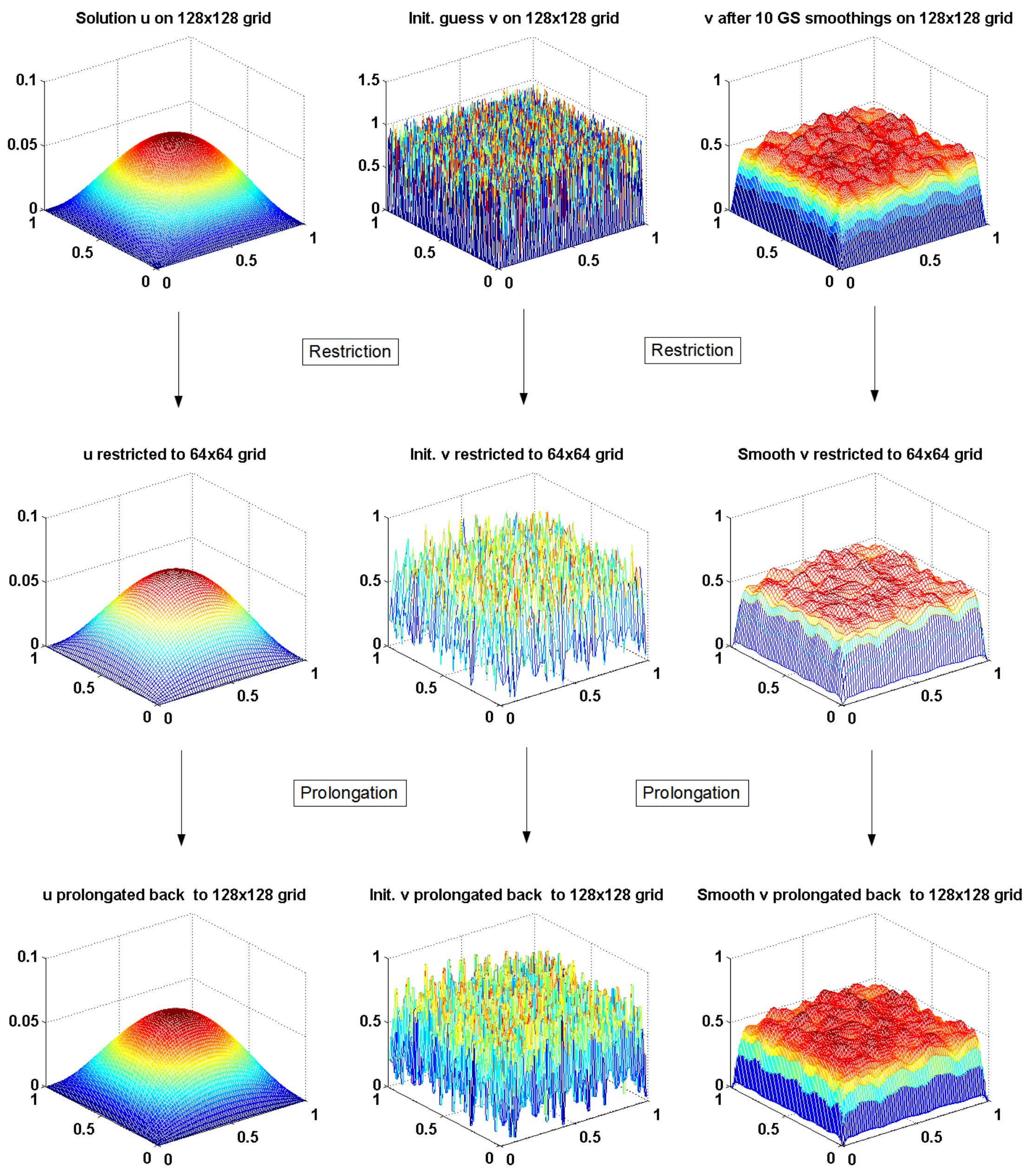


Figure 5.4: *Illustration of the importance of smoothing. Exact solution* $\boldsymbol{u}$*, initial guess* $\boldsymbol{v}$ *and smooth initial guess* $\boldsymbol{v}$ *restricted to a coarser grid and prolongated back again. After prolongation only the smooth quantities resemble themselves as before restriction. Note that the discrepancies on the boundary of the restricted and prolongated solutions are due to the grid being cell-centered.*

In the last three plots of Figure 5.4 the restricted quantities are interpolated (prolongated) back to the fine grid. The smooth quantities resemble themselves as they were before restriction. The non-smooth initial guess does not resemble itself.

The prolongation is done using the following stencil

$$I_{2h}^{h} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}_{2h}^{h}, \tag{5.12}$$

meaning that the information held in the coarse grid cell is prolongated (injected) to the 4 surrounding fine grid cells.

In practice, we are not at the coarsest grid seeking an approximation to the system of equations, but a correction term which can bring our initial guess closer to the exact solution.

5.1.3 A multigrid algorithm

Having introduced these two main principles, we are ready to describe a multigrid method for solving the linear system in (5.1). We define the residual of this system as

$$\mathbf{r} = \mathbf{f} - \mathbf{A}\mathbf{v}, \tag{5.13}$$

where $\mathbf{v}$ is an approximation to the exact solution $\mathbf{u}$. Using the definition of the error in (5.2), the linear system is rewritten into

$$\mathbf{A}\mathbf{u} = \mathbf{A}\left(\mathbf{v} + \mathbf{e}\right) = \mathbf{A}\mathbf{v} + \mathbf{A}\mathbf{e} = \mathbf{f} \tag{5.14}$$

Inserting (5.13) in (5.14) reveals a critical relationship

$$\mathbf{A}\mathbf{v} + \mathbf{A}\mathbf{e} = \mathbf{A}\mathbf{v} + \mathbf{r} \Leftrightarrow$$

$$\mathbf{A}\mathbf{e} = \mathbf{r}, \tag{5.15}$$

which is known as the residual equation, [28, Henson, p.1].

The concept in the multigrid algorithm described below is to consider the system $\mathbf{A}\mathbf{v} = \mathbf{f}$ at the finest grid. Here the high frequency contributions to the error are removed using the Gauss-Seidel method and the fine grid residual $\mathbf{r}_h$ is restricted to a coarser grid $\mathbf{r}_H = I_h^H$. Calling the algorithm recursively, we reach the coarsest grid, where the residual equation in (5.15) is solved, yielding a correction term $\mathbf{e}_H$. This correction is prolongated one grid up and used to correct the initial solution at that current grid level, $\mathbf{v}_h = \mathbf{v}_h + I_H^h \mathbf{e}_H$. Doing this recursively gives us an approximation to the solution to the fine grid problem $\mathbf{A}\mathbf{u} = \mathbf{f}$, [28, Henson, p.1]. If this approximation is not satisfactory, the procedure is repeated with the approximation being the new initial guess.

This method is described in pseudo code in Algorithm 5.1.

Algorithm 5.1 Multigrid method for solving linear system $\mathbf{Au} = \mathbf{f}$

```
1: Provide initial guess $\mathbf{v}^0$
2: $k = 0$
3: while not converged and $k < k_{max}$ do
4:     $k = k + 1$
5:     $\mathbf{v}^k = \text{MGCYC}(\mathbf{v}^{k-1}, \mathbf{f})$
6:     Compute residual $\mathbf{r}^k = \mathbf{f} - \mathbf{A}\mathbf{v}^k$
7: end while
```

Given an initial guess $\mathbf{v}^0$ of the solution to the linear system, the algorithm enters a while-loop with conditions on both the accuracy of the solution and the number of iterations used. At each iteration, an updated solution and the corresponding residual are computed. The residual is given by (5.13). If the norm of this residual is smaller than some user specified tolerance, the algorithm stops.

The solution update in line 5 in Algorithm 5.1 is computed using the *MGCYC* subroutine, which is a multigrid cycle algorithm following the above presented concept and the description in [43, Trottenberg, p.47]. This algorithm is given in pseudo code in Algorithm 5.2.

Algorithm 5.2 Multigrid Cycle - $\mathbf{v}_h = \text{MGCYC}(\mathbf{v}_h, \mathbf{f}_h)$

```
1: if on coarsest grid level then
2:     Solve $\mathbf{A}_h\mathbf{v}_h = \mathbf{f}_h$
3:     Return $\mathbf{v}_h$
4: else
5:     Pre-smoothings on $\mathbf{v}_h$
6:     Compute residual: $\mathbf{r}_h = \mathbf{f}_h - \mathbf{A}_h\mathbf{v}_h$
7:     Restrict residual: $\mathbf{r}_H = I_h^H\mathbf{r}_h$
8:     Recursive call: $\mathbf{e}_H = \text{MGCYC}(\mathbf{0}_H, \mathbf{r}_H)$
9:     Prolongate correction: $\mathbf{e}_h = I_H^h\mathbf{e}_H$
10:    Correction: $\mathbf{v}_h = \mathbf{v}_h + \mathbf{e}_h$
11:    Post-smoothings on $\mathbf{v}_h$
12: end if
```

The *MGCYC* algorithm is a recursive algorithm, which at the finest grid level is provided with an initial guess $\mathbf{v}$. In order to smooth the error, a number of pre-smoothings is done using the Gauss-Seidel method. With the error being smooth we can compute a residual, $\mathbf{r}_h$, according to (5.13) and restrict it to a coarser grid without losing essential information. At the coarser grid, a recursive call to the *MGCYC* algorithm is made, where a vector of all zeros, $\mathbf{0}_H$, is provided as initial guess and the restricted residual, $\mathbf{r}_H$, is provided as right-hand side. In this way the residual equation in $\mathbf{A}_H\mathbf{e}_H = \mathbf{r}_H$ is solved at the coarsest grid using some kind of iterative or direct solver. At the coarsest grid, the algorithm does not make a recursive call, but returns the correction term $\mathbf{e}_H$. This term is prolongated to the second-coarsest grid level and added to the initial guess at this level. Thereby, a solution to the residual equation at this grid level is obtained, which again serves as a correction term to the residual equation at the third-coarsest grid. Before prolongating this correction term, a number of post-smoothings are done. This correction procedure is repeated all the way to the fine grid, where the algorithm returns an approximation to the fine grid problem.

The multigrid method given in Algorithms 5.1 and 5.2 are using a V-shaped solution approach, where the algorithm interpolates the problem all the way down from the fine grid to the coarse grid in a straight line, before it in a straight line interpolates a correction term up to the fine grid again. Because of this V-shaped solution approach the method is said to use a V-cycle. There are many different kinds of cycles all depending on the number of grid levels used. Some alternatives are listed in Figure 5.5. All of these cycles are shown for 4 grid levels.

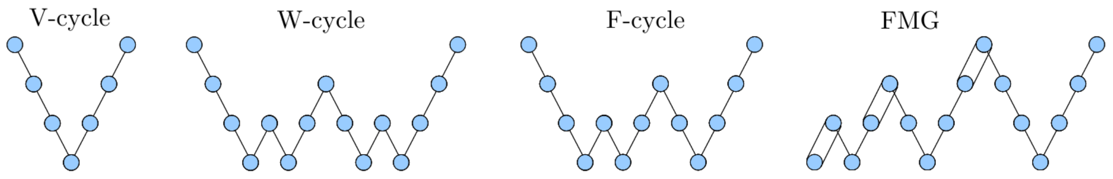


Figure 5.5: *Different types of multigrid cycles using 4 grid levels, where (●) indicates grid levels, (╲) indicates restriction, (╱) indicates prolongation and (╱╱) indicates FMG-interpolation.*

The cycles shown in Figure 5.5 all start at the finest grid, except for the full multigrid (FMG) cycle. The FMG method is based on the idea of nested iteration, meaning that the initial guess on a certain grid level is based on computations and interpolation of approximations on a coarser grid level, [43, Trottenberg, p.56]. The FMG method in Figure 5.5 is called a FMG V-cycle, since it uses V-cycles, but combinations such as FMG W-cycle and FMG F-cycle are also possible.

According to [43, Trottenberg, p.58], the FMG algorithm proceeds as in Algorithm 5.3.

Algorithm 5.3 FMG method for solving linear system $\mathbf{Au} = \mathbf{f}$

```
1: if on coarsest grid level then
2:     Provide initial guess $\mathbf{v}$
3:     Solve $\mathbf{Av} = \mathbf{f}$
4: else
5:     FMG-interpolation of $\mathbf{v}$
6:     Cycle call: $\mathbf{v} = \text{MGCYC}(\mathbf{v}, \mathbf{f})$
7: end if
```

At the coarsest level, an initial guess of the solution is provided to the linear solver. The resulting approximate solution is prolongated (FMG-interpolated) one grid level up to a finer grid, where it is used as an initial guess for a multigrid cycle algorithm like the one in Algorithm 5.2. The approximate solution found by this multigrid cycle algorithm is prolongated to a finer grid, where the multigrid cycle algorithm is called again. This procedure continues until an approximate solution at the finest grid is found.

5.2 The FAS method

With the basic concepts of linear multigrid being described in section 5.1 we move on to the topic of nonlinear multigrid, and in particular the FAS algorithm.

Consider the nonlinear system

$$\mathbf{A}\left(\mathbf{u}\right) = \mathbf{f}, \tag{5.16}$$

where $\mathbf{A}(\mathbf{u})$ is a nonlinear operator which depends on $\mathbf{u}$. We use the parentheses to indicate nonlinearity. Again the error is defined as $\mathbf{e} = \mathbf{u} - \mathbf{v}$. However, this time the residual for the approximated $\mathbf{v}$ is defined as

$$\mathbf{r} = \mathbf{f} - \mathbf{A}\left(\mathbf{v}\right) \tag{5.17}$$

Inserting (5.16) in (5.17) yields

$$\mathbf{A}\left(\mathbf{u}\right) - \mathbf{A}\left(\mathbf{v}\right) = \mathbf{r} \tag{5.18}$$

Since we are considering nonlinear systems, in general $\mathbf{A}\left(\mathbf{e}\right) \neq \mathbf{r}$. This means that we cannot use the multigrid approach described in section 5.1 to find a correction term at the coarsest grid. Instead we use (5.18) as residual equation and use a multigrid method suitable for solving this type of equation, [28, Henson, p.2].

5.2.1 Newton-MG or FAS

There are two approaches to apply multigrid methods for the solution of nonlinear problems. One approach is to apply a global linearization method such as Newton's method to the nonlinear problem, [43, Trottenberg, p.147]. This is similar to the approach described in section 4.2, meaning that we end up with the linear system $\mathbf{Jh} = -\mathbf{r}$, where $\mathbf{J}$ is the Jacobian matrix of the nonlinear system of equations.

Then, instead of solving the entire linear system $\mathbf{Jh} = -\mathbf{r}$ using some kind of solver as in chapter 4, a multigrid method, like the one described in section 5.1, could be applied to this linear system. This approach is called Newton-multigrid (Newton-MG).

As beforementioned, much research is carried out on solution methods using global linearization. Nonlinear multigrid also enables us to deal directly with the nonlinear systems using the FAS algorithm. As described in section 1.4, this is a relatively unexplored area in reservoir simulation and with recent trends in many-core parallel architectures, the locality of the method makes it interesting. The fact that global linearization can be avoided results in significant memory savings, since the Jacobian matrix does not need to be assembled for the finest grid level. Furthermore, the FAS rate of convergence is not contrained by the convergence rate of Newton's method, [9, Brandt, p. 84].

The FAS algorithm exploits the fact that the two main components of multigrid, error smoothing and coarse grid correction, are applicable to nonlinear problems, [43, Trottenberg, p.148].

5.2.2 FAS theory

As described in [28, Henson, p.3], the FAS method computes a coarse grid correction term based on the residual equation in (5.18), which using the error relation $\mathbf{u} = \mathbf{v} + \mathbf{e}$ can be rewritten as

$$\mathbf{A}\left(\mathbf{v} + \mathbf{e}\right) - \mathbf{A}\left(\mathbf{v}\right) = \mathbf{r} \tag{5.19}$$

Assuming uniform grids, consider this equation on the coarsest grid with mesh size $H = 2h$ (h being the mesh size of the finer grid one grid level up)

$$\mathbf{A}_H\left(\mathbf{v}_H + \mathbf{e}_H\right) - \mathbf{A}_H\left(\mathbf{v}_H\right) = \mathbf{r}_H \tag{5.20}$$

The coarse grid residual $\mathbf{r}_H$ is the restriction of the fine grid residual, meaning

$$\mathbf{r}_H = I_h^H \mathbf{r}_h = I_h^H\left(\mathbf{f}_h - \mathbf{A}_h\left(\mathbf{v}_h\right)\right) \tag{5.21}$$

Similarly, the coarse grid approximation $\mathbf{v}_H$ is the restriction of the fine grid approximation $\mathbf{v}_h$. This is in contrast to the approach used in linear multigrid, where only the residual is restricted. Using these definitions we rewrite equation (5.20) into

$$\mathbf{A}_H \underbrace{\left(I_h^H \mathbf{v}_h + \mathbf{e}_H\right)}_{\mathbf{u}_H} = \underbrace{\mathbf{A}_H\left(I_h^H \mathbf{v}_h\right) + I_h^H\left(\mathbf{f}_h - \mathbf{A}_h\left(\mathbf{v}_h\right)\right)}_{\mathbf{f}_H} \tag{5.22}$$

Since the right-hand side of (5.22) consists of known terms, the solution $\mathbf{u}_H$ to this equation can be determined. Based on this solution, the coarse grid correction term is computed as $\mathbf{e}_H = \mathbf{u}_H - I_h^H \mathbf{v}_h$. This correction term is prolongated one grid level up, where it is used to correct the solution to the residual equation at that grid level, which again can be used to determine a correction term for an even finer grid, and etc. In the end we find an approximation to the nonlinear problem at the fine grid.

Following [43, Trottenberg, p.157], the nonlinear FAS cycle algorithm is presented in pseudo-code in Algorithm 5.4.

In its construction this algorithm is very similar to the linear multigrid cycle algorithm *MGCYC* presented in Algorithm 5.2. The differences between the two algorithms lies in lines 8-11 of the FAS Cycle pseudo-code, where the approximated solution is restricted to a coarser grid and used to compute a right-hand side for the residual equation at that grid level. Furthermore, a system of the type $\mathbf{A}_H(\mathbf{u}_H) = \mathbf{f}_H$ is solved at coarsest grid, and based on this solution a correction term is determined. This is in contrast to the linear case, where the correction term is directly obtained at the coarsest level by solving $\mathbf{A}_H\mathbf{e}_H = \mathbf{r}_H$.

Algorithm 5.4 FAS Cycle - $\mathbf{v}_h = \text{FAScycle}(\mathbf{v}_h, \mathbf{f}_h)$

```
1:  if on coarsest grid level then
2:      Solve $\mathbf{A}_h(\mathbf{v}_h) = \mathbf{f}_h$
3:      Return $\mathbf{v}_h$
4:  else
5:      Nonlinear pre-smoothings on $\mathbf{v}_h$
6:      Compute residual: $\mathbf{r}_h = \mathbf{f}_h - \mathbf{A}_h(\mathbf{v}_h)$
7:      Restrict residual: $\mathbf{r}_H = I_h^H \mathbf{r}_h$
8:      Restrict approximate solution $\mathbf{v}_h$: $\mathbf{v}_H = I_h^H \mathbf{v}_h$
9:      Compute right-hand side: $\mathbf{f}_H = \mathbf{r}_H + \mathbf{A}_H(\mathbf{v}_H)$
10:     Recursive call: $\mathbf{u}_H = \text{FAScycle}(\mathbf{v}_H, \mathbf{f}_H)$
11:     Compute correction: $\mathbf{e}_H = \mathbf{u}_H - \mathbf{v}_H$
12:     Prolongate correction: $\mathbf{e}_h = I_H^h \mathbf{e}_H$
13:     Correction: $\mathbf{v}_h = \mathbf{v}_h + \mathbf{e}_h$
14:     Nonlinear post-smoothings on $\mathbf{v}_h$
15: end if
```

With the theory of FAS being introduced we move on to the implementation phase.

5.2.3 Getting familiar with FAS

In order to get acquainted with the FAS method before applying it to the more advanced oil reservoir problem, we consider the nonlinear problem presented in [28, Henson, p.6]

$$\begin{aligned} -\nabla^2 u(x,y) + \gamma\, u(x,y)\, e^{u(x,y)} &= f(x,y) && \in \Omega \\ u(x,y) &= 0 && \in \partial\Omega \\ \gamma &\geq 0 && \in \mathbb{R} \end{aligned} \tag{5.23}$$

Except for the nonlinear term, this is the same equation as in (5.3). We assume that the exact solution is given by

$$u(x,y) = (x - x^2)(y - y^2), \tag{5.24}$$

which gives us the righthand-side

$$f(x,y) = 2((x - x^2) + (y - y^2)) + \gamma(x - x^2)(y - y^2)e^{(x-x^2)(y-y^2)} \tag{5.25}$$

By solving this problem, where the exact solution is known, we can verify our implementation and thereby our understanding of the FAS method before applying it to a more advanced problem. Since solving this problem is done to gain knowledge about the method, and performance is not essential, the implementation of the FAS method is done in MATLAB. The implementation is in accordance with Algorithm 5.4 and follows the approach in [28, Henson], meaning that we use the same nonlinear Gauss-Seidel smoother and the same restriction and prolongation stencils. By using the same approach and setup for the problem as in the article, we expect to observe similar trends in the results. The nonlinear Gauss-Seidel smoother is described in section 5.3.1.

We use a regularly structured, vertex centered grid with fine grid spacing $h = \frac{1}{128}$, meaning that the fine grid has 129×129 nodes. The coarse grid problem is solved on a 3×3 coarse grid using the smoothing function. At the fine grid, the solution is accepted when the residual is smaller than 1e-10. We study the performance of the FAS solver for varying values of γ, starting with the linear case $\gamma = 0$ and ending with $\gamma = 10000$, where the nonlinear term dominates. The results are seen in Table 5.1.

γ	0	1	10	100	1000	10000
Convergence factor	0.064	0.062	0.054	0.035	0.019	0.005
Number of FAS cycles	10	10	10	9	8	7

Table 5.1: *Performance of the FAS method for the problem in* (5.23) *for various* γ.

We observe that the FAS solver is able to solve the problem for each of the test cases in the table. We also see that the problem becomes easier to solve the more nonlinear it is. This seems intuitively wrong, but the same trend is observed in [28, Henson, p.8], where a similar study is conducted. Even though we do not experience the exact same results as in [28, Henson, p.8], we are satisfied with the results, since the trend is the same. A possible reason for the results to slightly deviate from the ones presented in the article is that we use a different number of pre- and post-smoothings and smoothings at coarsest level.

The convergence factor in Table 5.1 is an average convergence factor over the number of FAS cycles, and it is calculated according to [43, Trottenberg, p54]

$$\text{Average convergence factor} = \sqrt[m]{\frac{||r_h^m||}{||r_h^0||}}, \tag{5.26}$$

where r_h^0 is the initial fine grid residual and r_h^m is the final fine grid residual.

Based on the above results we conclude that our implementation is correct and our understanding of the method sufficient to move on to the reservoir problem.

5.2.4 The reservoir model equations

In our introduction to the FAS method, we have used nonlinear systems of equations of the form in (5.16). The reservoir model equations can be formulated in the same nonlinear setting as given in (4.12).

5.3 Implementation

In this section, details and results for our first C++ implementation of the FAS method are presented. Some of the strategies used in this first version are not optimal with respect to algorithmic and numerical efficiency, but the essential part is to get the FAS simulator up and running before further optimization. Suggestions for improvement of the algorithm are implemented and tested in Chapter 6. This presentation approach is chosen to reflect the actual work process.

Algorithm 4.1 in section 4.3 illustrates how for each time step the reservoir problem is solved using Newton's method. We now substitute the function call to the *newton* routine in line 2 with a function call to the *FAS* routine seen in Algorithm 5.5.

Algorithm 5.5 FAS method - $\mathbf{x}_h^{n+1} = \text{FAS}(\mathbf{x}_h^n, \Delta t)$

```
1: while stop = false or kFAS < kFAS_max do
2:   Single V-cycle, $\mathbf{x}_h = \text{FASCycle}(\mathbf{x}_h, \mathbf{0}_h)$
3:   if Newton's method in FASCycle did not converge then
4:     break
5:   end if
6:   Compute flows
7:   Compute residuals
8:   if stop criteria are met then
9:     stop = true
10:  end if
11:  kFAS = kFAS + 1
12: end while
13: if kFAS $\geq$ kFAS_max or Newton's method in FASCycle did not converge then
14:   Chop time step to $\Delta t = \Delta t/2$
15:   Recursive call to FAS
16: end if
```

Provided the fine grid solution $\mathbf{x}_h^n$ to the current time step n, the *FAS* routine computes the fine grid solution to the next time step $n+1$. This is essentially done by updating the current solution a number of times using the *FASCycle* routine, until the stopping criteria are met. The cycle-type used in the *FASCycle* routine is a V-cycle, and the implementation is done in accordance with Algorithm 5.6. The stopping criteria are the same as used for the *newton* routine in section 4.3.5.

If Newton's method in the *FASCycle* routine does not converge within the allowed number of iterations, then exit the while-loop in the *FAS* routine, chop the time step and make a recursive call to *FAS*. If on the other hand, *FASCycle* finds an update to the solution, then check whether or not this new solution satisfies the stopping criteria. In order to do this, the residual is needed, meaning that the computations in line 6 and 7 of Algorithm 5.5 are necessary. The *residual* routine is the same as presented in section 4.3. The *flow* routine computes for each grid cell the total flow in or out of the cell. If the stopping criteria are satisfied, then accept current approximate solution, otherwise take one more iteration. If a valid solution is not found within the allowed number of cycles, then reduce the time step and call *FAS* recursively.

Algorithm 5.6 FAS cycle - $\mathbf{x}_h = \text{FASCycle}(\mathbf{x}_h, \mathbf{f}_h)$

```
 1: if on coarsest grid level then
 2:     Solve $\mathbf{A}_h(\mathbf{x}_h) = \mathbf{f}_h$ using Newton's method, $\mathbf{x}_h = \text{newton}(\mathbf{x}_h)$
 3: else
 4:     Nonlinear pre-smoothings on $\mathbf{x}_h$
 5:     Compute flow
 6:     Compute residual: $\mathbf{r}_h = \mathbf{f}_h - \mathbf{A}_h(\mathbf{x}_h)$
 7:     Restrict residual: $\mathbf{r}_H = I_h^H \mathbf{r}_h$
 8:     Restrict primary variables: $\mathbf{x}_H = I_h^H \mathbf{x}_h$
 9:     Compute properties based on $\mathbf{x}_H$
10:     Compute flow based on $\mathbf{x}_H$
11:     Compute $\mathbf{A}_H(\mathbf{x}_H)$ using residual routine
12:     Compute right-hand side: $\mathbf{f}_H = \mathbf{r}_H + \mathbf{A}_H(\mathbf{x}_H)$
13:     Recursive call: $\tilde{\mathbf{x}}_H = \text{FASCycle}(\mathbf{x}_H, \mathbf{f}_H)$
14:     Compute correction: $\mathbf{e}_H = \tilde{\mathbf{x}}_H - \mathbf{x}_H$
15:     Prolongate correction: $\mathbf{e}_h = I_H^h \mathbf{e}_H$
16:     Correction: $\mathbf{x}_h = \mathbf{x}_h + \mathbf{e}_h$
17:     Nonlinear post-smoothings on $\mathbf{x}_h$
18: end if
```

The *FASCycle* routine in Algorithm 5.6 follows the theory presented in section 5.2.2. The coarsest grid problem is solved using the *newton* routine presented in Algorithm 4.2. The *properties* routine is the same as presented in section 4.3. In lines 9 and 10, the properties and the flow at the coarser grid with grid spacing H are computed. Using this it is possible to apply the *residual* routine to compute $\mathbf{A}_H(\mathbf{x}_H)$.

The ideas behind the *properties* routine, *flow* routine and *residual* routine have already been introduced. The remaining parts of Algorithm 5.6 are presented below.

Notice that if the total number of grid levels equals one, then the *FAS* algorithm reduces to the *newton* routine presented in section 4.3.

As described in section 4.3.6, our time step size controller generates the size of the next time step based on the numbers k and k_{max}. For the standard newton simulator, presented in chapter 4, the value k_{max} is defined as the maximum number of newton iterations allowed per time step. The value k is the number of newton iterations used for the current time step. However, the FAS simulator also takes the maximum number of FAS cycles per time step, $k\text{FAS}_{max}$, and the number of FAS cycles used for the current time step into consideration. Hence, for the FAS simulator

$$k_{max} = \min(k\text{Newton}_{max}, k\text{FAS}_{max}) \text{ and } k = \max(k\text{Newton}, k\text{FAS}) \tag{5.27}$$

5.3.1 Smoother

The smoothing technique applied at this stage of the work is based on the pointwise nonlinear Gauss-Seidel method. This method is chosen in favour of the nonlinear Jacobi method, because of its higher convergence rate and smaller memory requirement, [43, Trottenberg, p.29].

In section 5.1.1 we presented the linear pointwise Gauss-Seidel method for a simple scalar problem. However, since we are now considering a nonlinear problem we have to use Newton's method when computing the grid point update. Hence, the method is known as the Gauss-Seidel-Newton (GSN) method, [43, Trottenberg, p.152]. Furthermore, we are now considering a system of equations instead of a scalar problem. A natural way to generalize scalar smoothing schemes to systems is to use collective relaxation, meaning that all unknowns at each single grid cell are updated simultaneously, [43, Trottenberg, p.289]. This implies that we relax each grid cell at a time by

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{h}_i, \quad i \in \mathcal{C}, \tag{5.28}$$

where $\mathbf{h}_i$ is determined by solving the linear system

$$\mathbf{J}_i\mathbf{h}_i = -\mathbf{r}_i, \quad i \in \mathcal{C}, \tag{5.29}$$

for $\mathcal{C} = \{1, ..., N\}$. Here $\mathbf{x}_i$ refers to the current solution for a given cell, meaning

$$\mathbf{x}_i = \begin{bmatrix} m_{o,i} & m_{g,i} & m_{w,i} & p_i \end{bmatrix}^T, \quad i \in \mathcal{C}, \tag{5.30}$$

and $\mathbf{r}_i$ refers to the corresponding residuals. The matrix $\mathbf{J}_i$ refers to the ith diagonal block in the full Jacobian matrix for the current grid level, meaning that $\mathbf{J}_i$ is a 4×4 matrix containing the elements seen in (4.15) for the diagonal block matrix. The reason for not considering the off-diagonal block matrices is that we consider a pointwise GSN method (point smoother), where only the derivatives of cell i wrt. cell i are computed. Had it been a line smoother, meaning that all grid cells in either x-, y- or z-direction are relaxed at a time, the off-diagonal blocks in the full Jacobian matrix should have been taken into consideration.

The GSN point smoother is implemented in accordance with Algorithm 5.7.

Algorithm 5.7 GSN point smoother - $\mathbf{x} = \text{smoother}(\mathbf{x})$

```
1: Compute properties
2: while $k <$ Number of smoothings do
3:     for All cells in current grid do
4:         Compute $\mathbf{J}_i$
5:         Compute $\mathbf{r}_i$
6:         Solve $\mathbf{J}_i\mathbf{h}_i = -\mathbf{r}_i$
7:         Update solution $\mathbf{x}_i = \mathbf{x}_i + \mathbf{h}_i$
8:         Update properties for cell $i$.
9:     end for
10:    $k = k + 1$
11: end while
```

The *smoother* routine performs a user-specified number of smoothings, where in each smoothing sweep, each cell is updated one at a time. For each cell we take only a single newton iteration, since we do not have to solve the system in line 6 of Algorithm 5.7 to any particular accuracy. The reason for this is that we only want to smooth the error, not eliminate it, and this is therefore done with minimal effort, [28, Henson, p.7].

When solving the linear system in line 6 of Algorithm 5.7 we use the MUMPS solver presented in section 4.3.4. To use a direct library solver developed for solving large sparse systems when solving a small 4×4 system is not an optimal strategy, since the time spent by the solver on analyzing the matrix pattern and choosing the optimal solving method introduces a lot of overhead and unnecessary work. With this said, we still use MUMPS, because it is a straightforward solution, since the framework is already up and running in the *newton* routine. This is without doubt a part of the code, where there is room for improvement.

As illustrated in section 5.1.1 for the linear scalar case, it is possible to use the Gauss-Seidel method as solver. However, due to the poor final convergence rate it is necessary to use many iterations to reach a reasonable accuracy. Even though this fact speaks against using the collective GSN method as the coarsest grid solver, we can still use the approach to verify our implementation of the method.

We solve the $1 \times 1 \times 3$ homogeneous problem presented in section 4.4.1 for a period of 150 days with an initial time step of 0.1 days and $\Delta t_{max} = 0.15$ days. At each time step 10000 GSN iterations are used in order to reach the required accuracy. The results are seen in Figure 5.6 below.

Since the *smoother* algorithm reaches a result comparable to the result ECLIPSE finds, we conclude that the implementation is correct. It has to be mentioned that the simulation illustrated in Figure 5.6 took a very long time to run, meaning that there is no practical perspective in using the *smoother* algorithm in its current state as the coarsest grid solver.

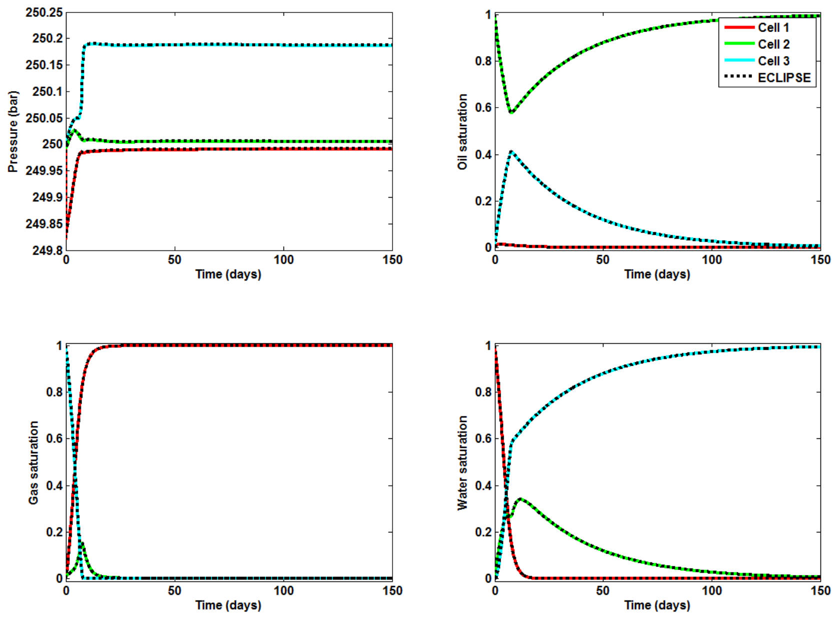


Figure 5.6: *Solution to* $1 \times 1 \times 3$ *homogeneous case generated by the GSN smoother, using 10000 smoothings at each time step.*

As it will be demonstrated later in section 7.2.1, the smoother makes up a large portion of the computation time. Furthermore, a nonlinear Gauss-Seidel smoother is local in that for each cell, the smoothing procedure is only based on information from its six neighbours. This makes it a good match for many-core architectures, where good scalability depends on reducing the number of global reductions and inner products or essentially any process that involves communication between grid cells in opposite sides of the domain. A massively scalable GPU implementation of a multigrid solver has been demonstrated in [21, Engsig-Karup] and in [1, Adams], a scalable Gauss-Seidel smoother for distributed systems is presented.

5.3.2 Restriction

The restriction stencil is a (x, y)-semicoarsening stencil, meaning that it only restricts along the x- and y-direction. The reason for this is that we consider gravity inversion test cases, leading to all the dynamics of the system happening along the z-direction. We therefore do not coarsen in the z-direction in order to preserve as much information as possible in this direction. The restriction stencil is given by

$$I_h^H = w \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}_h^H \tag{5.31}$$

meaning that four fine grid cells are restricted into one coarse grid cell, like it was the case in (5.11). This gives us a refinement factor of two in both the x- and y- direction. The weight w equals $\frac{1}{4}$ when restricting the primary variables, $\mathbf{x}$, and 1 when restricting the residual, $\mathbf{r}$. The reason for this distinction is that the primary variables are per volume, whereas the residual is for a specific volume. For the residual this means that we have to sum ($w = 1$) the four fine grid contributions in order to get the restricted residual for the coarse grid. Otherwise the coarse grid cell does not represent the same volume as the four fine grid cells, [48, Zhou].

5.3.3 Prolongation

The prolongation stencil is given by

$$I_H^h = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}_H^h, \tag{5.32}$$

meaning that the information held in the coarse grid cell is prolongated to the four surrounding fine grid cells. This is the same stencil as in (5.12).

5.4 Verification

The implementation of the *FAS* routine is verified in the same way as the *newton* routine was verified in section 4.4. The same test cases are used, except for the $1 \times 1 \times 3$ case, which is now a $2 \times 2 \times 3$ case. Once again ECLIPSE is used for comparison.

5.4.1 Simple homogeneous permeability field

The $2 \times 2 \times 3$ homogeneous test case, which is initialized as in Figure 4.4, is solved for 150 days. The results are depicted in Figure 5.7.

The FAS simulator using two grid levels, FAS(2), finds the same result as ECLIPSE. The small differences are due to the two simulators having different time stepping methods. This can be verified by forcing both simulators to take small identical time steps. Figure 5.8 shows the oil saturation if both ECLIPSE and FAS(2) integrates in time with 0.1 days per time step.

As depicted in Figure 5.8, the small differences seen in Figure 5.7 are not observed anymore. The same behaviour applies to the solution of the three other primary variables.

For all the test cases in this section we use two grid levels, 0.1 days as initial time step, 20 pre- and post-smoothings and $\Delta t_{max} = 30$ days. No more than 10 FAS cycles and 10 Newton iterations for the coarsest grid solver are allowed. Furthermore, the initial pressure at all cells is 250 bar.

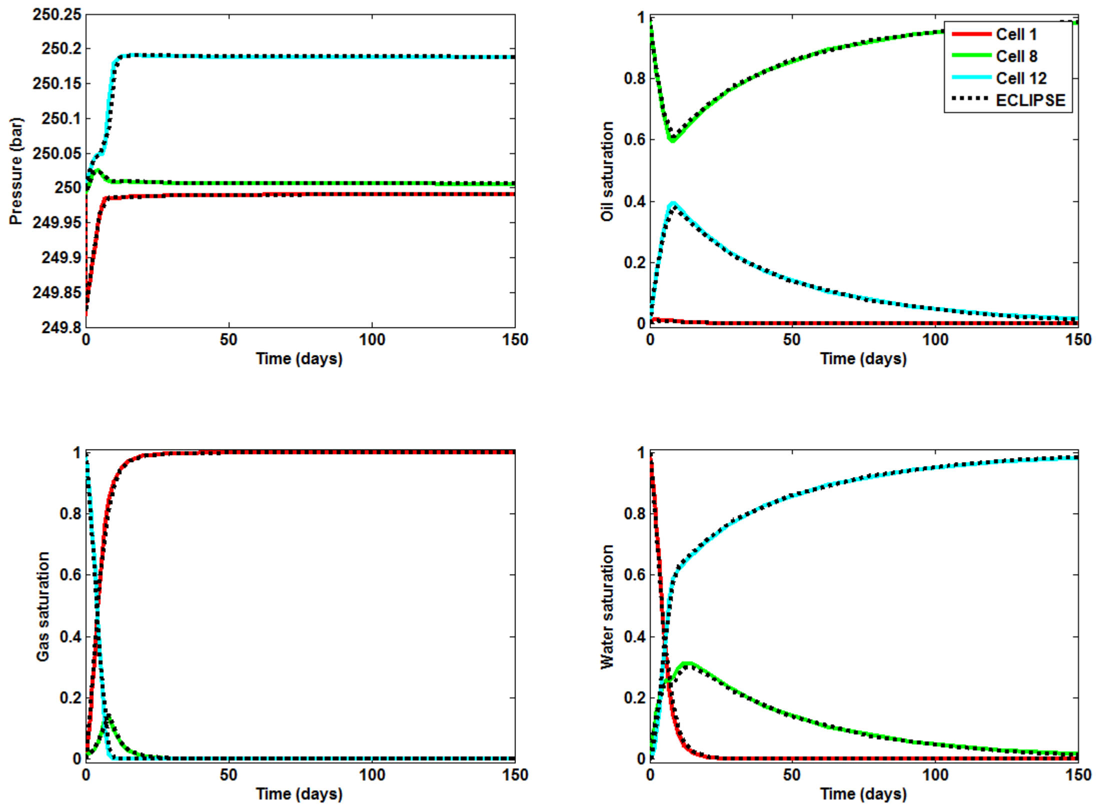


Figure 5.7: $2 \times 2 \times 3$, *150 days, 2 grid levels.*

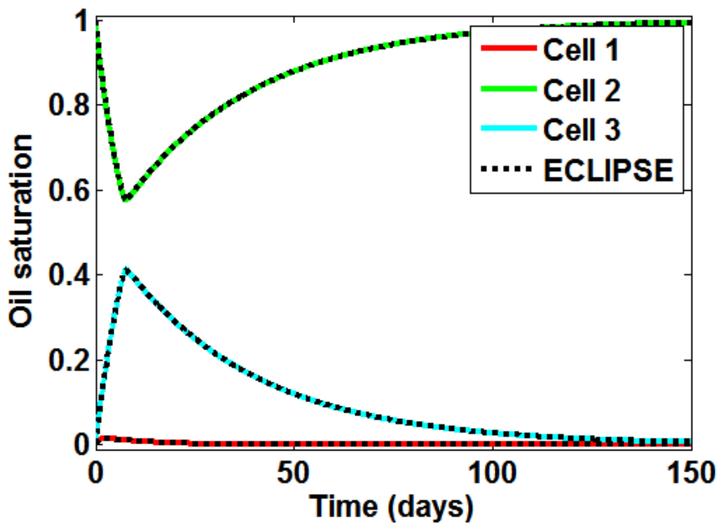


Figure 5.8: $2 \times 2 \times 3$, *150 days, 2 grid levels, Oil saturation, Fixed time steps of* 0.1 *days.*

5.4.2 Smooth heterogeneous permeability field

The next test case is the smooth heterogeneous permeability field presented in section 4.4.2, which has permeabilities between 200-600 mD. The problem is initialized as in Figure 4.7. The problem size is $18 \times 18 \times 18$ and the simulation horizon is 100 days.

When starting this simulation, we immediately observe that the simulator is very slow. In fact, it is slow beyond what is practically acceptable. A profiling using Valgrind identifies the linear solver in the smoother as the problem. Section 6.1 explains how the problem is identified and solved. After having replaced the linear solver in the smoother the numerical efficiency of the algorithm has increased significantly. The results of the simulation are in Figure 5.9.

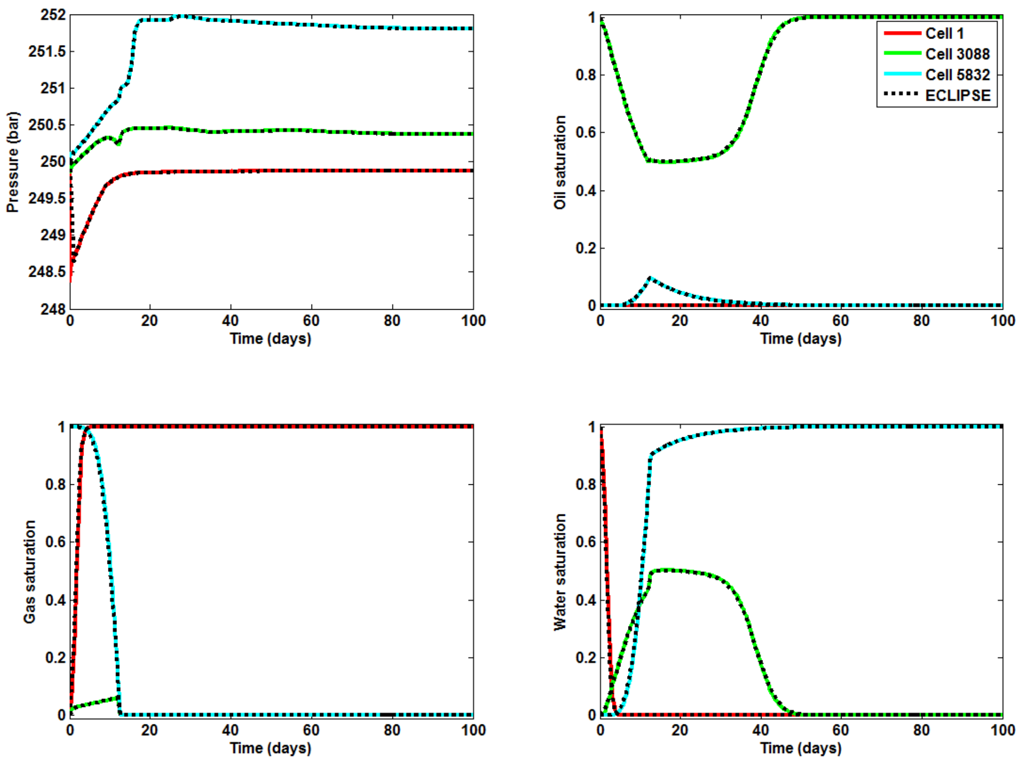


Figure 5.9: $18 \times 18 \times 18$, *100 days, 2 grid levels.*

The FAS(2) simulator finds the same results as ECLIPSE. Similar results for the FAS(3) and the FAS(4) simulators are found in Appendix F.

5.4.3 SPE10 derived heterogeneous permeability field

The final test case is the SPE10 derived heterogeneous permeability field presented in section 4.4.3. The problem is initialized as in Figure 4.10. The problem size is $10 \times 10 \times 10$

and the simulation horizon is 16000 ≈ 44 years. The results are in Figure 5.10.

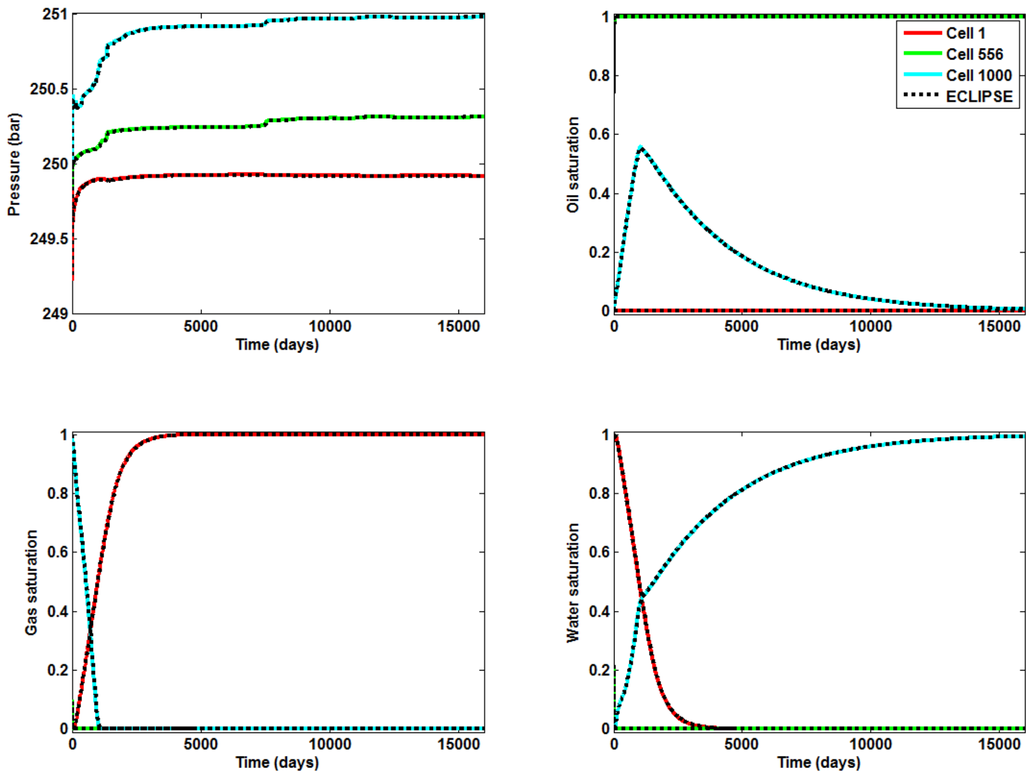


Figure 5.10: $10 \times 10 \times 10$, *16000 days, 2 grid levels.*

Once again the FAS(2) simulator finds the same results as ECLIPSE. Based on the test cases presented we conclude that the implementation of the FAS algorithm is correct.

5.5 Performance study

A small performance study of the FAS(2) simulator is now carried out. For comparison reasons this study is similar to the one for the standard newton simulator in section 4.6, using the same hardware specifications and input values as in that section.

We study the run time as a function of the number of grid cells for a reservoir with a fixed size of $480 \times 240 \times 48$ meters. The results seen in Figure 5.11 clearly illustrate the potential of the FAS algorithm. The results are generated using FAS(2), where 20 pre-smoothings and 20 post-smoothings are used. The same absolute tolerances apply for both simulators. These can be seen in Appendix A.

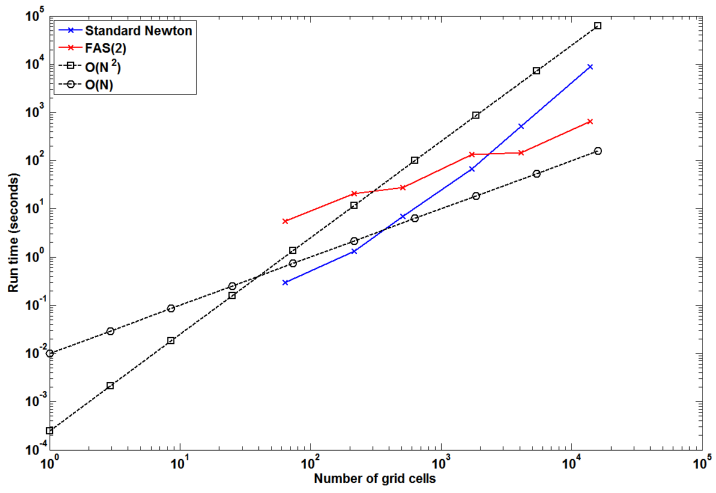


Figure 5.11: *Run time as a function of number of grid cells for the prototype FAS implementation.*

The figure shows that for small problem sizes there is no gain in using FAS(2). This seems intuitively correct, since the time saved in the linear solver due to a smaller problem size at coarsest grid is lost in the restriction, prolongation and especially the smoother function. However, after break even FAS(2) is found to be superior. The simulator based on FAS(2) appears to have linear scaling for these relatively small problem sizes, whereas the simulator based on global linearization with standard Newton's method is closer to $O(N^2)$. The scalability of the simulator based on standard newton directly reflects the poor scalability of the sparse direct solver MUMPS. Because the FAS(2) simulator restricts the problem size, the point at which poor scalability becomes evident is postponed. A more comprehensive performance study is conducted in chapter 7.

By using Valgrind to profile the above simulation study for FAS(2), the results in Figure 5.12 are produced.

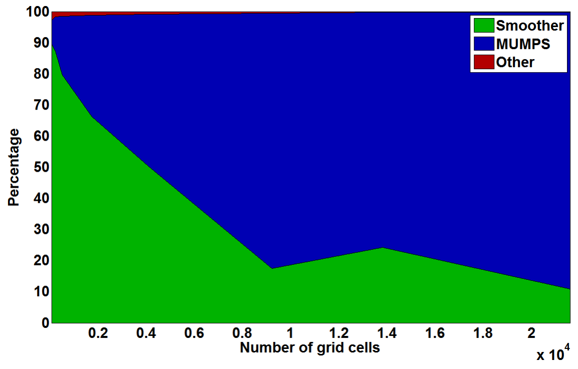


Figure 5.12: *Distribution of computation time for the FAS(2) simulator for various problem sizes.*

The figure illustrates that a significant amount of the computation time is used on error smoothing. As previously discussed, point smoothing operations are local and therefore well-suited for modern many-core architectures.

However, as the problem size increases, the general trend is that the linear solver at coarsest grid makes up more and more of the computation time. This indicates that it might be beneficial to use more grid levels as the problem size increases in order to keep the main workload in the smoother.

Furthermore, Figure 5.12 indicates that there is no potential in using MUMPS as coarsest grid solver if we want to solve very large problems due to the poor scaling. The largest problem considered in the figure is relatively small (21,600 cells), but MUMPS still makes up about 90% of the computation time when using two grid levels. As mentioned, we could use more grid levels, but also a faster linear solver would be beneficial. An improvement of this kind is described in section 6.3.

5.6 Summary

A first multiphase reservoir simulator based on the FAS method has been implemented and verified. In a small performance study, this simulator has been compared to the newton based simulator presented in chapter 4. The results show that the FAS simulator is able to perform better than the newton based simulator. Furthermore, the performance study shows a need for a faster linear solver, since we would like to keep the main workload in the highly parallelizable parts of the code, even for large problems. A more extensive performance study is carried out in chapter 7.

Chapter 6

Improvements

The starting point for this chapter is the prototype implementation of the FAS algorithm presented in section 5.3. In this chapter we suggest different changes to the implementation in order to improve algorithmic and numerical efficiency of the FAS simulator. The chapter reflects the actual work process, meaning some of the suggestions do in fact provide improvements and some do not.

In section 6.1, the Gaussian elimination approach replaces MUMPS as linear solver in the Gauss-Seidel-Newton point smoother. Section 6.2 proposes line smoothing as an alternative to point smoothing, and in section 6.3 the direct linear solver MUMPS is replaced as the coarsest grid solver by an iterative solver from the PETSc library. The effect of applying the Full Multigrid (FMG) approach in FAS is considered in section 6.4, and section 6.5 proposes alternative ways of restricting the cell-based input data. Finally, in section 6.6 the two-stage preconditioner CPR is presented and implemented, which enables us to apply a linear solver based on state-of-the-art choice of methods.

6.1 Removing MUMPS from the smoother

As mentioned in section 5.4.2, we cannot solve the $18 \times 18 \times 18$ smooth heterogeneous permeability field problem within reasonable time. Using the profiling tool Valgrind, we can identify which parts of the code that take the longest time to execute. We profile the simulation of the $2 \times 2 \times 3$ homogeneous permeability field problem described in section 5.4. The problem is initialized as usual with gas in the bottom layer, oil in the middle layer and water in the top layer. The simulation is run for 150 days, using two grid levels, 0.1 days as initial step, $\Delta t_{max} = 30$ days and no more than 10 FAS cycles and newton iterations. We use 20 pre-smoothings and 20 post-smoothings.

The profiling of this simulation run shows that the *smoother* routine makes up 98.7%

of the overall computation time. The time spent by the various subroutines of the *smoother* routine are distributed as in Figure 6.1 below. This figure illustrates that the linear solver, meaning MUMPS, is the critical part of the *smoother* routine. In fact, it makes up about 93% of the overall computation time and roughly 95% of the computation time used in *smoother*. In section 5.3.1 we mentioned our concerns about applying a direct linear solver developed for large sparse systems to the small 4×4 system that is solved in the smoother, and evidently it is a poor choice.

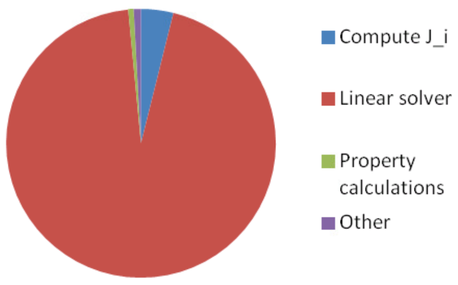


Figure 6.1: *Distribution of time spent in the smoother when using MUMPS as linear solver for a $2 \times 2 \times 3$ problem.*

By implementing a linear solver ourselves, we save the time spent by MUMPS on analyzing the system and the time wasted due to overhead of the MUMPS library routines. We choose to solve the linear system using Gaussian elimination, since the implementation is rather straightforward. Even though this type of algorithm has complexity $O(n^3)$ for dense systems, [35, LeVeque, p.67], it seems suitable as a linear solver in the *smoother* routine, since we only solve systems of the size 4×4.

Due to stability reasons the implementation of the Gaussian elimination algorithm uses partial pivoting, [20, Eldén, p.199], meaning that the pivot element of the column currently considered is the element with the largest absolute value in this column.

The implementation of the Gaussian elimination algorithm follows Algorithm 6.1.

Algorithm 6.1 Gaussian elimination - $\mathbf{r_i} = \text{GaussianElimination}(\mathbf{J}_i, \mathbf{r}_i, n)$

```
1: for $j = 1, .., n$ in $\mathbf{J}$ matrix do
2:     Consider rows $k = j + 1, .., n$ and swap rows, such that the row with largest
       absolute value in the $j$th column becomes row $j$
3:     Swap same rows in $\mathbf{r}$
4:     Eliminate entries in column $j$ below row $j$ using elementary row operations
5:     Apply same elementary row operations to $\mathbf{r}$
6: end for
7: for $j = n, .., 1$ in $\mathbf{J}$ matrix do
8:     Eliminate entries in column $j$ above row $j$ using elementary row operations
9:     Apply same elementary row operations to $\mathbf{r}$
10: end for
```

Note that the routine reduces the augmented matrix to reduced row echelon form. It might be faster to reduce only to row echelon form and then apply backward substitution, but by reducing all the way to reduced row echelon form the same approach can be applied for matrix inversion, which we need in section 6.2. We solve for the system $\mathbf{J}_i\mathbf{h}_i = \mathbf{r}_i$ presented in section 5.3.1. The parameter n is the size of the square matrix $\mathbf{J}_i$, which in our case always equals 4.

Notice that $\mathbf{r}_i$ holds the residual when the algorithm is initialized and the solution $\mathbf{h}_i$ when the algorithm terminates.

We remove MUMPS as the linear solver and apply the *GaussianElimination* routine instead. When rerunning the above simulation study, we immediately observe that the FAS simulator has become much faster. In fact, it is about 25 times faster than before for the homogeneous $2\times2\times3$ problem. The smoother still takes up a relatively large part of the computation time, 85.5%, but this is okay since the basis for this measurement has been reduced significantly. The time spent in the *smoother* routine is now distributed as illustrated in Figure 6.2.

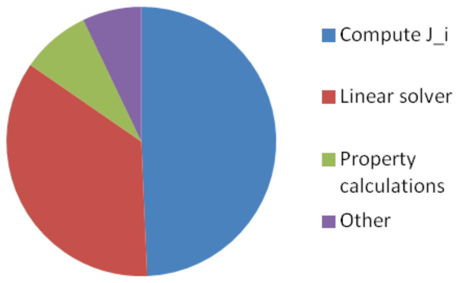


Figure 6.2: *Distribution of time spent in the smoother when using Gaussian-Elimination as linear solver for a $2 \times 2 \times 3$ problem.*

Regarding computation time, the linear solver now constitutes a much smaller part of the *smoother* routine than before when using MUMPS.

6.2 Line smoother

All the different oil reservoir test cases considered so far have been anisotropic problems. Even though we use isotropic permeability fields as mentioned in section 2.1, anisotropy is introduced into the problems through anisotropic cell sizes, which affect the transmissibility in (3.29). With anisotropic cell sizes we mean that the dimension of the cells are not the same in the x-, y- and z-direction.

The reason for considering anisotropic cell sizes is that a high grid resolution along the z-direction better represents the gravitational effect, [16, Dogru, p.6], and since oil fields are usually much larger in the x- and y-directions than in the z-direction, (see SPE10 and [16], [17], [18]) it is inevitable to use anisotropic cell sizes.

When a problem is very anisotropic, it is not possible to reach a satisfactory convergence factor using standard coarsening and point wise smoothing techniques, [43, Trottenberg, p.132]. Trottenberg et al. [43, p.133] suggests two approaches to deal with this, namely semicoarsening and stronger coupled smoothing techniques.

As mentioned in section 5.3.2, we are already using (x, y)-semicoarsening in order to preserve as much information as possible in the z-direction. The reason is that we consider gravity inversion test cases, meaning that nearly all of the dynamics in the system are happening along the z-direction. For the same reason we suggest using a linewise smoother in the z-direction in order to reach a higher convergence rate of the smoother routine.

The concept in a line smoother is to update all primary variables along a line simultaneously. In this way a stronger coupling of the primary variables along this line is ensured. As mentioned we consider a z-line smoother, meaning that for all the grid cells in the xy-plane, the cells in the z-direction are updated collectively.

The implementation of the *linesmoother* routine is very similar to the point smoother routine *smoother* in Algorithm 5.7. However, the big difference is that the system $\mathbf{J}\mathbf{h} = -\mathbf{r}$ solved in the Newton step in line 6 of Algorithm 5.7 is no longer a 4×4 system. Since all the grid cells along the z-axis are updated simultaneously, we now consider the block tridiagonal system $\mathbf{J}_{line}\mathbf{h}_{line} = -\mathbf{r}_{line}$ given by

$$\overbrace{\begin{bmatrix} \mathbf{B}_1 & \mathbf{C}_1 & & & \mathbf{0} \\ \mathbf{A}_2 & \mathbf{B}_2 & \mathbf{C}_2 & & \\ & \mathbf{A}_3 & \mathbf{B}_3 & \ddots & \\ & & \ddots & \ddots & \mathbf{C}_{Nz-1} \\ \mathbf{0} & & & \mathbf{A}_{Nz} & \mathbf{B}_{Nz} \end{bmatrix}}^{\mathbf{J}_{line}} \overbrace{\begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_{Nz-1} \\ \mathbf{h}_{Nz} \end{bmatrix}}^{\mathbf{h}_{line}} = - \overbrace{\begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_{Nz-1} \\ \mathbf{r}_{Nz} \end{bmatrix}}^{\mathbf{r}_{line}}, \tag{6.1}$$

where the matrices $\mathbf{B}_1, ..., \mathbf{B}_{Nz}$ are the 4×4 diagonal matrices presented in (4.15) and the matrices $\mathbf{A}_2, ..., \mathbf{A}_{Nz}$ and $\mathbf{C}_1, ..., \mathbf{C}_{Nz-1}$ are the 4×4 off-diagonal matrices presented in (4.15). The 4×1 vectors $\mathbf{r}_1, ..., \mathbf{r}_{Nz}$ contains the corresponding residuals. Nz is the number of grid cells in the z-direction. Note that all the matrices and vectors in (6.1) use local indices, not global.

This block tridiagonal system is solved using the block version of the tridiagonal matrix algorithm, also known as the Thomas algorithm.

The *linesmoother* routine is not restricted to be a z-line smoother. Depending on the input the routine can also be applied as a x-line smoother or a y-line smoother.

6.2.1 The Thomas algorithm

The Thomas algorithm is an algorithm for solving tridiagonal systems of equations. The algorithm is a simplified version of the Gaussian elimination algorithm and it has complexity $O(n)$, which is better than $O(n^3)$ for dense Gaussian elimination, [49, Zikanov, p.139].

The Thomas algorithm for solving a scalar PDE is presented in [41, Thomas, p. 87]. The algorithm consists of a forward sweep and a backward sweep, and the scalar version can be extended to a block version. The implementation of the block Thomas algorithm follows Algorithm 6.2, which is in accordance with [41, Thomas, p. 295].

Algorithm 6.2 Block Thomas - $\mathbf{h}_{line} = \text{Thomas}(\mathbf{J}_{line}, \mathbf{r}_{line}, Nz)$

```
 1: Compute $\mathbf{H}_1 = -\mathbf{B}_1^{-1}\mathbf{C}_1$
 2: Compute $\mathbf{g}_1 = \mathbf{B}_1^{-1}\mathbf{r}_1$
 3: for $j = 2, ..., Nz - 1$ do
 4:     Compute $\mathbf{H}_j = -\left(\mathbf{B}_j + \mathbf{A}_j\mathbf{H}_{j-1}\right)^{-1}\mathbf{C}_j$
 5:     Compute $\mathbf{g}_j = \left(\mathbf{B}_j + \mathbf{A}_j\mathbf{H}_{j-1}\right)^{-1}\left(\mathbf{r}_j - \mathbf{A}_j\mathbf{g}_{j-1}\right)$
 6: end for
 7: Compute $\mathbf{g}_{Nz} = \left(\mathbf{B}_{Nz} + \mathbf{A}_{Nz}\mathbf{H}_{Nz-1}\right)^{-1}\left(\mathbf{r}_{Nz} - \mathbf{A}_{Nz}\mathbf{g}_{Nz-1}\right)$
 8: Compute $\mathbf{h}_{Nz} = \mathbf{g}_{Nz}$
 9: for $j = Nz - 1, ..., 1$ do
10:     $\mathbf{h}_j = \mathbf{g}_j + \mathbf{H}_j\mathbf{h}_{j+1}$
11: end for
```

Note that in the implementation of the block Thomas algorithm we store the solution in $\mathbf{r}_{line}$ in order to reuse the memory. The matrix invertions are done using the Gaussian elimination approach described in section 6.1.

6.2.2 Performance of line smoother

A small performance study comparing the efficiency of the point smoother routine, *smoother*, to the line smoother routine, *linesmoother*, is now conducted.

In order to compare the algorithmic efficiency for the two methods, we consider how the residual for a single time step is reduced as a function of the number of FAS cycles. For both methods we consider the first time step of the simulation for the $18 \times 18 \times 18$ problem with smooth heterogeneous permeability field. The simulations are run using an initial step of 0.1 days and the input parameters in Appendix A. Hardware specification 2 in Appendix B is used. We use the anisotropic cell size $20m \times 10m \times 2m$.

For both the point smoother and the line smoother we use 5 pre-smoothings and post-smoothings and allow no more than 100 FAS cycles and 10 Newton iterations. The results are seen in Figure 6.3 below.

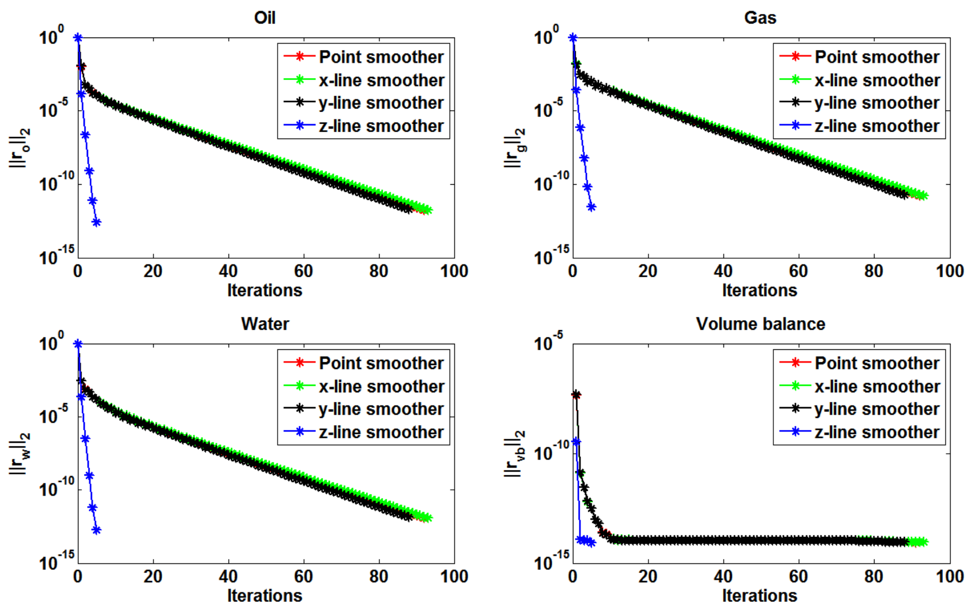


Figure 6.3: *Number of FAS cycles used to reduce the residual to a certain tolerance for FAS(2) for a* $18 \times 18 \times 18$ *problem.*

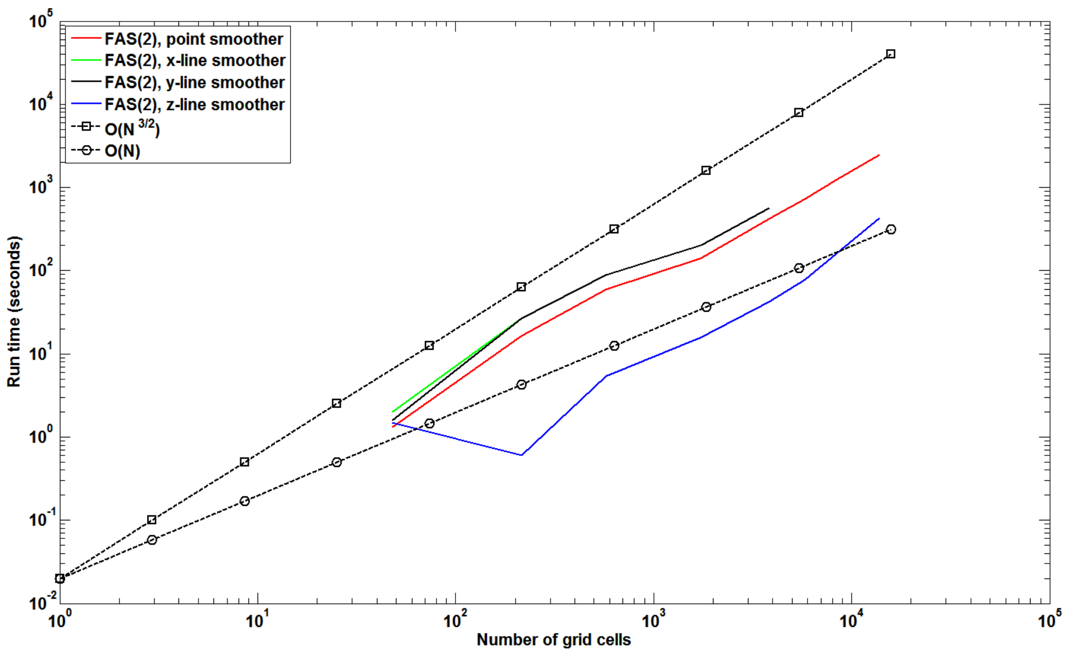


Figure 6.4: *Run time as function of problem size for FAS(2) using different smoothing strategies.*

The figure shows that for the given problem, the z-line smoother uses much fewer FAS cycles to reach a certain tolerance than the point smoother (The point smoother curve is almost hidden behind the green and black curves). This indicates that the z-line smoother is algorithmically more efficient than the point smoother, which is in line with theory. Furthermore, the figure shows that applying a x-line smoother or y-line smoother does not improve the algorithmic efficiency.

Even though the algorithmic efficiency of the FAS simulator is improved by using the line smoother, this might not be the case in terms of numerical efficiency. The reason is that the systems solved in the line smoother are larger than the systems solved in the point smoother. We investigate this by computing the simulation time for various problem sizes of the smooth heterogeneous permeability field problem. The cell sizes are fixed at $20m \times 10m \times 2m$. The simulation runs for 150 days, initial time step is 0.1 days, $\Delta t_{max} = 30$ days and the input parameters in Appendix A are applied. Hardware specification 2 in Appendix B is used.

The problems are solved to the same tolerance, and again we use 5 pre- and post-smoothings and allow no more than 10 FAS cycles and 10 newton iterations. The results are displayed in Figure 6.4.

This figure shows that even though the system solved in the line smoother is larger than the system solved in the point smoother, the z-line smoother is superior with respect to numerical efficiency. The reason is that the line smoother needs much fewer FAS cycles to reach a certain accuracy than the point smoother. This is not the case for the x-line and y-line smoothers, which have almost the same algorithmic efficiency as the point smoother according to Figure 6.3. The x-line and y-line smoothers perform worse than the point smoother, since the systems solved in the line smoothers are larger.

Only 5 pre-smoothings and post-smoothings have been used for this test. This is a reasonable amount of smoothings for the z-line smoother, but both algorithmic and numerical efficiency of the point smoother could be increased by using more smoothings. However, our experiences with the FAS simulator shows that the z-line smoother is always superior for the given test case no matter the number of pre- and post-smoothings used for the point smoother.

The scalability of the simulator is not limited by the point or line smoothers. In the point smoother, Gaussian elimination on N small 4×4 problems scales linearly. In the line smoother, the Thomas algorithm also has linear scalability. The limiting factor is still the sparse direct solver MUMPS for the coarsest grid solve. The outliers in Figure 6.4 can be caused by the fact that a permeability field perhaps favours the z-line smoother. Also the problem sizes are relatively small, which introduces larger relative measurement errors.

One has to bear in mind that the above results are for a single time step of a given gravity inversion test case using (x, y)-semicoarsening, where a strong connection in the z-direction is introduced due to the cell size $20m \times 10m \times 2m$. It is possible that the

z-line smoother would not be the optimal choice if we consider a flooding[1] test case with most of the dynamics along the xy-plane.

The superiority of the z-line smoother is also affected if we change the cell dimensions and thereby the anisotropy of the problem. This is illustrated by conducting the exact same test as in Figure 6.3, but using a different cell size. If isotropic cells with the size $10m \times 10m \times 10m$ are considered, the results in Figure 6.5 are obtained.

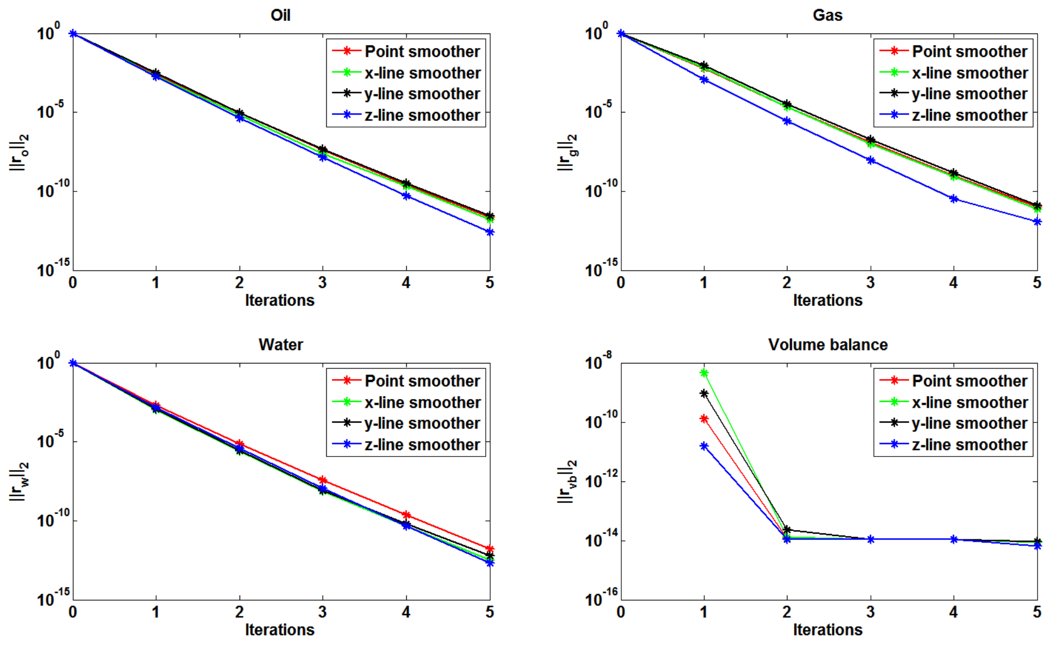


Figure 6.5: *Number of iterations used to reduce the residual to a certain tolerance for FAS(2) using cell size* $10m \times 10m \times 10m$.

This figure clearly shows, that the difference between the line smoothers and the point smoother reduces as the anisotropy of the problem decreases. In fact, the z-line smoother is only slightly better than the other smoothers. This is as expected, since our reason for considering a line smoother is the high anisotropy of the problem.

If the cell size is changed to $2m \times 10m \times 10m$, meaning that a relatively strong connection is made between the cells in the x-direction, the results are as in Figure 6.6. In this figure, it is illustrated that the x-line smoother is now the optimal choice.

All in all, the above results indicate that it is worth implementing a line smoother when considering an anisotropic problem. However, the line smoothing strategy should be chosen with care, since parameters such as cell size has large effect on the performance. Furthermore, the problem at hand should be taken into consideration.

[1] By flooding we mean that water is injected into the reservoir, leading the oil to the production wells.

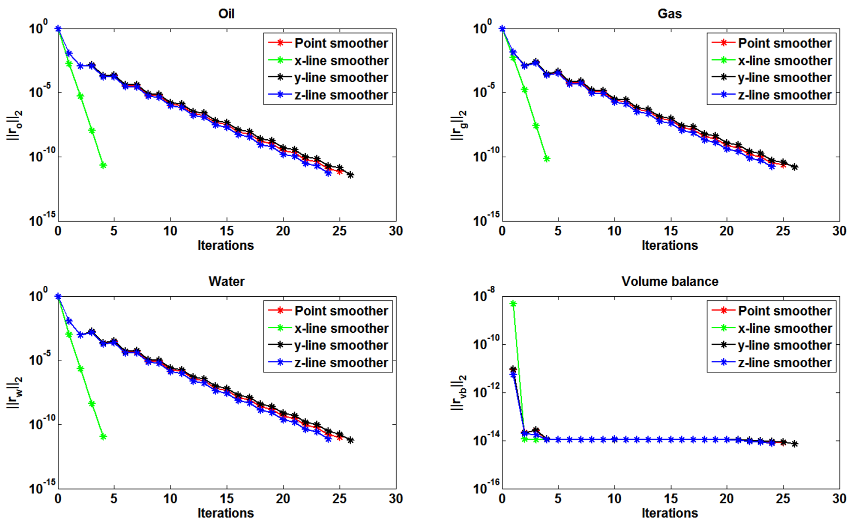


Figure 6.6: *Number of iterations used to reduce the residual to a certain tolerance for FAS(2) using cell size* $2m \times 10m \times 10m$.

Throughout the remainder of this thesis, we use the z-line smoother unless otherwise stated, since it performs very well for cell sizes giving a strong connection in the z-direction.

Note that a line smoother is less local than a point smoother, meaning a line smoother is less suitable for efficient implementation on modern many-core architectures than a point smoother. However, the Thomas algorithm is $O(n)$ and if there is enough work so that the threads can be kept busy, an implementation of a line smoother is still scalable. This is demonstrated in [21, Engsig-Karup]. Furthermore, the line smoother provides significantly better convergence rates.

6.3 Iterative linear solver

In this section, we describe how the solver for the linear systems on the coarsest grid is improved. We replace the direct solver MUMPS with an iterative solver. Specifically, the iterative solver method GMRES with ILU preconditioning is implemented using the PETSc[2] library.

In section 6.3.1, a brief overview of the commonly used linear solvers in reservoir simulation is given. Section 6.3.2 gives a basic introduction to GMRES and Arnoldi's method. Since the PETSc library is used to implement the iterative linear solver, only a basic theoretical introduction is given.

For reservoir simulation, GMRES is useless without good preconditioning. Section 6.3.3

[2] http://www.mcs.anl.gov/petsc/.

and 6.3.4 introduce Incomplete LU preconditioning and how it is used with GMRES. Section 6.3.5 provides some insight to implementing the iterative solver and preconditioner in Newton's method using PETSc. Lastly, the improvement from switching to an iterative linear solver instead of a direct solver is documented in section 6.3.6.

6.3.1 Introduction

Figure 5.12 reveals that using a sparse direct linear solver (in this case MUMPS) to solve the linear system in each newton iteration on the coarsest grid is prohibitive. Furthermore, an iterative linear solver will facilitate realistic comparisons between the standard newton approach with a global linearization and FAS.

For large three-dimensional reservoir simulation problems, iterative linear solvers to solve the nonsymmetric linear system $\mathbf{Jh} = -\mathbf{r}$ or more generally

$$\mathbf{Ax} = \mathbf{b} \tag{6.2}$$

are the only viable option, [6, Aziz, p.338]. An iterative linear solver is a procedure that, given an initial solution $\mathbf{x}^0$, finds the solution by successive approximations

$$\mathbf{x}^0 \to \mathbf{x}^1 \to \mathbf{x}^2 \ldots \to \mathbf{x}^k \ldots, \tag{6.3}$$

where $\mathbf{x}^k \to \mathbf{x}$ when $k \to \infty$. The simplest iterative methods are the stationary methods such as the Jacobi method, the Gauss-Seidel method or the Successive Over-Relaxation method (SOR). As shown in section 5.3.1, the Gauss-Seidel method is not efficient enough as stand-alone solver and from [43, Trottenberg, p. 52] we know that this applies to the Jacobi method and the SOR method as well.

The most popular linear solver methods for reservoir simulation are the Krylov subspace methods. These include the Conjugate Gradient method applied to the normal equations (CGN), the Generalized Minimal Residual method (GMRES) and the Biconjugate Gradient Stabilized method (BiCGSTAB) [12, Chen, p. 220]. Previously, ORTHOMIN, still used in the commercial simulator ECLIPSE, was the leading method. Currently, the most popular method is GMRES. Studies have shown that GMRES performs better than ORTHOMIN, since ORTHOMIN in general requires more arithmetic operations and storage than GMRES [12, Chen, p. 224].

GMRES is known to be a very efficient and robust method for solving general sparse nonsymmetric systems. Due to its wide usage in reservoir simulation, GMRES is also used in our simulator as the coarsest grid linear solver. In the following, we will give a basic introduction to GMRES. For a more detailed explanation see [38, Saad].

6.3.2 GMRES

The following description of the GMRES method follows the approach in [35, LeVeque, p.96]. Consider the linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{6.4}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$. In each iteration k of the GMRES method, an approximation $\mathbf{x}^k$ to the solution of (6.4) is determined by solving a least squares problem. The approximation $\mathbf{x}^k$ is chosen as the best approximation from the affine space $\mathbf{x}^0 + \mathcal{K}^k$, where $\mathcal{K}^k$ is the k-dimensional Krylov space

$$\mathcal{K}^k = \operatorname{span}\left(\mathbf{r}^0, \mathbf{A}\mathbf{r}^0, \mathbf{A}^2\mathbf{r}^0, ..., \mathbf{A}^{k-1}\mathbf{r}^0\right), \tag{6.5}$$

which is based on the initial residual $\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0$. Note that superscripts on $\mathbf{r}$ indicate the iteration number and superscripts on $\mathbf{A}$ indicate the power. In order to formulate this least squares problem we have to build a matrix

$$\mathbf{Q}^k = \begin{bmatrix}\mathbf{q}^1 & \mathbf{q}^2 & \cdots & \mathbf{q}^k\end{bmatrix} \in \mathbb{R}^{n\times k}, \tag{6.6}$$

whose columns form an orthonormal basis for the Krylov space $\mathcal{K}^k$. This orthonormal basis is computed using Arnoldi's Method, [5, Arnoldi].

Arnoldi's method applied to GMRES

Arnoldi's method is an orthogonal projection method onto the Krylov subspace $\mathcal{K}^k$. It was discovered that it can be used to efficiently find eigenvalues for large general non-Hermitian matrices. Later, the method was extended to solve large sparse linear systems, [38, Saad, p.160].

In iteration k of the Arnoldi method, the vector $\mathbf{q}^{k+1}$ is determined using a Gram-Schmidt type procedure. This is done by taking some vector $\mathbf{v}^k$, which is not in $\mathcal{K}^k$ and orthogonalizing it to $\mathbf{q}^1$, $\mathbf{q}^2$,...,$\mathbf{q}^k$. The vector $\mathbf{v}^k$ is chosen as $\mathbf{v}^k = \mathbf{A}\mathbf{q}^k$, since $\mathbf{q}^k$ is orthogonal to all the previous basis vectors, meaning that $\mathbf{v}^k = \mathbf{A}\mathbf{q}^k$ is not likely to be in $\mathcal{K}^k$. The first of the basis vectors is chosen as $\mathbf{q}^1 = \mathbf{r}^0/||\mathbf{r}^0||$.

The GMRES algorithm is outlined in pseudo code in Algorithm 6.3 below, where the first 9 lines constitute the basic Arnoldi method.

In the kth and $(k+1)$th iteration of this algorithm we have

$$\mathbf{Q}^k = \begin{bmatrix}\mathbf{q}^1 & \mathbf{q}^2 & \cdots & \mathbf{q}^k\end{bmatrix} \in \mathbb{R}^{n\times k} \quad \text{and} \quad \mathbf{Q}^{k+1} = \begin{bmatrix}\mathbf{Q}^k & \mathbf{q}^{k+1}\end{bmatrix} \in \mathbb{R}^{n\times(k+1)}, \tag{6.7}$$

which forms an orthonormal basis for $\mathcal{K}^k$ and $\mathcal{K}^{k+1}$, respectively.

Algorithm 6.3 GMRES method - $\mathbf{x} = \text{GMRES}(\mathbf{A}, \mathbf{x}^0, \mathbf{r}^0)$

```
1: Choose vector $\mathbf{q}^1 = \mathbf{r}^0/||\mathbf{r}^0||$
2: for $k = 1, 2, \dots$ do
3:    $\mathbf{v}^k = \mathbf{A}\mathbf{q}^k$
4:    for $i = 1, 2, \dots, k$ do
5:       Compute $h_{ik} = (\mathbf{q}^i)^T\mathbf{v}^k$
6:       Compute $\mathbf{v}^k = \mathbf{v}^k - h_{ik}\mathbf{q}^i$
7:    end for
8:    $h_{k+1,k} = ||\mathbf{v}^k||_2$
9:    $\mathbf{q}^{k+1} = \mathbf{v}^k/h_{k+1,k}$
10:   Compute $\tilde{\mathbf{H}}^k$
11:   Check residual $\mathbf{r}^k$ of least squares problem in (6.21)
12:   if $\mathbf{r}^k$ sufficiently small then
13:      Compute $\mathbf{x}$ based on (6.16)
14:      return $\mathbf{x}$
15:   end if
16: end for
```

All the h values computed after k iterations, except for $h_{k+1,k}$, are used to form an upper Hessenberg matrix

$$\mathbf{H}^k = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \cdots & h_{1,k-1} & h_{1,k} \\ h_{2,1} & h_{2,2} & h_{2,3} & \cdots & h_{2,k-1} & h_{2,k} \\ & h_{3,2} & h_{3,3} & \cdots & h_{3,k-1} & h_{3,k} \\ & & \ddots & \ddots & & \vdots \\ & & & & h_{k,k-1} & h_{k,k} \end{bmatrix}, \quad \in \mathbb{R}^{k\times k} \tag{6.8}$$

Based on (6.8) we define the matrix $\tilde{\mathbf{H}}^k$ as

$$\tilde{\mathbf{H}}^k = \begin{bmatrix} \mathbf{H}^k \\ \mathbf{h}^k \end{bmatrix}, \quad \in \mathbb{R}^{(k+1)\times k}, \tag{6.9}$$

where $\mathbf{h}^k$ is the row vector

$$\mathbf{h}^k = [0,\ 0,\ \dots,\ 0,\ h_{k+1,k}], \quad \in \mathbb{R}^{1\times k} \tag{6.10}$$

Note that the value $h_{k+1,k}$ is found in the kth iteration of the GMRES algorithm.

Consider the matrix product

$$\mathbf{A}\mathbf{Q}^k = \left[\mathbf{A}\mathbf{q}^1,\ \mathbf{A}\mathbf{q}^2,\ \dots,\ \mathbf{A}\mathbf{q}^k\right] \quad \in \mathbb{R}^{n\times k}, \tag{6.11}$$

and notice that for $j = 1, \dots, k$ the jth column of this matrix is equal to the starting vector $\mathbf{v}^j$ in the jth iteration of Algorithm 6.3. The jth column of (6.11) is alternatively expressed as

$$h_{j+1,j}\mathbf{q}^{j+1} = \mathbf{A}\mathbf{q}^j - h_{1,j}\mathbf{q}^1 - h_{2,j}\mathbf{q}^2 - \ldots - h_{j,j}\mathbf{q}^j, \tag{6.12}$$

which can be rearranged into

$$\mathbf{A}\mathbf{q}^j = h_{1,j}\mathbf{q}^1 + h_{2,j}\mathbf{q}^2 + \ldots + h_{j,j}\mathbf{q}^j + h_{j+1,j}\mathbf{q}^{j+1} \tag{6.13}$$

The equation in (6.12) is obtained by considering the starting vector in line 3 of Algorithm 6.3 and the computations done in lines 4-9. Primarily notice the computations in lines 6 and 9.

The left-hand side of (6.13) is the jth column of $\mathbf{A}\mathbf{Q}^k$, and for $j < k$ the right-hand side represents the jth column of $\mathbf{Q}^k\mathbf{H}^k$. This means that

$$\mathbf{A}\mathbf{Q}^k = \mathbf{Q}^k\mathbf{H}^k + h_{k+1,k}\mathbf{q}^{k+1}(\mathbf{e}^k)^T, \tag{6.14}$$

where $(\mathbf{e}^k)^T$ is a unit vector of length k with a 1 in the last element. The last term in (6.14) corresponds to the last term in (6.13) for $j = k$. Using the definitions in (6.7) and (6.9) we rewrite (6.14)

$$\mathbf{A}\mathbf{Q}^k = \mathbf{Q}^{k+1}\tilde{\mathbf{H}}^k \tag{6.15}$$

The least squares problem

We are now ready to define the least squares problem for finding an approximate solution $\mathbf{x}^k$ to the problem in (6.4). We know that the columns in $\mathbf{Q}^k$ form a basis for $\mathcal{K}^k$, meaning that $\mathbf{x}^k$ can be written as

$$\mathbf{x}^k = \mathbf{x}^0 + \mathbf{Q}^k\mathbf{y}^k, \tag{6.16}$$

for some vector $\mathbf{y}^k \in \mathbb{R}^k$. This means that the residual at iteration k is

$$\begin{aligned}\mathbf{r}^k &= \mathbf{b} - \mathbf{A}\left(\mathbf{x}^0 + \mathbf{Q}^k\mathbf{y}^k\right)\\ &= \mathbf{r}^0 - \mathbf{A}\mathbf{Q}^k\mathbf{y}^k\\ &= \mathbf{r}^0 - \mathbf{Q}^{k+1}\tilde{\mathbf{H}}^k\mathbf{y}^k,\end{aligned} \tag{6.17}$$

where we have used (6.15). Since the first column in $\mathbf{Q}^{k+1}$is defined as $\mathbf{q}^1 = \mathbf{r}^0/||\mathbf{r}^0||$, we rewrite the initial residual into $\mathbf{r}^0 = \mathbf{Q}^{k+1}\mathbf{w}$, where the vector $\mathbf{w}$ is

$$\mathbf{w} = \Big[\,||\mathbf{r}^0||_2,\ 0,\ \ldots,\ 0\,\Big]^T, \quad \in \mathbb{R}^{k+1} \tag{6.18}$$

Using this we rewrite (6.17)

$$\mathbf{r}^k = \mathbf{Q}^{k+1}\left(\mathbf{w} - \tilde{\mathbf{H}}^k\mathbf{y}^k\right) \tag{6.19}$$

Since the columns in $\mathbf{Q}^{k+1}$ form an orthonormal basis, it holds that $(\mathbf{Q}^{k+1})^T\mathbf{Q}^{k+1} = \mathbf{I}$. Using the definition of the dot products of vectors, computing $(\mathbf{r}^k)^T\mathbf{r}^k$ leads to

$$||\mathbf{r}^k||_2 = ||\mathbf{w} - \tilde{\mathbf{H}}^k\mathbf{y}^k||_2 \tag{6.20}$$

In the kth iteration of the GMRES method we choose $\mathbf{y}^k$ to solve the least squares problem

$$\min_{\mathbf{y}\in\mathbb{R}^k} ||\mathbf{w} - \tilde{\mathbf{H}}^k\mathbf{y}||_2, \tag{6.21}$$

and based on this we find the solution $\mathbf{x}^k$ as in (6.16).

Remarks

The Arnoldi method considered in this section is the basic Arnoldi procedure, which assumes exact arithmetic. In practice, Arnoldi's method is improved with a modified Gram-Schmidt procedure to obtain a much more reliable implementation in case of round-off. Even Arnoldi's method with modified Gram-Schmidt can be insufficient. To deal with this, a double orthogonalization procedure can be implemented. Another option is to use a different orthogonalization technique called the Householder Algorithm. These improvements are described in [38, Saad, p. 163].

If the Arnoldi method uses all $k = n$ iterations (n being the dimension of $\mathbf{A}$), then the starting vector $\mathbf{v}^n = \mathbf{A}\mathbf{q}^n$ lies in the Krylov space $\mathcal{K}^n$, which then will be all of $\mathbb{R}^n$, [35, LeVeque, p.98]. This means that GMRES is guaranteed to find the exact solution, at least in theory. However, the idea is that after a small amount of iterations, a sufficient approximation is found.

If only a small number of iterations is used, meaning $k << n$, then $\tilde{\mathbf{H}}^k$ and $\mathbf{w}^k$ are small, leading to the least squares problem being small. This least squares problem is relatively cheap to solve using a QR factorization of $\tilde{\mathbf{H}}^k$, since this matrix is nearly upper triangular. Furthermore, $\tilde{\mathbf{H}}^k$ consists of $\tilde{\mathbf{H}}^{k-1}$ with one additional row and column added. By reusing the factorization of $\tilde{\mathbf{H}}^{k-1}$ (which has already been QR factorized in the previous iteration) it is cheap to compute the factorization of $\tilde{\mathbf{H}}^k$, [35, LeVeque, p.99].

According to LeVeque, it is possible to compute the residual of (6.21) without actually solving for $\mathbf{y}^k$, using the $\mathbf{R}$ matrix from the QR factorization. In this way, the convergence criterion can be checked at each iteration k without having to solve for $\mathbf{y}^k$. This means that the final $\mathbf{y}^k$ and $\mathbf{x}^k$ are computed only after the convergence criterion is satisfied, [35, LeVeque, p.99].

Algorithm 6.3 describes the very basic GMRES method. It requires storing both the $\mathbf{q}^i$'s and $\mathbf{v}^k$'s. A modification can be implemented, where Arnoldi's method is replaced with the Householder method. In this way, the GMRES algorithm can be altered to only require storage of the $\mathbf{v}^k$'s. See [38, Saad, p. 174] for specific information on how to do this.

Even with modifications as the Householder method, the immediate concern of the GMRES method is the growing storage requirement. If GMRES needs a high number of iterations to converge, the memory requirement could outgrow the capacity. GMRES is guaranteed to converge in at most n iterations, however this would become impractical if the problem is large and many steps are required. In practice, GMRES is restarted after a given number of iterations m. This is done by setting $\mathbf{x}^0 = \mathbf{x}^m$ and restarting the algorithm. Restarting GMRES can introduce difficulties with stagnation. To overcome these difficulties, preconditioning is applied to lower the required number of GMRES iterations.

For further information about Arnoldi's method and GMRES see [38, Saad, p.160] and [38, Saad, p.171].

6.3.3 Preconditioning

For more complicated problems, where the linear systems come from linearizing strongly heterogeneous and nonlinear systems, applying iterative linear solvers without preconditioning results in slowly converging or even non-robust solvers. In reality, the choice of preconditioner has greater impact on performance than the Krylov method that it is used with.

Preconditioning is a technique, which transforms the original linear system into a different linear system that is likely to be easier to solve with an iterative solver, but where the solution remains the same as the one for the original system. There are three ways of performing preconditioning. The three techniques are left-preconditioning as in (6.22)

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \tag{6.22}$$

or right-preconditioning as in (6.23)

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{u} = \mathbf{b}, \tag{6.23}$$

where $\mathbf{x} \equiv \mathbf{M}^{-1}\mathbf{u}$ or split-preconditioning as in (6.24)

$$\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\mathbf{u} = \mathbf{M}_L^{-1}\mathbf{b}, \tag{6.24}$$

where $\mathbf{x} \equiv \mathbf{M}_R^{-1}\mathbf{u}$. Here $\mathbf{M}$ is a preconditioning matrix. It can be generated in many ways, but common for all of the approaches is that it should result in $\mathbf{M}$ being non-singular, close to A and in practice reduce the condition number of the resulting linear system. Applying preconditioning to GMRES can be done with all three techniques above. One very significant difference is that right preconditioning allows for the Flexible Generalized Minimal Residual method (FGMRES), where the preconditioner can be changed in each iteration [12, Chen, p. 229].

Some of the simplest methods used for preconditioning are the stationary methods: Jacobi, the Gauss-Seidel and SOR. These can only be applied with limited success in reservoir simulation [12, Chen, p. 231]. More powerful preconditioning techniques are based on LU factorizations. In the following we will describe the basics of Incomplete LU factorization.

6.3.4 Incomplete LU factorization

ILU factorization can be performed by applying Gaussian elimination and dropping some elements in predetermined nondiagonal positions. Common for all types of ILU factorizations is that they generate a sparse lower triangular matrix **L** and a sparse upper triangular matrix **U** such that the residual matrix

$$\mathbf{R} = \mathbf{LU} - \mathbf{A} \tag{6.25}$$

satisfies certain conditions, such as having zero entries in some locations [38, Saad, p. 301].

The simplest ILU factorization is ILU(0), meaning ILU with no fill-in. It uses the exact same zero pattern as the matrix **A**. By introducing fill-in, namely ILU(p), $p = 1, \ldots$, where p is an indicator for the amount of fill-in, more efficient and reliable ILU factorizations can be obtained. This involves some additional challenges when constructing and implementing the methods. Some of these challenges are addressed in [38, Saad, p. 311].

6.3.5 Implementation

For our simulator, PETSc is built with the standard BLAS kernels provided through PETSc for hardware specification 1 in Appendix B. For hardware specification 2, the ATLAS[3] implementation is used. In the following, a brief overview is given of how PETSc is integrated into our simulator. Before starting the time integration, the necessary data structures and PETSc operators are initialized in a setup phase. This is described in Code 6.1.

Code 6.1: *PETSc setup*

```
PetscInitialize()
VecCreate(sol)
PetscObjectSetName(sol,"Solution")
```

3) http://math-atlas.sourceforge.net/

```
VecSetSizes(sol,4*N) // N is number of coarsest grid cells
VecDuplicate(sol,b) // Vector sol is duplicated to rhs vector b
MatCreateSeqAIJ() // Create matrix
KSPCreate() // Create solver context
```

When creating the matrix, an average number of non-zero elements per row is specified. In our implementation, this is chosen to 14, since this is the highest possible number of non-zeros in a row for our Jacobian. Specifying a too low average number of non-zero elements per row results in the first matrix assembly in the first newton iteration to be very slow. During the time integration, PETSc functions are continously called in Newton's method. Code 6.2 gives an idea as to how the framework is integrated.

Code 6.2: *PETSc implementation in Newton's method*

```
For each newton iteration
    MatSetValue(A) // Function is called in loop to cache elements put in A
    MatAssemblyBegin() // Assemble A
    MatAssemblyEnd()
    VecSetValues(b) // Define rhs
    VecAssemblyBegin() // Assemble rhs
    VecAssemblyEnd()
    KSPSetOperators() // Give it solver context and system
    KSPGetPC() // Define preconditioner
    PCSetType() // Set ILU as preconditioner
    KSPSetTolerances() // Set tolerances for linear solver
    KSPSolve() // Solve linear system
```

The tolerances for the linear solver are, unless a given experiment demands otherwise, set to 10^{-10} for the absolute tolerance and 10^{-5} for the relative tolerance. The function *MatSetValue* can be replaced with *MatSetValues* to cache more than one element at a time. This would probably increase efficiency of assembling the matrix. However, the function constructing the sparse Jacobian is designed for coordinate list format (COO) instead of compressed sparse row format (CSR), which means *MatSetValues* cannot be used without converting to CSR format or changing the method of constructing Jacobian matrices. Both options would require extra computational work. For optimal performance, more sophisticated assembling procedures can be applied.

After completing the time integration, PETSc data structures and operators need to be destroyed. This is straightforward as demonstrated in Code 6.3.

Code 6.3: *PETSc destroy*

```
VecDestroy(sol) // Destroy solution, rhs and matrix
VecDestroy(b)
MatDestroy(A)
KSPDestroy() // Destroy linear solver context
PetscFinalize()
```

Keep in mind this is just a brief overview of the most essential functions. The real implementation contains all the necessary function calls and arguments.

6.3.6 Improvement

We conduct a small performance study, where the performance of the FAS(2) simulator using MUMPS as linear solver is compared to the performance of the FAS(2) simulator using the iterative linear solver GMRES with ILU(1) preconditioning from the PETSc library. The performance study is similar to the studies in sections 4.6 and 5.5.

We consider a fixed reservoir of size $480m \times 240m \times 48m$. The reservoir has a smooth heterogeneous permeability field with permeabilities in the range 200-600 mD. This permeability field is generated as described in Appendix E.

The simulation horizon is 150 days, the initial time step is 0.1 days and the maximum allowed time step is $\Delta t_{max} = 30$ days. We allow no more than 10 FAS cycles per time step and no more than 10 newton iterations per FAS cycle. We use 5 pre- and 5 post-smoothings. The input values in Appendix A and hardware specifications 2 in Appendix B are used.

GMRES is given a stack of 30, meaning that if GMRES has not converged within 30 iterations, it is restarted as described in section 6.3.2. We use GMRES with left preconditioning. Figure 6.7 shows the run time comparison for different problem sizes.

For small problems there is no advantage in using the iterative solver. MUMPS is in fact a little faster for smaller problems. However, for larger problems there is a significant gain in using GMRES with ILU preconditioning. When more than 10000 grid cells are used, the poor scaling of MUMPS is evident. Meanwhile, GMRES with ILU preconditioning is somewhere between $O(N)$ and $O(N^{3/2})$ (closer to $O(N^{3/2})$). GMRES scales linearly, however this is not the case for ILU(1) preconditioning, where fill-in is introduced.

Figure 6.8 depicts profiling of the FAS(2) simulator with the GMRES-ILU solver for the coarsest grid solve for different problem sizes.

Similarly to the sparse direct solver MUMPS, the iterative solver GMRES-ILU also makes up the majority of the computation time, when only two grid levels are used in FAS. The percentage of computation time spent in the GMRES-ILU solver increases as the problem size increases. In chapter 7, we investigate if it is possible to move the main workload to the more local smoother component by increasing the number of grid levels.

The above studies show that the iterative solver presented in this chapter is more efficient than MUMPS. From this point onwards, the GMRES-ILU solver is used as standard.

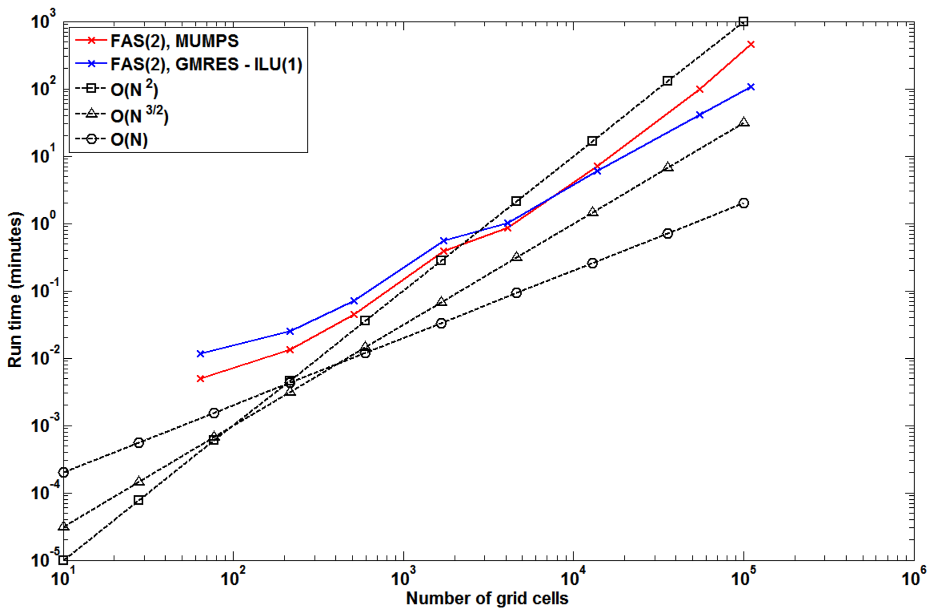


Figure 6.7: *Run time as a function of number of grid cells for the FAS implementation using either MUMPS or GMRES from PETSc.*

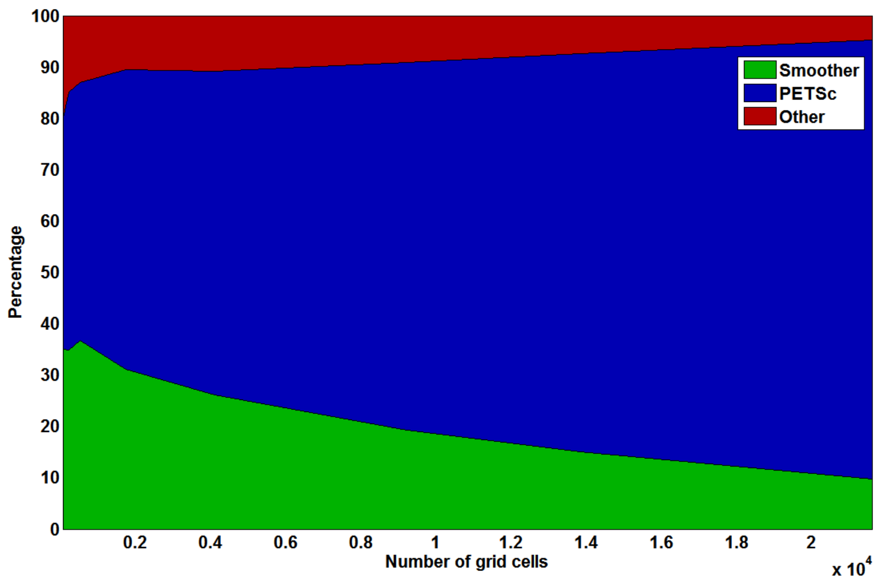


Figure 6.8: *Distribution of the computation time for the FAS(2) simulatior using GMRES for various problem sizes.*

6.4 Applying FMG

In section 5.1.3 different multigrid cycle types were presented. So far, only the V-cycle has been considered, but according to Trottenberg et al. the FMG scheme is often the most efficient multigrid version, [43, Trottenberg, p.56].

We consider a FMG V-cycle method, which is the FMG approach presented in section 5.1.3. Recall that the FMG method starts by solving the coarse grid problem and in the end produces an approximation to the fine grid problem. If this approximation does not satisfy the stopping criteria, additional V-cycles are carried out in order to make the approximation more accurate.

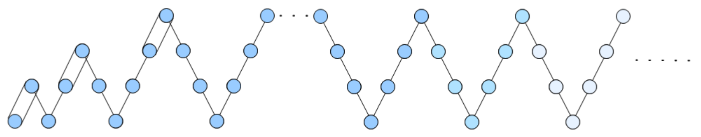


Figure 6.9: *A FMG V-cycle followed by one or more multigrid V-cycles.*

In our implementation at least one additional multigrid V-cycle, or FAS-cycle, is performed to reduce the residual below the tolerance. This approach is illustrated in Figure 6.9. The implementation of the FMG approach is in accordance with Algorithm 5.3, where the cycle routine call in line 6 is a call to the *FAScycle* routine presented in section 5.3. The additional V-cycles are carried out by calling the *FAS* routine described in section 5.3.

As initial guess for the coarsest grid problem considered in the beginning of the FMG approach, we use the restricted fine grid solution from the previous time step. This way of choosing the initial guess is illustrated in part a) of Figure 6.11 below. We will later consider other ways of generating the initial guess.

When applying the FMG method Trottenberg et al. suggest that the FMG-interpolation, which is the prolongation of the primary variables, uses an interpolation operator of higher accuracy than the prolongation operator used in the multigrid cycle, [43, Trottenberg, p.58]. Since we in the *FAScycle* routine use the 4-point prolongation stencil given in (5.32), we choose our FMG-interpolation as the 16 point interpolation operator given by

$$I_H^h = \frac{1}{16}\begin{bmatrix} 1 & 3 & 3 & 1 \\ 3 & 9 & 9 & 3 \\ 3 & 9 & 9 & 3 \\ 1 & 3 & 3 & 1 \end{bmatrix}_H^h \tag{6.26}$$

Note that the time step size controller seeks to optimize efficiency by taking the number of newton iterations used in the FMG approach into consideration. This means that the time step is not solely based on the FAS cycles and newton iterations in the additional

V-cycles, as it is the case without applying FMG. Hence, k and k_{max} in Algorithm 4.4 is defined as

$$\begin{aligned} k_{max} &= \min(k\text{Newton}_{max}, k\text{FAS}_{max}) \\ k &= \max(k\text{Newton}, k\text{FAS}, k\text{FMG}_{avg}), \end{aligned} \tag{6.27}$$

where $k\text{FMG}_{avg}$ is the average number of newton iterations used at the coarsest levels during the FMG approach.

We conduct a performance study to test whether or not applying FMG provides an improvement to the FAS simulator.

6.4.1 Improvement

Using various gravity inversion test cases we compare the performance of the FAS simulator without using FMG with the performance of the FAS simulator using FMG. For a given test case we measure the total number of time steps used and the run time of the simulation. Furthermore, we measure the average number of FAS cycles used per time step, meaning the number of V-cycles used per time step after the FMG approach.

By measuring the simulation run time we can immediately in practical tests observe how the numerical efficiency of the simulator is affected by introducing the FMG approach. The average number of FAS cycles used per time step is a direct indication of the algorithmic efficiency of the simulator. The total number of time steps indirectly provides information about the algorithmic efficiency. The reason is that the time step size controller takes the number of FAS cycles and the number of newton iterations used for previous time steps into account when computing the size of the next time step. Therefore, the time step size accepted by the time step controller is affected by the algorithmic efficiency.

We consider test cases with homogeneous permeability fields with permeabilities of 100 mD and test cases with smooth heterogeneous permeability fields with permeabilities in the range $200 - 600$ mD. The smooth heterogeneous permeability fields are generated as described in Appendix E. The test cases are initialized as described in section 4.4. We use the parameter settings listed in Appendix A and hardware specification 2 in Appendix B.

For all of the test cases we use a fixed choice of 5 pre- and 5 post smoothings, an initial time step of 0.1 days and $\Delta t_{max} = 30$ days. The simulation horizon is 150 days, and no more than 10 FAS-cycles and 10 newton iterations/FAS-cycle is allowed. We consider cells with the anisotropic cell size $20m \times 10m \times 2m$.

The results are listed in Table 6.1 below. All of the tests are solved to the same accuracy.

On the basis of the results in Table 6.1, it is difficult to conclude whether or not using FMG provides an improvement to the FAS simulator. For the $24 \times 24 \times 24$ homogeneous

Problem specifications	**Method**	**Apply FMG**	**Avg. no. of FAS cycles/time step**	**No. of time steps**	**Run time (secs)**
Homogeneous $24 \times 24 \times 24$	*FAS(2)*	-	1.4	32	81
		+	1	38	131
	FAS(3)	-	4.3	41	137
		+	1	36	70
	FAS(4)	-	4.7	36	117
		+	1	36	59
Homogeneous $32 \times 32 \times 32$	*FAS(2)*	-	1.4	45	333
		+	1.0	63	731
	FAS(3)	-	4.5	58	482
		+	1.1	73	405
	FAS(4)	-	4.7	65	508
		+	1.0	129	535
Smooth heterogeneous $24 \times 24 \times 24$	*FAS(2)*	-	1.7	91	366
		+	1	1021	5521
	FAS(3)	-	4.4	81	280
		+	1.0	1004	2256
	FAS(4)	-	4.7	98	317
		+	1.0	3322	5690

Table 6.1: *Comparisons of the FAS simulator using FMG and the FAS simulator without using FMG for various test cases.*

permeability field problem FMG improves the number of time steps and the run time when using three or four grid levels, but not when using two grid levels.

For the $32 \times 32 \times 32$ homogeneous permeability field problem, the only improvement when using FMG is the run time when considering three grid levels. The number of time steps is increased no matter how many grid levels are used. The run time when using two or four grid levels is also increased.

The last test case is the $24 \times 24 \times 24$ smooth heterogeneous permeability field problem. For this problem, the use of FMG increases both the number of time steps and the run times significantly.

One conclusion that can be drawn based on the results in Tabel 6.1 is that the algorithmic efficiency per time step is improved by using FMG. This is indicated by the average number of FAS cycles per time step generally decreasing when applying FMG. At first glance, this constitutes an improvement of the simulator, since the point in applying FMG is to provide a better initial starting guess for the additional V-cycles. However, this improvement might be due to the increase in time steps, which means smaller time steps, since the simulation horizon is fixed. If the time step size is small, then the coarse grid problem is easier to solve, since the initial guess (the solution from the previous time step) is closer to the solution for the current time step. Especially for the smooth heterogeneous case, the number of time steps is very high.

While generating the above test results we observed that the simulator often fails to converge at the first of the coarsest levels in the FMG approach. This coarse grid problem is indicated with a red circle in Figure 6.10.

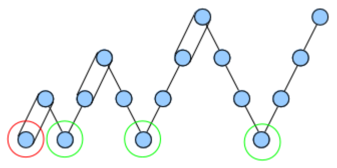


Figure 6.10: *When time steps are too large, FAS with FMG fails to converge at grid levels marked with circles.*

This means that the simulator has to reduce the length of the current time step in order to converge. Often this results in very small time steps. Once the size of the time step has been reduced so much, our observation is that it is difficult for the simulator to increase the time step size again. The reason is that it often fails at one of the coarsest levels in FMG marked with green circles in Figure 6.10.

One possible explanation is that the starting guess provided to FMG is not close enough to the solution to the coarse grid problem, which causes Newton's method to fail to converge. So far this initial guess has been generated by restricting the fine grid solution from the previous time step all the way to the coarsest grid. This is the case illustrated in part a) of Figure 6.11. We now consider two other strategies to generate the initial guess for FMG.

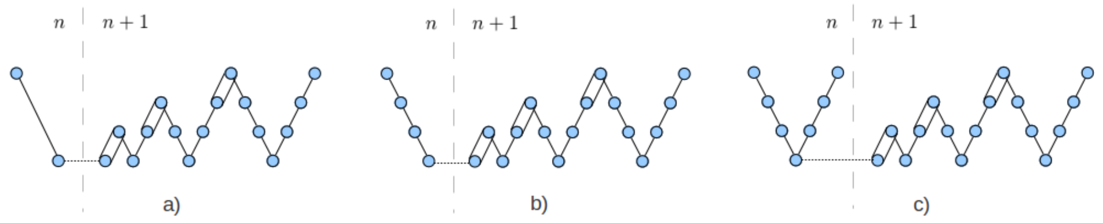


Figure 6.11: *Three strategies for generating initial guess for FMG. a) restrict fine grid solution from previous time step. b) use half V-cycle and c) use coarse grid solution from previous time step.*

An alternative strategy for generating the initial guess for FMG is to apply the operations corresponding to a half V-cycle. This means that the computations in lines 4-12 of Algorithm 5.6 are performed several times until the coarsest level is reached. At each of the grid levels visited in the half V-cycle we apply 10 smoothings. The reasoning behind this strategy is that the simulator not using FMG seldom diverges at the coarsest grid level. The third strategy is to use the coarse grid solution from the previous time step as initial guess for the FMG approach. These strategies are illustrated in parts b) and c) of Figure 6.11.

The right-hand side of the coarse grid problem has so far been the restriction of the right-hand side for the fine grid problem. However, applying strategy b) when generating the initial guess for FMG also affects the right-hand side of the problem. The explanation is found in line 12 of Algorithm 5.6, which describes how the right-hand side for a given grid level is generated. For strategy c), where the initial guess is the coarse grid solution from the previous time step, we choose the right-hand side for the current coarse grid problem as the right-hand side for the coarse grid problem at the previous time step.

The three different approaches for generating the initial guess (and the right-hand side) for FMG are tested and compared to the FAS simulator not using FMG. As test case we use the $16 \times 16 \times 16$ smooth heterogeneous permeability field problem. The input values are the same as used to generate the results in Table 6.1. The results are presented in Table 6.2.

Method	**Apply FMG**	**Avg. no. of FAS cycles/time step**	**No. of time steps**	**Run time (secs)**
FAS(2)	-	2.0	61	54
	a)	1	670	776
	b)	1	701	927
	c)	1	689	776
FAS(3)	-	4.0	71	65
	a)	1.1	390	260
	b)	1.1	347	305
	c)	1.1	369	240
FAS(4)	-	4.4	67	61
	a)	1.0	2956	1570
	b)	1.0	1989	1529
	c)	1.0	1880	998

Table 6.2: *Comparison of FAS simulator without FMG and FAS simulator using FMG for various strategies for generating initial guess.*

Based on the results in Table 6.11 it appears strategy c) performs the best of the three strategies, at least for the problem considered. However, the FAS simulator using FMG is far from as efficient as the FAS simulator not using FMG.

The literature suggests that multigrid performs better when applying FMG, [43] and [28]. This is not the result observed in this section. For the test cases considered in this section there is no improvement in using the FMG approach. The FAS simulator using FMG often has a longer run time and is using more time steps than the simulator not using FMG. The simulator applying FMG is not able to take as long time steps due to failure to converge at the coarsest grid. This leads to an increment in the number of time steps.

The conclusion is that it is not straightforward to make an efficient FMG implementation for the reservoir problem at hand.

6.5 Restricting cell-based input data

In the FAS method, restriction operators for the residual and for the primary variables are needed. In addition to these operators, restriction operators for the absolute permeabilities (and porosities if not kept constant) are needed as these properties are cell dependent. Currently, in a (x, y)-semicoarsening context, the absolute permeabilities are restricted by an arithmetic average over 4 cells equivalently to the restriction of the primary variables. The illustration on the left of Figure 6.12 shows how the permeabilities of four cells are restricted to one cell by taking the arithmetic average of the four cells.

Alterations to this approach of restricting the input data are tested. The idea is to include more of the expected physical behaviour in the coarser grid representations. Experiments are carried out, where the arithmetic average is replaced with the minimum of the permeabilities of the four cells. The reasoning is that the limiting permeability should reflect in the coarser grid cell average. This does not improve performance and it is therefore dismissed.

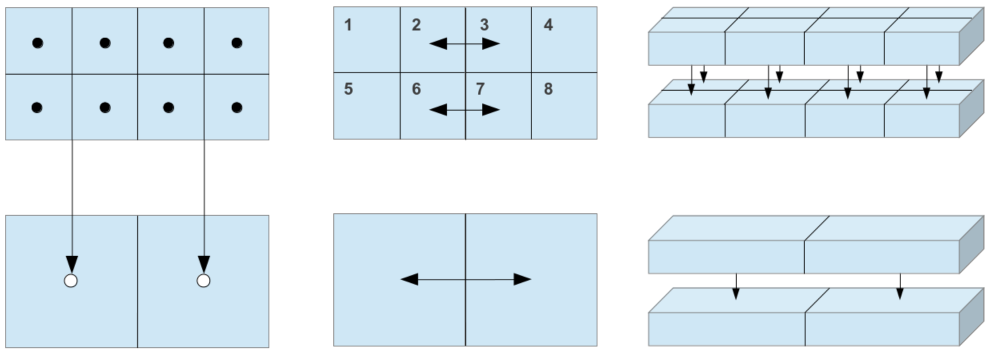


Figure 6.12: *Illustration of restricting input data. The left figure illustrates the restriction of permeabilities going from 4 to 1 cells. The middle figure illustrates the restriction of transmissibilities on an edge-based basis in the x- or y-direction. The right figure illustrates the restriction of transmissibilities in the z-direction.*

Instead of restricting cell-based permeabilities, it could be beneficial to restrict the edge-based transmissibilities. It can be argued that the permeabilities for cell 1, 5, 4 and 8 in the illustration in the middle of Figure 6.12 should not be included in the transmissibility for the coarser grid. This is because only the edges between cell 2 and cell 3 and between cell 6 and cell 7 are represented on the coarser grid.

As given in equation (3.29), the transmissibility between cell i and its neighbour cell j

is computed as

$$T_{ij} = \frac{A_{ij}k_{ij}}{\Delta h}$$

where Δh is either Δx, Δy or Δz depending on the flow direction and A_{ij} is the area of the interface perpendicular to the flow direction. Due to (x, y)-semicoarsening, the transmissibility on a $H = 2h$ coarser grid is

$$T_{ij} = \frac{2A_{ij}k_{ij}}{2\Delta h} = \frac{A_{ij}k_{ij}}{\Delta h}, \tag{6.28}$$

for flow in the x- or y-directions. H is the coarse grid mesh spacing and h is the fine grid mesh spacing. The two factors of 2 are due to the geometry of the coarser grid cells. When flow is along the x- or y-direction the factors cancel out. This means that the transmissibility over an edge on the coarser grid can be computed as an average of the two transmissibilities over the corresponding two edges on the fine grid. This is the situation illustrated in the middle part of Figure 6.12.

For flow along the z-direction, the transmissibility over an edge on the coarser grid can not be computed directly as an average of four transmissibilities over the fine grid edges. This can be seen from the transmissibility formula

$$T_{ij} = \frac{4A_{ij}k_{ij}}{\Delta h}, \tag{6.29}$$

where the factor 4 is because $A_{ijH} = \Delta x_H \Delta y_H$ with $\Delta x_H = 2\Delta x_h$ and $\Delta y_H = 2\Delta y_h$. This means that the transmissibility over an edge on the coarser grid is computed as 4 times an average of four transmissibilities over corresponding edges on the fine grid. For the arithmetic mean, this is the same as the sum of the four transmissibilities on the fine grid. This is the situation depicted in the right of Figure 6.12. Note that the void space with the arrows in the right illustration of Figure 6.12 does not exist, but it is only there to allow for illustration of the transmissibility of the interfaces in the z-direction.

Based on this alternative way of restricting the cell-based input data, two tests are carried out to see if this has any effect on the performance of the FAS solver. The two test cases are the SPE10 derived $10 \times 10 \times 10$ and a $16 \times 16 \times 16$ problem with a heterogeneous permeability field with a permeability range of 0.1-20000 mD generated as explained in Appendix E. These test cases are selected because of their extreme heterogeneity. The SPE10 derived permeability field consists of scattered high and low permeability values, whereas the second test case has a more smooth permeability field. Table 6.3 gives an overview of the performance of the FAS solver for the two ways of restricting the cell-based input data.

	Levels:	Restriction type:	Time steps:	Run time:
$10 \times 10 \times 10$ SPE10 derived permeability field, 200 days	FAS(2)	Restricting permeabilities	211	81
	FAS(2)	Restricting transmissibilities	229	83
$16 \times 16 \times 16$ smooth heterogeneous 0.1-20000 mD permeability field, 150 days	FAS(2)	Restricting permeabilities	126	122
	FAS(2)	Restricting transmissibilities	132	118
	FAS(3)	Restricting permeabilities	184	152
	FAS(3)	Restricting transmissibilities	175	151

Table 6.3: *Performance comparison of restricting permeabilities or transmissibilities for two different very heterogeneous test cases.*

The FAS solver is more or less indifferent in terms of computation time. For this reason, the original approach is kept. Tests have been conducted with harmonic means instead of arithmetic means. This proved to have a negative effect on performance.

6.6 Two-stage preconditioner: CPR

One of the main objectives of this thesis is to compare the conventional techniques based on global linearization in Newton's method against local linearization in FAS. As mentioned earlier, simulators based on global linearization are highly dependent on an effective linear solver for very large linear systems. Consequently, a choice was made to replace the direct linear solver with an iterative linear solver. The first iterative linear solver described and tested in section 6.3 is GMRES with ILU preconditioning. This is a generic combination of methods used in many fields as well as in reservoir simulation. This combination of linear solver and preconditioner results in a fairly high number of linear iterations as shown later in this section. Furthermore, the number of linear iterations increases as a result of increasing problem sizes or more heterogeneous systems.

The systems arising in reservoir simulation are of mixed character. They consist of a near-elliptic pressure part with long-range coupling and a near-hyperbolic conservation part with steep local gradients, [11, Cao]. Consequently, the systems are obvious targets for a two-stage preconditioner, where each stage deals effectively with the error characteristics of the individual parts of the systems. The two stages are obtained through decomposition of the systems. Currently, the most popular of such methods is the Constrained Pressure Residual (CPR), [45, 46, Wallis]. This preconditioner is implemented in the next-generation commercial reservoir simulator INTERSECT [11, Cao], and in the in-house reservoir simulators GPRS from Stanford University, [19], and GigaPOWERS from Saudi Aramco, [22, Fung].

In the following, we give a description of the algorithm, the implementation in PETSc and the improvements obtained compared to GMRES with ILU preconditioning.

6.6.1 Forming the pressure matrix

Before starting the linear solver, a full system matrix $\mathbf{A}$ (the Jacobian $\mathbf{J}$) and its corresponding pressure matrix $\mathbf{A}_p$ are set up in Newton's method. The pressure matrix is formed by reducing the full system matrix using an elimination process similar to Gaussian elimination. There are two ways to reduce the full system matrix, namely quasi-IMPES and true-IMPES [30, Jiang p. 128]. IMPES is an abbreviation for Implicit Pressure Explicit Saturations. It is a method used to reduce the size of the linear system of equations in Newton's method to one equation and unknown per grid cell, [6, Aziz, p.186]. The basic idea of IMPES is to

- obtain a single pressure equation by a combination of the flow equations
- solve this equation implicitly for pressure
- update saturations explicitly using the pressure computed in the previous step

Due to explicit treatment of saturations, the IMPES method imposes limitation on the size of the time step to maintain stable solutions. Keep in mind that CPR is a preconditioner for a linear solver and that an IMPES-like reduction technique is merely used as part of CPR and thus poses no limitations on the size of the time steps.

Having introduced the basic idea of IMPES, we can continue with the derivation of the pressure matrix used in the CPR preconditioner. The diagonal and off-diagonal block matrices used to construct the Jacobian or full system matrix have the form

$$\overbrace{\begin{bmatrix} A_{om_o} + F_{om_{o,i}} & 0 & 0 & A_{op} + F_{op_i} \\ 0 & A_{gm_g} + F_{gm_{g,i}} & 0 & A_{gp} + F_{gp_i} \\ 0 & 0 & A_{wm_w} + F_{wm_{w,i}} & A_{wp} + F_{wp_i} \\ VB_o & VB_g & VB_w & VB_p \end{bmatrix}}^{\text{Diagonal block matrix}} \tag{6.30}$$

$$\overbrace{\begin{bmatrix} F_{om_{o,j}} & 0 & 0 & F_{op_j} \\ 0 & F_{gm_{g,j}} & 0 & F_{gp_j} \\ 0 & 0 & F_{wm_{w,j}} & F_{wp_j} \\ 0 & 0 & 0 & 0 \end{bmatrix}}^{\text{Off-diagonal block matrix}}, \tag{6.31}$$

where

$$A_{cm_c} = \frac{\partial (V_p m)_{c,i}}{\partial m_{c,i}}$$

is the derivative of the accumulation of some component $c = \{o, g, w\}$ wrt. the molar density of the same component and

$$F_{\alpha p_i} = \frac{\partial f_i^\alpha}{\partial p_i}$$

is the derivative of the flow term for some phase $\alpha = \{o, g, w\}$ for cell i wrt. pressure for cell i. In the off-diagonal block, the flow term

$$F_{\alpha p_j} = \frac{\partial f_i^\alpha}{\partial p_j}$$

is derived wrt. a neighbour cell j. The derivatives of the residual of the volume balance constraints wrt. molar density of some component are

$$VB_\mathrm{c} = \frac{\partial r_{vb,i}}{\partial m_{c,i}}$$

The two IMPES reductions can be performed in the following ways.

Quasi-IMPES

In Quasi-IMPES, the derivatives VB_c are eliminated by a Gaussian elimination approach using elementary row operations. By computing the three fractions

$$\begin{aligned} F_1 &= \frac{VB_o}{A_{om_o} + F_{om_{o,i}}} \\ F_2 &= \frac{VB_g}{A_{gm_g} + F_{gm_{g,i}}} \\ F_3 &= \frac{VB_w}{A_{om_w} + F_{wm_{w,i}}}, \end{aligned} \tag{6.32}$$

we reduce our diagonal block matrix in (6.30) to an upper triangular matrix. This means that we have decoupled the system by eliminating the dependence on molar densities in the “pressure linked” equation (the volume balance equation). Therefore, a diagonal element of the pressure matrix $\mathbf{A}_p$ can be computed as the result of the elementary row operations on the element VB_p in (6.30)

$$\mathbf{A}_{p,(i,i)} = VB_p - F_1(A_{op} + F_{op_i}) - F_2(A_{gp} + F_{gp_i}) - F_3(A_{wp} + F_{wp_i}) \tag{6.33}$$

Since we are using elementary row operations on the full system Jacobian matrix, the off-diagonal elements of the pressure matrix $\mathbf{A}_p$ are formed as

$$\mathbf{A}_{p,(i,j)} = 0 - F_1 F_{op_j} - F_2 F_{gp_j} - F_3 F_{wp_j} \tag{6.34}$$

Applying this process to the full system $4N \times 4N$ Jacobian matrix results in a $N \times N$ pressure matrix, where N is the number of grid cells.

True-IMPES

In true-IMPES the molar densities (corresponding to saturations in other formulations) in the flow terms are treated explicitly. Therefore, the diagonal and off-diagonal block matrices can be simplified to

$$\overbrace{\begin{bmatrix} A_{om_o} & 0 & 0 & A_{op}+F_{op_i} \\ 0 & A_{gm_g} & 0 & A_{gp}+F_{gp_i} \\ 0 & 0 & A_{wm_w} & A_{wp}+F_{wp_i} \\ VB_o & VB_g & VB_w & VB_p \end{bmatrix}}^{\text{Diagonal block matrix}} \tag{6.35}$$

$$\overbrace{\begin{bmatrix} 0 & 0 & 0 & F_{op_j} \\ 0 & 0 & 0 & F_{gp_j} \\ 0 & 0 & 0 & F_{wp_j} \\ 0 & 0 & 0 & 0 \end{bmatrix}}^{\text{Off-diagonal block matrix}}, \tag{6.36}$$

Here the derivatives of the flow terms wrt. molar densities are dropped, since they are treated explicitly and as a result they do not depend on the next time step $n+1$. In the true-IMPES approach, the three fractions are given by

$$\begin{aligned} F_1 &= \frac{VB_o}{A_{om_o}} \\ F_2 &= \frac{VB_g}{A_{gm_g}} \\ F_3 &= \frac{VB_w}{A_{om_w}}, \end{aligned} \tag{6.37}$$

With these fractions the pressure matrix is computed like in Quasi-IMPES using equation (6.33) and (6.34). Having formed the pressure matrix, we can continue with the actual steps taking place inside the CPR preconditioner in each linear iteration.

6.6.2 Algorithm

Given the full system matrix $\mathbf{A}$, the pressure matrix $\mathbf{A}_p$ and the $3N$ fractions from either (6.32) or (6.37), we apply the following CPR preconditioner steps to the residual $\mathbf{r}$ given by the linear solver Flexible GMRES (FGMRES).

1. Restrict the full system residual $\mathbf{r}$ with length $4N$ to the pressure residual $\mathbf{r}_p$ with length N using the fractions F_1, F_2 and F_3.

$$r_{p,i} = r_{vb,i} - F_1 r_{o,i} - F_2 r_{f,i} - F_3 r_{w,i} \tag{6.38}$$

2. Solve the pressure system
$$\mathbf{A}_p\mathbf{x}_p = \mathbf{r}_p \tag{6.39}$$

3. Expand pressure solution to full system
$$\mathbf{s} = \begin{bmatrix} 0 & 0 & 0 & x_{p,1} & \cdots & 0 & 0 & 0 & x_{p,N} \end{bmatrix}^T \tag{6.40}$$

4. Correct the full system residual using the expanded pressure solution
$$\mathbf{r}_c = \mathbf{r} - \mathbf{A}\mathbf{s} \tag{6.41}$$

5. Apply the “second stage” preconditioning to the corrected full system
$$\mathbf{M}\mathbf{x} = \mathbf{r}_c \tag{6.42}$$

6. Add the expanded solution **s** from step 3 with the preconditioned vector **x** from step 5 and output this as the CPR preconditioned vector **y** to FGMRES
$$\mathbf{y} = \mathbf{x} + \mathbf{s} \tag{6.43}$$

The first stage of the CPR preconditioner is to solve for the pressure correction in step 2. This is a near-elliptic system and one of the most popular methods in reservoir simulation for solving this system is the Algebraic Multigrid (AMG) method. AMG is considered a black-box solver for elliptic linear systems, since it requires no grid information but rather operates directly on the linear (sparse) algebraic equations. In this thesis, a theoretical introduction of AMG is beyond the scope. Instead we refer to the introduction in [43, Stüben p. 413]. To ensure robustness in some special cases, the AMG method can be used as the preconditioner in GMRES when solving the pressure system in step 2. This is slower than just using AMG, but it may prove useful under very special circumstances, [30, Jiang, p.131].

The second stage of the CPR preconditioner in step 5 is often ILU(0). ILU(0) deals effectively with high frequency errors. If ILU(0) is used as preconditioner by itself, the low frequency error components linger and requires a large number of iterations to be affected, [11, Cao]. However, if ILU(0) is used after the pressure correction, only the high frequency errors remain and ILU(0) is quite effective at dealing with such systems. The reason only high frequency errors remain is that the first stage in CPR deals with the pressure system and resolves global coupling and low frequency errors. Individually, the two components ILU(0) and AMG would be inefficient, but together they constitute a powerful preconditioner.

6.6.3 Implementation

The CPR preconditioner is developed specifically for reservoir simulation and therefore it is not to be found in generic iterative linear solver libraries. In the PETSc framework it is possible to write your own preconditioner. This can be done in two ways. The first approach is to write your own preconditioner with functions defining interfaces to the rest of PETSc. The second approach is to use PCSHELL to create your own preconditioner class. In this way, it is only necessary to implement the essentials such as create, apply and destroy functions.

Both the true-IMPES and quasi-IMPES reductions are implemented and tested. Furthermore, it is possible to choose between solving the pressure system solely with AMG or with AMG as preconditioner for GMRES (GMRES+AMG). Both options are tested and evaluated. The tolerances used for GMRES+AMG or AMG (with more than one V-cycle) are a relative tolerance of 10^{-5} and an absolute tolerance of 10^{-10}. However, investigations have shown that the best overall performance is often achieved using just one V-cycle, [30, Jiang, p.132].

6.6.4 Improvement

In this section, the CPR preconditioner is compared to single-stage ILU preconditioning. Furthermore, GMRES-AMG is compared to just AMG for the first stage solve in CPR and lastly true-IMPES is compared to quasi-IMPES. All of these tests are conducted to establish the best preconditioner for the given problems.

The first test case is the $10 \times 10 \times 10$ problem with the SPE10 derived permeability field from section 4.4.3. The simulations are 16000 days $\approx$ 44 years, $\Delta t_{\max} = 30.0$ days and cell sizes are $20m \times 10m \times 2m$. The initial step size is 0.1 days and the number of pre- and post-smoothings are 5. Hardware specification 1 in Appendix B is used in this section.

Method:	**Time steps:**	**Run time:**	**Linears/newton:**
ILU(1)	997	800	62.59
CPR true-IMPES (1 AMG V-cycle)	994	326	4.08
CPR quasi-IMPES (1 AMG V-cycle)	1031	378	4.12
CPR true-IMPES (GMRES+AMG)	990	472	3.76

Table 6.4: *Comparison of preconditioners for* $10 \times 10 \times 10$ *heterogeneous problem. 44 years of simulation. Run time is in seconds.*

The test case is very heterogeneous due to the permeability field. The results in table 6.4 demonstrate the effectiveness of the CPR preconditioner compared to ILU for highly heterogeneous problems. The number of linear iterations per newton iteration is much lower for the CPR preconditioner compared to ILU(1). The performance of ILU(0) was too slow to wait for 44 years of simulation time.

True-IMPES reduction in the CPR preconditioner performs better than quasi-IMPES. This is confirmed by experiments in [19, Durlofsky, p.34]. Furthermore, using just one V-cycle of AMG instead of GMRES+AMG to solve the pressure system is faster. It takes 4 more time steps due to some loss of accuracy when not solving the pressure system to the given tolerances, but this is well worth it in terms of run time.

As a second experiment, we compare the CPR and ILU preconditioners for two similarly heterogeneous problems with different resolutions. Using the fixed pattern smooth heterogeneous permeability field generated as explained in Appendix E, a $480m \times 240m \times 48m$ reservoir is simulated for 150 days with two different cell sizes. Table 6.5 holds the results of the experiment.

	Method:	**Time steps:**	**Run time:**	**Linears per newton:**
$16 \times 16 \times 16$ *grid cell problem*	CPR true-IMPES (1 AMG V-cycle)	101	148	4.69
	CPR quasi-IMPES (1 AMG V-cycle)	99	155	5.28
	ILU(0)	167	1084	151.77
	ILU(1)	102	228	37.72
	ILU(2)	102	188	18.91
	ILU(3)	100	318	15.06
$24 \times 24 \times 24$ *grid cell problem*	CPR true-IMPES (1 AMG V-cycle)	152	737	4.83
	CPR quasi-IMPES (1 AMG V-cycle)	148	867	5.77
	ILU(0)	237	6370	198.14
	ILU(1)	181	2569	78.85
	ILU(2)	154	1383	33.49
	ILU(3)	155	2175	25.02

Table 6.5: *Comparison of preconditioners for two similarly heterogeneous problems with different resolution. 150 days of simulation. Run time is in seconds. Note that there is a limit of 1000 linear iterations per newton iteration.*

The results show that the CPR preconditioning is almost unaffected by larger problem sizes in terms of linear iterations per newton iterations. Of course, each linear iteration is more expensive due to the larger problem size, but the number of linear iterations per newton iteration only increases with very little. ILU preconditioning on the other hand results in a factor between 1.3-2.1 of additional linear iterations per newton iteration. Equally important is the fact that the relative increase in run time is larger for ILU preconditioning than for the CPR preconditioning. Table 6.5 does not contain results for CPR preconditioning with GMRES+AMG for the pressure solve, since it is not competitive for the given problems. The last conclusion, which can be made from the results in Table 6.5, is that true-IMPES continues to outperform quasi-IMPES.

Overall, CPR preconditioning effectively deals with highly heterogeneous problems and increasing problem sizes. It outperforms ILU preconditioning both in terms of convergence rate and timings. From this point onwards, the best performing solver in the above tests, namely FGMRES with CPR preconditioning with true-IMPES reduction and 1 AMG V-cycle is used as standard.

CHAPTER 7

Performance study

The performance studies considered so far have primarily been conducted in order to see how a change in some components of the simulator affects the performance. In this chapter, we provide a more extensive performance study of the two simulators developed in this work.

The chapter consists of two parts. The first part in section 7.1 compares the simulator based on standard Newton's method (the standard newton simulator) with the FAS based simulator. The second part in section 7.2 considers how specific parameters or choices in various components of FAS affect the performance.

In section 7.1.1, a study is carried out to see how the simulators scale in terms of run time as the number of grid cells increases. The algorithmic efficiency of the simulators are compared in 7.1.2, and in section 7.1.3 a study considering the number of property calculations per fine grid cell is done. Finally, a memory comparison and a heterogeneity stress test are carried out in sections 7.1.4 and 7.1.5.

In the second part of the chapter, a profiling study illustrating the effect of using more grid levels is conducted. This is done in section 7.2.1. In section 7.2.2 it is studied how the number of smoothings affects the performance of FAS. An alternative coarsening technique is investigated in section 7.2.3, and in section 7.2.4 it is studied how FAS is affected by using a specific number of newton iterations at coarsest level instead of solving to a certain accuracy.

All of the test cases considered in this section are gravity inversion cases. For these test cases, the molar densities are initialized with water in the top third of the cells, oil in the middle third of the cells and gas in the bottom third of the cells. This is similar to the initialization used for the verification cases described in section 4.4. The pressure is initialized to 250 bar. Hardware specification 2 in Appendix B is used for all timings.

7.1 FAS compared to Standard Newton

In this section, the FAS based reservoir simulator is compared to the reservoir simulator based on conventional global linearization using standard Newton's method.

7.1.1 Scalability

This test compares the run time for the FAS based reservoir simulator with the run time for the standard newton based reservoir simulator. The reservoir is kept at a fixed size and the resolution of the problem is increased while maintaining equivalent anisotropies. The reservoir is $480m \times 240m \times 48m$. The simulation horizon is 150 days, the initial time step is 0.1 days and the maximum allowed time step size is $\Delta t_{max} = 30$ days. No more than 10 FAS cycles and 10 newton iterations are allowed. We use the input values specified in Appendix A. The FAS simulator uses 5 pre- and 5 post-smoothings. The reservoir has a smooth heterogeneous permeability field with permeabilities in the range 200-600 mD. The permeability field is generated as described in Appendix E.

The same tolerances apply for both the standard newton simulator and the FAS simulator. Figure 7.1 displays the results.

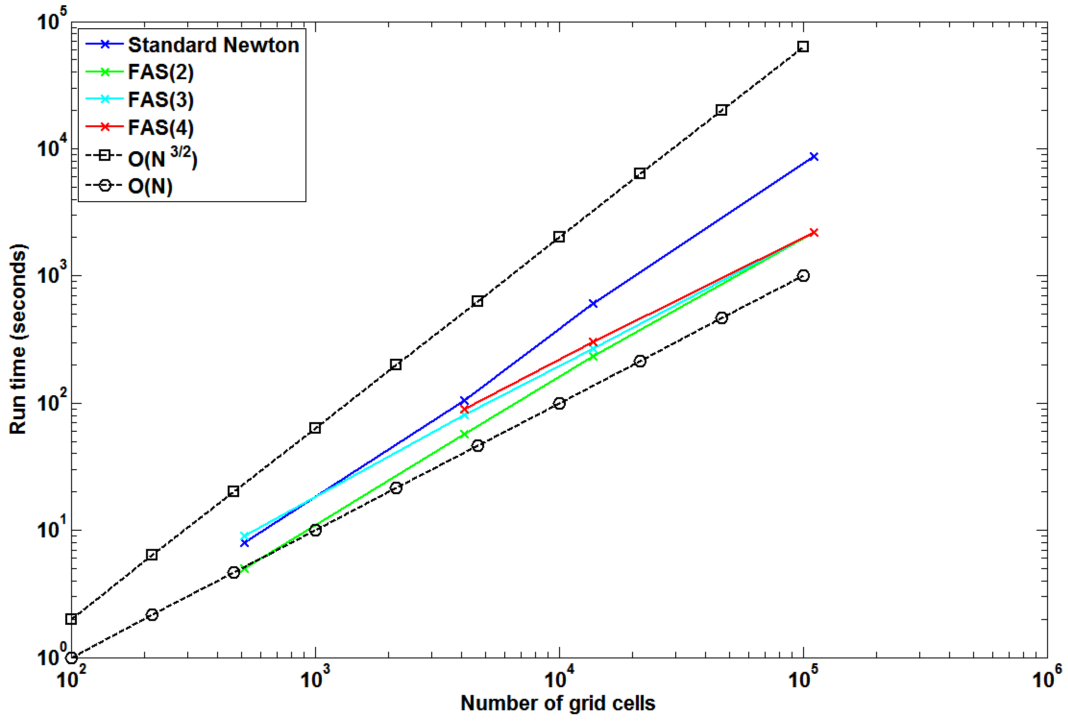


Figure 7.1: *Run time as a function of number of grid cells for SN, FAS(2), FAS(3) and FAS(4). The reservoir size is fixed at $480m \times 240m \times 48m$.*

The FAS based reservoir simulator outperforms the standard newton based reservoir simulator in terms of run time. The difference becomes larger as the problem sizes increase. With an appropriate number of grid levels, FAS appears to have linear scaling for the larger problem sizes considered in Figure 7.1. This is not the case for the standard newton based reservoir simulator, where the scaling appears to be somewhere between $O(N)$ and $O(N^{3/2})$. The limiting factor should be the ILU factorization in the second stage of the CPR preconditioner as a results of the involved factorization cost for the sparse Jacobian matrix. With increasing problem sizes, the 2 outer bands of the Jacobian matrix get further away from the diagonal. This means that the bandwidth of the matrix is increased when the problem size is increased. ILU factorization depends on the bandwidth, [24, Golub, p.152], and consequently, perfect linear scaling should not be expected.

Also, the performance of the algebraic multigrid method in BoomerAMG on these types of linear systems should probably be studied further. Nevertheless, Figure 7.1 displays the measured results using the popular PETSc and Hypre libraries. With FAS, it appears feasible to avoid scaling issues by increasing the number of grid levels and consequently keeping the coarsest grid systems to be solved relatively small. In this way, linear scaling should be possible for even larger sizes as indicated by the results in Figure 7.1.

Part of the reason that FAS is quicker than standard newton can be found in the number of time steps spent during simulations. For the run times in Figure 7.1, the corresponding number of time steps can be found in Table 7.1.

	Problem size			
Method	$8 \times 8 \times 8$	$16 \times 16 \times 16$	$24 \times 24 \times 24$	$48 \times 48 \times 48$
Standard newton	68	101	152	201
FAS(2)	40	60	91	171
FAS(3)	65	83	82	161
FAS(4)	N/A	93	94	127

Table 7.1: *Number of time steps for the simulations displayed in Figure 7.1.*

For all four problem sizes, the FAS based reservoir simulator spends fewer time steps than the standard newton based reservoir simulator. The length of the time steps are controlled as outlined in section 4.3.6. The time step lengths are directly related to the number of outer iterations for the previous time steps. For the standard newton solver, the outer iterations are the number of newton iterations and for the FAS solver, the outer iterations are the number of FAS cycles. This suggests that the FAS solver has better algorithmic efficiency than the standard newton solver for the given problems and for the given tolerances. This seem to be in line with theory, which states that the convergence rate of FAS is not bound by the convergence rate of Newton's method, [8, Brandt, p.85]. Note that the second type of nonlinear multigrid, namely Newton-MG would be bounded by the convergence rate of Newton's method.

7.1.2 Residual plots

In this test, the residuals for the outer iterations are studied. The outer iterations of the nonlinear solvers are respectively the newton iterations for standard newton and FAS-cycles for the FAS solver. We investigate how the residuals are reduced for each outer iteration. The absolute and relative tolerances for the linear solver are set to 10^{-14} and the tolerances for the outer iterations are set to $\tau_o = \tau_g = \tau_w = 10^{-12}$ and $\tau_{vb} = 10^{-14}$. Figure 7.2 shows the residuals until convergence for standard Newton's method, FAS(2) and FAS(3). Each line represents a time step. The first 50 time steps are depicted. Time integration is fixed with 0.1 days per time step. The test is for a $24 \times 24 \times 24$ grid cell problem with $20m \times 10m \times 2m$ cell sizes. 5 pre- and post-smoothings are used in FAS. The permeability field is generated as described in Appendix E with permeabilities ranging from 200-600 mD.

The black line, which is separated from the cluster of lines, is the first time step. We suspect that the reason for the first time step to deviate may be due to the fact that pressure is initialized to 250 bars for all grid cells. For the first few time steps and especially for the first time step, the pressure solution exhibits large variations until it settles with an incrementally higher pressure towards the bottom of the domain due to gravity. This large variation is reflected in the difficulty of converging for the first time step.

Note that the plotting of the residuals for the volume balance equation starts at iteration 1. This is because the residual for the volume balance equation for iteration 0 is equivalent to the residual for the volume balance equation for the last iteration of the previous time step. Consequently, it is very close to zero for iteration 0.

The 49 remaining lines in the plots of Figure 7.2 representing 49 time steps are all clustered and clearly show the trend of residual reduction in the nonlinear solvers. The standard newton solver typically converges after 3 or 4 iterations, whereas FAS(2) uses 4-5 iterations and FAS(3) uses 18-20 iterations. Evidently, convergence rate is deteriorating when using more than 2 levels for this problem. This is a general trend we observe when the number of grid levels is increased. However, as it can be observed in Figure 7.2, the initial residual reduction is very fast, which means for engineering accuracy purposes, FAS is competitive with standard Newton's method. In fact, as results in Table 7.1 show, the FAS solvers in general step longer than the standard newton solver for general engineering tolerances. Keep in mind that the length of time steps is directly reflected in the number of outer iterations for the previous time steps.

7.1.3 Property calculations

As mentioned in section 4.3.1, certain properties such as the formation volume factors, viscosities, phase molar densities, mass densities, saturations, relative permeabilities and mobilities are computed for each cell. These properties are based on the current

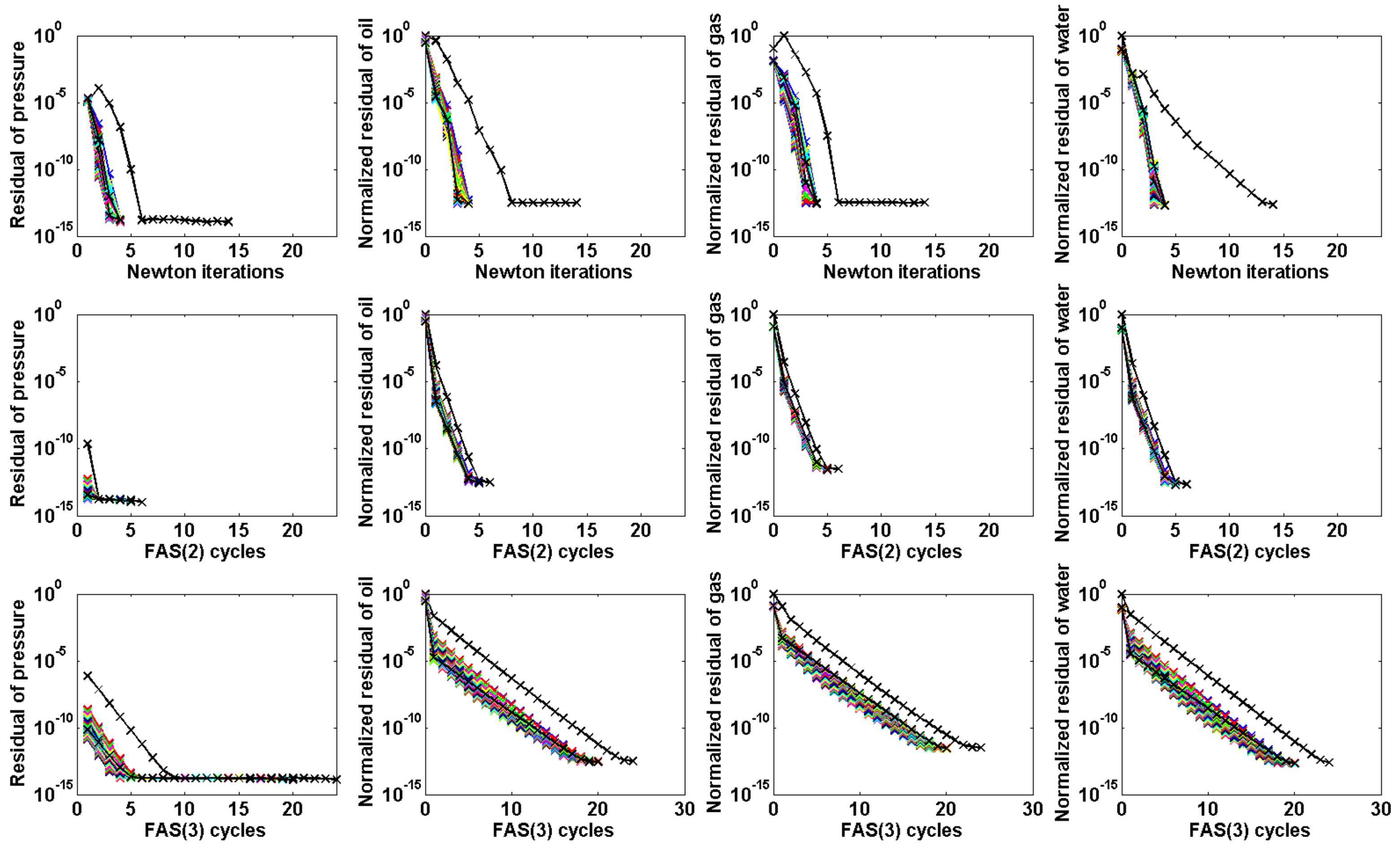


Figure 7.2: *Residual plots for standard newton, FAS(2) and FAS(3). The residuals for standard newton for the pressure, oil, gas and water are displayed in the first row. Similarly, residuals for FAS(2) are in the second row and residuals for FAS(3) are in the third row. Problem size:* $24 \times 24 \times 24$ *with* $20m \times 10m \times 2m$ *cell sizes. 5 pre- and post-smoothings are used for FAS.*

values for the primary variables, meaning that they have to be updated each time the approximation for the current solution is updated.

For more advanced reservoir simulation models than the one considered in this thesis (e.g. models considering more components and/or miscible fluids) the number of properties computed for each cell is larger and/or the properties are more complex to compute. This means that these property calculations can be quite expensive in terms of run time.

As described in Algorithms 5.6 and 5.7, the properties are updated at each grid level in the FAS cycle and in each smoothing sweep in the smoother. This results in the FAS simulator doing more property calculations than the standard newton simulator.

For the problems considered so far, the FAS simulator has proven to perform better in terms of run time than the standard newton simulator. Since the property calculations are more expensive for more advanced reservoir simulators, we study how much more expensive the property calculations in our model can be, before the standard newton simulator becomes faster than the FAS simulator.

This is done by adding a for-loop around the property calculations in the *properties* routine, such that the calculations are repeated a number of times.

As a test case we consider the $24 \times 24 \times 24$ smooth heterogeneous permeability field problem with permeabilities in the range 200-600 mD. This field is generated as described in Appendix E. The simulation horizon is 150 days, the initial time step is 0.1 days and the maximum allowed time step size is 30 days. The smoother uses 5 pre- and 5 post-smoothings and no more than 10 FAS cycles and 10 newton iterations are allowed. We use the parameter settings in Appendix A.

The number of property calculations per fine grid cell for this problem is for each of the simulators specified in Table 7.2.

Method	*SN*	*FAS(2)*	*FAS(3)*	*FAS(4)*
Prop. calcs.	894	2312	5665	7268

Table 7.2: *Number of property calculations per fine grid cell for the considered* $24 \times 24 \times 24$ *problem.*

The results of the study are depicted in Figure 7.3. The problem are solved to the same accuracy for all of the simulations.

These results show that for the considered problem FAS(4) can use up to about 150 times more time on computing property calculations before being slower than the standard newton simulator. For FAS(3) it is up to about 200 times more time, and for FAS(2) it is up to about 800 times more time.

These estimates are based on the number of property calculations per fine grid cell given in Table 7.2, meaning that the estimates depend on the problem considered. The number of property calculations per fine grid cell in a simulation varies from problem

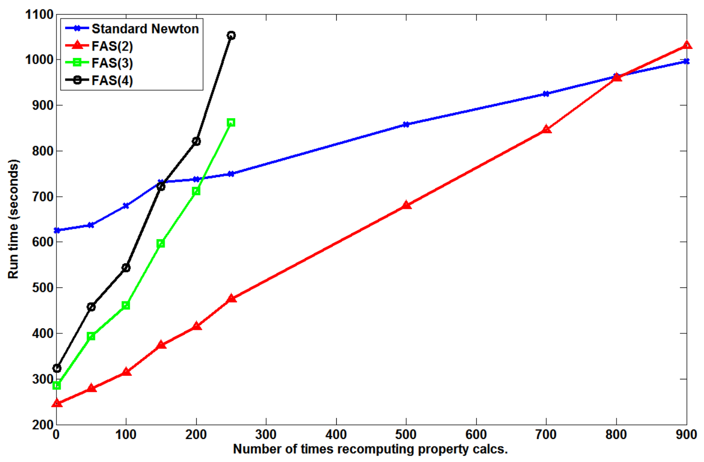


Figure 7.3: *Study of how many times the property calculations can be repeated before FAS becomes slower than Standard Newton.*

to problem. It depends on the number of FAS cycles used, the number of grid levels considered, the number of newton iterations used per FAS cycle and the number of smoothings used per grid level per FAS cycle. This means that the number of property calculations per fine grid cell does not necessarily increase as the size of the problem increases. It is how hard the problem is to solve that matters. This is illustrated by the results in Figure 7.4, where the number of property calculations per fine grid cell and the number of time steps as a function of problem sizes are plotted.

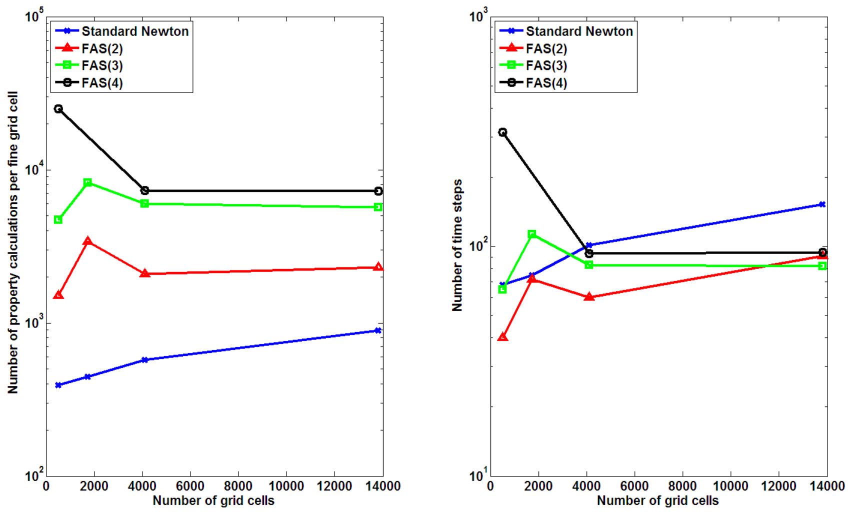


Figure 7.4: *Number of property calculations and number of time steps as a function of problem size.*

These results show that the number of property calculations per fine grid cell is more or less reflected in the number of time steps used in the simulation. Note that the step size controller takes the number of FAS cycles and the number of newton iterations per FAS cycle (and thereby how hard the problem is to solve) into consideration.

The results in Figure 7.4 are for a fixed reservoir size of $480m \times 240m \times 48m$. The permeability field is a smooth heterogeneous permeability field with permeabilities in the range 200-600 mD. The input specifications are the same as used for the problem considered in Table 7.2.

The results presented in this section indicate that there is room for considering more complex property calculations without the FAS simulator losing its advantage in terms of run time.

7.1.4 Memory comparison

As previously discussed, the memory required with FAS is significantly less than for the conventional methods with global linearization. Figure 7.5 shows actual memory measurements of the two simulators implemented in this work. Standard newton with global linearization is compared to FAS with 2, 3 or 4 grid levels. The tool *massif* in Valgrind[1] is used to measure the memory usage.

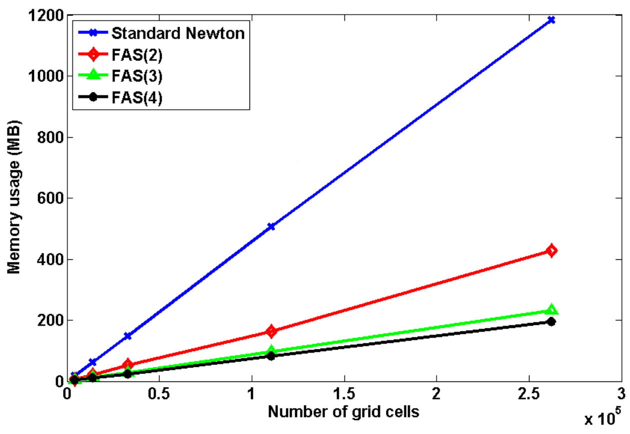


Figure 7.5: *Memory usage as a function of the number of grid cells. Measured with massif using Valgrind.*

Figure 7.5 shows, that the memory usage scales linearly with the number of grid cells for both simulators as expected. Compared to the standard newton based simulator, the FAS based simulator uses 3-6 times less memory depending on the number of grid levels used in FAS. For very large problems and a higher number of grid levels, the memory savings will be even more significant. In practice, this means it is possible to fit more grid cells on each node (multi-core CPU, GPU or MIC).

[1] http://valgrind.org/

The large memory savings are due to the fact that FAS linearizes locally. This means that the Jacobian matrix is not computed or stored on the finest grid. It can be argued that a matrix-free linear solver can be constructed, which might also result in memory savings. Very little work has been published on this topic in reservoir simulation. The complexity of designing an effective matrix-free preconditioner for reservoir simulations might be the limiting factor. Furthermore, the number of function evaluations (to continuously recompute entries in the Jacobian) might become prohibitive.

The fact that we can significantly reduce memory consumption with FAS is an interesting aspect. For memory bound applications, reducing memory consumption results in faster run times. Furthermore, the reduced memory consumption enables simulation with higher resolution on the same hardware.

7.1.5 Heterogeneity stress test

A large challenge in reservoir simulation is efficient simulation of very heterogeneous permeability fields. The performance of generic linear solvers such as GMRES with ILU preconditioning is very affected by the nature of the coefficients introduced through heterogeneity and anisotropy. As demonstrated in section 6.6 and in [11, Cao], a linear solver using CPR preconditioning effectively deals with these issues. However, this is in a linear setting. The nonlinear solver when using Newton's method is still affected by the heterogeneity, no matter how effective the linear solver is. This inevitably leads to shorter time steps.

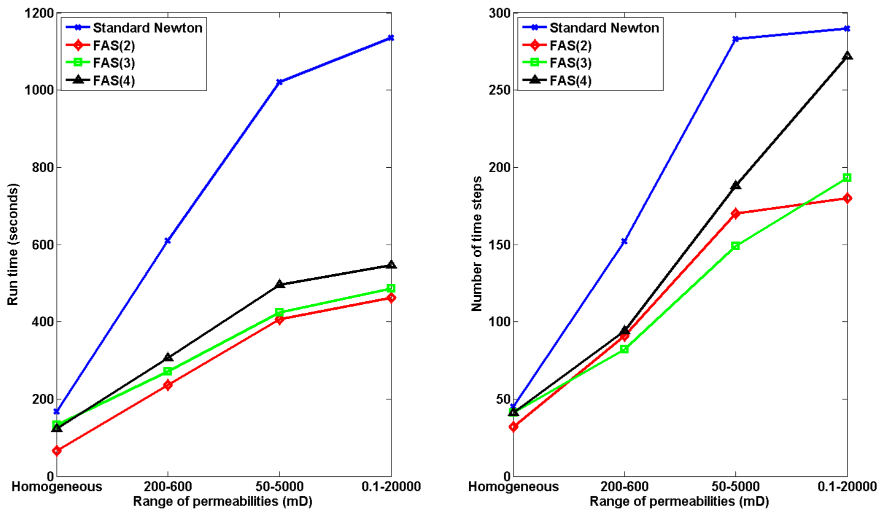


Figure 7.6: *Left plot: Run time as a function of range of permeabilities. Right plot: Number of time steps as a function of range of permeabilities. Problem size:* $24 \times 24 \times 24$, *5 pre- and post-smoothings,* 150 *days, cell sizes:* $20m \times 10m \times 2m$.

Figure 7.6 demonstrates results from a "stress test", where the range of permeabilities is varied between two extremes. The two extremes are a homogeneous permeability field and a permeability field with permeabilities ranging from $0.1 - 20,000$ mD. The heterogeneous permeability fields are generated by the technique explained in Appendix E. The left plot shows the run time for standard newton with global linearization and FAS with $2, 3$ and 4 grid levels. The corresponding number of time steps are displayed in the right plot.

The results are encouraging for the application of FAS on challenging reservoir problems. The FAS based reservoir simulator remains superior both in run time and algorithmic efficiency. Keep in mind that the number of time steps is a direct consequence of the number of outer iterations (FAS cycles). FAS(4) spends an increasing number of time steps for the $0.1 - 20,000$ mD case, but it is also a fairly high level of coarsening with a coarsest grid problem of $3 \times 3 \times 24$ grid cells. Despite this relatively larger number of time steps, the number of time steps is still lower than for standard newton. Furthermore, the run time exhibits the same trend as FAS(2) and FAS(3).

7.2 FAS components

In this section, further studies of the FAS simulator are conducted. We investigate how the distribution of work depends on the number of grid levels. Furthermore, we study how the performance of the simulator is affected by the number of smoothings used and the coarsening strategy applied. Lastly, a study is carried out to see how changing the accuracy requirements for the newton solver for the coarsest grid problem affects the performance.

7.2.1 Profiling

As mentioned in section 1.2, the global linearization approach used in conventional reservoir simulation techniques result in very large linear systems. This implies that the linear solver makes up a large part of the computation time. Since the FAS simulator only solves the linear systems at the coarsest grid, it is less dependent on the linear solver, which is illustrated in Figure 5.12 for the FAS(2) simulator. However, this figure shows that for larger problem sizes the linear solver still constitutes a large fraction of the computation time. We investigate if it is possible to reduce this dependence by using more grid levels. This is done by profiling the simulation for FAS(2), FAS(3) and FAS(4) for the same problem solved to the same accuracy. The profiling is done using Valgrind.

As test case we consider the $24 \times 24 \times 24$ smooth heterogeneous permeability field problem with permeabilities in the range 200-600 mD. This permeability field is generated as described in Appendix E. We consider a simulation horizon of 150 days, an initial time step of 0.1 days and the maximum time step size is 30 days. 5 pre- and 5 post-smoothings are used, and no more than 10 FAS cycles and 10 newton iterations per FAS cycle is allowed. Furthermore, we apply the input values in Appendix A.

The profilings show that the work in the simulators are distributed as depicted in Figure 7.7.

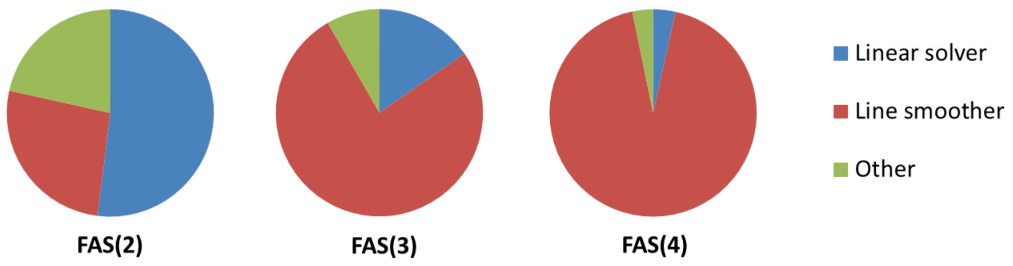


Figure 7.7: *Distribution of computational work for FAS(2), FAS(3) and FAS(4) for the same problem. The problem size is* $24 \times 24 \times 24$.

These results indicate that it is possible to reduce the dependence on the linear solver by using more grid levels. Note that the problem size considered in this study is quite small, meaning that for larger problems the linear solver would probably constitute a larger part of the work than depicted in Figure 7.7. However, the figure shows an interesting trend.

Furthermore, the results in Figure 7.7 indicate that by introducing more grid levels, it is possible to keep the main workload in the smoother. In the aspect of many-core architectures, this is an important result since smoothers are suitable for implementation on parallel architectures, as explained in section 5.3.1.

7.2.2 Number of smoothings

In this section we study how the number of smoothings affects the performance of the FAS simulator.

As seen in Figure 7.2, the FAS(3) simulator has fast initial convergence for the considered problem. However, below engineering accuracy the convergence rate curve is less steep. The results in Figure 7.2 is obtained when 5 pre- and 5 post-smoothings are applied. The results for FAS(3) is represented in the four top plots of Figure 7.8.

We redo the study in section 7.1.2 for the FAS(3) simulator, but now we use 10 pre- and 10 post-smoothings instead of 5. The results are depicted in the four bottom plots in Figure 7.8.

A comparison of the top plots with the bottom plots shows that it is possible to reach a higher accuracy within the first few iterations by increasing the number of smoothings, i.e. the algorithmic efficiency is increased. This result indicates that it is beneficial to

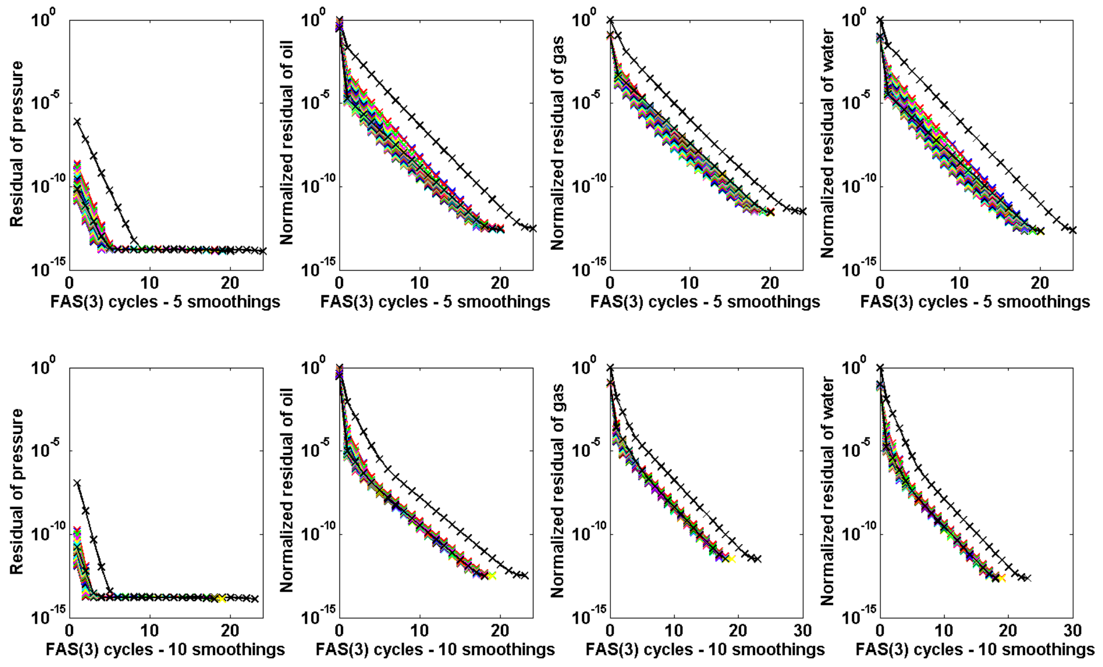


Figure 7.8: *Residual plots for FAS(3). For the four top plots 5 pre- and 5 post-smoothings are used. For the four bottom plots 10 pre- and 10 post-smoothings are used. Problem size:* $24 \times 24 \times 24$ *with* $20m \times 10m \times 2m$ *cell sizes.*

use a large amount of smoothings. However, if the number of smoothings is too large then the numerical efficiency suffers.

Figure 7.9 depicts a study of the run time as a function of the number of smoothings for FAS(2), FAS(3) and FAS(4). Furthermore, this figure shows a study of the number of time steps used as function of the number of smoothings. The test case is the $24 \times 24 \times 24$ smooth heterogeneous permeability field problem, with permeabilities in the range 200-600 mD. The simulations are run for 150 days, the initial time step is 0.1 days and the maximum time step size is 30 days. We allow no more than 10 FAS cycles ad 10 newton iterations. The size of the cells is $20m \times 10m \times 2m$. We use the input parameters listed in Appendix A.

Figure 7.9 shows that in terms of run time and number of time steps, there is a big difference in applying 1 pre- and post-smoothing or applying more than 1 pre- and post-smoothing. This is depicted in the two top plots of the figure. The two bottom plots show the exact same result as the two top plots, but the bottom plots consider a smaller range of values on the y-axis.

The results in Figure 7.9 show that by increasing the number of smoothings, the algorithmic efficiency is improved. Remember that the number of time steps implicitly indicates the algorithmic efficiency, since the time step size controller takes the number of FAS cycles and newton iterations into consideration.

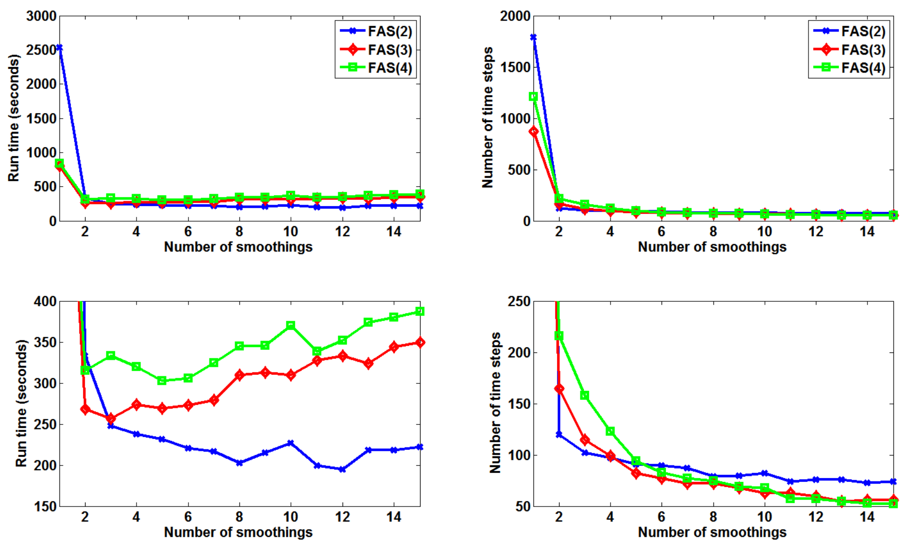


Figure 7.9: *Run time and number of time steps as a function of number of smoothings. Top plots and bottom plots show same results, but bottom plots zoom in on a smaller range of values on the y-axis. Problem size:* $24 \times 24 \times 24$ *with* $20m \times 10m \times 2m$ *cell sizes.*

However, at some point there is no gain in terms of run time in further increasing the number of smoothings. At this point the extra number of smoothings appears to be unnecessary work. This result is most significant for FAS(3) and FAS(4), which also in total uses more smoothings than FAS(2), since more grid levels are considered.

Based on the results in Figure 7.9, it seems that the optimal number of smoothings for the considered problem for FAS(2) is in the range 8-12 smoothings, and for FAS(3) and FAS(4) it is in the range 3-5 smoothings.

Similar studies considering a $32 \times 32 \times 32$ smooth heterogeneous permeability field problem and a $32 \times 32 \times 32$ homogeneous permeability field problem have been carried out. The same input values and hardware specifications have been used for these studies. The results are depicted in Figures F.1 and F.2 in Appendix F. For these problems the optimal number of smoothings seems to be in the range 7-12 for FAS(2) and 1-2 for FAS(3) and FAS(4). This indicates that the number of smoothings is problem specific.

Furthermore, Figures 7.9, F.1 and F.2 indicate that the larger the problem is, the more expensive it is to deviate from the optimal number of smoothings.

7.2.3 Standard coarsening vs semicoarsening

So far we have only considered (x, y)-semicoarsening. As mentioned in section 5.3.2, the reason for this is that we want to preserve as much information in the z-direction as possible.

In this section, we test the effect of using standard coarsening instead of using (x, y)-semicoarsening. By standard coarsening it is implied that coarsening takes place in all three dimensions simultaneously. Hence we extend the restriction stencil in (5.31) and the prolongation stencil in (5.32) to consider 8 points instead of 4.

We verify the implementation of the FAS(2) simulator using standard coarsening by comparing the solution to the solution obtained from ECLIPSE. As verification case for the FAS(2) simulator using standard coarsening, we use the $18 \times 18 \times 18$ problem with a smooth heterogeneous permeability field with permeabilities between 200-600 mD. This permeability field is generated as descibed in section E. The simulation horizon is 100 days, the initial time step is 0.1 day and the maximum allowed time step is also 0.1 day. We use 5 pre- and 5 post-smoothings and allow no more than 10 FAS cycles and 10 newton iterations per FAS cycle. We force ECLIPSE to take small time steps, such that the two simulations are comparable. The input values in Appendix A are used.

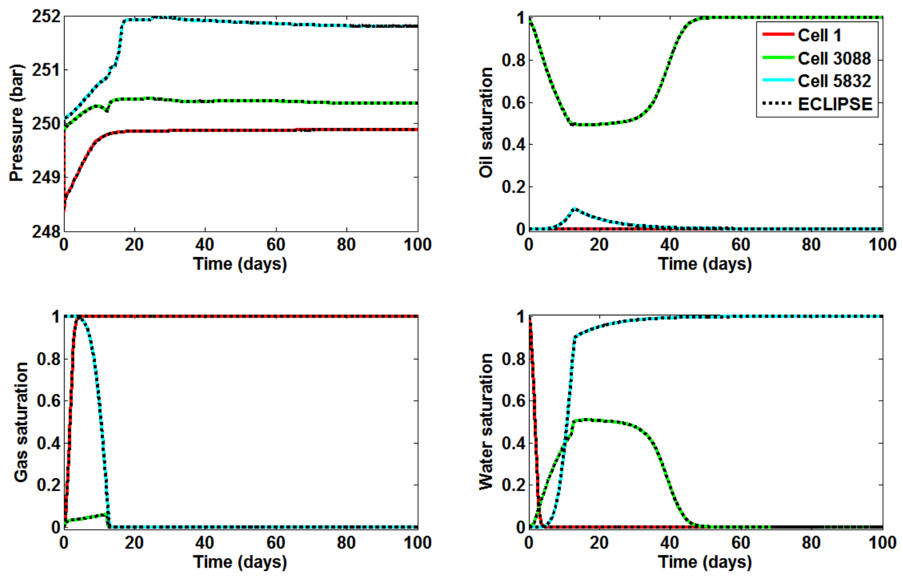


Figure 7.10: *Verification of FAS(2) using standard coarsening for* $18 \times 18 \times 18$ *problem with smooth heterogeneous permeability field.*

As illustrated in Figure 7.10 the FAS(2) simulator finds a solution similar to the solution from ECLIPSE. Verifications of the FAS(3) and the FAS(4) simulators are in Appendix F.

The performance of the FAS simulator using standard coarsening is compared to the FAS simulator using (x, y)-semicoarsening coarsening. As test cases we consider a $32 \times 32 \times 32$ homogeneous permeability field problem with permeabilities of 100 mD and a $24 \times 24 \times 24$ heterogeneous permeability field problem with permeabilities in the range 200-600 mD. We use the same parameter settings as for the verification test except the simulation horizon is now 150 days. The results are listed in Table 7.3.

Problem specifications	**Method**	**Coarsening method**	**No. of time steps**	**Run time (secs)**
Homogeneous $32 \times 32 \times 32$ 100 mD	FAS(2)	semi	45	243
		std.	39	258
	FAS(3)	semi	59	458
		std.	58	375
	FAS(4)	semi	65	493
		std.	69	447
Smooth heterogeneous $24 \times 24 \times 24$ 200-600 mD	FAS(2)	semi	91	235
		std.	426	1195
	FAS(3)	semi	82	269
		std.	563	1396
	FAS(4)	semi	95	305
		std.	524	1232

Table 7.3: *Comparison of the FAS simulator using (x, y)-semicoarsening and the FAS simulator using standard coarsening.*

For the homogeneous test case, the two coarsening techniques seem to perform nearly equally well in the FAS simulator. In terms of run time, FAS(3) and FAS(4) are faster when applying standard coarsening. However, for the smooth heterogeneous problem the simulator using (x, y)-semicoarsening is superior in terms of run time and number of time steps. A possible explanation is that the correction term obtained when using standard coarsening, is not as good a correction as the correction term obtained when applying (x, y)-semicoarsening. We investigate this by increasing the number of pre- and post-smoothings used in the simulation of the smooth heterogeneous problem. The result is in Table 7.4.

Problem specifications	**Method**	**Coarsening method**	**No. of time steps**	**Run time (secs)**
10 pre- and post smoothings	FAS(2)	std.	296	1363
	FAS(3)	std.	362	1660
	FAS(4)	std.	304	1317
20 pre- and post smoothings	FAS(2)	std.	192	1508
	FAS(3)	std.	280	1795
	FAS(4)	std.	174	1510

Table 7.4: *Performance of the FAS simulator using standard coarsening for the smooth heterogeneous $24 \times 24 \times 24$ problem using a different number of smoothings.*

By comparing these results with the results in Table 7.3 it is seen that the algorithmic efficiency is improved when more smoothings are applied. This is indicated by a reduced number of time steps when applying more smoothings, since the time step size controller takes the number of FAS cycles and newton iterations into consideration. However, the numerical efficiency is decreased when applying a higher number of smoothings as indicated by the increase in run times.

It seems that (x, y)-semicoarsening is the best strategy for the gravity inversion problems considered in this work.

7.2.4 Number of newton iterations

Until now, the coarsest grid problems in FAS have been solved with Newton's method for which the iterative process is continued until the conditions of the stopping criteria with the absolute tolerances in Appendix A have been met. Since the solution on the coarsest grid only is used to compute a correction, it might be beneficial not to solve the coarsest grid problems to a given tolerance but rather use a fixed number of newton iterations. This also removes any sum reductions necessary when computing the 2-norms used to evaluate whether the tolerances have been met.

Note that for all tests conducted here, the stopping criteria and tolerances for the non-linear solver (outer loop) is exactly the same. Furthermore, the permeability fields are generated as explained Appendix E with permeabilities ranging from $200 - 600$ mD.

Figure 7.11 demonstrates how using a fixed number of newton iterations on coarsest grid affects the run time and the number of time steps. The experiments are carried out using $1, 2, 3$ or 4 newton iterations on coarsest grid in FAS. The rightmost points with the label "Tolerance" are the run time and number of time steps when Newton's method for the coarsest grid problem is forced to converge to the given tolerances in Appendix A. The initial time step size is $\Delta t = 0.1$ days and 10 outer iterations are allowed for both FAS and Newtons's method. The input values in Appendix A are used.

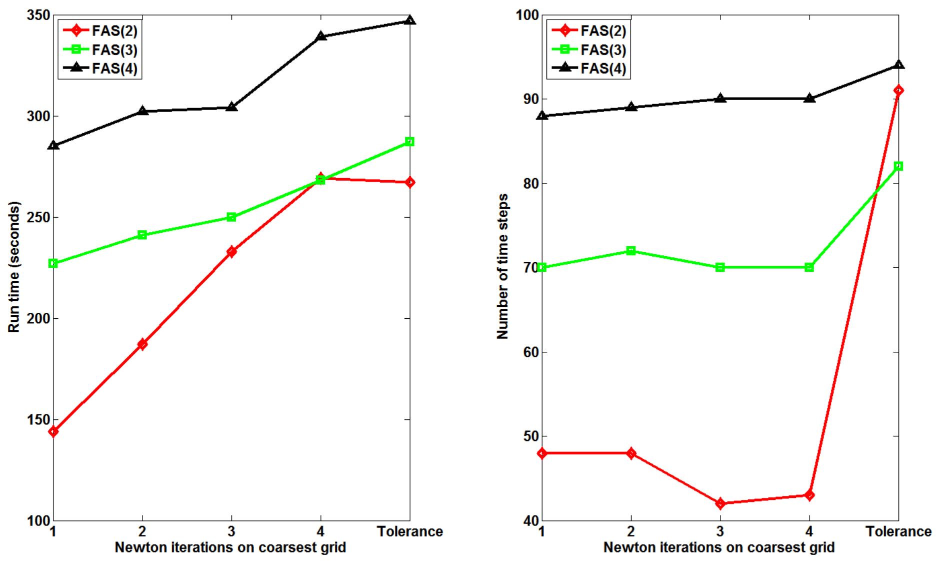


Figure 7.11: *Study of the effect of the number of newton iterations used on the coarsest grid problem for FAS with* $2, 3$ *and* 4 *grid levels for a* $24 \times 24 \times$ *smooth heterogenenous (*$200 - 600$ *mD range) in 150 days and 5 pre- and post-smoothings.*

Figure 7.11 shows that 1 newton iteration is the best in terms of run time for this problem. Furthermore, Figure 7.11 shows that for FAS with 3 or 4 grid levels, the number of time steps do not decrease when more newton iterations are applied on the coarsest grid problem. Experiments carried out for other problems have the same behaviour. For this reason, only 1 newton iteration is applied on the coarsest grid problems from now on.

The first test carried out in this chapter in section 7.1.1 studies the scaling of Standard Newton and FAS. This test is carried out again to see how only using 1 newton iteration on the coarsest grid problems affects this scaling. Figure 7.12 displays the run time as a function of the number of grid cells for a fixed reservoir size.

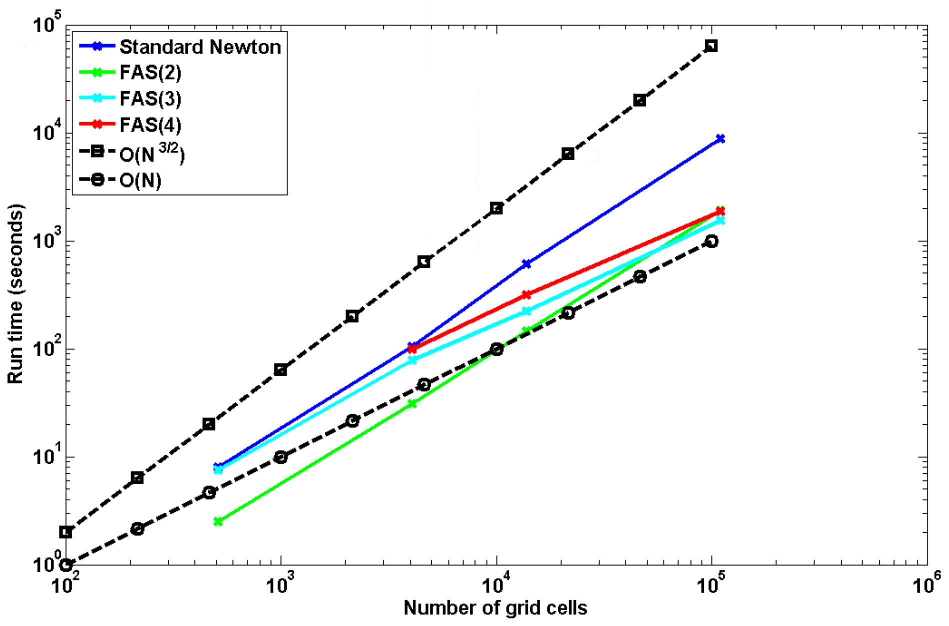


Figure 7.12: *Run time as a function of the number of grid cells for SN, FAS(2), FAS(3) and FAS(4). The reservoir size is fixed at* $480m \times 240m \times 48m$.

The trends in Figure 7.12 are similar to those observed in Figure 7.1. The only difference is that the run times are faster when only using 1 newton iteration for the coarsest grid problems. The simulator has been verified with this change. The verification plots are in Appendix F.

The results in Table 7.5 are far more interesting. The rows without the stars (*) are the same number of time steps as before in Table 7.1 and the rows with the stars (*) are the number of time steps when only using 1 newton iteration for the coarsest grid problems in FAS.

Table 7.5 demonstrates that by selecting an appropriate number of grid levels in FAS we are now able to maintain the same number of time steps when the number of grid cells increases. This means that the convergence rate of FAS does not deteriorate with

Method	**Problem size** $8 \times 8 \times 8$	$16 \times 16 \times 16$	$24 \times 24 \times 24$	$48 \times 48 \times 48$
Standard newton	68	101	152	201
FAS(2)	40	60	91	171
FAS(2)*	25	33	48	57
FAS(3)	65	83	82	161
FAS(3)*	65	79	70	58
FAS(4)	N/A	93	94	127
FAS(4)*	N/A	97	88	69

Table 7.5: *Number of time steps. Rows without the stars (*) are the same number of time steps as before in Table 7.1 and the rows with the stars (*) are the number of time steps when only using 1 newton iteration for the coarsest grid problems in FAS.*

increasing grid resolution as it is the case for Standard Newton. This result is very interesting as it addresses a serious problem in conventional reservoir simulators, which are based on global linearization with Newton's method. In reservoir simulation it is normal that the convergence rate of Newton's method deteriorates as the number of grid cells increases for a fixed reservoir size. Similar behaviour is observed for other applications, [10, Cai]. This will have an impact on the scalability of the standard Newton method, which is not favorable for very large-scale simulations.

To verify this result, a series of experiments are carried out. For different number of grid cells and a fixed reservoir size, we perform fixed time stepping with $\Delta t = 0.5$ days, $\Delta t = 1$ days and $\Delta t = 2$ days for 100 days and compute the average number of outer iterations per time step. The outer iterations are newton iterations for Standard Newton and FAS-cycles for FAS. Table 7.6, 7.7 and 7.8 show the results.

Method	**Problem size** $8 \times 8 \times 8$	$16 \times 16 \times 16$	$24 \times 24 \times 24$	$48 \times 48 \times 48$
Standard newton	3.2	3.3	4.3	8.1
FAS(2)	1.9	1.7	1.2	N/A
FAS(3)	2.9	3.2	2.3	N/A
FAS(4)	4.0	3.4	2.5	1.8

Table 7.6: *Average number of outer iterations (newton iterations for Standard Newton and FAS-cycles for FAS) for fixed $\Delta t = 0.5$ days.*

Method	**Problem size** $8 \times 8 \times 8$	$16 \times 16 \times 16$	$24 \times 24 \times 24$
Standard newton	4.0	6.1	9.4
FAS(2)	2.2	2.2	1.8
FAS(3)	3.6	4.2	3.3
FAS(4)	4.2	4.6	3.9

Table 7.7: *Average number of outer iterations (newton iterations for Standard Newton and FAS-cycles for FAS) for fixed $\Delta t = 1$ days.*

	Problem size		
Method	$8 \times 8 \times 8$	$16 \times 16 \times 16$	$24 \times 24 \times 24$
Standard newton	6.7	14.5	10.7
FAS(2)	2.5	2.8	3.0
FAS(3)	4.2	5.4	5.0
FAS(4)	5.08	5.8	5.8

Table 7.8: *Average number of outer iterations (newton iterations for Standard Newton and FAS-cycles for FAS) for fixed $\Delta t = 2$ days.*

The results in Table 7.6, 7.7 and 7.8 supports the beforementioned statement that the convergence rate of FAS does not deteriorate as the resolution increases. Generally, this is not the case for Standard Newton, where the average number of newton iterations increases as the resolution increases. For the $48 \times 48 \times 48$ grid cell problem, only Standard Newton and FAS(4) with $\Delta t = 0.5$ days is simulated due to time constraints. We expect similar trends for the other combinations as it is also indicated by the number of time steps for FAS(2) and FAS(3) in Table 7.5. This should be confirmed in tests in continuation of this work.

Another interesting study is the average number of linear iterations for the same tests as above. Table 7.9, 7.10 and 7.11 show the results. N/A is indicated for some of the $48 \times 48 \times 48$ problem sizes due to the long run times exceeding the time constraints of this work.

	Problem size			
Method	$8 \times 8 \times 8$	$16 \times 16 \times 16$	$24 \times 24 \times 24$	$48 \times 48 \times 48$
Standard newton	2.7	4.1	5.1	8.1
FAS(2)	2.5	3.0	4.0	N/A
FAS(3)	1.8	2.3	4.1	N/A
FAS(4)	1.0	1.7	3.0	5.5

Table 7.9: *Average number of linear iterations for fixed $\Delta t = 0.5$ days.*

	Problem size		
Method	$8 \times 8 \times 8$	$16 \times 16 \times 16$	$24 \times 24 \times 24$
Standard newton	3.6	6.2	7.9
FAS(2)	2.9	3.8	5.7
FAS(3)	2.5	3.0	5.1
FAS(4)	1.0	2.0	3.6

Table 7.10: *Average number of linear iterations for fixed $\Delta t = 1$ days.*

We observe that the number of linear iterations decreases when more grid levels are used in FAS. This indicates that the coarsest grid problems are easier to solve for the linear solver when more grid levels are used. Whether this is due to the fact that the coarsest grid problems are smaller when more grid levels are used or if it is because of the relaxation performed by the smoother is a subject for further study. In this work,

Method	**Problem size**		
	$8 \times 8 \times 8$	$16 \times 16 \times 16$	$24 \times 24 \times 24$
Standard newton	5.0	8.9	8.4
FAS(2)	3.6	5.7	8.8
FAS(3)	2.8	3.8	6.8
FAS(4)	1.0	2.5	4.8

Table 7.11: *Average number of linear iterations for fixed $\Delta t = 2$ days.*

we note that this effect also is of significant importance if simulation of higher resolution models is to be performed on a daily basis.

Chapter 8

Considerations for parallelization

The main motivation for the work presented in this thesis is to investigate if nonlinear multigrid techniques have the potential to enable efficient simulations of higher resolution models. Higher resolution models require parallel reservoir simulators. With the rapidly growing number of cores available, algorithms capable of good parallel performance are needed. FAS has interesting abilities in this direction. As demonstrated in section 7.2.1, the majority of the computational work can be kept in the smoother, which is a local operator in that for any given cell, information is only required from its six neighbours, meaning it is point-to-point communication. To understand the parallelization of multigrid, we need to consider how parallizable each component of multigrid is.

- **Restriction and prolongation:** The restriction and prolongation operators are local and hence they can be performed in parallel.

- **Smoothing:** Parallelizing the smoothing operator is the most challenging part. In this work, we are using a Gauss-Seidel line smoother. In parallel, a Gauss-Seidel smoother would either require some sort of coloring scheme or it can be combined with Jacobi smoothing on the edges of the subdomains assigned to each processor. The first approach can be cumbersome and requires a good balance between interior grid cells compared to boundary grid cells, [2, Adams]. In constrast to boundary grid cells, interior grid cells do not communicate with cells of another processor. A scalable implementation of Gauss-Seidel with a coloring scheme is presented in [1, Adams]. The second approach is to perform Jacobi smoothing on the boundary grid cells and Gauss-Seidel smoothing for the interior grid cells. This is perfectly scalable, but the resulting multigrid convergence usually suffers or in some circumstances the multigrid method even diverges with this approach, [2, Adams].

Another issue that has to be taking into consideration is the level of coarsening used in a parallel multigrid implementation. If too much coarsening is applied, the amount of communication between nodes compared to the work for each thread will become a bottleneck. Parallelization of multigrid on uniform meshes is quite easy to implement, however for non-uniform meshes it becomes more challenging, [40, Sundar]. Despite this, there are many success stories for regularly structered grids. For unstructered grids, the challenge becomes harder. It requires efficient construction of restriction and prolongation operators and the need for repartitioning and load balancing the mesh all the way to the coarse grid while minimizing communication, [40, Sundar].

A way to overcome the difficulties in implementing a parallel Gauss-Seidel smoother is by replacing it with a polynomial smoother. In [1, Adams], a comparison between Gauss-Seidel and polynomial smoothing is carried out on unstructered meshes. It is demonstrated that polynomial smoothing is competitive in serial and outperform Gauss-Seidel smoothing in parallel.

With respect to parallelization, one limitation in the current implementation of our FAS based simulator is the fact that we are using FGMRES as a linear solver on the coarsest grid. GMRES is using orthogonalization, which requires global communication for the inner products. This is a problem on massively parallel distributed systems, [39, Sturler].

Another limitation in terms of achieving a scalable implementation on modern many-core architectures is the components used in the CPR preconditioner. The current implementation is using AMG for the first stage and ILU(0) for the second stage of the CPR preconditioner. Research is going on into parallelizing both components on massively parallel architectures, [32, 29]. However, this is not a straightforward task. An alternative approach is to use polynomial preconditioning as described in [22, Fung]. This approach only uses matrix-vector products for which highly effective kernels have been developed.

Another interesting aspect of FAS is that by increasing the number of grid levels, we should be able to maintain a constant amount of work in the linear solver independent of the problem size. For this reason, we may not experience the same scaling issues for the linear solver as for conventional methods with global linearization in Newton's method.

Chapter 9

Conclusion

The primary work presented in this thesis is a thorough investigation of the application of nonlinear multigrid techniques, specifically the Full Approximation Scheme (FAS), for simulation of subsurface multiphase porous media flow. Two reservoir simulators have been implemented. The first simulator uses the FAS method and the second simulator uses global linearization with Newton's method. The reason for implementing both simulators is to enable comparison between a reservoir simulator using conventional techniques based on Newton's method and a reservoir simulator using the nonlinear multigrid technique FAS.

The most essential component of a conventional reservoir simulator is the linear solver. In this work, the linear solver has been implemented with state-of-the-art choice of methods. These methods include FGMRES with CPR preconditioning, where the first stage of the CPR preconditioner is using the algebraic multigrid method and the second stage is using ILU(0). The fact that state-of-the-art choice of methods are used for the linear solver has facilitated fair comparisons between the two beforementioned simulators.

Extensive comparisons have been carried out between the two simulators. In chapter 7, we have demonstrated that for the given model equations and for the range of problems considered here, the simulator based on FAS outperforms the simulator based on standard newton in terms of algorithmic efficiency, computation time and memory requirements. The range of problems considered in this work are all so-called "gravity inversion" test cases, where the water and gas components switch place. These tests have been conducted with both homogeneous and highly heterogenenous permeability fields. In both situations, the simulator based on FAS has proven robust and more efficient than the simulator based on global linearization with standard Newton's method.

In chapter 7 the effect of changing various parameters in the FAS solver has been studied. The optimal number of smoothings for the given problems has been investigated. Profiling has been carried out, which indicates that the majority of the computational work can be kept in the smoother. As discussed previously, the smoothing operator is due to locality considered a good match for many-core implementation.

The standard coarsening procedure has been compared with semicoarsening. We found that (x, y)-semicoarsening works better than standard coarsening for the range of problems considered here. Furthermore, a study has been carried out on the number of newton iterations performed for the coarsest grid problem in FAS. We found that one newton iteration on the coarsest grid is sufficient for finding a good correction to the finest grid solution.

The reservoir simulator based on FAS is a proof of concept to establish algorithmic performance in serial. As discussed in chapter 8, an implementation of FAS on a distributed system e.g. multi-core CPU or multi-GPU/MIC require many further investigations. Also the model equations include several simplifications. More complexity needs to be included in the model to see if FAS is capable of dealing with these as effectively as demonstrated in this work. However, compared to the previous work published on FAS for reservoir simulation, [36, Molenaar], these model equations and the range of problems considered here are much more complex and much closer to real world application.

9.1 Suggestions for future work

The proof of concept FAS based reservoir simulator presented in this thesis has been proven to have good algorithmic performance in serial for the model equations and for the range of problems considered here. Investigations should be carried out to see of this trend continues for more complex model equations and for a larger range of problems.

The following simplifications should be omitted.

- The relative permeabilities equal the saturations.
- The effects of capillary pressure are neglected.
- Immiscible fluids are assumed.
- The porosities are kept constant throughout the domain.

Furthermore, some sort of well modelling should be included to see how FAS deals with problems, where water is injected in one or more wells and oil is produced in some other wells.

An interesting study, especially in terms of parallel implementation, could be polynomial smoothing. It has been demonstrated for a range of problems that polynomial smoothing can be competitive with Gauss-Seidel smoothing, [2, Adams]. Polynomial smoothing is trivial to implement in parallel, since it only requires matrix-vector products.

If FAS continues to exhibit good performance for the challenges presented above, a natural extension is a parallel implementation. As a first step, a single GPU or MIC implementation would be a good starting point to see how FAS works with the large number of cores available. If FAS achieves good many-core performance on a single PCI accelerator, the next step would be a multi-GPU or multi-MIC implementation to see if the communication between GPU/MIC nodes can be sufficiently hidden to achieve a distributed massively parallel implementation.

In section 7.2.4, we show that by selecting an appropriate number of grid levels, the convergence rate of FAS does not deteriorate with increasing problem sizes. This is a very interesting result as it addresses a serious problem for conventional methods in large-scale reservoir simulations. Further investigations should be carried out to see if this behaviour is maintained when the simplifications stated above are disregarded.

Appendix A

Input values

This appendix contains the physical constants, reservoir specific data and simulator settings used in the implementation. To obtain the same units for the accumulation and flow terms, a so-called Darcy constant is multiplied with the transmissibilities and hence with the flow term. It is merely a unit conversion factor with a value of $C_{\text{Darcy}} = 0.00852702\ {}^{\text{cPm}^2}/_{\text{day/bar}}$.

Symbol	Value	Units
ρ^o_{sc}	800.0	$^{\text{kg}}/_{\text{m}^3}$
ρ^g_{sc}	0.9907	$^{\text{kg}}/_{\text{m}^3}$
ρ^w_{sc}	1022.0	$^{\text{kg}}/_{\text{m}^3}$
M^o	120.0	$^{\text{kg}}/_{\text{kmol}}$
M^g	25.0	$^{\text{kg}}/_{\text{kmol}}$
M^w	18.025	$^{\text{kg}}/_{\text{kmol}}$
g	0.0000980665	$^{\text{m}^2\text{kg}}/_{\text{bar}}$

Table A.1: *Physical constants.*

Symbol	Value	Units
φ	0.3	-
p_{ref}	250	bar
B^w_{pref}	1.03	-
μ^w_{pref}	0.3	cP
C_v	0.0	$^1/_{\text{bar}}$
C	0.000041	$^1/_{\text{bar}}$
C_{rock}	0.000053	$^1/_{\text{bar}}$

Table A.2: *Reservoir specific input values.*

Description	Value
Absolute tolerance for linear solver	10^{-10}
Relative tolerance for linear solver	10^{-5}
Absolute tolerance for Newton's method - τ_o	10^{-4}
Absolute tolerance for Newton's method - τ_g	10^{-4}
Absolute tolerance for Newton's method - τ_w	10^{-4}
Absolute tolerance for Newton's method - τ_{vb}	10^{-6}
Absolute tolerance for FAS - τ_o	10^{-4}
Absolute tolerance for FAS - τ_g	10^{-4}
Absolute tolerance for FAS - τ_w	10^{-4}
Absolute tolerance for FAS - τ_{vb}	10^{-6}

Table A.3: *Simulator settings.*

Appendix B

Hardware specifications

The specifications of the hardware used for tests with timings are given below.

Hardware specification 1

Processor:	Intel Core 2 Duo P8600
Clock speed:	2.4 GHz
Cache:	3 MB L2
FSB speed:	1066 MHz
Instruction set:	64-bit
Hard drive:	512 GB 7200 RPM
Memory:	2x4 GB DDR3 1066 MHz
OS:	Ubuntu 12.04

Hardware specification 2

Nodes:	64 x HP ProLiant SL2x170z G6
Each node:	2x Intel Xeon X5550 (quad-core)
Clock speed:	2.66 GHz
Cache:	8 MB L3
FSB speed:	1333 MHz
Instruction set:	64-bit
Hard drive:	500 GB internal SATA (7200 RPM) per node
Memory:	24 GB per node
Interconnect:	QDR Infiniband
OS:	Scientific Linux 6.1

Appendix C

Property calculations

This appendix contains the formulas used to evaluate the properties of a grid cell. Table C.1 gives an overview of the properties described in this appendix.

Property	**Symbol**	**Dependants**
Formation volume factor	B^α	p
Phase molar density	b^α	p
Saturation	S^α	$m_{c=\alpha}, p$
Viscosity	μ^α	p
Mass density	ρ^α	$p \quad {}^*(m_{c=\alpha}, p)$
Relative permeability	k_r^α	$m_{c=\alpha}, p$
Porosity	ϕ	p

Table C.1: *Overview of properties described in this appendix and their dependency on pressure p and molar densities m_c.*

Furthermore, the table shows the dependencies of the different properties with respect to pressure p and molar densities m_c. This information we need in order to derive the derivatives wrt. p and m_c of each of the properties. These derivatives are also listed in this appendix.

C.1 Formation volume factor

The formation volume factor B is defined as the ratio of the volume of the fluid measured at reservoir conditions to the volume of the same fluid measured at standard conditions [12, Chen, p. 12].

We compute formation volume factors differently for the oil and gas phases compared to the water phase. For the oil and gas phases linear interpolation on pressure from tabulated inputs is used, whereas for the water phase a formula is used. The reason for this is that water is less compresssible than oil and gas, meaning using a fixed formula is sufficient to obtain an accurate approximation. However, for the oil and gas phases, using a fixed formula is not always sufficiently representative of the lab experiments. Instead tabulated inputs are used to achieve a better approximation and a more flexible approach.

Oil and gas

The formation volume factor of oil, B^o, and of gas, B^g, depends on pressure and is computed using linear interpolation on reservoir specific input values. Note that oil and gas have different values in their interpolation tables, which are seen below.

Entry	p	$B^o(p)$	$\mu^o(p)$
1	50	1.18	0.8
2	600	1.08	1.6

Table C.2: *Interpolation table for oil.*

Entry	p	$B^g(p)$	$\mu^g(p)$
1	50	0.0205	0.014
2	600	0.0039	0.025

Table C.3: *Interpolation table for gas.*

In the implementation, the oil interpolation table in Table C.2 and the gas interpolation table in Table C.3 are respectively called PVDO and PVDG, short for pressure, volume, dead oil/dry gas.

Based on one of these tables the formation volume factor is calculated as

$$B^\alpha = B^\alpha[1] + (p - p[1])\frac{B^\alpha[2] - B^\alpha[1]}{p[2] - p[1]}, \quad \alpha = o, g, \tag{C.1}$$

where p is the actual pressure in the grid cell and [ ] indicates an index number in the interpolation table.

Example:

If a cell has pressure $p = 300$, the formation volume factor of gas is calculated as

$$B^g = 0.0205 + (300 - 50)\frac{0.0039 - 0.0205}{600 - 50} = 0.0130 \tag{C.2}$$

Derivatives for oil and gas

As indicated in Table C.1 the formation volume factor depends only on pressure p. The derivative of (C.1) wrt. p is

$$\frac{\partial B^\alpha}{\partial p} = \frac{B^\alpha[2] - B^\alpha[1]}{p[2] - p[1]}, \quad \alpha = o, g \tag{C.3}$$

Water

The formation volume factor for water also depends on pressure, but instead of linear interpolation the following formulas are used.

$$\begin{aligned} X &= C \cdot (p - p_{ref}) \\ B^w &= \frac{B^w_{pref}}{1 + X + \frac{1}{2}X^2}, \end{aligned} \tag{C.4}$$

Here C is the compressibility of water, p_{ref} is a reference pressure and B^w_{pref} is a reference formation volume factor measured at reference pressure. These values are listed in Table A.1. In the implementation all of these are user-given inputs.

Derivative for water

The derivative of (C.4) wrt. p is

$$\begin{aligned} \frac{\partial B^w}{\partial p} &= \frac{\partial}{\partial p}\left(\frac{B^w_{pref}}{1 + X + \frac{1}{2}X^2}\right) \\ &= \frac{\partial}{\partial p}\left(\frac{B^w_{pref}}{1 + C \cdot (p - p_{ref}) + \frac{1}{2}(C \cdot (p - p_{ref}))^2}\right) \\ &= -B^w_{pref}\frac{\frac{\partial}{\partial p}\left(1 + C \cdot (p - p_{ref}) + \frac{1}{2}(C \cdot (p - p_{ref}))^2\right)}{\left(1 + C \cdot (p - p_{ref}) + \frac{1}{2}(C \cdot (p - p_{ref}))^2\right)^2} \\ &= -B^w_{pref}\frac{(C + C^2 \cdot (p - p_{ref}))}{\left(1 + C \cdot (p - p_{ref}) + \frac{1}{2}(C \cdot (p - p_{ref}))^2\right)^2} \end{aligned} \tag{C.5}$$

C.2 Phase molar density

The phase molar density is defined as the number of moles per volume for a single phase, and it is given by

$$b^\alpha = \frac{b^\alpha_{sc}}{B^\alpha}, \tag{C.6}$$

where b_{sc}^{α} is the phase molar density at standard conditions. It is calculated as

$$b_{sc}^{\alpha} = \frac{\rho_{sc}^{\alpha}}{M^{\alpha}}, \tag{C.7}$$

where M^{α} is the molecular weight and ρ_{sc}^{α} is the mass density at standard conditions. The values for M^{α} and ρ_{sc}^{α} are physical constants, which are displayed in Table A.1.

Derivatives

The derivative for the phase molar density wrt. pressure is given by

$$\frac{\partial b^{\alpha}}{\partial p} = \frac{\partial}{\partial p}\left(\frac{b_{sc}^{\alpha}}{B^{\alpha}}\right) = b_{sc}^{\alpha}\frac{\partial}{\partial p}\left(\frac{1}{B^{\alpha}}\right) = -\frac{b_{sc}^{\alpha}}{(B^{\alpha})^2}\frac{\partial B^{\alpha}}{\partial p} = -\frac{b^{\alpha}}{B^{\alpha}}\frac{\partial B^{\alpha}}{\partial p}, \tag{C.8}$$

where $\frac{\partial B^{\alpha}}{\partial p}$ is defined in either (C.3) or (C.5).

C.3 Saturation

The saturation of a fluid phase is defined as the fraction of the pore volume of a porous medium filled by this phase. It is given by

$$S^{\alpha} = \frac{m_c}{b^{\alpha}}, \tag{C.9}$$

where m_c is the component molar density and b^{α} is the phase molar density.

Derivatives

The derivatives of the saturation wrt. m_c and p are given by

$$\frac{\partial S^{\alpha}}{\partial m_c} = \frac{\partial}{\partial m_c}\left(\frac{m_c}{b^{\alpha}}\right) = \frac{1}{b^{\alpha}}, \tag{C.10}$$

and

$$\frac{\partial S^{\alpha}}{\partial p} = \frac{\partial}{\partial p}\left(\frac{m_c}{b^{\alpha}}\right) = -\frac{m_c}{(b^{\alpha})^2}\frac{\partial b^{\alpha}}{\partial p} = -\frac{S^{\alpha}}{b^{\alpha}}\frac{\partial b^{\alpha}}{\partial p}, \tag{C.11}$$

where $\frac{\partial b^{\alpha}}{\partial p}$ is defined in (C.8).

C.4 Viscosity

The viscosity of a fluid is a measure of its resistance to flow. The computation of the viscosity is completely analogous to that of the formation volume factor B^{α}. Like for the formation volume factor, the viscosity for the oil and gas phases are computed differently than the viscosity for the water phase.

Oil and gas

The viscosity of oil μ^o and the viscosity of gas μ^g depend on pressure and is computed using linear interpolation on the table values in Table C.2 and Table C.3.

Based on one of these tables the viscosity is calculated as

$$\mu^\alpha = \mu^\alpha[1] + (p - p[1])\frac{\mu^\alpha[2] - \mu^\alpha[1]}{p[2] - p[1]}, \quad \alpha = o, g, \tag{C.12}$$

where p is the actual pressure in the grid cell and [ ] indicates an index number in the interpolation table.

Example:

If a cell has pressure $p = 300$, the viscosity is calculated as

$$\mu = 0.014 + (300 - 50)\frac{0.025 - 0.014}{600 - 50} = 0.019 \tag{C.13}$$

Derivatives for oil and gas

The derivative of (C.12) wrt. p is

$$\frac{\partial \mu^\alpha}{\partial p} = \frac{\mu^\alpha[2] - \mu^\alpha[1]}{p[2] - p[1]}, \quad \alpha = o, g \tag{C.14}$$

Water

The viscosity for water also depends on pressure, but instead of linear interpolation the following formula is used

$$\begin{aligned} X_v &= C_v \cdot (p - p_{ref}) \\ \mu^w &= \frac{\mu^w_{pref}}{1 + X_v + \frac{1}{2}X_v^2}, \end{aligned} \tag{C.15}$$

Here C_v is a constant indicating how much the viscosity of water changes as pressure changes, p_{ref} is a reference pressure and μ^w_{pref} is a reference viscosity. In the implementation all of these are user-given inputs.

Derivative for water

The derivation of the derivative of (C.15) wrt. p is completely analogous to the derivation of $\frac{\partial B^w}{\partial p}$ in (C.5). Thus

$$\frac{\partial \mu^w}{\partial p} = -\mu^w_{pref} \frac{(C + C^2 \cdot (p - p_{ref}))}{\left(1 + C \cdot (p - p_{ref}) + \frac{1}{2}(C \cdot (p - p_{ref}))^2\right)^2} \tag{C.16}$$

C.5 Mass density

The mass density ρ^α is defined as the mass per volume and it is given by

$$\rho^\alpha = \frac{\rho^\alpha_{sc}}{B^\alpha}, \tag{C.17}$$

where ρ^α_{sc} is the mass density at standard conditions (given in Table A.1) and B^α the formation volume factor from either (C.1) or (C.4).

Note that the mass density used in the reconstruction of the flux across an interface in (3.31) differs from the mass density defined in equation (C.17). In order to ensure symmetry regardless of the flow direction we are using a saturation weighted average

$$\rho^\alpha_{ij} = \frac{S^\alpha_i \rho^\alpha_i + S^\alpha_j \rho^\alpha_j}{S^\alpha_i + S^\alpha_j}, \quad i \in \mathcal{C}, \quad j \in \mathcal{N}^{(i)}, \tag{C.18}$$

where S^α is given in (C.9).

Derivatives

The mass density ρ^α depends on pressure alone and the derivative of (C.17) wrt. p is given by

$$\frac{\partial \rho^\alpha}{\partial p} = \frac{\partial}{\partial p}\left(\frac{\rho^\alpha_{sc}}{B^\alpha}\right) = \rho^\alpha_{sc}\frac{\partial}{\partial p}\left(\frac{1}{B^\alpha}\right) = -\frac{\rho^\alpha_{sc}}{(B^\alpha)^2}\frac{\partial B^\alpha}{\partial p} = -\frac{\rho^\alpha}{B^\alpha}\frac{\partial B^\alpha}{\partial p}, \tag{C.19}$$

where $\frac{\partial B^\alpha}{\partial p}$ is defined in either (C.3) or (C.5).

The saturation weighted average of the mass density in (C.18) is dependent on m_c and

p at both cell i and its neighbour cell j, leading to the derivatives

$$\frac{\partial \rho_{ij}^{\alpha}}{\partial m_{c,i}} = \frac{\partial}{\partial m_{c,i}} \left(\frac{S_i^{\alpha} \rho_i^{\alpha} + S_j^{\alpha} \rho_j^{\alpha}}{S_i^{\alpha} + S_j^{\alpha}} \right)$$

$$= \frac{\left(\frac{\partial S_i^{\alpha} \rho_i^{\alpha}}{\partial m_{c,i}} + \overbrace{\frac{\partial S_j^{\alpha} \rho_j^{\alpha}}{\partial m_{c,i}}}^{=0} \right) (S_i^{\alpha} + S_j^{\alpha}) - \left(S_i^{\alpha} \rho_i^{\alpha} + S_j^{\alpha} \rho_j^{\alpha} \right) \left(\frac{\partial S_i^{\alpha}}{\partial m_{c,i}} + \overbrace{\frac{\partial S_j^{\alpha}}{\partial m_{c,i}}}^{=0} \right)}{\left(S_i^{\alpha} + S_j^{\alpha} \right)^2}$$

$$= \frac{\frac{\partial S_i^{\alpha}}{\partial m_{c,i}} \rho_i^{\alpha} - \rho_{ij}^{\alpha} \frac{\partial S_i^{\alpha}}{\partial m_{c,i}}}{\left(S_i^{\alpha} + S_j^{\alpha} \right)} = \frac{\frac{\partial S_i^{\alpha}}{\partial m_{c,i}} \left(\rho_i^{\alpha} - \rho_{ij}^{\alpha} \right)}{\left(S_i^{\alpha} + S_j^{\alpha} \right)}, \tag{C.20}$$

where $i \in \mathcal{C}$ and $j \in \mathcal{N}^{(i)}$ and $\frac{\partial S_i^{\alpha}}{\partial m_{c,i}}$ is defined in (C.10). Following the same approach for the derivative wrt. m_c, j leads to

$$\frac{\partial \rho_{ij}^{\alpha}}{\partial m_{c,j}} = \frac{\frac{\partial S_j^{\alpha}}{\partial m_{c,j}} \left(\rho_j^{\alpha} - \rho_{ij}^{\alpha} \right)}{\left(S_i^{\alpha} + S_j^{\alpha} \right)}, \quad i \in \mathcal{C} \quad j \in \mathcal{N}^{(i)} \tag{C.21}$$

The derivative of (C.18) wrt to p_i is given by

$$\frac{\partial \rho_{ij}^{\alpha}}{\partial p_i} = \frac{\partial}{\partial p_i} \left(\frac{S_i^{\alpha} \rho_i^{\alpha} + S_j^{\alpha} \rho_j^{\alpha}}{S_i^{\alpha} + S_j^{\alpha}} \right)$$

$$= \frac{\left(\frac{\partial S_i^{\alpha} \rho_i^{\alpha}}{\partial p_i} + \overbrace{\frac{\partial S_j^{\alpha} \rho_j^{\alpha}}{\partial p_i}}^{=0} \right) (S_i^{\alpha} + S_j^{\alpha}) - \left(S_i^{\alpha} \rho_i^{\alpha} + S_j^{\alpha} \rho_j^{\alpha} \right) \left(\frac{\partial S_i^{\alpha}}{\partial p_i} + \overbrace{\frac{\partial S_j^{\alpha}}{\partial p_i}}^{=0} \right)}{\left(S_i^{\alpha} + S_j^{\alpha} \right)^2}$$

$$= \frac{\frac{\partial S_i^{\alpha} \rho_i^{\alpha}}{\partial p_i} - \rho_{ij}^{\alpha} \frac{\partial S_i^{\alpha}}{\partial p_i}}{\left(S_i^{\alpha} + S_j^{\alpha} \right)} = \frac{\frac{\partial S_i^{\alpha}}{\partial p_i} \rho_i^{\alpha} + S_i^{\alpha} \frac{\partial \rho_i^{\alpha}}{\partial p_i} - \rho_{ij}^{\alpha} \frac{\partial S_i^{\alpha}}{\partial p_i}}{\left(S_i^{\alpha} + S_j^{\alpha} \right)} = \frac{\frac{\partial S_i^{\alpha}}{\partial p_i} \left(\rho_i^{\alpha} - \rho_{ij}^{\alpha} \right) + S_i^{\alpha} \frac{\partial \rho_i^{\alpha}}{\partial p_i}}{\left(S_i^{\alpha} + S_j^{\alpha} \right)}, \tag{C.22}$$

where $i \in \mathcal{C}$ and $j \in \mathcal{N}^{(i)}$. The derivative $\frac{\partial S_i^\alpha}{\partial p_i}$ is defined in (C.11) and $\frac{\partial \rho_i^\alpha}{\partial p_i}$ is defined in (C.19). Following the same approach for the derivative wrt. p_j leads to

$$\frac{\partial \rho_{ij}^\alpha}{\partial p_j} = \frac{\frac{\partial S_j^\alpha}{\partial p_j}\left(\rho_j^\alpha - \rho_{ij}^\alpha\right) + S_j^\alpha \frac{\partial \rho_j^\alpha}{\partial p_j}}{\left(S_i^\alpha + S_j^\alpha\right)}, \quad i \in \mathcal{C} \quad j \in \mathcal{N}^{(i)} \tag{C.23}$$

C.6 Relative permeability

The relative permeability describes how the different fluids flow in the presence of each other. It is a dimensionless term devised to adapt Darcy's law (2.12) to multiphase flow conditions and it is defined as the ratio of effective permeability of a particular fluid at a particular saturation to absolute permeability of that fluid at total saturation.

If there is only a single fluid in a porous medium, its relative permeability is 1.

For simplicity we choose to use "miscible" relative permeabilities meaning

$$k_r^\alpha = S^\alpha \tag{C.24}$$

In the implementation we use normalized saturations to compute the relative permeabilities. This is done to ensure stability in case of numerical discrepancies.

Derivatives

The relative permeability depends on both m_c and p yielding

$$\frac{\partial k_r^\alpha}{\partial m_c} = \frac{\partial S^\alpha}{\partial m_c}, \tag{C.25}$$

and

$$\frac{\partial k_r^\alpha}{\partial p} = \frac{\partial S^\alpha}{\partial p}, \tag{C.26}$$

where $\frac{\partial S^\alpha}{\partial m_c}$ and $\frac{\partial S^\alpha}{\partial p}$ are defined in (C.10) and (C.11) respectively.

C.7 Porosity

The porosity is the percentage of the volume within a porous medium that is able to contain fluids. To account for rock compressibility under pressure we use

$$\begin{aligned} X_r &= C_{rock}(p - p_{ref}) \\ \phi &= \varphi(1 + X_r + \frac{1}{2}X_r^2), \end{aligned} \tag{C.27}$$

where φ is a porosity constant (Table A.2), which is a reservoir specific input, p_{ref} is a reference pressure, C_{rock} is the compressibility of the rock and p is the pressure.

Derivative

The derivative of (C.27) wrt. p is given by

$$
\begin{aligned}
\frac{\partial \phi}{\partial p} &= \frac{\partial\left(\varphi(1+X_r+\frac{1}{2}X_r^2)\right)}{\partial p} \\
&= \varphi\left(\frac{\partial X_r}{\partial p}+\frac{1}{2}\frac{\partial X_r^2}{\partial p}\right) \\
&= \varphi\left(\frac{\partial\left(C_{rock}(p-p_{ref})\right)}{\partial p}+\frac{1}{2}\frac{\partial\left(C_{rock}^2(p-p_{ref})^2\right)}{\partial p}\right) \\
&= \varphi\left(C_{rock}+\frac{C_{rock}^2}{2}\frac{\partial\left(p^2+p_{ref}^2-2pp_{ref}\right)}{\partial p}\right) \\
&= \varphi\left(C_{rock}+\frac{C_{rock}^2}{2}\left(2p-2p_{ref}\right)\right) \\
&= \varphi\left(C_{rock}+C_{rock}^2(p-p_{ref})\right) \\
&= \varphi\left(C_{rock}+C_{rock}X_r\right)
\end{aligned}
\tag{C.28}
$$

Appendix D

Derivatives of residuals

In this appendix, all derivatives used to set up the Jacobian described in section 4.2 are derived. This amounts to finding the derivative of the residuals with respect to molar densities m_c and pressure p. In the Jacobian we need both the derivatives in the diagonal blocks and in the off-diagonal blocks, which means we want to derive

- $\dfrac{\partial r_{c,i}}{\partial m_{c,i}}$ and $\dfrac{\partial r_{c,i}}{\partial m_{c,j}}$
- $\dfrac{\partial r_{c,i}}{\partial p_i}$ and $\dfrac{\partial r_{c,i}}{\partial p_j}$
- $\dfrac{\partial r_{vb,i}}{\partial m_{c,i}}$ and $\dfrac{\partial r_{vb,i}}{\partial p_i}$

The residual for the volume balance constraint does not contain a flow term which means it has no dependency on $m_{c,j}$ and p_j, where $j \in \mathcal{N}^{(i)}$. For this reason it is only derived wrt. $m_{c,i}$ and p_i.

D.1 Derivatives with respect to $m_{c,i}$

The derivatives wrt. mass of the residuals $r_{c,i}$ and $r_{vb,i}$ in the diagonal blocks in the Jacobian are derived in this section.

Derivative of $r_{c,i}$

We consider the derivatives of the first three equations in equation (4.12). We can use the α-notation, since the derivations are the same for all three components.

$$\frac{\partial r_{c,i}^{n+1}}{\partial m_{c,i}^{n+1}} = \frac{\partial\left(\Delta t\, \overbrace{s_i^{\alpha,n+1}}^{\text{Source}} + \Delta t\, \overbrace{f_i^{\alpha,n+1}}^{\text{Flow}} + \overbrace{(V_p m_c)_i^n}^{\text{Accumulation 1}} - \overbrace{(V_p m_c)_i^{n+1}}^{\text{Accumulation 2}}\right)}{\partial m_{c,i}^{n+1}} \tag{D.1}$$

The accumulation terms and the flow term are treated separately. Note that the derivative of the accumulation term at time step n wrt. m_c^{n+1} or p^{n+1} is equal to zero. We therefore omit the time step indication throughout this appendix to keep the notation more compact. However, all terms in this appendix are evaluated at time step $n+1$. The derivative of the source term evaluates to zero.

Accumulation terms

Keeping in mind that the volume V_i of the ith grid cell (V_i is equal to V since we consider regularly structured grids) is independent of mass and pressure and that ϕ only depends on the pressure, the derivative of the accumulation term is

$$\frac{\partial (V_p m)_{c,i}}{\partial m_{c,i}} = \frac{\partial (V\phi m)_{c,i}}{\partial m_{c,i}} = (V\phi)_i \tag{D.2}$$

Flow term

To keep notation compact we introduce

$$\Phi_{ij}^\alpha = \Delta p - \rho^\alpha g \Delta z \tag{D.3}$$

The derivative of f_i^α in (3.31) wrt. $m_{c,i}$ is given by

$$\begin{aligned}
\frac{\partial f_i^\alpha}{\partial m_{c,i}} &= \sum_{j\in\mathcal{N}^{(i)}} T_{ij} \frac{\partial\left(\lambda^\alpha(\Delta p - \rho^\alpha g \Delta z)\right)_{ij}}{\partial m_{c,i}} \\
&= \sum_{j\in\mathcal{N}^{(i)}} T_{ij} \left(\frac{\partial \lambda_{ij}^\alpha}{\partial m_{c,i}} \Phi_{ij}^\alpha + \lambda_{ij}^\alpha \frac{\overbrace{\partial(\Delta p}^{=0} - \rho^\alpha g \Delta z)_{ij}}{\partial m_{c,i}} \right) \\
&= \sum_{j\in\mathcal{N}^{(i)}} T_{ij} \begin{cases} \dfrac{\partial \lambda_i^\alpha}{\partial m_{c,i}} \Phi_{ij}^\alpha - \lambda_i^\alpha g \Delta z \dfrac{\partial (\rho^\alpha)_{ij}}{\partial m_{c,i}}, & \text{if } \Phi_{ij}^\alpha < 0 \\[2ex] \underbrace{\dfrac{\partial \lambda_j^\alpha}{\partial m_{c,i}}}_{=0} \Phi_{ij}^\alpha - \lambda_j^\alpha g \Delta z \dfrac{\partial \rho_{ij}^\alpha}{\partial m_{c,i}}, & \text{if } \Phi_{ij}^\alpha > 0 \end{cases}
\end{aligned}$$

$$= \sum_{j \in \mathcal{N}^{(i)}} T_{ij} \begin{cases} \dfrac{\partial \lambda_i^\alpha}{\partial m_{c,i}} \Phi_{ij}^\alpha - \lambda_i^\alpha g \Delta z \dfrac{\partial (\rho^\alpha)_{ij}}{\partial m_{c,i}}, & \text{if } \Phi_{ij}^\alpha < 0 \\ \\ -\lambda_j^\alpha g \Delta z \dfrac{\partial \rho_{ij}^\alpha}{\partial m_{c,i}}, & \text{if } \Phi_{ij}^\alpha > 0 \end{cases} \tag{D.4}$$

where $i \in \mathcal{C}$, $j \in \mathcal{N}^{(i)}$ and $\frac{\partial \rho_{ij}^\alpha}{\partial m_{c,i}}$ is defined in (C.20). The derivative of the mobility is

$$\frac{\partial \lambda_i^\alpha}{\partial m_{c,i}} = \frac{\partial}{\partial m_{c,i}} \left(\frac{b^\alpha k_r^\alpha}{\mu^\alpha} \right)_i = \frac{b_i^\alpha}{\mu_i^\alpha} \frac{\partial k_{r,i}^\alpha}{\partial m_{c,i}} = \frac{b_i^\alpha}{\mu_i^\alpha} \frac{\partial S_i^\alpha}{\partial m_{c,i}}, \tag{D.5}$$

where $k_r^\alpha = S^\alpha$ according to section C.6 and $\frac{\partial S_i^\alpha}{\partial m_{c,i}}$ is defined in (C.10).

Derivative of $r_{vb,i}$

Using that $S^\alpha = \dfrac{m_c}{b^\alpha}$, the derivative of the residual for the volume balance constraint is derived in equation (D.6).

$$\begin{aligned} \frac{\partial r_{vb,i}}{\partial m_{c,i}} &= \frac{\partial \left(S_i^o + S_i^g + S_i^w - 1 \right)}{\partial m_{c,i}} \\ &= \frac{1}{b_i^\alpha} \end{aligned} \tag{D.6}$$

D.2 Derivatives with respect to $m_{c,j}$

The derivatives wrt. mass of the residuals in the off-diagonal blocks in the Jacobian are derived in this section. The derivatives of the residual for the volume balance constraint evaluates to zero leaving only the derivatives of the first three equations in (4.12).

$$\frac{\partial r_{c,i}^{n+1}}{\partial m_{c,j}^{n+1}} = \frac{\partial \left(\Delta t \, \overbrace{s_i^{\alpha,n+1}}^{\text{Source}} + \Delta t \, \overbrace{f_i^{\alpha,n+1}}^{\text{Flow}} + \overbrace{(V_p m_c)_i^n}^{\text{Accumulation 1}} - \overbrace{(V_p m_c)_i^{n+1}}^{\text{Accumulation 2}} \right)}{\partial m_{c,j}^{n+1}} \tag{D.7}$$

The source and accumulation terms evaluate to zero leaving only the flow term to be derived.

Flow term

The derivative of f_i^α in (3.31) wrt. $m_{c,j}$ is given by

$$\begin{aligned}
\frac{\partial f_i^\alpha}{\partial m_{c,j}} &= \sum_{k\in\mathcal{N}^{(i)}} T_{ik}\frac{\partial\left(\lambda^\alpha(\Delta p - \rho^\alpha g\Delta z)\right)_{ik}}{\partial m_{c,j}} \\
&= T_{ij}\frac{\partial\left(\lambda^\alpha(\Delta p - \rho^\alpha g\Delta z)\right)_{ij}}{\partial m_{c,j}} + \overbrace{\sum_{k\in\mathcal{N}^{(i)}\setminus\{j\}} T_{ik}\frac{\partial\left(\lambda^\alpha(\Delta p - \rho^\alpha g\Delta z)\right)_{ik}}{\partial m_{c,j}}}^{=0} \\
&= T_{ij}\left(\frac{\partial \lambda_{ij}^\alpha}{\partial m_{c,j}}\Phi_{ij}^\alpha + \lambda_{ij}^\alpha\frac{\overbrace{\partial(\Delta p}^{=0} - \rho^\alpha g\Delta z)_{ij}}{\partial m_{c,j}}\right) \\
&= T_{ij}\begin{cases}\overbrace{\frac{\partial \lambda_i^\alpha}{\partial m_{c,j}}}^{=0}\Phi_{ij}^\alpha - \lambda_i^\alpha g\Delta z\frac{\partial \rho_{ij}^\alpha}{\partial m_{c,j}}, & \text{if } \Phi_{ij}^\alpha < 0 \\[2ex]
\frac{\partial \lambda_j^\alpha}{\partial m_{c,j}}\Phi_{ij}^\alpha - \lambda_j^\alpha g\Delta z\frac{\partial \rho_{ij}^\alpha}{\partial m_{c,j}}, & \text{if } \Phi_{ij}^\alpha > 0\end{cases} \\
&= T_{ij}\begin{cases}-\lambda_i^\alpha g\Delta z\frac{\partial \rho_{ij}^\alpha}{\partial m_{c,j}}, & \text{if } \Phi_{ij}^\alpha < 0 \\
\frac{\partial \lambda_j^\alpha}{\partial m_{c,j}}\Phi_{ij}^\alpha - \lambda_j^\alpha g\Delta z\frac{\partial \rho_{ij}^\alpha}{\partial m_{c,j}}, & \text{if } \Phi_{ij}^\alpha > 0\end{cases}
\end{aligned} \tag{D.8}$$

where $i \in \mathcal{C}$ and $k, j \in \mathcal{N}^{(i)}$ and $\frac{\partial \rho_{ij}^\alpha}{\partial m_{c,j}}$ is defined in (C.21). Following an approach similar to (D.5) the derivative of the mobility is found to

$$\frac{\partial \lambda_j^\alpha}{\partial m_{c,j}} = \frac{b_j^\alpha}{\mu_j^\alpha}\frac{\partial S_j^\alpha}{\partial m_{c,j}}, \tag{D.9}$$

where $\frac{\partial S_j^\alpha}{\partial m_{c,j}}$ is defined in (C.10).

D.3 Derivatives with respect to p_i

The derivatives wrt. pressure of the residuals $r_{c,i}$ and $r_{vb,i}$ in the diagonal blocks in the Jacobian are derived in this section.

Derivative of $r_{c,i}$

Again we consider the derivatives of the first three equations in equation (4.12)

$$\frac{\partial r_{c,i}^{n+1}}{\partial p_i^{n+1}} = \frac{\partial\left(\Delta t\,\overbrace{s_i^{\alpha,n+1}}^{\text{Source}} + \Delta t\,\overbrace{f_i^{\alpha,n+1}}^{\text{Flow}} + \overbrace{(V_p m_c)_i^n}^{\text{Accumulation 1}} - \overbrace{(V_p m_c)_i^{n+1}}^{\text{Accumulation 2}}\right)}{\partial p_i^{n+1}} \tag{D.10}$$

The derivative of the source term evaluates to zero.

Accumulation terms

As earlier mentioned the derivative of the accumulation term at time step n wrt. p evaluates to zero. The derivative of the accumulation term at time step $n+1$ wrt. p is

$$\frac{\partial(V_p m)_{c,i}}{\partial p_i} = \frac{\partial(V\phi m)_{c,i}}{\partial p_i} = V_i m_{c,i}\frac{\partial \phi}{\partial p_i}, \tag{D.11}$$

where $\frac{\partial \phi}{\partial p_i}$ is defined in (C.28).

Flow term

The derivative of f_i^α in (3.31) wrt. p_i is given by

$$\begin{aligned}
\frac{\partial f_i^\alpha}{\partial p_i} &= \sum_{j\in\mathcal{N}^{(i)}} T_{ij}\frac{\partial\left(\lambda^\alpha(\Delta p - \rho^\alpha g\Delta z)\right)_{ij}}{\partial p_i} \\
&= \sum_{j\in\mathcal{N}^{(i)}} T_{ij}\left(\frac{\partial \lambda_{ij}^\alpha}{\partial p_i}\Phi_{ij}^\alpha + \lambda_{ij}^\alpha\frac{\overbrace{\partial(\Delta p}^{=-1} - \rho^\alpha g\Delta z)_{ij}}{\partial p_i}\right) \\
&= \sum_{j\in\mathcal{N}^{(i)}} T_{ij}\begin{cases} \dfrac{\partial \lambda_i^\alpha}{\partial p_i}\Phi_{ij}^\alpha + \lambda_i^\alpha\left(-1 - \dfrac{\partial \rho_{ij}^\alpha}{\partial p_i}g\Delta z\right), & \text{if } \Phi_{ij}^\alpha < 0 \\ \underbrace{\dfrac{\partial \lambda_j^\alpha}{\partial p_i}}_{=0}\Phi_{ij}^\alpha + \lambda_j^\alpha\left(-1 - \dfrac{\partial \rho_{ij}^\alpha}{\partial p_i}g\Delta z\right), & \text{if } \Phi_{ij}^\alpha > 0 \end{cases}
\end{aligned} \tag{D.12}$$

$$= \sum_{j\in\mathcal{N}^{(i)}} T_{ij} \begin{cases} \dfrac{\partial \lambda_i^\alpha}{\partial p_i}\Phi_{ij}^\alpha + \lambda_i^\alpha \left(-1 - \dfrac{\partial \rho_{ij}^\alpha}{\partial p_i} g\Delta z\right), & \text{if } \Phi_{ij}^\alpha < 0 \\ \\ \lambda_j^\alpha \left(-1 - \dfrac{\partial \rho_{ij}^\alpha}{\partial p_i} g\Delta z\right), & \text{if } \Phi_{ij}^\alpha > 0 \end{cases} \tag{D.13}$$

where $i \in \mathcal{C}$, $j \in \mathcal{N}^{(i)}$ and $\frac{\partial \rho_{ij}^\alpha}{\partial p_i}$ is defined in (C.22). The derivative of the mobility is

$$\frac{\partial \lambda_i^\alpha}{\partial p_i} = \frac{\partial \left(\dfrac{b^\alpha k_r^\alpha}{\mu^\alpha}\right)_i}{\partial p_i} = \frac{\dfrac{\partial\left(b_i^\alpha k_{r,i}^\alpha\right)}{\partial p_i}\mu_i^\alpha - b_i^\alpha k_{r,i}^\alpha \dfrac{\partial \mu_i^\alpha}{\partial p_i}}{\left(\mu_i^\alpha\right)^2}$$

$$= \frac{\dfrac{\partial\left(b_i^\alpha k_{r,i}^\alpha\right)}{\partial p_i} - \lambda_i^\alpha \dfrac{\partial \mu_i^\alpha}{\partial p_i}}{\mu_i^\alpha} = \frac{b_i^\alpha \dfrac{\partial k_{r,i}^\alpha}{\partial p_i} + k_{r,i}^\alpha \dfrac{\partial b_i^\alpha}{\partial p_i} - \lambda_i^\alpha \dfrac{\partial \mu_i^\alpha}{\partial p_i}}{\mu_i^\alpha}, \tag{D.14}$$

where $\frac{\partial k_{r,i}^\alpha}{\partial p_i} = \frac{\partial S_i^\alpha}{\partial p_i}$ given in (C.11), $\frac{\partial b_i^\alpha}{\partial p_i}$ is given in (C.8) and $\frac{\partial \mu_i^\alpha}{\partial p_i}$ is given in (C.14) or (C.16).

Derivatives of $r_{vb,i}$

The derivatives of the residual for the volume balance constraint wrt. p_i are

$$\frac{\partial r_{vb,i}}{\partial p_i} = \frac{\partial\left(S_i^o + S_i^g + S_i^w - 1\right)}{\partial p_i} = \frac{\partial S_i^o}{\partial p_i} + \frac{\partial S_i^g}{\partial p_i} + \frac{\partial S_i^w}{\partial p_i}, \tag{D.15}$$

where $\frac{\partial S_i^\alpha}{\partial p_i}$ is defined in (C.11).

D.4 Derivatives with respect to p_j

The derivatives of the residuals in the off-diagonal blocks in the Jacobian are derived in this section. The derivatives of the residual for the volume balance constraint evaluate to zero leaving only the derivatives of the first three equations in (4.12).

$$\frac{\partial r_{c,i}^{n+1}}{\partial p_j^{n+1}} = \frac{\partial \left(\Delta t \overbrace{s_i^{\alpha,n+1}}^{\text{Source}} + \Delta t \overbrace{f_i^{\alpha,n+1}}^{\text{Flow}} + \overbrace{\left(V_p m_c\right)_i^n}^{\text{Accumulation 1}} - \overbrace{\left(V_p m_c\right)_i^{n+1}}^{\text{Accumulation 2}}\right)}{\partial p_j^{n+1}} \tag{D.16}$$

The source and accumulation terms evaluate to zero leaving only the flow term to be derived.

Flow term

The derivative of f_i^α in (3.31) wrt. p_j is given by

$$\begin{aligned}
\frac{\partial f_i^\alpha}{\partial p_j} &= \sum_{k\in\mathcal{N}^{(i)}} T_{ik}\frac{\partial\left(\lambda^\alpha(\Delta p-\rho^\alpha g\Delta z)\right)_{ik}}{\partial p_j}\\
&= T_{ij}\frac{\partial\left(\lambda^\alpha(\Delta p-\rho^\alpha g\Delta z)\right)_{ij}}{\partial p_j} + \overbrace{\sum_{k\in\mathcal{N}^{(i)}\backslash\{j\}} T_{ik}\frac{\partial\left(\lambda^\alpha(\Delta p-\rho^\alpha g\Delta z)\right)_{ik}}{\partial p_j}}^{=0}\\
&= T_{ij}\left(\frac{\partial\lambda_{ij}^\alpha}{\partial p_j}\Phi_{ij}^\alpha + \lambda_{ij}^\alpha\frac{\overbrace{\partial(\Delta p}^{=1}-\rho^\alpha g\Delta z)_{ij}}{\partial p_j}\right)\\
&= T_{ij}\begin{cases}\overbrace{\frac{\partial\lambda_i^\alpha}{\partial p_j}}^{=0}\Phi_{ij}^\alpha + \lambda_i^\alpha\left(1-\frac{\partial\rho_{ij}^\alpha}{\partial p_j}g\Delta z\right), & \text{if } \Phi_{ij}^\alpha<0\\[2ex] \frac{\partial\lambda_j^\alpha}{\partial p_j}\Phi_{ij}^\alpha + \lambda_j^\alpha\left(1-\frac{\partial\rho_{ij}^\alpha}{\partial p_j}g\Delta z\right), & \text{if } \Phi_{ij}^\alpha>0\end{cases} && \text{(D.17)}\\
&= T_{ij}\begin{cases}\lambda_i^\alpha\left(1-\frac{\partial\rho_{ij}^\alpha}{\partial p_j}g\Delta z\right), & \text{if } \Phi_{ij}^\alpha<0\\[2ex] \frac{\partial\lambda_j^\alpha}{\partial p_j}\Phi_{ij}^\alpha + \lambda_j^\alpha\left(1-\frac{\partial\rho_{ij}^\alpha}{\partial p_j}g\Delta z\right), & \text{if } \Phi_{ij}^\alpha>0\end{cases} && \text{(D.18)}
\end{aligned}$$

where $i\in\mathcal{C}$ and $k,j\in\mathcal{N}^{(i)}$ and $\frac{\partial\rho_{ij}^\alpha}{\partial p_j}$ is defined in (C.23). Following an approach similar to (D.14) the derivative of the mobility is found to

$$\frac{\partial\lambda_j^\alpha}{\partial p_j} = \frac{b_j^\alpha\frac{\partial k_{r,j}^\alpha}{\partial p_j} + k_{r,j}^\alpha\frac{\partial b_j^\alpha}{\partial p_j} - \lambda_j^\alpha\frac{\partial\mu_j^\alpha}{\partial p_j}}{\mu_j^\alpha}, \tag{D.19}$$

where $\frac{\partial k_{r,j}^\alpha}{\partial p_j} = \frac{\partial S_j^\alpha}{\partial p_j}$ given in (C.11), $\frac{\partial b_j^\alpha}{\partial p_j}$ is given in (C.8) and $\frac{\partial\mu_j^\alpha}{\partial p_j}$ is given in (C.14) or (C.16).

Appendix E

Generating permeability fields

Many of the test cases used throughout the thesis are using a generated permeability field based on “randomly” generated numbers and trigonometric functions. Using the same seed number, this enables us to generate permeability fields, which all have the same pattern independent of the size of the problem. Furthermore, we can set the range of permeabilities to generate anything from easy permeability fields to very demanding fields. Also, the variation of permeabilities in each dimension can be controlled to emulate real reservoirs, where the permeabilities in the z-direction often vary more than in the x- and y-directions.

The algorithm for generating permeability fields is given in Algorithm E.1. Notice that i and j in the algorithm do not indicate a given cell and its neighbour, but instead i and j are indexing coordinates in a regularly structured 3D grid. The inputs ox, oy, and oz control the variation of permeabilities in each dimension by setting the length of the sums. The inputs min and max determine the range of permeabilities.

Algorithm E.1 Generate permeabilities $\mathbf{k}$ = genPermFields(min,max,ox,oy,oz)

```
1: Set seed for random value generator
2: lx = $\Delta x N_x$; ly = $\Delta y N_y$; lz = $\Delta z N_z$ {Periodic scaling constants}
3: for ok = 0...oz-1 do
4:   for oj = 0...oy-1 do
5:     for oi = 0...ox-1 do
6:       scoeffs = rand {Generate random values}
7:       ccoeffs = rand
8:       xsscale = (rand·$\pi$)/lx {Generate random value with periodic scaling}
9:       ysscale = (rand·$\pi$)/ly
10:      zsscale = (rand·$\pi$)/lz
11:      xcscale = (rand·$\pi$)/lx
12:      ycscale = (rand·$\pi$)/ly
13:      zcscale = (rand·$\pi$)/lz
14:      Set all values in $k$ equal to zero
15:      for $k = 0 \dots N_z - 1$ do
16:        for $j = 0 \dots N_y - 1$ do
17:          for $i = 0 \dots N_x - 1$ do
18:            $k_{i,j,k} += \text{scoeffs} \cdot \sin(\text{oi} \cdot (\Delta x \cdot i + 0.5 \cdot \Delta x) \cdot \text{xsscale}) \cdot$
19:                           $\sin(\text{oj} \cdot (\Delta y \cdot j + 0.5 \cdot \Delta y) \cdot \text{ysscale}) \cdot$
20:                           $\sin(\text{ok} \cdot (\Delta z \cdot k + 0.5 \cdot \Delta z) \cdot \text{zsscale}) \cdot$
21:            $k_{i,j,k} += \text{ccoeffs} \cdot \cos(\text{oi} \cdot (\Delta x \cdot i + 0.5 \cdot \Delta x) \cdot \text{xcscale}) \cdot$
22:                           $\cos(\text{oj} \cdot (\Delta y \cdot j + 0.5 \cdot \Delta y) \cdot \text{ycscale}) \cdot$
23:                           $\cos(\text{ok} \cdot (\Delta z \cdot k + 0.5 \cdot \Delta z) \cdot \text{zcscale}) \cdot$
24:          end for
25:        end for
26:      end for
27:    end for
28:  end for
29: end for
30: Scale values in $\mathbf{k}$ between min and max
```

Appendix F

Additional plots

This chapter contains additional plots used to support some of the conclusions drawn in the thesis.

F.1 Increasing the number of smoothings

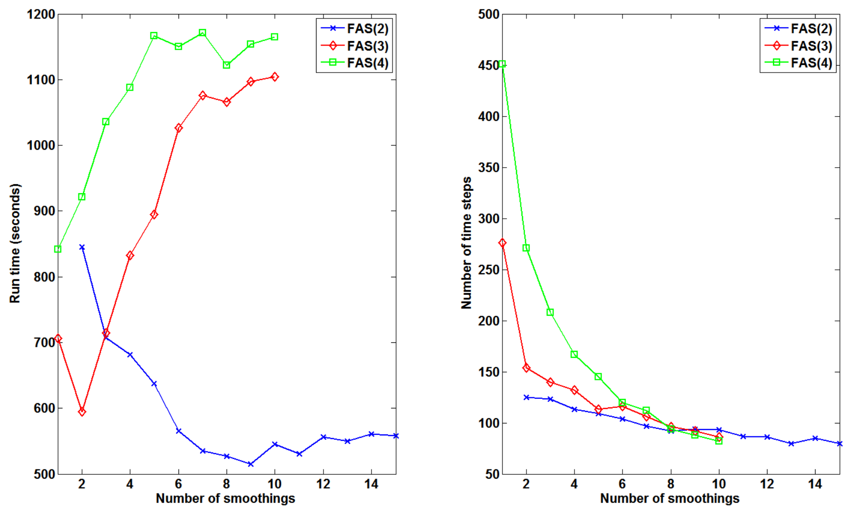


Figure F.1: *Run time and number of time steps as a function of number of smoothings.Problem:* $32 \times 32 \times 32$ *smooth heterogeneous permeability field problem with permeabilities in the range 200-600 mD. Cell size:* $20m \times 10m \times 2m$.

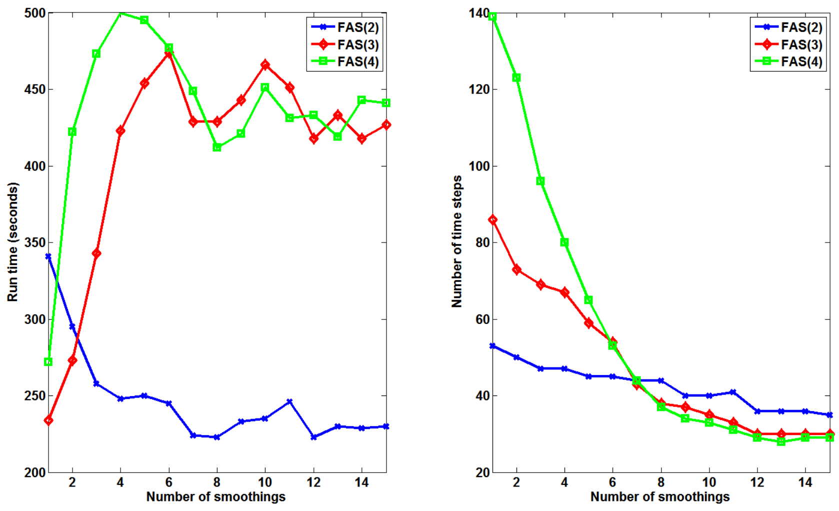


Figure F.2: *Run time and number of time steps as a function of number of smoothings.Problem:* $32 \times 32 \times 32$ *homogeneous permeability field problem with permeabilities of 100 mD. Cell size:* $20m \times 10m \times 2m$.

F.2 Verifying FAS using (x,y)-semicoarsening

In this section the FAS simulator using 3 or 4 grid levels with (x, y)-semicoarsening is verified by comparing solutions to the solutions from ECLIPSE.

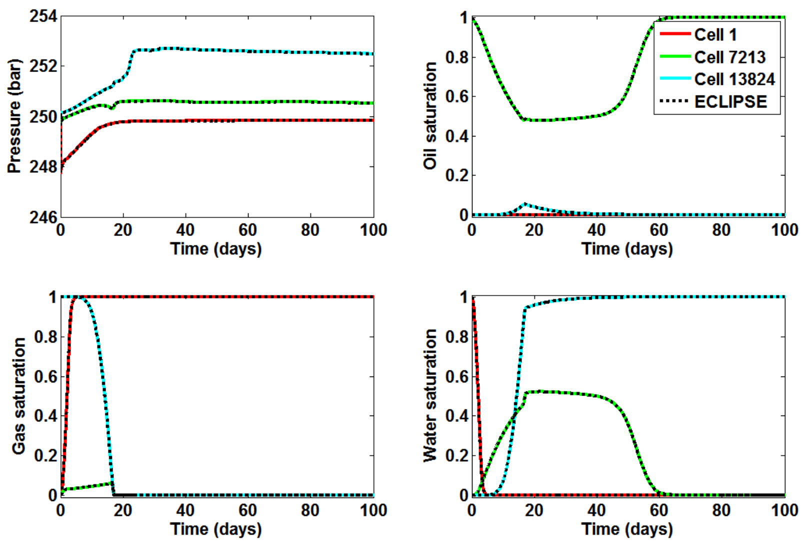


Figure F.3: *Verification of FAS(3) using (x, y)-semicoarsening for $24 \times 24 \times 24$ problem with smooth heterogeneous permeability field, for 100 days, initial time step 0.1 days, $\Delta t_{max} = 30$ days, 5- pre and 5 post-smoothings, and no more than 10 FAS cycles and 10 newton iterations per FAS cycle.*

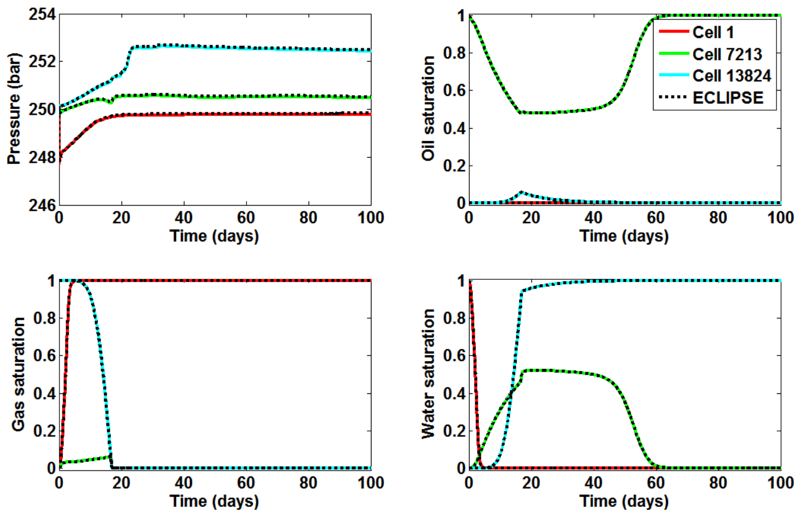


Figure F.4: *Verification of FAS(4) using (x, y)-semicoarsening for $24 \times 24 \times 24$ problem with smooth heterogeneous permeability field, for 100 days, initial time step 0.1 days, $\Delta t_{max} = 30$ days, 5- pre and 5 post-smoothings, and no more than 10 FAS cycles and 10 newton iterations per FAS cycle.*

F.3 Verifying FAS using standard coarsening

In this section the FAS simulator using 3 or 4 grid levels with standard coarsening is verified by comparing solutions to the solutions from ECLIPSE.

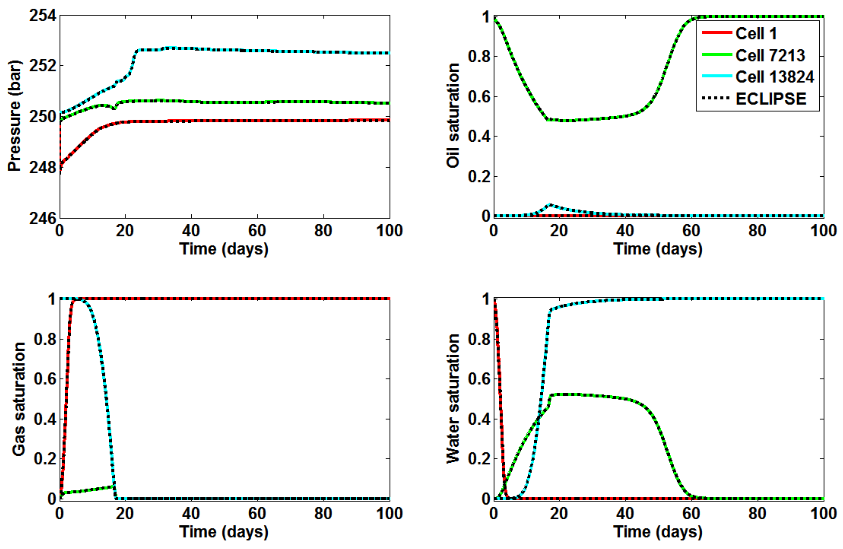


Figure F.5: *Verification of FAS(3) using (x, y)-semicoarsening for $24 \times 24 \times 24$ problem with smooth heterogeneous permeability field, for 100 days, initial time step 0.1 days, $\Delta t_{max} = 30$ days, 5- pre and 5 post-smoothings, and no more than 10 FAS cycles and 10 newton iterations per FAS cycle.*

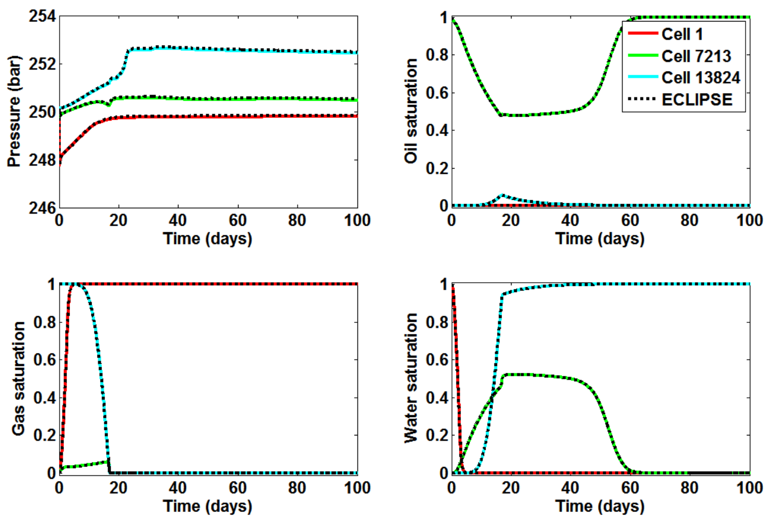


Figure F.6: *Verification of FAS(4) using* (x, y)*-semicoarsening for* $24 \times 24 \times 24$ *problem with smooth heterogeneous permeability field, for 100 days, initial time step 0.1 days,* $\Delta t_{max} = 30$ *days, 5- pre and 5 post-smoothings, and no more than 10 FAS cycles and 10 newton iterations per FAS cycle.*

F.4 Verifying FAS with one newton iteration on coarsest grid

In this section the FAS simulator using 3 or 4 grid levels with one newton iteration on coarsest grid are verified by comparing solutions to the solutions from ECLIPSE.

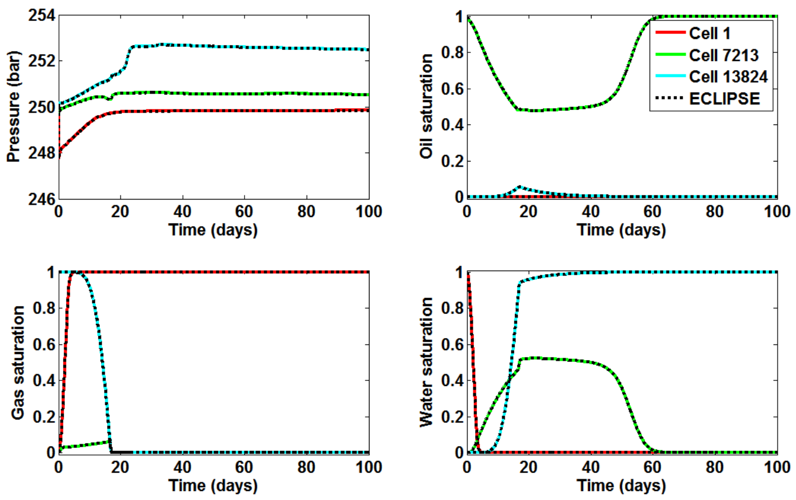


Figure F.7: *Verification of FAS(3), one newton iteration on coarsest grid for* $24 \times 24 \times 24$ *problem with smooth heterogeneous permeability field, for 100 days, initial time step 0.1 days,* $\Delta t_{max} = 30$ *days, 5- pre and 5 post-smoothings, and no more than 10 FAS cycles and 10 newton iterations per FAS cycle.*

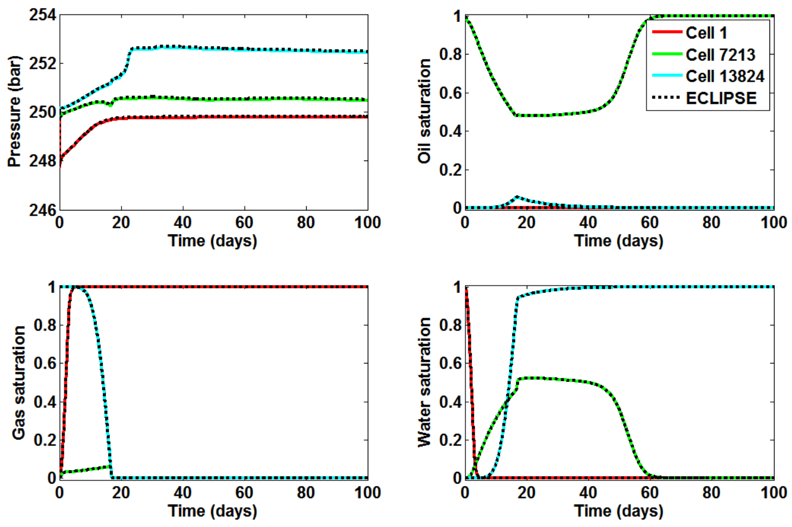


Figure F.8: *Verification of FAS(4), one newton iteration on coarsest grid for* $24 \times 24 \times 24$ *problem with smooth heterogeneous permeability field, for 100 days, initial time step 0.1 days,* $\Delta t_{max} = 30$ *days, 5- pre and 5 post-smoothings, and no more than 10 FAS cycles and 10 newton iterations per FAS cycle.*

Appendix G

Conference contributions

Based on the work presented in this thesis we have been selected to give presentations at the following conferences.

BIT Circus

- **Topic:** Numerical Mathematics and Computational Science
- **Date:** 23-24 August 2012
- **Location:** Technical University of Denmark, Lyngby, Denmark
- **Type:** Oral presentation
- **Abstract:**

  Nonlinear Multigrid for Oil Reservoir Simulation

  To keep up with current and future energy demands, new and better oil recovery techniques are a necessity. All these techniques require advanced oil reservoir simulation. Modern industrial simulation tools are presently able to run simulations with mega-cell scale models, however for many oil and gas reservoirs, larger amounts of seismic, geological and dynamic reservoir data is available. As a result, geological models are upscaled to fit current simulation capabilities. To fully utilize this large amount of data, reservoir simulation tools must accommodate giga-cell scale models. More accurate simulations will have a significant impact on reservoir decision making, resulting in improved oil recovery.

  Present industrial simulation tools are based on conventional simulation techniques also used before parallel hardware became a necessary part of the scientific computing field. It is standard in the conventional techniques to use a global linearization in a Newton-type method to solve the strongly nonlinear system of equations arising from the spatial and temporal discretization of the governing system of PDEs. Consequently, the memory requirement to store the sparse Jacobian requires significant resources. Such very large linear systems result in the linear solver component to constitute more than 70% of the computation time in reservoir simulators. Linear solvers are due to their dependence on a good preconditioner hard to parallelize and therefore result in poor scalability on modern parallel architectures.

We will present numerical methods that aim to reduce the dependence on the linear solver. Specifically, we have developed a nonlinear Multigrid solver based on the Full Approximation Scheme (FAS) for a black oil reservoir model. Using FAS we avoid having to assemble the Jacobian on the finest grid, which results in major memory savings. Furthermore, all components of the nonlinear Multigrid solver are highly local and therefore appropriate for efficient and scalable implementation on modern many-core architectures (e.g. such as many-core Graphics Processing Units (GPUs)). Studies on the robustness, efficiency and scalability of the simulator in case of highly heterogeneous models are presented, as well as estimates of memory savings in comparison with conventional methods.

Society of Petroleum Engineers (SPE)

- **Topic:** Fluid Dynamics and Simulation of Giant Oil and Gas Reservoirs
- **Date:** 3-5 September 2012
- **Location:** Swissotel, Istanbul, Turkey
- **Type:** Poster presentation (see end of this appendix)
- **Abstract:**

  Nonlinear Multigrid for Reservoir Simulation

  In the pursuit of higher resolution simulation models that use all seismic, geological, and dynamic reservoir data-and to make use of modern parallel computing architectures-we consider alternative numerical methods to solve the system of equations governing subsurface porous media flow.

  Current industrial simulation tools are based on conventional simulation techniques, predating the parallel hardware that has become a necessary part of scientific computing. With conventional techniques, it is standard to use a global linearization in a Newton-type method to solve the strongly nonlinear system of equations arising from the spatial and temporal discretization of the governing system of PDEs. Consequently, the memory requirement to store the sparse Jacobian is significant. Very large linear systems result in the linear solver component constituting more than 70% of the computation time in reservoir simulators. Iterative linear solvers depend on effective preconditioners, which can be hard to parallelize to the extent required by many-core simulations.

We present numerical methods that aim to reduce the dependence on the linear solver. Specifically, we have developed a nonlinear multigrid solver based on the full approximation scheme (FAS) for a black-oil reservoir model. By using FAS, we avoid having to assemble the Jacobian on the finest grid, which results in major memory savings. Furthermore, all components of the nonlinear multigrid solver are highly local and therefore appropriate for efficient and scalable implementation on modern, many-core architectures (e.g., many-core graphics processing units). Studies on the robustness, efficiency, and scalability of the simulator in the case of highly heterogeneous models are presented, as well as estimates of memory savings in comparison with conventional methods.

STC, German Section SPE (GSSPE)

- **Topic:** Petroleum engineering
- **Date:** 18-19 October 2012
- **Location:** German Oil Museum, Wietze, Germany
- **Type:** Oral presentation
- **Abstract:**

  Nonlinear Multigrid for Reservoir Simulation

  In the pursuit of higher resolution simulation models that use all seismic, geological, and dynamic reservoir data-and to make use of modern parallel computing architectures-we consider alternative numerical methods to solve the system of equations governing subsurface porous media flow.

  With conventional techniques, it is standard to use a global linearization in a Newton-type method to solve the strongly nonlinear system of equations. Consequently, the memory requirement to store the sparse Jacobian is significant. Very large linear systems result in the linear solver component constituting more than 70% of the computation time in reservoir simulators. Iterative linear solvers depend on effective preconditioners, which can be hard to parallelize to the extent required by many-core simulations.

  We present numerical methods that reduce the dependence on the linear solver. Specifically, we have developed a nonlinear multigrid solver based on the Full Approximation Scheme (FAS) for a black-oil reservoir model. By using FAS, we avoid having to assemble the Jacobian on the finest grid, which results in major memory savings. Furthermore, all components of the solver are local and therefore appropriate for efficient and scalable implementation on modern, many-core architectures (e.g., GPUs). Studies on the robustness, efficiency, and scalability of the simulator in the case of highly heterogeneous models are presented, as well as estimates of memory savings in comparison with conventional methods.

DTU Informatics
Department of Informatics and Mathematical Modeling

Nonlinear Multigrid for Reservoir Simulation

Max la Cour Christensen*, Klaus Langgren Eskildsen*, Allan P. Engsig-Karup*, Mark Wakefield**
*Technical University of Denmark (DTU), **Schlumberger

Motivation

In the pursuit of higher resolution simulation models that use all seismic, geological, and dynamic reservoir data - and to make use of modern parallel computing architectures - we consider alternative numerical methods to solve the system of equations governing subsurface porous media flow.
It is standard in conventional techniques to use a global linearization in a Newton-type method to solve the strongly nonlinear system of equations arising from the spatial and temporal discretization of the governing system of PDEs. Consequently, the memory requirement to store the sparse Jacobian is significant. Such very large linear systems result in the linear solver component to constitute more than 70% of the computation time in reservoir simulators. Iterative linear solvers depend on effective preconditioners, which can be hard to parallelize to the extent required by many-core simulations. In a first step, we investigate feasibility of using the locally linearizing nonlinear multigrid method Full Approximation Scheme (FAS) in serial to establish algorithmic performance.

Contribution

- Reservoir simulator based on the nonlinear multigrid method FAS.
- Comparison with conventional techniques in reservoir simulation, specifically a reservoir simulator based on global linearization in Newton's method. Linear solver is FGMRES with CPR-AMG preconditioning implemented using PETSc/BoomerAMG.

Full Approximation Scheme

Consider the nonlinear system

$$\mathbf{A}(\mathbf{u}) = \mathbf{f}, \qquad (1)$$

with error $\mathbf{e} = \mathbf{u} - \mathbf{v}$, where $\mathbf{v}$ is an approximation to the exact solution $\mathbf{u}$ The residual is

$$\mathbf{r} = \mathbf{f} - \mathbf{A}(\mathbf{v}) \qquad (2)$$

Inserting (1) in (2) gives

$$\mathbf{A}(\mathbf{u}) - \mathbf{A}(\mathbf{v}) = \mathbf{r} \qquad (3)$$

Using the error relation, equation (3) becomes

$$\mathbf{A}(\mathbf{v} + \mathbf{e}) - \mathbf{A}(\mathbf{v}) = \mathbf{r}$$

For coarsest grid with mesh size $H = 2h$, where h is the fine grid mesh size, this is

$$\mathbf{A}_H(\mathbf{v}_H + \mathbf{e}_H) - \mathbf{A}_H(\mathbf{v}_H) = \mathbf{r}_H \qquad (4)$$

The coarse grid residual $\mathbf{r}_H$ is computed by applying the restriction operator I_h^H to the fine grid residual

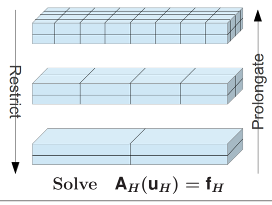
$$\mathbf{r}_H = I_h^H \mathbf{r}_h = I_h^H(\mathbf{f}_h - \mathbf{A}_h(\mathbf{v}_h))$$

Similarly, the coarse grid approximation $\mathbf{v}_H$ is the restriction of the fine grid approximation $\mathbf{v}_h$. Using these definitions, equation (4) is rewritten into

$$\mathbf{A}_H \underbrace{\left(I_h^H \mathbf{v}_h + \mathbf{e}_H\right)}_{\mathbf{u}_H} = \underbrace{\mathbf{A}_H\left(I_h^H \mathbf{v}_h\right) + I_h^H\left(\mathbf{f}_h - \mathbf{A}_h(\mathbf{v}_h)\right)}_{\mathbf{f}_H}$$

Based on this solution, the coarse grid correction term is computed as $\mathbf{e}_H = \mathbf{u}_H - I_h^H \mathbf{v}_h$. This correction term is prolongated to a finer grid, where it is used to correct the solution to the residual equation at that grid, which again can be used to determine a correction term for an even finer grid and etc.

The FAS concept is depicted below



Model equations

Conservation of mass

$$\frac{\partial(\phi m_c)}{\partial t} + \nabla \cdot \mathbf{f}^\alpha = 0,$$

where ϕ is the porosity dependent on pressure, m_c is molar density of component c and $\mathbf{f}^\alpha = b^\alpha \mathbf{v}^\alpha$ is the flux with the phase molar density b^α and the phase velocity $\mathbf{v}^\alpha$ given by Darcy's law.
Volume balance constraint

$$\sum_\alpha S^\alpha = 1,$$

where S^α is the saturation of phase α.

- 3D with gravitational effects.
- Three immiscible phases: oil, gas and water.
- No capillary effects.

Discretization

- Finite Volume method.
- Backward Euler - Fully Implicit.

FAS components

- V-cycles.
- Nonlinear collective Gauss-Seidel z-line smoother using 1 newton iteration.
- Thomas algorithm for block tridiagonal systems in smoother.
- (x, y)-semicoarsening, meaning only x- and y-directions are coarsened.

Numerical results

For fixed pattern permeability fields as depicted in Figure 1

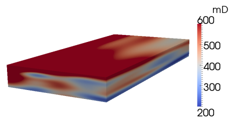


Figure 1: *Illustration of permeability field*

a comparison between a reservoir simulator based on conventional techniques (standard newton(SN)) with global linearization in Newton's method and the simulator based on FAS is displayed in Figure 2. Tests are "gravity inversion", where water and gas components switch place.

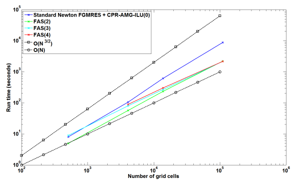


Figure 2: *Run time as a function of number of grid cells for standard newton and FAS with $2, 3$ and 4 grid levels. Fixed reservoir size: $480m \times 240m \times 48m$.*

FAS is faster and appears to have linear scaling for the larger problem sizes. The corresponding number of time steps is listed in Table 1.

	Problem size			
Method	$8 \times 8 \times 8$	$16 \times 16 \times 16$	$24 \times 24 \times 24$	$48 \times 48 \times 48$
SN	68	101	152	201
FAS(2)	40	60	91	171
FAS(3)	65	83	82	161
FAS(4)	N/A	93	94	127

Table 1: Number of time steps for the simulations displayed in Figure 2. An adaptive time stepping strategy is employed that seeks to minimize the overall computational effort.

Residuals for outer iteration

Figure 3 shows residual reduction for 50 time steps, where each line represents a time step.

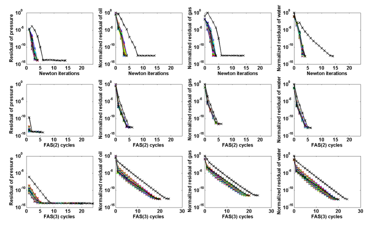


Figure 3: *Residual reduction for standard newton and FAS with 2 and 3 grid levels. Problem size: $24 \times 24 \times 24$.*

FAS has quick initial residual reduction suitable for engineering accuracy purposes.

Computational distribution

As indicated by Figure 4, the majority of the computational work can be kept in the parallelizable smoother by increasing the number of grid levels in FAS.

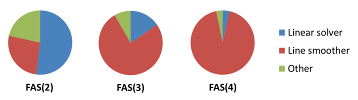


Figure 4: *Distribution of computational work for FAS with $2, 3$ and 4 grid levels. Problem size: $24 \times 24 \times 24$.*

Memory comparisons

FAS is based on local linearization, meaning we avoid having to assemble the Jacobian on the finest grid, which results in significant memory savings as outlined in Table 2.

Method:	**kB per grid cell:**
Standard newton	2.3
FAS(2)	0.7
FAS(3)	0.3
FAS(4)	0.2

Table 2: Memory comparisons for a $128 \times 128 \times 128$ grid cell problem.

Heterogeneity stress test

As demonstrated by Figure 5, FAS appears to handle very heterogeneous permeability fields better than standard newton.

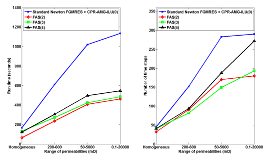


Figure 5: *Run time and number of time steps for problems with different ranges of permeability. Problem size: $24 \times 24 \times 24$.*

Conclusion and perspectives

- Promising aspects for modern many-core implementation, e.g. GPUs or Intel MICs.
- Fit larger problems on same hardware and extend to distributed large-scale simulations.
- Algorithmic performance established for model equations.
- Next step: More complicated model including wells, parallel implementation and feasibility study of polynomial smoothing.

Contacts: Max: s072245@student.dtu.dk - Klaus: s072272@student.dtu.dk - Allan: apek@imm.dtu.dk

Nomenclature

Physical quantities

Symbol	Description	Units
m_c	Component molar density	kmol/m^3
p	Pressure	bar
b^α	Phase molar density	kmol/m^3
$\mathbf{v}^\alpha$	Darcy phase velocity	m/day
K	Absolute permeability	mD
k_r^α	Relative permeability	-
μ^α	Viscosity of phase α	cP
S^α	Saturation of phase α	-
g	Gravitational acceleration	$\text{m}^2\text{kg}/\text{bar}$
ρ^α	Mass density of phase	kg/m^3
T	Transmissibility	$\text{cPm}^3/\text{day}/\text{bar}$
f^α	Flux	$\text{kmol}/\text{m}^2/\text{day}$
V	Volume (Bulk)	m^3
V_p	Pore volume	m^3
ϕ	Porosity	-

Abbreviations

AMG	Algebraic Multigrid
BiCGSTAB	Stabilized Bi-Conjugate Gradient
CGN	Conjugate Gradient applied to the Normal equations
COO	Coordinate list

CPR	Constrained Pressure Residual
CPU	Central Processing Unit
CSR	Compressed Sparse Row
FAS	Full Approximation Scheme
FGMRES	Flexible Generalized Minimal Residual
FMG	Full Multigrid
FVM	Finite Volume Method
GMRES	Generalized Minimal Residual
GPU	Graphics Processing Unit
GSN	Gauss-Seidel-Newton
ILU	Incomplete LU
LSPS	Line Solve Power Series
MG	Multigrid
MIC	Many Integrated Core
MIMD	Multiple Instruction Multiple Data
ORTHOMIN	Orthogonal Minimization
PDE	Partial Differential Equation
SIMD	Single Instruction Multiple Data
SOR	Successive Over-Relaxation

Bibliography

[1] Adams, M. F., *A Distributed Memory Unstructered Gauss-Seidel Algorithm for Multigrid Smoothers*, Sandia National Laboratories, November 19, 2001.

[2] Adams, M. F., Brezina, M., Hu, J., Tuminaro, R., *Parallel multigrid smoothing: polynomial versus Gauss-Seidel*, Journal of Computational Physics 188 (2003) 593-610.

[3] Amestoy, P. R., Duff, I. S., Koster, J., L'Excellent, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal of Matrix Analysis and Applications, Vol 23, No 1, pp 15-41 (2001).

[4] Amestoy, P. R., Guermouche, A., L'Excellent, J.-Y., Pralet, S., *Hybrid scheduling for the parallel solution of linear systems*, Parallel Computing Vol 32 (2), pp 136-156 (2006).

[5] Arnoldi, W. E., *The principle of minimized iteration in the solution of the matrix eigenvalue problem*, Quart. Apple. Math. 9 (1951) pp. 17-29.

[6] Aziz, K., Durlofsky, L., Tchelepi, H., *Notes on Petroleum Reservoir Simulation*, Department of Petroleum Engineering, School of Earth Sciences, Stanford University, California, USA, 2005.

[7] Birol, Dr. Fatih, *World Energy Outlook 2011*, International Energy Agency, Presentation at conference in Copenhagen 15th November 2011.

[8] Brandt, A., *Multi-Level Adaptive Solutions to Boundary-Value Problems*, Mathematics of Computation, Volume 31, Number 138, Pages 333-390, 1977.

[9] Brandt, A., *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics*, Gesellschaft für Mathematik und Datenverarbeitung mbH Bonn, 1984.

[10] Cai, X., Keyes, D. E., Young, D. P., *A Nonlinear Additive Schwarz Preconditioned Inexact Newton Method for Shocked Duct Flows*, Thirteenth International Conference on Domain Decomposition Methods, 2001.

[11] Cao, H., Tchelepi, H.A., Wallis, J.R., Yardumian, H., *Parallel Scalable Unstructered CPR-Type Linear Solver for Reservoir Simulation*, SPE paper 96809, SPE Annual Technical Conference and Exhibition, Dallas, Texas, USA, 9-12 Oct. 2005.

[12] Chen, Z., Huan, G. & Ma, Y., *Computational Methods for Multiphase Flows in Porous Media*, SIAM, Philadelphia, 2006.

[13] Christie M. A., Blunt, M. J., *Tenth SPE comparative solution project: A comparison of upscaling techniques*, SPE 72469, SPE Reservoir Evaluation and Engineering, August 2001.

[14] Danish Energy Agency, *Denmark's Oil and Gas Production 2010*, www.ens.dk.

[15] Darcy, H., *Les Fontaines Publiques de la Ville de Dijon*, Victor Dalmond, Paris, 1856.

[16] Dogru, Ali H. et al., *From Mega-Cell to Giga-Cell Reservoir Simulation*, SPE 116675, SPE Annual Technical Conference, Denver, Colorado, 21-24. September 2008.

[17] Dogru, Ali H. et al., *A Next-Generation Parallel Reservoir Simulator for Giant Reservoirs*, SPE 119272, SPE Symposium, The Woodlands, Texas, 2-4. February 2009.

[18] Dogru, Ali H. et al., *New Frontiers in Large Scale Reservoir Simulation*, SPE 142297, SPE Symposium, The Woodlands, Texas, 21-23. February 2011.

[19] Durlofsky, L. J., Aziz, K., *Advanced Techniques for Reservoir Simulation and Modeling of Nonconventional Wells*, Department of Petroleum Engineering, Stanford University, 2004.

[20] Eldén, L., Wittmeyer-Koch, L. & Nielsen, H. B., *Introduction to Numerical Computation - analysis and MATLAB®illustrations*, Studentlitteratur, Lund, 2004.

[21] Engsig-Karup, A. P., Madsen, M. G., Glimberg, S. L., *A massively parallel GPU-accelerated model for analysis of fully nonlinear free surface waves*, International Journal For Numerical Methods in Fluids, Wiley Online Library, DOI: 10.1002/fld.2675, 2011.

[22] Fung, L.S.K., Dogru, A.H. *Parallel Unstructered Solver Methods for Complex Giant Reservoir Simulation*, Saudi Aramco, SPE paper 106237, SPE Reservoir Simulation Symposium, Houston, Texas, USA, 26-28 Feb. 2007.

[23] Farber, R., *The GPU Performance Revolution, Intel's entry into the massively parallel chip market adds fuel to survival-of-the-fittest product evolution*, April 2012, ScientificComputing.com.

[24] Golub, G. H., Van Loan, C. F., *Matrix Computations*, Third Edition, The Johns Hopkins University Press, Baltimore, Maryland, 1996.

[25] Gould, N. I. M, Hu, Y., Scott, J. A., *A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations*, CCLRC Rutherford Appleton Laboratory, 2005.

[26] Grabowski, J. W., Vinsome, P. K., Lin, R. C., Behie, A., Rubin, B., *A Fully Implicit General Purpose Finite-Difference Thermal Model for In Situ Combustion and Steam*, SPE 8396, 1979.

[27] Gupta, A., Muliadi, Y., *An Experimental Comparison of Some Direct Sparse Solver Packages*, 2001 IEEE.

[28] Henson, V. E., *Multigrid methods for nonlinear problems: An overview*, Center for applied scientific computing, Lawrence Livermore National Laboratory.

[29] Heuveline, V., Lukarski, D., Weiss, J-P., *Enhanced Parallel ILU(p)-based Preconditioners for Multi-core CPUs and GPUs - The Power(q)-pattern Method*, Preprint Series of the Engineering Mathematics and Computing Lab (EMCL), ISSN 2191-0693, No. 2011-08.

[30] Jiang, Y., *Techniques for Modeling Complex Reservoirs and Advanced Wells*, PhD Thesis, Stanford University, 2007.

[31] Keckler, S. W., et al. (NVIDIA), *GPUs and the Future of Parallel Computing*, IEEE Computer Society, September/October 2011.

[32] Kraus, J., Förster, M., Brandes, T., *Using LAMA for efficient AMG on hybrid clusters*, Fraunhofer Institute, Springer-Verlag, 2012.

[33] LeVeque, R. J., *Numerical Methods for Conservation Laws*, second edition, Birkhäuser, Basel, 1999.

[34] LeVeque, R. J., *Finite-Volume Methods for Hyperbolic Problems*, Cambridge texts in applied mathematics, Cambridge University Press, 2004.

[35] LeVeque, R. J., *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*, SIAM, Philadelphia, 2007.

[36] Molenaar, J., *Multigrid methods for fully implicit oil reservoir simulation*, TU Delft, Report 95-40, 1995.

[37] Raju, M. P., Khaitan, S., *High Performance Computing Using Out-of-Core Sparse Direct Solvers*, World Academy of Science, Engineering and Technology 57, 2009.

[38] Saad, Y., *Iterative methods for sparse linear systems*, Second edition, 2003.

[39] Sturler, E. de, van der Vorst, H. A., *Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers*, Elsevier, Applied Numerical Mathematics, 18(1995)441-459.

[40] Sundar, H., Biros, G., Burstedde, C., Rudi, J., Ghattas, O., Stadler, G., *Parallel Geometric-Algebraic Multigrid on Unstructured Forests of Octrees*, University of Texas at Austin, Supercomputing 12, Salt Lake City, Utah, USA, November 2012.

[41] Thomas, J.W., *Numerical Partial Differential Equations, Finite Difference Methods*, Texts in Applied Mathematics 22, Springer, New York, 1995.

[42] Trangenstein, J. A. & Bell, J. B, *Mathematical Structure of the Black-Oil Model for Petroleum Reservoir Simulation*, SIAM vol. 49, 1989.

[43] Trottenberg, U., Oosterlee, C., Schüller, A., Guest contributions by: Stüben, K., Oswald, P., Brandt, A., *Multigrid*, Academic Press, San Diego, CA, 2001.

[44] U.S Energy Information Administration, *International Energy Outlook 2011*, September 2011.

[45] Wallis, J. R., *Incomplete Gaussian Elimination as a Preconditioning for Generalized Conjugate Gradient Acceleration*, SPE paper 12265, SPE Reservoir Simulation Symposium, San Francisco, 1983.

[46] Wallis, J.R., Kendall, R.P., Little, T.E., *Constrained Residual Acceleration of Conjugate Residual Methods*, SPE paper 13563, 8th Symposium on Reservoir Simulation, Dallas, Feb. 10-13, 1985.

[47] Younis, R. M., Tchelepi, H. A., Aziz, K., *Adaptively-Localized-Continuation-Newton: Reservoir Simulation Nonlinear Solvers That Converge All the Time*, SPE 119147, Stanford University, SPE Reservoir Simulation Symposium, Woodlands, Texas, U.S.A., 2-4 February 2009.

[48] Zhou, H., Lee, S.H., Tchelepi, H.A., *Multiscale Finite-Volume Formulation for the Saturation Equations*, SPE paper 119183, SPE Reservoir Simulation Symposium, The Woodlands, Texas, USA, 2-4. Feb. 2009.

[49] Zikanov, O., *Essential Computational Fluid Dynamics*, John Wiley & Sons, New Jersey, 2010.