Duy Duc Khuat

Improved Optimization Methods For Adjoint-Based Training Of Reduced-Order Models

June 2022

Master's thesis
2022
Duy Duc Khuat

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Improved Optimization Methods For Adjoint-Based Training Of Reduced-Order Models

Duy Duc Khuat

MTFYMA
Submission date: June 2022
Supervisor: Knut-Andreas Lie
Co-supervisor: Stein Krogstad, Øystein Andersson Klemetsdal

Norwegian University of Science and Technology
Department of Mathematical Sciences

NTNU

Kunnskap for en bedre verden

Improved Optimization Methods For Adjoint-Based Training Of Reduced-Order Models

Duy Duc Khuat,
supervised by Knut-Andreas Lie, Stein Krogstad
and Øystein Andersson Klemetsdal

June 19, 2022

Master's Thesis
in collaboration with
Sintef Digital Mathematics and Cybernetics

Abstract

Multiphase porous media flow simulation is through its wide range of applications of great interest and importance. However, computational cost has been often the limiting factor in using computerized methods to perform simulations. Therefore, calibration of reduced-order models have become an increasingly popular alternative to full forward simulations as a way to obtain faster and yet reliable forecasts [6, 18, 15, 28]. With the more efficient automatic differentiation based simulation [31] and adjoint-based optimization [3, 16], we find reduced-order model emerging from simple parameter optimization processes beneficial. To this end we consider for reservoir models non-linear least square optimization problems which penalize the misfit of the output of the reference fine-grid model with the output of the coarse model. Gauss–Newton methods are available for this set of problems and have a fast convergence rate which is crucial to reduce the number of expensive full forward simulations. We provide a comprehensive analysis on all mentioned aspects. The calibration process is performed on various notorious data sets including synthetic benchmarks model SPE10 [4], the Norne field model [32] and the Egg model [13] and the results are generated in MRST [26].

Norsk. Simulasjon av flerfasestrømninger i porøse medier er et viktig felt grunnet dets mange anvendelsesområder. Til tross for dette har beregningskostnadene ofte vært en begrensende faktor i bruken av numeriske metoder for å utføre slike simulasjoner. Dermed har kalibrering av modeller med redusert orden blitt et stadig mer populært alternativ til fullordens framover simulasjoner for å oppnå raskere prediksjoner, som fortsatt er pålitelige [6, 18, 15, 28]. Med den mer effektive automatiske differensieringbaserte simulasjonen [31] og adjungertbasert optimering [3, 16] finner vi at reduserte ordens modeller, som kommer fra optimeringsprosesser med enkle parametre, er gunstige. Til dette formålet ser vi på reservoarmodeller basert på ikke-lineære minste kvadraters optimeringsproblemer som straffer den grove modellens avvik fra den finere referanse modellen. Gauss-Newton-metoder er tilgjengelige for denne typen problemer og de konvergerer raskt, noe som er avgjørende for å redusere antall kostbare helordens framover simulasjoner. Denne oppgaven gir en grundig analyse av de nevnte aspektene.

Contents

Preface

The Master's Thesis is submitted in fulfillment of the requirements for the degree of Master of Science (MSc) Mathematics at the Rhine–Westphalia Technical University Aachen (RWTH) in Aachen, Germany. The thesis is submitted to the Norwegian University of Science and Technology and written in collaboration with Sintef Digital Mathematics and Cybernetics.

Acknowledgements

First in foremost I thank the Norwegian University of Science and Technology (NTNU), Trondheim, for providing me with such a great opportunity to write my Master's thesis in Norway and finally at SINTEF Digital. This huge gratefulness is shared in at least the same amount with my supervisors Knut-Andreas Lie, Stein Krogstad and Øystein Andersson Klemetsdal who have, despite their already overflowing agenda, agreed on guiding me through a complex topic and provided me with continuous and constructive feedback in repeated zoom-meetings. Working in their highly professional team has steadily encouraged and motivated me to enhance my writing and research.
Further, I thank my family for their support.

Duc Khuat
Trondheim, Norway
June 2022

Introduction

As multiphase porous media flow, we understand the flow of multiple fluids or gases through a rigid porous medium. This field has been mainly pioneered by Henry Darcy by his empirical derivations of the two-dimensional laws for porous media flow in water filtration processes in sandboxes from 1856. Through experiments he found out that the flow rate per cross-sectional area is proportional to the hydraulic head of the water and inverse proportional to the length of the cylinder. Since most of space is blocked by sand grains in his experiments, Darcy's law does not provide the velocity as it is often portrait but it provides the *apparent macroscopic* velocity obtained by averaging the flux inside of representative elementary volumes (REV).

Simulation of multiphase porous media flow has great importance due to it vast range of applications in chemistry, biology, medicine, and earth sciences, and many more. Reservoir simulation that belongs to the latter has caught arguably the most interest from not only engineers and scientist but also politicians and economists who recognized the importance of groundwater flows and a variety of processes of oil recovery [14]. Besides that, current hot topics which also involve subsurface porous media flow are processes related to CO_2-sequestration (see e.g., [27]) and geothermal power plants (see e.g., [33]). What connects the porous media flow in different fields mathematically is the base form of the governing equations, that is, conservation of mass for fluid phases or chemical species and Darcy's law for the superficial velocity of fluid flow through porous media. The non-linear, time-dependent partial differential equations

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

are based on the same idea of averaging the microscopic flow in a representative elementary volume to obtain the apparent macroscopic flow to describe the mass conservation of a fluid with density ρ filling a volume fraction ϕ of the bulk volume. The resulting superficial flow velocity $\vec{v}$ is driven by pressure differences ∇p and gravity g, resulting in the second partial differential equation. These, and more advanced model equations are discussed in more detail in Section 1.1. The averaging method makes the fluid simulation feasible, since the computational cost of simulating the microscopic flow in a whole reservoir model is sky rocketing. However, the microscopic flow simulation is used to find the right parameters for upscaling. In fact, to obtain a whole life-cycle reservoir simulation, we start to derive parameters from finer flow simulations both in time and space.

The general scheme for computing a numerical solution is based on the finite-volume method for spatial discretization and implicit-Euler for temporal discretization, as discussed in more detail in Section 1.2. This gives a large system of discrete nonlinear equations, which is typically solved with Newton–Raphson (see Section 1.3). Deriving the necessary linearizations and assembling the resulting Jacobi matrix used to compute the Newton increments can be very complex and error-

prone, but this process can be greatly simplified through the use of automatic differentiation, which is introduced in Section 1.4.

Simulating a highly detailed reservoir model can be computationally expensive, especially if multiple forward simulations are necessary, e.g., as part of a model calibration, uncertainty quantification, or production optimization process. Another important consideration for computational cost reduction is the use of reduced-order models. Many of those are obtained through deep learning and machine learning methods. Some of them are data-based machine learning algorithms which focus on finding relations between input and output of specific data sets. The caveats of those methods are that the resulting reduced-order models will likely not represent physical attributes or laws. A simpler approach to this end is to turn the calibration of a reduced-order model problem into a simple parameter constrained optimization problem of a parabolic convex function. In fact, we use a simple non-linear least squares function as the objective function and restrict the solution space to parameter values that lead to states that in turn represent a full forward simulation. This translation has the benefit that the calibration set up is simple. Most reservoir simulators are well set to perform efficient gradient-based optimization and provide therefore the required framework. In the *MATLAB Reservoir Simulation Toolbox* (MRST), the key technology which provides the gradient to the optimizer is based on the KKT-conditions of the Lagrange relaxation of the constrained optimization problem. These conditions give rise to the adjoint equations which can be implemented additionally in parallel to the simulation evaluation process and enable an AD-based efficient computation of the gradient.

Given the gradient, the choice of the right optimizer is then a modular designing task. Iteration methods usually split into two sub tasks: determine the search direction and computing the step length. Often, general optimizers are based on a quadratic approximation of the objective function. The choice of the second-order coefficient matrix is the name giving process. Davidon, in 1959, has pioneered the class of quasi-Newton methods when he was in desperate search for a more stable and efficient optimization method [7]. He generalized the secant method to arbitrary dimension and derived through a minimization sub problem a corresponding choice for the second-order coefficient matrix of the quadratic approximation. Based on his method Broyden, Fletcher, Goldfarb and Shannon [2, 9, 11, 35] in 1970 independently established the BFGS second-order matrix choice, which remains until now the backbone of the most promising optimizers for general purposes. In particular, a popular choice in MRST is BFGS with line search, due to its robustness, and self-correcting properties. The main drawback of BFGS is the relatively low convergence rate compared to the quadratic convergence rate of the Newton method. The calibration process of reduced-order models gives rise to employ the Gauss-Newton method which under certain condition convinces with a quadratic convergence rate but it pays with an increasing space complexity which scales with the number of residual components. This gives rise for a closer analysis. Based on numerical experiments we search for improved optimization methods.

Chapter 1 and 2 provide the mathematical foundation. This subdivides into the following tasks which are covered in Chapter 1: We introduce the reader to the single and multiphase porous media flow. Then we illustrate the spatial and temporal discretization and round up with a complete simulation loop for the single phase case. We follow up with a section on two key technologies to enhance stability

and performance for simulation: *automatic differentiation* and *adjoint equations*. At the end we give a brief overview on reduced-order model calibration.

Chapter 2 describes the structures of iteration optimizers. The focus in this presentation lies in an efficient implementation, more precisely, we ask how do we design an optimizer with few function evaluations of the objective function.

Chapter 3 is dedicated to present calibration processes of reduced-order models in MRST. To this end we discuss a simple code snippet for a small Cartesian example of multiphase porous media flow and highlight the performance of the key technologies mentioned in Section 1.6. We describe the physical models in MRST in detail and present at the end the results obtained from the numerical experiments. We conclude with further work and discuss amongst other how to generalize performance in a broader spectrum of application.

1 Porous Media Fluid Flow

In this first chapter we want to provide a mathematical introduction into porous media flow. This chapter is inspired by Lie's book [22, Chapter 4-8]. Porous media flow has been studied also in more detail in [37] and the references therein indicate the roots of the journey, beginning from the two-dimensional empirical derivations of water filtration processes in sandboxes from 1856 from Henry Darcy up to more general extensions, in particular, the development of Darcy's law to the full three-dimensional case. Here, we reduce the presentation to practical purposes such that at the end the reader can understand the general structure of industry-standard porous media flow simulators and optimizers. The topics are chosen as general as possible to assess a variety of problems and also to cover a wide range of implementations for porous media fluid flow solvers.

The chapter starts with the governing equations of porous media flow and a derivation of the parameters in Section 1.1. After having established the physical background, we are concerned with the numerical methods to compute an approximate solution of our fluid flow system in Section 1.2. It is a finite-volume method used for time-independent non-linear partial differential equation in 3D space.

It is based on the idea to connect differential operators to the specific underlying unstructured grid system. We are considering a space discretization through a grid structure and the governing variables (here mainly pressure) are averaged over the grid cells.

Section 1.3 covers the temporal discretization of our system of equations with implicit-Euler followed by Newton–Raphson to determine the solution of the arising system of non-linear equations. The section gives a total overview on a finite-volume method-based simulation for porous media flow where we explicitly set up all required equations and explain the simulation loop.

Section 1.4 explains the principles of forward automatic differentiation. We discuss further the reason why AD is particularly useful for high dimensional reservoir simulation which includes complex fluid physics.

In Section 1.5 we discuss the method of adjoint equations which is used to obtain the gradient of a parameter constrained optimization problem. Often, in calibration optimization processes for full scale reservoir models the number of parameters is much larger than the number cells. The latter is the size of the systems emerging from the adjoint equations. Noticing this fact enables a simple way to enhance performance of gradient-based optimization which we want to discuss here as well.

1.1 Physical Model

We understand porous media as a solid matrix with interconnected void spaces (pores). The solid matrix is not permeable for the fluid but the void pore space allows to transmit and store fluids. Depending on the application, we have different scales on which we simulate porous media flow. For instance, *microscopic* models represent

the void space between individual particles (e.g., grains of a rock/molecules/ion lattices) and are used to provide effective properties of the medium like permeability, porosity, electrical, and elastic properties to models on larger scales. The next level is *mesoscopic* models. These represent heterogeneous structures such as internal laminations, membrane layering or sedimentation (in geological reservoir models this also includes impermeable faults, shale layers and the whole variety of phenomenas). Finally, overviewing the whole system is the task of the *macroscopic* model which we are mainly interested in.

For reservoir simulation there is also a strong focus on different parameters describing various physical interactions and balances. In particular, complex molecular forces such as *capillary forces* and *surface tension* and many more lead to a high number of microscopic variations which cannot be transferred in the same resolution to the macroscopic model. For instance, if we flood water into a porous rock filled with hydrocarbon one is interested in many sub questions. When is the entry point for a certain size of a pore cell? This depends on the capillary forces which in turn is depending on the saturation of the fluid and, in particular, from its *irreducible saturation limits* . On the other hand those quantities depend on the rock type and so on. Then it depends on the wettability (see Section 1.1.2) of the fluids whether we have a *drainage* or an *imbibition* process. In order to provide precise predictions of oil production for certain well controls, this microscopic physical processes are essential but must be heavily simplified to remain feasible in macroscopic full scale models.

We restrict ourselves in this work to cover the relative minimum for above considerations. Throughout this section we will make standard assumptions which simplify the actual physical processes crucially.

1.1.1 Single-Phase Flow

We first consider single-phase flow through porous media. The governing equations for conservation of mass and momentum are

$$\frac{\partial(\phi\rho)}{\partial t} + \nabla \cdot (\rho\vec{v}) = \rho q \tag{1.1}$$

and

$$\vec{v} = -\frac{\mathbf{K}}{\mu}(\nabla p - g\rho\nabla z), \tag{1.2}$$

where ρ is the density of the fluid, ϕ is the rock porosity, $\vec{v}$ is the macroscopic Darcy velocity, and q denotes fluid sources and sinks, i.e., outflow and inflow of fluids per volume at certain locations.

Darcy's law here is the porous media flow equivalent to models of regular diffusion processes of gas or fluids (Fick's law) and also electrical potential (Ohm's law). The driving forces here are gravity and pressure differences. The permeability $\mathbf{K}$ is tied to the solid medium and it is derived from the microscopic model. The viscosity μ, sometimes also replaced by the inverse of the mobility coefficient of a fluid, emerges as a property from the fluid.

Last but not least, neglecting the porosity (which results from the solid medium), these two equations result from the Navier–Stokes equations by neglecting the inertial forces. In regular reservoir models, the inertial forces will be neglected. Reservoirs in the Nowegian Sea extend over 10–100 km in each direction (see e.g., [21]).

Since we have more unknowns than equations, we need to establish further relations. To link pressure to density, we consider the relations between temperature and pressure. Let V be the volume for a fixed number of particles. Then V is a function of p and the temperature T. From this follows

$$\frac{\mathrm{d}V}{V} = \frac{1}{V}\left(\frac{\partial V}{\partial p}\right)\mathrm{d}p + \frac{1}{V}\left(\frac{\partial V}{\partial T}\right)\mathrm{d}T.$$

Since ρV is constant for a fixed number of particles, we obtain $V\mathrm{d}\rho + (\mathrm{d}V)\rho = 0$, and therefore

$$\frac{\mathrm{d}\rho}{\rho} = \frac{1}{\rho}\left(\frac{\partial \rho}{\partial p}\right)\mathrm{d}p + \frac{1}{\rho}\left(\frac{\partial \rho}{\partial T}\right)\mathrm{d}T = c_f \mathrm{d}p + \alpha_f \mathrm{d}T$$

relating the density difference with the pressure and temperature difference, where c_f is the isothermal compressibility and α_f the termal expansion coefficient. If we assume that the temperature is constant, which is the case for many subsurface systems, the equation simplifies to

$$c_f = \frac{1}{\rho}\frac{\mathrm{d}\rho}{\mathrm{d}p}. \tag{1.3}$$

The fluid compressibility $c_f(p)$ relates ρ to the pressure. Combining all our equations, we find the following relation for fluid pressure

$$c_t \phi \rho \frac{\partial p}{\partial t} - \nabla \cdot \left[\frac{\rho \mathbf{K}}{\mu}(\nabla p - g\rho\nabla z)\right] = \rho q \tag{1.4}$$

where $c_t = c_r + c_f$ and $c_r = \frac{1}{\phi}\frac{\mathrm{d}\phi}{\mathrm{d}p}$ is the rock compressibility, a coefficient that relates the porosity to the pressure p. Indeed, we have $c_t\phi\rho\frac{\partial p}{\partial t} = \frac{\partial(\phi\rho)}{\partial t}$ by inserting the identities

$$\rho\phi\mathrm{d}p c_t = \rho\mathrm{d}\phi + \phi\mathrm{d}\rho, \tag{1.5}$$

$$\frac{\partial(\phi\rho)}{\partial t} = \frac{\partial\phi}{\partial t}\rho + \phi\frac{\partial\rho}{\partial t} \tag{1.6}$$

into Eq. (1.1). Let us draw some theoretical conclusions from Eq. (1.4).

For instance, we can observe that if we consider incompressible fluid and rock, that is, $c_t = 0$, we are left with

$$\nabla \cdot \mathbf{K}\nabla\Phi = q, \tag{1.7}$$

where $\Phi = p - g\rho z$ is the fluid potential. This is simply the elliptic partial differential equation known as Poisson's equation.

Remark 1.1 (Constant Fluid Compressibility):
Assume that $c_f = c_t$ is constant and porosity ϕ and viscoscity μ are independent from pressure. Then the equation

$$c_t \phi \rho \frac{\partial p}{\partial t} - \nabla \cdot \left[\frac{\rho \mathbf{K}}{\mu} (\nabla p - g\rho \nabla z) \right] = \rho q$$

can equivalently be expressed as

$$\frac{\partial \rho}{\partial t} - \frac{1}{\mu \phi c_f} \nabla \cdot \left(\mathbf{K} \nabla \rho - c_f g \rho^2 \mathbf{K} \nabla z \right) = \frac{\rho q}{\phi}$$

that is the heat equation for $q = 0$. Observe that if c_f is large then

Proof. With constant compressibility, the differential Eq. (1.3) has the solution

$$\rho(p) = \rho_0 e^{\frac{1}{c_f}(p - p_0)}. \tag{1.8}$$

Also we have the equation

$$c_f \rho \mathrm{d}p = \mathrm{d}\rho,$$

where $\mathrm{d}p$ is a small change in pressure and $\mathrm{d}\rho$ a small change in density, respectively. Dividing both sides by a partial derivative and replacing the difference operator d with a partial derivative, we obtain

$$\nabla p = (c_f \rho)^{-1} \nabla \rho, \tag{1.9}$$

$$\frac{\partial p}{\partial t} = (c_f \rho)^{-1} \frac{\rho p}{\partial t}. \tag{1.10}$$

Inserting those identities to eliminate the pressure, we get

$$c_f \phi \rho (\rho c_f)^{-1} \frac{\partial \rho}{\partial t} - \nabla \cdot \frac{\rho K}{\mu} \left((\rho c_f)^{-1} \nabla \rho - g \rho \nabla z \right) = \rho q, \tag{1.11}$$

$$\phi \frac{\partial \rho}{\partial t} - \frac{1}{\mu c_f} \nabla \cdot K \left(\nabla \rho - c_f g \rho^2 \nabla z \right) = \rho q. \tag{1.12}$$

□

Constant fluid compressibility is reasonable to assume when no large quantities of gas are dissolved. A more difficult task is the assumption for the rock compressibility c_r. If the rock of the reservoir is non-rigid then c_r is simply zero and porosity must not be governed in our equation. Compressibility can have a significant variation, for instance, as evidences in the Ekofisk area in the North Sea show [22, Section 2.4]. The assumption of a constant rock compressibility leads to

$$\phi(p) = \phi_0 \exp[c_r (p - p_0)].$$

However, a linear relation between ϕ and p is often used for simplified models and computational wise it might make sense to consider a piecewise continuous spatial function for $\phi(p)$. An important observation is that the compressibilities c_f and c_r are required to establish these relations between $\rho(p)$ and $\phi(p)$, respectively, enabling in first place that our system of partial differential equations (1.28) is solvable.

Permeability: Permeability is a property for porous media which measures the ability to transmit a single fluid when the void space is completely filled with it. In other words it is a measure for the connectivity of pores in the subsurface system. Permeability is measured in millidarcies (mD). Commonly, the permeability ranges from 100 to 500mD for petroleum reservoir rock [25]. Darcy's law (1.2) simplifies to the equation

$$\vec{v} = -\frac{K}{\mu}\nabla\Phi, \tag{1.13}$$

where $\nabla\Phi$ is pressure or the potential gradient, that is, $\Phi = p - g\rho z$. K is the proportionality factor between the flow rate $\vec{v}$ and $\nabla\Phi/\mu$. In general, $\mathbf{K}$ is a tensor of the form

$$\mathbf{K} = \begin{bmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz} \end{bmatrix}$$

with unit D $[\mathrm{m}^2]$ = [D] which stands for "darcy" and $1\mathrm{D} \approx 0.987 \cdot 10^{-12}\mathrm{m}^2$. Here, K_{ij} is the coefficient relating the pressure drop in direction j to the flux in direction i. That means that if K_{ij} is large, a large pressure drop in direction j implies a large contribution to the flux in direction i. If the permeability can be represented as a scalar, we call the permeability isotropic. However, in the anisotropic case, where we need a full tensor to represent permeability, the matrix must be symmetric due to Onsager principle of reciprocal relations and positive definite because the eigenvalues must be positive, which is equivalent to the fact that the flux follows the direction of the pressure drop

Well model: We use the standard Peaceman well model that is just a linear law for the source term q_c and the pressure difference at the well connection p_W and in the reservoir at the well connection p. Let J be the proportionality constant, also called *well index*. The relation includes the viscosity and reads

$$q = \frac{J}{\mu}(p_W - p).$$

The well has also a non-trivial pressure distribution. For simplicity we assume that it is always in hydrostatic balance, that is,

$$p_W = p_{\mathrm{bhp}} + g\Delta z_w \rho(p_{\mathrm{bhp}},$$

where p_{bhp} is referred to as the bottom hole pressure of the well. The bottom hole pressure is usually governed by a giving control function and is therefore the reference point of the well.

1.1.2 Multiphase Flow

Before stating the mathematical flow equations, we first introduce some additional physical quantities that are necessary to describe multiphase flow.

Saturation: In multiphase flow we are distinguishing not only between different densities for the fluids but also consider the fraction of pore volume occupied by the fluid α. We call this fraction s_α and we naturally require

$$\sum_\alpha s_\alpha = 1.$$

Saturation is a crucial governing variable when we simulate flow of immiscible fluids. To this end consider the following example. The accumulation of hydrocarbons in an aqueous environment usually starts with porous rock being fully filled with water. By the time hydrocarbons migrate into the reservoir displacing water until the water saturation reduces to approximately 5% to 40 %. At this state, water cannot flow and forms small droplets through molecular forces. Then the water immersed in the hydrocarbons is in an immobile state. The limit value for water to reach from the mobile state (funicular state) to the immobile state (pendular state) is called irreducible water saturation s_{wir}. Vice versa hydrocarbons also have limit value called residual oil saturation s_{or} which is typically higher than s_{wir} meaning that oil is immobile in earlier stages of the displacement process through water injection, that is, s_{or} is in the range of 10% to 50%. The thresholds are due to surface tension and so-called *wettabilities*, a quantity that measures the affinity to the solid medium over the competing other fluid. For instance a higher wettability of water than oil means that the water remains more in contact with the solid medium. We call such a porous media water-wet and the opposite is called oil-wet porous media. The former of those two types is more common and hence this explains also why s_{or} is usually the smaller value. For instance, for perfect water-wet porous media this means that the only water has contact to the solid medium and the surface tension between the two fluids does not stop the water flow anymore.

On the other hand, a perfect oil-wet porous media would imply that water forms a perfect spherical droplet at the solid medium. Then the droplets are trapped in this state and immobile.

Capillary pressure: Is not explicitly part of the equations but it relates the pressure of the different phases with the saturation. Capillary pressure is the positive pressure difference in a two phase flow. Since the non-wetting fluid pressure is always bigger than the wetting fluid pressure, the capillary pressure p_c can be written as $p_c = p_n - p_w$ where p_n is the non-wetting fluid pressure and p_w is the wetting fluid pressure.

We can see porous media as an assortment of capillary tubes where the pore size is the diameter of the tubes. Now for small pore size the capillary pressure is large and will play a main role in the fluid distribution.

Consider an upward migrating hydrocarbon phase into a porous rock media filled with water. To enter the reservoir, the buoyancy force must exceed the capillary pressure that is required to enter the system. At first, the hydrocarbon enters the widest tube, since it has the lowest capillary pressure. The more hydrocarbon enters the system, the lower these capillary pressures become and and the lower the water saturation becomes. There is a relation between capillary pressures and the water saturation. With a low water saturation, the capillary pressure is high and hence the hydrocarbon can not enter pores and stays immobile. Flooding a reservoir with water to push hydrocarbon to the wells leaves immobile parts behind in the reservoir. For the relation between the capillary pressure p_c and the saturation we usually use

the tabulated *Leverett* **J**-*function* [20]

$$J(s_w) = \frac{p_c}{\sigma \cos\theta}\sqrt{\frac{K}{\phi}}$$

where s_w is the saturation of the wetting fluid and σ is the surface tension to the other fluid and θ is the contact angle of the wetting fluid. We can then relate the two immiscible fluids with $p_n - p_w = p_c(s_w)$.

Relative permeability: The absolute permeability $\mathbf{K}$ from the previous section is an immanent rock property. In a multiphase flow, each fluid α will experience an effective permeability $\mathbf{K}_\alpha^e$ that is lower than the absolute permeability: adding more phases to the fluid flow will result in more obstacles that present resistance to flow. The effective mobility of a fluid therefore decreases in the presence of other fluid phases in the pore space. Through the interfacial tensions between the immiscible fluid, the sum of effective permeabilities for all fluids must be smaller or equal to the absolute permeability. This gives rise to the *relative permeability* $k_\alpha = \mathbf{K}_\alpha^e/\mathbf{K}$. Usually, they are functions of saturation. We can expect relative permeabilities to be different during drainage and imbibition. Consider for instance, a drainage where non-wetting hydrocarbon phase migrates into a water-wetting porous medium completely saturated with water. The hydrocarbon is highly immobile as long as its saturation is below the irreducble saturation s_{oir}. Vice versa the water relative permeability start high and decreases while we approach s_{wir}. In a drainage process, the drained fluid (in this case the water) gets flushed out slowly and hence the relative permeability of water reduces slowly. The opposite effect would occur in an imbibition process.

System of equations: We state the generic multiphase flow model and follow up with the constitutive equations to relate physical quantities as we did for the single fluid flow equation. For N immiscible fluids we have for each fluid the mass conservation equation

$$\frac{\partial}{\partial \mathrm{t}}(\phi\rho_\alpha s_\alpha) + \nabla\cdot(\rho_\alpha \vec{v}_\alpha) = \rho_\alpha q_\alpha \tag{1.14}$$

with Darcy's flux

$$\vec{v}_\alpha = -\frac{\mathbf{K}k_{\mathrm{r}\alpha}}{\mu_\alpha}(\nabla p_\alpha - g\rho_\alpha \nabla z). \tag{1.15}$$

As it was mentioned for the relation of saturation and capillary pressure, the characteristics of physical quantity relations and dependencies vary a lot across different regimes. To this end, simplifying assumption are necessary to make the model more computational tractable for simulations. A common change in notation is to replace $\frac{k_{r\alpha}}{\mu_\alpha}$ by λ_α, also known as the mobility of α.

$$\begin{aligned} \frac{\partial}{\partial t}\left(\phi\rho_\alpha s_\alpha\right) + \nabla\cdot(\rho_\alpha\vec{v}_\alpha) &= \rho_\alpha q_\alpha \\ \vec{v}_\alpha = -\mathbf{K}\lambda_\alpha\left(\nabla p - \rho_\alpha g\nabla z\right), &\qquad \lambda_\alpha = k_{r\alpha}/\mu_\alpha \\ q_\alpha = \lambda_\alpha^w J\left(p^w - p\right) & \end{aligned} \tag{1.16}$$

where

α :	phase (oil/water/gas)	$k_r(s)$:	relative permeability
p :	pressure	$\mu(p)$:	viscosity
s :	saturation	g :	gravity constant
$\vec{v}$:	Darcy flux	z :	depth
$\mathbf{K}$:	permeability (tensor)	q :	mass source
$\phi(p)$:	porosity	J :	well connection factor
$\rho(p)$:	density	$\lambda^w(s,p)$:	well mobility
$\lambda(s,p)$:	mobility	p^w :	well pressure

: state variables governed in the system

The relations are indicated by the brackets and required to provide enough equations to solve the system (1.16). We will state these relations briefly. The *isothermal compressibility of the rock* is measure of change in pore volume per change in fluid pressure [1]. Isothermal indicates that the temperature is assumed to be constant. Then we have

$$\frac{\mathrm{d}\phi}{\mathrm{d}p} = c_r\phi.$$

We can assume that c_r is piece wise constant, then $\phi(p) = \phi_0 \exp(c_r(p - p_0))$. Similarly, the fluid compressibility is characterized through the change in density per change in pressure with the temperature being fixed. We get

$$\frac{\mathrm{d}\rho}{\mathrm{d}p} = c_f\rho.$$

The *mobility* combines the viscosity and the relative permeability. Note that $k_{r\alpha}$ is often a monotone function in the range [0,1]. This is due to the fact that if $k_{r\alpha} = 1$ then $s_\alpha = 1$ and vice versa for $k_{r\alpha} = 0$. The viscosity $\mu(p)$ is often assumed be linearly depending on the pressure. Such that we can assume a relation of the form

$$\mu(p) = \mu_0 \left[1 + c_\mu(p - p_0)\right].$$

The well mobility is the fraction of relative permeability with the viscosity of the fluid. Then

$$\lambda_\alpha(s,p) = \frac{k_{r\alpha}(s)}{\mu(p)}.$$

1.2 Numerical Solution

To illustrate the simulation of porous media flow, we consider again the governing equations of compressible, single-phase fluid flow

$$\frac{\partial(\phi\rho)}{\partial t} + \nabla\cdot(\rho\vec{v}) = \rho q, \qquad \vec{v} = -\frac{\mathbf{K}}{\mu}(\nabla p - g\rho\nabla z). \tag{1.17}$$

The approach presented herein follows the routines implemented in the open-source software MRST (which we will come back to in Chapter 3).

First we consider a time-independent equation and illustrate the use of a finite volume method, which is called the two-point flux approximation (TPFA) scheme.

1.2.1 Discretization in Space

There are two discretization methods required to spatially discretize the system in Eq. (1.16). This will also lead to a solution which is defined over the underlying discretization in the grid. The solution is then a vector of dimension in $\mathcal{O}(n_c)$ and the number of cells will be the magnitude of the dimension of the residual function and the state dimension. The first method derives a relation between the discretized pressure gradient to the flux and incorporates the parameters of Darcy's law. In particular, the permeability is part and the geometry of the grid structure are the defining parameters of such a proportionality constant.

The second set of methods are related to the differential operators. We can simplify those analytical operators by setting a lower bound on the actual notion of infinitesimal small deviation: the lower bounds are the distances between neighboring cells in the grid. The discretization is again then of dimension $\mathcal{O}(n_c)$ and presents a way to provide a finite representation of such an abstract notion for differentiation, depending solely on the geometry of the grid structure. By the linearity of the analytical differential operators it follows that the finite representation of those operators must be a matrix and we achieve a boiled down equation with only basic operators such as minus, plus, multiplication and some few basic unary functions such as $\exp, \log$ or other tabulated property functions for the fluid parameters.

TPFA: The two-point flux approximation uses the variational formulation of our governing equations and it is required to find the constant of the proportionality (called the transmissibility) that relates a pressure difference to an intercell flux. Both transmissibility and the differential operators depend on the geometry of the grid. However, while it is an exclusive dependency for the differential operators, the transmissibility is also formed by the permeability of the medium.

We consider a single-phase flow with an incompressible fluid. In the absence of gravity, Eq. (1.17) simplifies to the well-known Poisson-equation for the fluid pressure p, here written in first-order form

$$\nabla \cdot \vec{v} = q, \quad \vec{v} = -\mathbf{K}\nabla p, \quad \text{on } \Omega \subseteq \mathbb{R}^3. \tag{1.18}$$

We consider a partition $(\Omega_i)_{i\in I}$ of Ω, where we can identify Ω_i as one cell of the grid. Let us consider an arbitrary but fixed cell Ω_i. Using integration on both sides of the first equation and applying the divergence theorem on the left hand side, yields

$$\int_{\partial\Omega_i} \vec{v} \cdot \vec{n} ds = \int_{\Omega_i} q d\vec{x}. \tag{1.19}$$

The integral on the left-hand side can be written as a sum of integrals that are restricted to the interface with each of the neighbouring cells. This results in sub integrals of the form

$$v_{i,k} = \int_{\Gamma_{ik}} \vec{v} \cdot \vec{n} \mathrm{dS}, \quad \Gamma_{i,k} = \partial\Omega_i \cap \partial\Omega_k \tag{1.20}$$

for all k where $k \neq i$. Here, $v_{i,k}$ describes the flux from cell i to cell k on the intersection of those two cells. Let $x_{i,k}$ be the centroid of $\Gamma_{i,k}$ and $A_{i,k}$ be the area of

$\Gamma_{i,k}$. Assuming small variations of $v_{i,k}$ on the face $\Gamma_{i,k}$, we can use the second equation with the pressure gradient replaced by a standard finite-difference approximation to obtain the approximation of $v_{i,k}$

$$v_{i,k} \approx A_{i,k}\mathbf{K}_i \frac{(p_i - \pi_{i,k})\,\vec{c}_{i,k}}{\|\vec{c}_{i,k}\|^2} \cdot \vec{n}_{i,k} = T_{i,k}\,(p_i - \pi_{i,k})\,. \tag{1.21}$$

Here,

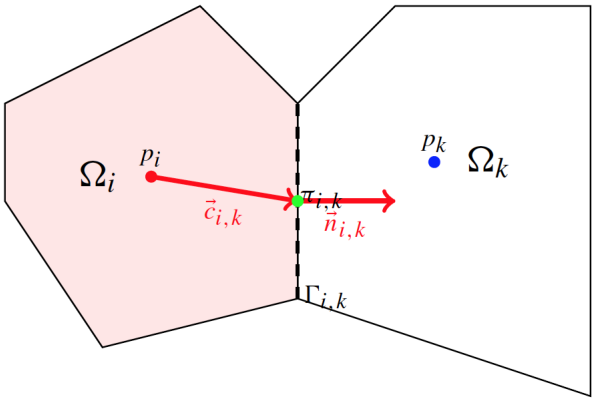


Figure 1.1: An illustration of the TPFA scheme with two neighboring cells. Taken from [22, Section 4.4].

$v_{i,k}$:	flux at $\Gamma_{i,k}$	K_i :	permeability $\mathbf{K}_i$ in Ω_i
$A_{i,k}$:	area of $\Gamma_{i,k}$	$\pi_{i,k}$:	pressure on $\Gamma_{i,k}$
$x_{i,k}$:	the midpoint of the face $\Gamma_{i,k}$	p_i :	pressure on $\Gamma_{i,k}$
$\vec{c}_{i,k}$:	the connection vector of the inner cell to $x_{i,k}$	$T_{i,k}$:	half transmissibility on $\Gamma_{i,k}$
		T_{ik} :	transmissibility between Ω_i and Ω_k

If we consider the relations of pressure and flux from both sides, we obtain

$$T_{i,k}^{-1} v_{ik} = p_i - \pi_{ik}, \quad -T_{k,i}^{-1} v_{ik} = p_k - \pi_{ik}.$$

Finally, by eliminating the interface pressure π_{ik}, we end up with the following two-point scheme for the flux approximation,

$$v_{ik} = \left[T_{i,k}^{-1} + T_{k,i}^{-1}\right]^{-1} (p_i - p_k) = T_{ik}\,(p_i - p_k)\,.$$

where

$$T_{i,k} = \frac{A_{i,k}}{\|\vec{c}_{i,k}\|^2}\mathbf{K_i}\vec{c}_{i,k} \cdot \vec{n}_{i,k}, \qquad T_{ik} = \left(T_{i,k}^{-1} + T_{k,i}^{-1}\right)^{-1}. \tag{1.22}$$

The special case where we have isotropic permeability (that is $\mathbf{K}_i \in \mathbb{R}$) and $\frac{\vec{c}_{i,k}\cdot\vec{n}_{i,k}}{\|\vec{c}_{i,k}\|^2} = \frac{\vec{c}_{k,i}\cdot\vec{n}_{k,i}}{\|\vec{c}_{k,i}\|^2}$ emits that T_{ik} up to a constant is the harmonic mean of the permeabilities of the two cells. In particular, exploiting properties of the harmonic mean, we have

$$\min\{T_{i,k}, T_{k,i}\} \le 2T_{ik} \le 2\min\{T_{i,k}, T_{k,i}\},$$

that is, the transmissibility times the area of the face is in the magnitude of the smaller permeability value, as one would expect. This observation can be generalised

by considering the cosinus of the angle between $\vec{c}_{i,k}$ and $\vec{n}_{i,k}$ divided by the length of $\vec{c}_{i,k}$

$$\frac{\vec{c}_{i,k} \cdot \vec{n}_{i,k}}{\|\vec{c}_{i,k}\|^2}$$

as the weight for the permeability from i to k. In Example 1.2 the cosinus value is 1 and hence we are weighting the permeability only in terms of the length of $\vec{c}_{i,k}$ and $\vec{c}_{k,i}$, respectively.

By summing up over all neighboring cells of Ω_i, we obtain an approximation of $\int_\Omega \vec{v} \cdot \vec{n} \mathrm{dS}$. Thus by solving the equation

$$\sum_k T_{ik}(p_i - p_k) = \int_{\Omega_i} q \mathrm{d}\vec{x}, \quad \forall \Omega_i \subseteq \Omega \tag{1.23}$$

after the pressure, we obtain an approximated solution of the system (1.18). Assembling all equations of the form of Eq. (1.23) into a matrix $T \in \mathbb{R}^{n_f \times n_f}$, results in a symmetric linear system with variables representing the pressure differences. Further, with a small modification T is sparse with a banded structure: that means in 1D the matrix is tridiagonal (since we only have two neighboring cells for each cell), and penta- heptadiagonal for a Cartesian grid in 2D and 3D, respectively with the same argumentation for the 1D case.

As the name of the scheme suggests, we relate the pressure of two points with the flux between two cells.

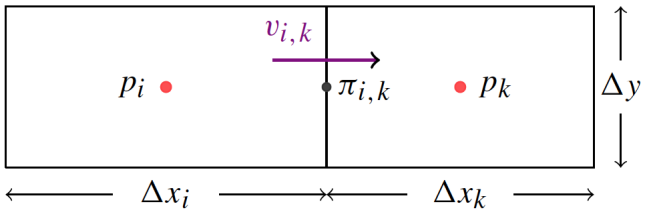


Figure 1.2: A simple 2D example to illustrate the computation of the transmissibility constant and its physical interpretation. From [22, Section 4.4].

Example 1.2:
Consider the situation of Figure 1.2.1, that is, a Cartesian grid in 2D, where permeability is isotropic, that is, $\mathbf{K}_i \in \mathbb{R}$. Further consider the equation

$$\nabla \cdot v = 0$$

for incompressible fluid flow. We then observe the following:

$$A_{i,k} = \Delta y, \qquad c_{k,i} = \begin{pmatrix} -1/2\Delta x_k \\ 0 \end{pmatrix}$$
$$c_{i,k} = \begin{pmatrix} 1/2\Delta x_i \\ 0 \end{pmatrix} \qquad \vec{n}_{i,k} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = -\vec{n}_{k,i}.$$

This yields

$$v_{i,k} = \Delta y \frac{(p_i - \pi_{i,k})}{\left(\frac{1}{2}\Delta x_i\right)^2} K_i \begin{pmatrix} \frac{1}{2}\Delta x_k \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \Delta y \frac{2K_i}{\Delta x_i} (p_i - \pi_{i,k}) \tag{1.24}$$

and likewise

$$v_{k,i} = \Delta y \frac{(p_k - \pi_{k,i})}{\left(\frac{1}{2}\Delta x_k\right)^2} K_k \begin{pmatrix} -\frac{1}{2}\Delta x_k \\ 0 \end{pmatrix} \cdot \begin{pmatrix} -1 \\ 0 \end{pmatrix} = \Delta y \frac{2K_k}{\Delta x_k} (p_k - \pi_{k,i}) . \tag{1.25}$$

The transmissibility then computes to

$$T_{ik} = \left(T_{i,k}^{-1} + T_{k,i}^{-1}\right)^{-1} = 2\Delta y \left(\frac{\Delta x_i}{\mathbf{K}_i} + \frac{\Delta x_k}{\mathbf{K}_k}\right)^{-1}.$$

T_{ik} is indeed the harmonic mean of the permeabilities per length unit times the area of the face $\Gamma_{i,k}$. This matches the physical expectation that the lower permeability value will be the dominant component of the transmissibility.

Concluding this section, we are able to solve the system (1.16) in space with the finite-volume method TPFA. This algorithm is, due to its robustness and efficiency, currently industry standard (state 2019). Equation (1.22) indicates how transmissibility incorporates geometric information of the grid through the permeability and the angles between cell mid-points and face mid-points. It is explicitly not depending on the fluid. To generalize to other system of equations, the idea is to introduce discrete operators for the divergence and the gradient operators which incorporate information about the grid. Just as the transmissibility constants emits a sparse matrix, we can obtain those in a more general setting.

Discrete Differential Operators A convenient concept for the space discretization of Eq. (1.1) is that of *discrete gradient* and *discrete divergence* operators. It is based on the idea to connect differential operators to the specific underlying grid system, thereby hiding away information about is geometry and topology.

We are considering a space discretization and the governing variables (here mainly pressure) are averaged over the grid cells. A pressure field then becomes a vector

with dimension equal to the number of grid cells. The divergence in a physical interpretation quantifies the amount of outflow minus the amount of inflow in an infinitesimal small point. In the discrete case, we instead define the cells as the infinitesimal small points and then the discrete divergence is simply a sum of the net flow across the interfaces the cell make with its neighbors.

Similarly, we can represent a flux on the grid as a value for each face in the grid that accounts for the difference the cell value on opposite sides of the interface. In this way, the produced analogy matches the analytical definition of the gradient in space as the fraction of the difference between function values of two infinitesimal deviated points in space divided by the deviation itself. Now the spatial deviation is replaced by considering neighboring cells and we assign the function value difference to the corresponding face between two neighboring cells. Using indicator functions this results in a sparse matrix mapping from $\mathbb{R}^{n_c} \to \mathbb{R}^{n_f}$ where n_c is the number of cells in the grid and n_f is the number of faces in the grid.

Unsurprisingly, the discrete differential operators behave quite similar to their analytical analogy. For instance there is a version of the Gauss–Green formula for the discrete divergence and gradient operator. In functional analysis, the Gauss–Green formula simplifies to the statement that for the Hilbert space $\mathcal{L}_2$ the gradient operator is the negative adjoint operator of the divergence operator. Translated to the matrix language this implies that the linear mappings are the negative transposed of each other. This lays the foundation of breaking down Eq. (1.1) into a system of $\mathcal{O}(n_c)$ equations which only involves basic operations without any differential operators.

We want to formalize the ideas of the former in the following. Consider a grid G with cells $\{1, \dots, n_c\}$ and faces $\{1, \dots, n_f\}$. Further the faces are all orientated as defined by mappings $C_1, C_2 : \{1, \dots, n_f\} \to \{1, \dots, n_c\}$ such that f is oriented from $C_1(f)$to $C_2(f)$. We define the discrete divergence div of a flux v for a cell c as the sum of the outgoing fluxes minus the sum of the incoming fluxes, that is,

$$\operatorname{div}(v)[c] = \sum_{f \in C_1^{-1}(\{c\})} v(f) - \sum_{f \in C_2^{-1}(\{c\})} v(f).$$

Complementary, the discrete gradient grad for pressure is defined for a face where we compare the pressures on the neighboring cells of the face. That is,

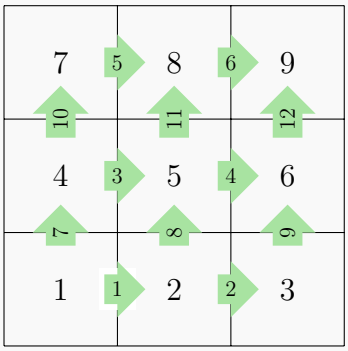
$$\operatorname{grad}(p)[f] = p(C_2(f)) - p(C_1(f)).$$

To illustrate this, we consider the following example.

Example 1.3:
Consider the Poisson equation in the box geometry, which we write as follows using the discrete operators

$$-\mathrm{div}(T\mathrm{grad}(p)) = q, \quad \Omega = [0,1] \times [0,1], \tag{1.26}$$

where T is the transmissibility matrix depending on the permeability and the specific cell geometries of the underlying grid. Let us consider a basic Cartesian grid in 3×3 of Ω.



First we consider the cells $\{1, 2, \ldots, 9\}$ and the faces $\{1, 2, \ldots, 12\}$ to which we associate unknown pressures and fluxes, respectively; that is, pressures $p \in \mathbb{R}^9$ and fluxes $v \in \mathbb{R}^{12}$. We can represent div, grad as linear mappings which are transposed up to sign to each other, that is, there exist tatrices $A \in \mathbb{R}^{9\times12}$ and $B \in \mathbb{R}^{12\times9}$ such that

$$(Av)_c = \mathrm{div}(v)[c]$$

and

$$(Bp)_f = \mathrm{grad}(p)[f]$$

hold for all $c \in \{1, \ldots, 9\}, f \in \{1, \ldots, 12\}, v \in \mathbb{R}^{12}$, and $p \in \mathbb{R}^9$. Moreover, $A = -B^{\mathrm{T}}$ and this is independent from the grid and the grid separation. The solution of (1.26) can be found by solving the linear equation

$$-ATA^{\mathrm{T}}p = q$$

for p, where $q \in \mathbb{R}^9$ contains the source value for each cell.

(Proof sketch.) With the definition of the grid we also define the orientation of each face. Each face f has exactly one origin cell $C_1(f)$ and another cell $C_2(f)$ that f is pointing towards, where $C_1, C_2 : \{1, \ldots, 12\} \to \{1, \ldots, 9\}$. The divergence operator is defined for a cell $c \in \{1, \ldots, 9\}$ on a vector $v \in \mathbb{R}^{12}$

$$\mathrm{div}(v)[c] = \sum_{f \in C_1^{-1}(\{c\})} v_f - \sum_{f \in C_2^{-1}(\{c\})} v_f,$$

that is, the sum of the fluxes going out of the cell minus the sum of the fluxes going

into the cell. Hence, we can also write div as the matrix

$$A \in \mathbb{R}^{9\times 12}, \qquad A_{i,j} = \begin{cases} 1, & i = C_1(j), \\ -1, & i = C_2(j), \\ 0, & \text{else}, \end{cases}$$

such that

$$\operatorname{div}(v)[c] = (Av)_c.$$

Furthermore, we define the discrete gradient operator grad as

$$\operatorname{grad}(p)[f] = p(C_2(f)) - p(C_1(f))$$

which clearly yields the matrix

$$B \in \mathbb{R}^{12\times 9}, \qquad B_{i,j} = \begin{cases} 1, & C_2(i) = j, \\ -1, & C_1(i) = j, \\ 0, & \text{else} \end{cases}$$

such that we have

$$(Bp)_f = \operatorname{grad}(p)[f]$$

for all $p \in \mathbb{R}^9$. Hence, $A = -B^{\mathrm{T}}$ as claimed.

The matrix A has the form

$$\begin{array}{c} \text{cells / faces} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\ \begin{pmatrix} 1 & & & & & & 1 & & & & & \\ -1 & 1 & & & & & & 1 & & & & \\ & -1 & & & & & & & 1 & & & \\ & & 1 & & & & -1 & & & 1 & & \\ & & -1 & 1 & & & & -1 & & & 1 & \\ & & & -1 & & & & & -1 & & & 1 \\ & & & & 1 & & & & & -1 & & \\ & & & & -1 & 1 & & & & & -1 & \\ & & & & & -1 & & & & & & -1 \end{pmatrix} \end{array}. \tag{1.27}$$

Now assuming that the permeability is isotropic and equal to one, we obtain

$$v = T\operatorname{grad}(p).$$

To solve the system (1.26) with respect to the discrete operators, we instead determine $p \in \mathbb{R}^9$ such that

$$AT(-A^{\mathrm{T}})p = q,$$

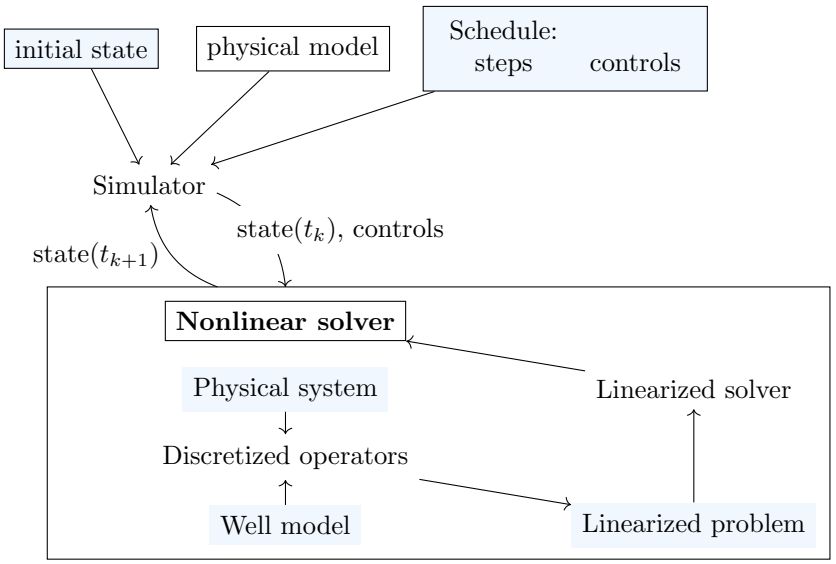
where $q \in \mathbb{R}^9$ and each entry q_i equals the production or injection amount in cell i. □

1.2.2 References

The system of equations (1.16) is taken from Stein Krogstad's presentation at Geilo Winterschool 2022. The derivation of the TPFA method and the figures 1.2.1 and 1.2.1 are inspired by and taken from Lie's book [22, Chapter 4.4], respectively.

1.3 Reservoir Simulation

The following section aims to bring the former topics together, in order to compute a full forward reservoir simulation based on finite volume methods and Newton–Raphson.



The diagramm shows the classical simulation loop and can be read as follows: First, the setup for the simulator consists of three main components. The initial state (represented as a set of discrete quantities), the discrete mathematical equations that model the physical behavior of the system on a finite grid, and a set of instances in time at which we seek to compute the discrete state of the system. Each of these time instances have associated definitions of the boundary conditions and the source terms, which in turn control the behavior of the overall system. Herein, we refer to the time steps and these controls as the schedule that determines how the solution process evolves.

Given these three, the task of the simulator is simply to generate the discrete states at the given instances in time: state(t_1), ..., state(t_N). This is done consecutively by solving the step from t_k to t_{k+1} for $k = 0, \dots, N$, where N is the number of overall time steps.

To define the discrete model equations on a given time step, the simulator uses discrete operators to approximate in space and implicit-Euler for the time discretization of the partial differential equation emerging from the physical model. The system emerging from the physical model includes mainly conservation of mass, Darcy's law, the well model and fluid and rock compressibility to relate porosity and density with pressure. We solve the accumulated and discretized *nonlinear* system using a standard Newton–Raphson iterative solver. That means, if the have the set of equations written in residual form as $F(x) = 0$, where x are the governing variables, then we update x_i by the iteration

$$x_{i+1} \leftarrow x_i + \frac{\partial F(x_i)}{\partial x}^{-1} F(x_i),$$

where x_0 is given from the state at time t_k. The solution yields a new state at time t_{k+1} and the simulation step continues with the next time step. The described

procedure is rather a general simulation loop. We will generalize the example in Lie's book [22, Section 7.2]. The general procedure is as follows:

1. Generate a discretization of time and space via a schedule and a time step size and grid cell structure.
2. Set up relations for porosity, density, viscosity in dependence of pressure.
3. Set up the well model—production at the well connection and resulting production/injection rate of the well.
4. Set up laws for conservation of mass and Darcy's law.
5. Discretize all equations with implicit-Euler and discrete divergence and gradient operators as in Eq. (1.33) and compute the transmissibility as in Eq. (1.27) and collect all equations with right hand side zero into a system F.
6. Find a zero of F with Newton–Raphson regarding the governing variables update the state.

We have previously described the equations for mass conservation, Darcy's law, and many of the closure relationships that relate different physical quantities. What remains is to describe how to model source terms from injection and production wells.

Well models: For the source term q we consider wells as thin cylinders with a one-dimensional expansion in the grid. The flow inside the wellbore is described by a pressure at a datum point inside the well (e.g., at the bottom of the well). The pressure in the remaining parts in the well are assumed to be hydrostatically distributed, that is, the pressure satisfies the ordinary differential equation

$$\frac{\partial p}{\partial z} = g\rho(p).$$

The flow in or out of the well is then determined by the difference between the pressure inside the wellbore and the pressure in the surrounding rock.

Let us consider a well W and let $N(W)$ be the set of cells which contain a part of this well. We call those cells also *well connections*. We have to distinguish two production rates. One is the right-hand side for our governing equation of the mass conservation law. It is based on the assumption that the pressure difference from the well connection is proportional to the pressure inside of the grid cells directly at the well. The proportionality constant includes the dimensionless well index WI, which depends on the geometry of the well bore and the petrophysical properties near the well, and the viscosity of the fluid. That results in

$$q_c = \frac{\rho}{\mu}\mathrm{WI}\big(p_c - p\big), \quad c \in N(W)$$

which is referred as the standard Peaceman model. Here, p_c is the pressure in the well connection and we assume in the Peaceman model that it is hydrostatically distributed. In particular, we obtain

$$p_c = p_{\mathrm{bhp}} + g\Delta z_w \rho(p_{\mathrm{bhp}}), \quad c \in N(W),$$

where Δz_w is the vertical distance from the datum point (bottom hole) of the well to the connection point and p_{bhp} is the bottom-hole pressure given by the controls as boundary conditions of our system. It defines if a well is a producer or injector —if p_{bhp} is below the pressure at the well connection then the well is a producer and vice versa.

The other production rate is referred to the amount of fluid that leaves or enters the system. For this we simply sum up all production rates in the cells which contain the well and divided the sum by the surface density ρ_S, that is,

$$q_P = \frac{1}{\rho_S} \sum_{c \in N(W)} q_c(p, p_{\text{bhp}}).$$

Note that q_P is the quantity of interest to assess fluid recovery from a reservoir.

The complete system: Given the relations between porosity and density with pressure, all the previous equations can be summarized as the complete system

$$\frac{\partial \phi \rho}{\partial t} + \nabla \cdot \left[\rho \left(-\frac{\mathbf{K}}{\mu} (\nabla p - g\rho \nabla z) \right) \right] = \begin{cases} 0 & , x \notin N(W) \quad \forall W, \\ q_c & , \exists W \text{ such that } x \in N(W) \end{cases} \tag{1.28}$$

$$q_c = \frac{\rho}{\mu} \text{WI}(p_c - p), \quad c \in N(W) \tag{1.29}$$

where

$$p_c = p_{\text{bhp}} + g\Delta z_w \rho(p_{\text{bhp}}) \tag{1.30}$$

$$q_P = \frac{1}{\rho_S} \sum_{c \in N(W)} q_c(p, p_{\text{bhp}}) \tag{1.31}$$

$$p_{\text{bhp}} = h(p, t, q_P) \tag{1.32}$$

for some function h depending on the time and given from the predefined well schedule. The governing variables of this system will be p, p_{bhp}, q_P. We consider this system of equations on a discrete grid structure, hence for n_c cells in the grid, we have $p \in \mathbb{R}^{n_c}$, $p_{\text{bhp}}, q_P \in \mathbb{R}$. The equations (1.29)–(1.32) can be evaluated for each time step in a straight forward way in vector form. For the continuity equation 1.28, we essentially use the same finite-volume discretization we have discussed earlier. If we let p_0 be the pressure from the previous time step, the equation for the unknown pressure p at the current time step reads,

$$\frac{1}{\Delta t} \Big(\phi(p)p - \phi(p_0)p_0 \Big) - \text{div} \left[\rho(p_{\text{avg}}) \left(\frac{T}{\mu} (\text{grad}(p) - g\rho(p_{\text{avg}}) \text{grad}(z) \right) \right] - q\big(p, p_{\text{bhp}}, q_P\big) = 0, \tag{1.33}$$

where $\rho(p_{\text{avg}}) \in \mathbb{R}^{n_f}$ is defined on each face via the arithmetic average of the pressure between the two neighboring cells of each face. That is, for a face f from cell c_1 to c_2 and $p \in \mathbb{R}^{n_c}$ we have $p_{\text{avg}}(f) = \frac{1}{2}(p(c_1) + p(c_2))$. It should be noted that in the more general case, the viscosity μ is depending on the pressure as well and one finding a similar expression for $\mu = \mu(p)$. In this case, we need to use $\mu(p_{\text{avg}}) \in \mathbb{R}^{n_f}$ again,

since the discrete divergence operator is defined on each face. This is not required when evaluating q_c for $c \in N(W)$.

We stack this equation and the equations (1.29)–(1.32) upon each other and redistribute all terms to the left-hand side such that we can write this in a system $F(p_0, p, p_{\text{bhp}}, q_P) = 0$. Hence, F is a system of non-linear equations and in order to find a solution, we perform Newton–Raphson iterations. Let J be the Jacobian of F, then the iteration is

$$J\Delta x = -F(p_0, p_i, p_{\text{bhp}i}, q_{Pi}), \quad (p_{i+1}, p_{\text{bhp}(i+1)}, q_{P(i+1)}) \leftarrow (p_i, p_{\text{bhp}i}, q_{Pi}) + \Delta x,$$

where we continue to compute a new value of $\Delta x \in \mathbb{R}^{n_c+2}$ until we reach a sufficient close approximation for a zero of F. The outer scheme of the simulation is then as summarized in Algorithm 1. In the next section we discuss a powerful numerical technique we can use to avoid differentiating F by hand to derive analytic expressions for the Jacobian J.

Algorithm 1: Simulation loop

Input: TotalTime, Δt, iteration tolerance $10^{-5} = \epsilon > 0$, number of maximal iteration per step maxIt = 10, initial conditions p_init, $h(p, t)$
Output: sol

```
1  sol = repmat(struct('time', ,'pressure', ,'bhp', , 'qP', ), [nSteps +1, 1]) ;
2  t = 0; step = 0;
3  (p,qP,bhp) = (p_init,0, h(p_init, 0));
4  while t < TotalTime do
5      t = t + dt; step = step +1; resNorm = 10^99; nit = 0 ;
6      p0 = p; % Fixed for the Newton iterations
7      while resNorm > ε and nit ≤ maxIt do
8          Set up F;
9          F = F(p0, p, bhp, qP);
10         J = dF(p0, p, bhp, qP) / d(p0, p, bhp, qP); % implicitly with ADI
11         resNorm = norm(F(p0,p, bhp, qP));
12         Solve JΔx = −F ;
13         (p, bhp, qP) = (p,bhp,qP) + Δx ;
14         nit = nit +1 ;
15     if nit > maxits then
16         error("Newton solver did not converge.");
17     else
18         sol(step +1 ) = struct ('time', t, 'pressure', p, 'bhp', bhp, 'qP', qP );
```

In a straight forward implementation this would require to compute the derivative separately and numerically. This does not only produce round-off errors but it also requires us to solve systems of the size of $\mathcal{O}(n_c^2)$ $\mathcal{O}(n_c)$ times. With increasing n_c which can be several million in real asset reservoir models, the numerical differentiation becomes more and more error prone. Automatic differentiation allows a stable employment of higher dimensions and also offers the advantage for including high complex fluid physics with tabulated property functions. This is for instance the case for the dependence of the capillary pressure and the saturation.
The multiphase flow simulation is not covered here in detail. For a two immiscible fluid flow simulation explanation consider for example [22, Section 8.3.2] which illustrates one process with the derivation of the fractional-flow formulation.

1.4 Automatic Differentiation

Automatic differentiation [31] is also called algorithmic differentiation or computational differentiation, has caved the path for efficient gradient computation in numerics. These sets of methods exploit the fact the any computational function is based on a sequence of elementary arithmetic operations and elementary functions. By applying the chain rule, the partial derivatives can be computed automatically while computing the function value. Compared to numerical differentiation and symbolic differentiation, it offers a much more stable and efficient method.

The idea of using automatic differentiation to develop reservoir simulators is not new. It has been mentioned in more commercial intersect simulators by DeBaun et. al 2005 [8] but has been mainly pioneered for the GPSS research simulator by Geoffrey 1978 [12].

To give a brief practical overview of automatic differentiation, we consider here an exemplary algorithm to illustrate the forward mode of this technique. For a deeper introduction consider for example [29].

First, we need to introduce automatic differentiation objects (AD objects). An AD object consists of the value of a function itself and the values of the partial derivatives with respect to a given set of primary variables, evaluated at the specific values of these variables that were used as input to the function evaluation. There are different implementations of automatic differentiation. Here we illustrate the principles used in MRST on a subset of possible operations: $\cdot$, $\pm$, and $()^k$ for $k \in \mathbb{R}$. The routine shown in Algorithm 2 can be generalized easily.

Algorithm 2: AD – Automatic differentiation

Input: $x_{\mathrm{AD}} = (x, \{\mathbf{I}_n, \mathbf{0}^{m\times m}\}), y_{\mathrm{AD}} = (y, \{\mathbf{0}^{n\times n}, \mathbf{I}_m\}), f$
Output: $f_{\mathrm{AD}}(x_{\mathrm{AD}}, y_{\mathrm{AD}}) = (f(x,y), \{\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y)\}$

if $f(x,y) = u(x,y) \cdot v(x,y)$ **then**
  $u_{\mathrm{AD}} = \mathrm{ADI}(x_{\mathrm{AD}}, y_{\mathrm{AD}}, u)$;
  $v_{\mathrm{AD}} = \mathrm{ADI}(x_{\mathrm{AD}}, y_{\mathrm{AD}}, v)$;
  return $(u(x)v(y), \{\frac{\partial u}{\partial x}v + \frac{\partial v}{\partial x}u, \frac{\partial u}{\partial y}v + \frac{\partial v}{\partial y}u\})$;
if $f(x,y) = u(x,y) \pm v(x,y)$ **then**
  $u_{\mathrm{AD}} = \mathrm{ADI}(x_{\mathrm{AD}}, y_{\mathrm{AD}}, u)$;
  $v_{\mathrm{AD}} = \mathrm{ADI}(x_{\mathrm{AD}}, y_{\mathrm{AD}}, v)$;
  return $(u(x) \pm v(y), \{\frac{\partial u}{\partial x} \pm \frac{\partial v}{\partial x}, \frac{\partial u}{\partial y} \pm \frac{\partial v}{\partial y}\})$;
if $f(x,y) = u(x,y)^k$ **then**
  $u_{\mathrm{AD}} = \mathrm{ADI}(x_{\mathrm{AD}}, y_{\mathrm{AD}}, u)$;
  return $(u(x,y)^k, \{ku^{k-1}\frac{\partial u}{\partial x}, ku^{k-1}\frac{\partial u}{\partial y}\})$;
if $f(x,y) = c \in \mathbb{R}$ **then**
  return $(c, \{\mathbf{0}^{n\times n}, \mathbf{0}^{m\times m}\})$;
if $f(x,y) = \vec{c}\cdot x$ *and* $\vec{c} \in \mathbb{R}^n$ **then**
  return $(\vec{c}\cdot x, \{\vec{c}\mathbf{I}, \mathbf{0}\})$;
if $f(x,y) = \vec{c}\cdot y$ **then**
  return $(\vec{c}\cdot y, \{\mathbf{0}, \vec{c}\mathbf{I}\})$;

We mainly consider partial derivatives with respect to two complementary sets of variables. Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$. The algorithm uses a recursion to compute partial derivative from basic operators using the chain rule. In general this also

includes $\log, \sin, \cos, \exp$. Consider a small example

$$f(x, y) = \sin(xy).$$

Now for computing the derivative after x at $(1, 2)$ we can introduce $u(x, y) = xy$ and write $\frac{\partial f}{\partial x}(x, y) = \frac{\partial u}{\partial x}(x, y)\cos(u)$, where $\frac{\partial u}{\partial x}(x, y) = y$, and hence

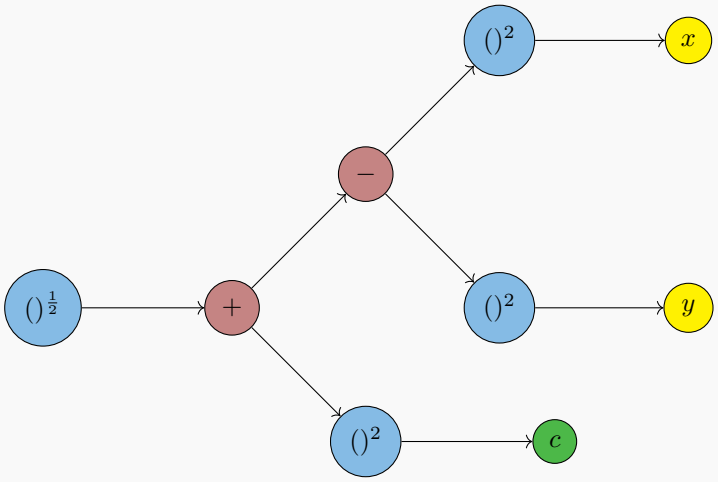
$$\frac{\partial f}{\partial x}(1, 2) = 2\cos(1 \cdot 2).$$

We can illustrate this substitutions of inner functions also by an operation tree.

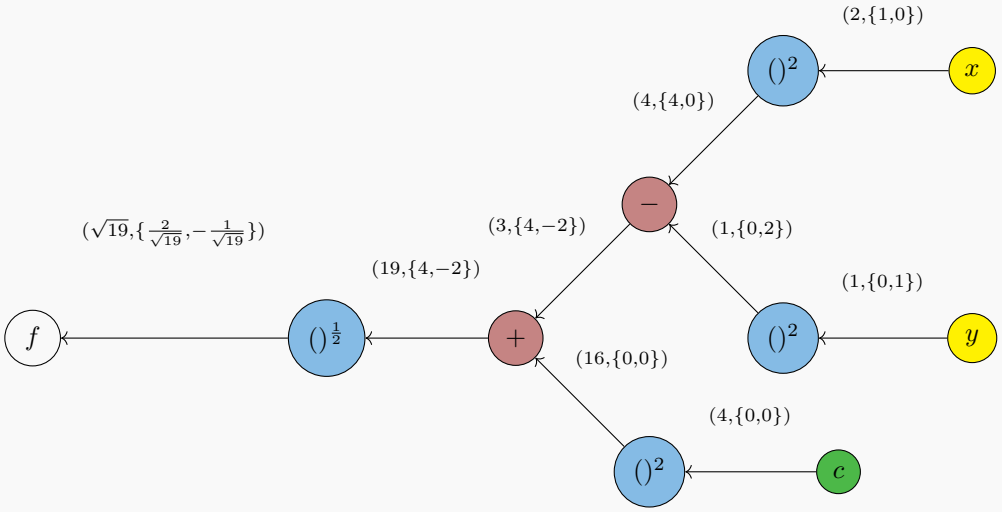
Example 1.4 (Automatic Differentiation):
Consider

$$f(x, y) = \left(x^2 - y^2 + 4^2\right)^{\frac{1}{2}}.$$

The task is to compute $f_{\mathrm{AD}}(2, 1)$. We set up the operation tree of f and perform the computation of Algorithm (2). The subtraction signs means that we subtract the lower branch from the upper branch.



We now evaluate the leaves with $x_{\mathrm{AD}} = (2, \{1, 0\})$ and $y_{\mathrm{AD}} = (1, \{0, 1\})$ and perform backtracing, using the rules of derivation for each partial derivative (as indicated in Algorithm 2). It is also referred as *forward mode automatic differentiation*.



Hence we have $\frac{\partial f}{\partial x}(2, 1) = \frac{2}{\sqrt{19}}$ and $\frac{\partial f}{\partial y}(2, 1) = -\frac{1}{\sqrt{19}}$.

We see that the partial derivatives can be computed in the time and space complexity of the evaluation of f. When computing the gradient numerically with difference approximations, which is used in `lsqnonlin` or `fmincon` (see [24, Section Input Arguments/options], [10, Section Input Arguments/options]), we have to evaluate

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + h_i e_i) - f(x - h_i e_i)}{2h_i} \tag{1.34}$$

for $i = 1, \ldots, m$ using $2m$ function evaluations and it is error prone for high dimensions. The drawback is the required structure—all functions and variables has to be defined as AD-objects.

Now every function evaluation comes with a constant overhead (as indicated in Example 1.4) but at the end of each function evaluation, we obtain the derivative automatically without any further computation steps.

1.5 Adjoint Equations

The original method goes back to Cea [3] from 1986 where he used this method with the Lagrangian multiplier formulation to obtain the gradient for an optimization problem with respect to parameters. We will omit the derivation through the Karush–Kuhn–Tucker conditions. Consider

$$\begin{aligned} \min_u \quad & j(x(u), u) \\ \text{s.t.} \quad & D(x(u), u) = 0 \end{aligned} \tag{1.35}$$

where $x(u) \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$. A straight forward computation of the gradient would be equivalent to solve the system

$$\frac{\mathrm{d}j}{\mathrm{d}u} = \frac{\partial j}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}u} + \frac{\partial j}{\partial u}.$$

Also with the continuity of $x(u)$ we have

$$0 = \frac{\mathrm{d}D}{\mathrm{d}u} = \frac{\partial D}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}u} + \frac{\partial D}{\partial u}$$

and hence

$$\frac{\partial D}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}u} = -\frac{\partial D}{\partial u}$$

is an equation with m rows. On the contrary, the adjoint equation is

$$\lambda\frac{\partial D}{\partial x} = \frac{\partial j}{\partial x}$$

where the right hand side has n many rows. The latter leads to

$$\frac{\mathrm{d}j}{\mathrm{d}u} = \lambda\frac{\partial D}{\partial u} + \frac{\partial j}{\partial u}.$$

Clearly, the first approach leads also to the desired total derivative of j after u. Hence both approaches will result in the gradient. However, we are considering for the calibration of reduced-order models (see Section 1.6) optimization problems

where the number of parameters is usually much larger than the state dimension x. The advantage of adjoint equations is evident in this case.
Let us derive the method in more detail. In particular, we exploit the form of our residual governing equations emerging from the implicit Euler discretization which has a special block bi-diagonal structure.
Let $f : [0, T] \times \mathbb{R}^n \to \mathbb{R}^n$ and $x(t) : \mathbb{R} \to \mathbb{R}^n$ represent the solution curve of

$$\frac{\mathrm{d}}{\mathrm{dt}} x(t) = f(t, x). \tag{1.36}$$

Consider a time discretization $(0 = t_0, t_1, t_2, \ldots, t_N = T)$ of $[0, T]$ with $N \in \mathbb{N}$ time steps and width $\Delta t_k = t_k - t_{k-1}$ for $k = 1, \ldots, N$. Recall the implicit Euler method on an ordinary differential equation. We obtain an approximated discretized solution by solving the implicit equation

$$x_{k+1} = x_k + (t_{k+1} - t_k) f(t_{k+1}, x_{k+1}), \quad x_k \in \mathbb{R}^n \tag{1.37}$$

for all $k = 0, \ldots, N - 1$ such that x_k is an approximation of $x(t_k)$.

Applying implicit Euler on the system 1.16, we obtain the discretization

$$\begin{aligned}(\phi \rho_\alpha s_\alpha)^{k+1} = (\phi \rho_\alpha s_\alpha)^k \\ - (t_{k+1} - t_k) \left(\nabla \cdot (\rho_\alpha K \lambda_\alpha (\nabla p - \rho_\alpha g \nabla z))^{k+1}) + q_\alpha^{k+1} \right),\end{aligned} \tag{1.38}$$

where the divergence operator is realized in terms of the TPFA method from Section 1.2.1. We can rewrite this in a system of functions $F_k : \mathbb{R}^{2n+m} \to \mathbb{R}^n$ with the conditions

$$F_k(x_k, x_{k-1}, u) = 0, \quad k = 1, \ldots, N \tag{1.39}$$

where x_k denotes the joined vector of $p_\alpha, s_\alpha, p_\alpha^W$ at time step k and u contains the parameters.

In MRST, the Newton–Raphson method will be used to compute an approximate solution of Eq. (1.39) after x_k, which describes the state of the reservoir in each cell regarding the parameters pressure, saturation and well pressure of water, oil and gas at time t_k.

We want to compute the gradient while we are evaluating our system (1.16) by using automatic differentiation and adjoint equations. To illustrate this we consider an optimization problem with m parameters represented by $u \in \Omega \subseteq \mathbb{R}^m$ and objective function $J(\mathbf{x}, u) = J(\mathbf{x}(u), u)$ where

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix}. \tag{1.40}$$

Together this yields the following optimization problem for initial data $\overline{x}_0 \in \mathbb{R}^n$

$$\min_{u \in \Omega} \quad J(x_0, \mathbf{x}, u) \tag{1.41}$$

$$\text{subject to} \quad F_k(x_k, x_{k-1}, u) = 0, \quad k = 1, \ldots, N, \tag{1.42}$$

$$x_0 = \overline{x}_0. \tag{1.43}$$

$$\frac{\mathrm{d}J}{\mathrm{d}u} = \frac{\partial J}{\partial \mathbf{x}} \frac{\mathrm{d}x}{\mathrm{d}u} + \frac{\partial J}{\partial u}. \tag{1.44}$$

Let

$$F(\mathbf{x}, u) = \begin{pmatrix} F_1(x_1, x_0, u) \\ \vdots \\ F_N(x_N, x_{N-1}, u) \end{pmatrix}.$$

Since $F(\mathbf{x}(u), u) = 0$ for all $u \in \Omega$, we have

$$0 = \frac{\mathrm{d}F}{\mathrm{d}u} = \frac{\partial F}{\partial \mathbf{x}} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}u} + \frac{\partial F}{\partial u} \tag{1.45}$$

giving

$$\frac{\partial F}{\partial \mathbf{x}} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}u} = -\frac{\partial F}{\partial u} \tag{1.46}$$

with

$$\frac{\partial F}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \frac{\partial F_3}{\partial x_2} & \frac{\partial F_3}{\partial x_3} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \frac{\partial F_N}{\partial x_{N-1}} & \frac{\partial F_N}{\partial x_N} \end{pmatrix}, \tag{1.47}$$

where $\frac{\partial F_k}{\partial x_k} \in \mathbb{R}^{n \times n}$. Hence $\frac{\partial F}{\partial \mathbf{x}}$ is a squared matrix and since $\frac{\partial F_k}{\partial x_k}$ is invertible as

$$\frac{\partial F_k}{\partial x_k} = \mathbf{I} - h_k \frac{\partial f(t_k, x_k)}{\partial x_k}$$

where f is defined as in Eq. (1.38) and h_k is the step size between timestep $k-1$ and k. The matrix on the right hand side is invertible for small h_k and hence $\frac{\partial F}{\partial \mathbf{x}}$ is invertible. At this point we are distinguishing between two options. In order to compute $\frac{\mathrm{d}J}{\mathrm{d}u}$ we can either compute $\frac{\mathrm{d}x}{\mathrm{d}u}$ first and multiply it with $\frac{\partial J}{\partial \mathbf{x}}$, or we can insert $\frac{\mathrm{d}x}{\mathrm{d}u}$ into Eq. (1.44) and consider the adjoint equation first, that is,

$$\lambda := -\frac{\partial J}{\partial \mathbf{x}} \left(\frac{\partial F}{\partial \mathbf{x}} \right)^{-1} \tag{1.48}$$

$$\frac{\mathrm{d}J}{\mathrm{d}u} = \underbrace{-\frac{\partial J}{\partial \mathbf{x}} \left(\frac{\partial F}{\partial \mathbf{x}} \right)^{-1}}_{=\lambda} \frac{\partial F}{\partial u} + \frac{\partial J}{\partial u}. \tag{1.49}$$

The name adjoint variable for λ comes from an equivalent derivation by considering the unconstrained optimization problem of Problem (1.41) obtained by applying the Lagrange multiplier method. Explicitly, we obtain

$$\min_{u \in \Omega, \lambda \in \mathbb{R}^{1 \times Nn}} J(x_0, \mathbf{x}(u), u) + \lambda F(\mathbf{x}(u), u) \tag{1.50}$$

where the introduced λ is often referred to as adjoint variable or Lagrange multiplier, and the condition for λ emerging from further analysis is the adjoint equation $\lambda = -\frac{\partial J}{\partial x} \left(\frac{\partial F}{\partial x} \right)^{-1}$.

Implementation of Adjoint Equations To prevent confusion, we explain the dimensions of the stacked equations. $\mathbf{x}$ is vector which has all the N states stacked on each other, where each state consists of n values and hence $\mathbf{x} \in \mathbb{R}^{Nn\times 1}$. $F(\mathbf{x})$ is the residual for each time step from 1 to N. The residual is also a vector of n entries, since the F_is have the same dimension as the states x_j.

Let $x_0 \in \mathbb{R}^n$ be fixed. We have

$$\frac{\mathrm{d}J}{\mathrm{d}u} = \lambda \frac{\partial F}{\partial u} + \frac{\partial J}{\partial u} \tag{1.51}$$

and

$$\lambda \frac{\partial F}{\partial \mathbf{x}} = -\frac{\partial J}{\partial \mathbf{x}} \tag{1.52}$$

with $u \in \mathbb{R}^m, \mathbf{x} \in \mathbb{R}^{Nn\times 1}, F(\mathbf{x}) \in \mathbb{R}^{Nn\times 1}$ and $J(x_0, \mathbf{x}, u) \in \mathbb{R}$. That means for instance that $\lambda \in \mathbb{R}^{1\times Nn}$. From the sensitivity matrix in Eq. (1.47) , we obtain the equation

$$\begin{aligned}
&\begin{pmatrix} \lambda_1 & \dots & \lambda_N \end{pmatrix} \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \frac{\partial F_3}{\partial x_2} & \frac{\partial F_3}{\partial x_3} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \frac{\partial F_N}{\partial x_{N-1}} & \frac{\partial F_N}{\partial x_N} \end{pmatrix} && (1.53) \\
&= \begin{pmatrix} \lambda_1 \frac{\partial F_1}{\partial x_1} + \lambda_2 \frac{\partial F_2}{\partial x_1}, & \lambda_2 \frac{\partial F_2}{\partial x_2} + \lambda_3 \frac{\partial F_3}{\partial x_2}, & \dots, & \lambda_N \frac{\partial F_N}{\partial x_N} \end{pmatrix} && (1.54)
\end{aligned}$$

where $\lambda_i \in \mathbb{R}^{1\times n}$ and $\frac{\partial F_i}{\partial x_j} \in \mathbb{R}^{n\times n}$. Equivalently, we have

$$\lambda_N \frac{\partial F_N}{\partial x_N} = -\frac{\partial J}{\partial x_N} \tag{1.55}$$

$$\lambda_k \frac{\partial F_k}{\partial x_k} = -\frac{\partial J}{\partial x_k} - \lambda_{k+1} \frac{\partial F_{k+1}}{\partial x_k} \tag{1.56}$$

$$\text{for } k = N-1, \dots, 1. \tag{1.57}$$

We assume that the simulation is given with states $\mathbf{x}(u) \in \mathbb{R}^{Nn\times 1}$ which are depending on the choice of the parameters u. Then with automatic differentiation and by solving the adjoint equation, we can formulate the following algorithm.

Algorithm 3: Adjoint equation – gradient

Input: $u \in \mathbb{R}^m, \mathbf{x} \in \mathbb{R}^{Nn\times 1}, J, F$
Output: $\frac{\mathrm{d}J}{\mathrm{d}u}$

```
solve λ_N ∂F_N/∂x_N = −∂J/∂x_N;
res = λ_N ∂F_N/∂u;
for k = N − 1, ..., 1 do
    solve λ_k ∂F_k/∂x_k = ( −∂J/∂x_k − λ_{k+1} ∂F_{k+1}/∂x_k );
    res = res + λ_k ∂F_k/∂u;
return res;
```

There is also the possibility to solve the forward sensitivity equations, which is solving for $\frac{\mathrm{d}x}{\mathrm{d}u}$ in

$$\frac{\partial F}{\partial \mathbf{x}} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}u} = -\frac{\partial F}{\partial u} \tag{1.58}$$

which is equivalent to

$$\begin{aligned} \frac{\partial F_1}{\partial x_1} \frac{\mathrm{d}x_1}{\mathrm{d}u} &= -\frac{\partial F_1}{\partial u} \\ \frac{\partial F_2}{\partial x_2} \frac{\mathrm{d}x_2}{\mathrm{d}u} &= -\frac{\partial F_2}{\partial u} - \frac{\partial F_2}{\partial x_1} \frac{\mathrm{d}x_1}{\mathrm{d}u} \\ &\vdots \\ \frac{\partial F_N}{\partial x_N} \frac{\mathrm{d}x_N}{\mathrm{d}u} &= -\frac{\partial F_N}{\partial u} - \frac{\partial F_N}{\partial x_{N-1}} \frac{dx_{N-1}}{\mathrm{d}u}. \end{aligned} \tag{1.59}$$

Note that $\frac{\mathrm{d}x_k}{\mathrm{d}u} \in \mathbb{R}^{n \times m}$ and hence we will need to save Nnm values in order to store $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}u}$. In both routines we have to solve N $n \times m$ linear systems but Algorithm 3 saves memory by only having to store n entries from λ_k in each iteration. Observe that in our reservoir models n is in the magnitude of the number of grid cells.

References This section about adjoints is inspired by Stein Krogstad's presentation in the Geilo Winterschool 2022 and [14].

1.6 Parameter Sensitivities and Calibrating Reservoir Models

Production optimization in reservoir simulation requires many full forward simulations. Reducing the order of the reservoir model (e.g. by partition the cells to coarser cells. That can be often a reduction from several million cells to hundreds of cells) decreases the computational cost tremendously while the forecasting ability can maintained to almost arbitrary accuracy through *calibration* of parameters in the reduced-order model. This calibration process can be done by data based machine learning tools or deep neural networks [6, 15, 34], but there is also a simpler approach to this end. We consider a simple parameterized box-constraint optimization problem

$$\min_{\zeta \in [0,1]^n} \quad f(\zeta) = \frac{1}{2} \|r(\zeta)\|^2 \tag{1.60}$$

where $r(\zeta)$ is the residual vector where each component represent the misfit of an outcome parameter from the reduced-order model with the same parameter in a fine-scale reference model.

Adjoint Equation. In Section 1.5 we have discussed a method to obtain the gradient of the objective function in a way where we solve a linear system with ℓ equations for each time step where ℓ is the dimension of a state in the reduced-order model. The choice of the method to obtain the gradient becomes important. Recall that the forward sensitivity equations (1.59) require to solve systems with m rows where

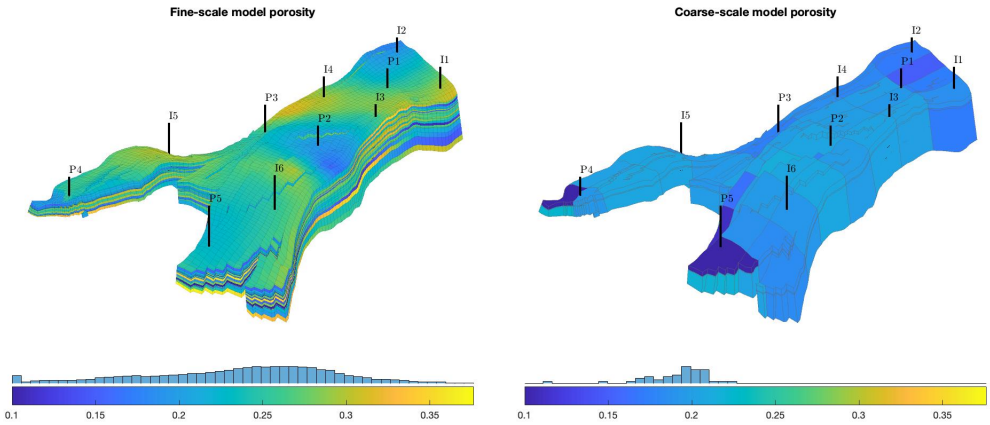


Figure 1.3: We reduced the dimension from 44915 to only 53 cells. The parameters can be for example pore volume on each cell or transmissibility.

m equals the number of parameters. Hence the adjoint-equations enable a more efficient way to assemble the gradients ∇r_i for the optimization iteration.

Chapter 1. The method of adjoint equation is used in In Chapter 3 we treat the many possible optimization methods for this purpose. Since our objective function is a non-linear least square function the Gauss–Newton methods can be used with a promising theoretical convergence advantage over the quasi–Newton methods.

Optimizer. There are many more consideration for the optimal choice of the optimizer. Often the notion of Gauss–Newton or quasi-Newton methods is misunderstood as full optimization methods. In fact, these names only determine the second-order coefficient matrix for the second-order approximation of the objective function. That is the matrix B in

$$q(p) = c + g^{\mathrm{T}} p + \frac{1}{2} p^{\mathrm{T}} B p$$

where usually $c = f(\zeta_0)$, $g = \nabla f(\zeta_0)$ and p indicates the search direction. Now we can choose between different paths. E.g. the classical Newton method emerges if $B = \nabla^2 f(\zeta_0)$ we assume $f(\zeta_0 + p) \approx q(p)$ and $\nabla f(\zeta_0 + p) = 0$. Then

$$0 = \nabla q(p) = g + Bp$$

gives the usual Newton step update formula. An advanced optimizer would then use a routine to use the ideal step length. Many of those involve a lot of function evaluations which is not computable feasible in our application. A totally different approach expects q to describe f precisely within a so-called trust-region. A simplified version of this trust-region minimization is explicitly solvable and hence the solution has a predefined step length. An iteration must be repeated only if the computed step is not a decrease. A detailed analysis will be given in Chapter 2

Example. In Fig. 1.3 we see the Norne field. The governing variables forming a state of the simulation are hence only the number of coarse grid cells times the number of governing variables.

2 Optimization Methods

This chapter provides a general comprehensive overview on optimization methods relevant for the optimization problem from (1.60) in Section 1.6. The parts are drafted based on the standard textbook of Nocedal and Wright [30] on optimization and they connect the descriptions with state of the art MATLAB implementations taken from [24, 10].

We start with a short recap of the setting in Section 1.6. In order to calibrate coarse grid models to access well production rate as accurate as the fine grid models does, we compare the well production rates of our coarse grid model with the the rates of the fine grid model in every time step. The resulting objective function is a residual vector where each time step and well is represented as one residual components, that is, we are concerned with solving the following optimization problem.
Therefore we are interested in solvers for box constrained non-linear least square problems, that is,

$$\min_{x\in[0,1]^n} \frac{1}{2}\|r(x)\|^2,$$

where $r(x) \in \mathbb{R}^m$ and all parameters are scaled. In our application, evaluating the objective function $f(x) = \frac{1}{2}\|r(x)\|^2$ comes with the simulation of a whole life cycle of the coarse grid model. That implies, in order to remain within feasible time complexity, we require a "small" number of function evaluation. Some of the standard optimization methods which we present in this chapter will not fall into this category. However, they are included to motivate and derive industry standards routines, used in e.g., the MATLAB Optimization Toolbox and also to establish more theoretical results.
In later sections, we restrict our analysis on iterative gradient-based optimization methods, since they are the main tools in MATLAB and MRST which have proven to be robust and efficient. First, we will derive the theoretical background on iteration methods of the form

$$x_{k+1} = x_k + \alpha_k p_k$$

where p_k is the search direction and α_k is the step length. We establish a criteria to show convergence for this general problem type and also consider more specific types such as $p_k = B_k^{-1}\nabla f(x_k)$ and $B_k \in \mathbb{R}^{n\times n}$ is a symmetric positive definite matrix, imitating the regular Newton method but doesn't require the computation of the Hessian matrix. While we repeat known algorithms such as the quasi-Newton BFGS method, Gauss–Newton method and the Levenberg–Marquardt method, we want to provide insight into the implementation of those methods in the MATLAB Optimization Toolbox and also discuss more practical issues.

Therefore, at the end of each section we will return to the above questions and provide remarks regarding implementation.

Figure 2 gives an overview on the requirements to understand the construction of the optimizers which we are considering. Section 2.1 and Section 2.2 provide the basic procedure of the line search method but also establish theoretical results

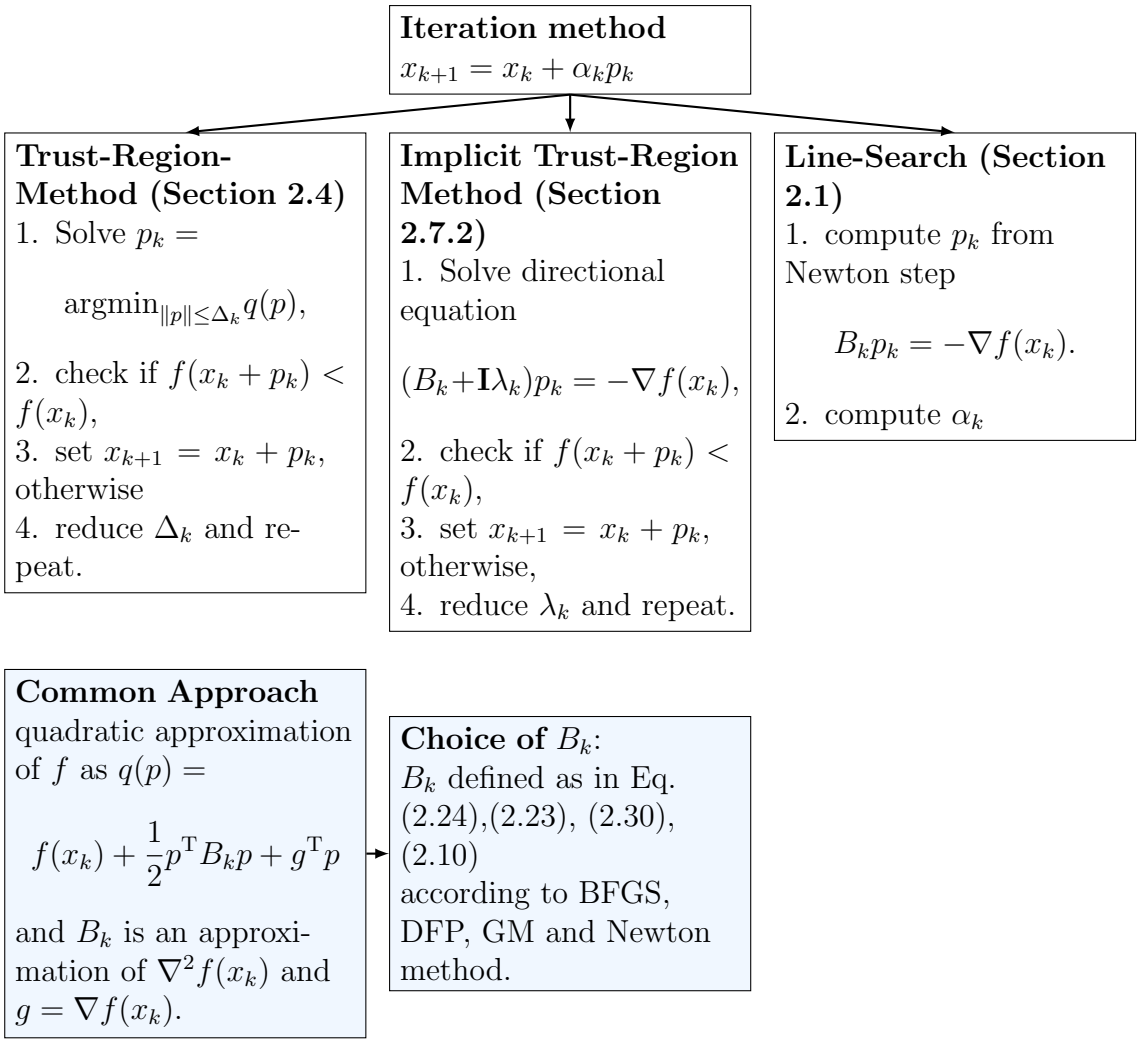


Figure 2.1: The different approaches presented in this work are all based on the approximation of f by a quadratic function. Often, the choices of B_k define the theoretical convergence rate, efficiency and stability. However, each choice can be paired with each of the outer schemes.

for more general iteration methods. Every section provides the derivation of the respective method (or class of methods) and provides a references to theoretical results regarding convergence and time complexity. At Section 2.8 we draw the lines back to Figure 2 and list the components of the algorithms from the MATLAB Optimization Toolbox and MRST based on their function documentation.

2.1 Line Search

The mathematical content of Section 2.1 and Section 2.2 follows [30, Chapter 03] and includes relevant information about the MATLAB Optimization Toolbox including remarks on implementation.

For Chapter 2 we will use the notation of [30]. Consider a twice differentiable function $f : \mathbb{R}^n \mapsto \mathbb{R}$ which is bounded from below. Let x^* be a local minimum of f. We will discuss the so-called line search methods to find the minimizer x^*. These

are iterative methods of the form

$$x_{k+1} = x_k + \alpha_k p_k \tag{2.1}$$

where p_k is the search direction and α_k is the step size. We call p_k a *descent direction* to f at x_k if p_k points in the negative direction of the gradient of f at x_k, that is,

$$\frac{\partial f(x_k)}{\partial p_k} = p_k^{\mathrm{T}} \nabla f(x_k) < 0.$$

Observe that if p_k is a descent method, we can find a step length α_k such that $f(x_{k+1}) < f(x_k)$. Suppose we can compute the gradient of f efficiently. We will show that choosing α_k with the so called Wolfe condition (2.2) results in a convergent line search method. Hence an efficient convergent method is simple to obtain theoretically. But how can we do better?
Naturally, if we have curvature information for each point x_k, that is, some information about the Hessian matrix $\nabla^2 f(x_k)$, the convergence rate of the resulting method can be become quadratic. More explicity, by the choice $p_k = -\nabla^2 f(x_k)^{-1} \nabla f(x_k)$ and $\alpha_k = 1$ we obtain the Newton–Raphson method with quadratic convergence rate in vicinity of a local minimum. Formally, this result follows by applying Taylor's formula on $\nabla f(x_k) - \nabla f(x^*)$ to obtain a second factor of $(x_k - x^*)$ where x^* is the local minimum (see Theorem 2.5). Intuitively, we recall that the Newton method is of the form

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

which sets x_{k+1} as the zero of the straight line

$$g_k(t) = f(x_k) + f'(x_k)(t - x_k),$$

that is, we represent f by its first order taylor approximation to find a zero of f. Geometrically, that means we expect the zero to be closer to x_k if the slope $f'(x_k)$ is large and vice versa. This expectation will be met if f is close to linear in the considered area. Turning back to our multidimensional case where we search for a local minimum, the Hessian matrix $\nabla^2 f(x_k)$ represents the gradient of $\nabla f(x_k)$. In vicinity of the zero of f it follows by the Taylor expansion of f that the error to the local minimum x^* of f which is in $\mathcal{O}(\|x_{k+1} - x^*\|^2)$ converges quadratically to zero. However, in practice we cannot make sure that we are close to a local minimum and what is the procedure if the Hessian matrix $\nabla^2 f(x_k)$ is singular? Also the Hessian matrix becomes expensive to compute in large systems emerging from, for instance, reservoir simulation. This performance issues were the motivation for Davidon in 1951, to develope a more reliable algorithm [7] and he established a new class of methods for iterative non-linear optimization solvers (see Section 2.6.1).
First we will focus on establishing the theory around iteration methods. Later, we discuss the influence of those historical accomplishments to the current industry standard for optimization solvers, used in MATLAB Optimization Toolbox.

2.2 Step Length and Wolfe Condition

Assume we have a descent step direction p_k, that is $\nabla f(x_k)^{\mathrm{T}} p_k < 0$, the Wolfe conditions for α_k are the following

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \nabla f(x_k)^{\mathrm{T}} p_k \alpha_k, \tag{2.2}$$
$$\nabla f(x_k + \alpha_k p_k)^{\mathrm{T}} p_k \geq c_2 \nabla f(x_k)^{\mathrm{T}} p_k, \tag{2.3}$$
$$\text{for } 0 < c_1 < c_2 < 1.$$

Let us define the level function $\phi(\alpha) := f(x_k + \alpha p_k)$. Then the derivative $\phi'(\alpha) = \nabla f(x_k + \alpha p_k)^{\mathrm{T}} p_k$ is negative for $\alpha = 0$ as p_k is a descent direction. We interprete the Wolfe condition in the two dimensional plane on the graph of ϕ. The first equation requires that the new iteration point $\phi(\alpha_k)$ is below a straight line which starts from $f(x_k)$ and has slope $c_1 \nabla f(x_k)^{\mathrm{T}} p_k \leq 0$. The second equation requires that the slope in p_k direction of f at x_{k+1} is greater than slope at x_k, such that the absolute value of the slope of ϕ is closer to zero. Combined with (2.2), we select a local minimum of $\phi(\alpha)$ that satisfies a decrease condition.

Why the Wolfe condition?

We have two reasons to choose these conditions. The first one is about convergence.

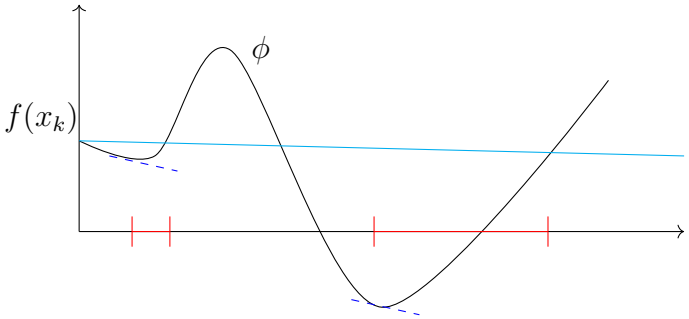


Figure 2.2: The first Wolfe condition is illustrated by the blue straight line. It requires the next step to be below this line. The second condition requires that $\phi'(x_{k+1}) = \nabla f(x_k + \alpha_k p_k)^{\mathrm{T}} p_k \geq c_2 \phi'(0)$ and is indicated by the dotted blue line. Since $\phi'(0)$ is negative $\phi'(x_{k+1})$ is required to be positive or closer to zero and hence x_{k+1} is closer to a local minimum. The red marked areas indicate where the Wolfe conditions are satisfied. Observe that, since p_k is a descent direction and $c_1 < 1$, the existence of an α_k satisfying the Wolfe condition is guaranteed.

2.3 Convergence with Wolfe Condition

Now assume an iterative method where we can provide a descent direction p_k which is bounded from being orthogonal to $\nabla f(x_k)$ and α_k is chosen according to the Wolfe condition (2.2). In this case Zoutendijk's theorem guarantees convergence for this method. Before we discuss the result, we define the angle between the search direction p_k and the negative of the gradient $-\nabla f(x_k)$.

$$\cos\theta_k = \frac{-\nabla f(x_k)^{\mathrm{T}} p_k}{\|\nabla f(x_k)\| \|p_k\|}.$$

Theorem 2.1 (Zoutendijk's Theorem):
Assume that f is twice differentiable, ∇f is Lipschitz continuous to bounding constant $L > 0$ and f is bounded from below by M. Further let p_k be a descent direction, that is,

$$p_k^{\mathrm{T}} \nabla f(x_k) < 0$$

and α_k satisfies the Wolfe conditions (2.2). Then

$$\sum_{k \geq 0} \cos^2 \theta_k \|\nabla f(x_k)\|^2 < \infty$$

and in particular $\cos^2 \theta_k \|\nabla f(x_k)\|^2 \to 0$ for $k \to \infty$.

For the proof see in the Appendix 4.1. We can formulate the following criteria for convergence:

Corollary 2.2:
Consider a line search method such that p_k is a descent direction and α_k satisfies the Wolfe condition (2.2) and (2.3).
If there is a $k_0 \in \mathbb{N}$ and $\delta > 0$ such that

$$\cos \theta_k \geq \delta \quad \text{for all } k \geq k_0,$$

then $(x_k)_{k \in \mathbb{N}}$ converges to a stationary point of f.

The second reason for using the Wolfe condition is: It is crucial for the well posedness of the sub-problem for the quasi-Newton methods BFGS and DFP. By imposing Inequality (2.3), we obtain directly the inequality

$$(\nabla f(x_{k+1}) - \nabla f(x_k))^{\mathrm{T}}(x_{k+1} - x_k) \geq \alpha_k(c_2 - 1)\nabla f(x_k)^{\mathrm{T}}(x_{k+1} - x_k) \geq 0.$$

The outer parts of this inequality are equal to the precondition Inequality (2.20) of the sub-problem from both, the BFGS and DFP method. Hence the secant equation Eq. (2.19) for B_{k+1} has a solution. This implies that B_k and H_k, respectively, are good approximation for the optimal choice for the second order Taylor approximation of f. For details consider Section 2.6.1. Moreover, quasi-Newton methods converge against a stationary point of f if the condition number of B_k is uniformly bounded. This can be stated as a corollary.

Corollary 2.3 (Convergences of quasi-Newton methods):
Let $p_k = -B_k \nabla f(x_k)$ for a symmetric matrix B_k such that the following requirements are met:

1. $\|B_k\| \|B_k\|^{-1} \leq M$ —the condition number of B_k is uniformly bounded by M.

2. the conditions of Zoutendijk's Theorem 1.1 are met.

Then the quasi-Newton method defined by α_k and B_k converges against a stationary point of f.

Proof. We only need to show that $\cos\theta_k$ is uniformly bounded away from zero. Hence consider

$$\cos\theta_k = \frac{\nabla f(x_k)^{\mathrm{T}} B_k^{-1} \nabla f(x_k)}{\|\nabla f(x_k)\| \|B_k^{-1} \nabla f(x_k)\|} \geq \frac{\nabla f(x_k)^{\mathrm{T}} B_k^{-1} \nabla f(x_k)}{\|\nabla f(x_k)\|^2 \|B_k\|^{-1}}.$$

The inequality is due to the properties of the operator norm. Now the numerator is a scalar product with a symmetric positive definite matrix. We have

$$\nabla f(x_k)^{\mathrm{T}} B_k^{-1} \nabla f(x_k) \geq \nabla f(x_k)^{\mathrm{T}} \nabla f(x_k) \frac{1}{\|B_k\|}$$

and thus we obtain an overall lower bound for $\cos\theta_k$ as

$$\cos\theta_k \geq \frac{\nabla f(x_k)^{\mathrm{T}} \nabla f(x_k)}{\|\nabla f(x_k)\|^2 \|B_k\| \|B_k\|^{-1}} = \frac{1}{\|B_k\| \|B_k\|^{-1}} \geq \frac{1}{M} > 0.$$

This concludes the proof by applying Zoutendijk's Theorem. □

To see a standard step length algorithm consider Appendix 4.1.

2.4 Trust-Region Methods

This section is inspired by Nocedal-Wright [30, Chapter 04]. In both iteration method schemes we consider the second order approximation of f

$$m_k(p) = f_k + g_k^{\mathrm{T}} p + \frac{1}{2} p^{\mathrm{T}} B_k p,$$

where $f_k = f(x_k)$, $g_k = \nabla f(x_k)$ and B_k is symmetric and an approximation of the Hessian matrix of f in x_k.

The step length is implicitly defined before each iteration by the choice of the parameter Δ_k, the trust-region radius. The step p_k is the minimizer of the following non-linear constrained quadratic program

$$\min_p \frac{1}{2} p^{\mathrm{T}} B_k p + g_k^{\mathrm{T}} p + f_k, \quad \|p\| \leq \Delta_k. \tag{2.4}$$

Note that the choice of the two parameter Δ_k and B_k define the method. Before we come to the solution of the subproblem (2.4), let us consider the algorithm. The algorithm 4 is taken from [5].

Δ_k is updated in a standard routine and in particular reduced when $f(x_k + p_k) \geq f(x_k)$ holds. Note that the reduction routine presented in Nocedal-Wright [30, Chapter 4] is not the method of choice when function evaluations of f are expensive.
In fact, `lsqnonlin` trust-region reflective routine is based on this simple scheme [24].
A simple question remains: *How do we solve the quadratic minimization problem (2.4)?* The following Theorem 2.4 is from [30, Theorem 4.1]. The proof is included by me.

Algorithm 4: Trust-region method

Input: x_0, f
Output: x_N approximation of a local minimum of f

$k := 0$;
while $\nabla f(x_k) \geq \epsilon$ **do**
  Solve (2.4) with solution p_k;
  if $f(x_k + p_k) < f(x_k)$ **then**
    $x_{k+1} = x_k + p_k$;
  Adjust Δ_k;
  $k := k + 1$;

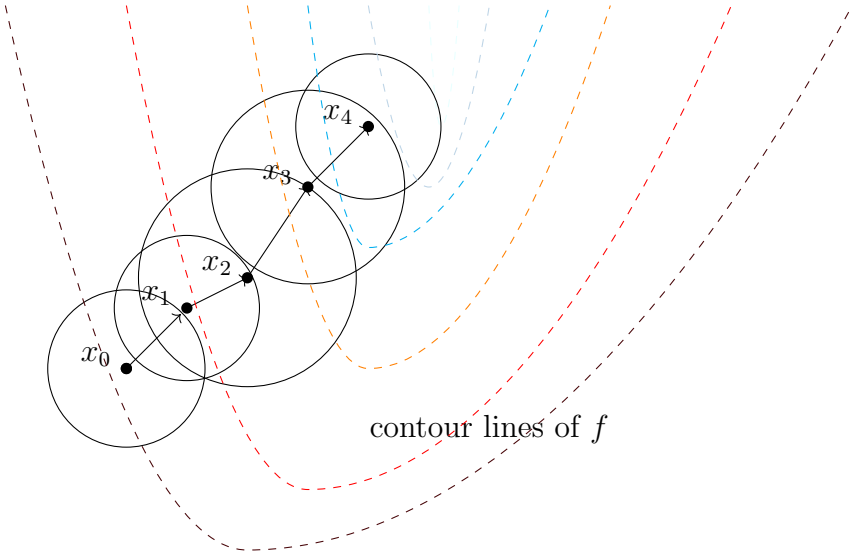


Figure 2.3: The trust-region method. In each iteration we represent $p \mapsto f(x_k + p)$ with a quadratic approximation $q(p) = f(x_k) + g^{\mathrm{T}}p + \frac{1}{2}p^{\mathrm{T}}B_k p$ and compute p as the minimizer of q in the trust-region indicated by the circle. The dark colored contour lines indicate high function values of f. Observe further that the adjustment of Δ_k depends on how accurate q approximates f. If q fails to generate a point with $f(x_{k+1}) < f(x_k)$ then Δ_k will be reduced and the iteration will be repeated.

Theorem 2.4:
Given a symmetric positive definite matrix $B \in \mathbb{R}^{n\times n}$, $g \in \mathbb{R}^n$, $c \in \mathbb{R}$ and $\Delta \geq 0$. Consider the quadratic function

$$\mathbf{m} : \mathbb{R}^n \to \mathbb{R}, p \mapsto \frac{1}{2}p^{\mathrm{T}}Bp + g^{\mathrm{T}}p + c$$

and the quadratic minimization problem

$$p^* = \arg\min_{p\leq\Delta} \mathbf{m}(p).$$

Then p^* satisfies one of the following conditions is satisfied:

1. $Bp^* = -g$, that is, the global minimizer is in the ball of radius Δ.
2. $\|p^*\| = \Delta$ and there is a $\lambda \geq 0$ with $\lambda p^* = -\nabla\mathbf{m}(p^*) = -Bp^* - g$. That is, the solution p^* lies on the sphere of radius Δ and points towards $-\nabla\mathbf{m}(p^*)$. This simplifies to the equation

$$(B + \lambda\mathbf{I})p^* = -g, \quad \lambda \geq 0. \tag{2.5}$$

Proof. We have

$$\nabla\mathbf{m}(p) = Bp + g$$

and

$$\nabla^2\mathbf{m}(p) = B$$

which is symmetric positive definite. Hence $\mathbf{m}$ has its unique minimum at p^* with $0 = \nabla\mathbf{m}(p^*) = Bp^* + g$.
If the solution to this equation is not in $K_\Delta(0) = \{x \in \mathbb{R}^n| \quad \|x\| \leq \Delta\}$ then no inner minimum can be attained. Consequently, $\|p^*\| = \Delta$ and by the Lagrangian multiplier on $h(p) := \frac{1}{2}(p_1^2 + p_2^2 + \ldots + p_n^2 - \Delta^2)$ and

$$\text{maximize } -\mathbf{m}(p) \tag{2.6}$$
$$\text{subject to: } h(p) = 0 \tag{2.7}$$

we obtain the condition $-Bp^* - g = -\nabla\mathbf{m}(p^*) = \lambda\nabla g(p^*) = \lambda p^*$ (the inner equation formalizes p^* points in the direction of steepest descent) for a global minimum on the sphere of radius Δ. This simplifies to

$$Bp^* + \lambda p^* = (B + \lambda\mathbf{I})p^* = -g$$

concluding the proof.

□

A minimizer p^* is consequently either the global minimum of $\mathbf{m}$ or it is collinear to the gradient of $\mathbf{m}$ and hence orthogonal to the contour lines of $\mathbf{m}$. Equivalently, we can formulate the necessary conditions for p^* from Theorem 1.4 as

1. $(B + \lambda\mathbf{I})p^* = -g$,
2. $\lambda(p - \|\Delta\|) = 0$.

From this theoretical basis we derive some methods to solve the quadratic minimization problem (2.4) which are used in the Matlab optimization toolbox.
We introduce the two-dimensional subspace minimization for which we consider the problem

$$\min_{p\in\mathbb{R}^n} \mathbf{m}(p) = c + g^T p + \frac{1}{2} p^T B p \quad \text{s.t. } \|p\| \leq \Delta, p \in \text{span}\left[g, B^{-1}g\right]. \tag{2.8}$$

It is cheap to compute a good approximation of the former. For instance, this problem can be reduced to finding roots of a polynomial in one variable of degree four.
Why is the choice of those two directions advantageous?
Solving the quadratic minimization problem in the direction g yields the Cauchy point method (see below), which under certain (basic) choices of Δ and B_k is globally convergent. The direction $-B^{-1}p$ is simply the direction towards the global minimum of $\mathbf{m}$ (assuming B is positive definite) that turns out to be a promising choice.
Global Convergence
For the sub problem restricted to one dimension, that is,

$$\min_{p} \mathbf{m}(p) = f + g^T p + \frac{1}{2} p^T B p \quad \text{s.t. } \|p\| \leq \Delta, p \in \text{span}\left[g\right] \tag{2.9}$$

and use a standard but adequate routine for the adaptation of Δ, we can already prove global convergence for the trust-region method Algorithm 4. In this case, we can identify the solution of the sub problem (2.9) with the Cauchy point which is simply

$$p^C = -\frac{g^{\mathrm{T}} g}{g^{\mathrm{T}} B g} g.$$

Hence the trust-region method using the two dimensional subspace method as a subroutine converges globally. See in [30, Section 4.2] for a detailed proof.

2.5 Newton Method

The theoretical background for Section 2.6.1 and Section 2.5 and for algorithm were inspired by [30, Chapter 3, Chapter 6]. The Newton method is defined as

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k). \tag{2.10}$$

The Newton method is in general not a descent method.

$$\nabla f(x_k)^{\mathrm{T}} p_k = -\nabla f(x_k)^{\mathrm{T}} \nabla^2 f(x_k)^{-1} \nabla f(x_k) < 0$$

is not necessarily satisfied if $\nabla^2 f(x_k)$ is not positive definite. Hence we might even increase along p_k. However, since x^* is a minimum, $\nabla^2 f(x^*)$ is positive definite in vicinity of x^*. We obtain the following result

Theorem 2.5 (Newton method):
Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be a twice continuous differentiable and $x \mapsto \nabla^2 f(x)$ Lipschitz–continuous to constant $L > 0$. Further let x_0 be close enough to x^* such that $\nabla^2 f(x)$ is positive definite. Then

1. the Newton method (2.10) will converge to x^* with a quadratic convergence rate, that is,
$$||x_{k+1} - x^*|| = \mathcal{O}(||x_k - x^*||^2)$$
for all $k \geq 0$.
2. $\|\nabla f(x_k)\|$ converges to zero.

Proof. The proof is straight forward by using Taylor's formula. That is, the line of the vector field $t \mapsto \nabla f(x + t(y - x))$ derives to $\nabla^2 f(x + t(y - x))(y - x)$. Hence

$$\nabla f(x) - \nabla f(y) = \int_0^1 \nabla^2 f(x + t(y - x))(y - x)\mathrm{d}t,$$

and consequently by setting $\nabla^2 f(x_k) = H$, we obtain

$$\begin{aligned}
\|x_{k+1} - x^*\| &= \|x_k + p_k - x^*\| \\
&= \|x_k - x^* - (\nabla^2 f(x_k))^{-1} \nabla f(x_k)\| \\
&= \| \left(\nabla^2 f(x_k)\right)^{-1} \left[\nabla^2 f(x_k)(x_k - x^*) - (\nabla f(x_k) - \nabla f(x^*))\right] \| \\
&= \|H^{-1} \int_0^1 H(x_k - x^*) - \nabla^2 f(x^* + t(x_k - x^*))(x_k - x^*)\mathrm{d}t\| \\
&\leq \|H^{-1}\| \int_0^1 \left\|H - \nabla^2 f(x^* + t(x_k - x^*))\right\| \mathrm{d}t \|x_k - x^*\| \\
&\leq \|H^{-1}\| \int_0^1 Lt\|x_k - x^*\|\mathrm{d}t \|x_k - x^*\| \\
&= C\|x_k - x^*\|^2
\end{aligned}$$

where $C = \|H^{-1}\|L$. Hence $\|x_{k+1} - x^*\| = \mathcal{O}(\|x_k - x^*\|^2)$.
Moreover,

$$\begin{aligned}
\|\nabla f(x_k)\| = \|\nabla f(x_k) - \nabla f(x^*)\| &= \int_0^1 \nabla^2 f(x_k + t(x^* - x_k))(x^* - x_k)\mathrm{d}t \quad (2.11) \\
&\leq L\|x_k - x^*\|^2 \quad (2.12)
\end{aligned}$$

With the assumptions that x_0 is close enough to x^* we can conclude the statements. □

2.6 Quasi-Newton Method

We want to provide an overview over convergence analysis of quasi-Newton methods. First, we need the following definition.

Definition 2.6:
Let $(X, \|\cdot\|)$ be a normed space. A convergent sequence $(x_k)_{k\in\mathbb{N}} \subseteq X$ with limit $x^* \in X$ converges superlinear if there exist a sequence $(c_k)_{k\in\mathbb{N}} \subseteq \mathbb{R}$, converging towards zero, such that

$$\|x_{k+1} - x^*\| \leq c_k \|x_k - x^*\|. \tag{2.13}$$

We denote this as

$$\|x_{k+1} - x^*\| = o(\|x_k - x^*\|).$$

Theorem 2.7:
Consider the iteration $x_{k+1} = x_k + \alpha_k p_k$, where p_k is a descent direction and α_k satisfies the Wolfe conditions with $c_1 \leq 1/2$. If the sequence $(x_k)_k$ converges to a point x^* such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, and if the search direction satisfies

$$\lim_{k\to\infty} \frac{\|\nabla f_k + \nabla^2 f_k p_k\|}{\|p_k\|} = 0$$

then

(i) the step length $\alpha_k = 1$ is admissible for all k greater than a certain index k_0

(ii) if $\alpha_k = 1$ for all $k > k_0$, $\{x_k\}$ converges to x^* superlinearly.

If we apply this to the quasi-Newton method we obtain the result

Theorem 2.8:
Consider the iteration $x_{k+1} = x_k + B_k \nabla f(x_k)$. Let us assume also that $(x_k)_k$ converges to a point x^* such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then $(x_k)_k$ converges superlinearly if and only if holds

$$\lim_{k\to\infty} \frac{\|(B_k - \nabla^2 f(x^*))\, p_k\|}{\|p_k\|} = 0. \tag{2.14}$$

These results are proven in [30, Theorem 3.6] and provide one way to show global superlinear convergence of the two methods BFGS and DFP. The general convergence is shown by using Zoutendjik's Theorem 2.1, and hence combined with Theorem 2.8, we will come to vicinity of x^* and obtain superlinear convergence of $(x_k)_{k\in\mathbb{N}}$ with step length 1.
For details see [30, Chapter 6.4]. Moreover, as we will see in the next section, Theorem 2.8 provides an approach to construct reasonable quasi-Newton methods.

2.6.1 BFGS and DFP

In 1951, W.C. Davidon, a computational physicist at Argonne National Laboratory, proposed the first quasi-Newton method similar to the one presented in this section. Frustrated over long computation times and repeated computation crashes,

he developed a more stable and efficient method than the Newton method. With its simplicity, robustness and efficiency it has stood the test of time and the DFP (Davidon-Fletcher-Powell) method, described in [7], and the very similar BFGS (Broyden-Fletcher-Goldfarb-Shannon), developed in 1970 independently from its eponyms [2, 9, 11, 35], are yet state of the art for most general applications. In fact the standard optimization method from MATLAB `fmincon` is based on the BFGS method.
Let us discuss the derivation of the method. Assume that B_k, p_k and α_k define an iteration method such that

$$x_{k+1} = x_k + \alpha_k p_k$$

and B_k is used to determine p_k and α_k is determined afterwards.
Both DFP and BFGS are based on a generalized form of the secant method. However, BFGS updates the inverted curvature matrix and hence the computational complexity in each iteration of BFGS is in $\mathcal{O}(n^2)$, compared to $\mathcal{O}(n^3)$ in Newton's method.
We first observe that if

$$\lim_{k\to\infty} \frac{\|(B_k - \nabla^2 f\,(x^*))\, p_k\|}{\|p_k\|} = 0 \tag{2.15}$$

then our iteration method converges superlinearly by Theorem 2.8 even if $\alpha_k = 1$ for all k. In particular, B_k is an approximation of the Hessian matrix of f in the direction p_k. But how can we find such an approximation? As suggested by Eq. (2.15), B_k should be chosen as an approximate of $\nabla^2 f(x_k)$, provided that x_k converges towards x^*. In order to do so Davidon considered the modified function of the second order Taylor approximation of f. That is, the quadratic function

$$m_k : p \mapsto f(x_k) + \nabla f(x_k)^{\mathrm{T}} p + \frac{1}{2} p^{\mathrm{T}} B_k p.$$

If B_k is a good approximation of the Hessian $\nabla^2 f(x_k)$ this function becomes the second order Taylor approximation of f. In order to compute B_{k+1} from B_k, it is now reasonable to interpolate ∇m_{k+1} at the position x_{k+1} and x_k. That is equivalent to require B_k must approximate $\nabla^2 f(x_k)$ in the direction of p. The latter is achieved by requiring

$$\nabla m_{k+1}(0) = \nabla f(x_{k+1}), \tag{2.16}$$
$$\nabla m_{k+1}(-\alpha_k p_k) = \nabla f(x_k). \tag{2.17}$$

The second equation simplifies to

$$\nabla f(x_{k+1}) - B_k(-\alpha_k p_k) = \nabla f(x_k) \tag{2.18}$$

and gives our condition on B_k. We call it the secant equation

$$B_k(-\alpha_k p_k) = \nabla f(x_k) - \nabla f(x_{k+1}). \tag{2.19}$$

Observe that $-\alpha_k p_k = x_k - x_{k+1}$. The constraint that B_k should be symmetric positive definite implies that we require

$$(x_k - x_{k+1})^{\mathrm{T}} B_k (x_k - x_{k+1}) = (x_k - x_{k+1})^{\mathrm{T}} (\nabla f(x_k) - \nabla f(x_{k+1})) \geq 0. \tag{2.20}$$

In general, if f is convex this equation is always satisfied and the Eq. (2.18) has a guaranteed solution. However, if the Wolfe conditions is satisfied we also have Eq. (2.20), which follows immediately by the curvature condition, Inequality (4.4). Thus, if we use the Wolfe condition, the existence of B_k is assured.
To achieve uniqueness consider the minimization formulation

$$\min_{B} \quad \|B - B_k\| \tag{2.21}$$

$$\text{subject to} \quad B(x_k - x_{k+1}) = \nabla f(x_k) - \nabla f(x_{k+1}). \tag{2.22}$$

The formulation aims to minimize the step size. B_{k+1} will be set as the unique solution to above problem. In order to obtain a non-dimensional scaling, we make use of the weighted Frobenius norm $\|B\|_W = \|W^{1/2}BW^{1/2}\|_F$ with weight matrix

$$W^{-1} = G_k := \left[\int_0^1 \nabla^2 f(x_k + \tau\alpha_k p_k)\mathrm{d}\tau\right].$$

From Taylor's Theorem we have $G_k(x_k - x_{k+1}) = \nabla f(x_k) - \nabla f(x_{k+1})$ that implies

$$[\left(G_k^{-1/2}(B - B_k)G_k^{-1/2}\right)_{ij}] = [1]$$

for all entry indices i, j and thus the norm is non-dimensional, that is,

$$[\|B\|_{G_k^{-1}}] = [1].$$

For simplicity, we write $s_k = x_k - x_{k+1}$ and $y_k = \nabla f(x_k) - \nabla f(x_{k+1})$. One closed form solution can be determined as

$$\text{(DFP)} \quad B_{k+1} = \left(\mathbf{I} - \rho_k y_k s_k^T\right) B_k \left(\mathbf{I} - \rho_k s_k y_k^T\right) + \rho_k y_k y_k^T, \tag{2.23}$$

$$\rho_k = \frac{1}{y_k^{\mathrm{T}} s_k}.$$

Observe that s_k lies in the kernel of $(\mathbf{I} - \rho_k s_k y_k^{\mathrm{T}})$, because $\rho_k s_k y_k^{\mathrm{T}} s_k = s_k$ and that $\rho_k y_k y_k^{\mathrm{T}} s_k = y_k$ and thus

$$B_{k+1} s_k = y_k$$

as required. Using the Sherman-Morrison-Woodbury Formula, we obtain

$$\text{(BFGS)} \quad H_{k+1} = \left(\mathbf{I} - \rho_k s_k y_k^T\right) H_k \left(\mathbf{I} - \rho_k y_k s_k^T\right) + \rho_k s_k s_k^T. \tag{2.24}$$

The algorithm can be formulated as follows:
For Algorithm 5, an effective heuristic is to scale H_0 after the first iteration before the first BFGS update is performed. If we start with $H_0 = \mathbf{I}$ then set

$$H_0 \leftarrow \frac{y_k^{\mathrm{T}} s_k}{y_k^{\mathrm{T}} y_k}\mathbf{I}.$$

The factor in front of $\mathbf{I}$ is an approximation of the smallest eigenvalue of $\nabla^2 f(x_0)^{-1}$. Each iteration has running time in $\mathcal{O}(n^2)$. The overall superlinear convergence is proved in [30, Chapter 6.4]. The general advantage of BFGS is its robustness and low cost for each iteration, while having a steady convergence property. In fact, it has self-correcting properties. If there are miss estimations of the curvature of the objective function, it will likely correct itself in a few steps (provided the step length satisfies the Wolfe conditions).

Algorithm 5: BFGS

Input: x_0, convergence tolerance $\epsilon > 0$, inverse Hessian approx. H_0
Output: x_k approximate stationary point of f

```
k ← 0;
while ‖∇f(x_k) > ε do
    p_k = −H_k ∇f_k;
    Determine α_k;
    x_{k+1} = x_k + α_k p_k;
    s_k = x_{k+1} − x_k;
    y_k = ∇f_{k+1} − ∇f_k;
    Compute H_{k+1};
    k ← k + 1;
```

2.7 Algorithms For Nonlinear Least-Squares-Problems

The theory behind the following sections about the Gauss–Newton and Levenberg–Marquardt method is inspired by [30, Chapter 10].
In least-square problems, the objective function f has the form

$$f : \mathbb{R}^n \to \mathbb{R}, x \mapsto \frac{1}{2}\sum_{j=1}^{m} r_j^2(x), \tag{2.25}$$

where each r_j is a smooth function from $\mathbb{R}^n$ to $\mathbb{R}$. Least-square problems are the largest source of unconstrained optimization problems. Often they are used for parameterized problems in a chemical, physical or financial, or economic application. To measure the discrepancy between the model and the observed behavior of the system, f is often minimized to obtain optimal parameter values for the model.
Rewriting r_j (2.25) to a residual vector $r : \mathbb{R}^n \to \mathbb{R}^m, r(x) = (r_1(x), \ldots, r_m(x))^{\mathrm{T}}$, we obtain the equivalent form

$$f = \frac{1}{2}\left\|r(x)\right\|^2.$$

Now, let

$$J(x) = \begin{pmatrix} \nabla r_1(x)^{\mathrm{T}} \\ \vdots \\ \nabla r_m(x)^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{m\times n} \tag{2.26}$$

be the Jacobian of the residual vector r. Then

$$\nabla f(x) = \sum_{j=1}^{m} r_j(x)\nabla r_j(x) = J(x)^T r(x), \tag{2.27}$$

$$\nabla^2 f(x) = \sum_{j=1}^{m} \nabla r_j(x)\nabla r_j(x)^T + \sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x) \tag{2.28}$$

$$= J(x)^T J(x) + \sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x). \tag{2.29}$$

In many applications J is cheap to compute. Therefore, the first term $J(x)^{\mathrm{T}}J(x)$ of the Hessian of f is efficiently computable. $J^{\mathrm{T}}J$ is a good approximation as long as r_j's are close to affine and small such that $\sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x)$ is small.

2.7.1 Gauss–Newton Method

Consider a function f of the form from Eq. (2.25). In Gauss–Newton, the Newton method equation for the search direction defined with the Hessian matrix $\nabla^2 f(x_k)$ by $\nabla^2 f(x_k)p = -\nabla f(x_k)$ simplifies into

$$J(x_k)^{\mathrm{T}} J(x_k) p_k = -J(x_k)^{\mathrm{T}} r(x_k). \tag{2.30}$$

Here we exploited the structure of $\nabla^2 f(x_k)$ and $\nabla f(x_k)$ from (2.29), neglecting the residual term $\sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x)$.
The first advantages of Gauss–Newton over the Newton method is that the calculation of J is cheap. In contrast, $\nabla^2 r_j$ is expensive to compute. If r is approximately affine, then $J^{\mathrm{T}}J$ is a good approximation and we obtain a quadratic convergence rate. Note also that if $J(x_k)$ has full rank, p_k is a descent direction. This follows from

$$(p_k)^{\mathrm{T}} \nabla f(x_k) = (p_k)^{\mathrm{T}} J(x_k)^{\mathrm{T}} r(x_k) = -(p_k)^{\mathrm{T}} J(x_k)^{\mathrm{T}} J(x_k) p_k = -\|J(x_k)p_k\|^2 \leq 0.$$

Implementation remarks: If the number residuals m is large while the dimension n is small, it might be useful to compute

$$J(x_k)^{\mathrm{T}} J(x_k) = \sum_{i=1}^{m} (\nabla r_i(x_k))(\nabla r_i(x_k))^{\mathrm{T}}, \quad J(x_k)^{\mathrm{T}} r(x_k) = \sum_{i=1}^{m} r_i(x_k)\nabla r_i(x_k) \tag{2.31}$$

successively, since $J(x_k)^{\mathrm{T}}J(x_k) \in \mathbb{R}^{n\times n}$ and $J(x_k)^{\mathrm{T}} r(x_k) \in \mathbb{R}^n$.
In practice the Gauss–Newton method has nearly quadratic convergence rate when the eigenvalues of $J(x_k)^{\mathrm{T}}J(x_k)$ are large compared to the term $|r_j(x)|\|\nabla^2 r_j(x)\|$. However, if $J(x_k)^{\mathrm{T}}J(x_k)$ is only semi-definite (2.30) is not always solvable. A more sophisticated implementation based on the directional equation (2.30) was provided by Levenberg and Marquardt.

Convergence of Gauss–Newton Method: To see that Gauss–Newton Method has nearly quadratic convergence rate, we exploit the fact that $J(x_k)^{\mathrm{T}}J(x_k)$ is a sufficiently good approximation of $\nabla^2 f(x_k)$. This is the case if x_k is in vicinity of x^*. Essentially, the proof is the same as for the Newton method 5. For simplicity let $J = J(x_k)$. The line of the vector field $t \mapsto \nabla f(x + t(y - x))$ derives to $\nabla^2 f(x + t(y - x))(y - x)$. Therefore

$$\begin{aligned} \nabla f(x_k) - \nabla f(x^*) = &\int_0^1 J^T J\left(x^* + t\left(x_k - x^*\right)\right)\left(x_k - x^*\right) \mathrm{d}t \\ &+ \int_0^1 H\left(x^* + t\left(x_k - x^*\right)\right)\left(x_k - x^*\right) \mathrm{d}t \end{aligned}$$

where $H(x)$ denotes the second-order term in of $\nabla^2 f(x_k)$. We then have that $x_{k+1} - x^*$ is equal to

$$\begin{aligned} x_k + p_k - x^* &= x_k - x^* - \left[J^T J\right]^{-1} \nabla f\left(x_k\right) \\ &= \left[J^T J\right]^{-1} \left[J^T J\left(x_k - x^*\right) + \nabla f\left(x^*\right) - \nabla f\left(x_k\right)\right]. \end{aligned}$$

Assuming Lipschitz continuity of $H(\cdot)$ near x^*, shows that

$$\begin{aligned}\|x_{k+1} - x^*\| &= \|x_k + p_k^{\mathrm{GN}} - x^*\| \\ &\leq \int_0^1 \left\| \left[J^T J\right]^{-1} H\left(x^* + t\left(x_k - x^*\right)\right) \right\| \|x_k - x^*\| \, \mathrm{d}t + \mathcal{O}\left(\|x_k - x^*\|^2\right) \\ &\approx \left\| \left[J^T J\right]^{-1} H\left(x^*\right) \right\| \|x_k - x^*\| + \mathcal{O}\left(\|x_k - x^*\|^2\right).\end{aligned}$$

If $H(x^*) = 0$ the convergence rate is quadratic.

2.7.2 Levenberg–Marquardt Method

We use a similar approximation as in Gauss–Newton method but replace the line search by a trust-region strategy. Thus we avoid one of the weak points of Gauss–Newton method: Jacobian matrix rank-deficiency. In order to derive the method, let us first consider the sub problem

$$\min_p \frac{1}{2} \|J_k p + r_k\|^2, \tag{2.32}$$

$$\text{subject to} \quad \|p\| \leq \Delta_k \tag{2.33}$$

where Δ_k is the trust region radius. Further let us focus on the sub problem (2.32) and drop the index k in the following. By Theorem 1.4 p^* is a solution of (2.32) if and only if there is a $\lambda \geq 0$ such that

$$(J^{\mathrm{T}} J + \lambda \mathbf{I}) p^* = -J^{\mathrm{T}} r \tag{2.34}$$

and 2

$$\lambda(\|p^*\| - \Delta) = 0.$$

Observe that (2.34) is always solvable for almost all $\lambda \in \mathbb{R}$.

The routine implemented in Matlab simplifies the idea above. Instead of choosing Δ_k and using a method to solve the sub problem 2.32, we choose the damping factor λ_k and adjust it in each iteration. It is closely related to the value of Δ_k. That is, if Δ_k tends to zero then λ_k tends towards infinity. The solution of (2.32) is then close to the steepest descent direction, with magnitude tending towards zero. On the other hand, if λ_k is small the solution p_k tends to be the global minimizer of (2.32) implying that the corresponding Δ_k is large. The following algorithm is inspired from [19].

If λ_k is close to zero then p_k is close to a global minimizer of (2.32). On the other hand, as λ_k tends to infinity, p_k tends to the steepest descent direction with magnitude tending towards zero. Consequently, for some sufficiently large λ_k

$$f(x_k + p_k) < f(x_k)$$

holds.

Implementation remarks: We have two function evaluations per iteration. In addition, we need the Jacobian matrix J explicitly. To be efficient, providing J during the computation of r or f is crucial. Here, the dimension of J determines the complexity of each iteration.
Further, observe that the gradient $\nabla f(x_{k+1})$ can be stored during the evaluation of $f(x_k + p_k)$, hence no additional evaluation of f is required.

Algorithm 6: Levenberg–Marquardt method

Input: x_0, $f = \frac{1}{2}\|r\|^2$, tolerance $\epsilon > 0$
Output: x_k approximated stationary point of f

$\lambda_0 \leftarrow 0.01$;
$k \leftarrow 0$;
while $\|\nabla f(x_k)\| \geq \epsilon$ **do**
    Try to solve $(J(x_k)^{\mathrm{T}} J(x_k) + \lambda_k \mathbf{I})p_k = -J(x_k)^{\mathrm{T}} r(x_k)$;
    if p_k *is updated* ***and*** $f(x_k + p_k) < f(x_k)$ **then**
        $\lambda_{k+1} = \lambda_k / 10$;
        $x_{k+1} = x_k + p_k$;
        $k \leftarrow k + 1$;
    else
        $\lambda_{k+1} = \lambda_k \cdot 10$;
        $x_{k+1} = x_k$;
        $k \leftarrow k + 1$;
return x_k;

2.8 General Implementation Remarks

Which algorithms are used in `fmincon` *and* `lsqnonlin`*?*
`fmincon` has the option to use several algorithms. The default algorithm is the interior-point algorithm [30, Chapter 14] for some explanation and Karmarkar's algorithm from 1984 [17] which outperformed the simplex algorithm to solve linear programming problems. It has been further developed to a non-convex optimization solver and is now among the best performing constrained optimization algorithms [30, Chapter 19]. In MATLAB's function `fmincon` the interior point algorithm is split in two possible iterations [10]. Either it computes a step directly from solving the Karush-Kuhn-Tucker equations or it uses the trust-region method. Either way, in our case we have no further constraints than the box constraint $x \in [0, 1]^n$, and the algorithm results in a simple trust-region method with B_k defined as in the BFGS method Eq. (2.24) as default. `lsqnonlin` has as a default algorithm the trust-region method which uses for B_k the matrix according to Eq. (2.30). And to solve the quadratic minimization problem, it uses the two-dimensional subspace method (Section 2.4) . `lsqnonlin` has as a second option the Levenberg–Marquardt method which is fully described in Algorithm 6. We categorize it to the implicit trust-region methods.

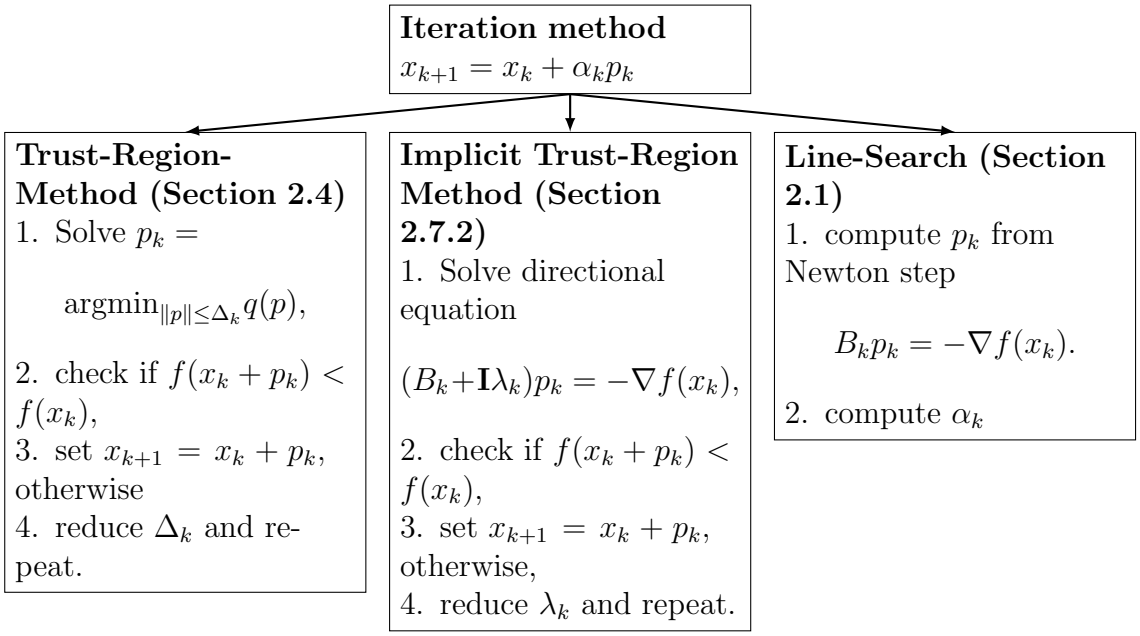


Figure 2.4:
Implementations
`fmincon` uses trust-region method with BFGS (for the interior-point and trust-region method)
`lsqnonlin` uses trust-region method or implicit trust-region method with B_k as in Eq. 2.30.
`unitBoxBFGS` (MRST) uses line search with BFGS.
`unitBoxLM` (MRST) uses implicit trust-region method with B_k as in Eq. 2.30 (the damping factor λ is adapted more flexible)

3 Tuning Reduced-Order Models in MRST

The model equations and the discretization techniques discussed earlier in this chapter are all implemented in the MATLAB Reservoir Simulation Toolbox (MRST) [26], which is a comprehensive and open-source toolbox for reservoir modeling and simulation, developed by the Computational Geosciences group in the Department of Mathematics and Cybernetics at SINTEF Digital. The software also includes an automatic differentiation library aimed to be simple to use and efficient for vector quantities (which in many ways are the cornerstone idea in MATLAB). MRST also contains an object-oriented framework that implements discrete differential operators, a class hierarchy of physical model of increasing complexity, efficient linear and nonlinear solvers, and gradient-based iterative optimizers, to name a few.

An extensive introduction to reservoir modeling and the usage of MRST is documented in Lie's book [22], with more details on advanced usage documented in a follow-up volume [23].

In this last chapter we want to connect the theory from the previous chapters in concrete calibrations of reduced-order models (see Section 1.6). We will present results from numerical experiments to compare coarse-scale model calibration of Gauss–Newton methods with the quasi-Newton method BFGS. First, we need to understand first how a reservoir model is incorporated in MRST and how optimization is performed.

The chapter is divided into four sections. First, we discuss a code snippet in MRST about this calibration process. Then we provide the data about the considered models. This includes, the size, important properties and the parameters on which we are performing the optimization. Then we introduce the relaxations of the Gauss–Newton methods and explain its expected behaviour. And afterwards we present the results of the calibrations with all optimizer of a natural subclass of relaxations of the Gauss–Newton methods and BFGS. This will be data of 63 calibrations with each 30 iteration.

3.1 Implementation of Reduced-Order Models

How to generate a porous media subsurface flow model and simulate the model in MRST?

We consider this process in an example. First we compute a grid describing a 400 $\times$ 400 m domain using a 50 $\times$ 50 Cartesian grid, which through its geometry defines the discrete gradient and divergence operator.

```
nxyz = [ 50,  50,  1];
Dxyz = [400, 400, 10];
rng(0)
G    = computeGeometry(cartGrid(nxyz, Dxyz));
rock = getSPE10rock(1:nxyz(1), (1:nxyz(2)), 1:nxyz(3));
rock.poro = max(rock.poro, 0.1);
```

In this case we have a Cartesian grid and hence the transmissibility, which is connecting the flux to the pressure drop, is a scalar. The function `computeGeometry` is crucial to determine the matrix for the discrete divergence operator. The rock porosity is taken from `SPE10`—a complex rock structure with a realistic porosity distribution. To maintain all cells active, we set the minimum porosity to 0.1. Next, we add production and injection wells in opposite corners.

```
%% wells/schedule
W = [];
% Injectors (lower-left and upper-right)
[wx, wy] = deal([1, nxyz(1)], [1, nxyz(2)]);
for k  = 1:2
    W = verticalWell(W, G, rock, wx(k), wy(k), 1:nxyz(3), ...
                     'Type' , 'rate', 'Val', 300*meter^3/day,...
                     'Name', sprintf('I%d', k), ...
                     'comp_i', [1 0], 'Sign' , 1);
end
% Producers (upper-left and -right)
[wx, wy] = deal([1, nxyz(1)], [nxyz(2), 1]);
for k  = 1:2
    W = verticalWell(W, G, rock, wx(k), wy(k), 1:nxyz(3),
                     'Type' , 'bhp',v'Val', 100*barsa,...
                     'Name', sprintf('P%d', k), ...
                     'comp_i', [1 0], 'Sign' , -1);
end
```

The producers are set to operate at a constant bottom-hole pressure ('type' = 'bhp'), in which case the value 'Val' has units bar. The injectors are set to operate at a fixed injection rate ('type' = 'rate'), and the value 'Val' is given in [m^3/s].

The fluid is a structure of up to three phases (water, oil, gas). `initSimpleADIFluid` defines a fluid with required parameters `'phases', WOG` defining the phases (here water, oil and gas are present) and the density ρ. For instance, if the have the phases `WOG` we initialize the density by `[1000, 700, 100]*kilogram/meter^3` meaning that water, oil and gas has densities of 1000 kg/m^3,700kg/m^3 and 100kg/m^3, respectively. The lines

```
pRef  = 200*barsa;
fluid = initSimpleADIFluid('phases', 'WO',...
                           'mu' , [.3, 3]*centi*poise,...
                           'rho', [1014, 859]*kilogram/meter^3, ...
                           'n', [2 2]);
c = 5e-5/barsa;
p_ref = 200*barsa;
fluid.bO = @(p) exp((p - p_ref)*c);
modelRef = GenericBlackOilModel(G, rock, fluid, 'gas', false);
```

creates a fluid with phases water and oil. The first entry of each vector defines the values of the options for water and the second entry defines the option for oil. Relative permeability is understood as the shift in permeability due to different saturation value. The rock might be less permeable for oil due to high saturation value

of water. Usually, the relative permeability is modelled as a monomial function in terms of the saturation. n is the degree of the monomial function modeling the relative permeability of the fluid. The input c defines the (constant) fluid compressibility. In the case of the absence of large amounts of gas in the fluid it can be assumed to be constant. Then `flud.bO` is the relation for density with pressure from Eq. (1.8).

```
% Set up 4 control-steps each 150 days
scheduleRef = simpleSchedule(...
rampupTimesteps(2*year, 30*day, 5), 'W', W);

%% run reference simulation
stateInitRef = initState(G, W, 200*barsa, [0, 1]);
modelRef.toleranceCNV = 1e-8;
[wsRef, statesRef] =  ...
simulateScheduleAD(stateInitRef, modelRef, scheduleRef);
```

For the simulation we set up the time discretization in a straight forward way, where we consider a total time span of 2 years. Last but not least, the simulation function `simluateScheduleAD` uses the discretized divergence and gradient operator to obtain a solution for each time steps. Then a time discretization with implicit Euler is set up and solved by Newton–Raphson method. The output from each time step is a vector of pressure and saturation for each cell in the simulation grid, in addition to well production rates and bottom-hole pressures.

How do we set up a coarse grid model and parameters to perform adjoint-based training?

MRST provides a framework to generate a coarse grid from a regular grid in a very intuitive way. Here we just reshape Cartesian grid cells to match the new dimension $3 \times 5 \times 1$.

```
%% make a coarse model and run
p     = partitionCartGrid(modelRef.G.cartDims, [3 4 1]);
model = upscaleModelTPFA(modelRef, p);
model.toleranceCNV = 1e-6;
schedule = upscaleSchedule(model, scheduleRef);

stateInit = upscaleState(model, modelRef, stateInitRef);
[ws0, states0] = simulateScheduleAD(stateInit, model, schedule);
```

To train our coarse grid model, we define parameters which we want to optimize. While the transmissibility in the fine grid reference model was computed using cell geometries and permeability, we now allow it to to be tuned so as to match the simulation results of the fine grid reference model in terms of the well production rates at each time stamp. The parameter 'conntrans' models connection transmissibility relating the pressure difference at the well with the production rate which is also usually derived from the properties of the well and the surrounding properties but here it is enabled to fit with the averaged pressure over larger cells.

```
%% parameter options
setup = struct('model', model, 'schedule',...
schedule, 'state0', stateInit);
nc =  modelRef.G.cells.num;
nf =  numel(modelRef.operators.T);
% transmissibility
parameters{1} = ModelParameter(setup, 'name', 'transmissibility', ...
                                      'type', 'value');
parameters{2} = ModelParameter(setup, 'name', 'conntrans', ...
                                      'type', 'value');
```

At the end, we define the residual function which adds up the mismatch between the well production rates at every time step with a predefined weight. The oil and water rate are here considered more important, since we are relating cost factors with those.

```
%% Setup function handle to evaluateMatch
u = getScaledParameterVector(setup, parameters);
% Define weights for objective
weighting =  {'WaterRateWeight',  (300/day)^-1, ...
              'OilRateWeight',    (300/day)^-1, ...
              'BHPWeight',        (500*barsa)^-1};

% 1. gradient case - objective is sum of mismatches squared
obj1 = @(model, states, schedule, statesRef, tt, tstep, state)
matchObservedOW(model, states, schedule, statesRef,...
       'computePartials', tt, 'tstep', tstep, weighting{:},...
       'state', state, 'from_states', false,'mismatchSum', true);
f1 = @(u)evaluateMatch(u, obj1, setup ,parameters,...
       statesRef, 'enforceBounds', false);
```

Observe that `evaluateMatch` does not only provide the residual function value, but also performs the gradient computation with the use of automatic differentiation, which allows us to perform the reverse mode simulation to obtain the adjoints. If we now use an optimizer from MRST or the MATLAB Optimization Toolbox, we can provide the gradient and save important computation complexity compared to using finite differences to compute the gradient with $2m$ many function evaluations (see Eq. (1.34)). These function evaluations of `f1` are coming with a simulation of the coarse grid model for a long simulation with multiple timesteps, and thus the advantage of using automatic differentiation is significant. Usually, a training cycle uses up to 30 iterations and quasi-Newton often only requires one function evaluation per iteration. Hence, computing the gradient via finite differences already takes more computation time than the whole training process when we provide the gradient in each iteration beforehand.

3.2 Reduced-Order Models in MRST

The problem we consider compares well production rates of the reduced-order models with the fine-scale reference models. Note that our goal is to generate low cost alternatives for forecasting production rates to optimize well control strategies [14]. The general setting of Section 1.6 will be refined in the following. Hence we have again

$$\min_{\zeta\in[0,1]^n} \quad f(\zeta) = \frac{1}{2}\|r(\zeta)\|^2 \tag{3.1}$$

where $r(\zeta) \in \mathbb{R}^m$. The restriction on the unit box $[0,1]^n$ can be achieved by scaling the parameters. We restrict ourselves to the case where the r_i's are related to the well production rates for each time step. For a fixed time step and well, we compute the well production mismatch of the coarse-scale model by comparing the different values of pressure at wells or production rates of wells. At the end our coarse-scale model predicts well production rates for the fine-scale model over the whole lifespan of a reservoir despite using only a small fraction of the number of cells in the fine-scale model.
We call the the process where we iteratively find an approximation of a local minimum *tuning* or *training* of the coarse-scale grid model. However, we have no method to find the global minimum. Therefore, different optimization algorithms with similar performance find different solutions. An ensemble of those method potentially improves the end result. In particular the hierarchy of Gauss–Newton method we will introduce in Section 3.3 provide a basis for such an ensemble of optimizers.

As indicated above, the MRST optimization algorithm is using BFGS with line search. Another task here is to compare the performance of relaxations of the Gauss–Newton methods from the MRST Optimization Toolbox with the current algorithm and approve the theoretical faster convergence rate. The examples we will train are three reservoir models which are all contributed to the MRST software. A comprehensive list of all available data sets in MRST can be opened with the command `mrstStartupMessage` in MATLAB followed by clicking on `mrstDatasetGUI()`. We consider a simple Cartesian grid model by Stein Krogstad with a SPE10 rock [4] permeability and porosity field, a model of the Norne field in the Norwegian Sea by Sintef Digital [32] and the synthetic Egg-model, generated at Delft University by Jansen et al. [13] from 2014. The reduced-order training files were contributed by Sintef Digital.

We consider the following reservoir models:

1. A Cartesian grid with four wells and 2500 cells and rock porosity SPE10. See Figure 3.1.

2. The Norne field model in the Norwegian Sea with 11 wells and 44915 cells. See Figure 3.3.

3. The Egg model, a synthetic model with 12 wells and 18533 cells. See Figure 3.4.

The following plots show the fine and coarse-scale models with a description of how many wells and cells we are considering. The more comprehensive data is provided in the datasheet where also the dimension of the Jacobian of r from Eq. (2.26) and the tuning parameters are listed (see Table 3.1). From this table we can deduce the computational complexity of the given optimization problem but we can also analyze it in more pratical terms.

Let us consider the easiest reservoir model of the three to perform exemplarily an analysis.

1. The Sensitivity model has a 3D cart grid of dimension $50 \times 50 \times 1$. The rock is of constant porosity and permeability or is part of SPE10—a complex rock structure feature from MRST [4, Model 2].

2. The coarse grid is a 3D cart grid of dimension $3 \times 4 \times 1$.

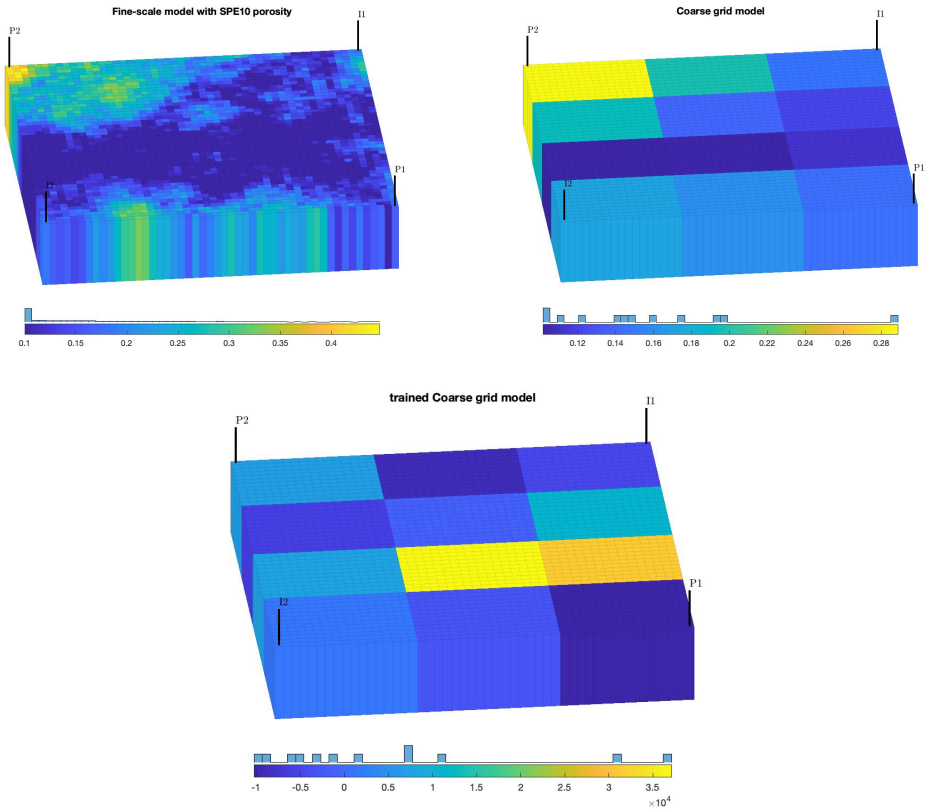


Figure 3.1: The colorbar indicates the porosity. The left upper plot shows a $50 \times 50 \times 1$ Cartesian grid with a SPE10 porosity field with four wells. The right plot shows the coarse grid model of dimension $3 \times 4 \times 1$ before training. The porosities are obtained by averaging the porosity of the cells in the fine grid model. I_1, I_2 are the injector and P_1, P_2 are the producer wells. The least-square function f consists of the sum of the mismatches of the well production rates in both models for each time step. The training result is shown in the third plot. The porosity is nowhere close to represent the distribution.

3. The parameters are 'transmissibility', 'conntrans' and 'porovolume'.

The transmissibility and the connection transmissibility define the values of the discrete gradient and divergence operator. In the reference grid the transmissibility is the proportionality constant of the flux and the pressure difference and depends on the permeability of the rock and the geometry of the cell partition. More precisely, it is defined here in Eq. (1.22). 'conntrans' stands for connection transmissibility and is the proportionality constant between the flux and the pressure difference at the wells. Let us consider a small computation to obtain n, the number of parameter variables, from such a physical description.
There are two ways to obtain this. First, MRST provides grid structures with a lot of geometrical information such as orientation of faces, centroids of cells, volumes. Now let `f = model.G.faces.neighbors` where `f` consists of two columns where each entry indicates an oriented face from the first entry to the second entry. We let all outer faces be oriented inwards, that means all entries for faces to the outside

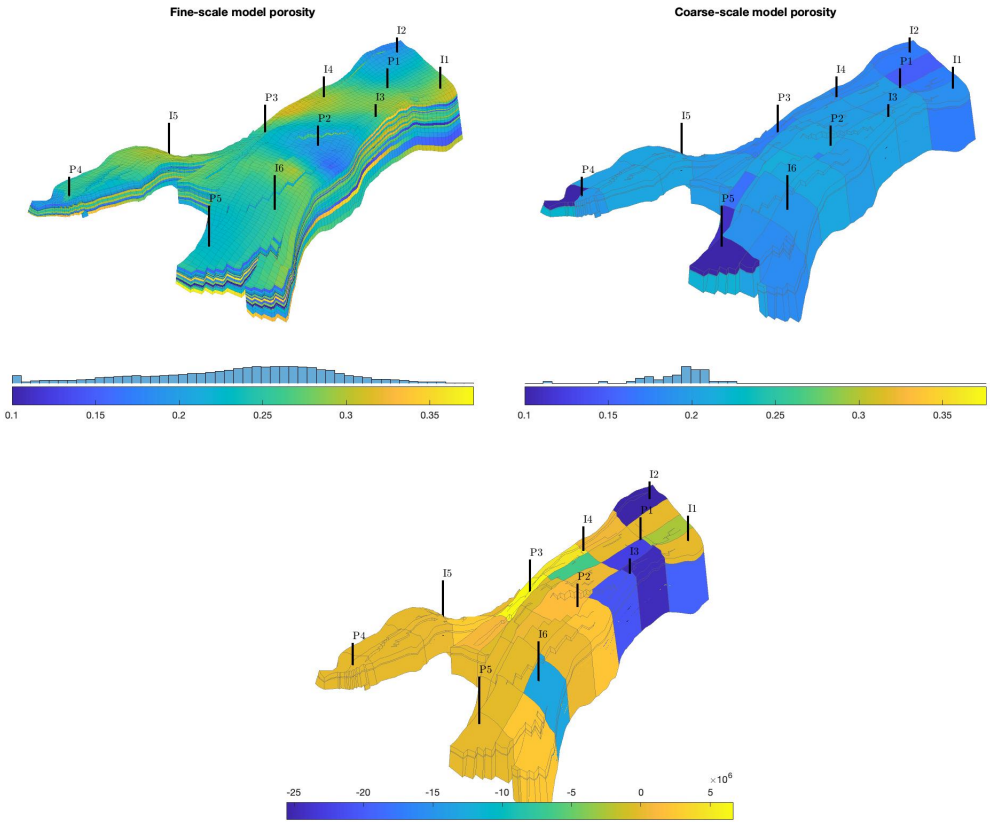


Figure 3.2: The Norne field in the Norwegian Sea with 11 wells. The fine grid model has 44915 cells and the coarse grid has 53 cells . The parameters are 'transmissibility' , 'conntrans' , 'porevolume', and saturation limits of the cells for each fluid ('swcr', 'swl', 'sowcr' etc.).

are starting with a zero, where zero indicates the outside cell.
Then the number of rows in `f(~(f(:,1) == 0),:)` equals the number of inner faces which is also equals the number variables for the parameter 'transmissibility'. Secondly, in our example one could compute the number of inner faces by hand. Let us consider the more general case.

Lemma 3.1 (Inner Faces in Cartesian Grid):
Let us consider the Cartesian grid of dimension $x_1 \times \ldots \times x_N$ where $x_i \in \mathbb{N}$ then the number of inner faces in the grid is equal to

$$\sum_{j=1}^{N} (x_j - 1) \prod_{i \neq j} x_i.$$

Proof. Let e_j be the standard basis vector and let the Cartesian grid be embedded in a Cartesian coordinate system aligned with the coordinate axes. Then we can count the faces orthogonal to e_j which are exactly $\prod_{i \neq j} x_i$ faces for a fixed j coordinate where we find a face orthogonal to e_j, and since we are counting only inner faces we have to multiply this by $x_j - 1$. Since every face is exactly orthogonal to one of the

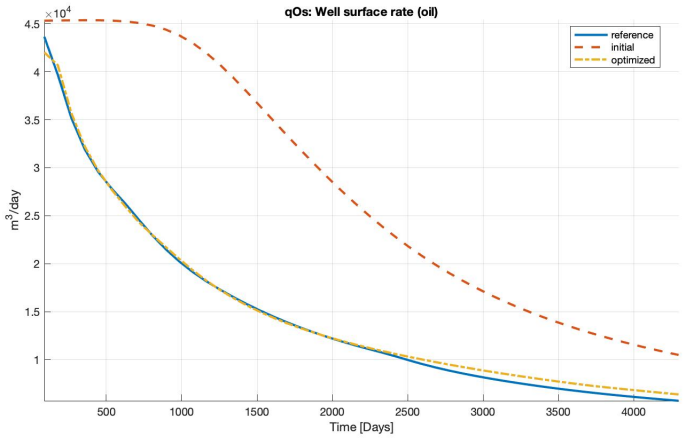


Figure 3.3: The simulation of the well production rates of the trained coarse-scale model matches well to the production rates of the fine grid model.

e_j, we can sum up the number of faces as in the formula. □

Therefore our grid with dimensions $3 \times 4 \times 1$ has $(4-1)3 + (3-1)4 = 17$ inner faces which matches the output in MATLAB for the command `size(f(~(f(:,1) == 0),:),1)`. Hence we have 17 variables for the transmissibility parameter and the connection transmissibility is a variable for each well, hence we have 4 variables for this parameter. The porosity is defined upon each cell and yielding 12 variables for the tuning of 'porosity' giving $n = 33$ in our example. Now the Sensitivity example uses a schedule of 30 time steps and contains 4 wells. Since each well production will be assessed by 3 pressure values (water pressure, oil pressure and 'bhp' —bottom hole pressure) we obtain $3 \cdot 4 \cdot 30 = 360 = m$ residual entries. Hence the Jacobian for the Gauss–Newton methods is of dimension 360×33. In order to solve a Gauss–Newton directional equation, we consider the matrix $J^{\mathrm{T}}J \in \mathbb{R}^{33\times 33}$ which is very cheap compared to the simulation of the coarse-scale model and hence neglible.
This is also the case for the other examples hence the main issue is to minimize the number of function evaluations. Since Levenberg–Marquardt uses relatively many function evaluations per iteration, the trust-region method has a more promising overall performance and the same expected quadratic convergence rate. One drawback, however, which we will encounter while considering relaxations of the Gauss–Newton directional equation (2.30) is that the existence of solution for $J^{\mathrm{T}}Jp_k = -J^{\mathrm{T}}r$ can be problematic and then we must switch to Levenberg–Marquardt.

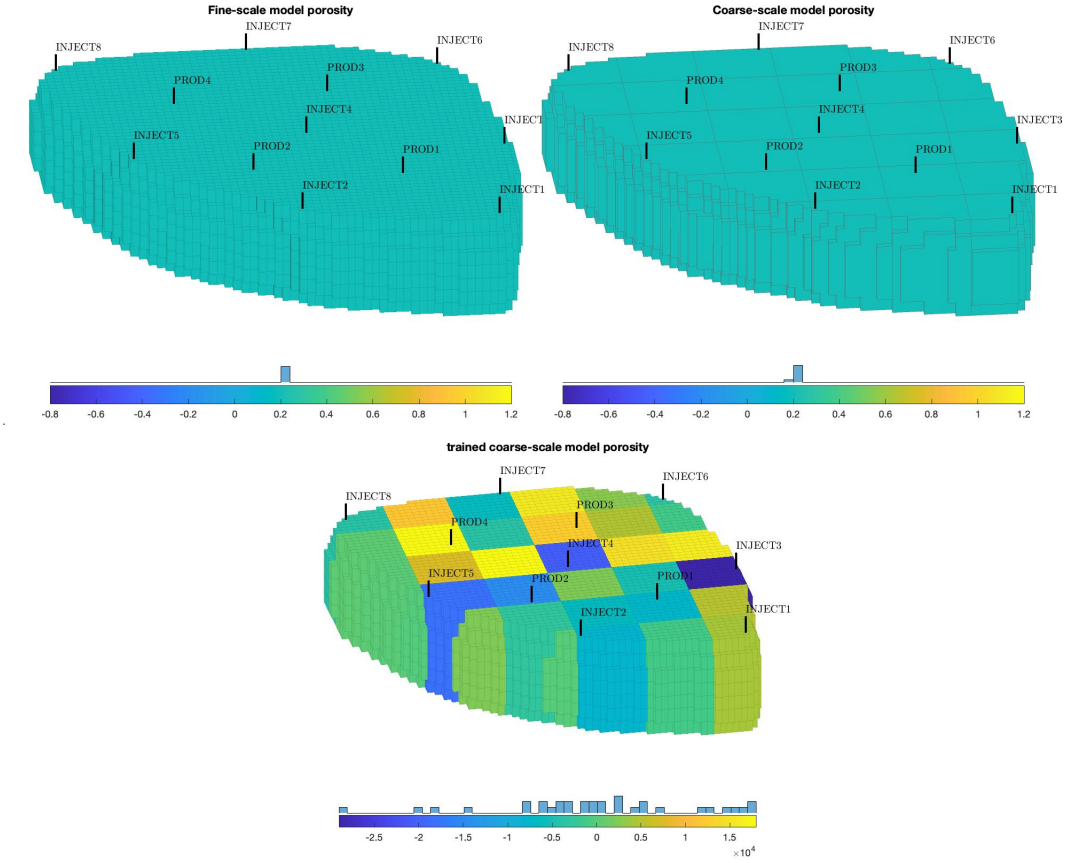


Figure 3.4: The Egg-model with uniform rock porosity. The fine-scale grid consists 18533 cells and the coarse-scale grid consists of 33 cells. The parameters are 'transmissibility' , 'conntrans' , 'porevolume', and saturation limits of the cells for each fluid ('swcr', 'swl', 'sowcr' etc.). In all three reservoir models we can achieve almost arbitrary accuracy by increasing the iteration maximum.

Table 3.1: The attributes provide a small insight into the complexity of a training iteration. Recall that updating the direction for BFGS takes $\mathcal{O}(n^2)$ computation steps. Solving the system with $J^{\mathrm{T}}J + \lambda\mathbf{I}$ takes $\mathcal{O}(n^3)$ computation steps. One could also compute m for each column (or model) by taking the product of number of wells times number of time steps times number of fluids plus one—pressure at the wells for each fluid plus the 'bhp', that is, the bottom hole pressure. The tuning parameters define n. In order to illustrate that conisder the column of the Egg model. We can observe that with all the saturation and relative permeability parameters are defined for each cell in the coarse grid. Hence n for the Egg model must be equal to $33 \cdot 7 + 12 + 54 = 297$, where 54 is the number of inner faces in the grid of the Egg model and 12 is the number of wells matching the number of variables for 'conntrans'. The saturation is defined on a fluid for each cell. The relative permeability is in the physical model due to differences in permeability in different states of saturation. Fully with oil saturated porous media is usually less permeable for water than unsaturated porous media.

Attribute	Norne	Egg	Sensitivity
N. of cells (fine-scale)	44915	18533	2500
N. of cells (coarse-scale)	53	33	12
N. of wells	11	12	4
training time steps (coarse)	24	60	30
Fluid types	WO	WO	WO
dimension of J	792×472	2160×297	360×33
Tuning parameters			
porevolume	**true**	**true**	**true**
conntrans	**true**	**true**	**true**
transmissibility	**true**	**true**	**true**
smallest water saturation (SWL)	**true**	**true**	**false**
highest water saturation (SWCR)	**true**	**true**	**false**
max. water saturation (SWU)	**true**	**true**	**false**
highest oil-in-water sat.(SOWR)	**true**	**true**	**false**
max. rel. permab.water (krw)	**true**	**true**	**false**
max. rel. permab. oil (kro)	**true**	**true**	**false**

3.3 Complexity and Gauss–Newton Relaxation

The data sheets are of importance for the complexity analysis. Recall that we pass the gradient from the simulation of the coarse grid model to the optimizer otherwise it will compute it numerically with $2n$ function evaluations, where n is the dimension of the parameters. That however, implies that we have to compute and store the Jacobian matrix J. That is the main drawback of the Gauss–Newton methods compared to BFGS, since $J \in \mathbb{R}^{m \times n}$ where in the considered cases $m \gg n$. For the BFGS method it is enough to pass $\nabla f(x_k) \in \mathbb{R}^{n \times 1}$. Other from that the iteration complexity is mainly formed by the number of function evaluation, since computing the direction of the next step only costs $\mathcal{O}(n^2)$ for BFGS and $\mathcal{O}(n^3)$ computation steps for the Gauss–Newton methods and n is in the magnitude of the number of coarse grid cells. For comparison, alone in order to compute one space discretization for one time step in the simulation of the coarse grid model we require to solve the system with the transmissibility matrix (1.27) which costs at least $\mathcal{O}(N^2)$ computation steps, where N equals the number of cells in the coarse-scale model. Also we solve for each time steps the implicit-Euler Equation (1.38) with Newton–Raphson, which also requires $\mathcal{O}(N^3)$ computations in each iteration. Hence the function evaluations of f are clearly the main contributor of computational complexity in each iteration and the number of required function evaluation vs. the decreasing residual gives us the main indicator on the performance of the algorithm. It is to be expected that the algorithms with quadratic convergence needs a significantly smaller number of iteration and function evaluations and hence and overall better performance than BFGS with line search. However, the quadratic convergence of Gauss-Newton is only given in special conditions (when the residual entries r_i are small or close to affine mappings). The performance of the Gauss-Newton methods on the three reservoir models will be discussed in Section 3.4.
So even though we are expecting that saving J is not as expensive as running more function evaluations, we want to experiment with the reduction of the dimension of J leading to a certain relaxation of the Gauss-Newton methods. This class of Gauss-Newton methods reduce the complexity by considering partial sums of r to reduce dimensions of the Jacobian matrix. We write those dimension reduced residual vectors as

$$\tilde{r} = (\tilde{r}_I = \sum_{i \in I} r_i, I \in P)^{\mathrm{T}} \tag{3.2}$$

where P is a partition of $\{1, 2 \ldots, m\}$. A proper definition of those methods is the following:

Definition 3.2 (Relaxations of Gauss-Newton):
Let

$$J = \begin{pmatrix} \nabla r_1^{\mathrm{T}} \\ \vdots \\ \nabla r_m^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{m \times n}.$$

For a partition $(I_1, \ldots, I_\ell)$ of $\{1, \ldots, m\}$ we consider relaxations of the Gauss-Newton methods of the form

$$\tilde{J}^{\mathrm{T}} \tilde{J} p_k = -J^{\mathrm{T}} r \tag{3.3}$$

where

$$\tilde{J} = \begin{pmatrix} \left(\nabla \sum_{i \in I_1} r_i\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla \sum_{i \in I_\ell} r_i\right)^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{\ell \times n}. \tag{3.4}$$

Observe that for the trivial partition $(\{1, \ldots, m\})$, $\tilde{J}$ is one row of dimension n and the resulting method is $\tilde{J}^{\mathrm{T}} \tilde{J} p_k = -\nabla f(x_k)$ where $\tilde{J}^{\mathrm{T}} \tilde{J}$ has dimension 1 and the system is likely to be not solvable. In general, if $\ell < n$, the trust-region method for least square problems from MATLAB Optimization Toolbox automatically changes to the Levenberg–Marquardt method which reduces to

$$(\lambda \mathbf{I} + \tilde{J}^{\mathrm{T}} \tilde{J}) p_k = -\nabla f(x_k)$$

which is then solvable for some $\lambda \neq 0$. We restrict ourselves to the subclass of partitions of $\{1, \ldots, m\}$ which are equal sized and only merges residuals for the same well and pressure point and fluid but for different consequent time steps. For example if we consider the Sensitivity model, then we have 4 wells with pressure values for oil and gas and bottom hole pressure resulting in 12 residuals for the first time steps. The next 12 residuals represent those values for the second time step and so on. If we now consider this enumeration and merge consequent time steps with a partition of equal-size 2, we obtain the corresponding partition of $\{1, 2, \ldots, 360\}$ as

$$\begin{array}{cccccccc} r_1 & r_{13} & r_{25} & r_{37} & r_{49} & r_{61} & & r_{337} & r_{349} \\ r_2 & r_{14} & r_{26} & r_{38} & r_{50} & r_{62} & & r_{338} & r_{350} \\ r_3 & r_{15} & r_{27} & r_{39} & r_{51} & r_{63} & \cdots & r_{339} & r_{351} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ r_{12} & r_{24} & r_{36} & r_{48} & r_{60} & r_{72} & & r_{348} & r_{360} \end{array}$$

For a short analysis on the performance of the relaxation, we consider the identity from Eq. (2.29), that is,

$$\nabla^2 f = J^{\mathrm{T}} J + \sum_{i=1}^{m} r \nabla^2 r = \sum_{i=1}^{m} \nabla r_i \nabla r_i^{\mathrm{T}} + \sum_{i=1}^{m} r \nabla^2 r.$$

Remark 3.3:
For a partition $(I_1, \dots, I_\ell)$ of $\{1, \dots m\}$ $\tilde{J}$ defined as in Eq. (3.4, we have

$$\tilde{J}^{\mathrm{T}}\tilde{J} = J^{\mathrm{T}}J + \sum_{i=1}^{\ell} \sum_{j \neq k \in I_i} \nabla r_j \nabla r_k^{\mathrm{T}}.$$

Proof. A straightforward computation gives

$$\begin{aligned}
\tilde{J}^{\mathrm{T}}\tilde{J} &= \sum_{i=1}^{\ell} \left(\nabla \sum_{j \in I_i} r_j \right) \left(\nabla \sum_{j \in I_i} r_j \right)^{\mathrm{T}} \\
&= \sum_{i=1}^{\ell} \left(\sum_{j=k \in I_i} \nabla r_j \nabla r_j^{\mathrm{T}} + \sum_{j \neq k \in I_i} \nabla r_j \nabla r_k^{\mathrm{T}} \right) \\
&= J^{\mathrm{T}}J + \sum_{i=1}^{\ell} \sum_{j \neq k \in I_i} \nabla r_j \nabla r_k^{\mathrm{T}}.
\end{aligned}$$

□

The remark shows that the perturbation term $\sum_{i=1}^{\ell} \sum_{j \neq k \in I_i} \nabla r_j \nabla r_k^{\mathrm{T}}$ is small for finer partitions. Hence $\tilde{J}^{\mathrm{T}}\tilde{J}$ becomes a worse approximation of $\nabla^2 f$ for coarser partitions of $\{1, \dots, m\}$ which matches the intuition.

In order to approve these theoretical observations, we run the calibrations of coarse-scale models for all three models on all those relaxations. In our case this will result in 63 training processes (30 for the trust-region relaxations with Gauss-Newton, 30 for Levenberg–Marquardt relaxations and 3 for BFGS). However, as indicated most of the trials with the trust-region method will end up with the results from Levenberg–Marquardt, since `lsqnonlin` automatically switches to Levenberg–Marquardt when J has less rows than columns in order to avoid the situation where

$$\tilde{J}^{\mathrm{T}}\tilde{J}p_k = -J^{\mathrm{T}}r$$

has no solution.

3.4 Numerical Experiments

We will provide in this section a concise presentation of our results. First we present the results of the standard Gauss-Newton methods from `lsqnonlin`, that are, the trust-region method with Gauss-Newton directional Eq. (2.30) and Levenberg–Marquardt. Then we will discuss the results of the relaxations. We focus on the 'good' results and explain surprising performances against the intuition of Remark 3.3. We will conclude this section with suggestions for improvements and ideas which could be applied for further developments.

The implementation of the examples for tuning parameters for coarse-scale models were contributed by Sintef Digital. However, the script to generate the tables and the figures, and also systematically going through the relaxations is all automated and contributed by me.

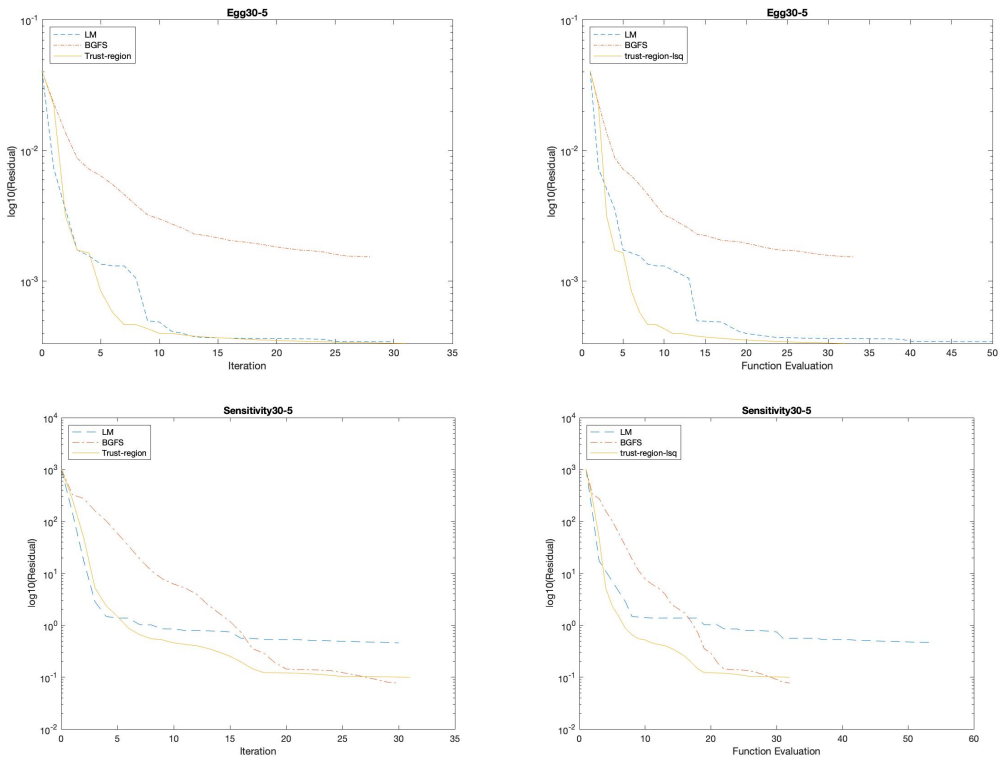


Figure 3.5: The format for the title name is 'field name' + 'number of iterations' + '-' + 'k' where k indicates that we merge the residuals regarding to k consequent time steps. We consider the relaxations of the Gauss-Newton methods from Definition 3.2 with merging residuals from 5 consequent time steps. Levenberg–Marquardt method performs worse in terms of function evaluation but also in terms of optimal value. The plots on the right hand side indicate the real complexity of the training process.

Gauss-Newton against BFGS. On 30 iterations the Gauss-Newton methods outperform BFGS in the Sensitivity model and the Egg-model. However, the result is not so clear for the Norne model (see Figure 3.6). The trust-region method performs much worse than the Levenberg–Marquardt method and even worse than BFGS. However, the Levenberg–Marquardt method outperforms BFGS when we train with 50 iterations—even when the Levenberg–Marquardt method uses for this results 20 more function evaluations than BFGS. A fair comparison here would be achieved by running 20 more iterations of BFGS. The similarity of the convergence rates of both Gauss-Newton methods are observeable for the Egg and Sensitivity models but for the Norne model it is not the case.

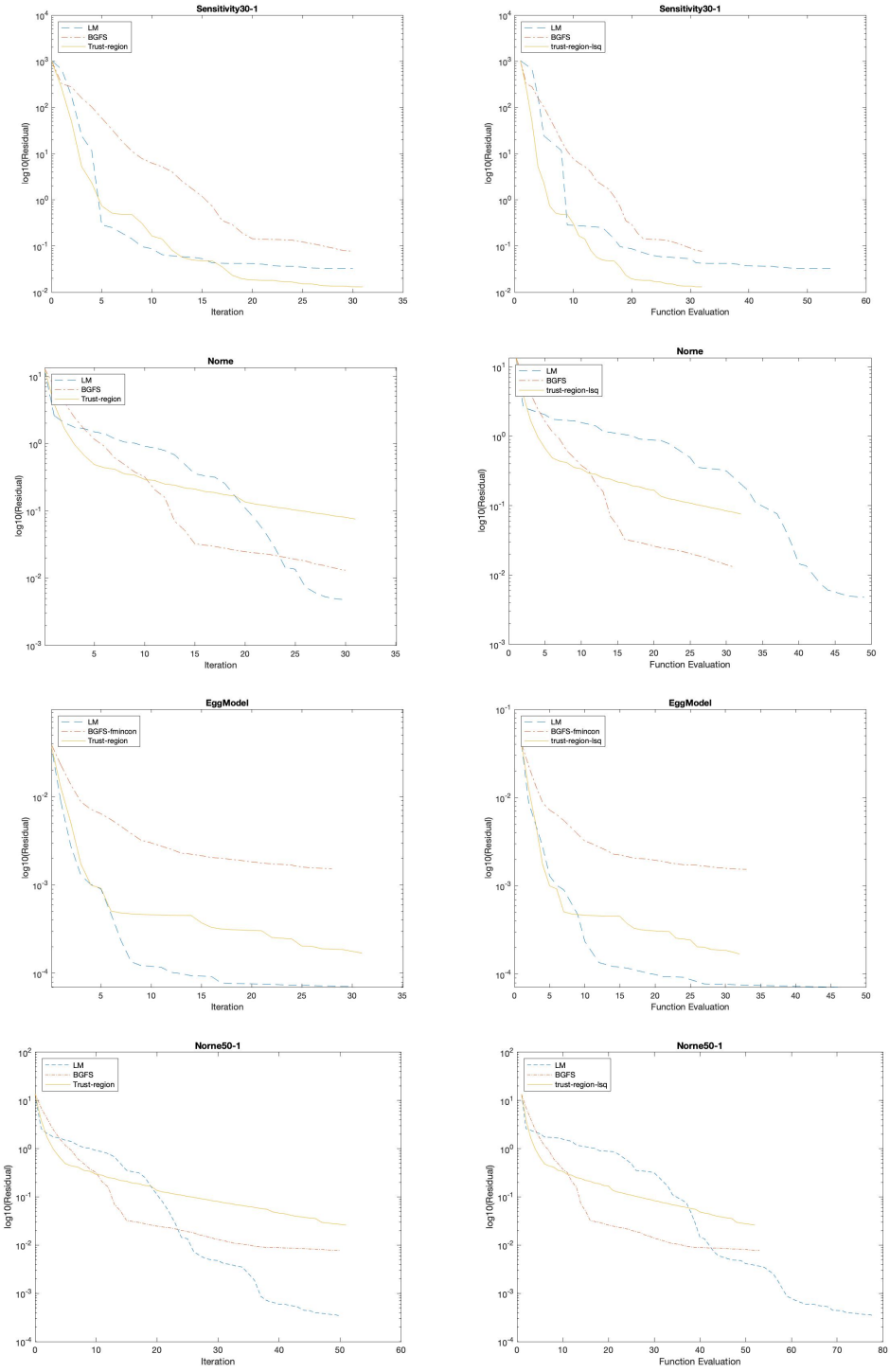


Figure 3.6: Perfomance of the Gauss-Newton methods against BFGS. For the Sensitivity example the quadratic convergence rate of Levenberg–Marquardt is clearly visible in the beginning. For the Norne model the steady convergence of BFGS seems to outperform Levenberg–Marquardt here, except at the end where the quadratic convergence rate of Levenberg–Marquardt creates the better end result with respect to the number of iterations.

Relaxations of the Gauss-Newton Methods. Table 3.2 shows the end residual value and the number of function evaluations for each of the relaxations of the Gauss-Newton methods. The blank spaces indicate that the integer in the first column doesn't divide the number time steps for the respective schedule of the model or the trust-region method could not be applied in that case. There we don't look at the partition, because it is not equal-sized (even though there is particular reason for the partition to be equal-sized).
As expected the results show a tendency to poorer performance of the relaxations with large partition size. On top of that the trust-region method automatically switches to the Levenberg–Marquardt method when $m < n$, so there is no data on larger equal-sized partitions for the trust-region method.
Some results indicate that the relaxation can produce equally good results to the classical methods. For instance Norne with partition of size 3,4 and 6 and also the Sensitivity model for partition of size 2 with the Levenberg–Marquardt method produces better results than the classical Gauss-Newton methods.
with the same time complexity. An important addition is in the Appendix 4.2. There are more plots showing the calibration process with the relaxations. One can see that not only the result improves but also the convergence behaviour can change completely. That means, that some of those relaxations can find equally good but completely different optimization paths.

Table 3.2: The tables show the misfit value of the solution and the number of function evaluation produced by the relaxations (with respect to the time steps) of the trust-region method and the Levenberg–Marquardt method with 30 iteration. The first column contains the divisor of the number of overall time steps, which defines the size of a class of the equal-sized partition as in Eq. (3.2). For instance the first row describes the standard case where each time steps and well and type yields one residual summand. We are expecting the best result in the first row, however, we observe that this is not always the case. Small divisors produce even better solutions, for instance in the second table for 3,,4 and 6 in the Norne field. Even with big divisors as 12 we obtain feasible results. However, those are worse than BFGS and considering the high time complexity due to many function evaluation, BFGS is to be preferred. Further observe that the trust-region method finds a local minimum for divisor 2 and 3 even before reaching the iteration maximum of 30. This makes the trust-region method surprisingly more effective in the those cases.

Trust-region method						
Div.	Norne - res.	Egg - res.	Sens. - res.	Norne_F	Egg_F	Sens_F
1	0.075533	0.00016919	0.013087	32	32	32
2		0.00024668	0.0062764		22	32
3	0.54222	0.00023337	0.0099922	26	18	32
4		0.0002943			21	
5		0.00033397	0.099332		32	32
6		0.0011701	0.039545		32	32
10			0.033208			32
12						
			Levenberg–Marquardt			
1	0.0047734	7.1259e-05	0.033009	49	46	54
2	0.0053582	0.00013787	0.01997	49	49	52
3	0.0032048	0.00018697	1.1195	50	48	30
4	0.0038343	0.00030101		47	48	
5		0.00034416	0.46061		50	54
6	0.004298	0.00062888	0.075597	50	49	52
8	0.027227			53		
10		0.0011051	0.20978		50	54
12	0.0091827	0.0012233		53	50	
15		0.00133	0.10099		49	52
16	0.034075			51		
20		0.0014075			51	
24	0.034075			51		
30		0.0013314	11.9948		52	58
48	0.034075			51		
60		0.0023196			49	

Conclusion. From Fig. 3.6 we see a clear tendency for the faster convergence rate of the Gauss–Newton methods. In all three examples the convergence rate of the Levenberg–Marquardt method convinces, except for the Norne field model. The last two plots in Fig. 3.6 clarify those doubts. The Levenberg–Marquardt method manages to obtain significant improvements in the last 20 more iterations. The closer we come to a local minima, the better the Gauss–Newton update strategy Eq. 2.30 provides a approximation of the Hessian matrix (because the residual values are small, compare Eq. (2.29)) and also the better the Newton method would have had performed.

The trust–region method convinces in the Sensitivity example and the Egg model but the second row of plots in Fig. 3.6 shows it performs much worse for the Norne field.
We don't observe these fluctuating steep error reduction from the Levenberg–Marquardt method even though both methods rely on the same Gauss–Newton update strategy. Hence we can attribute the discrepancy in convergence within the Gauss–Newton methods to the weak adjustment strategy of the trust–region radius Δ_k or the inaccuracies of the two-dimensional sub space method in Section 2.4.

From Table 3.2 we observe that there is no partition size which performs better in all examples. But what we can observe is that we have an equally well performing set of optimizers with those relaxations of the Gauss–Newton methods.

Hence it enables us to consider an ensemble of methods. Paired with parallel computation this opens up the opportunity to create ensemble optimizers running

3.5 Further Work

There are a couple of remarks emerging from those previous numerical experiments. First, the terms $J^{\mathrm{T}}J$ and $J^{\mathrm{T}}r$ can be computed without storing J as remarked in Section 2.7.1. This would enhance the space complexity without increasing the time complexity.
A further possible improvement could be to use ensemble methods. This could either be in the big scale where we run different methods in parallel and choose the best solution afterwards. Or we could use an ensemble of iteration methods in each iteration in parallel and choose the step with the largest decrease in the residual values. What also could be useful for the theoretical analysis is to implement the Newton-method with automatic differentiation to attribute the non-quadratic convergence to either a bad approximation of the Hessian or a bad condition of the problem.

We will summarize and name various more possible extensions in the following:

- implement the remark in Section 2.7.1 at the end which saves us the computation of J.

- *improve the choice* of the Gauss–Newton relaxation. The partitions on which we decide which residual will be summed up was made arbitrary. This could also be decided or chosen upon physical reasoning, since we employ a certain weight on the different residuals implicitly by merging them together. Generalizing this idea could also lead to a structured way to impose weight parameters.

- find the ideal *outer scheme* for the Gauss–Newton method. In particular, try

different trust–region radius adjustment strategies.

- extend the test library (different types of order reductions, different types of process/production optimization, different use-cases co2lab [27] etc.) and automatize the test scheme for an arbitrary number of models.

- employ more complex *machine learning* and *deep learning* methods combined. E.g. in [36] convolution neural networks are employed to parameterize approximated governing equations. Instead of using fixed parameters, we could replace them by a dynamic neural network which governs those parameters.

- implement an *ensemble optimizer*, since there is no clear best method. We ensemble the step vectors for all different methods and decide afterwards which step we choose. The selection either will be done by evaluating all resulting new states (expensive) or one could use cheaper measures to decide for the best step (e.g. employ gradient and curvature conditions). Regarding the iteration expenses to compute the different step, for all Gauss–Newton relaxations from Def. 3.2 we only require the Jacobian J. Reducing the size by merging some of the rows of J together and set up the system Eq. 3.3 can be done without much computational effort.

At the end I want to emphasize that there was much focus on the simulation of reservoir flow.

The beautiful part about those implementations is that MRST, as an open source software with a highly accessible and comprehensive documentation [26, 22, 23], provides a tool to generalize those findings, and developments and use them in various ways and it can be employed almost anywhere where complex fluid dynamics are part of an optimization process.

4 Appendix

Let f be the objective function and

$$x_{k+1} = x_k + \alpha_k p_k$$

and iteration method where p_k is the search direction and α_k is the step length. p_k and the negative of the gradient $-\nabla f(x_k)$. Further we define the angle between the negative gradient and the step direction

$$\cos\theta_k = \frac{-\nabla f(x_k)^{\mathrm{T}} p_k}{\|\nabla f(x_k)\|\|p_k\|}.$$

Theorem 4.1 (Zoutendijk's Theorem):
Assume that f is twice differentiable, ∇f is Lipschitz continuous to bounding constant $L > 0$ and f is bounded from below by M. Further let p_k be a descent direction, that is,

$$p_k^{\mathrm{T}} \nabla f(x_k) < 0$$

and α_k satisfies the Wolfe conditions (2.2). Then

$$\sum_{k\geq 0} \cos^2\theta_k \|\nabla f(x_k)\|^2 < \infty$$

and in particular $\cos^2\theta_k \|\nabla f(x_k)\|^2 \to 0$ for $k \to \infty$.

Proof. Consider the second Wolfe condition (2.3)

$$\nabla f(x_{k+1})^{\mathrm{T}} p_k \geq c_2 \nabla f(x_k)^{\mathrm{T}} p_k.$$

Subtracting $\nabla f(x_k)^{\mathrm{T}} p_k$ on both sides, yields

$$(\nabla f(x_{k+1}) - \nabla f(x_k))^{\mathrm{T}} p_k \geq (c_2 - 1)\nabla f(x_k)^{\mathrm{T}} p_k.$$

Applying the Cauchy-Schwarz inequality on the left hand side and use the Lipschitz continuity of ∇f, we derive the following lower bound for α_k

$$L\alpha_k p_k^{\mathrm{T}} p_k \geq (c_2 - 1)\nabla f(x_k)^{\mathrm{T}} p_k$$

and finally,

$$\alpha_k \geq \frac{c_2 - 1}{L\|p_k\|^2} \nabla f(x_k)^{\mathrm{T}} p_k.$$

If we turn back to the first Wolfe condition (2.2) and estimate α_k by its lower bound, we obtain

$$f(x_{k+1}) \leq f(x_k) + c_1 \frac{c_2 - 1}{L\|p_k\|^2} (\nabla f(x_k)^{\mathrm{T}} p_k)^2.$$

Note that $c_1\alpha_k > 0$ and $\nabla f(x_k)^{\mathrm{T}} p_k < 0$, justifying the last step. Now set $c := -c_1\frac{c_2-1}{L} \geq 0$ and use the inequality on the right hand side recursively such that

$$f(x_{k+1}) \leq f(x_0) - c\sum_{i=1}^{k} \frac{(\nabla f(x_k)^{\mathrm{T}} p_k)^2}{\|p_k\|^2\|\nabla f(x_k)\|^2}\|\nabla f(x_k)\|^2 \tag{4.1}$$

$$= f(x_0) - c\sum_{i=1}^{k} \cos^2\theta_k\|\nabla f(x_k)\|^2. \tag{4.2}$$

Letting $k \to \infty$ and estimating by the lower bound of f, we end up with

$$c\sum_{k\geq 0} \cos^2\theta_k\|\nabla f(x_k)\|^2 \leq f(x_0) - M < \infty.$$

□

4.1 Step Length -Wolfe conditions

How can we find a step length that does not violate the Wolfe condition?

There are different methods to find a step length α_k that satisfies the Wolfe condition. The following line search algorithm exploits the idea that we can compute a local minima by a cubic or quadratic interpolation if the interval in which we analyze the function is small enough.
First let us introduce once again the strong Wolfe condition for $\phi(\alpha) := f(x_k + \alpha p_k)$. This leads to the reformulation

$$\phi(\alpha) \leq \phi(0) + c_1\alpha\phi'(0), \tag{4.3}$$

$$|\phi'(\alpha)| \leq -c_2\phi'(0). \tag{4.4}$$

We refer to (4.3) as the *sufficient decrease condition* and to (4.4) as the *curvature condition*.

The function **zoom** will be explained in the following. For now it is only important to assume that **zoom** will compute a point which satisfies the Wolfe condition given that the existence in the given interval is guaranteed. Also, note that the first argument in **zoom** must be the argument with smaller function value with respect to ϕ, but must not be necessarily smaller than the second argument.
Observe that we have the following options for entering in **zoom**

1. $\phi(\alpha_i) \geq \phi(a_i), i > 1$ or $\phi(\alpha_i)$ does not satisfy the sufficient decrease condition
2. α_i satisfies the sufficient decrease condition and has a smaller function value as α_{i-1} as well as $\phi'(\alpha_i) \geq 0$.

These cases both imply that between α_{i-1} and α_i must lie a point satisfying the Wolfe condition. Because both cases (1) and (2) imply the existence of a local minimum between α_{i-1} and α_i which satisfies the sufficient decrease condition. If we enter **zoom** by option (2) then we find ourself in the situation where α_i satisfies the sufficient decrease condition and $\phi(\alpha_i) < \phi(\alpha_{i-1})$. With a nonnegative slope at α_i this guarantees us the existence of a point between α_{i-1}, α_i satisfying the Wolfe

Algorithm 7: Line search – step length

Input: $\alpha_0 = 0, \alpha_{\max}, \alpha_1 \in (0, \alpha_{\max})$ (by a heuristic)
Output: step length α_* satisfying the Wolfe conditions

```
while true do
    Evaluate φ(α_i);
    if φ(α_i) ≥ φ(0) + c_1 α_i φ'(0) or [φ(α_i) ≥ φ(α_{i-1}) and i > 1] then
        α_* ← zoom(α_{i-1}, α_i);
        return α_*;
    Evaluate φ'(α_i);
    if |φ'(α_i)| ≤ −c_2 φ'(0) then
        α_* ← α_i;
        return α_*;
    if φ'(α_i) ≥ 0 then
        α_* ← zoom(α_i, α_{i-1});
        return α_*;
    Choose α_{i+1} ∈ (α_i, α_max);
    i ← i + 1;
```

condition as we have argued before. The only difference here now is that the smaller function value is attained by the right bound α_i.
Now, how can we actually compute a point which satisfies the Wolfe condition? The function **zoom** iteratively shortens the search interval until the quadratic or cubic interpolation is a sufficiently correct approximation of ϕ. Then by using the formula for a minima for the quadratic or cubic interpolation we obtain our candidate. If our canditate fails on one of the Wolfe conditions we will shorten the interval, according to invariances we want to keep in each step. These invariances for the interval $(\alpha_{\text{lo}}, \alpha_{\text{hi}})$ are

1. The interval between α_{lo} and α_{hi} contains a point which satisfies the Wolfe condition.

2. α_{lo} has among all evaluated values generated so far (in **LineSearch** and in **zoom**) the smallest function value.

3. α_{hi} is chosen so that $\phi'(\alpha_{\text{lo}})(\alpha_{\text{hi}} - \alpha_{\text{lo}}) < 0$ which is equivalent to if $\alpha_{\text{lo}} > \alpha_{\text{hi}}$ then at α_{lo} the function ϕ increases and vice versa.

Algorithm 8: Zoom

Input: ϕ, $\alpha_{\text{lo}}, \alpha_{\text{hi}}$
Output: step length α_k

```
while true do
    Interpolate φ to find a trial step length α_j between α_lo and α_hi;
    if φ(α_j) > φ(0) + c_1 α_j φ'(0) or φ(α_j) ≥ φ(α_lo) then
        α_hi ← α_j;
    else
        Evaluate φ'(α_j);
        if |φ'(α_j)| ≤ −c_2 φ'(0) then
            Set α_* ← α_j;
            return α_*;
        if φ'(α_j)(α_hi − α_lo) ≥ 0 then
            α_hi ← α_lo;
            α_lo ← α_j;
```

4.2 Additional Results for Relaxations Of Gauss–Newton

Here are some results omitted in the main part of the thesis. The name indicates on which model we ran the calibration process, the number indicates the maximum number of iterations and the second number indicates which relaxation type we have used. For instance 'Egg30-3' is the calibration on the Egg model with maximum 30 iterations where residuals regarding 3 consequent time steps were merged.

Egg. Fig. 4.1 shows some surprising results from the calibration processes with the Gauss–Newton relaxations. While the best result still came from Fig. 3.6 with the standard Levenberg–Marquardt, some very reasonable results were produced by the relaxations. In particular, the last plot, where we merge residuals regarding 5 consequent time steps, has an equally good optimization results to the standard Gauss–Newton methods.

Norne. Fig. 4.2 shows that we can even have a better result by merging certain residuals and considering the method from Definition 3.2. The first calibration even shows a better error path in the middle part of the process. That indicates that the information loss through adding together residuals might have lead to desirable weightings.

Sensitivity Example. Fig. 4.3 again shows this phenomena which occurred in the Norne field calibration. The relaxation produces an actual better result then the standard Gauss–Newton method.

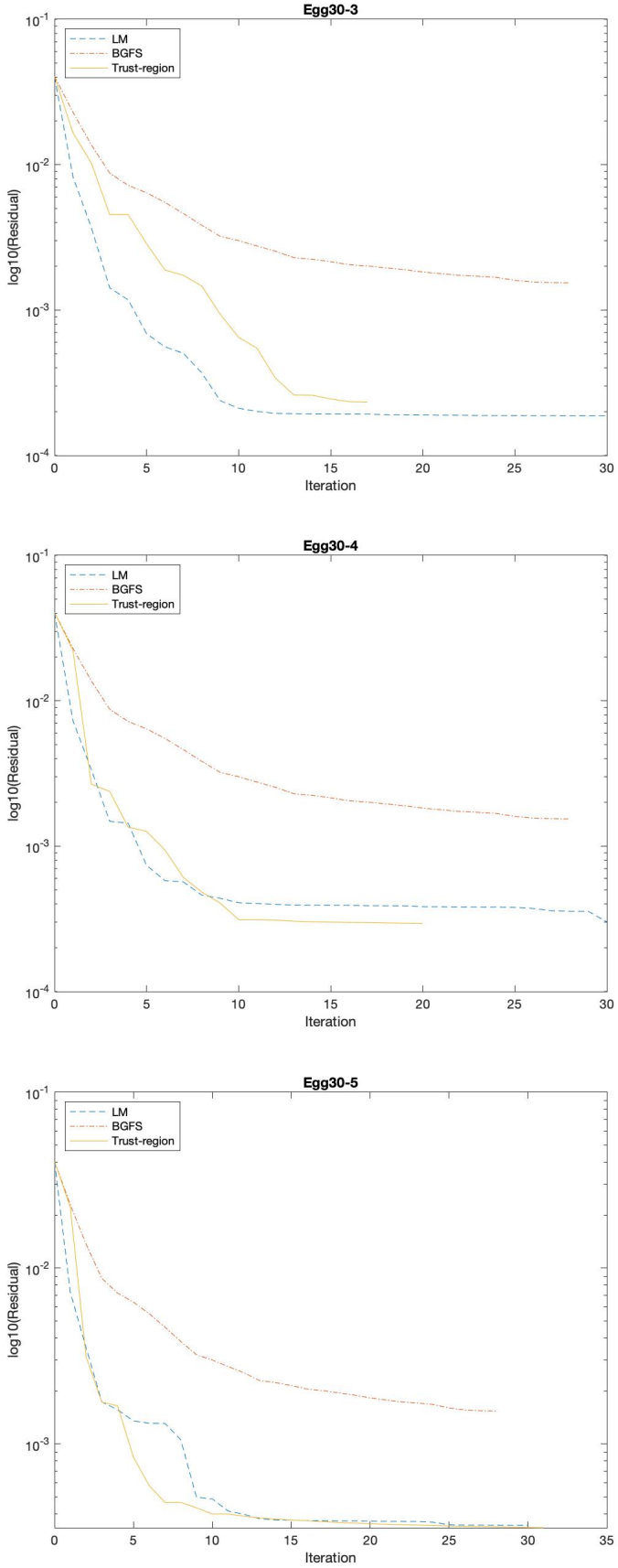

Egg30-3
LM
BGFS
Trust-region
log10(Residual)
Iteration
Egg30-4
LM
BGFS
Trust-region
log10(Residual)
Iteration
Egg30-5
LM
BGFS
Trust-region
log10(Residual)
Iteration

Figure 4.1:

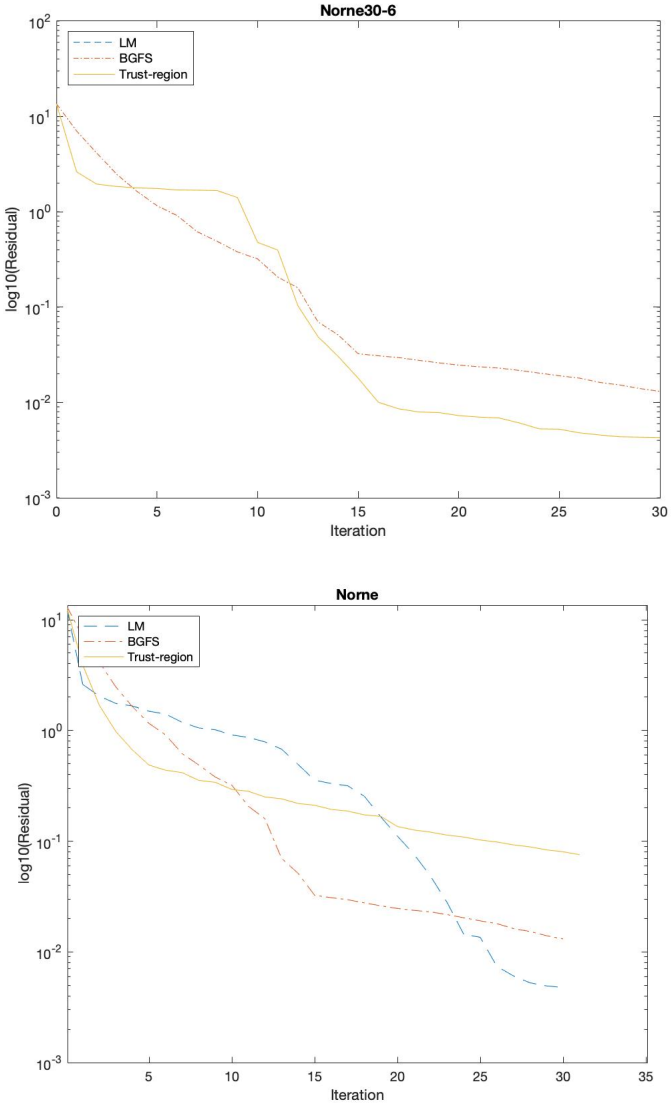

Norne30-6
LM
BGFS
Trust-region
log10(Residual)
Iteration
Norne
LM
BGFS
Trust-region
log10(Residual)
Iteration

Figure 4.2:

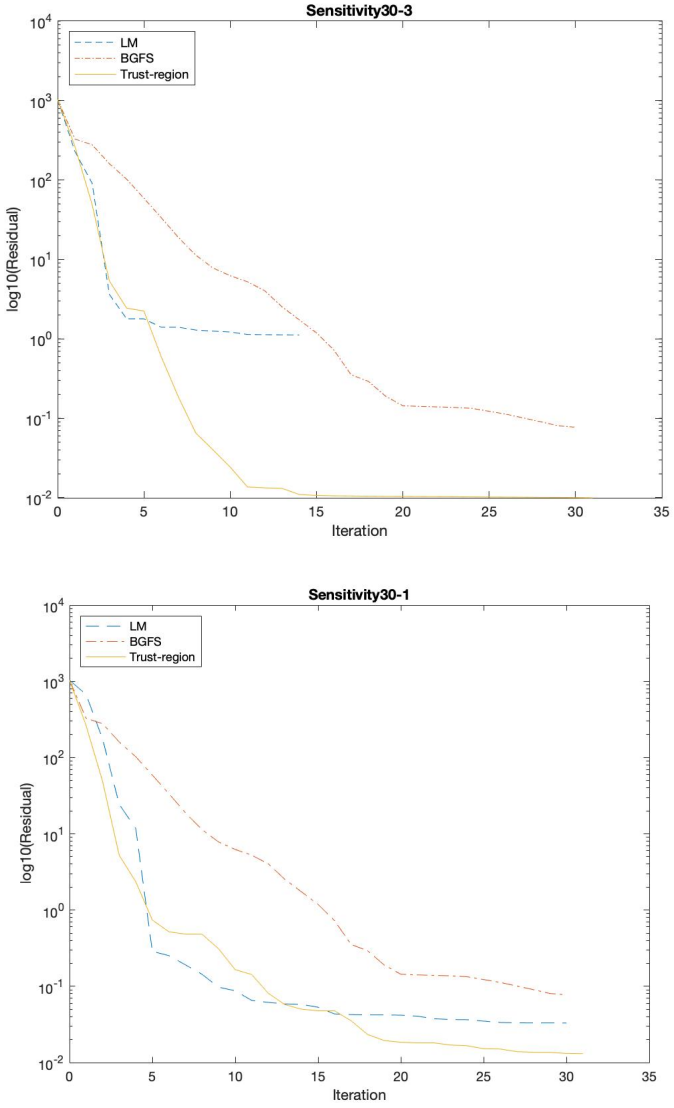

Sensitivity30-3
LM
BGFS
Trust-region
log10(Residual)
Iteration
Sensitivity30-1
LM
BGFS
Trust-region
log10(Residual)
Iteration

Figure 4.3:

Bibliography

[1] Richard O. Baker, Harvey W. Yarranton, and Jerry L. Jensen. "7 - Conventional Core Analysis–Rock Properties". In: *Practical Reservoir Engineering and Characterization*. Ed. by Richard O. Baker, Harvey W. Yarranton, and Jerry L. Jensen. Boston: Gulf Professional Publishing, 2015, pp. 197–237. ISBN: 978-0-12-801811-8. DOI: https://doi.org/10.1016/B978-0-12-801811-8.00007-9. URL: https://www.sciencedirect.com/science/article/pii/B9780128018118000079.

[2] C. G. BROYDEN. "The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations". In: *IMA Journal of Applied Mathematics* 6.1 (Mar. 1970), pp. 76–90. ISSN: 0272-4960. DOI: 10.1093/imamat/6.1.76. eprint: https://academic.oup.com/imamat/article-pdf/6/1/76/2233756/6-1-76.pdf. URL: https://doi.org/10.1093/imamat/6.1.76.

[3] Jean Cea. "Conception optimale ou identification de formes, calcul rapide de la dérivée directionnelle de la fonction coût". fr. In: *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique* 20.3 (1986), pp. 371–402. URL: http://www.numdam.org/item/M2AN_1986__20_3_371_0/.

[4] M. A. Christie and M. J. Blunt. "Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques". In: *SPE Reservoir Evaluation & Engineering* 4.04 (Aug. 2001), pp. 308–317. ISSN: 1094-6470. DOI: 10.2118/72469-PA. eprint: https://onepetro.org/REE/article-pdf/4/04/308/2586053/spe-72469-pa.pdf. URL: https://doi.org/10.2118/72469-PA.

[5] *Constrained Nonlinear Optimiziation Algorithms*. URL: https://de.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html.

[6] Luís Augusto Nagasaki Costa, Célio Maschio, and Denis José Schiozer. "Application of artificial neural networks in a history matching process". In: *Journal of Petroleum Science and Engineering* 123 (2014). Neural network applications to reservoirs: Physics-based models and data models, pp. 30–45. ISSN: 0920-4105. DOI: https://doi.org/10.1016/j.petrol.2014.06.004. URL: https://www.sciencedirect.com/science/article/pii/S0920410514001557.

[7] W C Davidon. "VARIABLE METRIC METHOD FOR MINIMIZATION". In: (May 1959). DOI: 10.2172/4252678. URL: https://www.osti.gov/biblio/4252678.

[8] *An Extensible Architecture for Next Generation Scalable Parallel Reservoir Simulation*. Vol. All Days. SPE Reservoir Simulation Conference. SPE-93274-MS. Jan. 2005. DOI: 10.2118/93274-MS. eprint: https://onepetro.org/spersc/proceedings-pdf/05RSS/All-05RSS/SPE-93274-MS/1836240/spe-93274-ms.pdf. URL: https://doi.org/10.2118/93274-MS.

[9] R. Fletcher. "A new approach to variable metric algorithms". In: *The Computer Journal* 13.3 (Jan. 1970), pp. 317–322. ISSN: 0010-4620. DOI: 10.1093/comjnl/13.3.317. eprint: https://academic.oup.com/comjnl/article-pdf/13/3/317/988678/130317.pdf. URL: https://doi.org/10.1093/comjnl/13.3.317.

[10] *fmincon*. URL: https://de.mathworks.com/help/optim/ug/fmincon.html#busp5fq-6.

[11] Donald Goldfarb. "A family of variable-metric methods derived by variational means". In: *Mathematics of Computation* 24 (1970), pp. 23–26.

[12] Geoffrey Gordon. "The Development of the General Purpose Simulation System (GPSS)". In: *SIGPLAN Not.* 13.8 (Aug. 1978), pp. 183–198. ISSN: 0362-1340. DOI: 10.1145/960118.808382. URL: https://doi.org/10.1145/960118.808382.

[13] J. D. Jansen et al. "The egg model – a geological ensemble for reservoir simulation". In: *Geoscience Data Journal* 1.2 (2014), pp. 192–195. DOI: https://doi.org/10.1002/gdj3.21. eprint: https://rmets.onlinelibrary.wiley.com/doi/pdf/10.1002/gdj3.21. URL: https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/gdj3.21.

[14] J.D. Jansen. "Adjoint-based optimization of multi-phase flow through porous media – A review". In: *Computers & Fluids* 46.1 (2011). 10th ICFD Conference Series on Numerical Methods for Fluid Dynamics (ICFD 2010), pp. 40–51. ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2010.09.039. URL: https://www.sciencedirect.com/science/article/pii/S0045793010002677.

[15] Zhaoyang Larry Jin, Yimin Liu, and Louis J. Durlofsky. "Deep-learning-based surrogate model for reservoir simulation with time-varying well controls". In: *Journal of Petroleum Science and Engineering* 192 (2020), p. 107273. ISSN: 0920-4105. DOI: https://doi.org/10.1016/j.petrol.2020.107273. URL: https://www.sciencedirect.com/science/article/pii/S0920410520303533.

[16] Georgios Kanellis, Antti Oksanen, and Jukka Konttinen. "Adjoint-based optimization in the development of low-emission industrial boilers". In: *Engineering Optimization* 53.7 (2021), pp. 1230–1250. DOI: 10.1080/0305215X.2020.1781842. eprint: https://doi.org/10.1080/0305215X.2020.1781842. URL: https://doi.org/10.1080/0305215X.2020.1781842.

[17] N. Karmarkar. "A new polynomial-time algorithm for linear programming". In: *Combinatorica* 4.4 (1984), pp. 373–395. DOI: 10.1007/BF02579150. URL: https://doi.org/10.1007/BF02579150.

[18] Joonyi Kim, Hyungjun Yang, and Jonggeun Choe. "Robust optimization of the locations and types of multiple wells using CNN based proxy models". In: *Journal of Petroleum Science and Engineering* 193 (2020), p. 107424. ISSN: 0920-4105. DOI: https://doi.org/10.1016/j.petrol.2020.107424. URL: https://www.sciencedirect.com/science/article/pii/S0920410520304964.

[19] *Least-Squares (Model Fitting) Algorithms Matlab*. URL: https://de.mathworks.com/help/optim/ug/least-squares-model-fitting-algorithms.html#f204.

[20] M.C. Leverett. "Capillary Behavior in Porous Solids". In: *Transactions of the AIME* 142.01 (Dec. 1941), pp. 152–169. ISSN: 0081-1696. DOI: 10.2118/941152-G. eprint: https://onepetro.org/TRANS/article-pdf/142/01/152/2178004/spe-941152-g.pdf. URL: https://doi.org/10.2118/941152-G.

[21] *Licensing policy for oil and gas production in Norway*. URL: https://www.norskoljeoggass.no/en/industry-policy2/licensing-policy/#:~:text=One%20of%20the%20Norwegian%20Oil,term%20development%20of%20the%20NCS..

[22] Knut-Andreas Lie. *An Introduction to Reservoir Simulation Using MATLAB/GNU Octave: User Guide for the MATLAB Reservoir Simulation Toolbox (MRST)*. Cambridge University Press, 2019. DOI: 10.1017/9781108591416.

[23] Knut-Andreas Lie and Olav Møyner, eds. *Advanced Modeling with the MATLAB Reservoir Simulation Toolbox*. Cambridge University Press, Oct. 2021. ISBN: 9781009019781. DOI: 10.1017/9781009019781.

[24] *lsqnonlin*. URL: https://de.mathworks.com/help/optim/ug/lsqnonlin.html.

[25] Leslie B. Magoon. "Petroleum System: Nature's Distribution System for Oil and Gas". In: *Encyclopedia of Energy*. Ed. by Cutler J. Cleveland. New York: Elsevier, 2004, pp. 823–836. ISBN: 978-0-12-176480-7. DOI: https://doi.org/10.1016/B0-12-176480-X/00251-5. URL: https://www.sciencedirect.com/science/article/pii/B012176480X002515.

[26] *MATLAB Reservoir Simulation Toolbox (MRST)*. URL: https://www.sintef.no/projectweb/mrst/.

[27] *MRST-co2lab*. URL: https://www.sintef.no/projectweb/mrst/modules/co2lab/.

[28] Jiří Navrátil et al. "Accelerating Physics-Based Simulations Using End-to-End Neural Network Proxies: An Application in Oil Reservoir Modeling". In: *Frontiers in Big Data* 2 (2019). DOI: 10.3389/fdata.2019.00033. URL: https://www.frontiersin.org/article/10.3389/fdata.2019.00033.

[29] Richard D. Neidinger. "Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming". In: *SIAM Review* 52.3 (Jan. 2010), pp. 545–563. DOI: 10.1137/080743627. URL: https://doi.org/10.1137/080743627.

[30] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. New York, NY, USA: Springer, 1999.

[31] Louis B. Rall. *Automatic Differentiation: Techniques and Applications*. Vol. 120. Lecture Notes in Computer Science. Springer, 1981. ISBN: 3-540-10861-0. DOI: 10.1007/3-540-10861-0. URL: https://doi.org/10.1007/3-540-10861-0.

[32] Atgeirr Rasmussen et al. "The Open Porous Media Flow reservoir simulator". In: *Computers & Mathematics with Applications* 81 (June 2020). DOI: 10.1016/j.camwa.2020.05.014.

[33] *Release Notes for MRST 2019b*. URL: https://www.sintef.no/projectweb/mrst/download/release-notes-for-mrst-2019b/.

[34] *Numerical Comparison Between ES-MDA and Gradient-Based Optimization for History Matching of Reduced Reservoir Models*. Vol. Day 1 Tue, October 26, 2021. SPE Reservoir Simulation Conference. D011S002R005. Oct. 2021. DOI: 10.2118/203975-MS. eprint: https://onepetro.org/spersc/proceedings-pdf/21RSC/1-21RSC/D011S002R005/2508775/spe-203975-ms.pdf. URL: https://doi.org/10.2118/203975-MS.

[35] David F. Shanno. "Conditioning of quasi-Newton methods for function minimization". In: *Mathematics of Computation* 24.111 (1970), pp. 647–656.

[36] *Reduced Order Reservoir Simulation with Neural-Network Based Hybrid Model*. Vol. Day 3 Thu, October 24, 2019. SPE Russian Petroleum Technology Conference. D033S018R004. Oct. 2019. DOI: 10.2118/196864-MS. eprint: https://onepetro.org/SPERPTC/proceedings-pdf/19RPTC/3-19RPTC/D033S018R004/1131267/spe-196864-ms.pdf. URL: https://doi.org/10.2118/196864-MS.

[37] Stephen Whitaker. "Flow in porous media I: A theoretical derivation of Darcy's law". In: *Transport in Porous Media* 1.1 (1986), pp. 3–25. DOI: 10.1007/BF01036523. URL: https://doi.org/10.1007/BF01036523.