NTNU
Norwegian University of
Science and Technology

Nonlinear Two-Point Flux Approximation Schemes for Reservoir Simulation

Pia-Kristina Heigrestad

Master of Science in Physics and Mathematics
Submission date: July 2018
Supervisor: Knut Andreas Lie, IMF
Co-supervisor: Olav Møyner, SINTEF

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

The main objective of this thesis has been to investigate some methods for simulating CO_2 storage and hydrocarbon recovery on complex polyhedral grids, with main focus on the nonlinear two-point flux approximation method. The industry standard has for many years been the (linear) two-point flux approximation method, but as this method only calculates the flux across an internal face by approximating the transmissibilities as averages between two cells, it does not account for transverse flux across the face making it inconsistent for problems where transverse flux occurs. The nonlinear two-point flux approximation method does take transverse flux into consideration, and so it is of interest to implement it as a part of SINTEF's Matlab Reservoir Simulation Toolbox (MRST). This thesis will begin by going through the physical aspects of the flow equation and it's discretization. We will follow up by deriving the linear and nonlinear two-point flux approximation methods, as well as a multipoint flux approximation method. Further, use and syntax of MRST is presented, together with explanation of the written code. Through a few examples and comparisons, we investigate the robustness and accuracy of the nonlinear two-point flux approximation method compared to established solvers in MRST. It is shown that the nonlinear two-point flux approximation method is versatile and an equally good candidate compared to other established methods within reservoir simulation. However, it is sensitive to full permeability tensors on complex grids, and thus, as is, ineffective for the purpose it is intended.

Sammendrag

Hovedmålet med denne avhandlingen har vært å utforske metoder innen simulering av CO2 lagring og hydrokarbonutvinning på avanserte polyhedrale rutenett, med fokus på en ikke-lineær topunkts flukstilnærmings metode. Industristandarden har i mange år vært en (lineær) topunkt flukstilnærmingsmetode, men ettersom denne metoden kun bruker to punkter i to naboceller til å beregne fluksen mellom cellene tar den ikke høyde for tverrgående fluks, noe som gjør den ukonsistent. Den ikke-linære metoden tar hensyn til tverrgående fluks ved å benytte seg av trykk i flere naboceller til å beregne transmissibiliteten, før også den bruker to punkter for å beregne fluksen mellom to celler. Med bakgrunn i dette er det av interesse å implementere den ikke-lineære topunktsmetoden som en del av SINTEF's Matlab Reservoir Simulation Toolbox (MRST). Avhandlingen begynner med å presentere de fysiske aspektene ved flytligningen, og dens diskretisering. Vi fortsetter med utledningene av den lineære og den ikke-lineære topunkts metoden, i tillegg til en flerpunkts flukstilnærmingmetode. Videre er bruk og syntaks i MRST presentert, sammen med forklaring på koden som er skrevet. Avsluttningsvis har vi gjennom eksempler og sammenligninger med etablerte løsere i MRST utforsket bruksområdene og nøyaktigheten til den ikkelineære topunkts metoden. Det er vist at den ikke-lineære topunkts flukstilnærmingsmetoden er alsidig og en likeverdig god kandidat sammenlignet med andre metoder innen reservoar simulering. Metoden er på den annen side sensitiv for fullstendige permeabilitets tensorer på komplekse grid, noe som gjør den, som den er, ineffektiv innen de feltene den er tiltenkt.

Preface

This report is written as my final master's thesis in the field of Industrial Mathematics at the Norwegian University of Science and Technology, NTNU Trondheim. The work was carried out from March 1st to July 26th of 2018. The report is written in collaboration with the Department of Mathematics and Cybernetics at SINTEF Digital. I would like to thank my supervisors Knut-Andreas Lie and Olav Møyner for suggesting the problem, helping me throughout the period, and answering all my questions. I would also like to thank my parents for letting me stay with them closer to Oslo while the work was done, and the support they, and the rest of my family, have given me throughout my studies. Lastly, I want to thank my boyfriend. Your support has been invaluable.

Spikkestad, July 26, 2018

Pia-Kristina Heigrestad

Contents

Chapter 1

Introduction

Reservoir simulation is the process of modelling and predict the flow of various fluids through porous media. The fluids are typically oil, gas and water, and the porous media are sedimentary rocks located sub-sea that have been formed throughout centuries. These sedimentary rocks constitute a reservoir, and does usually have a layer of impermeable rock over it that prevents the hydrocarbon from migrating from the reservoir to the surface. The size of the reservoirs vary greatly, from small reservoirs of a few cubic kilometers, to large ones on thousands of cubic kilometers. The ability to accurately predict and simulate the flow in porous media has become of great importance when developing and planning a new field, as well as plan and improve ongoing recovery from existing fields.

Oil companies became increasingly aware of the importance of predicting reliable reservoir behaviour during the early twentieth century (Care, 2010). Mathematical models for pressure gradients and fluid flow were developed, but the increasing complexity made the engineers turn to analogue models called reservoir analyzers. In the 1930's, researchers working for large petroleum companies started developing electrical models, with reference to the analogue analyzers, to model hydrodynamics in oil reservoirs. Since then, great development has been made within the field of reservoir simulation, but there is still room for improvements. The industry standard relies today on methods based on finite difference, and the two-point flux approximation method is frequently applied to solve sub-sea flow problems (Lie, 2016). The method approximates the flux through a cell wall by using the pressure in the two neighboring cells. However, as the method is linear and only accounts for two pressure points to approximate the flux, it will not be consistent on grids that are not so-called K-orthogonal, i.e., grids where the vector between two solution nodes is not parallel to the normal vector of the cell wall. It may thus fail to give reasonable results on geologically difficult grids (Wu and Parashkevov, 2009). Methods that are consistent, such as the multipoint flux approximation method (Aavatsmark, 2002), the mixed finite element method (Brezzi and Fortin, 1991), or the mimetic finite-difference methods (Brezzi et al., 2005), are in general not monotone for quadrilateral grids, and thus may cause unnatural oscillations (Keilegavlen et al., 2009; Nordbotten et al., 2007). One could get around the problem of not having a method that is both monotone and consistent by applying different methods on the flow-problem and combine them to find the most likely solutions, as reservoir simulations will have many uncertainties anyway. However, a solution that has been developed is to make the solver nonlinear, and thus monotone and consistent by construction. One such method is the nonlinear two-point flux approximation method, where the first to propose the method are Potier (2009) and Nikitin et al. (2014). The method has shown great potential, but has little research done on its accuracy.

1.1 Objectives

The main objective of this thesis has been to improve a preliminary implementation of the nonlinear two-point flux approximation method (NTPFA) done by Olav Møyner in SINTEF's Matlab Reservoir Simulation Toolbox (MRST) (Berge et al., 2017). The existing code was close to finished, but used a search method to decompose the directional derivatives of the pressure variables. The new implemented method uses optimization. A comparison between the results of the two decompositions has been done and is presented. Further, the NTPFA method is compared to other established methods already existing in MRST to investigate its applications.

1.2 Approach

Understanding both the physical aspects of subsurface flow and established methods for solving the incompressible flow equations is inevitable to implement the NTPFA method. In the project done during the first half of my fifth year, I began studying these topics and made a first, preliminary implementation of the NTPFA method (Heigrestad, 2018). This has been used as background, and further investigations has been made built on results and insights from this project. A few methods already incorporated to MRST has been used as learning tool and validation of implementations done.

1.3 Outline

In this thesis, I will introduce you briefly to the physics of flow in porous media, present the key governing equations for the basic models, and introduce you to the problem of formulating consistent and monotone schemes on general polyhedral grids. A more thorough investigation is made for the nonlinear two-point flux approximation method, on which this thesis has main focus. Further, comparison and validational tests are done and presented. In Chapter 2, a brief introduction to reservoir physics is given, as well as an introduction to key mathematical models (Darcy's law). Further, the elliptic model, discretization, and derivations are presented. In Chapter 3, MRST notations and a thorough explanation of the implementations done are given. We discuss through examples and comparisons to a few established methods the NTPFA method's soundness in Chapter 4. The thesis is completed in Chapter 5 with a conclusion and recommendations for further work.

Chapter 2

Theory

2.1 Reservoir Properties

When modeling subsurface flow it is necessary to have some background knowledge on the flow through permeable rocks. All natural reservoirs consist of sedimentary rocks with sufficient porosity and permeability to store and transmit fluids (Nordbotten and Celia, 2012; Lie, 2016). A rock's porosity is its ability to store fluids, and is determined by the volume fraction of pores. The porosity will thus have a value between $[0, 1)$, and the value is mainly determined by the pore and grain-size distribution. For rigid mediums, the porosity will be static, whereas for non-rigid mediums it is common to model the porosity as a pressure-dependent variable. Denoting the porosity by ϕ, we can define the rock compressibility, c_r, depending on the porosity and overall reservoir pressure p by

$$c_r = \frac{1}{\phi}\frac{d\phi}{dp} = \frac{d\ln\phi}{dp}. \tag{2.1}$$

The porosity is usually assumed to be a piece-wise continuous spatial function. Assuming constant compressibility, integration of Equation (2.1) followed by linearization yields

$$\phi = \phi_0\big(1 + c_r(p - p_0)\big).$$

The permeability is the ability to transmit fluids, and is given by the interconnection of the pores. We denote permeability by $\mathcal{K}$. The permeability will generally be a full tensor

$$\mathcal{K} = \begin{bmatrix} \mathcal{K}_{xx} & \mathcal{K}_{xy} & \mathcal{K}_{xz} \\ \mathcal{K}_{yx} & \mathcal{K}_{yy} & \mathcal{K}_{yz} \\ \mathcal{K}_{zx} & \mathcal{K}_{zy} & \mathcal{K}_{zz} \end{bmatrix}, \tag{2.2}$$

where the diagonal elements represent how pressure drop in one axial direction effects the flow rate in the same direction. The connection between the flow in the axial direction and pressure drop in perpendicular directions is described by the off-diagonal elements in (2.2). It is common to specify permeability in millidarcys (mD), where 1D$\approx 0.987 \cdot 10^{-12}$m^2, whereas porosity is dimensionless. When modeling a physical system, we must require that the permeability tensor in Equation (2.2) is symmetric and positive definite. The requirement of positive definiteness comes from the fact that the flow component parallel to the pressure drop should be in the same direction as the pressure drop.

2.2 Darcy's Law

Darcy's law describes the connection between pressure and flow rate, and is, together with the equation of conservation of mass, one of the main equations describing flow and continuity of fluid phases. Henry Darcy wanted to understand the physics of flow through the sand filters used to filtrate the water supply in the city of Dijon in the 19th century France. He performed a series of experiments on a sand pack and established that the flow rate of that pack was proportional to the cross-section area and the difference in water height, and in addition it was inversely proportional to the flow length of the tank. His results were published as an appendix in Darcy (1856), where the law, which today is known as Darcy's law, was stated

$$Q = AK\frac{(h_1 + z_1) - (h_2 - z_2)}{L}. \tag{2.3}$$

In Equation (2.3), Q denotes the volume flow rate. Furter, A is the area cross section, L the length of the flow path, K the hydraulic conductivity, z the elevation on top and bottom, and h the pressure head in the position of z, i.e. the pressure divided by the specific weight (Brown, 2002). By replacing Q/A with $\mathbf{u}$ we get the specific flow rate, known as Darcy flux, through the sand pack. Darcy flux has dimension [m/s] and measures the volume of fluid per total area per time. The hydraulic conductivity is given by $K = \rho g \mathcal{K}/\mu$, where g is the gravitational acceleration constant, ρ is the fluid density and μ is the fluid viscosity. Introducing the fluid potential Φ, Equation (2.3) can now be written

$$\mathbf{u} = -\frac{\mathcal{K}}{\mu}\nabla\Phi. \tag{2.4}$$

We see that the permeability is a proportionality factor between the flow rate and the applied pressure.

2.2.1 Single-phase Flow

Exploring Darcy's equation we find the extension for a single-phase flow

$$\mathbf{u} = -\frac{\mathcal{K}}{\mu}(\nabla p - \rho g \nabla z), \tag{2.5}$$

where $(\nabla p - \rho g \nabla z)$ corresponds to $\nabla\Phi$ in Equation (2.4). In Equation (2.5), ρ and g are defined as before, and z is the vertical coordinate. We have that the gradient of the vertical coordinate can be denoted by its unit vector, e_z, and further the gravitational acceleration can be expressed $\mathbf{g} = g e_z$, so Equation (2.5) can be written

$$\mathbf{u} = -\frac{\mathcal{K}}{\mu}(\nabla p - \rho \mathbf{g}).$$

For more general flow equations for single-phase flow we consider the law of mass conservation. All mass produced inside a specific volume or area must equal the total flux over the boundaries. This can be expressed

$$\frac{\partial}{\partial t}\int_{\Omega} \phi\rho \, d\vec{x} + \int_{\partial\Omega} \rho\vec{u}\cdot\vec{n}\, ds = \int_{\Omega} \rho q \, d\vec{x}, \tag{2.6}$$

where $\vec{n}$ is the normal at the boundary $\partial\Omega$ of the domain Ω, and q denotes the sinks and sources. Equation (2.6) must be satisfied for any (infinitesimal) region, and thus must satisfy the following continuity equation

$$\frac{\partial(\phi\rho)}{\partial t} + \nabla\cdot(\rho\vec{u}) = \rho q.$$

By combining this equation with Darcy's law as well as fluid and rock compressibilities, we obtain a parabolic equation for the fluid pressure

$$c_t \phi \rho \frac{\partial p}{\partial t} - \nabla \big[\frac{\rho \mathcal{K}}{\mu} (\nabla p - \rho \mathbf{g}) \big] = \rho q,$$

where c_t is the total compressibility in the system. If incompressible flow is considered, $c_t = 0$, and we get a linear elliptic equation

$$-\nabla \cdot \left[\frac{\mathcal{K}}{\mu} \nabla (p - g\rho z) \right] = q. \tag{2.7}$$

We introduce $\Phi = p - g\rho z$ and recognize Equation (2.7) as the generalized Poisson equation.

We henceforth neglect gravity and set viscosity to one, and consider an elliptic Poisson equation on the form

$$-\nabla \cdot \mathcal{K} \nabla p = q, \tag{2.8}$$

where we assume that $\mathcal{K}$ is symmetric and positive definite. In Equation (2.8), Φ from Equation (2.7) coincides with the pressure. To be able to find a well-posed solution, boundary conditions to Equation (2.8) are required. We thus impose the boundary conditions

$$\begin{aligned} p &= p_D, \ \text{on}\, \Gamma_D \\ -(\mathcal{K}\nabla p) \cdot \vec{n} &= \nu_N, \ \text{on}\, \Gamma_N. \end{aligned}$$

Here, the boundary for each cell in the grid is divided into Dirichlet boundary, Γ_D, and Neumann boundary, Γ_N. In reservoir simulations, it is common to assume no flow across the outer boundaries to the reservoir, and so the Neumann condition, ν_N, will be 0 on the boundary of the reservoir.

2.3 Discretization

To discretize Equation (2.8) by a finite-volume method, we integrate both sides over a control volume Ω_i to get

$$\int_{\partial \Omega_i} -\mathcal{K} \nabla p \cdot \vec{n} \, dS = \int_{\Omega_i} q \, d\vec{x}, \tag{2.9}$$

where the mass is conserved for each cell. We denote $\vec{v} = -\mathcal{K}\nabla p$ and use Darcy's law to define the flux across the face between cell i and j as

$$v_{i,j} = \int_{\Gamma_{i,j}} \vec{v} \cdot \vec{n}\, dS. \tag{2.10}$$

We then have the following

$$\sum_j v_{i,j} = \int_{\Omega_i} q d\vec{x}. \tag{2.11}$$

In Equation (2.10), $\Gamma_{i,j}$ is considered a half-face that bounds cell i against its neighboring cell j. The set of all faces is defined

$$\mathcal{F} := \{\Gamma_{i,j} | \Gamma_{i,j} = \Omega_i \cap \Omega_j, j \in \mathcal{N}(i)\} \cup \{\Gamma_{i,\Gamma_D \cup \Gamma_N} | \Gamma_{i,\Gamma_D \cup \Gamma_N} = \Omega_i \cap (\Gamma_D \cup \Gamma_N)\}, \tag{2.12}$$

where $\mathcal{N}(i)$ is the set of neighboring cell indices, such that $\partial\Omega_i = \cup_{j \in \mathcal{N}(i)} \Gamma_{i,j}$. The half-face from j to i will be identical, but with opposite normal vector. We now approximate the integral in (2.10) by the midpoint rule and obtain

$$v_{i,j} \approx A_{ij}\vec{v}(\vec{x}_{ij}) \cdot \vec{n}_{i,j} = -A_{ij}(\mathcal{K}\nabla p)(\vec{x}_{ij}) \cdot \vec{n}_{i,j}, \tag{2.13}$$

where A_{ij} is the area of the shared face between cell i and j, and $\vec{x}_{ij}$ denotes the centroid on $\Gamma_{i,j}$. Note that A_{ij} and $\vec{x}_{ij}$ is without comma between i and j as $A_{i,j} = A_{j,i}$ and $\vec{x}_{i,j} = \vec{x}_{j,i}$. As we only know the averaged value of the pressure inside each cell, and not the pressure at the face centroids, which we will denote $\pi_{i,j}$, we need to make some assumptions to be able to evaluate the pressure gradient. The right hand side of Equation (2.9) is found using quadrature rules.

2.4 Two-Point Flux Approximation

For the TPFA method, only the cell pressure in one neighboring cell is assumed to contribute to the flux across the interface between the two cells. Since, as mentioned, we do not have the pressure $\pi_{i,j}$ at the face centroid, we need to approximate the pressure gradient in Equation (2.13). As we know the average pressure p_i inside the cell, we assume that the pressure is linear inside each cell so that the pressure at the cell center is identical to p_i. The flux $v_{i,j}$ can thus be approximated as

$$v_{i,j} \approx A_{ij}\mathcal{K}_i \frac{(p_i - \pi_{i,j})\vec{c}_{i,j}}{|\vec{c}_{i,j}|^2} \cdot \vec{n}_{i,j} = T_{i,j}(p_i - \pi_{i,j}),$$

where $T_{i,j}$ is the one-sided transmissibility and is associated with the half face between cell i and j, and $\vec{c}_{i,j}$ is the vector between the solution nodes p_i and $\pi_{i,j}$ as seen in Figure 2.1.

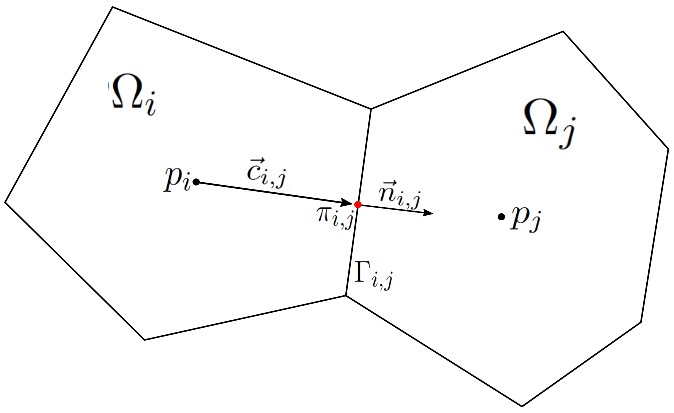


Figure 2.1: Two cells, Ω_i and Ω_j, for the two-point finite-volume discretization. The pressure p_i and p_j are at the cell centroids, and the pressure $\pi_{i,j}$ is at the centroid of the face $\Gamma_{i,j}$.

By taking continuity of fluxes across all faces and continuity of face pressures into account, so that $v_{i,j} = -v_{j,i} = v_{ij}$ and $\pi_{i,j} = \pi_{j,i} = \pi_{ij}$, we end up with the following scheme for the TPFA method by eliminating the interface pressure

$$v_{ij} = \left[T_{i,j}^{-1} + T_{j,i}^{-1}\right]^{-1}(p_i - p_j) = T_{ij}(p_i - p_j). \tag{2.14}$$

T_{ij} describes the transmissibility between the two cells. We now insert the flux in Equation (2.14) into Equation (2.11) to get the following system of equations

$$\sum_j T_{ij}(p_i - p_j) = q_i, \qquad \forall\, \Omega_i \subset \Omega,$$

where we define the matrix $\mathbf{A} = \{a_{ik}\}$ with

$$a_{ik} = \begin{cases} \sum_j T_{ij}, & \text{if } k = i. \\ -T_{ik}, & \text{if } k \neq i. \end{cases}$$

This system is symmetric due to the zero Neumann condition, and a solution is defined up to an arbitrary constant for the continuous problem. By specifying the pressure in a single point, the system is both positive definite and symmetric.

2.5 Consistency

To guarantee convergence of a numerical method, it is necessary for the method to be consistent. As mentioned, the TPFA method is consistent for K-orthogonal grids only. To see that it is not consistent for grids that are not K-orthogonal, we look at an example with a full permeability tensor in 2D on the form

$$\mathcal{K} = \begin{bmatrix} \mathcal{K}_{xx} & \mathcal{K}_{xy} \\ \mathcal{K}_{xy} & \mathcal{K}_{yy} \end{bmatrix},$$

which is symmetric and assumed positive definite. As we know that the flux across a face $\Gamma_{i,j}$ is approximated by the pressure in the two neighboring cells, Ω_i and Ω_j, we consider a simple Cartesian grid as the one seen in Figure 2.2, and see that the normal vector from cell i to cell j will be $\vec{n}_{i,j} = [1, 0]$, and is equal to the vector $\vec{c}_{i,j}$. Thus the flux in Equation (2.10) will be

$$v_{i,j} = \int_{\Gamma_{i,j}} (-\mathcal{K}\nabla p) \cdot \vec{n}\, dS = -\int_{\Gamma_{i,j}} (\mathcal{K}_{xx}\frac{\partial p}{\partial x} + \mathcal{K}_{xy}\frac{\partial p}{\partial y})\, dS.$$

However, since we only consider the two pressure-points varying in the x-direction, we do not have an estimation for $\frac{\partial p}{\partial y}$, and the TPFA method will thus not be

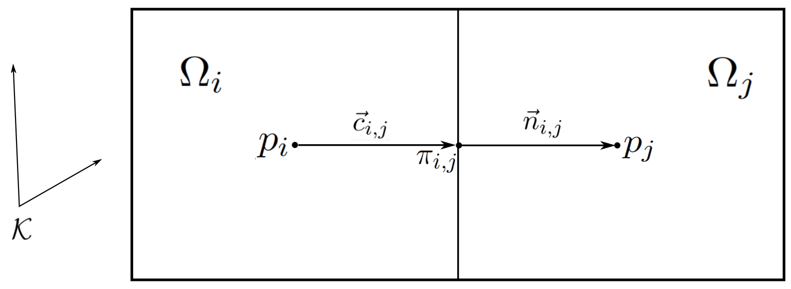


Figure 2.2: A simple cartesian grid where $\vec{n}_{i,j} = \vec{c}_{i,j} = [1, 0]$, and where the axes of the full permeability tensor does not align with the coordinate system.

consistent unless $\mathcal{K}_{xy} \equiv 0$ in all cells in the grid. In this case, there is no flux in the transverse direction, and $(\mathcal{K}\vec{n}_{i,j}) \| \vec{c}_{i,j}$, i.e., the grid is K-orthogonal. The NTPFA method solves this problem by utilizing several pressure points to estimate the flux across a face, and is thus potentially consistent for all grids.

2.6 Nonlinear Two-Point Flux Approximation

The NTPFA scheme lets the transmissibilites depend on one or more pressure values. We then get a two point expression for the flux on the form

$$v_{i,j} = T_i(\vec{p})p_i - T_j(\vec{p})p_j.$$

The main idea for the NTPFA method is to approximate the directional derivatives of p in the conormal direction, $\vec{d}_\Gamma := \mathcal{K}\vec{n}_\Gamma$ (Berge et al., 2017; Schneider et al., 2017), where Γ denotes the faces as defined in Equation (2.12). Further, $\vec{n}_\Gamma$ describes the integrated normal,

$$\vec{n}_\Gamma = \int_\Gamma \vec{n}\, dS, \tag{2.15}$$

as the faces generally cannot be assumed to be planar. The integration of Equation (2.15) is done in Subsection 2.9.1.

Let m_i denote the number of faces in cell i. In conormal decomposition, the conormal $\vec{d}_\Gamma$ for each face $k = 1, ..., m_i$ of a cell i, is expressed as a linear combination of vectors $\vec{t}_k$, for which ∇p is assumed known, as follows

$$\nabla p \cdot \vec{d}_\Gamma = \sum_k \alpha_k \nabla p \cdot \vec{t}_k = \sum_k \Big[\alpha_k\big(p(\vec{x}_k) - p(\vec{x}_i)\big) + \mathcal{O}\big(\|\vec{x}_k - \vec{x}_i\|^2\big)\Big], \quad \vec{t}_k = \vec{x}_k - \vec{x}_i. \tag{2.16}$$

Here, $\vec{x}_i$ denotes the centroid of cell i, and $\vec{x}_k$ is some point in a neighboring cell k. The vectors $\vec{t}_k$ and corresponding coefficients α_k can be found using search algorithms with the constraint $\alpha_k \geq 0$, such that $\vec{d}_\Gamma = \sum_k \alpha_k \vec{t}_k$. The choice of the vectors $\vec{t}_k$ is important as the decomposition in general is neither trivial nor unique. Thus, search algorithms might give a variety of solutions, where the best one is difficult to distinguish. As proposed by Schneider et al. (2017), optimization techniques can be applied. We then let the points $\vec{x}_k$ be face interpolation points and denote them $\vec{x}_{\Gamma_{i,k}}$. An illustration of a cell with the mentioned variables can be seen in Figure 2.3, where the permeability is so that it changes the flow direction across the face $\Gamma_{i,j}$, as illustrated by the vector $\vec{d}_{\Gamma_{i,j}}$.

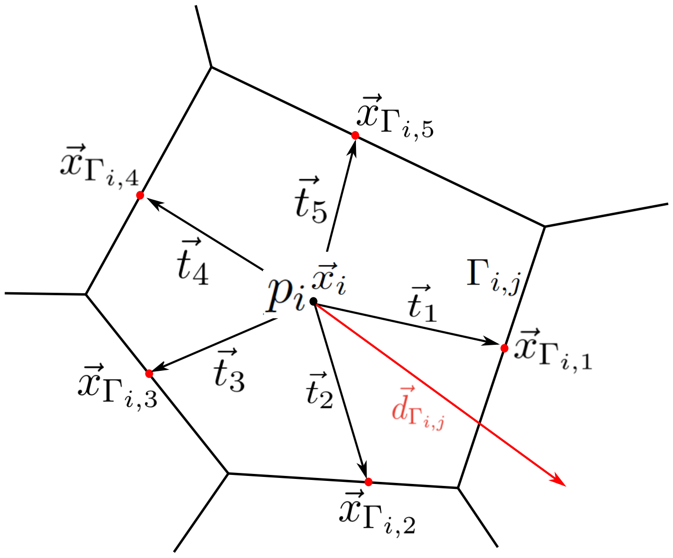


Figure 2.3: A cell with cell centroid $\vec{x}_i$ and $k = 5$ faces with face interpolation points $\vec{x}_{\Gamma_{i,k}}$, together with the decomposed conormal vector for face 1 in red.

The face interpolation points $\vec{x}_{\Gamma_{i,j}}$ can be calculated using various strategies, where the harmonic averaging point is suitable for complex grid geometries (Agélas et al., 2009). The harmonic averaging point does not require any sub-grid information, and is thus easy to calculate. For the harmonic averaging point of a face $\Gamma_{i,j}$, it is assumed that the pressure is an affine function in the two cells Ω_i and Ω_j, and it is assumed to be continuous over the face. Thus, the following continuity condition holds

$$\nabla p_i \cdot \mathcal{K}_i \cdot \vec{n}_{\Gamma_{i,j}} = \nabla p_j \cdot \mathcal{K}_j \cdot \vec{n}_{\Gamma_{j,i}}.$$

By taking the continuity and the restriction of the affine function into account, the pressure in a point $\vec{x}$ on the face $\Gamma_{i,j}$ can be defined

$$p|_{\Gamma_{i,j}}(\vec{x}) = \frac{\beta_{i,j} p_i + \beta_{j,i} p_j}{\beta_{i,j} + \beta_{j,i}} + \vec{g}_{\Gamma_{i,j}} \cdot (\vec{x} - \vec{x}_{\Gamma_{i,j}}),$$

where $\vec{g}_{\Gamma_{i,j}}$ is a tangential gradient to the face $\Gamma_{i,j}$, and $\vec{x}_{\Gamma_{i,j}}$ now denotes the harmonic averaging point on $\Gamma_{i,j}$ defined with $\beta_{i,j}$ and $\beta_{j,i}$ as follows

$$\vec{x}_{\Gamma_{i,j}} = \frac{\beta_{i,j}\vec{x}_i + \beta_{j,i}\vec{x}_j + (\mathcal{K}_i - \mathcal{K}_j)\cdot \vec{n}_{\Gamma_{i,j}}}{\beta_{i,j} + \beta_{j,i}}, \quad \beta_{i,j} = \frac{\vec{n}_{\Gamma_{i,j}} \cdot \mathcal{K}_i \cdot \vec{n}_{\Gamma_{i,j}}}{\operatorname{dist}(\vec{x}_i, \Gamma)} \qquad (2.17)$$
$$\beta_{j,i} = \frac{\vec{n}_{\Gamma_{i,j}} \cdot \mathcal{K}_j \cdot \vec{n}_{\Gamma_{i,j}}}{\operatorname{dist}(\vec{x}_j, \Gamma)}.$$

Here, $\operatorname{dist}(\vec{x}, \Gamma) = \inf_{\vec{z}\in\Gamma} \|\vec{x} - \vec{z}\|_2$, and is the distance from some point $\vec{x}$ to a face Γ. The face interpolation points are always calculated such that $\vec{x}_{\Gamma_{i,j}} = \vec{x}_{\Gamma_{j,i}} = \vec{x}_{\Gamma_{ij}}$.

Having defined the face interpolation points as the harmonic averaging points, we can move on to the problem of optimization. To find optimal coefficients α_k so that $\vec{d}_\Gamma = \sum_k \alpha_k \vec{t}_k$, we search for a solution to the constrained minimization problem

$$\min_{\alpha\in \mathrm{R}^{m_i}} F(\alpha) \qquad \text{subject to} \qquad \vec{d}_\Gamma = \mathbf{t}\alpha, \quad \alpha_k \geq 0 \quad k = 1, ..., m_i, \qquad (2.18)$$

where the matrix $\mathbf{t} \in \mathrm{R}^{d\times m_i}$ contains the vectors that are used for the decomposition, i.e., $\mathbf{t} = (\vec{t}_1, ..., \vec{t}_{m_i})$. The objective function can be set to $F(\alpha) = \sum_k \alpha_k$. However, there will exist cell geometries where the restriction to nonnegative α's may cause the minimization problem to not have a solution. Further, problems can occur if the vector $\alpha_k \vec{t}_k$ is too long, so that it points beyond the neighboring cell; for instance if the neighboring cell is very thin. To be able to get past this problem, the restriction to nonnegative α's must be weakened. This can be done by choosing a different objective function and adding weighting parameters in the decomposition so that the chosen objective function is less dependent on the vector lengths. We thus introduce the weighting parameters

$$\sigma_k = \frac{\|\vec{t}_k\|_2}{\|\vec{d}_\Gamma\|_2},$$

and normalize $\vec{d}_\Gamma$ and $\mathbf{t}$

$$\vec{d}^{\,n}_\Gamma = \frac{\vec{d}_\Gamma}{\|\vec{d}_\Gamma\|_2}, \quad \mathbf{t}^n = \frac{\vec{t}}{\|\vec{t}\|_2}.$$

We then set the objective function to $F(\alpha) = \sum_k \sigma_k \alpha_k$, so that we now search for a solution to the following minimization problem

$$\min_{\gamma\geq 0, \sigma_i\alpha_i \in \mathrm{R}^{m_i}} \kappa\gamma + \sum_k \sigma_k \alpha_k \quad \text{subject to} \quad \vec{d}^{\,n}_\Gamma = \mathbf{t}^n \sigma\alpha, \qquad (2.19)$$
$$\sum_k \alpha_k \geq \delta, \; \sigma_k\alpha_k \geq -\gamma.$$

We require that $\delta \geq 0$, and that the constant $\kappa \gg \frac{m_i}{\min \sigma_k}$ so that $\gamma = 0$ if a positive conormal decomposition exists. We define the face stencil $\mathcal{S}_\Gamma$ as the set of indices needed for the conormal decomposition in each cell, i.e., the number of faces in a cell where $\alpha_k \neq 0$.

After the decomposition of $\vec{d}_{i,j}$ and $\vec{d}_{j,i}$ is done for each face $\Gamma_{i,j} \in \mathcal{F}$, we get

$$\vec{d}_{i,j} = \sum_{k \in \mathcal{S}_{ij}} \alpha_{ij,k} \vec{t}_{i,k}, \quad \vec{d}_{j,i} = \sum_{k \in \mathcal{S}_{ji}} \alpha_{ji,k} \vec{t}_{j,k}, \tag{2.20}$$

where $\vec{t}_{i,k}$ and $\vec{t}_{j,k}$ are defined as in Equation (2.16) for the two cells i and j connected by the face Γ_{ij}. To get an approximation for the flux using the conormal decomposition, we assume that $\mathcal{K}$ is a constant tensor and look at the integrand in Equation (2.10) to see that $\vec{v} \cdot \vec{n} = -\mathcal{K}\nabla p \cdot \vec{n} = -\nabla p \cdot (\mathcal{K}\vec{n}) = -\nabla p \cdot \vec{d}_\Gamma$. We now use (2.16) and the notation in (2.20) to get the following approximation for the face flux on each side of a face $\Gamma_{i,j}$

$$v_{i,j} = -\sum_{k \in \mathcal{S}_{ij}} \alpha_{ij,k}(p_{\Gamma_{ik}} - p_i), \quad v_{j,i} = -\sum_{k \in \mathcal{S}_{ji}} \alpha_{ji,k}(p_{\Gamma_{jk}} - p_j). \tag{2.21}$$

In (2.21), p_i and p_j are the primary cell pressure values evaluated at the cell centroids, and $p_{\Gamma_{ik}}$ and $p_{\Gamma_{jk}}$ are the secondary face values evaluated at the harmonic averaging point. The secondary values are reconstructed from cell pressures,

$$p_{\Gamma_{i,j}} = \frac{\beta_{i,j} p_i + \beta_{j,i} p_j}{\beta_{i,j} + \beta_{j,i}} = \omega_{i,j} p_i + \omega_{j,i} p_j, \tag{2.22}$$

where $\beta_{i,j}$ and $\beta_{j,i}$ are defined as in (2.17). The approximation in (2.22) is allowed by the harmonic averaging point. We now let $\tilde{\alpha}_{ij,k} = \alpha_{ij,k}\omega_{ki}$, and insert (2.22) into (2.21) to get

$$v_{i,j} = -\sum_{k \in \mathcal{S}_{ij}} \tilde{\alpha}_{ij,k}(p_k - p_i), \quad v_{j,i} = -\sum_{k \in \mathcal{S}_{ji}} \tilde{\alpha}_{ji,k}(p_k - p_j). \tag{2.23}$$

Finally, we have the total face flux as a weighted sum of the fluxes in (2.23) as follows

$$v_{ij} := \mu_{i,j} v_{i,j} - \mu_{j,i} v_{j,i}, \tag{2.24}$$

with $\mu_{i,j} + \mu_{j,i} = 1$, $0 \leq \mu_{i,j} \leq 1$, where we require that $v_{ji} = -v_{ij}$. Notice that for the total flux, the comma between i and j is removed. The half-fluxes in (2.23) inserted into (2.24) gives us

$$v_{ij} = T_{i,j} p_i - T_{j,i} p_j + (\mu_{j,i}\lambda_{j,i} - \mu_{i,j}\lambda_{i,j}),$$

with transmissibilities

$$\begin{aligned} T_{i,j} &= \mu_{i,j} \sum_{k \in S_{ij}} \tilde{\alpha}_{ij,k} + \mu_{j,i} \sum_{k \in S_{ji} \cap \{i\}} \tilde{\alpha}_{ji,k}, \\ T_{j,i} &= \mu_{j,i} \sum_{k \in S_{ji}} \tilde{\alpha}_{ji,k} + \mu_{i,j} \sum_{k \in S_{ij} \cap \{j\}} \tilde{\alpha}_{ij,k}, \end{aligned} \tag{2.25}$$

and

$$\lambda_{i,j} := \sum_{k \in S_{ij} \setminus \{j\}} \tilde{\alpha}_{ij,k} p_k, \quad \lambda_{j,i} := \sum_{k \in S_{ji} \setminus \{i\}} \tilde{\alpha}_{ji,k} p_k.$$

The main goal now is to calculate the weights so that the residual $r_{\Gamma_{i,j}} = \mu_{j,i}\lambda_{j,i} - \mu_{i,j}\lambda_{i,j}$ is minimized. We thus choose these weights so that

$$\begin{aligned} \mu_{i,j} = \mu_{j,i} &= 0.5, && \text{if } \lambda_{i,j} = \lambda_{j,i} = 0, \\ \mu_{i,j} = \frac{|\lambda_{j,i}|}{|\lambda_{i,j}| + |\lambda_{j,i}|}, \ \mu_{j,i} &= \frac{|\lambda_{i,j}|}{|\lambda_{i,j}| + |\lambda_{j,i}|}, && \text{otherwise.} \end{aligned} \tag{2.26}$$

The mentioned residual, $r_{\Gamma_{i,j}}$, can be viewed as a truncation error, and will be zero if $\lambda_{i,j}\lambda_{j,i} \geq 0$, which follows from Equation (2.26), and the expression for r:

$$r_{\Gamma_{i,j}} = \begin{cases} \mu_{j,i}\lambda_{j,i} - \mu_{i,j}\lambda_{i,j}, & \text{if } \Gamma_{i,j} \in \partial\Omega_i \cap \partial\Omega_j, \\ -\lambda_{i,j} & \text{if } \Gamma_{i,j} \in \partial\Omega_i \cap \Gamma_D, \end{cases}$$

as seen in Potier (2009). Given Equation (2.25) and (2.26), we see that the transmissibilities in general depend on unknown pressure values, and thus the flux expression is likely to be nonlinear. If $\lambda_{i,j}\lambda_{j,i} < 0$, the flux can be reformulated as

$$\begin{aligned} v_{ij} = &\left(T_{i,j} + \frac{|r_{\Gamma_{i,j}}| + r_{\Gamma_{i,j}}}{2(p_i + \varepsilon)} \right) p_i - \left(T_{j,i} + \frac{|r_{\Gamma_{i,j}}| - r_{\Gamma_{i,j}}}{2(p_j + \varepsilon)} \right) p_j \\ &+ \varepsilon \left(\frac{|r_{\Gamma_{i,j}}| + r_{\Gamma_{i,j}}}{2(p_i + \varepsilon)} - \frac{|r_{\Gamma_{i,j}}| - r_{\Gamma_{i,j}}}{2(p_j + \varepsilon)} \right), \end{aligned} \tag{2.27}$$

as presented by Gao and Wu (2015), where ε is a positive number up to machine precision. By neglecting the last term in Equation (2.27) multiplied with ε, the remaining first two terms represents the flux v_{ij}, i.e.,

$$v_{ij} = \left(T_{i,j} + \frac{|r_{\Gamma_{i,j}}| + r_{\Gamma_{i,j}}}{2(p_i + \varepsilon)} \right) p_i - \left(T_{j,i} + \frac{|r_{\Gamma_{i,j}}| - r_{\Gamma_{i,j}}}{2(p_j + \varepsilon)} \right) p_j. \tag{2.28}$$

The neglected term is the truncation error, τ, which will only impact the numerical result if $\tau \in (-|r_{\Gamma_{i,j}}|, |r_{\Gamma_{i,j}}|)$ and at the same time $|\tau| \nleq Ch^2$, as the system is of second order. To find the final system to be solved, the final flux in Equation (2.28) is inserted into the discrete system in Equation (2.11) to obtain a non-linear system on the form

$$\mathbf{A}(\vec{p})\vec{p} = \mathbf{b}(\vec{p}, Q, p_D, \nu_N),$$

with $Q \approx \int_{\Omega_i} q d\vec{x}$.

2.7 Monotonicity

Another important property of convergent and precise methods is the monotonicity of the method. This section will briefly go through the concept of monotone methods before monotonicity of the NPTFA method is outlined.

We consider the elliptic Poisson equation in Equation (2.8), and let L denote the operator $Lp = -\nabla \cdot (\mathcal{K}\nabla p)$ so that Equation (2.8) can be written on the form

$$Lp = q \tag{2.29}$$

in some open domain D. We assume as before that $\mathcal{K}$ is symmetric and positive definite. Further, we also assume $\mathcal{K}$ to be sufficiently smooth and that the source term q is non-negative: $q \geq 0$. It follows from Hopf's Lemma that if there is a point $x_0 \in D$ such that $p(x_0) \geq p(x)$ for all other $x \in D$, then p is constant in D (Prottern and Weinberger, 1984). A weaker form saying that if $q \geq 0 \in D$, then there is no point $x_0 \in D$ such that $p(x_0) < p(x)$ for all other $x \in D$, is used in Nordbotten et al. (2007). Thus, as this can be stated for any subdomain D, it follows that p can have no local minima in D. This property is used to define monotone methods.

Green's function is used to formulate the solution of Equation (2.29) with sufficiently smooth boundary $\partial\Omega$ of a domain $\Omega \subset D$. The solution will be on the form

$$p(x) = \int_{\Omega} G_{\Omega}(x, \varsigma) q(\varsigma) dS_{\varsigma}, \tag{2.30}$$

where $G_{\Omega}(x, \varsigma)$ is Green's function and $q(x) = \delta(x - \varsigma)$. The assumption of sufficiently smooth $\mathcal{K}$ and $\partial\Omega$ makes Green's function continuous at all points but ς. This gives that

$$G_{\Omega}(x, \varsigma) \geq 0 \quad \forall\, x, \varsigma \in \Omega. \tag{2.31}$$

It follows from Equation (2.30) and (2.31) that

$$q \geq 0 \quad \Rightarrow \quad p \geq 0 \quad \text{in } \Omega. \tag{2.32}$$

Equation (2.32) is referred to as the monotonicity property (Nordbotten et al., 2007).

We now look at a given grid in D, and assume that the discretization of Equation (2.29), with homogeneous Dirichlet boundary conditions, leads to a system on the form

$$\boldsymbol{A}\vec{p} = \vec{q}, \tag{2.33}$$

where both $\vec{p}$ and $\vec{q}$ are vectors of same length as the number of cells in the grid. If each element in $\boldsymbol{A}^{-1}$ is nonnegative, i.e.,

$$\boldsymbol{A}^{-1} \geq \mathbf{0}, \tag{2.34}$$

where $\mathbf{0}$ is the zero matrix, then the matrix is called monotone and the system will satisfy the same monotonicity property as in Equation (2.32), but in the discrete form

$$\vec{q} \geq \mathbf{0} \quad \Rightarrow \quad \vec{p} \geq \mathbf{0}.$$

We use this to define a monotone method.

Definition 2.7.1 (Monotone Method)**.** If Equation (2.33) is a discretization of Equation (2.29) with homogeneous Dirichlet boundary conditions on any subgrid of a grid in D, then a method which defines this discretization is said to be monotone if inequality (2.34) is satisfied.

For the NTPFA method using optimization for the conormal decomposition, monotonicity is guaranteed when a positive decomposition exists, i.e., when all coefficients $\alpha \geq 0$. When all coefficients are positive, the transmissibilities $T_{i,j}$ and $T_{j,i}$ in Equation (2.25) will also be positive and thus the matrix $\mathbf{A}(\vec{p})$ will be monotone. This follows from the fact that the columns in $\mathbf{A}(\vec{p})$ will have non-negative sum, and is thus irreducible.

2.8 Multipoint Flux Approximation

For the sake of completeness and convergence results presented later on, a brief overview of the multipoint flux approximation (MPFA) method is given.

The MPFA method more accurately approximates the pressure derivatives parallel to the cell faces to obtain a consistent discretisation, also for grids that are not K-orthogonal. This approximation is done by developing a multipoint stencil. A dual grid is introduced inside each control volume cell, where each cell in the dual grid is termed interaction regions (Aavatsmark, 2002). The parts of the interfaces of the main grid inside each interaction region are termed subinterfaces. A construction of the dual grid for a grid consisting of four cells is shown in Figure 2.4, where the dashed lines illustrate the dual grid. The lines are drawn from each cell centroid to the face centroids, and are extended outside the primary grid. For

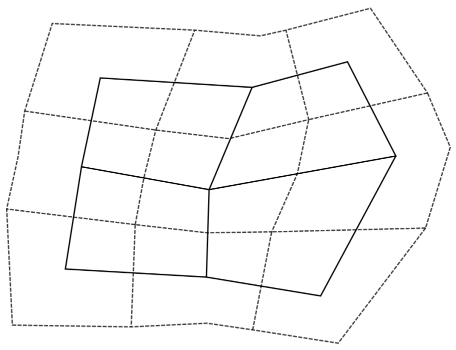


Figure 2.4: A grid consisting of four cells, and its dual grid in dashed lines.

the MPFA method, transmissibility coefficients are found for the subinterfaces, and contributes to the transmissibility for the main cell. That way, still taking into account the same continuity conditions, transmissibilities from multiple cells contribute to determine the transmissibility of the cell interface. We thus get a flux expression through a face $\Gamma_{i,j}$ on the form

$$v_{i,j} = \sum_{l \in L} \tau_{ij,l} p_l, \tag{2.35}$$

where $\tau_{ij,l}$ are transmissibility coefficients with $\sum_{l \in L} \tau_{ij,l} = 0$, and L consists of the number on the cells that are connected by the dual grid, including cell i. For the grid in Figure 2.5, $L = (1, 2, 3, 4, 5, 6)$ for the flux between cell $i = 3$ and cell $j = 4$.

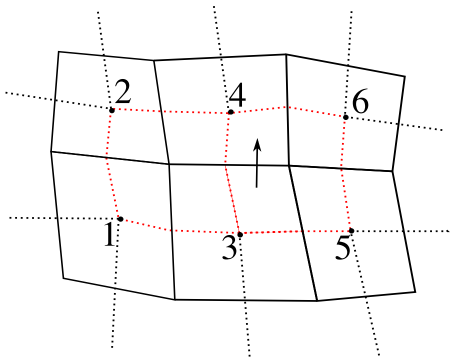


Figure 2.5: A grid with 6 cells in solid lines and the corresponding dual grid in dashed lines. The red lines on the dual grid indicate interaction regions that contribute to the flux from cell 3 to cell 4.

There is a whole class of MPFA methods, and if one assumes linear pressure inside each sub region, the method is called MPFA-O. In 2D, each interaction region will consist of four subinterfaces Γ_l and four cells Ω_i, each with two local surfaces k. For the MPFA-O method, the flux through a subinterface Γ_l in an interaction region will then be

$$v_l = \sum_{k=1}^{2} \tilde{\omega}_{lik}(\pi_{lk} - p_{x_{\Gamma_l}}), \quad \tilde{\omega}_{lik} = -\frac{\vec{n}_l^T \mathcal{K}_i \vec{u}_{ik}}{\det(X_i)},$$

where

$$X_i = \begin{bmatrix} (x_{\Gamma_{l1}} & -x_{\Omega_i})^T \\ (x_{\Gamma_{l2}} & -x_{\Omega_i})^T \end{bmatrix},$$

with cell centroid x_{Ω_i} and face centroid $x_{\Gamma_{lk}}$, $k \neq l$. Further, $\vec{u}_{ik}$ is defined

$$\vec{u}_{i1} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} (x_{\Gamma_{l2}} - x_{\Omega_i}), \quad \vec{u}_{i2} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} (x_{\Gamma_{l1}} - x_{\Omega_i}),$$

and is the inner normal vector to the triangle edge joining x_{Ω_i} and $x_{\Gamma_{lk}}$, with length equal to the length of this edge. Lastly, π_{ik} is the pressure at the centroid of the two local faces of cell i. An illustration of the given variables for one interaction region, with cell centroids of the four cells involved denoted 1, 2, 3 and 4, is given in Figure 2.6.

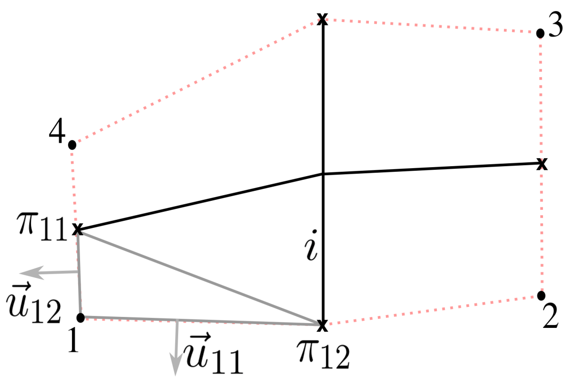


Figure 2.6: One interaction region in dashed lines, consisting of four cells with cell centroids 1,2,3 and 4, and four subinterfaces marked in solid lines. The vectors $\vec{u}_{i1}$ and $\vec{u}_{i2}$ (marked for cell $\Omega_1 = 1$) are for the approximation of the flux through subinterface Γ_l. The pressures π_{ik} is positioned at the face centroids, here denoted by $\mathbf{x}$.

Overall, this method requires a more dense calculation, resulting in higher cost computation, but on the other hand yields consistency. It is, however, not unconditionally monotone as inequality (2.34) in general is not satisfied (Aavatsmark, 2002), with the matrix of coefficients $\mathbf{A}$ defined through Equation (2.35)

$$\sum_j v_{i,j} = \sum_j a_{ij} p_j = \mathbf{A}\vec{p}.$$

2.9 Integration and Optimization

In this section, we will go through some theory on a common optimization method, and the integration of the normal vector in Equation (2.15). We begin with the integrated normal, and finish the section and chapter with a brief overview of the interior point method, which is a suitable method for the optimization problem in Equation (2.19).

2.9.1 Integrated Normal

We start of by repeating the integral in question

$$\vec{n}_\Gamma = \int_\Gamma \vec{n} dS.$$

For evaluation in 3D, the transformation from the unit cube to physical space is considered. The transformation is shown in Figure 2.7, where eight points belonging to a cell are mapped by a trilinear mapping to $\vec{x}_i, i \in [1, 8]$.

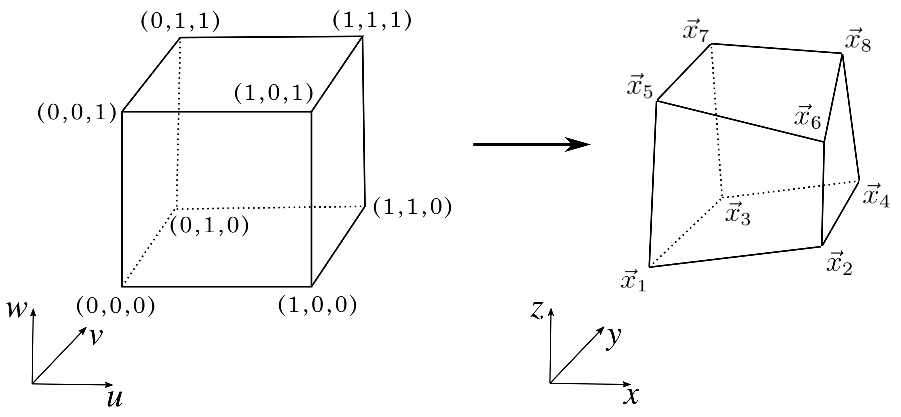


Figure 2.7: The trilinear transformation from the unit cube to the physical cell.

We consider one face in the unit box, see Figure 2.8. The unit normal vector to this face can be written

$$\vec{n} = \frac{\partial \vec{x}}{\partial u} \times \frac{\partial \vec{x}}{\partial v},$$

where $\vec{x}$ is the bilinear mapping for the face transformation (Aavatsmark, 2002). Thus any point $\vec{x}$ at the surface in Figure 2.8 are given by

$$\vec{x} = u(v\vec{x}_4 + (1-v)\vec{x}_3) + (1-u)(v\vec{x}_2 + (1-v)\vec{x}_1).$$

We can now derive an expression for the integrated face normal using the partial derivatives of x given by

$$\frac{\partial \vec{x}}{\partial u} = v(\vec{x}_4 - \vec{x}_3) + (1-v)(\vec{x}_2 - \vec{x}_1)$$
$$\frac{\partial \vec{x}}{\partial v} = u(\vec{x}_4 - \vec{x}_2) + (1-u)(\vec{x}_3 - \vec{x}_1).$$

Thus

$$\begin{aligned}\vec{n}_\Gamma &= \int_0^1\int_0^1 \frac{\partial \vec{x}}{\partial u} \times \frac{\partial \vec{x}}{\partial v} du\, dv\\ &= \int_0^1\int_0^1 [v(\vec{x}_4 - \vec{x}_3) + (1-v)(\vec{x}_2 - \vec{x}_1)] \times [u(\vec{x}_4 - \vec{x}_2) + (1-u)(\vec{x}_3 - \vec{x}_1)] du\, dv.\end{aligned}$$

Using cross-product properties on the integrand we obtain

$$\begin{aligned}\vec{n}_\Gamma = \int_0^1\int_0^1 & vu(\vec{x}_4 - \vec{x}_3) \times (\vec{x}_4 - \vec{x}_2) + v(1-u)(\vec{x}_4 - \vec{x}_3) \times (\vec{x}_3 - \vec{x}_1) + \\ &(1-v)u(\vec{x}_2 - \vec{x} - 1) \times (\vec{x}_4 - \vec{x}_2) + (1-v)(1-u)(\vec{x}_2 - \vec{x}_1) \times (\vec{x}_3 - \vec{x}_1)\, du\, dv.\end{aligned}$$

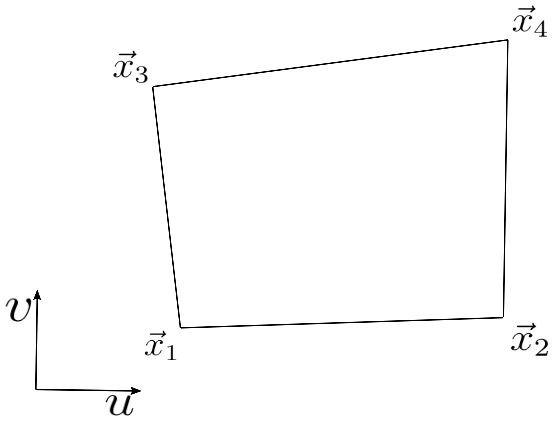


Figure 2.8: Cell surface for corner-point geometry used for the integrated normal.

Integration yields

$$\begin{aligned}\vec{n}_\Gamma = \Big[&\frac{v^2u^2}{2}(\vec{x}_4 - \vec{x}_3) \times (\vec{x}_4 - \vec{x}_2) + \frac{uv^2}{2}(1 - \frac{u}{2})(\vec{x}_4 - \vec{x}_3) \times (\vec{x}_3 - \vec{x}_1)\\ &+ \frac{vu^2}{2}(1 - \frac{v}{2})(\vec{x}_2 - \vec{x}_1) \times (\vec{x}_4 - \vec{x}_2)\\ &+ vu(1 - \frac{v}{2})(1 - \frac{u}{2})(\vec{x}_2 - \vec{x}_1) \times (\vec{x}_3 - \vec{x}_1)\Big]\Big|_{(u,v)=0}^{(u,v)=1},\end{aligned}$$

and we finally obtain

$$\begin{aligned}\vec{n}_\Gamma &= 0.25\Big((\vec{x}_4 - \vec{x}_3) \times (\vec{x}_4 - \vec{x}_2) + (\vec{x}_4 - \vec{x}_3) \times (\vec{x}_3 - \vec{x}_1)\\ &\quad + (\vec{x}_2 - \vec{x}_1) \times (\vec{x}_4 - \vec{x}_2) + (\vec{x}_2 - \vec{x}_1) \times (\vec{x}_3 - \vec{x}_1)\Big)\\ &= 0.5(\vec{x}_1 \times \vec{x}_2 - \vec{x}_1 \times \vec{x}_3 + \vec{x}_2 \times \vec{x}_4 - \vec{x}_3 \times \vec{x}_4).\end{aligned}$$

2.9.2 Optimization

The optimization problem in Equation (2.19) is a linear problem, and thus there are many optimization methods that can be used to solve it. For the code written, Matlab's `fmincon` function has been used. The `fmincon` function allows for nonlinear constraints, and so we can insert the third constraint of Equation (2.19) as a nonlinear constraint. The function's default algorithm is the interior-point method, which is a method that solves convex optimization problems, but can be extended to non-convex functions (Nocedal and Wright, 2006, Chapter 19). We will shortly explain the approach for the interior-point method, and use of `fmincon` for the written code is found in Section 3.4, but first a few basic concepts for numerical optimization is required.

We look at a general constrained optimization problem on the form

$$\min_{x\in\mathrm{R}^d} f(x) \quad \text{subject to} \begin{cases} c_i(x) = 0,\, i \in \mathcal{E}, \\ c_i(x) \geq 0,\, i \in \mathcal{I}, \end{cases} \tag{2.36}$$

where $f(\cdot)$ is the objective function and is a mapping from R^d to R, and $x \in \mathrm{R}^d$ is a real vector. Further, $\mathcal{E}$ is the set of indices for the equality constrains, and $\mathcal{I}$ is the set of indices for the inequality constraints. We say that the solution to the optimization problem is the optimal point x^*, so that the optimal value of the function is at $f(x^*)$. I.e., the point x^* is a global minimizer if $f(x^*) \leq f(x)\,\forall\, x$ in a feasible set Ω_{opt}, which is defined

$$\Omega_{opt} = \{x \,|\, c_i(x) = 0,\, i \in \mathcal{E};\, c_i(x) \geq 0,\, i \in \mathcal{I}\}, \tag{2.37}$$

i.e., the set of points where all constraints are satisfied. Further, we say that an inequality constraint is active at a feasible point x if $c_i(x) = 0$, and inactive if $c_i(x) > 0$. The active set $\mathcal{A}(x)$ at any feasible x is thus defined

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \,|\, c_i(x) = 0\}.$$

We use the active set to define the linear independence constraint qualification (LICQ), which we need later on to state the first- and second-order necessary conditions for a solution to be optimal. We say that the LICQ holds if the active constraint gradients, $\{\nabla c_i(x), i \in \mathcal{A}(x)\}$, are linearly independent at a point x in the active set $\mathcal{A}(x)$.

We will also need the set of linearized feasible directions. Given a feasible point x and the active constraint $\mathcal{A}(x)$, we can define the set of linearized feasible directions $\mathcal{D}(x)$ as

$$\mathcal{D}(x) = \left\{ d \middle| \begin{array}{ll} d^T \nabla c_i(x) = 0, & \forall\, i \in \mathcal{E}, \\ d^T \nabla c_i(x) \geq 0, & \forall\, i \in \mathcal{A}(x) \cap \mathcal{I} \end{array} \right\},$$

where d is a tangent to Ω_{opt} at a point x, and the set of all d is called the tangent cone.

We now introduce the Lagrangian function $\mathcal{L}$ with the Lagrange multiplier λ for Equation (2.36)

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x), \tag{2.38}$$

and notice that $\nabla_x \mathcal{L}(x, \lambda_i) = \nabla f(x) - \lambda_i \nabla c_i(x)$. Thus, at the optimal point x^* of Equation (2.36), there is a scalar λ_i^* such that $\nabla_x \mathcal{L}(x^*, \lambda_i^*) = 0$. This implies that we can search for solutions to Equation (2.36) by seeking the stationary points of Equation (2.38).

The first-order necessary conditions for x^* to be a local minimizer can now finally be stated.

Definition 2.9.1 (KKT conditions)**.** The first-order necessary conditions, popularly known as the KKT conditions after Karush, Kuhn, and Tucker (Karush, 1939; Kuhn and Tucker, 1951), are

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned} \tag{2.39}$$

Given that the LICQ holds at a local solution x^*, there exists a Lagrange multiplier vector λ^* with components $\lambda_i^*, i \in \mathcal{E} \cup \mathcal{I}$, that satisfies the KKT conditions in Equation (2.39) for the local solution x^* and continuously differentiable functions f and c_i.

If we have that both the LICQ and KKT conditions hold for a local solution x^* and Lagrange multiplier λ^*, we have that x^* is a strict local solution if the second order necessary conditions hold. The second order necessary conditions are given by

$$w^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) w \geq 0, \quad \forall\, w \in \mathcal{C}(x^*, \lambda^*),\, w \neq 0,$$

where $\mathcal{C}(x^*, \lambda^*)$ is the critical cone defined

$$\mathcal{C}(x^*, \lambda^*) = \{w \in \mathcal{D}(x^*) \,|\, \nabla c_i(x^*)^T w = 0, \forall\, i \in \mathcal{A}(x^*) \cap \mathcal{I} \text{ with } \lambda_i^* > 0\}.$$

2.9.3 Interior-Point Methods

Interior-point methods, also known as barrier methods, have proved themselves to be effective for nonlinear optimization (Nocedal and Wright, 2006). For interior-point methods, the solution is sought inside the feasible region defined by Equation (2.37), and a barrier function, or barrier parameter, is used to prevent the solution from going outside the feasible region. There are several ways to interpret interior-point methods, where we limit ourselves to take a closer look at a continuation approach.

We begin by transforming the inequality constraints of Equation (2.36) into equality constraints. This is commonly done by introducing a nonnegative vector s of so-called slack variables, so that we get a problem formulation on the form

$$\min_{x \in \mathrm{R}^d} f(x) \quad \text{subject to} \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) - s_i = 0, & i \in \mathcal{I} \\ s_i \geq 0. \end{cases} \tag{2.40}$$

We let $J_{\mathcal{E}}$ and $J_{\mathcal{I}}$ denote the Jacobi matrix of the equality and inequality constraint functions respectively, and $\lambda_{\mathcal{E}}$ and $\lambda_{\mathcal{I}}$ their Lagrange multipliers. The KKT conditions of Definition 2.9.1 can for Equation (2.40) then be written as follows

$$\begin{aligned} \nabla f(x) - J_{\mathcal{E}}^T(x)\lambda_{\mathcal{E}} - J_{\mathcal{I}}^T(x)\lambda_{\mathcal{I}} &= 0, \\ S\lambda_{\mathcal{I}} - \mu\mathbb{1} &= 0, \\ c_E(x) &= 0, \\ c_I(x) - s &= 0, \end{aligned} \tag{2.41}$$

where μ is introduced as the barrier parameter. Further, S is a diagonal matrix whose diagonal entries are given by s, and $\mathbb{1}$ is a vector on ones.

Due to condition two of Equation (2.41), we want to force μ to be strictly positive so that the variables s and $\lambda_{\mathcal{I}}$ are also positive. The interior-point method now consists of approximately solving the KKT conditions of Equation (2.41) for a sequence of positive parameters $\{\mu_k\}$ which converges to zero as k increase, while maintaining $s, \lambda_{\mathcal{I}} > 0$. The iterates are decreased using a function that determines whether a step is productive and should be accepted. Such a function is often referred to as a merit function, and by using a merit function to decrease the iterates, the iteration is likely to converge to a minimizer and a KKT point. Further, if the KKT conditions and the second order necessary conditions holds, and we have that $\lambda_i^* > 0$ for each $i \in \mathcal{I} \cap \mathcal{A}(x^*)$, then the system in Equation (2.40) has a locally unique solution for all sufficiently small positive values μ.

When numerically solving an optimization problem, the optimal solution is sought iteratively, and a search direction determines in which direction the next step should be taken. The search direction p for the interior-point method is found by applying Newton's method to Equation (2.40). We get a system on the form

$$\begin{bmatrix} \nabla_{xx}^2\mathcal{L} & 0 & -J_{\mathcal{E}}^T(x) & -J_{\mathcal{I}}^T(x) \\ 0 & Z & 0 & S \\ J_{\mathcal{E}}(x) & 0 & 0 & 0 \\ J_{\mathcal{I}}(x) & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_s \\ p_{\lambda_{\mathcal{E}}} \\ p_{\lambda_{\mathcal{I}}} \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - J_{\mathcal{E}}^T(x)\lambda_{\mathcal{E}} - J_{\mathcal{I}}^T(x)\lambda_{\mathcal{I}} \\ S\lambda_{\mathcal{I}} - \mu\mathbb{1} \\ c_{\mathcal{E}}(x) \\ c_{\mathcal{I}}(x) - s \end{bmatrix}, \tag{2.42}$$

where Z is a diagonal matrix with diagonal entries given by $\lambda_{\mathcal{I}}$. The Lagrangian for Equation (2.40) is written

$$\mathcal{L}(x, s, \lambda_{\mathcal{E}}, \lambda_{\mathcal{I}}) = f(x) - \lambda_{\mathcal{E}}^T c_{\mathcal{E}}(x) - \lambda_{\mathcal{I}}^T(c_{\mathcal{I}}(x) - s).$$

The search direction is then used to compute the new iterate

$$\begin{aligned} x_{k+1} &= x_k + \beta_s^{\max} p_x, \quad s_{k+1} = s_k + \beta_s^{\max} p_s, \\ \lambda_{\mathcal{E},k+1} &= \lambda_{\mathcal{E},k} + \beta_{\lambda_{\mathcal{I}}}^{\max} p_{\lambda_{\mathcal{E}}}, \quad \lambda_{\mathcal{I},k+1} = \lambda_{\mathcal{I},k} + \beta_{\lambda_{\mathcal{I}}}^{\max} p_{\lambda_{\mathcal{I}}}, \end{aligned} \tag{2.43}$$

where

$$\begin{aligned} \beta_s^{\max} &= \max\{\beta \in (0,1] \, : \, s + \beta p_s \geq (1-\tau)s\}, \\ \beta_{\lambda_{\mathcal{I}}}^{\max} &= \max\{\beta \in (0,1] \, : \, \lambda_{\mathcal{I}} + \beta p_{\lambda_{\mathcal{I}}} \geq (1-\tau)\lambda_{\mathcal{I}}\}, \end{aligned} \tag{2.44}$$

with $\tau \in (0,1)$. The values of β in Equation (2.44) are there to prevent the variables s and $\lambda_{\mathcal{I}}$ from approaching 0 too quickly.

The pseudo-code for the interior-point algorithm can be seen in Algorithm 1, where E is some error function. Nocedal and Wright (2006) suggest an error function, with some vector norm $\|\cdot\|$, based on the KKT system of Equation (2.41):

$$\begin{aligned} E(x, s, \lambda_{\mathcal{E}}, \lambda_{\mathcal{I}}; \mu) = \max\{&\|\nabla f(x) - J_{\mathcal{E}}(x)^T\lambda_{\mathcal{E}} - J_{\mathcal{I}}(x)^T\lambda_{\mathcal{I}}\|, \|S\lambda_{\mathcal{I}} - \mu\mathbb{1}\|, \\ &\|c_{\mathcal{E}}(x)\|, \|c_{\mathcal{I}}(x) - s\|\}. \end{aligned}$$

From the pseudo-code, we see that the barrier parameter is fixed until the KKT conditions are satisfied for the current parameter to some accuracy in the inner loop. The parameter is then decreased in the outer loop, which is run according to some stopping test. The inner loop can however be removed to ensure that the barrier parameter is updated at each iteration. The barrier parameter must then be chosen according to a dynamic function on the form

$$\mu_{k+1} = \sigma_k \frac{s_k^T \lambda_{\mathcal{I},k}}{m},$$

where m is the number of inequality constraints in problem (2.40).

Algorithm 1 Basic pseudo-code for the interior-point method

```
set $x_0$ and $s_0 > 0$, compute initial values for $\lambda_{\mathcal{E},0}$ and $\lambda_{\mathcal{I},0} > 0$. Select initial
barrier parameter $\mu_0 > 0$ and parameters $\sigma, \tau \in (0,1)$.
$k \leftarrow 0$
while stopping test for Equation (2.40) is not satisfied do
  while $E(x_k, s_k, \lambda_{\mathcal{E},k}, \lambda_{\mathcal{I},k}; \mu_k) > \mu_k$ do
    $p \leftarrow$ Equation (2.42);
    $\beta_s^{\max}, \beta_{\lambda_{\mathcal{I}}}^{\max} \leftarrow$ Equation (2.44);
    $(x_{k+1}, s_{k+1}, \lambda_{\mathcal{E},k+1}, \lambda_{\mathcal{I},k+1}) \leftarrow$ Equation (2.43);
    $\mu_{k+1} \leftarrow \mu_k$;
    $k \leftarrow k+1$;
  end while
  Choose $\mu_k \in (0, \sigma\mu_k)$;
end while
```

Chapter 3

Tools

Throughout this work, SINTEF's add-on to Matlab, the Matlab Reservoir Simulation Toolbox (MRST), has been used as programming platform. A full description of MRST can be found in Lie (2016). The toolbox already has modules for solving equations for incompressible flow, such as the `incomp`, `mimetic`, and `mpfa` modules. There is also an unfinished module for the NTPFA method using search methods for the decomposition. Many of the methods already implemented into MRST involve in some way or another differentiation of large amount of data, and automation of these differentiations is crucial for efficient and exact evaluation. There is support for automatic differentiation (AD) in the AD-OO framework of MRST, which has been used for the NTPFA method to solve the Newton iteration performed after the coefficients are found.

A brief description of AD and the AD-OO framework follows an introduction to the grid setup in MRST. At the end of the chapter, the NTPFA method, as it has been written for MRST, is presented together with an example of a simple set-up.

3.1 Unstructured Grids

When modeling and simulating a physical system, the domain at hand is subdivided into a set of discrete, non-overlapping cells, which together make up the grid. Each cell will have a number of nodes/vertices and a set of edges/faces connecting the nodes. The nodes are numbered, however, not necessarily systematically. The faces are represented in a set so that cells sharing faces are easy to identify. Two cells sharing a face are called connected. If all cells in the grid share the same simple shape and are distributed in a regular repeating pattern, the grid is called a structured grid. In an unstructured grid, all cells can have different simple shapes, and any number of cells can share the same node. Thus the topology of the grid can change throughout space depending on the reservoir's structural architecture, and take into account that real reservoirs will have spatially varying permeability. This can for instance be done by generating a grid where nodes are placed on the face of a cell and thus leaving a face to connect more than two cells, as seen in Figure 3.1. For these kinds of grids, the faces that have floating nodes will be split at that node and numbered accordingly, so that, again, each face connects no more than two cells.

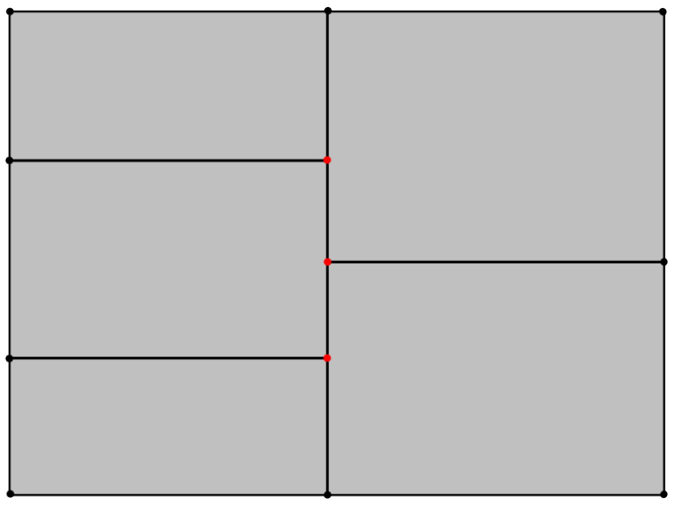
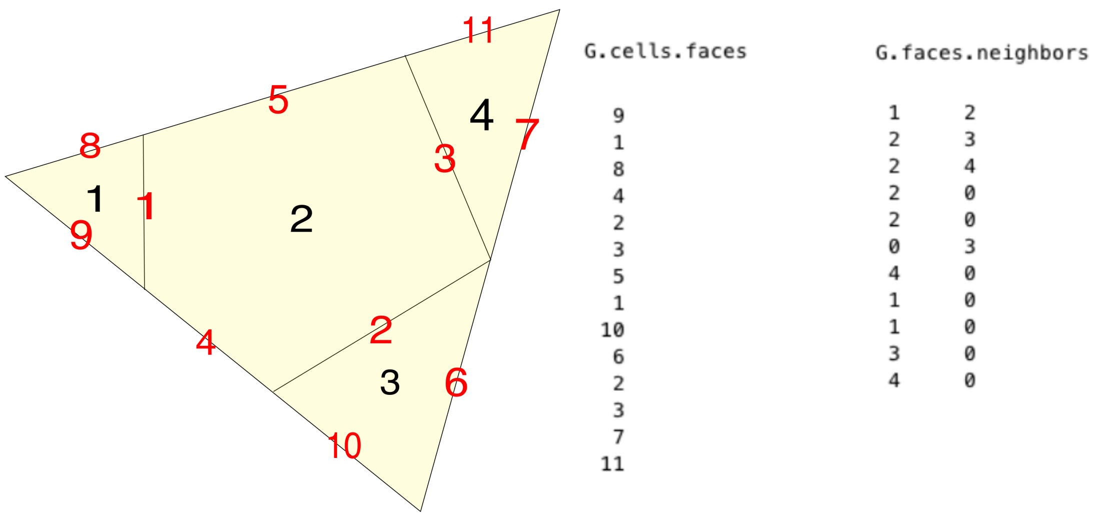


Figure 3.1: A grid with floating nodes in red.

In MRST, each grid `G` is stored as a structure which consists of three mandatory substructures; `cells`, `faces`, and `nodes`. The `cell` structure has three mandatory fields: `num`, `faces`, and `facePos`. `num` represents the number of cells in the grid, `faces` gives an index to the faces defining a cell, and all faces belonging to cell i are found in elements `facePos(i)` to `facePos(i+1)-1`. The `face` structure has four mandatory fields: `num`, `nodes`, `nodePos`, and `neighbors`. Again, the first is the number of faces in the grid, the second is an index to the nodes connected by a face, and through the third all nodes connected by a face is found. The fourth, `neighbors`, identifies the cells that share a common face. The `node` structure has two mandatory fields: `num`, and `coords`, where the first is the number of nodes in the grid, and the second gives the coordinates to each node. Each cell in the grid has a corresponding set of faces, and if two cells have a common face, the cells are neighboring cells. Each face corresponds to a set of edges, where the edges are determined by the nodes. Hence, by numbering and construction of the grids, it is easy to find neighboring cells and common faces. This is used when implementing and solving the flow equation using numerical methods discussed. In Figure 3.2, an example of a set up is shown for a simple unstructured grid. The grid is to the left with cell numbers marked with black, and face numbers marked with red. The corresponding MRST-representation for the faces and neighbors are to the right. In the grid, we see that cell 1 has three faces. These



```
G.cells.faces

 9
 1
 8
 4
 2
 3
 5
 1
10
 6
 2
 3
 7
11
```

```
G.faces.neighbors

1    2
2    3
2    4
2    0
2    0
0    3
4    0
1    0
1    0
3    0
4    0
```

Figure 3.2: A simple unstructured grid with cell numbers in black, face numbers in red, and the corresponding MRST structure to the right.

corresponds to the first three rows in `G.cells.faces`. Cell 2 has 5 faces, and thus row 4 through 8 correspond to cell 2. For structured grids, `G.cells.faces` will have a second column representing the direction of each face. The number of faces for each cell is found through the call `diff(G.cells.facePos)`, and the information for cell i is found through `G.cells.faces(G.cells.facePos(i) : G.cells.facePos(i+1)-1,:)`. For the faces, each row corresponds to the respective face number, i.e., face number one has neighboring cells 1 and 2, face number 2 has neighboring cells 2 and 3, etc. The routine `computeGeometry(G)` will add additional information to the substructures, such as face areas, centroids of both cells and faces, and face normals. The face normal is stored as one value for each face, and thus the sign of the normal must be changed if we are dealing with the second neighboring cell.

3.2 Discrete Differentiation Operators

As a basic model, the single-phase continuity equation is considered

$$\frac{\partial}{\partial t}(\phi\rho) + \nabla \cdot (\rho\vec{v}) = q, \quad \vec{v} = -\frac{\kappa}{\mu}(\nabla p - g\rho\nabla z),$$

with p as primary unknown. The basic implicit discretization reads

$$\begin{aligned} \frac{(\phi\rho)^{n+1} - (\phi\rho)^n}{\Delta t^n} + \nabla \cdot (\rho v)^{n+1} &= q^{n+1}, \\ \vec{v}^{n+1} &= -\frac{\kappa}{\mu^{n+1}}\big(\nabla(p^{n+1}) - g\rho^{n+1}\nabla(z)\big), \end{aligned}$$

where $\nabla \cdot (\cdot)$ is the divergence operator, and $\nabla(\cdot)$ is the gradient. All variables are defined as before. The divergence operator is a differential vector operator that gives the quantity of a vector field's source at each point, and produces a scalar field. For a continuous vector field $F = F_x + F_y + F_z$, where the subscript describes the unit direction, the divergence is defined

$$\nabla \cdot F = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}.$$

Further the gradient is formally defined

$$\nabla F = (\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}).$$

We will denote the discrete versions of the operators `div` and `grad`. They are introduced to mimic their continuous counterparts, and enable us to write a discrete version of the Poisson Equation given in Equation (2.8) in a very compact form. For the grid structure as presented in the previous section, the `div` operator is a linear mapping from faces to cells. The divergence describes the total amount of matter leaving a cell Ω and the discretized form can thus be written

$$\texttt{div}(v)[\Omega] = \sum_{\Gamma \in \mathcal{F}(\Omega)} \vec{v}[\Gamma]\mathbf{1}_{\{\Omega=\Omega_1\}} - \sum_{\Gamma \in \mathcal{F}(\Omega)} \vec{v}[\Gamma]\mathbf{1}_{\Omega=\Omega_2},$$

where Ω_1 is the first neighbor to the face Γ and Ω_2 is the second, i.e., first and second column of `G.faces.neighbors(`Γ`,:)`. We denote by m_Ω the total number of cells in the grid, and by m_Γ the total number of faces. The `grad` mapping is defined for any $\vec{p} \in \mathrm{R}^{m_\Omega}$ as

$$\texttt{grad}(\vec{p})[\Gamma] = \vec{p}[\Omega_2(\Gamma)] - \vec{p}[\Omega_1(\Gamma)],$$

and maps R^{m_Ω} to R^{m_Γ}.

Taking the flux approximation given by Equation (2.14) into consideration, we see that we can write Equation (2.8) with $\vec{v} = -\mathcal{K}\nabla p$ with the discrete operators as follows

$$\begin{aligned} \texttt{div}(\vec{v}) &= q \\ \vec{v} &= -T\texttt{grad}(p), \end{aligned}$$

where T is the transmissibility.

The discrete forms of the divergence and the gradient operators can be combined with automatic differentiation to write equations on a compact form, and thus automatically estimate Jacobi matrices without explicit calculations.

3.3 Automatic Differentiation

The key idea for automatic differentiation (AD) is to implement basic operations in a numerical environment (Neidinger, 2010). By use of basic derivative rules, the derivatives of a function can be calculated automatically in the numerical environment, allowing the user to evaluate both the function and it's derivatives.

In MRST, an AD object oriented (AD-OO) framework has been introduced. The AD-OO framework separates the implementation of physical models, discrete operators, nonlinear solvers and time-stepping, and assembly and solution of the linear system. Thus, by exploiting the AD-OO framework, new methods can be implemented to work with existing solvers in MRST, and models with more complex features, such as multi-phase flow, can be simulated more easily by use of inheritance of classes. This inheritance property has also been used for the NTPFA method implemented for this thesis, and will be explained shortly.

The AD-OO framework allows the user to define a new type of data for vectors in R. The new data uses operator overloading to calculate both resulting operations as well as the partial derivatives with respect to all primary variables. For instance, a variable x will be stored as an object with both the value of x and the derivative of x which will be 1. For a vector variable or multiple variables, the derivative will be the Jacobian. The standard is to use a simple matrix with respect to all primary variables to represent the Jacobian. However, MRST uses a list of matrices that represent the derivatives with respect to each individual variable that will constitute sub-blocks in the Jacobian of the full system (Lie, 2016, p. 609). In Example 3.3.1 a simple demonstration of the use of the AD in MRST is given.

Example 3.3.1. *We look at the function*

$$f(x, y) = 3x + 2y - 1,$$

with derivatives

$$\nabla f(x, y) = [3, 2].$$

We let $x = 5$, and $y = 10$. The function `[x,y] = initVariablesADI(5,10)` *returns the* `ADI` *representation of x and y, and f can now be assembled to obtain* *a third `ADI` variable. The output in Matlab is shown in Listing 3.1, where we see that the value of the function is `f.val = 34`, and the Jacobian is a matrix with two submatrices corresponding to x and y, i.e. `f.jac = [3] [2]`.*

Listing 3.1: Matlab output for AD representation of the problem in Example 3.3.1.

```
f =

  ADI with properties:

    val: 34
    jac: {[3]  [2]}
```

3.4 NTPFA in MRST

To finish off this chapter, we will go through the setup of the NTPFA method in MRST. We start by repeating the flux-expression for the NTPFA method as presented in Equation (2.28):

$$v_{ij} = \left(T_{i,j} + \frac{|r_{\Gamma_{ij}}| + r_{\Gamma_{ij}}}{2(p_i + \varepsilon)} \right) p_i - \left(T_{j,i} + \frac{|r_{\Gamma_{ij}}| - r_{\Gamma_{ij}}}{2(p_j + \varepsilon)} \right) p_j.$$

To be able to solve the nonlinear system of equations, the base class `PhysicalModel` in MRST has been used. This base class uses AD to implement a generic discretized model. To exploit the already existing functions in MRST, a subclass to `PhysicalModel` has been written, where a model for the NTPFA method is built. To distinguish the method using optimization for the decomposition from the one using search methods, the written model is henceforth referred to as NTPFAopt while the other will be referred to as NTPFA. The model is built using two different functions; `getCollactionSetOPT` and `computeNonLinearTransForOpt`. The `getCollactionSetOPT` function is run once for each grid to find all constant values. The `computeNonLinearTransForOpt`-function assembles the values together with pressure values to find the nonlinear transmissibilities and the gradient operator.

In the `getCollactionSetOPT`-function, the harmonic averaging points in Equation

(2.17) are found for the grid `G`, as well are the values for β and ω. To find the distance in Equation (2.17), we use the normal vectors $\vec{n}_\Gamma$ already calculated by `computeGeometry(G)`, and the face centroid $x_{\Gamma_i} = (x_i, y_i, z_i)$. We denote the cell centroid from which we want to find the distance by $c_i = (x_0, y_0, z_0)$. The shortest distance is then calculated through the function

$$d(x_{\Gamma_i}, c_i, \vec{n}_\Gamma) = \frac{|\vec{n}_\Gamma(x)(x_i - x_0) + \vec{n}_\Gamma(y)(y_i - y_0) + \vec{n}_\Gamma(z)(z_i - z_0)|}{\|\vec{n}_\Gamma\|_2}.$$

Further, in `getCollactionSetOPT`, the decomposition $\vec{d}_{\Gamma_{ij}} = \mathbf{t}\alpha$ is done twice for each face Γ in a `for`-loop. The optimization problem in Equation (2.19) has been solved using Matlab's `fmincon`-function. To use the `fmincon`-function in Matlab on this problem, we collect the variables in vectors and matrices so that the minimization problem is on the form

$$\min_{x \in \mathbb{R}^{m_\Gamma + 1}} c^T x \quad \text{subject to } \mathbf{A}x = \vec{d}_\Gamma^n, \quad -\mathbf{b}x \leq \vec{0},$$

where we let $c^T = [1, 1, ..., 1, \kappa]$, $x^T = [\alpha_1\sigma_1, \alpha_2\sigma_2, ..., \alpha_{m_{\Gamma_i}}\sigma_{m_{\Gamma_i}}, \gamma]$, $\mathbf{A} = [\mathbf{t}^n, \vec{0}]$, and

$$\mathbf{b} = \begin{bmatrix} \sigma_1^{-1} & \sigma_2^{-1} & ... & \sigma_{m_{\Gamma_i}}^{-1} & 0 \\ 1 & 0 & ... & 0 & 1 \\ 0 & 1 & 0 & ... & 1 \\ 0 & 0 & 1 & ... & 1 \\ \vdots & & & & \\ 0 & 0 & ... & 1 & 1 \end{bmatrix}.$$

The code can be seen in Listing 3.2. In the listing, the value `coeff` are the α coefficients and is a vector of length $m_\Gamma + 1$ to allow for the γ-value, and `t` is as in Equation (2.18). Further, `center` is the coordinates to the current cell centroid, `d`

Listing 3.2: Matlab code for the decomposition of $\vec{d}_{\Gamma_{ij}}$ using Matlabs's `fmincon`-function.

```matlab
function [t, coeff] = decomp(center,d,hap)

% input:
% center = centroid of cell
% d = scaled normal vector
% hap = harmonic averaging points for each face in current cell
```

```

N = size(hap,1);  %number of faces in current cell
coeff = zeros(N+1,1);

t = bsxfun(@minus,hap,center);
t_v = sqrt(sum(t.^2,2));   %norm of each row in t
t_u = bsxfun(@rdivide, t, t_v)';  %normalized t

d_v = norm(d,2);  %norm of facenormal
d_u = d./d_v;     %normalized facenormal

s = bsxfun(@rdivide,t_v,d_v);  %sigma
k = N/min(s)+10000;            %kappa >> N/min(s)
c = ones(N+1,1);
c(end) = k;

funk = @(x) c'*x;
A = t_u;
A(:,end+1) = 0;
lb = zeros(N+1,1);
lb(1:end-1) = -1000;
nonlinCon = @(x) con(x,s); %nonlinear constraint

coeff = fmincon(funk, coeff, [], [], A, d_u', lb, [], nonlinCon(
    coeff));
g = coeff(end);
coeff(end) = [];
coeff = coeff./s;
end
function [c,ceq] = con(x,s)
% function for nonlinear constraint.
b = zeros(length(x));
b(1,1:end-1) = s.^(-1);
b(2:end,end) = 1;
```

```
b(2:end,1:end−1)=eye(length(x)−1);
c = −b*x;
ceq = [];
end
```

is the face normal weighted with the permeability, and `hap` is the coordinates to the harmonic averaging points to each face of the cell.

As mentioned, the decomposition is done twice for each face inside a `for`-loop. This was shown necessary due to all the different variables needed for the optimization, and has had noticeably effect on the running time of the code. A solution tried for the problem, was to assemble matrices for the different variables in the loop, and do the optimization once throughout the code. However, `fmincon` was not able to solve the resulting problem satisfactory, and the implementations were returned to the original of loop-based optimization. Even if `fmincon` had been able to solve the problem satisfactory, the building of the matrices also had a long running time, and thus little improvements would have been made. However, considering that the decomposition is done before the Newton iterations, they only have to be done once for each grid. Thus source terms, boundary conditions, and wells can be changed and/or added after the tedious decomposition has been done.

After the decomposition if found, the `getCollactionSetOPT`-function calculates $\tilde{\alpha}_{i,j}$ and $\tilde{\alpha}_{j,i}$. The values are stored as matrices where each row represents one internal face. For the λ's, we notice that they consist of the values $\tilde{\alpha}_{ij,k}$, but excludes the neighbor sharing the face. Instead of making separate matrices for the λ's, active points are stored in a matrix as logicals. Each row in the active matrix represents the same internal face of the grid, and consist of true (1) or false (0) to indicate which neighboring cell is active in the sum. By representing all $\tilde{\alpha}$'s and active cells in λ as matrices, the values for μ and λ can easily be assembled to find the transmissibilies without too many matrices stored, and without using `for`-loops in each Newton iteration. In addition, a matrix representing the neighboring cells are stored, as to avoid having to find these again later on. All values are assembled with the respective pressure values in the function `computeNonLinearTransOPT`.

To be able to test the written function for the NTPFA method in Matlab, a few values have to be defined. In the following, a simple example on how to define and test the written function is described for a Cartesian grid, using MRST's AD-core.

Example 3.4.1. *First of all, the grid* `G` *must be defined as described previously. This is done in line 1 through 3 in Listing 3.3. In line 4, the function* `computeGeometry(G)` *computes the cell centroids and volumes, as well as face areas and normals. It also computes the face centroids, which are used to conpute the harmonic averaging point. Further, rock properties has to be determined. As seen in line 5, this is done by a call to the function* `rock = makeRock(G, perm, poro)`*, where* `perm` *is the permeability in millidarcys, and* `poro` *is the porosity of the rock.*

Listing 3.3: Set up of variables and model to test the NTPFA method in MRST

```
1  dims = [5, 5];
2  pdims = [1, 1];
3  G = cartGrid(dims,pdims);  %create simple cartesian grid
4  G = computeGeometry(G);  %computes cell centroids and volumes, and face areas, centroids and normals.
5  rock = makeRock(G, 100*milli*darcy, 0.002);  %add rock properties
6  fluid = initSimpleADIFluid();  %initiate fluid
7  [bc, src, W] = deal([]);  %initiate boundary conditions, source, and wells
8  bc = pside(bc, G, 'Left', 1, 'sat', [1]);
9  bc = pside(bc, G, 'Right', 0, 'sat', [1]);
10 src = addSource(src, [1,G.cells.num], [1,1], 'sat', [2]);
11
12 state0 = initResSol(G, 0, [0 1]); %initial value
13 model = PressureOilWaterModelNTPFAopt(G,rock,fluid);
14 state = incompSinglePhaseNTPFA(model, state0,'bc', bc, 'src',src,'wells',W);
```

To be able to do a full simulation, we also need fluid properties. A fluid object in MRST is again stored as a structure, this time containing the density, viscosity and compressibility of the fluids. For incompressible fluid, only the density and viscosity are necessary. In line 6, the fluid is initialized using the function `initSimpleADIFluid()`*, which creates a structure based on automatic differenti-*

ation. When no input is given, it applies standard water properties. In line 7, boundary conditions, source terms and wells are initialized as empty. At least one of them has to be prescribed a value for a force term to exist. In lines 8 − 9, pressure boundary conditions are set through a call to the function `pside(bc, G, 'side', pressure, 'sat', sat)`*. The function adds a boundary condition to a Cartesian grid on a global side indicated by* `'side'`*. For grids that are not Cartesian, the function* `addBC(bc, faces, 'type', values)` *must be used. Here,* `faces` *indicate on which external faces the boundary should be applied, and* `type` *can be either pressure or flux. The source terms* `src` *is added through the function* `addSource(src, cells, values)` *in line 10, which adds a source to a new or existing source object. The input* `cells` *indicates on which cells the source is active, and* `values` *the respective strength for each active source cell. Negative values indicate sinks, and positive values indicate producers. The well is left empty to create a simple model for testing. In line 12, the initial value for the Newton iteration is initiated, and the NTPFA model using optimization is built in line 13. The* `PressureOilWaterModelNTPFAopt` *model runs the function* `getCollactionSetOPT` *as described previously. Lastly, the problem is solved iteratively in line 14 by a call to* `incompSinglePhaseNTPFA`*. Here the function* `computeNonLinearTransOPT` *is run. The resulting structure* `state` *is displayed in Listing 3.4.*

Listing 3.4: The resulting structure `state` of Listing 3.3.

```
state =


        pressure: [25x1 double]
            flux: [60x1 double]
               s: [25x2 double]
         wellSol: [1x0 struct]
              rs: 0
              rv: 0
    upstreamFlag: [40x2 logical]
        timestep: 1
           dpRel: [25x1 double]
```

The result of the given example can be seen in Figure 3.3, which has been plotted using the function `plotCellData(G,state.pressure)`*.*

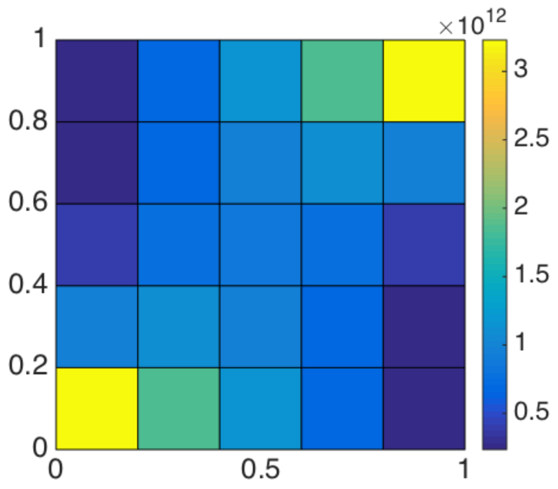


Figure 3.3: Result of the simple setup given in Listing 3.3.

Chapter 4

Results

We will look at two examples illustrating the convergence results for the NTPFAopt method in 2 and 3D, before we compare the NTPFAopt method to other methods through several test cases. To illustrate the use of the NTPFAopt method, a set of illustrative examples follows, where we investigate different permeability tensors, grid-types, and source terms.

4.1 Convergence Results

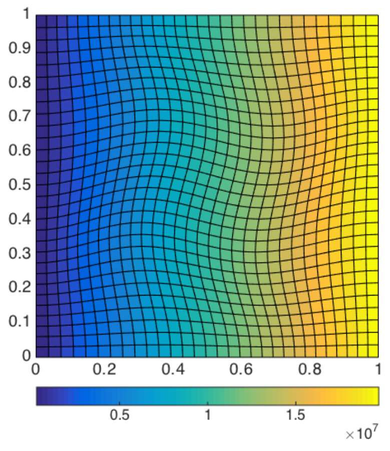
We look at a simple Cartesian grid on the unit square and the unit cube with linear pressure drop from right to left, before we investigate the convergence for the NTPFAopt method compared to the NTPFA and MPFA methods.

Example 4.1.1. *We consider a Cartesian grid of dimension* 30×30 *on the unit square. We use isotropic permeability with value* $100mD$*. We impose pressure boundary conditions to the left and right of the grid, where we let the left hand side be* $0bar$ *and the right hand side* $200bar$*. Hence, a linear pressure drop from right to left is expected. To somewhat challenge the solver, we twist the grid by a call to the MRST function* `G=twister(G)`*. The function permutes the* x *and* y

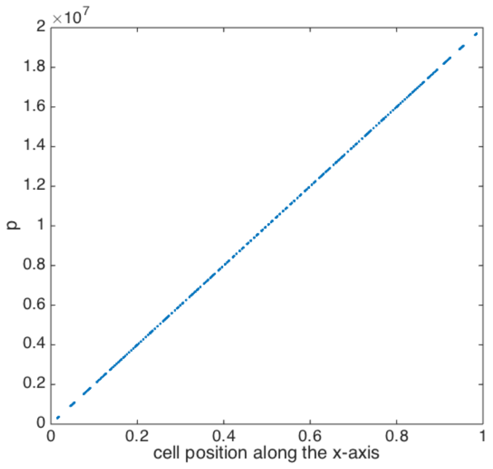
coordinates in the grid according to the mapping

$$(x_i, y_i) \to \big(x_i + f(x_i, y_i), y_i - f(x_i, y_i)\big), \quad f(x, y) = 0.03 \sin\big(\pi x\big) \sin(3\pi(y - 1/2)\big).$$

The result is shown in Figure 4.1a, and the pressure distribution along the x-axis is shown in Figure 4.1b. We see from Figure 4.1b that the pressure drop is linear from right to left as expected. The permutation of the grid did not effect the result.



(a) Resulting pressure distribution of Example 4.1.1.



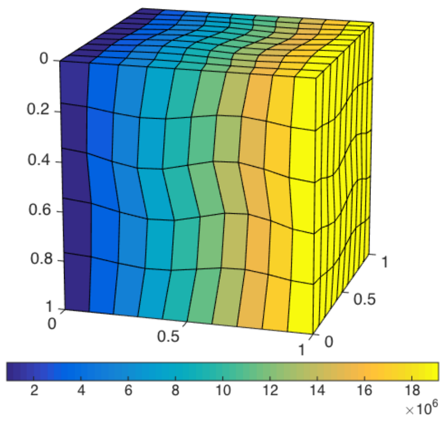
(b) The linear pressure drop along the x-axis.

Figure 4.1: The linear pressure drop of Example 4.1.1.

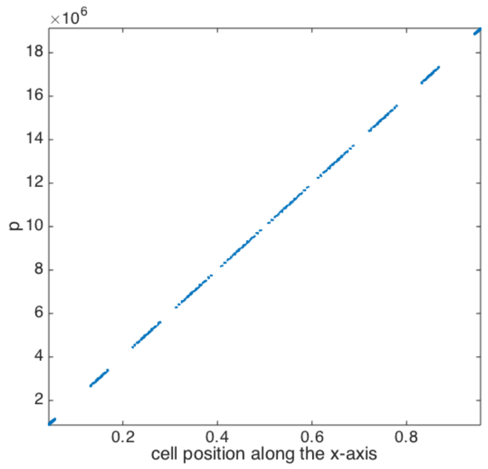
Example 4.1.2. *We still consider a linear pressure drop, but look at a grid of dimension $10 \times 10 \times 5$ on the unit cube. We use the same* `twister`*-function on the Cartesian grid, and impose the same pressure boundary conditions as in Example 4.1.1, so that the west side is $0 bar$ and the east side is $200 bar$. The result is shown in Figure 4.2, where we see that the pressure drop is linear as expected.*

We move on to a few examples showing convergence of the NTPFAopt method. The discrete relative L^2-norm has been used to estimate the numerical error, giving an error estimate on the form

$$e = \sqrt{\frac{\sum_i |\Omega_i| (p_{exact}(x_i) - p(x_i))^2}{\sum_i |\Omega_i| p_{exact}^2(x_i)}}.$$



(a) Resulting pressure distribution of Example 4.1.2.



(b) The linear pressure drop along the x-axis.

Figure 4.2: The linear pressure drop of Example 4.1.1.

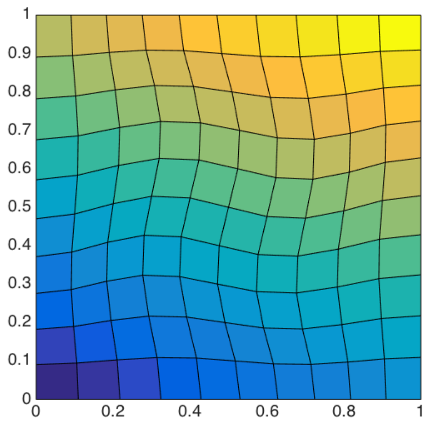
Example 4.1.3. *We look at the unit square with permeability tensor*

$$\mathcal{K} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} mD,$$

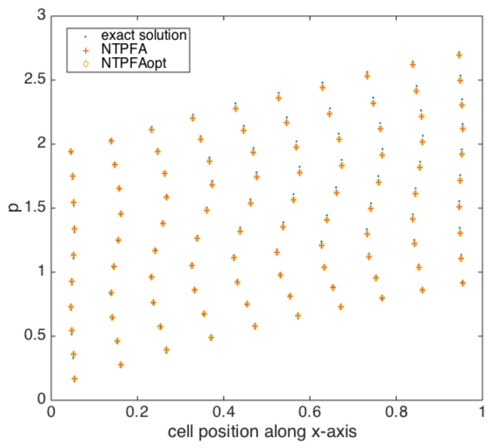
and exact solution

$$p_{exact}(x, y) = \sin(x) + 2y.$$

The simulation has been run on grids with 100, 200, 1600 *and* 6400 *cells, which has all been perturbed by the* `twister`*-function. Boundary conditions are set to the exact solution. We expect the numerical solutions to improve as the grid is refined, and as the grid is Cartesian both solvers should be able to estimate the solution closely on all grids and should align. A plot showing the exact solution on the grid with* 100 *cells is shown in Figure 4.3a, and the pressure distribution along the x-axis on the same grid for the exact solution, the solution found using NTPFA and the solution found using NTPFAopt are shown in Figure 4.3b. We see from Figure 4.3b that already on the smallest grid, the exact solution and the numerical solutions almost completely align all throughout the grid. Both numerical solutions align as expected. In Figure 4.4, a loglog-plot of the error is shown, and we see that the errors does decrease as the number of cells increase. Both numerical methods*



(a) Exact solution on the grid with 100 cells.



(b) Pressure distribution along the x-axis for the exact solution, and the numerical solutions.

Figure 4.3: Plots from Example 4.1.3.

have the same convergence rate.

For the next example we will look at the error in 3D, and compare the result to both the NTPFA and the MPFA methods.

Example 4.1.4. *We look at a Cartesian grid on the unit cube, and impose on the boundary the exact solution*

$$p(x, y, z) = \sin(\pi x) \sin\left(\pi(y + \frac{1}{2})\right) \sin\left(\pi(z + \frac{1}{3})\right) + 1$$

with permeability tensor

$$\mathcal{K} = \begin{bmatrix} 1 & 0.5 & 0 \\ 0.5 & 1 & 0.5 \\ 0 & 0.5 & 1 \end{bmatrix} mD.$$

We compare the convergence rate to the convergence rate of the NTPFA method and the MPFA method. We expect that the MPFA method will perform slightly better than the other two methods, but they should all show convergence of the error towards 0*. The exact solution on a grid with* 500 *cells is shown in Figure*

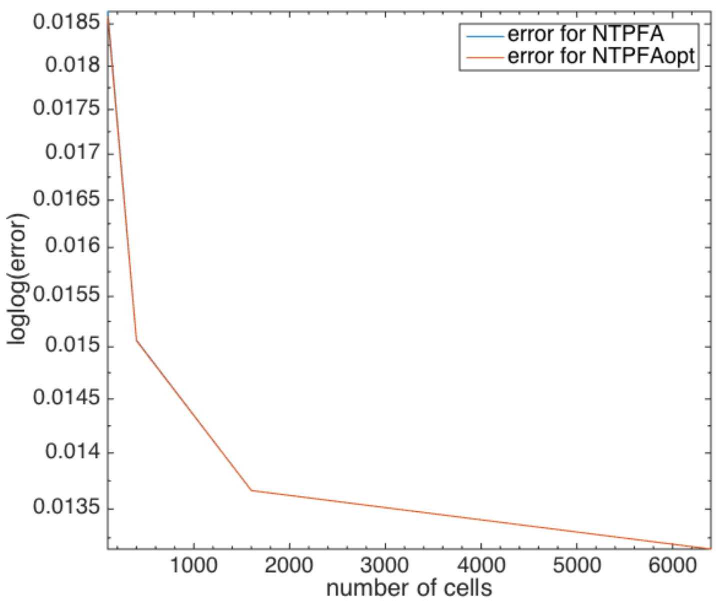
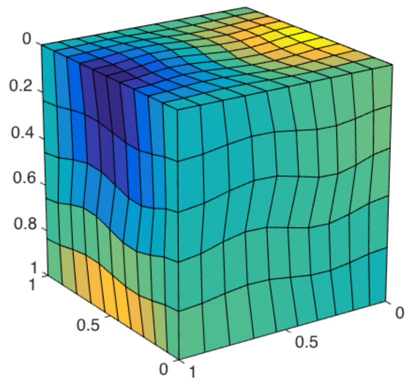


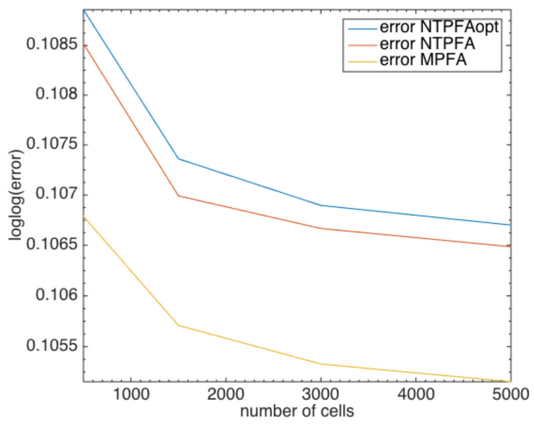
Figure 4.4: Convergence plot from Example 4.1.3. The numerical solution is found on grids with $100, 200, 1600$, and 6400 cells.

4.5a, and the error of all three methods on grids with $500, 1500, 3000,$ *and* 5000 *cells is shown in Figure 4.5b. We see that the error is decreasing, however very slightly. The MPFA method shows somewhat better results than the other two, but they are all within the* 10^{-3} *range.*

Example 4.1.5. *We again consider a linear pressure drop with the pressure on the right side set to* $0\,bar$*, and the left side* $100\,bar$*. We look at a mixed grid with cells of different shapes and sizes, as seen in Figure 4.6a. The grid has floating nodes as described in Section 3.1, and thus some faces have been split. We expect the result to be a linear pressure drop as in Example 4.1.1, but following the different cell shapes, some irregularities can occur. The grid with the result of the NTPFAopt method is seen in Figure 4.6a, and the pressure distribution along the* x*-axis is seen in Figure 4.6b. We see from Figure 4.6b that the pressure drop is linear for all the numerical methods, but the result of the MPFA method slightly outperforms the other two methods. The NTPFAopt method is shown to be the method that deviates the most from the straight line we were expecting.*

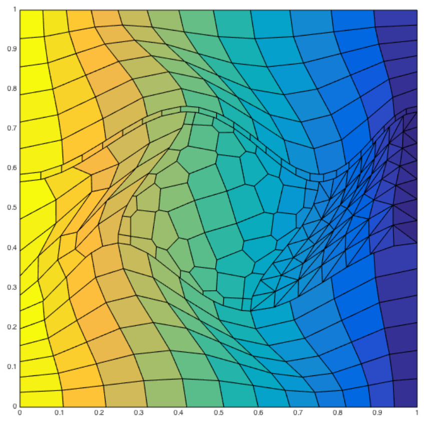


(a) Exact solution on the grid with 500 cells from Example 4.1.4

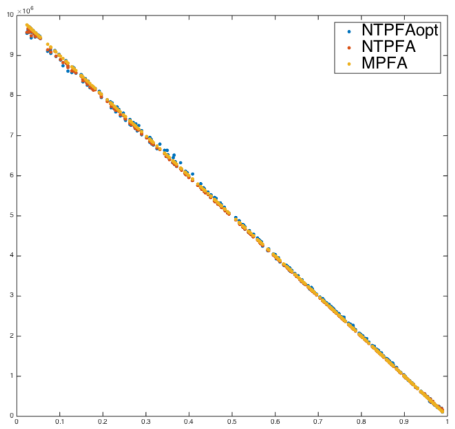


(b) Convergence plot of NTPFAopt, NTPFAlin and the MPFA methods on a grid with 500, 1500, 3000, and 5000 cells.

Figure 4.5: Exact solution of Example 4.1.4 and error plot using the MPFA, NTPFA, and NTPFAopt methods.



(a) The solution using NTPFAopt on a mixed grid with a linear pressure drop.



(b) The pressure distribution of the NTPFA, NTPFAopt, and MPFA methods on the linear pressure drop on a mixed grid

Figure 4.6: Grid and pressure distribution on the mixed grid from Example 4.1.5.

4.2 Illustrative Examples

In this section, examples using the NTPFAopt method on different grids and with different conditions are given. All solutions are compared to an established method, such as the standard TPFA-method or the MPFA-O method. We begin by looking at a simple example as to see the efficiency of the NTPFAopt method compared to the standard TPFA method on a grid that is not K-orthogonal. The first example will be followed by more complex problems to illustrate use of the NTPFAopt method.

Example 4.2.1. *We look at a skew Cartesian grid with two sinks placed equidistantly from the side edge on the bottom of the grid. The sinks are distinguished from the source by a negative value. In the center at the top, a source is placed. The grid is not K-orthogonal, and so grid orientation errors are expected to appear in the solution produced by the TPFA method. The result for both the NTPFAopt method and the TPFA method are showed in Figure 4.7, where the source/sinks are marked with a white circle. As the sources are symmetric to the sink, so should the streamlines be. The streamlines for both methods can be seen in Figure 4.8.*

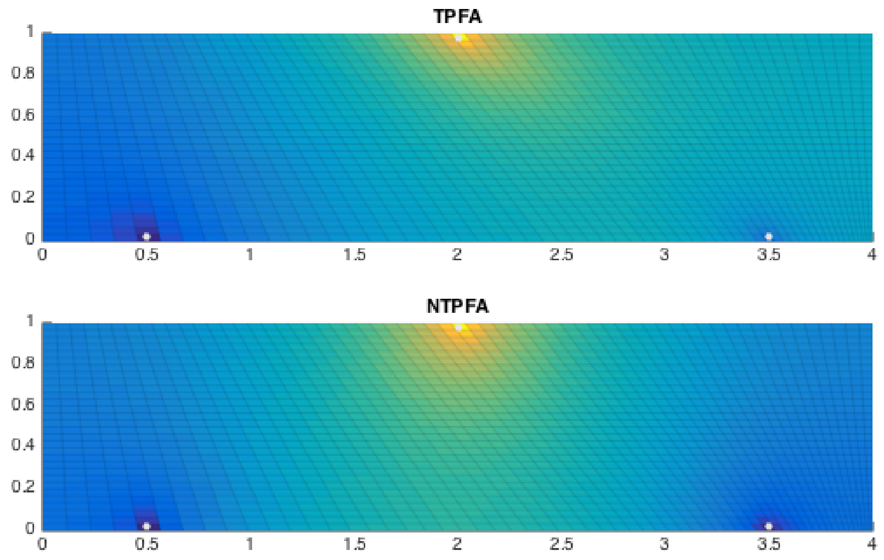


Figure 4.7: The solutions produced by the TPFA and NTPFAopt methods on a skew grid with symmetric sources and sinks, as described in Example 4.2.1.

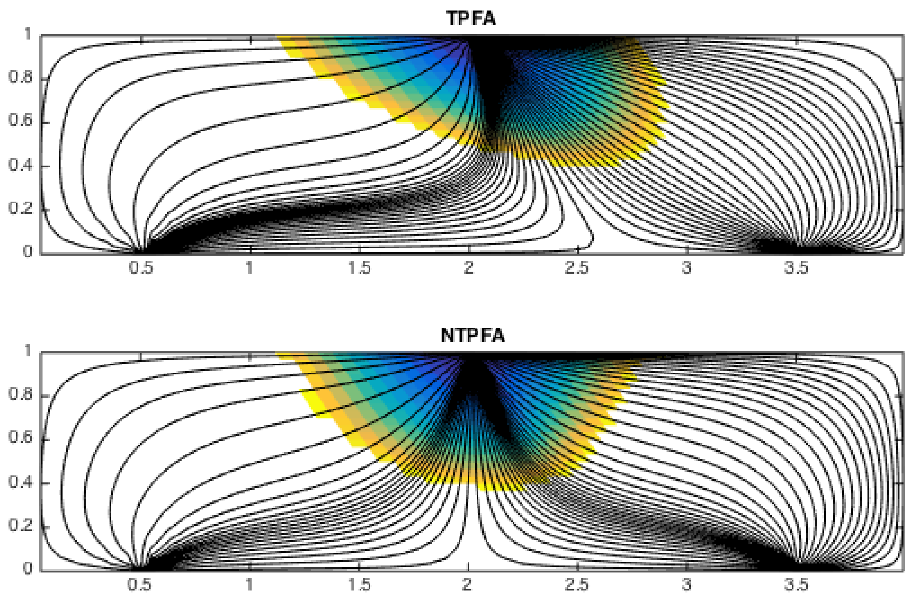


Figure 4.8: Time of flight for the skew grid in Example 4.2.1, solved with the TPFA method on top and the NTPFAopt method on the bottom.

From the pressure distribution in Figure 4.7 we see that the solution produced by the TPFA method is not symmetric, whereas the NTPFA solution shows better results. This is enhanced by the streamlines in Figure 4.8, where we see that the NTPFAopt method manages to capture the symmetry.

For the purpose of discussion, we include the results produced by the NTPFA method. They should be equal, or at least similar, to the results produced by the NTPFAopt method, but from Figure 4.9 we see that this is not the case. The method completely fails to produce the expected results on the grid of dimension 61×30 *cells.*

On a coarser grid of dimension 31×20*, the NTPFAopt method did not manage to converge to a solution at all. The NTPFA method did, but the results are still strongly asymmetrical. Further, all pressure values are negative, whereas they are positive for the TPFA and NTPFAopt methods. Thus, the NTPFA method does not manage to find a solution for this problem, whereas the NTPFAopt method*

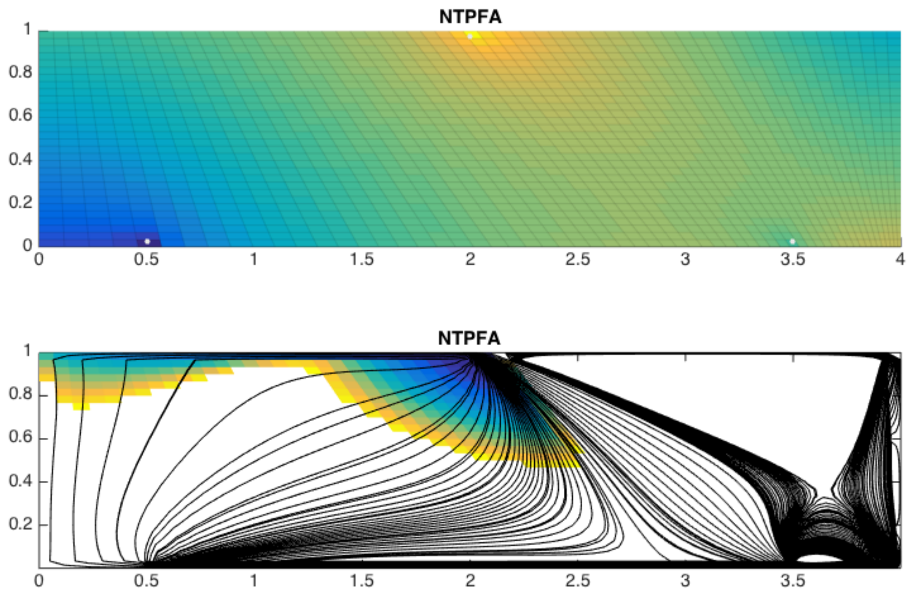
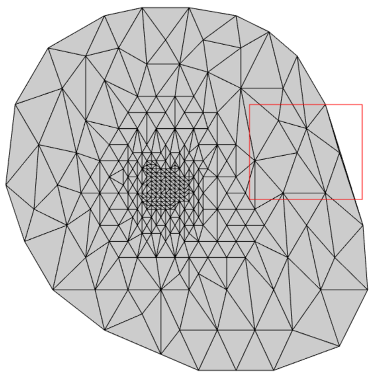


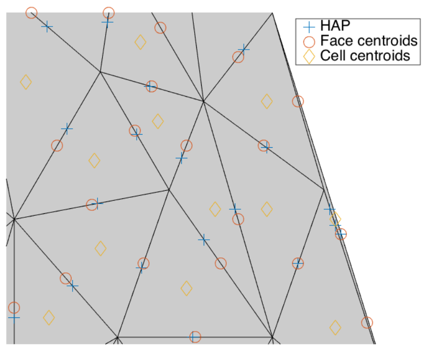
Figure 4.9: Pressure distribution and streamlines produced by the NTPFA method for the problem in Example 4.2.1.

does produce a symmetric and positive solution on a more refined grid.

Example 4.2.2. *For this example we generate a triangular grid using the Seamount demo case provided by Matlab. In Figure 4.10, the grid is shown to the left in Figure 4.10a, and the part marked in red is zoomed in on and shown to the right, in Figure 4.10b. In Figure 4.10b, the cell centroids, face centroids, and harmonic averaging points are marked with* $\Diamond$, $\circ$, *and* $+$ *respectively. We see that for this particular part of the grid, the harmonic averaging points differs greatly from the face centroids, due to the long thin cell. One of the harmonic averaging points even move to the neighboring face, leaving one face unmarked. This may cause unwanted errors in the numerical solution. Further, the long thin cells are expected to cause trouble for the NTPFA method's decomposition.*

A source is placed at the center of the grid, and the boundary pressure is set to 50*bar. The numerical results using NTPFAopt and MPFA can be seen in Figure 4.11. As expected, the NTPFA method failed to decompose all faces and hence are not dispayed as a part of the result.*


(a) The Seamount triangle grid of Example 4.2.2, with zoomed in area marked in red.



(b) Part of the Seamount grid zoomed in, with harmonic averaging points(HAP), cell centroids, and face centroids marked.

Figure 4.10: The Seamount grid of Example 4.2.2.

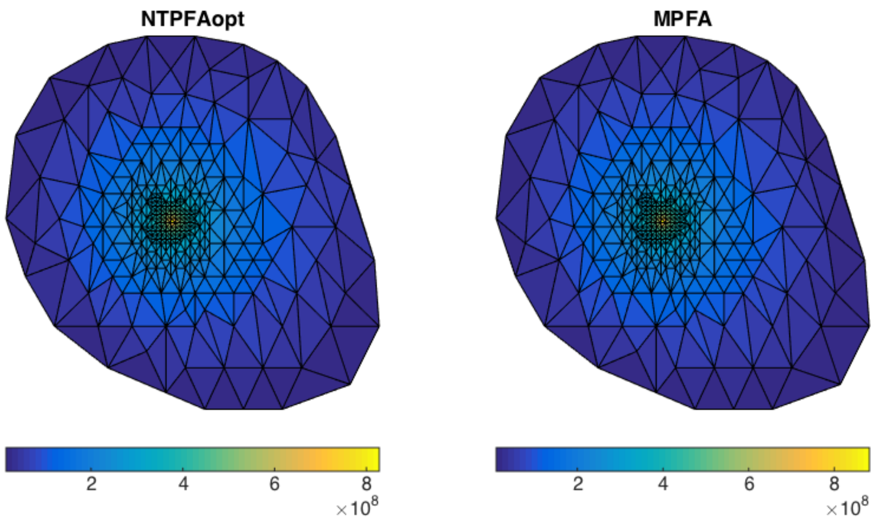


Figure 4.11: Numerical results using NTPFAopt and MPFA on the Seamount grid.

Both the NTPFAopt method and the MPFA method produce similar results. The thin cell did not cause for trouble as expected, but when investigating the decompositions there are several faces where the optimization does not satisfy all constraints given by Equation (2.19)*. These include* $\sum_i \alpha_i \geq \delta$*, and* $\sigma_i \alpha_i \geq -\gamma$*. The optimiza-*

tion is run to a precision of 10^{-20}*, and so these violations should have minimal effect. This is still something that should be kept in mind when evaluating the results for grids with small and/or narrow cells, and a different way of calculating the harmonic averaging point should be considered. A warning has been added and will be displayed if not all constraints are met so that the user is alerted of the issue, but as long as a solution to the optimization problem is found the code will complete the model.*

Example 4.2.3. *We look again at a mixed, perturbed grid on the unit square as seen in Figure 4.12, but impose a strongly anisotropic permeability tensor*

$$\mathcal{K} = \begin{bmatrix} 1 & 0 \\ 0 & 100 \end{bmatrix} mD.$$

We again consider a linear pressure drop, with the left hand side set to $100 bar$

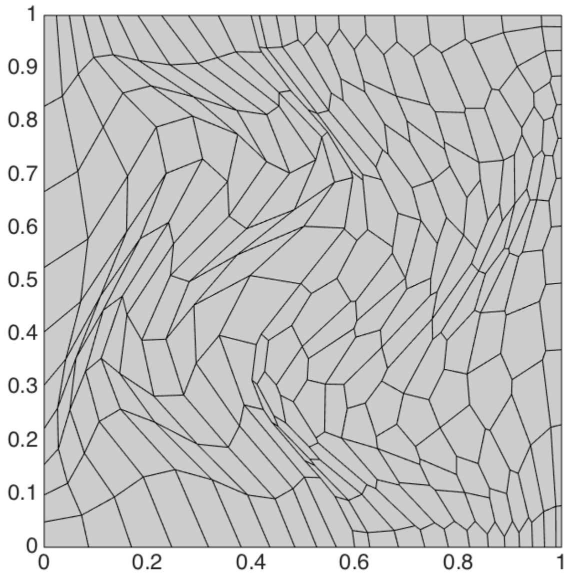
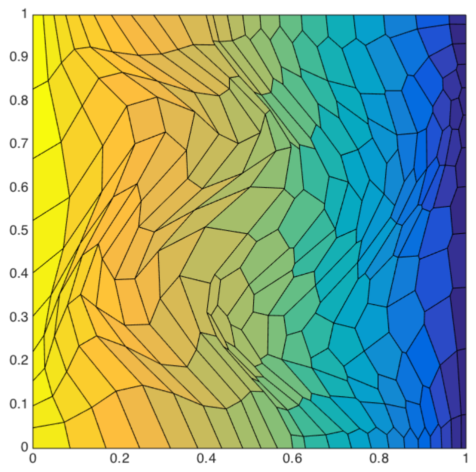
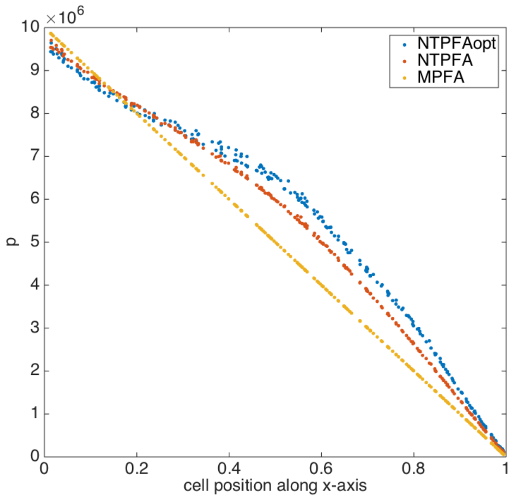


Figure 4.12: The mixed, perturbed grid of Example 4.2.3.

and the right hand side 0*. We use the NTPFAopt, NTPFA, and MPFA methods to solve the problem. The grid with dimension* 10×10 *and numerical result using NTPFAopt is shown in Figure 4.13a. In Figure 4.13b, the pressure distribution*



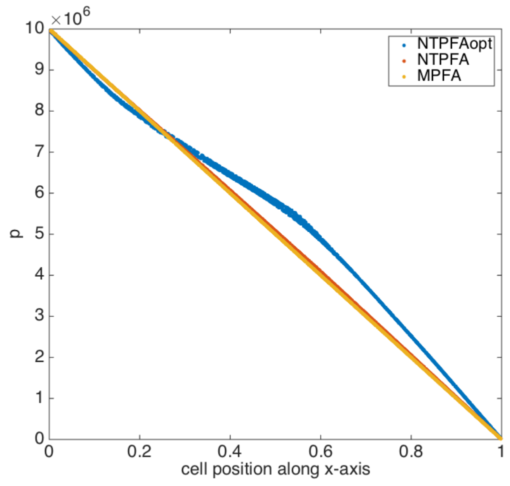
(a) The grid of dimension 10×10 from Example 4.2.3, with numerical results using NTPFAopt.



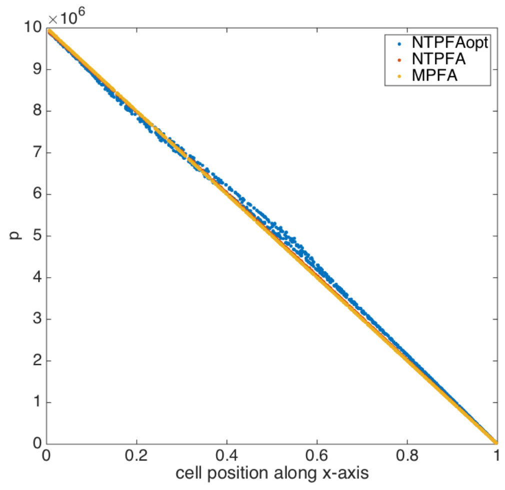
(b) Pressure distribution along the x-axis for the test case in Example 4.2.3.

Figure 4.13: The linear pressure

along the x-axis is shown for the three methods applied. We clearly see that both the NTPFA and the NTPFAopt method does not manage to capture the linear pressure drop with the anisotropic permeability tensor on a twisted, mixed grid. In Figure 4.14a, the result on a grid of dimension 40×40 is shown, and we see that the result has improved greatly for the NTPFA method, but the NTPFAopt method still shows significant errors. When investigating the harmonic averaging points, we notice again that not all of them lie on their respective faces. Some even move into the cell, something which is likely to effect further calculations. When reducing the largest element of the permeability by $1/10$th, the results improve, as seen in Figure 4.14b, where the grid is of dimension 20×20. However, the NTPFAopt method still produces inaccurate results, where both the NTPFA and MPFA methods produce satisfactory results. As the harmonic averaging points are calculated using the permeability tensor, they are bound to rely greatly on the magnitude of the tensor. This will necessarily effect the decompositions and further calculations. Again, a different method of determining the face interpolation points should be considered.



(a) Pressure distribution on a grid of dimension 30×30 from Example 4.2.3.



(b) Pressure distribution along the x-axis with a smaller permeability tensor from Example 4.2.3.

Figure 4.14: Pressure distribution along x-axis with two different permeability tensors and grid dimensions.

Chapter 5

Conclusion and Final Remarks

The NTPFA method using optimization as decomposition has been successfully implemented to run with MRST, both in 2D and 3D, for incompressible, single-phase flow. We have shown that it produces satisfactory results on Cartesian grids both with and without anisotropic permeability tensors. Further, we have shown through examples that it is consistent where the TPFA method is not, and monotone where the NTPFA method fails to give positive results. It has also been shown to find decompositions on triangular grids, where the NTPFA method failed to decompose all faces. However, the NTPFAopt method fails to give satisfactory results on mixed grids with strongly anisotropic permeability tensors, which was one of the main reasons why the method was of interest to implement. As the method of determining the face interpolation points rely greatly on the permeability tensor, we see following errors throughout the result when the face interpolation points are found outside the face.

Due to the fact that decompositions has to be done twice for each internal face, the method has a long running time regarding building the model. However, as the model is built as a preprocessing step, boundary conditions, sources, and wells can be added and/or removed after the model has been built. It would still be desirable to find a way to improve the running time of the preprocessing step, something that would be a natural next step in improving the written code.

Overall, the NTPFAopt method has not been shown to outperform the MPFA method in any of the given examples, and is, on the best, equally good. It has, on the other hand, been shown to be consistent on a grid that is not K-orthogonal, as seen in Example 4.2.1. The main issue of the method is the calculations of the face interpolation points, as these, in some of the examples presented, are located outside their respective faces. Further investigations should include improving the calculations of the face interpolation points, as this is expected to improve the final result. The NTPFAopt method does manage to find decompositions where the NTPFA method does not, and so it is of interest to further investigate and improve the method.

Bibliography

I. Aavatsmark. An Introduction to Multipoint Flux Approximation Methods for Quadrilateral Grids. *Computational Geosciences*, 6(3-4):405–432, 2002.

L. Agélas, R. Eymard, and R. Herbin. A Nine Point Finite Volume Scheme for the Simulation of Diffusion in Heterogeneous Media. *Comptes rendus de l'Académie des sciences*, Série I, Mathématique, Elsevier(347 (11-12)):673–676, 2009. doi: 10.1016/j.crma.2009.03.013.

R. L. Berge, Øystein S Klemetsdal, K.-A. Lie, H. M. Nilsen, and O. Møyner. Unstructured Gridding and Consistent Discretizations for Reservoirs With Faults and Complex Wells. *Society of Petroleum Engineers*, 2017.

F. Brezzi and M. Fortin. *Mixed and Hybrid Finite Element Methods*. Springer-Verlag New York, 1 edition, 1991. doi: 10.1007/978-1-4612-3172-1.

F. Brezzi, K. Lipnikov, and V. Simoncini. A Familiy of Mimetic Finite Difference Methods on Polygonal and Polyhedral Meshes. *Mathematical Models and Methods in Applied Sciences*, 15(10):1533, 2005. doi: 10.1142/S0218202505000832.

G. O. Brown. Henry Darcy And The Making Of A Law. *Water Resources Research*, 38(7), 2002.

C. Care. *Technology for Modelling Electrical Analogies, Engineering Practice, and The Development of Analogue Computing.* Springer-Verlag London, 2010. doi: 10.1007/978-1-84882-948-0.

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

Z. Gao and J. Wu. A Second-Order Positivity-Preserving Finite Volume Scheme for Diffusion Equations on General Meshes. *SIAM Journal on Scientific Computing*, 37(1):A420–A438, 2015.

P.-K. Heigrestad. Nonlinear Two-Point Flux Approximation Schemes. *Master's Project, TMA4500*, 2018.

W. Karush. Minima of Functions of Several Variables with Inequalities as Side Constraints. *M.Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago, Chicago, Illinois*, 1939.

E. Keilegavlen, J. M. Nordbotten, and I. Aavatsmark. Sufficient Criteria Are Necessary For Monotone Control Volume Methods. *Applied Mathematics Letters*, 22(8):1178–1180, 2009. doi: 10.1016/j.aml.2009.01.048.

H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley, Calif., 1951. University of California Press. URL `https://projecteuclid.org/euclid.bsmsp/1200500249`.

K.-A. Lie. *An Introduction to Reservoir Simulation Using MATLAB.* SINTEF ICT, Departement of Applied Mathematics, Oslo, Norway, 1. edition, 2016.

R. D. Neidinger. Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming. *SIAM Review*, 52(3):545–563, 2010. doi: 10.1137/080743627.

K. Nikitin, K. Terekhov, and Y. Vassilevski. A Monotone Nonlinear Finite Volume Method for Diffusion Equations and Multiphase Flows. *Computational Geosciences*, 18(3):311–324, 2014. doi: 10.1007/s10596-013-9387-6.

J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer-Verlag, New York, 2 edition, 2006. doi: 10.1007/978-0-387-40065-5.

J. M. Nordbotten and M. A. Celia. *Geological Storage of CO2.* Wiley, 1. edition, 2012.

J. M. Nordbotten, I. Aavatsmark, and G. T. Eigestad. Monotonicity of Control Volume Methods. *Numerische Mathematik*, 106(2):255–288, 2007. doi: 10.1007/s00211-006-0060-z.

C. L. Potier. A Nonlinear Finite Volume Scheme Satisfying Maximum and Minimum Principles for Diffusion Operators. *International Journal on Finite Volumes*, pages 1–20, 2009. URL `https://hal.archives-ouvertes.fr/hal-01116968`.

M. H. Prottern and H. F. Weinberger. *Maximum Principles in Differential Equations.* Springer-Verlag New York. Inc, 1984. doi: 107.1007/978-1-4612-5282-5.

M. Schneider, B. Flemisch, K. T. R Helmig, and H. Tchelepi. Monotone Nonlinear Finite-Volume Method for Challenging Grids. *Stuttgart Research Centre for Simulation Technology*, 2017.

X.-H. Wu and R. Parashkevov. Effect of Grid Deviation on Flow Solutions. *SPE journal*, 14(1):67–77, 2009. doi: 10.2118/92868-PA.