Master's thesis

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

Ingvild Strømsheim Devold

Graph-based methods for data-driven reservoir modeling

Master's thesis in Industrial Mathematics
Supervisor: Knut-Andreas Lie
Co-supervisor: Øystein Klemetsdal and Stein Krogstad
June 2023

NTNU
Norwegian University of
Science and Technology

Ingvild Strømsheim Devold

Graph-based methods for data-driven reservoir modeling

Master's thesis in Industrial Mathematics
Supervisor: Knut-Andreas Lie
Co-supervisor: Øystein Klemetsdal and Stein Krogstad
June 2023

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

NTNU
Norwegian University of
Science and Technology

Abstract

Data-driven approaches in reservoir modeling range from fully data-driven machine learning techniques to history matching of traditional mathematical models. In this thesis, we propose hybrid, graph-based models which are both data-driven and physically consistent. The models are based on a standard finite-volume discretization formulated on geometrically flexible graphs, and their parameters are calibrated freely by adjoint-based optimization to give predictions matching observed behavior. The computational costs herein call for the use of reduced-order models. To that end, we consider two main approaches to construct coarse graphs. A CGNet is based on a coarse partition of the original grid, whereas a TriNet is constructed from a triangulation of the wells, selected points along the domain boundary and possibly additional internal nodes. The resulting models offer a rich set of connections between wells, and thus a larger set of tunable parameters than a typical interwell network model. Furthermore, we consider different approaches to construct non-uniform graphs adapting to the flow, including a priori techniques using residence times and distance to well, and a refinement algorithm where the graph automatically refines itself based on parameter sensitivities.

The results show that surprisingly coarse models can be calibrated to successfully mimic the observed behaviour. We observe that sufficiently accurate estimates for physical quantities like pore volumes and initial saturations are a prerequisite for successful calibration. If we have a poor initial saturation guess, including it in the calibration gives significantly improved results. The automatic refinement does generally not give better results than a uniform model, but offers the advantage of not being dependent of the user's understanding of the flow physics, and enables starting from a very coarse graph.

Sammendrag

Datadrevne metoder innen reservoarmodellering inkluderer både fullt ut databaserte maskinlæringsmetoder og historietilpasning av tradisjonelle matematiske modeller. I denne oppgaven foreslår vi en type hybride, grafbaserte modeller som er både datadrevne og fysikkbaserte. Modellene baserer seg på en standard endelig volum-metode, formulert på geometrisk fleksible grafer. Modellparametere kalibreres fritt gjennom adjungert-basert optimering for å gi prediksjoner som samsvarer med observasjoner, og beregningskostnadene her krever at vi bruker reduserte modeller. I den hensikt studerer vi to metoder for å konstruere grove grafer. En CGNet-modell er basert på en grov partisjon av det opprinnelige gridet, mens en TriNet-modell er basert på en triangulering av brønnpunktene, utvalgte punkter langs randen, og eventuelt noen ekstra interne noder. De resulterende modellene har et rikt antall koblinger mellom brønner, og dermed en større mengde trenbare parametere enn en typisk inter-brønn nettverksmodell. Videre diskuteres ulike metoder for å lage ikke-uniforme grafer som er tilpasset flyten, inkludert a priori-teknikker basert på residenstider og avstand til nærmeste brønn, og en forfiningsalgoritme hvor grafen automatisk forfiner seg selv basert på parametersensitiviteter.

Resultatene viser at overraskende grove modeller kan kalibreres for å replikere observert reservoaroppførsel. Vi observerer at tilstrekkelig gode estimater for fysiske størrelser som porevolumer og initialmetninger er en forutsetning for å lykkes med kalibreringen. I de tilfeller hvor vi har en dårlig gjetning for initiell vannmetning, viser vi at å inkludere dette som en trenbar parameter gir betydelig forbedring av resultatet. Den automatiske forfiningsalgoritmen gir generelt ikke bedre resultater enn en uniform modell, men har den fordelen at den ikke er avhengig av brukerens forståelse av fysikken, og muliggjør å starte fra en veldig grov modell.

Preface

This thesis concludes my five-year M.Sc. in Applied Physics and Mathematics with a specialization in Industrial Mathematics at the Norwegian University of Science and Technology. The project has been carried out in collaboration with the Computational Geosciences group at SINTEF Digital.

I would like to thank my supervisors Knut-Andreas Lie, Stein Krogstad and Øystein Klemetsdal for suggesting the topic, for their many ideas and suggestions along the way, for their help with both programming and writing, and for always being available and enthusiastic. I appreciate the opportunity to stay in SINTEF's Oslo office for the past months and am thankful also to the remaining Computational Geosciences group members for welcoming me and making this time enjoyable.

Ingvild Strømsheim Devold
Oslo, Norway
June 2023

Contents

Chapter 1

Introduction

Modeling flow in porous media is of interest and importance in applications spanning from chemistry and biology to geology. The latter is particularly motivated by the petroleum industry, where simulations are the backbone of important decision-making processes, essential for both economical, environmental, and safety reasons. Hydrocarbon reservoirs are highly complex systems, often exhibiting both complicated geometries and non-trivial interactions between different fluids. Consequently, a detailed reservoir model is often large and complex. In fact, a conventional physics-based reservoir model can have millions of cells, implying that significant computational costs are associated with a single forward simulation. While recent advances in computing power have reduced some of the computational obstacles, too complex models remain computationally prohibitive in certain applications. Although running a day-long simulation once may be fully feasible for the patient researcher, doing so repeatedly soon becomes problematic. In an optimization setting, such as production optimization or parameter calibration, hundreds of simulation runs may be required. This renders the traditional fine-scale model intractable and calls for the use of reduced-order or proxy models. The same applies to digital twins, as one of their crucial components is continuous model updates from data and this is computationally challenging for complex models.

Data-driven methods have evolved at an exponential speed in the last decades, making their grand entry in all branches of science, natural and social alike. The utilization of large and ever-expanding data sets has become a focal point in industry, and data-driven methods have been suggested also as replacements for physics-based models. Reservoir modeling is no exception to this trend. Neural network-based models have been used to, e.g., simulate gas reservoirs about 10^8 times faster than a commercial simulator [1], and as proxy models in a history matching setting [2]. However, it is important to acknowledge the limitations of such purely data-driven machine-learning models. Machine-learning methods in general require large amounts of measured or simulated data and are computationally costly to train. More importantly, in the context of reservoir simulation, it is vital that the results are physically reliable, as inaccurate simulations can lead to severe consequences. A purely data-driven model lacks an understanding of the underlying physics and is solely trained to give the desired input-output relation. Even if the model accurately predicts responses for its training data, there is no guarantee that it will give physically consistent predictions for unseen input data. To address this, physics-informed neural networks have been proposed. These incorporate the physics, either by learning the governing equations directly, or by penalizing physically invalid solutions, see, e.g., [3, 4].

In this thesis, we propose a different, hybrid approach, where we keep our mathematical model of the flow physics, and formulate coarse or reduced models which can be calibrated to replicate observed behavior or simulated data. Specifically, we formulate the coarse models using geometrically flexible graphs. This is motivated by the recognition that a conventional finite-volume reservoir simulator can in fact be interpreted as a computational graph. Nodes store fluids and edges transmit them. This perspective poses little restriction on the model geometry, and inspires the use of more flexible graphs.

Previous research on network models for reservoirs includes, among others, GPSNet, which models the reservoir using one-dimensional flow paths between wells [5], and StellNet, which instead uses 3D flow paths and a standard finite-volume discretization [6]. Both these models fall within the category of interwell network models, which can be seen as a graph-based analogue to streamline methods, and have also been studied in, e.g., [7, 8, 9] under the name interwell numerical simulation models (INSIM). We shall focus instead on models with a richer topology, more possible flow paths between wells, and consequently a larger set of tunable parameters. Herein, a straightforward model reduction is to use a graph whose topology mimics that of a three-dimensional coarse finite-volume grid. This strategy has previously been employed in [10, 11, 12], and we will adopt the naming CGNet (coarse-grid network) for such models. A fine model can easily be reduced to a CGNet through partitioning. We could, however, imagine that we do not have a fine-scale model in the first place. In that case, we propose a triangulation-based model type which requires little information about the reservoir geometry and geology. Here, the main idea is to construct a graph from some triangulation of the wells and selected points along the boundary, and we will refer to this model type as TriNet.

When tampering with the geometry, we will heavily rely on the ability to calibrate the models afterwards. In particular, we calibrate the model to reproduce observed well responses. To that end, we take physical parameters from our mathematical model and treat them as tunable parameters which we adjust almost freely to get the desired input-output relation. More specifically, the parameters are calibrated by means of the Levenberg–Marquardt algorithm, a form of adjoint-based optimization. This approach differs from the traditional history matching setting, since we do not seek universally valid models, do not interpret the parameter values as physical, and do not penalize deviation from an a priori geological model.

The resulting methods are hybrid, that is, both physics-based and data-driven. This allows us to exploit available data, but without discarding our knowledge of physics. The coarse nature of the models enables the use of computer-intensive methods like the Levenberg–Marquardt algorithm, using data to calibrate the model. Moreover, since the simulations still use the standard discretization of the flow equations, we can expect physically meaningful predictions, even if the parameter values themselves are no longer physical.

Uniform models are often the default choice, but in some cases, the system we attempt to model has features suggesting a non-uniform approach. For realistic reservoirs, hydrocarbons may for instance be concentrated in one part of the domain, or wells distributed in such a way that some parts of the reservoir have limited fluid flow, and there is little information to be deducted from data considering fluid injection and production in wells. In such cases, we often try to tailor our numerical methods, using higher resolution in the more active parts of the domain. This can be accomplished in different ways. First, we can do it a priori, before calibration, by computing a flow indicator such as residence time [13]. Second, we can use a more naïve approach, perhaps best described as an educated guess, increasing the resolution in the near-well regions. Finally, we can take the more passive route, allowing the model to decide its own resolution through some automatic refinement procedure. We will compare these strategies with the zero-thinking alternative of using a uniform model.

1.1 Contribution

The overarching goal of this thesis is to explore the use of graph-based methods in reservoir modeling. Since these models directly incorporate the physics by using the discretized flow equations, they can potentially offer a more physically sound alternative to machine-learning methods. Moreover, given a successful calibration, the coarse nature of the models can prove useful in applications like control optimization.

Specifically, we present and compare the two main model types CGNet and TriNet. Herein, we test the effect of tweaking the topology. In addition to the a priori flow adaptation, we suggest an automatic refinement algorithm for the triangulation-based models. Furthermore, we investigate the value of information, testing whether a model using more information from the original model

is easier to calibrate. Here, one could argue that there are two main settings. In the data-driven modeling context, we construct the models using little to no information from the original model, and calibrate it as almost a black-box model to give the desired predictions. On the other hand, we can try to map as much information as possible from the original model, which is easy if we use a partitioning approach. Moreover, we test the influence of different parameters in calibration. In addition to comparing calibration results, we briefly test and compare the generality and predictive abilities of the models. Given a successfully calibrated model, we also demonstrate its use in a control optimization application.

This project has expanded the work on CGNet models from [10, 11, 12], by looking at different use cases, constructing a priori flow-adapted models from residence-time fields, as well as introducing some new tunable parameters. The triangulation-based models, TriNet, are new altogether, and suggested as a more radically data-driven alternative, applicable also when you do not have a fine-scale reservoir model in the first place. Programming has been a major part of this project and the code is openly available from a Bitbucket repository[1].

We emphasize that the thesis builds on and expands the results of the author's specialization project carried out in the fall semester 2022 [14]. There is some overlap in order to make this text self-contained. Chapters or sections that are reused with only minor modifications are marked by an asterisk (*). This mainly includes the background chapters 2–4 and Appendix A. All material on partition-based models is new. The triangulation-based models were also used in the specialization project, but have been extended to both 2.5D models and a priori flow-adapted models. Moreover, gravitational effects and initial water saturation have been added as tunable parameters, the latter giving significantly improved results. The codebase has faced a major reorganization aiming at improved generality, and thus easier testing of modified methods. As for the simulation results, a demonstration case has been reused with some modifications. The SAIGUP model is now used with its original initial saturation, as opposed to in the specialization project where we initialized the reservoir with oil only. All results on the Egg model, the Norne field, and the Brugge benchmark case are new.

1.2 Outline

This thesis is structured as follows. Chapters 2–4 constitute the background part. In particular, Chapter 2 is devoted to the theory behind flow in porous media, deriving the flow equations for single- and multiphase flow. Next, Chapter 3 outlines the steps to discretize these equations using a finite-volume method with a two-point flux approximation. Chapter 4 explains how model parameters can be calibrated using the adjoint-based Levenberg–Marquardt optimization algorithm. In Chapter 5, the graph-based perspective is motivated, and the two main model types are introduced. Chapter 6 holds the simulation results for a number of cases, and finally, Chapter 7 contains some concluding remarks and suggestions for future work. In addition, Appendix A gives a brief introduction to the basic definitions and notations of graph theory.

[1] https://bitbucket.org/ingvilddevold/graph-based-methods/

Chapter 2

Flow in Porous Media*

Modeling flow in porous media is of interest in a wide range of applications, spanning from geophysical flow to chemical, hydrological, and even physiological phenomena. The most well-known application is perhaps hydrocarbon recovery, an industry where accurate modeling is essential for both economical, environmental, and safety reasons. The first step towards an accurate simulation is a mathematical model. In this chapter, the physical and mathematical foundations of reservoir simulation are outlined. A full description of flow in a reservoir involves three key components: a geological model describing the subsurface reservoir, a flow model describing fluid flow within the porous rock, and a model of the wells and near-well region.

While the presentation in this chapter is tailored to the hydrocarbon reservoir setting, many of the fundamental principles in discussion are readily applicable in other contexts. For example, both geothermal energy and carbon storage modeling share a common focus on subsurface geological formations and fluid flow. See, for instance, [15, 16, 17] to learn more about modeling geothermal energy systems, and [18] for a general overview of carbon storage modeling and simulation.

2.1 Geological model

Hydrocarbon reservoirs are characterized by porous rocks, which on the small scale consist of small pores between grains of solid, as illustrated in Figure 2.1. These void spaces allow fluids to be stored in and transmitted through the reservoir. When simulating flow at the scale of a reservoir, we are neither interested nor able to describe storage and transport at the pore-scale. Instead, we attempt to build models that capture the geological properties of the rock at the macroscopic

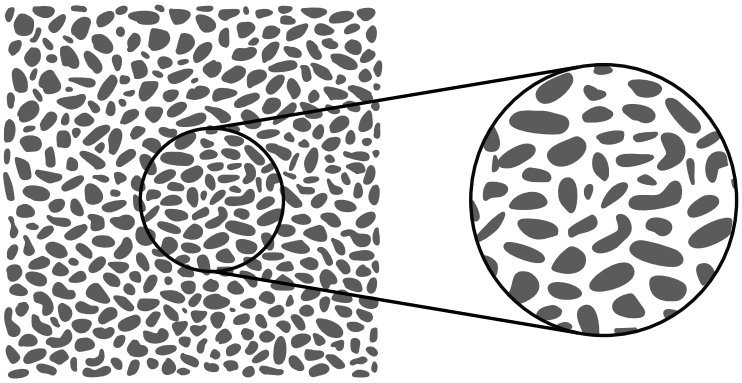


Figure 2.1: A conceptual illustration of a porous medium on the microscopic level, including a representative elementary volume (REV). The gray parts are the grains, and the open white space represents the void space available for fluids.

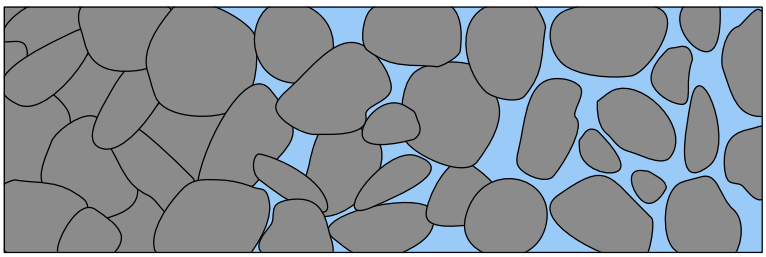


Figure 2.2: A conceptual illustration of the porosity-permeability relation. The permeability depends on both the distribution and shapes of the void spaces. Here, the leftmost part is both non-porous and non-permeable, the middle part is porous and non-permeable, whereas the rightmost part is both porous and permeable.

scale. These models use macroscopic petrophysical properties based on a continuum hypothesis and volume averaging over representative elementary volumes (REVs), which we define as the smallest volume where a property can be measured and be representative of the entire volume [19]. Figure 2.1 illustrates such a representative elementary volume.

In reservoir simulation, the macroscale geological model seeks to represent the reservoir, including both its geometry and its ability to store and transmit fluids. To that end, the reservoir rock is modeled by a volumetric grid, consisting of grid cells of sizes ranging from $\mathcal{O}(0.1)$–$\mathcal{O}(1)$ meters in the vertical direction and $\mathcal{O}(10)$–$\mathcal{O}(100)$ meters in the horizontal direction. Each grid cell is then assigned a constant value for petrophysical properties such as porosity and permeability [19].

Here, we define the *porosity* ϕ as the fraction of interconnected void space in a rock, and note that ϕ clearly must satisfy the bounds $0 \leq \phi < 1$. We only include interconnected spaces since disconnected unavailable pores are not of particular interest when simulating flow.

While the porosity is a static quantity independent of the flow for fully rigid rocks, it is dependent on the pressure for *compressible* rocks. Introducing the rock compressibility c_r, we have the relation

$$c_r = \frac{1}{\phi}\frac{d\phi}{dp} = \frac{d\ln(\phi)}{dp}, \tag{2.1.1}$$

where p is the overall reservoir pressure.

The *permeability* is a fundamental property of a porous medium describing how easily a fluid can flow through it; more precisely, the medium's ability to transmit a single fluid when the void space is completely filled with it. In effect, this measures the connectivity of the pore spaces. It depends not only on the porosity, but also on how the void spaces are shaped and distributed. For example, winding paths are harder to flow through than straight paths. If the void spaces are not connected, the medium is not permeable at all, as illustrated in Figure 2.2.

Permeability has SI unit m^2, but is more commonly measured in unit millidarcies (mD), with typical values ranging from 100 to 500 mD for hydrocarbon reservoir rock. Here, $1\,\mathrm{D} \approx 0.987 \cdot 10^{-12}\,\mathrm{m}^2$ [19]. The permeability K, together with the fluid viscosity μ, appear as the proportionality factor between the flow rate or macroscopic velocity $\vec{v}$ and pressure or potential gradient $\nabla\Phi$ in Darcy's law, a fundamental concept in flow in porous media, which reads

$$\vec{v} = -\frac{K}{\mu}\nabla\Phi. \tag{2.1.2}$$

Darcy's law stems from the work of the French hydraulic engineer Henry Darcy's work on water flowing through sand [19]. Its physical interpretation is conservation of momentum, and the observant reader may notice that Darcy's law is on the same form as several other physical laws, such as Fourier's law on heat conduction, Ohm's law on electric potential, and Fick's law on diffusion.

Reservoirs commonly exhibit anisotropic permeability, characterized by significant variations in permeability between vertical and horizontal planes, deviating from the assumption of homogeneous (isotropic) permeability. In the general case, we thus need to represent the permeability as a full

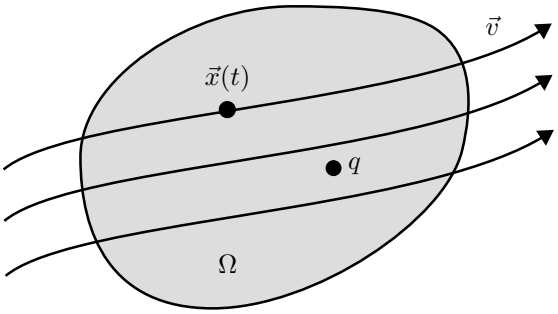


Figure 2.3: A conceptual illustration of the control volume Ω used when deriving the macroscopic continuity equations. Here, q is a source term, $\vec{v}$ the bulk velocity, and $\vec{x}(t)$ the position at time t of a point or imaginary particle moving along the velocity field. Adapted from Figure 4.3 in [19].

tensor in a macroscale model to accurately model local flow in directions at an angle to the coordinate axes. Specifically, we let

$$\boldsymbol{K} = \begin{bmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz} \end{bmatrix}, \tag{2.1.3}$$

where the entry K_{ij} relates the pressure drop in direction j to the flux in direction i. We note that the permeability tensor $\boldsymbol{K}$ must be symmetric and positive definite [19].

With these concepts in place, we may now move on to formulate the simplest case of governing equations for single-phase flow.

2.2 Single-phase flow

The single-phase flow equations arise from the combination of Darcy's law and conservation of mass. For a single-phase fluid, Darcy's law (2.1.2) reads

$$\vec{v} = -\frac{\boldsymbol{K}}{\mu}(\nabla p - g\rho\nabla z), \tag{2.2.1}$$

where p is the fluid pressure, z the vertical coordinate, g the gravitational acceleration, $\boldsymbol{K}$ the permeability tensor, ρ the fluid density, and μ the fluid viscosity. It should be stressed that the Darcy velocity $\vec{v}$ is not an intrinsic fluid velocity in the microscopic sense, but rather an apparent macroscopic velocity, or an average volumetric flux, for the bulk movement of fluid through the medium, obtained from REVs.

Consider a fluid with density $\rho(x,t)$ moving with bulk velocity $\vec{v}(x,t)$, as illustrated in Figure 2.3. Defining some control volume Ω with porosity ϕ, the fluid mass inside that volume is given by

$$\int_\Omega \phi\rho(x,t)d\vec{x}.$$

Conservation of mass implies that the accumulation of mass in the volume equals the sum of the net flow of mass into the volume over its boundary, and the mass from sources inside the volume. This can be expressed mathematically as

$$\underbrace{\frac{\partial}{\partial t}\int_\Omega \phi\rho\, d\vec{x}}_{\text{Accumulation term}} + \underbrace{\int_{\partial\Omega} \rho\vec{v}\cdot\vec{n}\, ds}_{\text{Flux term}} = \underbrace{\int_\Omega \rho q\, d\vec{x}}_{\text{Source term}}. \tag{2.2.2}$$

We can apply the divergence theorem to the flux term, and assuming that the functions are bounded and sufficiently smooth, we can move the time derivative in the accumulation term into

the integral. This gives

$$\int_\Omega \left[\frac{\partial(\phi\rho)}{\partial t} + \nabla \cdot (\rho \vec{v}) \right] d\vec{x} = \int_\Omega \rho q \, d\vec{x}. \tag{2.2.3}$$

Since this holds for arbitrary control volumes Ω, we can reformulate the conservation equation in differential form as

$$\frac{\partial(\phi\rho)}{\partial t} + \nabla \cdot (\rho \vec{v}) = \rho q. \tag{2.2.4}$$

Here, we have more unknowns than equations, and we need some additional equations to get a closed system. To that end, we use constitutive equations, which relate different states of the system to each other.

Recall that the rock compressibility in (2.1.1) describes how the porosity ϕ and pressure p are related. We can define the fluid compressibility relating the fluid density ρ and pressure p analogously. By simple partial differentiation, we obtain

$$\frac{dV}{V} = \frac{1}{V} \left(\frac{\partial V}{\partial p} \right)_T dp + \frac{1}{V} \left(\frac{\partial V}{\partial T} \right)_p dT, \tag{2.2.5}$$

where the subscripts indicate which variables are kept constant. Now assuming a constant number of particles, we have that ρV is constant, and thus $V d\rho = -\rho dV$. Exploiting this, we get

$$\frac{d\rho}{\rho} = \frac{1}{\rho} \left(\frac{\partial \rho}{\partial p} \right)_T dp + \frac{1}{\rho} \left(\frac{\partial \rho}{\partial T} \right)_p dT = c_f dp + \alpha_f dT, \tag{2.2.6}$$

where we have introduced the *isothermal compressibility* c_f and the *thermal expansion coefficient* α_f. In subsurface systems, the density change is typically slow, allowing heat conduction to keep the temperature constant [19]. In that case, we can disregard the last term, and (2.2.6) simplifies to

$$c_f = \frac{1}{\rho} \frac{d\rho}{dp} = \frac{d \ln(\rho)}{dp}, \tag{2.2.7}$$

similar to the rock compressibility definition in (2.1.1). We refer to c_f as the *fluid compressibility*, and note that it is non-negative and generally dependent on pressure and temperature.

Combining Darcy's law (2.2.1) and the rock and fluid compressibilities with the mass conservation equation (2.2.4), we arrive at the parabolic equation for fluid pressure,

$$c_t \phi \rho \frac{\partial p}{\partial t} - \nabla \cdot \left[\frac{\rho \boldsymbol{K}}{\mu} (\nabla p - g \rho \nabla z) \right] = \rho q, \tag{2.2.8}$$

where the compressibilities have been combined into the *total compressibility* $c_t = c_r + c_f$.

2.3 Multiphase flow

For hydrocarbon reservoirs, single-phase flow rarely makes a sufficient model. In most applications, the point of interest is how one fluid phase displaces others, such as water displacing oil in water-flooding oil recovery. A multiphase and multicomponent flow model is needed to accurately represent the system. For such a model, we need to define three new physical properties, namely the saturation, relative permeability, and capillary pressure, which will be used to extend Darcy's law. Combining this with mass conservation for each fluid phase (or fluid component), we arrive at a system of governing partial differential equations describing the multiphase flow.

2.3.1 Physical properties

We consider the setting with two or more *immiscible* fluid phases, meaning that the phases do not mix. There is no mass transfer between the phases. Figure 2.4 shows a representative elementary volume for two-phase water-oil flow.

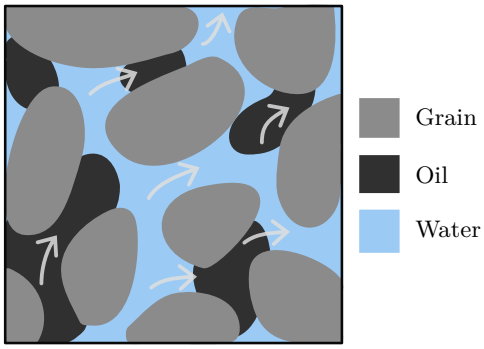


Figure 2.4: A representative elementary volume for two-phase flow in a reservoir.

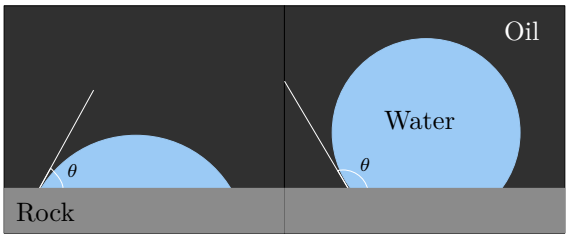


Figure 2.5: An illustration of wettability. Here, the left panel shows a water wet system ($0° \leq \theta < 90°$), whereas the right panel illustrates an oil wet system ($90° \leq \theta < 180°$), where θ is the contact angle.

We now define the *saturation* S_α as the fraction of the pore volume occupied by phase α. In the single-phase setting, we assumed that the void space was completely filled with the present fluid. The multiphase assumption is similar, stating that the void space is completely filled with one or more fluid phases [19]. Thus, the saturations must sum to unity,

$$\sum_\alpha S_\alpha = 1. \tag{2.3.1}$$

The *wettability* of a liquid phase is its ability to maintain contact with a solid surface. When there are two immiscible fluid phases, the cohesion forces between molecules of the same phase are greater than the adhesive forces between molecules in different phases, which causes a surface to form between the two phases. The associated surface tension measures the force required to change the shape of the surface. In a reservoir, molecules are also attracted to the surface of the rock. When there are two liquid phases in a pore space, one will be more drawn to the rock than the other, and we refer to that phase as the wetting phase. The other is called the non-wetting phase. Figure 2.5 illustrates a water wet versus an oil wet system, that is, when water or oil, respectively, is the wetting phase. Due to the surface tension on the interface between the two phases, there will generally be a difference in the equilibrium pressure [19]. We define this as the *capillary pressure*,

$$p_c = p_n - p_w, \tag{2.3.2}$$

where we have used subscripts n and w to denote the non-wetting and wetting phase, respectively. The capillary pressure will always be positive, since the pressure in the non-wetting phase is always greater than the pressure in the wetting phase.

Recall from the single-phase setting that the permeability $\boldsymbol{K}$ measured the rock's ability to transmit fluids. When there are several fluid phases, they will interfere with each other, one acting as an additional obstacle for the other. The resulting interfacial tensions will slow down the flow [19]. As a result, each phase α will experience an *effective* permeability $\boldsymbol{K}_\alpha^e$ which is smaller than the intrinsic rock permeability $\boldsymbol{K}$. Also the sum of the effective phase permeabilities will generally be less than the original $\boldsymbol{K}$,

$$\sum_\alpha \boldsymbol{K}_\alpha^e < \boldsymbol{K}. \tag{2.3.3}$$

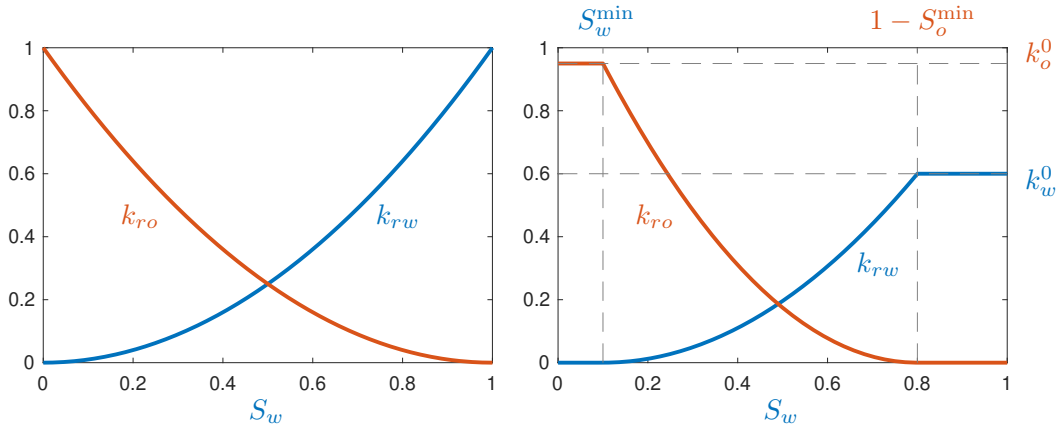


Figure 2.6: Brooks-Corey relative permeabilities with $n_w = n_o = 2$. In the right plot, there is residual saturation, and the maximum relative permeabilities are less than 1. Adapted from Figure 2.6 in [20].

Modeling this flow reduction exactly is not trivial, and the common approach is to use the *relative permeability*. This multiplicative factor will always lie between 0 and 1, and for an isotropic medium it is defined as

$$k_{r\alpha} = \frac{K_\alpha^e}{K}. \tag{2.3.4}$$

Even though the relationship may be different for each component in the anisotropic case, it is common to still define the relative permeability as scalar, with

$$\boldsymbol{K}_\alpha^e = k_{r\alpha}\boldsymbol{K}. \tag{2.3.5}$$

The relative permeabilities are typically assumed to be monotone functions of the saturations, and are either given as tabulated quantities or represented by simple analytic relationships. The latter often use the normalized, or effective, saturation,

$$\hat{S}_w = \frac{S_w - S_w^{\min}}{S_w^{\max} - S_w^{\min}}, \tag{2.3.6}$$

where $S_w^{\min}$ and $S_w^{\max}$ are the minimum and maximum values the saturation can take during displacement. A straightforward model for relative permeability is the Corey model, which is a power-law relationship describing the behavior of a two-phase water-oil system,

$$\begin{aligned} k_{rw} &= (\hat{S}_w)^{n_w} k_w^0, \\ k_{ro} &= (1 - \hat{S}_w)^{n_o} k_o^0, \end{aligned} \tag{2.3.7}$$

where the exponents $n_w, n_o \geq 1$ and the constants k_w^0, k_o^0 are used for end-point scaling and should both be fit based on measured data [19]. Figure 2.6 shows two examples, both with $n_w = n_o = 2$, but with different residual saturations and maximum relative permeabilities.

2.3.2 Flow equations

To obtain the governing equations for the multiphase flow setting, we again use conservation of mass. To begin with, we consider immiscible, single-component phases. Requiring that the mass is conserved for each phase α, we get a set of equations on the form

$$\frac{\partial}{\partial t}(\phi\rho_\alpha S_\alpha) + \nabla \cdot (\rho_\alpha \vec{v}_\alpha) = \rho_\alpha q_\alpha. \tag{2.3.8}$$

Using the relative permeabilities, we can formulate an extended version of Darcy's law applicable to multiphase flow,

$$\vec{v}_\alpha = -\lambda_\alpha \boldsymbol{K}(\nabla p_\alpha - g\rho_\alpha \nabla z), \tag{2.3.9}$$

where we have introduced the *phase mobility* $\lambda_\alpha = k_{r\alpha}/\mu_\alpha$ as a shorthand.

The flow equations can be reformulated in a number of ways, depending on which phases are present and whether they represent immiscible or miscible, single- or multicomponent fluid systems. For our discussion, we will state the *black-oil model*, which is the industry standard for oil and gas simulations. This is a simplified compositional model with no diffusion among the components, using three phases: the liquid oleic phase (o), the gaseous phase (g) and the aqueous phase (w). Likewise, we have three components: water, hydrocarbons that appear in the liquid phase at surface conditions (oil), and hydrocarbons that appear as gases at surface conditions (gas).

The two hydrocarbon components will partition differently among the two hydrocarbon phases depending upon the (phase) pressure. We introduce the shrinkage factor $b_\alpha = V_{\alpha s}/V_\alpha$, which is the volume occupied by phase α at surface conditions divided by the volume occupied at reservoir condition, and let R_s model the solubility of gas in the oleic phase and R_v the solubility of oil in the gaseous phase. These can be used to express the density of each phase, resulting in conservation of mass for the respective fluid components as

$$\begin{aligned} \partial_t[\phi(b_o S_o + b_g R_v S_g)] + \nabla \cdot (b_o \vec{v}_o + b_g R_v \vec{v}_g) - (b_o q_o + b_g R_v q_g) &= 0 \\ \partial_t(\phi b_w S_w) + \nabla \cdot (b_w \vec{v}_w) - b_w q_w &= 0 \\ \partial_t[\phi(b_g S_g + b_o R_s S_o)] + \nabla \cdot (b_g \vec{v}_g + b_o R_s \vec{v}_o) - (b_g q_g + b_o R_s q_o) &= 0. \end{aligned} \tag{2.3.10}$$

Consult, e.g., [19] for a detailed explanation.

2.4 Well model

To extract hydrocarbons from a reservoir, a set of wells are drilled into the porous rock. We separate between producers, where the hydrocarbons are brought to the surface, and injectors, where fluids are injected into the reservoir to increase or maintain the reservoir pressure, or retard its natural decline, thereby contributing to push the hydrocarbons towards the producers.

The diameter of a well is typically less than a meter, and thus much smaller than the size of a grid cell, which can be hundreds of meters wide. While the pressure variations are often small far from the wells, allowing the simplification of constant pressure within each cell, this is inaccurate close to the wells. This motivates the introduction of a well model, providing better accuracy in the near-well regions.

Wells are normally controlled by requiring that the injected or produced fluids satisfy a given surface rate or bottom-hole pressure. By bottom-hole pressure we mean the pressure at some point inside the wellbore, typically at the bottom-most perforation. A well model attempts to describe the pressure at the well radius when the injection or production rate is known, or vice versa. The model takes the form of an *inflow-performance relation*. The simplest example is the linear law

$$q_0 = J(p_R - p_w), \tag{2.4.1}$$

where q_0 is the flow rate, p_R the average pressure in the cell that contains the well, p_w the pressure at the wellbore, and J is a proportionality constant which we call *productivity index* for production wells and *well injectivity index* for injection wells. For the standard Peaceman type well model [21], the well indices are calculated from the analytical solution of an infinitely repeated five-spot pattern. We refer to [19, Chapter 4] for a more detailed explanation.

2.5 The full model

Let us conclude the chapter by gluing the pieces together, arriving at the full mathematical model of flow in a reservoir. For each phase α we have three equations,

$$\frac{\partial}{\partial t}(\phi\rho_\alpha S_\alpha) + \nabla\cdot(\rho_\alpha\vec{v}_\alpha) = \rho_\alpha q_\alpha, \tag{2.5.1a}$$

$$\vec{v}_\alpha = -\lambda_\alpha \boldsymbol{K}(\nabla p_\alpha - g\rho_\alpha\nabla z), \tag{2.5.1b}$$

$$q_\alpha = \lambda_\alpha^{wb} J(p^{wb} - p_\alpha), \tag{2.5.1c}$$

representing the conservation of mass inside the reservoir, Darcy's law, and the inflow/outflow relationship for individual wells. The different quantities entering these equations are:

t	– time	$\boldsymbol{K}$	– permeability tensor
ϕ	– porosity	p_α	– pressure
α	– phase subscript	g	– gravitational acceleration
ρ_α	– density	z	– depth coordinate
S_α	– saturation	λ_α^{wb}	– wellbore mobility
$\vec{v}_\alpha$	– macroscopic (Darcy) velocity	J	– well index
q_α	– source term	p^{wb}	– wellbore pressure
λ_α	– mobility $k_{r\alpha}/\mu_\alpha$		

In the following chapter, we will see how this model can be discretized using a finite-volume method.

Chapter 3

Discretization*

The mathematical model derived in Chapter 2 is fairly complex and, as most equations describing real-world physics, far beyond what we can solve analytically. We need to resort to numerical solutions. To this end, we will now describe a discretization of the model. The discretization is based on a finite-volume method with a two-point flux approximation for the Laplace operator and upstream mobility weighting for all hyperbolic transport terms. We also introduce the open-source MATLAB Reservoir Simulation Toolbox (MRST) [19, 22], in which this discretization is implemented using automatic differentiation and discrete differential and averaging operators for code brevity and flexibility.

3.1 The finite-volume method

The finite-volume method is a popular choice for spatial discretization of partial differential equations in the form of conservation laws. Considering some computational domain Ω, the first step is to divide the domain into a set of polyhedra $\Omega_i \subset \Omega$, $i = 1, \ldots, n_c$, which we may refer to as *finite volumes*, *control volumes* or *control cells*. We typically require that the cells should cover the entire domain, $\cup_i \Omega_i = \Omega$, and be pairwise disjoint, or non-overlapping; see [23]. Figure 3.1 shows an example of a hexagonal control volume.

Having a set of control volumes, the conservation law is formulated on each individual cell Ω_i. Herein, we represent the conserved quantity, say u, for each cell by its cell average, $\overline{u}$. We assume that the cell-averaged quantities are multiplicative, meaning that if $\overline{ab}$ is conserved, then $\overline{ab} = \overline{a} \cdot \overline{b}$. Moreover, we often need to reconstruct the pointwise solution, e.g., to calculate fluxes. To that end, we make an additional assumption about the shape of u inside the cell. We will use a variant of this method to derive the spatial discretization of the governing equations for single- and multiphase flow.

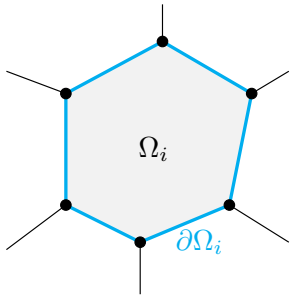


Figure 3.1: A typical two-dimensional hexagonal control volume used in a finite-volume method. We use the convention that the boundary is part of the cell, $\partial\Omega_i \subset \Omega_i$.

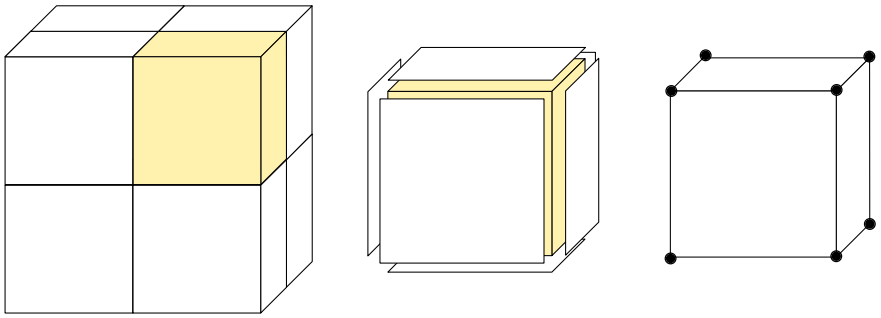


Figure 3.2: A three-dimensional grid consisting of grid cells. The cells are delimited by a set of faces, the faces by a set of edges, and the edges by two vertices.

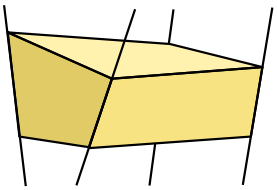


Figure 3.3: A single cell in a corner-point grid, with the four grid lines that together define a pillar of cells stacked on top of each other.

3.2 Computational grid

To perform a spatial discretization of the flow equations, we first need a computational grid representing our reservoir domain Ω, consisting of a set of cells. In the two-dimensional case, a cell is in general a closed polygon, which is defined by a set of vertices and a set of edges connecting two vertices and representing the interface between cells. In three dimensions, the definition can be extended to a closed polyhedron, which is defined by a set of vertices, a set of edges connecting pairs of vertices, and a set of faces representing the interfaces between cells. Figure 3.2 illustrates a three-dimensional grid. We let n_c denote the number of cells and n_f the number of faces.

An industry-standard grid representation in reservoir simulation is the stratigraphic or corner-point grid. In the simplest form, each cell is hexahedral and defined by its eight corners. These corner-points are specified as four pairs of depth coordinates defined along four vertical or inclined coordinate lines, see Figure 3.3, that emanate from a quadrilateral in the lateral direction. Together, the four coordinate lines define a pillar of cells, and this type of grid is thus also often referred to as a *pillar grid*. The grid format has a logical Cartesian numbering, but the corner-points of neighboring cells need not coincide, which in essence means that the resulting grids can have an unstructured topology [19]. This type of unstructured grid is highly flexible, but maintains a simple ijk-indexing, and is well-suited to adapt to the geological features of a reservoir, including, e.g., layers and faults.

3.3 Two-point flux approximation

The spatial discretization of the flow equations (2.5.1) is based on a simple finite-volume method called the two-point flux approximation (TPFA). Figure 3.4 illustrates some of the main ingredients of the method. For a simple demonstration, consider the simplified single-phase flow case for a single incompressible fluid (a fluid with constant density), in which case the conservation equation

$$\frac{\partial(\phi\rho)}{\partial t} + \nabla \cdot (\rho\vec{v}) = \rho q. \tag{3.3.1}$$

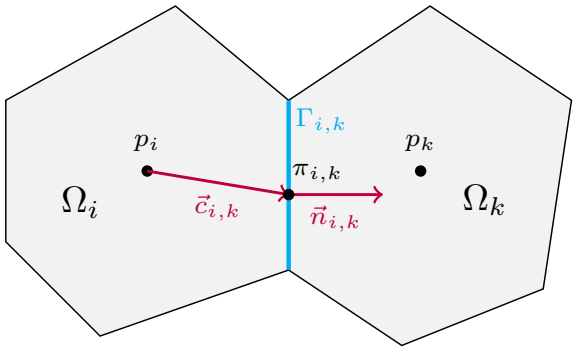


Figure 3.4: The two cells and related quantities used in the two-point flux approximation of the Laplace operator. Adapted from Figure 4.10 in [19].

simplifies to $\nabla \cdot \vec{v} = q$, as the accumulation term vanishes because ϕ and ρ are independent of time, and we eliminate ρ. The resulting model,

$$\begin{aligned} \nabla \cdot \vec{v} &= q \\ \vec{v} &= -K \nabla p, \end{aligned} \tag{3.3.2}$$

should originally hold pointwise in Ω. Let us now impose it on a single cell instead.

When integrating the conservation law $\nabla \cdot \vec{v} = q$ over a single cell Ω_i, and then applying the divergence theorem, we obtain

$$\int_{\partial\Omega_i} \vec{v} \cdot \vec{n} \, ds = \int_{\Omega_i} q \, d\vec{x}, \tag{3.3.3}$$

which ensures that the mass is conserved for that cell. The surface integral is naturally decomposed into a sum of integrals over the faces that bound the cell. In most reservoir models, the flow across the exterior boundary is assumed to be zero, so we can focus on interior cell faces only.

Now, let $\Gamma_{i,k} = \partial\Omega_i \cap \partial\Omega_k$ denote a *half-face* (Figure 3.4); that is, a face associated with grid cell Ω_i, for which the normal vector $\vec{n}_{i,k}$ is pointing *from* Ω_i *to* Ω_k. Assuming a matching grid, all interior half-faces have an opposite half-face with the same area, $A_{k,i} = A_{i,k}$, and opposite normal vector, $\vec{n}_{k,i} = -\vec{n}_{i,k}$. We can define the flux across a half-face as

$$v_{i,k} = \int_{\Gamma_{i,k}} \vec{v} \cdot \vec{n}_{i,k} \, ds, \tag{3.3.4}$$

and approximate this integral using the midpoint rule, obtaining

$$v_{i,k} \approx A_{i,k} \vec{v}(\vec{x}_{i,k}) \cdot \vec{n}_{i,k}, \tag{3.3.5}$$

where $\vec{x}_{i,k}$ denotes the centroid of $\Gamma_{i,k}$. Applying Darcy's law, we get the following expression for the flux:

$$v_{i,k} \approx -A_{i,k} \, (K \nabla p)(\vec{x}_{i,k}) \cdot \vec{n}_{i,k}. \tag{3.3.6}$$

The next step is to approximate the pressure gradient with a one-sided finite difference. Consider again the cells Ω_i and Ω_k and let $\pi_{i,k}$ be the pressure at the face centroid $\vec{x}_{i,k}$, and p_i the average pressure inside the cell Ω_i. Since we need the pressure at a certain point in Ω_i, we make an additional assumption that the pressure is linear within each cell. Then the pressure at the cell center equals the average. Letting $\vec{c}_{i,k}$ denote the vector from this cell centroid to the face centroid $\vec{x}_{i,k}$ we get the approximation

$$v_{i,k} \approx A_{i,k} K_i \frac{(p_i - \pi_{i,k}) \vec{c}_{i,k}}{|\vec{c}_{i,k}|^2} \cdot \vec{n}_{i,k} = T_{i,k}(p_i - \pi_{i,k}), \tag{3.3.7}$$

where $T_{i,k}$ denotes the one-sided transmissibility or half-transmissibility associated with $\Gamma_{i,k}$.

Finally, if we require continuity of fluxes across all faces, $v_{i,k} = -v_{k,i} =: v_{ik}$, as well as continuity of face pressures, $\pi_{i,k} = \pi_{k,i} =: \pi_{ik}$, we get the two-point approximation scheme

$$v_{ik} = T_{ik}(p_i - p_k), \tag{3.3.8}$$

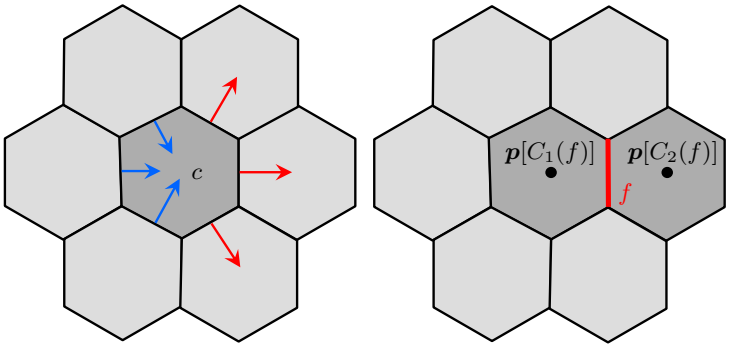


Figure 3.5: Discrete divergence (left) and gradient (right) operators. In this case, the discrete divergence in cell c is calculated by subtracting the influxes (blue) from the outfluxes (red). The discrete gradient operator on face f subtracts the value in its left cell from that in its right cell.

where we define the transmissibility on the interface as $T_{ik} = [T_{i,k}^{-1} + T_{k,i}^{-1}]^{-1}$. As indicated by the name, the TPFA scheme approximates the flux across the interface using the average pressure inside each of the two cells.

Considering again the full domain Ω and summing the contributions from all faces that bound each cell, we arrive at the full TPFA discretization of the simplified single-phase flow equation (3.3.2),

$$\sum_k T_{ik}(p_i - p_k) = q_i, \quad \forall \Omega_i \in \Omega. \tag{3.3.9}$$

We remark that the TPFA method is generally not consistent, since the transverse flux can not be approximated by the pressure difference $p_i - p_k$. It is, however, convergent for K-orthogonal grids, where $(\boldsymbol{K}\vec{n}_{ik}) \parallel \vec{c}_{ik}$ for all cells [19].

3.4 Discrete operators

By introducing discrete divergence and gradient operators, we can easily convert continuous equations like (3.3.2) to their discrete counterpart. These operators not only simplify notation when describing discrete equations but are also simple to realize in software and lead to compact code that can be made to look very similar to the corresponding mathematical formulas. We denote these new operators by `div` and `grad`, respectively, and refer to Figure 3.5 for an illustration.

The discrete divergence operator is a mapping from faces to cells, which can be applied to some discrete flux $\boldsymbol{v} \in \mathbb{R}^{n_f}$ [19]. Let $\boldsymbol{v}[f]$ denote its restriction onto the face f. For a matching grid, each interior face will have a cell on each side, which we call $C_1(f)$ and $C_2(f)$. If we now assume that the flux across a face f is always pointing from $C_1(f)$ to $C_2(f)$, we can subtract the influxes from neighboring cells from the outfluxes from the cell itself to get the total amount of matter leaving a cell c, the *discrete divergence*,

$$\texttt{div}(\boldsymbol{v})[c] = \sum_{f \in F(c)} \boldsymbol{v}[f]\mathbf{1}_{c=C_1(f)} - \sum_{f \in F(c)} \boldsymbol{v}[f]\mathbf{1}_{c=C_2(f)}, \tag{3.4.1}$$

where $F(c)$ is the set of all faces that bound cell c and $\mathbf{1}$ is the indicator function that equals 1 if the subscripted condition holds and 0 otherwise.

The discrete gradient operator maps cell pairs to faces. It can be applied to any cell quantity $\boldsymbol{p} \in \mathbb{R}^{n_c}$ and is defined as

$$\texttt{grad}(\boldsymbol{p})[f] = \boldsymbol{p}[C_2(f)] - \boldsymbol{p}[C_1(f)]. \tag{3.4.2}$$

To demonstrate the usefulness of these newly defined operators, we can immediately translate (3.3.2) to the discrete case

$$\texttt{div}(\boldsymbol{v}) = q, \qquad \boldsymbol{v} = -T\texttt{grad}(\boldsymbol{p}), \tag{3.4.3}$$

where T is a vector that holds the intercell transmissibilities defined in (3.3.8).

The discrete differential operators are naturally defined as sparse matrices. In fact, the `div` matrix is the negative transpose of the `grad` matrix, in analogy to the continuous case where the gradient is the adjoint operator of the divergence. The matrix representing the `grad` operator (3.4.2), say D, is easy to compute given the adjacency map describing the topology of a structured or unstructured grid. Here, D is an $n_f \times n_c$ matrix, and the sparse matrix representation of `div` follows from a sign flip and transpose [19]. The operators are then succinctly defined as

$$\texttt{grad}(\boldsymbol{p}) = D\boldsymbol{p}, \qquad \texttt{div}(\boldsymbol{v}) = -D^T\boldsymbol{v}. \tag{3.4.4}$$

Similar to (3.4.3), we can apply the discrete operators to the multiphase flow equations (2.5.1), and get the fully implicit system

$$\frac{1}{\Delta t}\left[(\boldsymbol{\Phi}\boldsymbol{S}_\alpha\boldsymbol{\rho}_\alpha)^{n+1} - (\boldsymbol{\Phi}\boldsymbol{S}_\alpha\boldsymbol{\rho}_\alpha)^n\right] + \texttt{div}(\boldsymbol{\rho}_\alpha\boldsymbol{v}_\alpha)^{n+1} = \boldsymbol{q}_\alpha^{n+1} \tag{3.4.5}$$

and

$$\boldsymbol{v}_\alpha^{n+1} = -\boldsymbol{\lambda}_\alpha\boldsymbol{T}[\texttt{grad}(\boldsymbol{p}_\alpha^{n+1}) - g\boldsymbol{\rho}_\alpha^{n+1}\texttt{grad}(\boldsymbol{z})], \tag{3.4.6}$$

where we have used a backward temporal discretization. The superscripts indicate discrete time steps, and Δt is the associated step length. Moreover, $\boldsymbol{\Phi}$ is the vector of pore volumes, $\boldsymbol{\rho}_\alpha$ is the vector of phase densities, and $\boldsymbol{S}_\alpha$ contains cell-averaged saturations. Finally, the vector $\boldsymbol{p}_\alpha$ holds the cell-averaged pressures, and $\boldsymbol{v}_\alpha$ holds fluxes for phase α for each face. In Eq. (3.4.6), the phase mobilities $\boldsymbol{\lambda}_\alpha$ and densities $\boldsymbol{\rho}$ are evaluated at the faces, but they are given in terms of quantities that are only available as cell averages. To define density on a face $\Gamma_{i,j}$, we can simply use an arithmetic mean,

$$\rho_{\alpha,ij} = \frac{1}{2}(\rho_{\alpha,i} + \rho_{\alpha,j}). \tag{3.4.7}$$

Furthermore, to define the phase mobilities, we use the single-point upstream scheme,

$$\lambda_{\alpha,ij} = \begin{cases} \lambda_{\alpha,i} & \text{if } \texttt{grad}(\boldsymbol{p}_\alpha)[\Gamma_{ij}] - g\rho_{\alpha,ij}\texttt{grad}(\boldsymbol{z})[\Gamma_{ij}] \leq 0, \\ \lambda_{\alpha,j} & \text{otherwise.} \end{cases} \tag{3.4.8}$$

3.5 Newton's method

After applying the discrete operators, we end up with a system of highly nonlinear equations, which can be reformulated in residual vector form as $\boldsymbol{F}_n(\boldsymbol{x}^{n+1}; \boldsymbol{x}^n) = \boldsymbol{0}$ for each time step n, where $\boldsymbol{x}^n$ denotes the known state of the primary variables at the start of the time step and $\boldsymbol{x}^{n+1}$ is the unknown state at the end. For simplicity, we drop the superscript and write this as $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$. This residual equation can be solved iteratively using Newton's method. Starting from some initial guess $\boldsymbol{x}^0$ for $\boldsymbol{x}$, we repeat for $k = 0, 1, \ldots$

$$\begin{aligned} \text{solve } & J_{\boldsymbol{F}}(\boldsymbol{x}^k)\boldsymbol{\delta}\boldsymbol{x}^k = -\boldsymbol{F}(\boldsymbol{x}^k) \\ \text{set } & \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha\boldsymbol{\delta}\boldsymbol{x}^k \end{aligned}$$

until the residual is sufficiently small [24]. Here, $0 < \alpha \leq 1$ is a dampening effect that is often introduced in practice to reduce the computed increment to improve convergence. The algorithm requires obtaining the Jacobian matrix $J_{\boldsymbol{F}}$ of $\boldsymbol{F}$ in each iteration. The Jacobian contains the derivatives of $\boldsymbol{F}$ with respect to all primary variables,

$$J_{\boldsymbol{F}}(\boldsymbol{x}) = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}, \qquad \big(J_{\boldsymbol{F}}(\boldsymbol{x})\big)_{ij} = \frac{\partial F_i}{\partial x_j}(\boldsymbol{x}). \tag{3.5.1}$$

One can prove that Newton's method is quadratically convergent for a sufficiently accurate initial guess and under certain smoothness assumptions on $\boldsymbol{F}$. In practice, obtaining such convergence heavily depends on the availability of a sufficiently accurate Jacobian. Deriving the Jacobian analytically and coding it is highly error-prone and time-consuming [19]. Using *automatic differentiation*, we can avoid this.

Automatic differentiation relies on the fact that computer code, when broken down, consists of basic operations like addition or multiplication, or function evaluations like sines or exponentials. The derivative of a series of these operations with respect to a given input variable follows from simply applying the chain rule together with well-known differentiation rules for each operation, evaluated at the specific value of the input variable. Automatic differentiation has become a cornerstone in mathematical software and scientific computing, and is a crucial part also in the MATLAB Reservoir Simulation Toolbox, which we use in this thesis.

3.6 The MATLAB Reservoir Simulation Toolbox (MRST)

After presenting some of the essentials of reservoir simulation, it is evident that the full implementation of an advanced simulator is extensive and far beyond the scope of this project. The upcoming simulation studies will therefore rely heavily on existing software, and extending its functionality to our specific needs. The software in use is called the MATLAB Reservoir Simulation Toolbox (MRST). This is a free and open-source software for reservoir modeling and simulation, primarily developed by the Computational Geosciences research group at SINTEF Digital [22]. It is built with rapid prototyping in mind, aiming to substantially reduce the time from a novel idea to a working demonstration [19, 25, 26, 27].

Herein, we will use the object-oriented, automatic-differentiation-based simulator framework (AD-OO) that has been built for rapid prototyping of fully differentiable simulators for various complex fluid models, including the black-oil model discussed earlier in the thesis. A simulation setup within the AD-OO framework has three main components: A model, a schedule and an initial state. The model includes a geological model consisting of a grid and a set of cell-wise properties that together describe the rock geometry and its petrophysical properties; a fluid model defining the present phases and their properties; as well as a system of algebraic nonlinear equations that describe the discretized form of the pertinent flow equations. The schedule holds the well model, as well as possible sources and boundary conditions, for each time step. Finally, the initial state stores the values of the state variables, such as pressure, saturation and fluxes, that together describe the distribution and the possible instantaneous movement of all fluids at the start of the simulation.

MRST comes with a rich set of modules, many of which are used in this project. Table 3.1 gives a short summary of the modules in use, what they hold, and what we have used them for. Note that the used functions column is not exhaustive, as it only includes functions and classes that are used explicitly. The call stack will typically include many layers of functions, all the way down to the AD modules. The MRST-based implementation of the graph-based reservoir models is further discussed in Chapter 5.

A key feature of MRST is the object-oriented framework for automatic differentiation, referred to as the AD-OO framework [19, 28]. The implementation is based on the ADI class, which keeps track of a variable and its derivative simultaneously. Whenever an operation is performed on the variable, the corresponding differentiation rule is applied to its derivative through operator overloading. For example, consider the ADI pair $\langle f, f' \rangle$, where f is the variable and f' its derivative. Some examples of operator overloadings include

$$\begin{aligned} \langle f, f' \rangle + \langle g, g' \rangle &= \langle f + g, f' + g' \rangle \\ \langle f, f' \rangle * \langle g, g' \rangle &= \langle fg, fg' + f'g \rangle \\ \sin\langle f, f' \rangle &= \langle \sin(f), \cos(f)f' \rangle. \end{aligned}$$

If the primary variables of the reservoir models are defined as ADI variables, the calculation of the residual equations automatically calculates the Jacobian as well.

Table 3.1: Relevant modules and used functions of MRST.

Module	Description	Used functions
`ad-core`	Object-oriented framework (AD-OO) for solvers based on automatic differentiation.	
`ad-blackoil`	Extends `ad-core` with, e.g., black-oil equations and single-, two- and three-phase solvers.	
`ad-props`	Functionality for property calculations for the `ad-core` framework.	

`coarsegrid`	Functionality for defining coarse grids based on a partition of an underlying fine grid.	`partitionUI`
		`processPartition`
		`compressPartition`

`deckformat`	Support for reading ECLIPSE decks. We use it to read and set up industry-standard reservoir models.	`readEclipseDeck`
		`convertDeckUnits`
		`initEclipseRock`
		`compressRock`

`diagnostics`	Flow diagnostics (i.e., methods for understanding the fluid communication within the reservoir), from which we use the time-of-flight computation.	`computeTimeOfFlight`
`ensemble`	Ensemble simulation.	
`network-models`	Experimental module including CGNet and GPSNet implementation. We use a utility function for perturbing controls.	`makeRandomTraining`

`optimization`	Functionality for solving optimal control problems through forward and adjoint simulations with automatic differentiation.	`OptimizationProblem`
		`ModelParameter`
		`unitBoxLM`
		`matchObservedOW`
		`NPVOW`

`test-suite`	Framework for setting up test cases for the AD-OO framework.	`TestCase`
		`egg_wo`
		`saigup_wo`
		`norne_simple_wo`

`upr`	Tools for creating Voronoi grids adapting to, e.g., wellpaths and faults. We use its modified implementation of DistMesh and generator of clipped Voronoi grids.	`distmesh2d` `clippedPebi2D`

`upscaling`	Methods for flow-based upscaling of, e.g., permeabilities and transmissibilities.	`upscaleModelTPFA`
		`upscaleState`
		`upscaleSchedule`

Chapter 4

Model Calibration

A reservoir model as described in the previous chapters has a number of physical parameters, such as transmissibilities $\boldsymbol{T}$, pore volumes $\boldsymbol{\Phi}$ and well indices $\boldsymbol{J}$. In practice, it is difficult to determine correct values for these parameters a priori, but fortunately, their values can be tuned. In particular, we can adjust the parameters until the model output matches sufficiently well with observed behaviour.

The traditional approach herein is history matching, an instance of an inverse problem using observations to assign values to the model parameters [29]. As opposed to purely data-driven, machine learning-type models, there is now an underlying mathematical model including physical parameters. This entails that the parameter tuning should not only give accurate predictions, but also physically meaningful parameters. Moreover, an equally important goal of history matching is improved reservoir characterization. Here, the matching seeks to deviate as little as possible from the a priori geological model. In particular, ensemble-based history matching, which is common nowadays, fits multiple models to sample a probability field imposed on the model, e.g., for the distribution of petrophysical parameters.

In this project, we part from the traditional history matching and leave behind its physical constraints. We no longer view the tunable parameters as physical, but instead treat them as mere algebraic coefficients that can be calibrated freely until the model produces the desired predictions. While history matching generally seeks universally valid models, our aim is limited to predicting states close to the training data. Mathematically, the goal is to minimize an objective function measuring the mismatch in the model output compared to the observed data; see Figure 4.1. We will now succinctly define the optimization problem and suggest an iterative algorithm for its solution.

4.1 Formulating the optimization problem*

Let $\boldsymbol{x}^n$ denote the state vector at time step n, typically containing all pressures, saturations and well rates/bottom-hole pressures. Moreover, let the vector $\boldsymbol{\theta} \in \mathbb{R}^{N_\theta}$ hold the model parameters, which we assume to be constant over time. As seen in Section 3.5, we then solve the system $\boldsymbol{F}_n(\boldsymbol{x}^{n+1}, \boldsymbol{x}^n; \boldsymbol{\theta}) = 0$ at each time step n. Having N time steps in total, we get a set of states $\{\boldsymbol{x}^1, \dots, \boldsymbol{x}^N\}$.

Assume now that we have some set of observations $\boldsymbol{y} \in \mathbb{R}^{N_y}$, with corresponding model predictions $\hat{\boldsymbol{y}} = \hat{\boldsymbol{y}}(\boldsymbol{x}^1, \dots, \boldsymbol{x}^N)$. For our purpose, the observations will be the output of a fine-scale simulation, but they could also be real measurements. We define the weighted residuals r_j as

$$r_j = w_j(\hat{y}_j - y_j).$$

Here, w_j is a weight chosen to normalize the residuals (so that $r_j \sim 1$), by for example using the reciprocal of a typical value magnitude. Note that the residual is also referred to as the misfit or

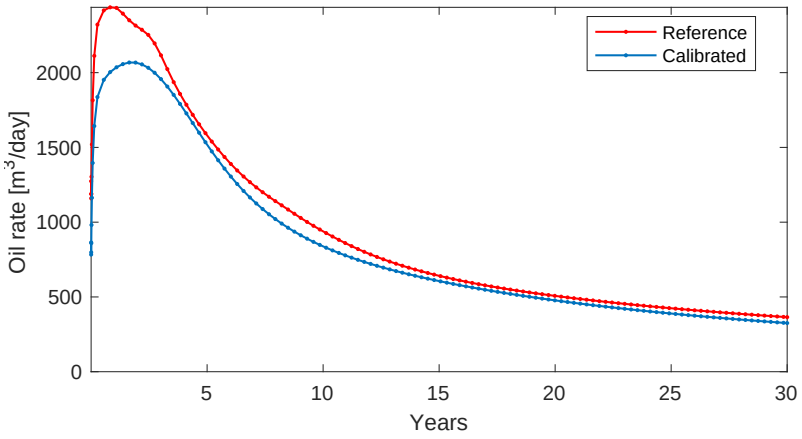


Figure 4.1: A typical well production curve. The red curve represents the true/measured oil rate, while the blue curve is the coarse-model prediction. The objective function (4.1.1) sums up the squared difference over all time steps for all wells.

mismatch and should not be confused with the residual in the discretized flow equations. We can assemble the N_y residual terms into the residual vector $\boldsymbol{r}$,

$$\boldsymbol{r} = (r_1, r_2, \ldots, r_{N_y})^T,$$

and formulate our objective function,

$$M = \frac{1}{2}\sum_{j=1}^{N_y} r_j^2 = \frac{1}{2}\boldsymbol{r}^T\boldsymbol{r}, \tag{4.1.1}$$

which we aim to minimize. The misfit minimization is thus an instance of a nonlinear least-squares problem, for which there exist a number of algorithms. Here, we will consider the Levenberg–Marquardt algorithm.

4.2 The Levenberg–Marquardt algorithm*

The Levenberg–Marquardt algorithm is an iterative method for solving nonlinear least-squares problems [30], aiming to minimize an objective function on the form (4.1.1). The algorithm makes use of the misfit Jacobian $J_{\boldsymbol{r}}$, a matrix of dimension $N_\theta \times N_y$ containing the derivatives of all residuals with respect to all parameters,

$$J_{\boldsymbol{r}} = [\nabla_\theta r_1, \nabla_\theta r_2, \ldots, \nabla_\theta r_{N_y}]. \tag{4.2.1}$$

Using the residual vector and the Jacobian matrix, the gradient and Hessian of the objective function M can be succinctly written as

$$\nabla_\theta M = J_{\boldsymbol{r}}\boldsymbol{r}, \tag{4.2.2}$$

$$\nabla_\theta^2 M = J_{\boldsymbol{r}} J_{\boldsymbol{r}}^T + \sum_{j=1}^{N_y} r_j \nabla_\theta^2 r_j. \tag{4.2.3}$$

Several observations can be made from these expressions. First, the residual gradients are usually easily calculated, making the Jacobian matrix $J_{\boldsymbol{r}}$ itself easily available. Having the Jacobian, we can obtain the gradient $\nabla_\theta M$ in (4.2.2) and the first term of the Hessian in (4.2.3) relatively easily and inexpensively. Algorithms for nonlinear least-squares problems often exploit this, as well as the fact that the first term in the Hessian (4.2.3) is typically more important than the second. In fact, the second term vanishes when the residuals vanish, so algorithms excluding this term often perform well in cases with sufficiently small residuals.

We aim to identify an appropriate parameter update that decreases the objective value. In a line-search approach, a descent direction $\boldsymbol{p_k}$ is chosen, and the distance α_k to move along that direction decided by solving $\min_{\alpha_k} M(\boldsymbol{\theta_k} + \alpha_k \boldsymbol{p_k})$. Here, k is the iteration number in the iterative optimization algorithm. A common choice of descent direction is the Newton direction $\boldsymbol{p}^N$, which comes from solving

$$\nabla_\theta^2 M \, \boldsymbol{p}^N = -\nabla_\theta M = -J_{\boldsymbol{r}} \boldsymbol{r}, \tag{4.2.4}$$

where we have dropped the subscript k for simplicity. The Newton direction can be derived from the second-order Taylor series expansion of the objective function,

$$M(\boldsymbol{\theta}_k + \boldsymbol{p}_k) \approx M_k + \boldsymbol{p}_k^T \nabla_\theta M_k + \tfrac{1}{2} \boldsymbol{p}_k^T \nabla_\theta^2 M_k \boldsymbol{p}_k =: m_k(\boldsymbol{p}_k). \tag{4.2.5}$$

Under the assumption that the Hessian $\nabla_\theta^2 M_k$ is positive definite, we minimize $m_k(\boldsymbol{p_k})$ with respect to $\boldsymbol{p_k}$ by setting its derivative to zero, and obtain the Newton direction

$$\boldsymbol{p}_k^N = -(\nabla_\theta^2 M_k)^{-1} \nabla_\theta M_k. \tag{4.2.6}$$

A drawback of the Newton direction is the need for the full Hessian $\nabla_\theta^2 M$. Quasi-Newton methods avoid this by using an approximation of the Hessian instead. In particular, Gauss–Newton methods use the approximation $\nabla_\theta^2 M \approx J_{\boldsymbol{r}} J_{\boldsymbol{r}}^T$, solving

$$J_{\boldsymbol{r}} J_{\boldsymbol{r}}^T \boldsymbol{p}^{GN} = -J_{\boldsymbol{r}} \boldsymbol{r} \tag{4.2.7}$$

instead of the original Newton equations (4.2.4) in each iteration. In this project, we do not use a line-search approach like Quasi-Newton methods, but instead a trust-region method based on the same Hessian approximation; in particular, the Levenberg–Marquardt algorithm. Unlike a line-search method that first finds a direction and then a step length, a trust-region method first decides a maximum step length and then finds the best direction inside the allowed region. Assuming a spherical trust region, we find a descent direction inside the region at each iteration, specifically a direction $\boldsymbol{p}$ solving

$$\min_{\boldsymbol{p}} \frac{1}{2} \left\| J_{\boldsymbol{r}} \boldsymbol{r} \boldsymbol{p} + \boldsymbol{r} \right\|^2, \qquad \text{subject to } \|\boldsymbol{p}\| \leq \Delta, \tag{4.2.8}$$

where Δ denotes the radius of the trust region. If the direction $\boldsymbol{p}^{GN}$ from (4.2.7) is inside the trust region, satisfying $\left\|\boldsymbol{p}^{GN}\right\| \leq \Delta$, then it also solves (4.2.8). Otherwise, the solution to (4.2.8) is on the trust-region boundary, $\|\boldsymbol{p}\| = \Delta$, and there is some damping parameter $\alpha > 0$ such that it satisfies

$$(J_{\boldsymbol{r}} J_{\boldsymbol{r}}^T + \alpha I)\boldsymbol{p} = -J_{\boldsymbol{r}} \boldsymbol{r} = -\nabla_\theta M. \tag{4.2.9}$$

Observe that if $\alpha = 0$, then this gives us the Gauss-Newton direction $\boldsymbol{p}^{GN}$. If $\alpha \to \infty$, then $\boldsymbol{p}$ tends towards the steepest descent direction.

Algorithm 1 summarizes the Levenberg–Marquardt method.

Algorithm 1 The Levenberg–Marquardt algorithm

```
while ∇_θ M ≥ ε do
    Solve (J_r J_r^T + α_k I) p_k = −J_r r
    if M(θ_k + p_k) < M(θ_k) then          ▷ Accept step and shrink trust region
        α_{k+1} = α_k / α_dec
        θ_{k+1} = θ_k + p_k
        k = k + 1
    else                                   ▷ Reject step and increase trust region
        α_{k+1} = α_k · α_inc
        θ_{k+1} = θ_k
        k = k + 1
    end if
end while
```

The damping parameter α is updated between iterations following a quasi-trust-region logic, where we increase or decrease the damping parameter instead of directly increasing or shrinking the trust

region radius Δ itself. In particular, we decrease α by a factor α_{dec} if a step is accepted, or increase it by a factor α_{inc} if the step is rejected. In the MRST implementation of Levenberg–Marquardt (`unitBoxLM` from the `optimization` module), the default values are $\alpha_{dec} = 5$ and $\alpha_{inc} = 8$. The damping parameter prevents singular systems, thus avoiding the problematic behaviour of Gauss-Newton methods when the Jacobian is rank-deficient [24].

4.3 Calculating the Jacobian from an adjoint simulation

The Levenberg–Marquardt algorithm requires the full mismatch Jacobian matrix $J_{\boldsymbol{r}}$. This can either be computed by N_y forward simulations, or more effectively through N_θ adjoint simulations [31, 32].

To see this, first observe that the gradient of the objective function, $\nabla_\theta M$, can be obtained by running a standard adjoint simulation,

$$\nabla_\theta M = \sum_{n=1}^{N} \frac{\partial \boldsymbol{F}_n}{\partial \boldsymbol{\theta}^n}^T \boldsymbol{\lambda}^n. \tag{4.3.1}$$

Here, $\boldsymbol{\lambda}^n$ denote the Lagrange multipliers, which can be obtained by solving the linear adjoint equations

$$\frac{\partial \boldsymbol{F}_n}{\partial \boldsymbol{x}^n}^T \boldsymbol{\lambda}^n = -\frac{\partial M}{\partial \boldsymbol{x}^n}^T - \frac{\partial \boldsymbol{F}_{n+1}}{\partial \boldsymbol{x}^n}^T \boldsymbol{\lambda}^{n+1}, \qquad n = N, N-1, \ldots, 1. \tag{4.3.2}$$

Each column in the mismatch Jacobian matrix $J_{\boldsymbol{r}}$ corresponds to the gradient of a single residual r_j with respect to all parameters $\boldsymbol{\theta}$, and we can calculate one such gradient $\nabla_\theta r_j$ through an adjoint equation. Replacing M with r_j in (4.3.1)-(4.3.2), we get

$$\nabla_\theta r_j = \sum_{n=1}^{N} \frac{\partial \boldsymbol{F}_n}{\partial \boldsymbol{\theta}}^T \boldsymbol{\lambda}^n, \tag{4.3.3}$$

with the Lagrange multipliers now given by

$$\frac{\partial \boldsymbol{F}_n}{\partial \boldsymbol{x}^n}^T \boldsymbol{\lambda}^n = -\frac{\partial r_j}{\partial \boldsymbol{x}^n}^T - \frac{\partial \boldsymbol{F}_{n+1}}{\partial \boldsymbol{x}^n}^T \boldsymbol{\lambda}^{n+1}, \qquad n = N, N-1, \ldots, 1. \tag{4.3.4}$$

In (4.3.3), the left and right hand sides are $N_\theta \times 1$ column vectors, and we have N_y such equations. Instead of solving one adjoint equation for each data point, we can stack them and solve one system with multiple right hand sides,

$$J_{\boldsymbol{r}} = \sum_{n=1}^{N} \frac{\partial \boldsymbol{F}_n}{\partial \boldsymbol{\theta}}^T \boldsymbol{\Lambda}^n \tag{4.3.5}$$

Now, $\boldsymbol{\Lambda}^n$ is a matrix, where each column is the vector of Lagrange multipliers $\boldsymbol{\lambda}^n$ for one data point from (4.3.3). These can be obtained from the matrix adjoint equations

$$\frac{\partial \boldsymbol{F}_n}{\partial \boldsymbol{x}^n}^T \boldsymbol{\Lambda}^n = -\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}^n}^T - \frac{\partial \boldsymbol{F}_{n+1}}{\partial \boldsymbol{x}^n}^T \boldsymbol{\Lambda}^{n+1}, \quad n = N, N-1, \ldots, 1. \tag{4.3.6}$$

4.4 Parameter limits and scaling

In its original form, the Levenberg–Marquardt algorithm does not impose any bounds on the parameter values. Keeping in mind that one of the motivations behind our hybrid methods was to give physically consistent results, it does however seem reasonable to set some, rather loose, limits even if we do not intend to try to interpret the calibrated parameters as physically representative.

We could set some predetermined lower and upper box limits, and require that the parameters remain within the interval. This is natural for parameters like saturations, which should always be

between 0 and 1. For other parameter types, it is better to set non-uniform relative limits, tailored to each single parameter on the cell/face level. Specifically, given a parameter θ with initial value θ_0, we use a lower and an upper bound $[l_{min}, l_{max}]$ and require that $l_{min}\theta_0 \leq \theta \leq l_{max}\theta_0$. As a default, we use [0.01, 10] for pore volumes $\boldsymbol{\Phi}$ and [0.01, 100] for transmissibilities $\boldsymbol{T}$ and well indices $\boldsymbol{J}$. This implies, e.g., that given a cell with initial pore volume 1, the pore volume has to remain within the $[0.01, 10]$ interval during calibration. A complete overview of the parameter limits used in the upcoming simulations is given in the following chapter (Table 5.1).

Scaling is a widely used preconditioning technique for optimization algorithms, which alleviates ill-conditioning and thus improves numerical behaviour. In the MRST version of the Levenberg–Marquardt algorithm, we do indeed work with scaled parameters. In fact, all parameters are scaled using their limits, so that they remain within the unit interval $[0, 1]$. If the Levenberg–Marquardt algorithm suggests an illegal step, with parameter values outside the bounds, the step is simply projected onto the allowed interval; that is, if the suggested (relative) value is too large, the upper limit is used, and if it is too small, the lower limit is used.

Chapter 5

Graph-based Reservoir Simulation

We saw in Chapter 2 that the governing equations for flow in a reservoir arose from a combination of Darcy's law and conservation of mass for each phase. These equations were then discretized using a finite-volume method with a two-point flux approximation in Chapter 3, giving the final discrete system

$$\frac{\Phi}{\Delta t}\left[(\boldsymbol{S}_\alpha\boldsymbol{\rho}_\alpha)^{t+\Delta t} - (\boldsymbol{S}_\alpha\boldsymbol{\rho}_\alpha)^{t}\right] + \texttt{div}\left((\boldsymbol{\rho}_\alpha\boldsymbol{v}_\alpha)^{t+\Delta t}\right) = \boldsymbol{q}_\alpha \tag{5.0.1a}$$

$$\boldsymbol{v}_\alpha = -\boldsymbol{T}\boldsymbol{\lambda}_\alpha\texttt{grad}\left(\boldsymbol{p} - g\boldsymbol{\rho}_\alpha\boldsymbol{z}\right) \tag{5.0.1b}$$

$$\boldsymbol{q}_\alpha = \boldsymbol{\lambda}_\alpha^{wb}\boldsymbol{J}\left(\boldsymbol{p}^{wb} - \boldsymbol{p}\right). \tag{5.0.1c}$$

The key idea behind this thesis is to shift the perspective and recognize that a TPFA model like (5.0.1) can be interpreted as a computational graph in which nodes store fluids and edges transmit them. This interpretation naturally emerges from the underlying conservation laws and their discretization. On the discrete level, the conservation laws ensure that the accumulation of mass in a control cell equals the sum of its net influx and possible source terms. Here, the cell flux, the flow in and out of the cell, corresponds to the movement of fluids to or from its neighboring cells.

In a standard finite-volume grid, the neighbor relation has a natural physical interpretation, as the neighboring cells are next to each other in physical space. However, in reservoir simulation it is also common to include so-called non-neighboring connections to enable flow between grid cells that are not physically adjacent, e.g., to represent flow through conductive faults that have so small volumes that they are not naturally represented as volumetric cells in the grid. Several simulators therefore have a preprocessing phase in which the physical grid is turned into a computational graph, which is then used as the key data structure in the subsequent simulation. Somewhat simplified, this graph is formed by extracting nodes from the list of the cells' pore volumes and edges from the intercell transmissibilities, possibly enhanced by non-neighboring connections. The same approach is also common in mixed-dimensional models of fractured media; see for instance [33]. But why stop here: We could argue that the conservation of mass is not dependent on a "physical" grid, but may just as well be imposed on some flexible graph. Accepting this, we are left with one major question: how should the graph look?

Previous research includes different versions of interwell network models, using wells as their nodes and adding edges between them, see e.g., GPSNet [5], StellNet [6], FlowNet [34, 35], RGNet [36], and INSIM [7, 8, 9]. This gives a coarse model, and a limited number of pathways between wells. From a data-driven point of view, more parameters give more flexible models that may be easier to calibrate well. Also, from a physical perspective, fluids do not generally follow a single path from injector to producer. Perhaps the connection graph should be richer, allowing fluids to take multiple different paths. This hypothesis has previously been investigated in [12], where the CGNet model type was compared with the interwell model GPSNet. The models demonstrated similar predictive power, but the CGNet appeared more efficient and robust. The CGNet model type was

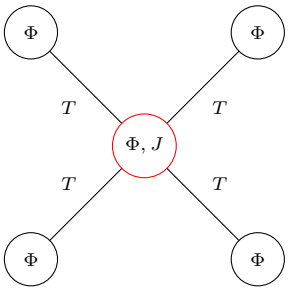


Figure 5.1: An example graph illustrating where the tunable parameters belong. Here, each node has an associated pore volume Φ, each edge a transmissibility T, and each well node (red) a well index J.

further studied in [11], also including automatic tuning of network granularity.

This chapter presents two main genres of graph-based models, both offering a richer set of connections and thus more tunable parameters. The first is a variant of the above mentioned CGNet, based on partitioning. We discuss how to construct such a model by starting from a fine-scale model and performing a partition, uniform or not, to give a coarsened model. The second provides a simple and well-defined way of constructing a model of desired granularity using little to no information about the reservoir and is based on a triangulation of the wells and selected points along the reservoir boundary. Similar ideas have been employed in, e.g., [36] and [37].

When tampering with the geometry, we will heavily depend on the subsequent model calibration. To that end, we can use the physical parameters of the discrete system (5.0.1), such as pore volumes $\boldsymbol{\Phi}$, transmissibilities $\boldsymbol{T}$ and well indices $\boldsymbol{J}$. When simulating on a graph, these all have their natural places, at the nodes, edges, and well nodes, respectively. This is illustrated in Figure 5.1 for an imaginary model with a single well. Each node has an associated pore volume, each edge a transmissibility, and each well node (corresponding to a perforation) a well index. When calibrating the graph-based models, we will no longer interpret the tunable parameters as physical, but rather allow them to be adjusted almost freely to give the desired input-output relation, as described in Chapter 4. An advantage of this approach is that since we still use our mathematical model, we can expect physically consistent predictions. Moreover, as long as we keep the graphs small, the models enable rapid evaluations and thereby inexpensive calibration. Finally, the models fit almost immediately into a standard reservoir simulator like MRST. The implementation is briefly explained at the end of this chapter.

We remark that the graph-based simulation framework described here could be applied to any finite-volume based simulation. In particular, the context where models are optimized based on well responses is directly transferable to geothermal energy and carbon storage applications, but this project is confined to hydrocarbon reservoirs.

5.1 Partition-based network models (CGNet)

Starting from a traditional fine-scale model, there are a number of ways to construct *reduced-order models*, coarsened models that are more suitable for applications requiring multiple simulation runs, such as optimization. If you want to generate a coarse grid instead of a fine one, you could of course just generate it in the same way as the original, only with a lower spatial resolution. This, however, has its disadvantages. When the original grid is geometrically complex, as is often the case for realistic reservoirs, it is challenging to preserve this geometry with a coarse grid. Moreover, you generally do not have a one-to-one mapping between cells in the fine and coarse grids [19]. Using a *partition*, we avoid these issues.

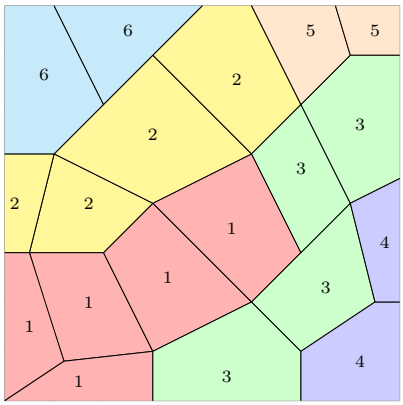


Figure 5.2: An example of a partition in 2D. The number in each cell c_i corresponds to that cell's entry in the partition vector, p_i. Adapted from figure in [19, Page 519].

5.1.1 Constructing the coarse graph

Coarse grids that are based on a direct partition of a fine grid can be represented by a *partition vector*. Say that the fine grid has n cells, and the coarse grid groups these cells into blocks B_l, $l = 1, 2, \ldots, N$. Then, the partition vector p has n elements, and each element p_i takes the value l if cell c_i belongs to block B_l. Thus, we can define our block B_l as the set of cells it contains,

$$B_l = \{c_i \mid p_i = l\}. \tag{5.1.1}$$

Figure 5.2 shows a simple example of a partition with six blocks, where the number in each cell corresponds to that cell's entry in the partition vector.

A partition can in principle take any form, only limited by the creator's imagination. Still, it seems reasonable to enforce a few rules. We will require that each cell in the fine grid belongs to exactly one block in the coarse grid. In other words, all blocks should be non-overlapping, and the total number of block cells should equal the number of cells n,

$$B_l \cap B_k = \varnothing, \quad l \neq k$$
$$\Big|\bigcup_l B_l\Big| = n.$$

Moreover, we require that all blocks consist of a connected subset of cells. That is, all cells in a block have to share a face with at least one other cell in that block,

$$\bigcup_{\substack{c_j \in B_l \\ c_j \neq c_i}} \overline{c_i} \cap \overline{c_j} \neq \varnothing \quad \forall\, c_i \in B_l.$$

With such partitions, it is straightforward to convert the coarse grid to a graph. Each coarse block is a node, and we add edges between coarse blocks that share one or more cell faces (or more generally, have an associated intercell transmissibility). For the example in Figure 5.2, this gives the graph $G = (V, E)$, where

$$V = \{1, 2, 3, 4, 5, 6\}$$
$$E = \{(1,2), (1,3), (2,3), (2,5), (2,6), (3,4), (3,5)\}.$$

Here, we have used the standard graph notation with G denoting the graph, V the nodes or vertices and E the edges (see Appendix A).

The most straightforward way to create a partition is to do it uniformly, by specifying some coarse rectilinear grid. This can be accomplished either in physical space or, for a corner-point grid, in index space. We use a sufficiently high resolution for no wells to end up within the same block.

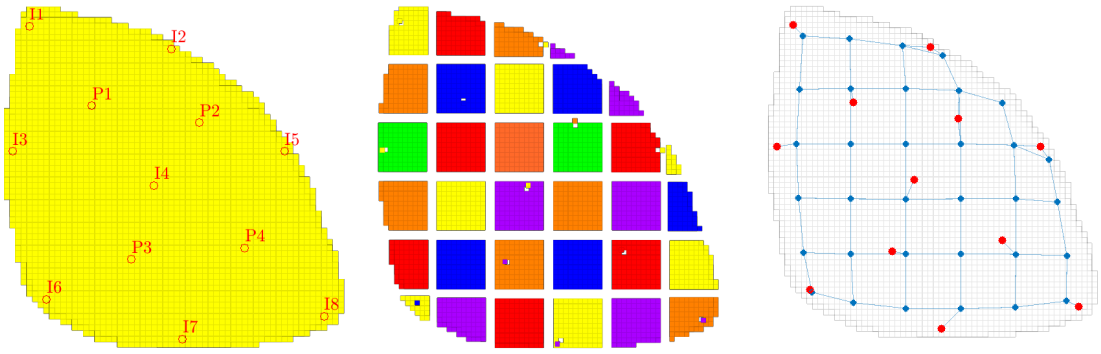


Figure 5.3: Construction of a partition-based network model for the Egg case. On the left is the fine grid and wells, in the middle an explosion view of the $6 \times 6 \times 1$ coarse partition after culling, and on the right the resulting CGNet. The wells are assigned to separate blocks and thus nodes (shown in red).

This can also be achieved by splitting blocks, at the expense of a slightly more complicated graph topology. Reservoir domains are often non-rectangular, so we use a fictitious domain approach, culling the cells that fall outside the reservoir. Figure 5.3 shows an example for the Egg model [38]. Here, we have used a $6 \times 6 \times 1$ coarse grid, and separate blocks/nodes for wells. The model is tested numerically in the next chapter.

5.1.2 Completing the model with upscaling

The partition itself gives only the coarse grid. Then, the model needs to be tailored to that grid. Cell- and face-wise properties, including parameters and initial states, need to be mapped over to the coarse grid with appropriate dimensions. When we know the original fine-scale model, this can be done through *upscaling*. In MRST, the necessary tools are implemented and ready to use in the `upscaling` module.

Since three-dimensional reservoir models can have millions of cells, upscaling has become a natural part in the reservoir modeling workflow. The upscaling process aims to convert the cell-wise properties of an original fine-scale model to blocks in a coarse grid through some homogenization. Properties like saturations and pore volumes are additive and can therefore be upscaled through a simple weighted arithmetic average, whereas non-additive properties like permeabilities are far less trivial. For more details on this, consult, e.g., [19]. We remark that it is of course no disadvantage to set accurate values for upscaled properties, but do keep in mind that parameters will be calibrated. A *reasonable* initial guess should be adequate.

5.1.3 Modifying the partition

A partition does not only allow a straightforward construction of a coarse grid. Given a coarse grid represented by a partition vector, the grid can easily by altered by tweaking the partition vector. If, for example, you want to merge two neighboring blocks k and l, you may simply replace all k values in p with l, or opposite. Then, the partition vector can be compressed to maintain a logical numbering. Similarly, if you want to split a block into its fine cells, you can simply assign unused block numbers to those cells. This flexibility is useful if you want to create non-trivial grids, e.g., adapted to the flow.

5.1.4 Flow-adapted models using residence times

In real reservoirs, oil may be concentrated in a cap, and wells distributed in such a way that the fluid flow is very limited in certain parts of the reservoir. An example of this is the Brugge benchmark model [39], for which Figure 5.4 demonstrates that the oil and wells are concentrated

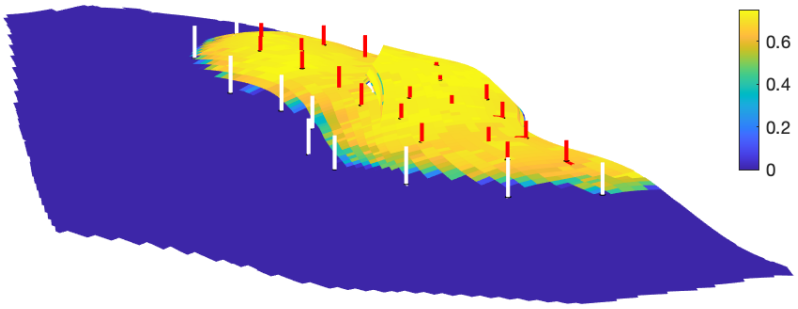


Figure 5.4: Initial oil saturation for the Brugge benchmark model. Injectors in white and producers in red.

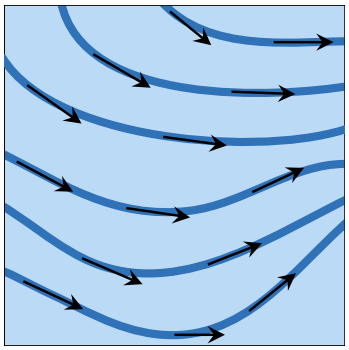


Figure 5.5: A set of streamlines, including the tangential velocity field vectors.

in a small part of the domain. In such cases, the network model will likely benefit if we adapt the grid partition to the flow. We want higher resolution in high-flow areas, and lower in the more uneventful region.

Can we achieve this a priori? That is, based on information from the fine model, can we construct a coarse grid that adapts to the flow before calibration? A simple solution is to just use the distance from any injector/producer to infer the block density. Another, more physically sophisticated, approach is to use a *flow indicator* to determine the local granularity [13]. Possible flow indicators include permeability, velocity and *residence time*, where we will focus on the latter. The flexibility in the partition-based representation of a coarse grid makes it relatively easy to define the non-uniform models. First, we need to calculate the indicator.

Time-of-flight and residence times

To understand time-of-flight, we first need to introduce the concept of streamlines. A streamline consists of a family of curves that are tangential to the velocity field $\vec{v}$ at some time t, as illustrated in Figure 5.5. Thus, the streamlines indicate where a fluid element would travel at that point in time [19]. We can parameterize a single streamline at time $\hat{t}$ by $\vec{x}(r)$, where

$$\frac{d\vec{x}}{dr} \times \vec{v}(\vec{x}, \hat{t}) = 0, \tag{5.1.2}$$

or equivalently,

$$\frac{d\vec{x}}{dr} = \frac{\vec{v}(\hat{t})}{|\vec{v}(\hat{t})|}. \tag{5.1.3}$$

It is common to parameterize a streamline using time-of-flight instead of the arc length r, as this accounts for the reduced volume available for flow through the porosity ϕ. The time-of-flight τ expresses the time it takes a particle to flow along a streamline for some distance r, and is defined as

$$\tau(r) = \int_0^r \frac{\phi(\vec{x}(s))}{|\vec{v}(\vec{x}(s))|}\, ds. \tag{5.1.4}$$

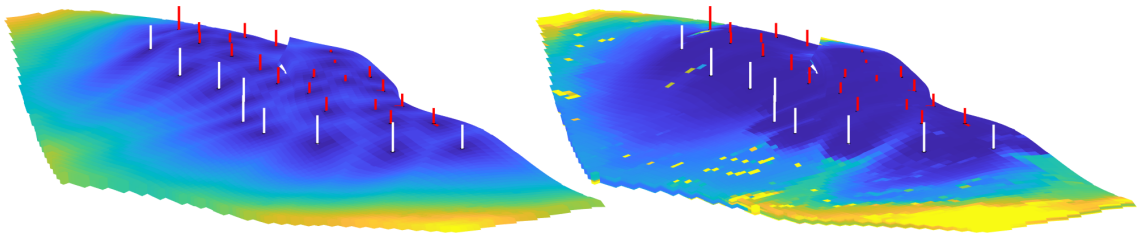


Figure 5.6: Distance from wells (left) and residence-time field (right) for the Brugge model. Injectors in white and producers in red. The two metrics are qualitatively similar.

We can compute the directional derivative of τ along a streamline, and apply the fundamental theorem of calculus to (5.1.4),

$$\frac{\vec{v}}{|\vec{v}|} \cdot \nabla\tau = \frac{d}{dr}\tau(r) = \frac{\phi}{\vec{v}}. \tag{5.1.5}$$

This yields the differential form,

$$\vec{v} \cdot \nabla\tau = \phi,$$

which we refer to as the time-of-flight equation [19].

In a reservoir setting, we define the forward time-of-flight as the time it takes a neutral particle to flow from the nearest fluid source (injector or inflow boundary) to each point in the reservoir,

$$\vec{v} \cdot \nabla\tau_f = \phi, \quad \tau|_{\text{inflow}} = 0. \tag{5.1.6}$$

We can also compute the backward time-of-flight, which is the time it takes a particle to flow from each point in space to the nearest fluid sink (producer or outflow boundary),

$$\vec{v} \cdot \nabla\tau_b = \phi, \quad \tau|_{\text{outflow}} = 0. \tag{5.1.7}$$

If we sum up the two, we get the *residence time*, $\tau_f + \tau_b$. This is the total time an imaginary particle spends in the reservoir as it travels from the nearest inflow point to the closest outflow point. This is an interesting quantity, as it says something about which parts of the reservoir are more and less eventful.

The Brugge benchmark model is a particularly illustrative example due to the concentration of oil and wells in a limited area. Figure 5.6 shows the distance to closest well and computed residence-time field. In the outer areas, outside the wells, the residence time is generally above 500 years, indicating that the flow is negligible there. The model, and the use of well distance and residence time to construct non-uniform network models will be further discussed and tested numerically in Chapter 6.

Construction of a non-uniform coarse graph

Having computed a flow or refinement indicator $I\colon V \to \mathbb{R}$, as just described, we can easily create an adapted non-uniform grid by tweaking the partition vector. Start with a coarse $n_x \times n_y \times n_z$ partition, with a partition vector p_c. We want to use the flow indicator to decide which blocks to refine. To that end, we need to map the flow indicator from the fine cells onto the coarse blocks. We will do this by simply summing up, so

$$I(B_l) = \sum_{c_i \in B_l} I(c_i), \tag{5.1.8}$$

where the notation $c_i \in B_l$ means that cell c_i belongs to block B_l, or equivalently, $p(i) = l$.

Construct a second, slightly finer and overlapping partition, such as $2n_x \times 2n_y \times n_z$. For example, you could use a coarse $10 \times 10 \times 1$ partition with partition vector p_c, and a finer $20 \times 20 \times 1$ partition with partition vector p_f. This corresponds to a 2×2 horizontal splitting of each block in the coarsest partition.

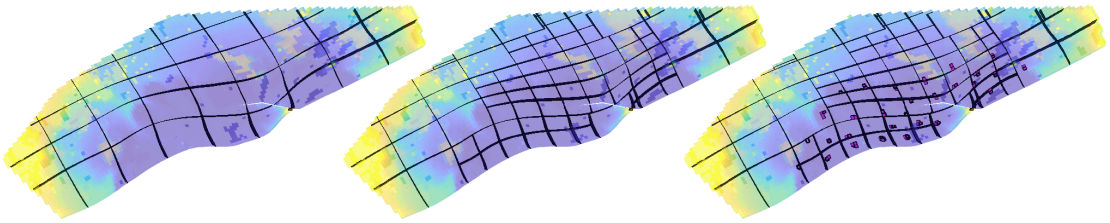


Figure 5.7: Construction of a non-uniform grid based on residence time for Brugge. The three plots show the outline of the coarse grid overlaid on the residence-time field, after each step in the process. 1) Coarse $9 \times 5 \times 1$ uniform partition. 2) Split blocks with residence time above median. 3) Add separate blocks for the wells.

Next, we need to define some criterion under which we select the blocks to split. This can be done by setting some tolerance, and selecting the blocks with indicator above that, giving a set of selected blocks

$$B^* = \{B_l \mid I(B_l) > tol\}. \tag{5.1.9}$$

A simple option is to split blocks with indicator above the median. Alternatively, this criterion can be adjusted by multiplying the median by some factor to obtain the desired granularity.

Finally, we split the selected blocks B^* by altering the partition vector. In practice, we can do this by first shifting the numbering of p_f to not conflict with p_c,

$$p_f = p_f + \max(p_c). \tag{5.1.10}$$

Then, starting with $p = p_c$, we modify the partition by

$$p(c_i) = p_f(c_i), \quad c_i \in B_l,\, B_l \in B^*, \tag{5.1.11}$$

followed by a compression to avoid unused block numbers.

An example for the Brugge model is illustrated in Figure 5.7. Here, we have started from a $9 \times 5 \times 1$ partition, and split all blocks with residence time above the median in four, via a $18 \times 10 \times 1$ partition. Finally, we added separate blocks for all wells.

5.2 Triangulation-based network models (TriNet)

A partition-based model assumes that we already have access to a detailed fine-scale geological model, and is well-suited as a reduced-order or proxy model. However, we can consider a more radical data-driven setting, starting from close-to-zero knowledge of the reservoir and calibrating the model largely as a black-box method. This has previously been tested for CGNet, where the model was constructed through a fictitious domain approach by wrapping a coarse rectilinear mesh around the assumed reservoir outline, and removing cells which fall outside the domain [10]. The resulting graph abides the original domain outline to some extent, but if the rectilinear mesh is very coarse, the resolution may be too low to get an accurate representation of the boundary. Using a triangulation, we can construct a graph that better accounts for the presumed domain boundary by explicitly including boundary points as outmost nodes in the triangulation. In particular, we form a triangulation from the wells and selected points along the reservoir boundary.

5.2.1 Constructing the coarse graph

As indicated by the name, a TriNet relies on a triangulation. Herein, we restrict the discussion to two dimensions. When the original model is three-dimensional, we work with the lateral projection of the reservoir, performing the triangulation in the xy-plane. Moreover, we assume wells to be vertical, so that each well can be represented by a single point in the xy-plane. We propose an extension to 2.5D in Section 5.2.3.

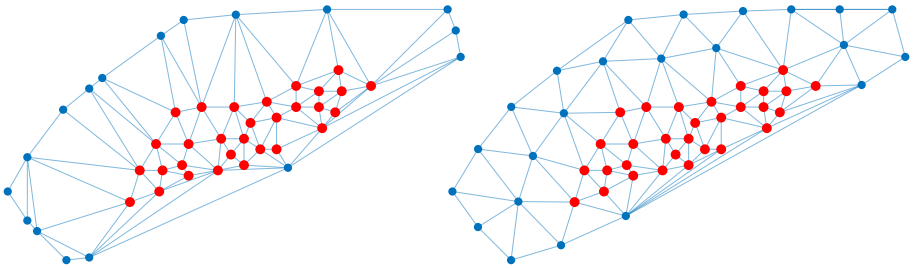


Figure 5.8: Two triangulation-based network models for the Brugge case, without (left) and with (right) DistMesh.

To perform a triangulation, we first need to select a set of points. To that end, we first collect the coordinates of all the wells. Additionally, we include a selection of points along the boundary. For simple geometries, we can extract the boundary through a convex hull. We suggest to set some minimum distance between points, and merge all points that are closer than this threshold. Once we have selected our set of points, we can proceed with a Delaunay triangulation. This is a triangulation with the property that the disc circumscribed to each triangle contains no vertex (none of our selected points) [23]. The resulting triangulation directly translates into a graph, where the selected points serve as the nodes, and the triangle sides as the edges.

The direct Delaunay triangulation can often be highly uneven, which is associated with poor numerical performance when used directly as a simulation grid. A perhaps better alternative is to use DistMesh [40], in particular, the extended version of DistMesh implemented in UPR [41]. DistMesh builds upon the Delaunay triangulation, but incorporates a force-based smoothing procedure aiming to optimize the node locations. This generally gives well-shaped meshes. It also supports, among other options, setting an initial edge length h_0, thereby providing a simple way of adjusting the granularity of the model. We propose to set h_0 relative to the diagonal of the bounding box of the domain, l_{diag}; that is, use $h_0 = \alpha_0 \cdot l_{diag}$ for some constant α_0. Figure 5.8 shows two possible triangulation-based models for Brugge, one without and the other with DistMesh. The effect of DistMesh is clear, giving a significantly more even triangulation. In this case, the DistMesh-based model greatly benefits from the relative edge length option, as the convex hull gives two long line segments in the bottom part. It is possible to enforce some maximum edge length in the none-DistMesh construction as well. Instead of directly using the points returned by the convex hull, we could manually add or adjust the points before performing the Delaunay triangulation. However, since this is already implemented in DistMesh, along with numerous other options, it may not be worth the effort.

Physical interpretation*

Unlike a traditional grid-based model, a network model does not necessarily have a natural physical interpretation. In theory, you can add edges between any pair of nodes within the network. However, for structured geometries, it is possible to convert the graph back into a grid. For rectilinear CGNet-type models, the graph-to-grid mapping is simply the reverse of the construction. That is, since we originally used the cells/blocks in the grid as our nodes, and mapped faces to edges, we can easily revert this process and obtain a physical grid. For graphs based on a Delaunay triangulation, we can use the Voronoi diagram. The Voronoi diagram and Delaunay triangulation are dual to each other, with the nodes of the Voronoi control cells corresponding to the centers of the circles circumscribed around the Delaunay triangles [23]. Figure 5.9 provides two illustrations of the Delaunay-Voronoi duality.

Although a network model in general need not have an interpretation in physical space, it can be useful to keep this backwards mapping in mind. In particular, observe how a node in the graph represents some physical volume, just like it did when we constructed a CGNet from a coarse grid.

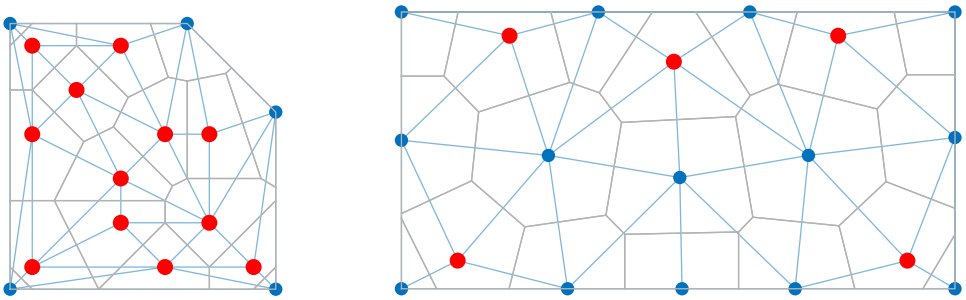


Figure 5.9: Two examples of triangulation-based network models, including the dual Voronoi diagram clipped against the outer boundary in gray.

5.2.2 Calibrating the initial saturation

The simulation captures how the reservoir fluids evolve from an initial state, when impacted by some external driving forces like injection wells. If the initial state itself is far off, the model may consequently be hard to calibrate. For the partition-based models, we converted the initial state to the coarse grid through some upscaling procedure. At least to some extent, this keeps the local heterogeneity of properties like initial saturations. For triangulation-based models, the geometry generally makes it harder to map properties from the original model to the network model. We could construct a mapping using the Voronoi grid, but this would be more computationally expensive and complicated to implement and has not been included in this project. The TriNet models may consequently suffer from poor initializations.

To remedy this, we propose to instead add variables like initial water saturation to the tunable parameters and try to calibrate our way to a reasonable value. As long as we simulate two-phase water-oil systems, it suffices to add the water saturation S_w; the oil saturation S_o will be given implicitly since $S_w + S_o = 1$. To calibrate S_w, we first need to make some initial guess. In lack of a more sophisticated mapping from the fine model, we can average over all fine cells. If the total reservoir contains, say, 60% water and 40% oil, we initialize $S_w = 0.6$ and $S_o = 0.4$ in all nodes. Then we can tune the water saturation within the box constraints $0 \leq S_w \leq 1$.

5.2.3 Extending to 2.5D

The original TriNet framework gives two-dimensional models. However, many reservoirs exhibit a layered structure and involve non-negligible buoyancy effects, possibly requiring a three-dimensional model in order to give accurate simulations. To this end, a simple extension of the TriNet model is suggested, stacking copies of the two-dimensional graph on top of each other, with vertical connections between the layers. We refer to this as 2.5D.

Unless the reservoir is perfectly box-shaped, the nodes of the stacked model may fall outside the original domain. To circumvent this, one could interpolate the coordinates onto the surface of the reservoir. The heights of the nodes affect the gravitational effects in the simulations. An alternative remedy is to consider the gravitational effect itself as a tunable parameter.

Recall from Chapter 3 the fully implicit system, including the discrete flux (3.4.6),

$$\boldsymbol{v}_\alpha^{n+1} = -\lambda_\alpha \boldsymbol{T}[\texttt{grad}(\boldsymbol{p}_\alpha^{n+1}) - g\boldsymbol{\rho}_\alpha^{n+1}\texttt{grad}(\boldsymbol{z})], \tag{5.2.1}$$

For the 2.5D models, the heights of the layers can be chosen freely. Tuning the gravity term $g \cdot \texttt{grad}(\boldsymbol{z})$ could prevent inaccuracy stemming from poor height choices. When doing so, we can use the bottom and top of the reservoir to set suitable box limits. In particular, we identify the maximum and minimum z coordinate of all centroids, say z_{min}, z_{max}, and calculate our limit $l = g(z_{max} - z_{min})$. Then, we set box limits $[-l, l]$.

We remark that the $g \cdot \texttt{grad}(z)$ term can be both negative and positive, depending on the defined direction of the face/edge in question. An improved box limit could account for this by prohibiting

the term to change sign.

5.2.4 Flow-adapted models using distance to closest well

For CGNets, we proposed to use flow indicators like residence time to construct flow-adapted models a priori. Since TriNets should not rely on the availability of a fine-scale model, we propose a less sophisticated and more naïve approach, using distance to wells. That is, we use the distance to the closest well to indicate high-flow areas of the reservoir. In most cases, it is a reasonable assumption that more fluids will be displaced closer to wells. In fact, we saw in Figure 5.6 that the well-distance and residence-time fields were qualitatively similar for the Brugge model. Hence, the distance metric seems like a good alternative to construct TriNet models adapted to the flow.

As long as we construct our models using DistMesh, it is fairly straightforward to modify the local granularity. This is done by providing a tailored *edge length function*, $h(x)$, which returns the desired relative edge length for each point x in space [40]. We can construct $h(x)$ in such a way that it gives shorter edge lengths closer to wells and longer edge lengths farther from wells. In particular, we let $d(x)$ be the distance from point x to the closest well. Then, define

$$\begin{aligned} t(x) &\coloneqq \min\left\{ \exp\left(\frac{d(x)}{0.25 * l_{diag}}\right) - 1,\, 1\right\}, \\ h(x) &\coloneqq t(x) \cdot h_{\max} + (1 - t(x)) \cdot h_{\min}, \end{aligned} \tag{5.2.2}$$

where $h_{\min}$ and $h_{\max}$ denote the minimum and maximum relative edge length, respectively, and l_{diag} is the diagonal length of the bounding box of the domain. Here, $t(x)$ goes to 0 near the wells, and 1 sufficiently far away, with the effect that $h(x)$ goes towards $h_{\min}$ close to wells and $h_{\max}$ far away.

5.3 Automatic graph refinement*

The graphs used in our network models have a modest number of degrees of freedom compared to standard reservoir simulation models, enabling applications like adjoint-based gradient optimization. They should however not be too coarse. For the model calibration to succeed, a certain number of tunable parameters is required. If a model proves too coarse, the obvious action is to try a new and finer model. However, increased resolution may not be equally needed everywhere and it is desirable to add parameters where they have the largest effect. We have already discussed some approaches to construct non-uniform models a priori, that is, before calibration. In that case, we relied on our own understanding of the flow physics, expecting the high-flow regions to require higher resolution. We now consider a quite different approach, where we let the model automatically refine itself based on parameter sensitivities obtained in calibration.

5.3.1 Selection for refinement

When tuning the model parameters, the solution of the adjoint problem computes the Jacobian matrix of the mismatch functions via automatic differentiation. At a certain data point, a derivative of the mismatch function with respect to some parameter intuitively tells how the mismatch is influenced by a change in that parameter. Calculating the standard deviation across all data points for a certain parameter, we get an indicator of where we need more information. If the standard deviation is large, there is disagreement between the data points on how to adjust the parameter to improve the objective function. Adding more parameters by refining the graph in that area may help.

We define the refinement indicator I_{ref} as the standard deviation of the entries in the mismatch Jacobian $J_{\boldsymbol{r}}$ belonging to a single parameter. Recall that $J_{\boldsymbol{r}}$ (4.2.1) has dimension $N_\theta \times N_y$, so

one row contains the derivatives of all residuals r_j (for all N_y data points) with respect to a single parameter θ_i, such as the pore volume of one specific node,

$$J_{\boldsymbol{r}} = [\nabla_\theta r_1, \nabla_\theta r_2, \ldots, \nabla_\theta r_{N_y}] = \begin{bmatrix} \frac{\partial r_1}{\partial \theta_1} & \frac{\partial r_2}{\partial \theta_1} & \cdots & \frac{\partial r_{N_y}}{\partial \theta_1} \\ \frac{\partial r_1}{\partial \theta_2} & \ddots & \ddots & \frac{\partial r_{N_y}}{\partial \theta_2} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial r_1}{\partial \theta_{N_\theta}} & \frac{\partial r_2}{\partial \theta_{N_\theta}} & \cdots & \frac{\partial r_{N_y}}{\partial \theta_{N_\theta}} \end{bmatrix}. \tag{5.3.1}$$

We thus take the standard deviation across one row,

$$I_{\text{ref}}(\theta_i) = \text{STD}\left[\frac{\partial r_1}{\partial \theta_i}, \frac{\partial r_2}{\partial \theta_i}, \ldots, \frac{\partial r_{N_y}}{\partial \theta_i}\right]. \tag{5.3.2}$$

If we select nodes or edges, we can use the sensitivities directly. For a triangle, on the other hand, we need some kind of mapping from its nodes and edges to the triangle. It seems natural that a triangle with larger refinement indicators on its components should be refined before triangles with smaller indicators. Possible mappings include the maximum, mean and sum over the parameters on the triangle's nodes and edges. Note that the sum indicator will naturally favor well nodes, since they have an extra parameter, and might therefore not be suitable. The max indicator has the drawback of discarding information. If two triangles have the same maximum, but one has large values all over, while the other has small, they are treated as equal. The mean indicator avoids both these issues. Letting θ_T denote the set of parameters θ_i belonging to the nodes and edges of triangle T, we can formally define the mean refinement indicator

$$I_{\text{ref}}(T) = \frac{1}{|\theta_T|} \sum_{\theta_i \in \theta_T} I_{\text{ref}}(\theta_i). \tag{5.3.3}$$

Having such a refinement indicator I_{ref}, we need to decide the extent of the refinement. One option is to select the parts where I_{ref} is larger than some prespecified tolerance, or alternatively, selecting a subset of these triangles. This would give a stopping criterion for the graph update loop as well, when no triangles have indicators above the bound, but setting an appropriate tolerance can be challenging. An alternative is to select some prespecified fraction, e.g., selecting the 10% triangles with the largest refinement indicator. This may result in only one of two triangles with equal indicator being selected, especially when using a maximum mapping. Using a mean or sum mapping, on the other hand, this will likely occur very seldom. When using this selection criterion, we need an additional stopping criterion for the refinement. Alternatives include setting a maximum number of refinements, a maximum number of parameters, and a mismatch tolerance.

5.3.2 Refinement

In addition to nodes and edges, our graphs have structures that allow us to identify triangles or rectangles, depending on the chosen strategy when first constructing it. Refinement of a graph may be based on either of these, or a combination. In this setting, we will only consider refinement of triangle-structured graphs. We proceed to introduce methods for refining a selected triangle, edge or node.

For a triangulation-based graph, we can select and refine triangles. Since automatic graph refinement will likely include repeated refinements, the triangle refinement should preferably maintain the triangular structure. A simple option is to split each triangle in three, drawing edges from its centroid to each corner. This would give progressively sharper triangles, so we instead propose a method inspired by red-green mesh refinement in the finite element method. Here, the refinement consists of two steps: the red step where we refine the selected triangles and the green step where we fix hanging nodes. In the red step, the selected triangles are split in four by splitting each of the three edges. The new nodes are connected together, forming a new triangle in the middle. The procedure is illustrated in Figure 5.10. The refinement gives the graph three additional node

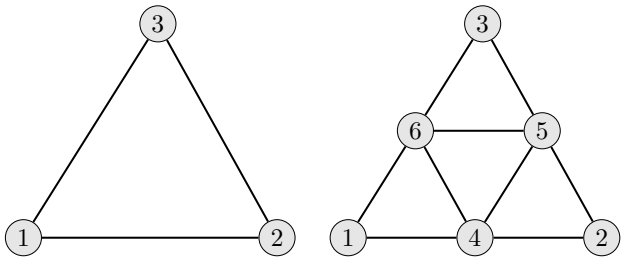


Figure 5.10: A triangle before and after the red step of triangle refinement.

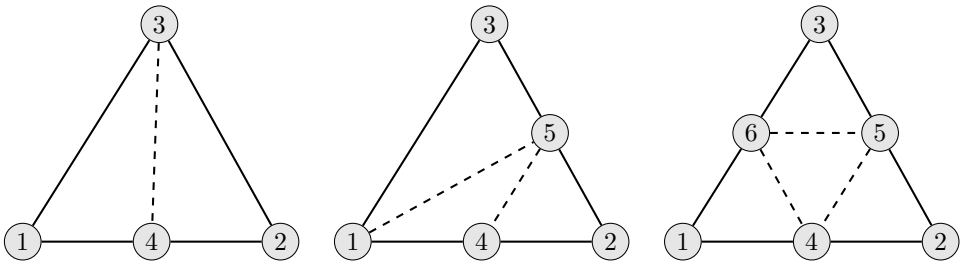


Figure 5.11: The three cases in the green step of triangle refinement. In each case, the hanging nodes are removed by adding new edges.

parameters and six additional edge parameters.

When refining the selected triangles in the red step, we keep track of and flag their neighbors. We can then iterate over the flagged triangles and fix them, that is, remove hanging nodes while recovering the triangular structure. A flagged triangle can have either one, two or three red-refined neighbor triangles. The green step of refinement in each case is illustrated in Figure 5.11. For one hanging node, it is simply connected to the node opposite to it. For two hanging nodes, an edge is added between them, as well as an edge from one of them to its opposite node. Finally, if the flagged triangle has three hanging nodes, the refinement looks just like in the red step, connecting all the hanging nodes to each other.

Having selected an edge for refinement, a natural way to refine is to split that edge. To maintain the triangular structure, we also connect the new middle node to the opposite corners, as illustrated in Figure 5.12. The resulting graph has one more node parameter and three more edge parameters.

Finally, selected nodes can most easily be refined by refining all triangles the given node is part of. While this approach is well-defined and symmetric, it is very extensive, heavily changing the graph. It is, however, not trivial coming up with an alternative which maintains the triangular structure. Selecting triangles instead, we avoid this problem, and ill-represented nodes should still be effectively refined given a reasonable refinement indicator mapping to triangles, $I_{\text{ref}}(T)$.

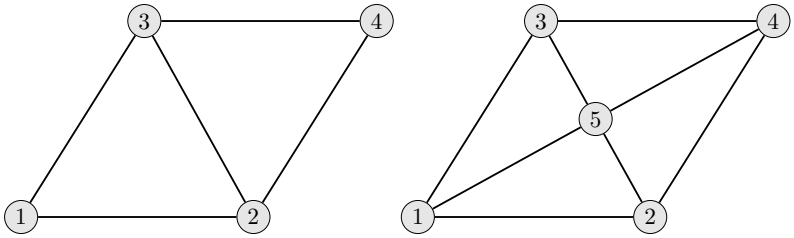


Figure 5.12: Edge refinement of the edge (2,3). The edge is split in two and the new node is connected to the opposite nodes of the two triangles.

5.3.3 Initial parameter values for new nodes and edges

After refining the graph, we need to assign initial parameter values to all new nodes and edges. For the unchanged nodes and edges, we map over the tuned parameters from the previous graph, while for new parts, we need to come up with some initial guess. This is done locally, taking the average value of the neighbourhood. For a new node $v_{new} \in V^{new}$, a parameter θ is set to

$$\theta(v_{new}) = \frac{1}{|N^*(v_{new})|} \sum_{v \in N^*(v_{new})} \theta(v), \tag{5.3.4}$$

where $N^*(v_{new})$ is the neighborhood $N(v_{new})$, but excluding other new nodes, that is, $N^*(v_{new}) = \{v \in N(v_{new}) \mid v \notin V^{new}\}$. In other words, when assigning a new node parameter, we use the average of all the nodes that the node is adjacent to, excluding those that are also new. In the case when a new node is only adjacent to other new nodes, a global average is used instead. Similarly for a new edge $e \in E^{new}$, the parameter is calculated by averaging over the edges that the new edge is adjacent to, excluding new ones,

$$\theta(e_{new}) = \frac{1}{|N^*(e_{new})|} \sum_{e \in N^*(e_{new})} \theta(e). \tag{5.3.5}$$

We remark that it is possible to largely avoid the global average fallback using an iterative strategy, skipping the nodes/edges with only new neighbors until at least one of their neighbors have been assigned a value. However, these new parameters are part of the initial guess of the Levenberg–Marquardt algorithm for the updated graph and will be tuned. The goal is therefore not more ambitious than to set *reasonable* values.

Note that when using this strategy, the sum of the parameters will increase after each graph refinement. For parameters such as pore volume, this is problematic, since the total reservoir volume would in effect increase in each iteration. To circumvent this, the pore volumes can be rescaled to keep the sum constant. Doing this globally, simply dividing all parameters with the current sum and multiplying by the previous, is the simplest way. However, this would affect areas of the graph where there was no refinement. A local strategy avoids this. This requires keeping track of which parts of the graph were affected by the refinement, for example choosing all new nodes and their neighbors and rescaling only their pore volume values. A fully local approach could even redistribute pore volumes in each single triangle refinement. In this project, we use a semi-local strategy in which we identify the affected areas, and rescale all pore volumes therein by the same factor.

5.3.4 Full graph optimization algorithm

Combining the steps just described, we get a full graph optimization procedure, which starts from an initial graph and attempts to optimize both the graph topology and the model parameters. The resulting algorithm can be seen as a double loop. The inner loop optimizes the model parameters on each graph using the Levenberg–Marquardt algorithm. The outer loop updates the graph topology based on sensitivities from the parameter tuning. This gives a highly flexible algorithm with a number of options, including choosing the initial graph, convergence criteria for both the inner and outer loop, methods for both refinement selection and the refinement itself, and methods to calculate parameters for the new nodes and edges.

We will use the relative mismatch reduction,

$$\frac{M(j-1) - M(j)}{M(j)},$$

as a stopping criterion for the parameter tuning. When this is small ($< tol$), it indicates that the improvement for the current model has stagnated. If the achieved mismatch is not satisfactory, we can add more parameters and try again. That is, we refine the graph until the absolute mismatch meets some tolerance ($M(j) < TOL$). A pseudocode for the graph refinement procedure follows.

procedure OPTIMIZENETWORKMODELTOPOLOGY(G_0)
  for $i = 1, 2, \ldots$ **do** ▷ Outer loop: Graph topology
    for $j = 1, 2, \ldots$ **do** ▷ Inner loop: Graph parameters
      Calibrate parameters θj using Levenberg–Marquardt.
      Simulate and evaluate objective function M.
      Simulate adjoint system and assemble mismatch Jacobian.
      if $(M(j-1) - M(j))/M(j) < tol$ **then** ▷ Indicates stagnation
        break ▷ Stop tuning on current graph
      end if
    end for
    if $M(j) < TOL$ **then return** ▷ Satisfied with result: Stop.
    else ▷ Not satisfied: Refine graph.
      Calculate refinement indicator $I_{\text{ref}}(T)$ (5.3.3) for all triangles.
      Select parts for refinement. ▷ e.g., top 10%
      $G_i = \text{REFINEGRAPH}(G_{i-1})$. ▷ Red-green triangle refinement
      Fill in parameter values on G_i using (5.3.4), (5.3.5). ▷ Prepare for new tuning
    end if
  end for
end procedure

5.4 Implementation and MRST integration

Both model types in consideration offer the advantage of fitting almost immediately into a standard finite-volume-based simulator. This section provides a brief explanation of their implementation and the measures required to simulate them with MRST. For all details, consult the source code and associated documentation [42].

The main infrastructure for a graph-based reservoir model can be found in the `NetworkModel` class. A `NetworkModel`, like an MRST simulation setup, holds three objects/classes: a `model`, a `schedule` and an initial state `state0`. In addition, we include two important properties: the `network` and the `params` object.

The `network` is an instance of the `BaseGraph` class, which implements various utilities for working with a graph. The `network` stores the MATLAB graph `G`, and works as a wrapper for the built-in graph functionality. For an introduction to graphs and the MATLAB graph, see Appendix A. You could say that the `network` replaces the grid. Note, however, that when you want to simulate the model, you need to update the MRST grid, found in `model.G`, but you do *not* need a fully defined geometry. It is sufficient to:

- Update the discrete operators with appropriate dimensions, e.g.,

```
pv = mean(model.operators.pv)*ones(network.numNodes, 1)
T  = mean(model.operators.T)*ones(network.numEdges, 1)
model = model.setupOperators(model.G, ...
            model.rock, ...
            'trans', T,  ...
            'neighbors', network.G.Edges.EndNodes, ...
            'porv', pv);
```

  Here, we use a mean value as our initial guess for pore volumes and transmissibilities. The `neighbors` operator (`model.operators.N`) can be set directly to the edge list representation (`EndNodes`) of the graph.

- Update the number of grid cells to match the number of nodes in the `network`.

```
model.G.cells.num = network.numNodes();
```

- Update the cell centroids to the node coordinates, e.g.,

```
model.G.cells.centroids(:,1) = network.G.Nodes.x
```

The node coordinates are stored as columns in the node table of the graph. Note that the coordinates in most cases do not impact the simulation, except the gravitational effect in 3D cases, but this is necessary for the simulation to run, and also useful for visualization purposes.

Moreover, you need to make sure that the initial state in `state0` matches the dimensions of your network. There should be as many pressures and saturations as there are nodes.

The `schedule` contains the well structure. As most models are originally three-dimensional, the wells have several perforations. When constructing two-dimensional network models, we need to create new well structures with the right number of perforations, keeping track of which nodes in the graph are the well nodes.

Both the partition-based and triangulation-based network models fit into this framework, and their implementation is based on inheritance from the `NetworkModel` class, with separate implementation of more specialized functionality and properties. For example, a partition-based model should store the partition vector and original model, whereas the triangulation-based model needs a data structure representing the triangles.

The `params` property is a cell array of `NetworkModelParameter` objects, which inherit from MRST's `ModelParameter` class. These are the tunable parameters, and each `NetworkModelParameter` holds information about one parameter type, such as pore volumes. The parameter values themselves are not stored here, but each parameter holds information about where in the simulation setup (`model, schedule`, `state0`) its values are found.

The network models will be calibrated using the methodology described in Chapter 4. Table 5.1 summarizes the tunable parameters we can include and also includes suggestions for their limits. Note that in the original schedule, each well has one well index per perforation. In the network model, each perforation is represented by its own well node, so that each well index belongs to a single well node.

We remark that the computational overhead of MATLAB is significant. When simulating with MATLAB and MRST, we do not fully unlock the speed potential of very coarse models. Simulations, and thereby calibrations, would likely be significantly faster in a language like Julia, e.g., using the JutulDarcy reservoir simulator [43].

Table 5.1: Available tunable parameters in calibration of the network models.

Name	Description	Location	Suggested limits
porevolume	Pore volumes $\boldsymbol{\Phi}$	Nodes	$[0.01, 10]$ relative
transmissibility	Transmissibilities $\boldsymbol{T}$	Edges	$[0.01, 100]$ relative
conntrans	Well indices $\boldsymbol{J}$	Wells	$[0.01, 100]$ relative
swl	Rel.perm. scaling	Nodes	$[0, 0.4]$ box
swcr	Rel.perm. scaling	Nodes	$[0, 0.4]$ box
swu	Rel.perm. scaling	Nodes	$[0.8, 1]$ box
sowcr	Rel.perm. scaling	Nodes	$[0, 0.4]$ box
krw	Rel.perm. scaling	Nodes	$[0.2, 2]$ box
kro	Rel.perm. scaling	Nodes	$[0.2, 2]$ box
sw	Initial water saturation	Nodes	$[0, 1]$ box
gdz	Gravitational effect	Edges	$g \max(\mathtt{div}(z))$ box

Chapter 6

Simulation Results

The graph-based reservoir simulation framework described in the previous chapter is best illustrated by means of examples. In the following sections, the models are demonstrated for a number of cases. We first consider a constructed two-dimensional demo problem, well-suited for a demonstration of the methods. Then, we apply the ideas to the Egg model, a synthetic channelized reservoir, where we also test the model generality by introducing control perturbations. Next, we test the effect of multiple layers on a simulation model of the Norne field, before demonstrating the importance of an accurate initial saturation for a synthetic (but realistic) model realization from the SAIGUP study. As a final example, we discuss the Brugge benchmark model, also demonstrating a priori flow-adapted models and a production optimization application.

All results should be reproducible. To that end, MATLAB function scripts for the different examples and experiments are available in a Bitbucket repository [42], where they are found in the `examples/master-thesis` directory.

6.1 A demo problem

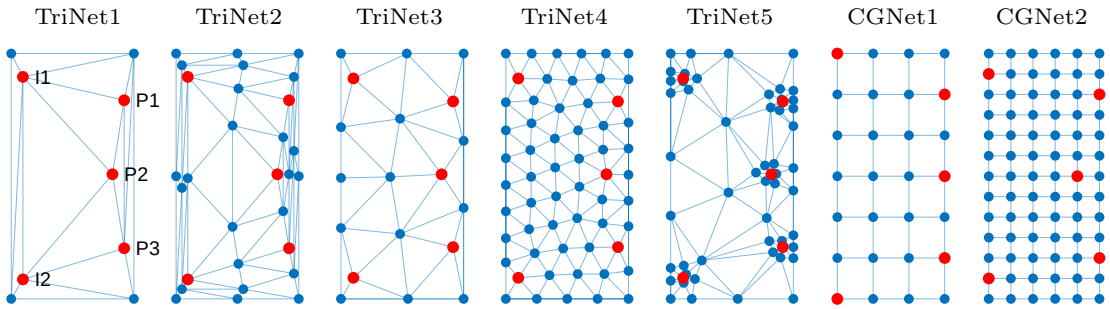
For a first demonstration of the methods, we consider a simple constructed two-dimensional setup with five wells, taken from chapter 13 in [19] [1]. The two injectors inject at a $0.0594\,\mathrm{m}^3/\mathrm{s}$ rate, while the three producers maintain a 100 bar pressure. We use an immiscible two-phase fluid model, with initial oil saturation equal to 1, formulated on a 5000×2500 m domain divided into 4096 grid cells. The duration is 2 years, divided into 33 time steps. We are free to choose the porosity and permeability fields of the model, and will consider a homogeneous and a heterogeneous case.

Figure 6.1 depicts seven possible coarse graph representations of the model. The leftmost is a pure Delaunay TriNet, followed by a preprocessed variant where all triangles have been refined once. Next are three DistMesh-based TriNets, the first with a maximum relative edge length 0.25, the second with 0.1, and the third with increased granularity close to wells. Finally, we have a $7 \times 4 \times 1$ and a $13 \times 6 \times 1$ CGNet. In all calibrations for this case, we use pore volumes, transmissibilities, and well indices as tunable parameters, with the limits given in Table 5.1. Figure 6.1 also holds a table stating the dimensions and total number of parameters for each model.

6.1.1 Homogeneous case

For the first demonstration, we use homogeneous permeability and porosity, with constant values 695 md and 0.40, respectively. We may then expect some symmetry in the graph refinement.

[1] This example is based on one from the specialization project [14], where the same setup was used, but the results and discussion presented herein have been modified.



Model	Nodes	Edges	Parameters	Description
TriNet1	9	20	34	Pure Delaunay
TriNet2	29	76	110	Uniform refinement of TriNet1
TriNet3	19	46	70	DistMesh, $\alpha_0 = 0.25$
TriNet4	59	162	226	DistMesh, $\alpha_0 = 0.1$
TriNet5	56	152	213	DistMesh, well-adapted ($\alpha_0 = 0.075$, $h_{min} = 0.05$, $h_{max} = 0.5$)
CGNet1	28	45	78	Cartesian $7 \times 4 \times 1$
CGNet2	78	137	220	Cartesian $13 \times 6 \times 1$

Figure 6.1: Six graph representations for the demo case.

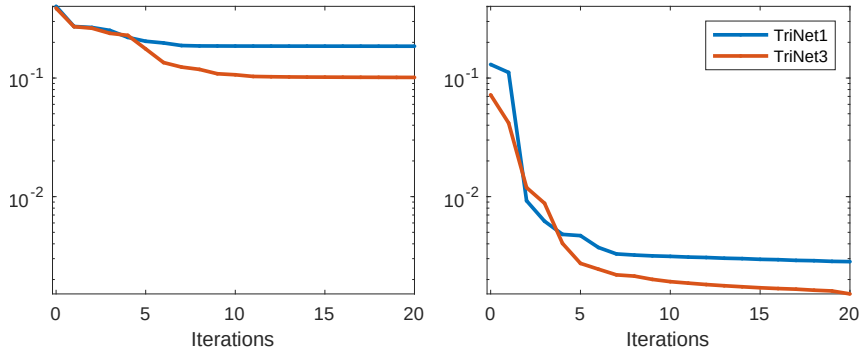


Figure 6.2: Comparison of mismatch reduction using TriNet1 and TriNet3. For the left case, the pore volumes are the average over the fine-scale model cells, and in the right case they have been rescaled to preserve the total volume.

Misfit reduction – the impact of rescaling

We start by comparing the performance of TriNet1 and TriNet3, tuning the parameters for 20 iterations. The resulting mismatch reductions during the Levenberg–Marquardt iterations are shown in Figure 6.2. For the left case, the pore volumes are set directly to the average pore volume in the fine-scale model. The right case includes a rescaling of the pore volumes, forcing the total pore volume to equal that in the original model. The results indicate that a poor initial guess for the pore volumes can make the models hard to calibrate, and they will thus be rescaled in all upcoming examples. In both cases, the DistMesh-based model TriNet3 achieves a slightly better mismatch than the TriNet1 model. However, it should be noted that TriNet1 is more coarse, with only 34 parameters, while TriNet3 has 70. Although TriNet3 calibrates better, it is therefore difficult to conclude which one is actually better. Still, looking at the graphs, and recalling the previously discussed Delaunay-to-Voronoi mapping, the more even DistMesh option does seem like a good choice.

Even though the mismatch reduction is less than an order of magnitude for the left, non-rescaled, case, the models are learning, and we can investigate what happens more closely. Since the parameter tuning seeks to minimize the error in the well curves, a mismatch reduction should correspond to the calibrated curves closing in on the reference curves. This effect is visualized in Figure 6.3. The water and oil rates in producer P1 are plotted for the first ten iterations of the TriNet3 tuning. The calibrated curves become visibly closer to the reference curve with each iteration. There is even a significant improvement between iteration 5 and 6, in agreement with the steeper part of the mismatch reduction plot.

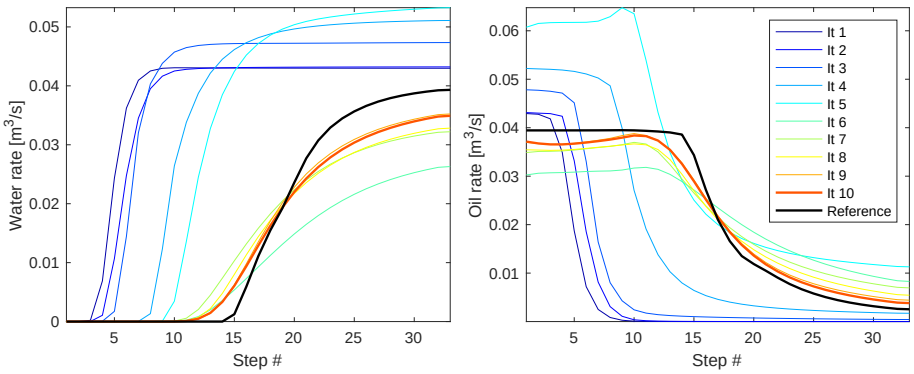
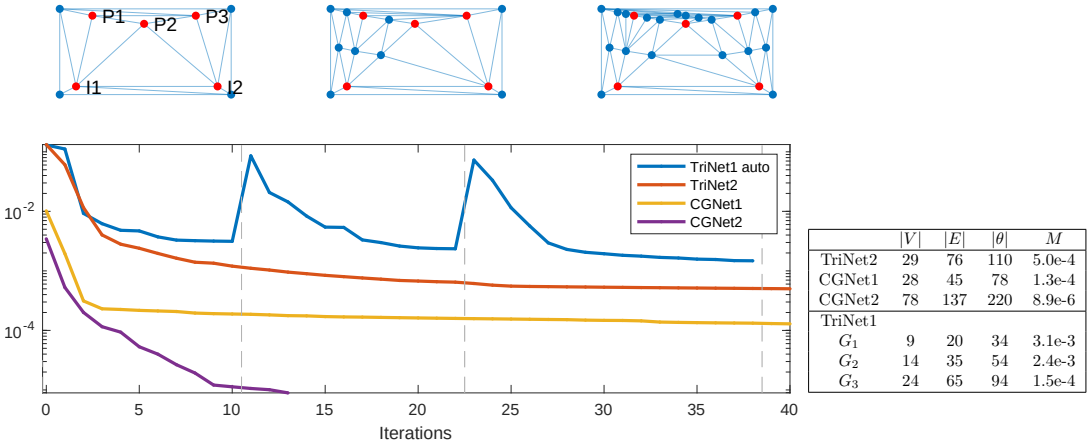


Figure 6.3: Water rate (left) and oil rate (right) for the well P1 after each iteration of the parameter tuning on TriNet3.



	$\|V\|$	$\|E\|$	$\|\theta\|$	M
TriNet2	29	76	110	5.0e-4
CGNet1	28	45	78	1.3e-4
CGNet2	78	137	220	8.9e-6
TriNet1				
G_1	9	20	34	3.1e-3
G_2	14	35	54	2.4e-3
G_3	24	65	94	1.5e-4

Figure 6.4: Mismatch reduction for the homogeneous demo case, using the pure Delaunay TriNets and the two CGNets. Above the mismatch plot, the three graphs G_1-G_3 in the automatic refinement of TriNet1 are shown.

Autotuning model granularity

If the tuning has stagnated and we want to reduce the mismatch further, a natural action is to add more parameters, which corresponds to using a finer graph. We can refine the graph uniformly, increasing the resolution everywhere, or adaptively, e.g., using the graph refinement algorithm introduced in Section 5.3. The two next experiments include refinement of coarse TriNets, and compare the results with a uniformly finer TriNet, as well as a coarse and finer CGNet.

For the first refinement example, we use TriNet1 as the initial graph. The stopping criterion for parameter tuning on each graph is the relative mismatch reduction in a single iteration being less than 1%. When this occurs, 20% of the triangles are selected for refinement using the indicator in (5.3.3). We also include static parameter tuning on TriNet2, CGNet1 and CGNet2. The results are shown in Figure 6.4, and the code for this example is found in `demoHomogeneousCalibration.m`.

The finer CGNet2 calibrates very quickly, reaching the 10^{-5} stopping criterion in only 14 iterations. The fine TriNet2, having half as many parameters, trains slower and stagnates around $5 \cdot 10^{-4}$. Looking at the first iteration, we see that the initial mismatch of the TriNets is at least an order of magnitude larger than those of the CGNets. In fact, the initial objective values are about 0.13 for TriNet1 and TriNet2, 0.01 for CGNet1, 0.0034 for CGNet2. With that in mind, the TriNet2 calibration is comparable to that of the CGNets in terms of reduction relative to the starting point. The fact that the TriNets have larger initial mismatch values suggests that the initial parameter guesses work better for CGNet, at least in this case. The TriNet1 refinement increases resolution most around the producers, with each refinement giving little extra reduction in the mismatch. The final graph G_3 is slightly coarser than TriNet2, and stagnates at a slightly larger final mismatch.

We again stress that the tunable parameters are not considered to be physically representative,

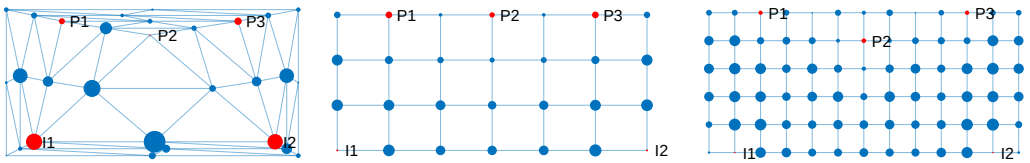
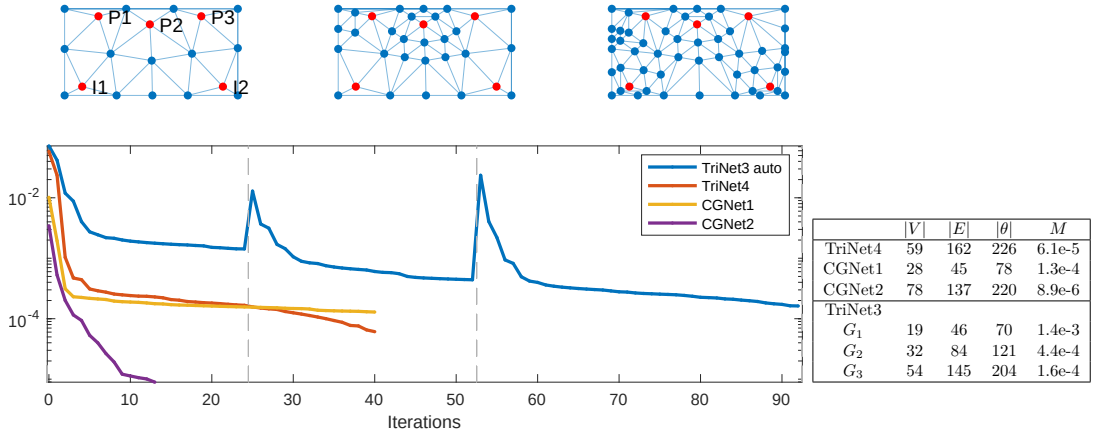


Figure 6.5: Calibrated pore volumes illustrated as node sizes for TriNet2, CGNet1 and CGNet2.



	$\|V\|$	$\|E\|$	$\|\theta\|$	M
TriNet4	59	162	226	6.1e-5
CGNet1	28	45	78	1.3e-4
CGNet2	78	137	220	8.9e-6
TriNet3				
G_1	19	46	70	1.4e-3
G_2	32	84	121	4.4e-4
G_3	54	145	204	1.6e-4

Figure 6.6: Mismatch reduction for the homogeneous demo case, using the DistMesh-based TriNets and the two CGNets. The three graphs G_1-G_3 in automatic refinement of TriNet3 are shown on the top.

but calibrated freely within some loose bounds. Figure 6.5 illustrates the calibrated pore volumes for TriNet2, CGNet1 and CGNet2. In particular, the node sizes are scaled according to their pore volumes. Since this model has homogeneous porosity, physically speaking, the pore volumes should be equal in a uniform model. This is clearly not the case after calibration.

We now perform a similar experiment, but this time using the more even DistMesh-based TriNet3 as the initial graph and comparing with the finer graph TriNet4 in addition to CGNet1 and CGNet2 as in the previous case; see Figure 6.6 for the results and `demoHomogeneousDistmeshCalibration.m` for the script.

This time, the fine TriNet4 is slightly better than the coarse CGNet1 but still not quite competing with the fine CGNet2, which has approximately the same number of parameters. The TriNet3 refinement is, like for the TriNet1 refinement, first concentrated around the producers and then around the injectors. The two refinements give some extra mismatch reduction, with G_3 ending at a mismatch slightly larger than for CGNet1, but with significantly more parameters.

In this case, the refinement looks quite symmetrical. This may be an advantage, and a more even initial graph is a good starting point for a more even refinement. However, one should not read too much from how the graphs look. They are plotted using physical coordinates mostly for visualization purposes. This can be misleading, since the physical coordinates do not really matter in the simulation. Figure 6.7 shows an alternative, circular way of illustrating TriNet3, which is just as correct. Clustering of refinement around a well, which may look bad in the physical plots, is not necessarily a problem. The effect is adding more possible paths to or from the given well. That said, this should be investigated, e.g., by testing if penalization of repeated refinement improves the results. Either way, one should not focus too much on the graphs *looking* good, when plotted with physical coordinates.

The achieved mismatch values are quite small, which naturally corresponds to a close match in the well curves. With the final mismatch in TriNet3 refinement being as small as $1.6 \cdot 10^{-4}$, we expect a good match, which is what we see in Figure 6.8. As the producer curves are hard to separate, we take a closer look in Figure 6.9, showing the water and oil rate in P1, including the tuned CGNet2, the refined TriNet3 (G_3) and the reference. The curves are hard to distinguish in the top row, but looking at the difference between the calibrated curves and the reference (bottom row), we see that CGNet2 is slightly better, in agreement with the difference in final mismatch.

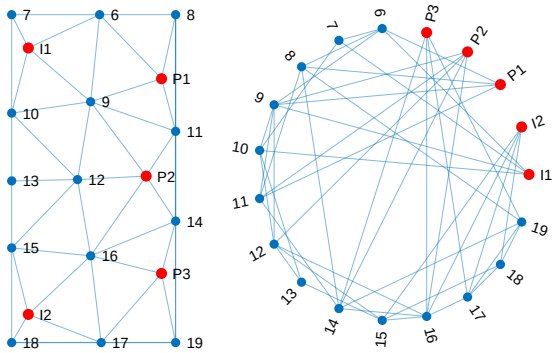


Figure 6.7: Two alternatives for illustrating TriNet3: the plot to the left uses physical coordinates from the generating triangulation to lay out the nodes of the graph and the plot to the right uses a circular layout that disregards the physical position of the nodes.

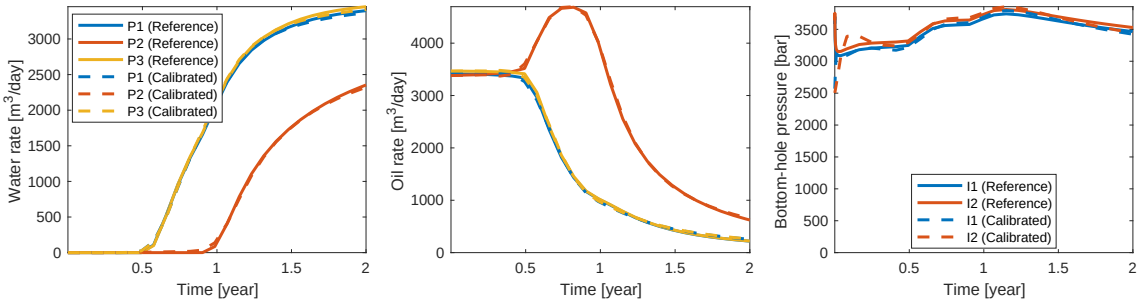


Figure 6.8: Comparison of well responses for the refined TriNet3 network model ("calibrated" in the legends) and the fine-scale model ("reference" in the legends).

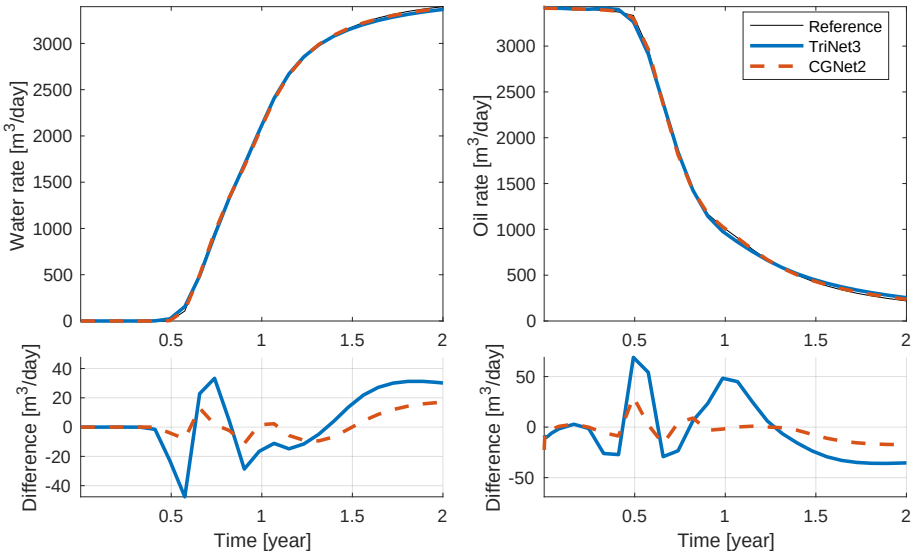


Figure 6.9: Comparison of well curves from producer P1 for CGNet2 and the refined TriNet3. The top row shows the water and oil rates and the bottom row the difference between the calibrated curves and the reference.

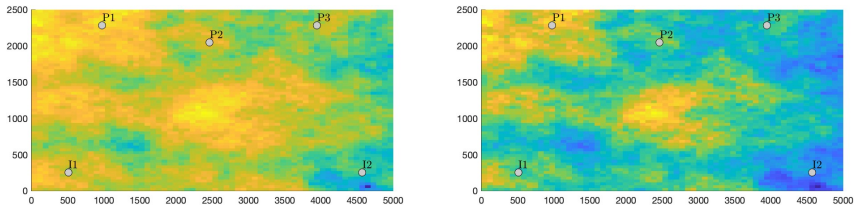
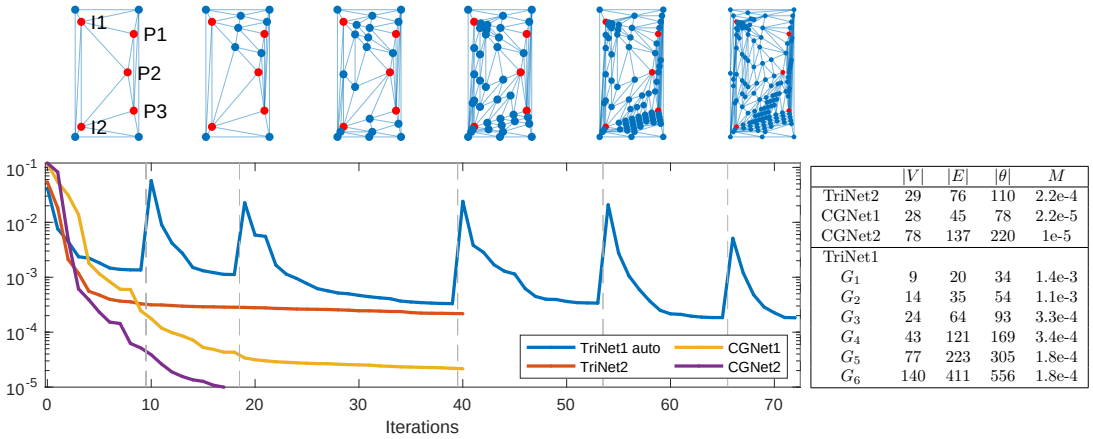


Figure 6.10: Logarithmic permeability (left) and porosity (right) fields for the heterogeneous version of the demo case.



	$\|V\|$	$\|E\|$	$\|\theta\|$	M
TriNet2	29	76	110	2.2e-4
CGNet1	28	45	78	2.2e-5
CGNet2	78	137	220	1e-5
TriNet1				
G_1	9	20	34	1.4e-3
G_2	14	35	54	1.1e-3
G_3	24	64	93	3.3e-4
G_4	43	121	169	3.4e-4
G_5	77	223	305	1.8e-4
G_6	140	411	556	1.8e-4

Figure 6.11: Mismatch reduction for the heterogeneous demo case, using the pure Delaunay TriNets and the two CGNets. Above the mismatch plot, we show the six graphs G_1-G_6 obtained in automatic refinement of TriNet1.

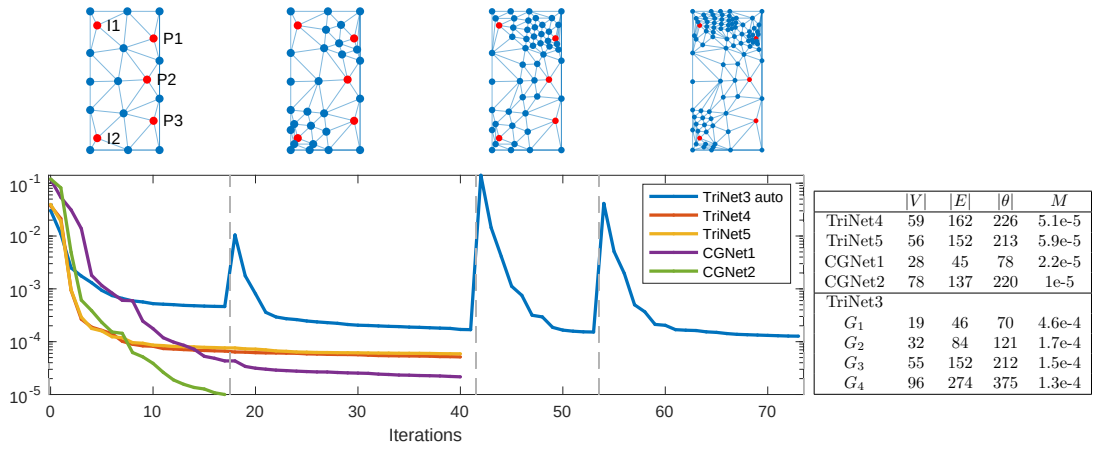
6.1.2 Heterogeneous case

The homogeneous porosity and permeability are now replaced with fields based on a normal and log-normal distribution, respectively, with the previous constant values as means, illustrated in Figure 6.10. Notice that the permeability and porosity are higher in the upper left part of the domain, close to the P1 producer. We run the same experiments as in the homogeneous case, first using the pure Delaunay graphs TriNet1 and TriNet2, and then using the DistMesh-based graphs TriNet3, TriNet4. We now also include the non-uniform TriNet5 based on distance to wells in the second experiment.

Figure 6.11 shows the mismatch reductions for the first experiment, and the corresponding code is found in `demoHeterogeneousCalibration.m`. Perhaps somewhat surprisingly, CGNet1 performs better now than in the homogeneous case. We can also observe that now, as opposed to the homogeneous case, it is the CGNets that have a worse starting point, but they are still able to beat the TriNets and end up at smaller final mismatch values. CGNet2 is again the better fine model, reaching its 10^{-5} stopping criterion in 18 iterations. This model should perhaps have been tuned even further, as it does not appear to have stagnated yet. On the other hand, we should not generally seek as small mismatch values as possible, as we then run the risk of overfitting the model.

The TriNet1 refinement clusters around I2 and P3. Despite claiming that a graph does not need to *look* good, refining this heavily around one well may be unfortunate. The initial graph TriNet1 reduces its mismatch by two orders of magnitude, after which it takes four refinements, and 271 more parameters, to get almost an additional order of magnitude reduction. In other words, the refinement algorithm does not do very well.

The results for the second experiment are shown in Figure 6.12, and the corresponding code is found in the script `demoHeterogeneousDistmeshCalibration.m`. The refinement now clusters around the injectors and producer P1, and we get little extra mismatch reduction with each refinement. This case also includes the a priori adapted TriNet5, which comes with higher resolution close to



	$\|V\|$	$\|E\|$	$\|\theta\|$	M
TriNet4	59	162	226	5.1e-5
TriNet5	56	152	213	5.9e-5
CGNet1	28	45	78	2.2e-5
CGNet2	78	137	220	1e-5
TriNet3				
G_1	19	46	70	4.6e-4
G_2	32	84	121	1.7e-4
G_3	55	152	212	1.5e-4
G_4	96	274	375	1.3e-4

Figure 6.12: Mismatch reduction for the heterogeneous demo case, using the DistMesh-based TriNets and the two CGNets. The top row shows the four graphs G_1-G_4 in the automatic refinement of TriNet3.

wells. This performs almost identically to the uniform TriNet4, so for this case, a uniform model seems to be adequate.

In all the automatic refinement results seen so far, we notice that the mismatch increases at each graph refinement. Recalling how the initial parameter guesses on updated graphs were set, this is not that surprising. There is no guarantee that the parameters that are optimal for one graph are optimal for the next, and it is even less likely that the initial guesses on the new nodes and edges are optimal. At the same time, part of the purpose of updating the graph is to perturb the parameters away from local minima. Attempting to construct the ideal parameter mapping between graphs may not be worthwhile. The jumps are not necessarily problematic in themselves, as long as the new graph achieves a closer match than the previous. Still, we could hope that the new graphs would take some advantage of the previous tuning. This may be more likely if smaller changes are made in each refinement. Testing a stricter bound on the number of parameters added should be considered.

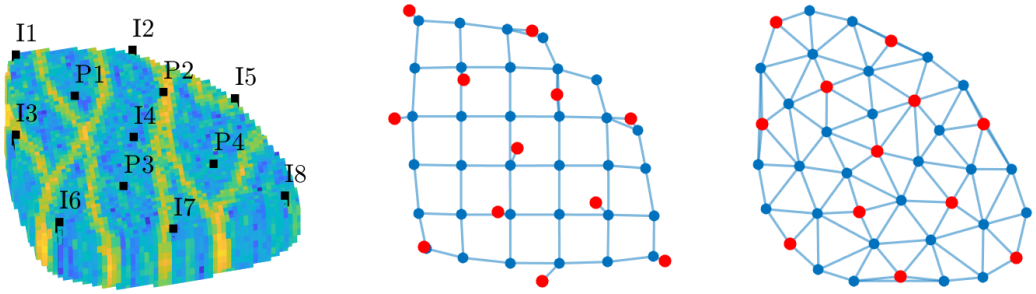
To maintain physically meaningful model outputs, we enforce conservation of the total pore volume at each graph refinement. This is done semi-locally, identifying the affected parts of the graph, and rescaling all pore volumes herein by the same factor. It should be considered to do a fully local rescaling instead, redistributing pore volumes at the triangle level. This could improve the parameter mapping between graphs, and thus reduce the jumps.

6.2 The Egg model

As our next example case, we use the Egg model [38], a synthetic model of a three-dimensional channelized reservoir. The original fine-scale model consists of 18 553 cells divided between 7 horizontal layers. Eight injectors and four producers are distributed across the egg-shaped domain, and we consider a water-flooding scenario in which the injectors operate at a constant water injection rate of 79.5 m^3/day and the producers at a constant bottom-hole pressure of 395 bar. The model has porosity 0.2 in all cells. The left plot of Figure 6.13 shows the log-scale permeability field. The water-oil system is governed by a relatively simple black-oil model with weak compressibility, cubic and quartic relative permeability curves, and 5 cP and 1 cP viscosities for the oil and water phases, respectively. The initial saturations are $S_w = 0.1$ and $S_o = 0.9$ in all cells.

6.2.1 Calibration

We will compare two coarse models, in particular a CGNet and a TriNet of comparable granularity. Both models are shown in Figure 6.13. The CGNet is based on a $6 \times 6 \times 1$ uniform partition with



Model	Nodes	Edges	Parameters	Description
CGNet	45	69	396	Uniform partition ($6 \times 6 \times 1$)
TriNet	40	101	393	DistMesh $\alpha_0 = 0.13$

Figure 6.13: Original model with log-scale permeability and wells, CGNet and TriNet for Egg.

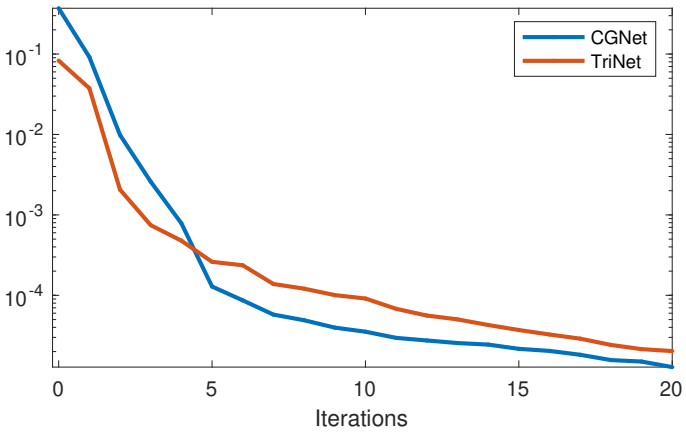


Figure 6.14: Mismatch reduction during calibration of the Egg CGNet and TriNet.

separate well blocks, as was illustrated in Figure 5.3. The resulting graph after culling has 45 nodes and 69 edges. The TriNet model is constructed using DistMesh, with a maximum edge factor of 0.13. This model has 40 nodes and 101 edges. As our tunable parameters, we include pore volumes, transmissibilities, well indices, and the six relative permeability scaling parameters. This gives a total of 396 parameters for CGNet and 393 for TriNet. The script for this case is `eggCalibration.m`.

Figure 6.14 shows the mismatch reduction during calibration of the two models. Both models calibrate quickly to a good accuracy. A successful calibration entails that the given model makes accurate predictions for the training data, but does not directly imply that the model is valid in a more general sense in that it will give accurate predictions for different data.

6.2.2 Testing predictive ability through control perturbations

In a practical setting, model calibration alone is not the main interest. A useful model should work well on scenarios different from, but not necessarily far from, those it was trained on. As a simple test of the model generality, we now perturb the controls and run a new fine- and coarse-scale simulation. We can again use the mismatch in well responses as our metric and repeat the experiment with increasingly large perturbations. The injection rates are perturbed by $y + 2\alpha(x - 0.5)y$, where $\alpha \in \{0.05, 0.1, 0.15, 0.25\}$, x is a random number between 0 and 1, and y the original value. The producer bottom hole pressures are similarly perturbed by $y - \beta(x - 0.2)y$, for $\beta \in \{5, 7.5, 10, 15\}$. We consider four levels, with increasing perturbations in both rates and bottom hole pressures.

Table 6.1: Mismatch of calibrated CGNet and TriNet for different levels of control perturbation.

Perturbation level	Mismatch	
	CGNet	TriNet
0	1.3e-5	2.0e-5
1	1.1e-4	3.7e-4
2	2.0e-4	7.9e-4
3	3.1e-4	1.4e-3
4	5.7e-4	2.6e-3

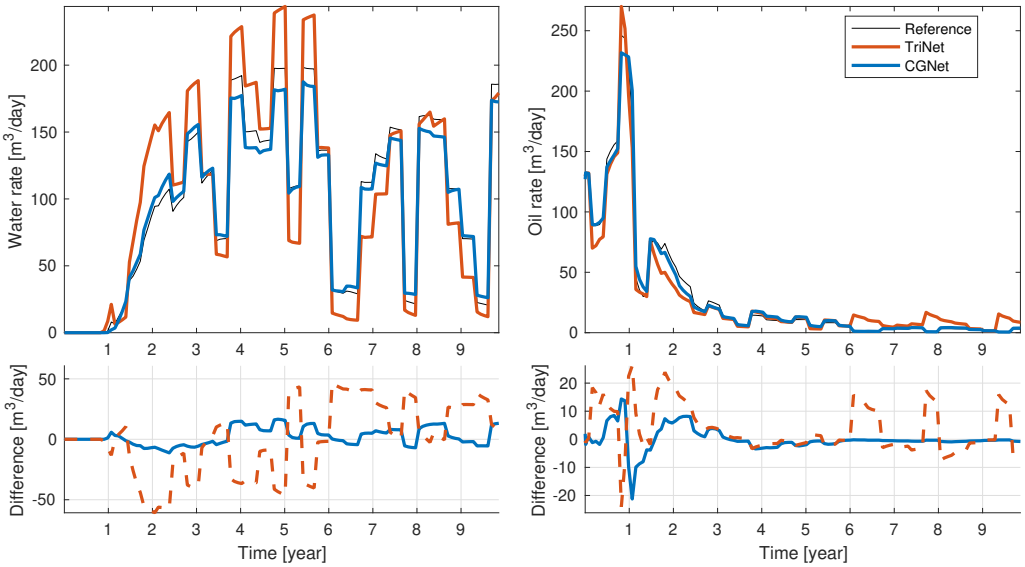


Figure 6.15: A comparison of the water and oil rates in producer well P1 predicted by the original Egg model ("reference" in the legend), the CGNet and TriNet when simulating with level 4 perturbation in the well controls.

Table 6.1 gives the mismatch of the tuned CGNet and TriNet for each level of perturbation. The CGNet appears to be more general, as increased perturbation levels give a smaller mismatch increase than for the TriNet.

Figure 6.15 shows production responses for producer P1, including the difference between predicted and true solutions, for the largest perturbation (level 4). There is generally a good agreement between the true and predicted curves, but it is also obvious that TriNet is visibly less accurate than CGNet. Figure 6.16 shows the resulting well responses in all the producers for all four levels of perturbation. Both these figures confirm the results in Table 6.1, with the TriNet giving slightly less accurate predictions than the CGNet. This could indicate that the CGNet model generalizes better.

It should be commented that there is one significant topological difference between the CGNet and TriNet in this case. For this specific CGNet, we added separate nodes for the wells. These nodes are themselves connected to a single different node. In the TriNet model, each well node was set as an integral part of the triangulation. Then, the well nodes are directly connected to multiple other nodes. This has the effect that CGNet is somewhat refined around the wells, offering a few extra tunable parameters there. It should be considered to test the effect of this extra well node more thoroughly. This can either be done by trying a CGNet model without it, or by including it also for TriNet.

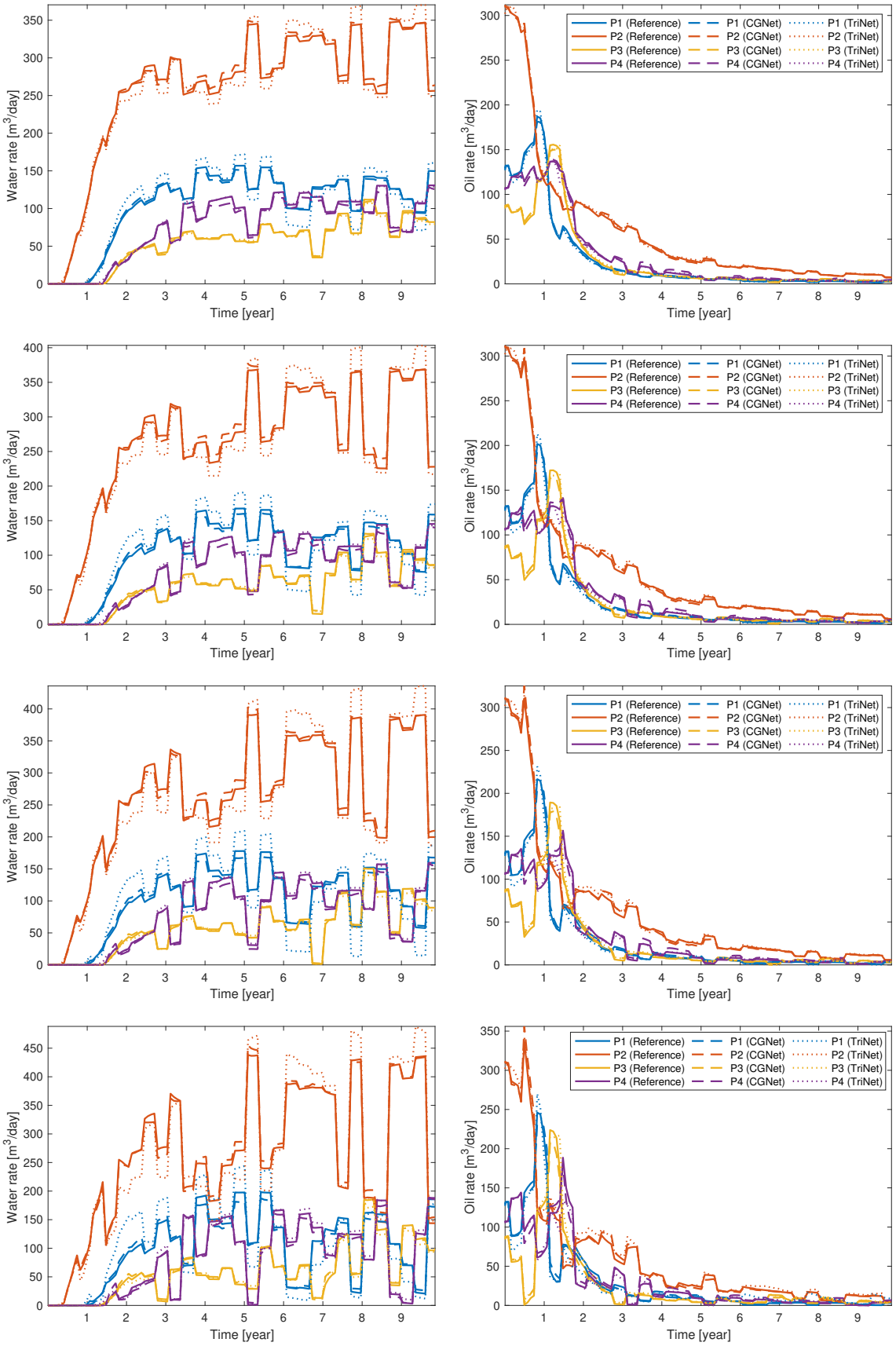
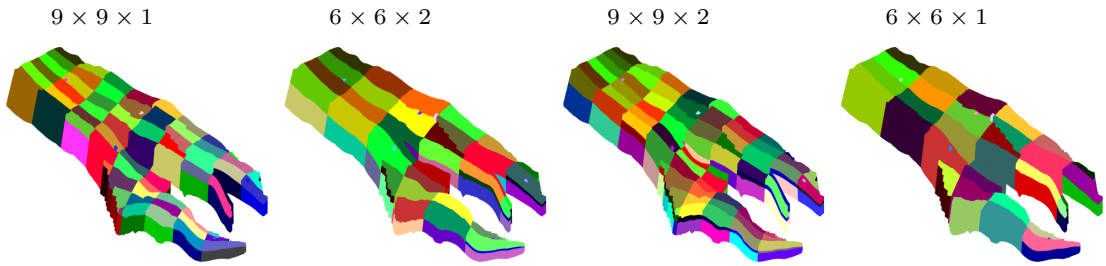


Figure 6.16: Production well responses predicted by the original Egg model (reference), CGNet and TriNet, for each of the four levels of control perturbation. Top to bottom row corresponds to increasing levels 1-4.



Model	Nodes	Edges	Parameters
CGNet $9 \times 9 \times 1$	112	202	330
CGNet $6 \times 6 \times 2$	97	221	334
CGNet $9 \times 9 \times 2$	184	448	648
CGNet $6 \times 6 \times 1$	59	95	170

Figure 6.17: The four partitions of the Norne model, where disconnected blocks have been split and we have added separate blocks for wells. The table shows their dimensions, including the total number of parameters when calibrating pore volumes, transmissibilities and well indices.

6.3 The Norne field

Next, we consider a simulation model of the Norne field in the Norwegian Sea. This model is available as an open data set via the Open Porous Media (OPM) initiative [44] and is one of the standard test cases in the MRST `test-suite` module. We consider a simplified case with immiscible waterflooding into a reservoir initially filled with oil ($S_o = 1$ in all cells). The reservoir has 5 injectors and 6 producers, where the injectors operate at a constant water injection rate of $16\,000\,\mathrm{m}^3/\mathrm{day}$, and the producers at a constant bottom-hole pressure of 100 bar. The original model is formulated on a $46 \times 112 \times 22$ corner-point grid with 44 915 active cells, displaying complex geological features such as faults and displaced layering.

We compare four CGNet models of varying lateral and vertical resolution, aiming to test the effect of multiple layers; see `norneCalibration.m`. Note that since this model has a layered structure, with two lateral layers that are completely disconnected in parts of the domain, a two-dimensional partition-based model will initially have disconnected blocks. This can cause trouble, for instance, in upscaling of transmissibilities, so we choose to split these blocks so that all blocks are connected. This has the effect that the models that initially were two-dimensional, end up having two layers in the disconnected parts. Moreover, as we did for the Egg model, we add separate blocks for wells. Figure 6.17 illustrates the resulting partitions.

Figure 6.18 shows the mismatch reduction during calibration. The four models perform very similarly, all reaching a small mismatch value in only ten iterations. Perhaps somewhat surprisingly, the coarsest $6 \times 6 \times 1$ model performs the best.

Note that we have not included triangulation-based models for Norne, since the non-convex reservoir domain would not allow the direct use of the TriNet framework. In particular, it does not allow the use of a convex hull to extract the boundary, and a triangulation would include non-physical connections through areas that are unavailable for flow. These challenges could of course be overcome with some manual hard coding. A suggested workaround is to construct the TriNet in the regular way, and then manually remove the edges we deem unreasonable. It would even be an interesting experiment to compare a dummy TriNet with non-physical connections, similar to CGNet in [10], to one which we have forced to abide the physical domain.

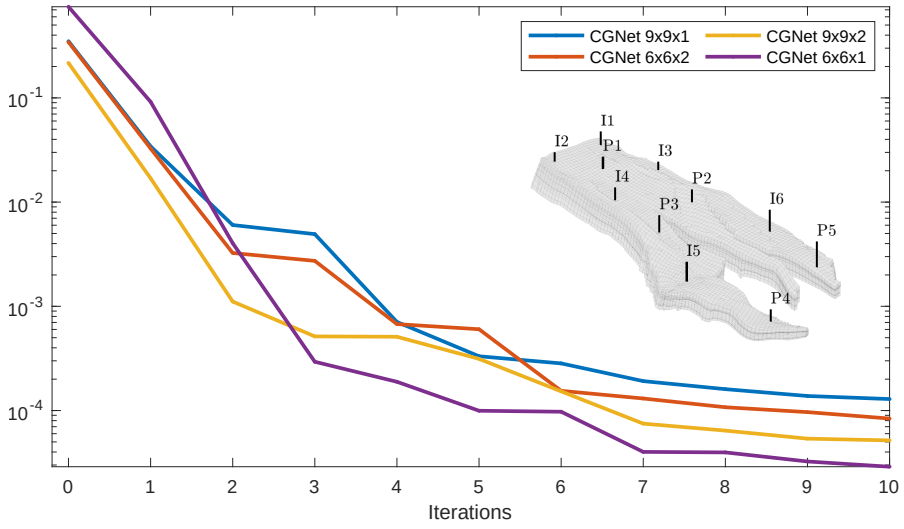


Figure 6.18: Mismatch reduction during calibration of the four CGNet models for the Norne case.

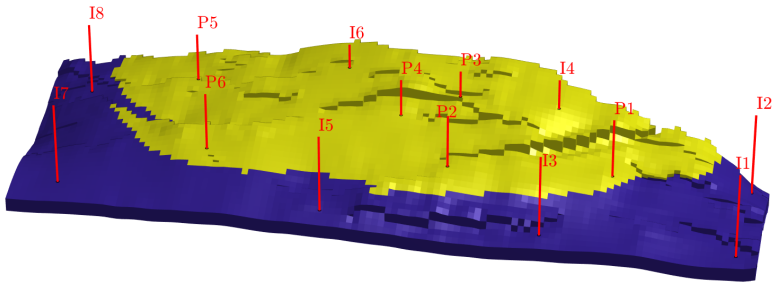


Figure 6.19: Initial water saturation for the SAIGUP model. The yellow part corresponds to $S_o = 1$, and the blue part to $S_w = 1$.

6.4 The SAIGUP model

In the models considered so far, the initial saturations have been homogeneous and, for the demo and Norne case, the reservoir was even initially filled with oil only ($S_o = 1$ in all cells). In those cases, it was trivial to map the initial saturation from the fine model onto the coarse graph-based models. However, this is not a particularly realistic model setup. For real reservoirs, the saturation is rarely homogeneous. Now, we will try a model where there is a clear separation between the oil and water part of the reservoir.

The model is from the SAIGUP project [45]. It is originally formulated on a grid consisting of 78 720 cells. Herein, we use a conceptual waterflooding scenario that has been introduced as part of the MRST development. The reservoir is operated with eight injectors and six producers, where the injector wells maintain constant water injection rates, and the producers operate at a fixed bottom-hole pressure of 200 bar. We simulate a schedule lasting for 30 years, divided into 118 time steps. A characteristic feature of this model, similarly to Brugge, is the concentration of oil in a cap, as illustrated in Figure 6.19. It could thus be interesting to try to calibrate a triangulation-based network and test the effect of tuning the initial saturation.

6.4.1 Calibrating the initial saturation

When constructing the TriNet, all nodes are assigned a globally averaged value for saturation. In this case, this gives $S_w = 0.5433$ and $S_o = 0.4567$. For models like SAIGUP, this is a poor guess which may make the model hard to calibrate. To test this, we try to construct a TriNet

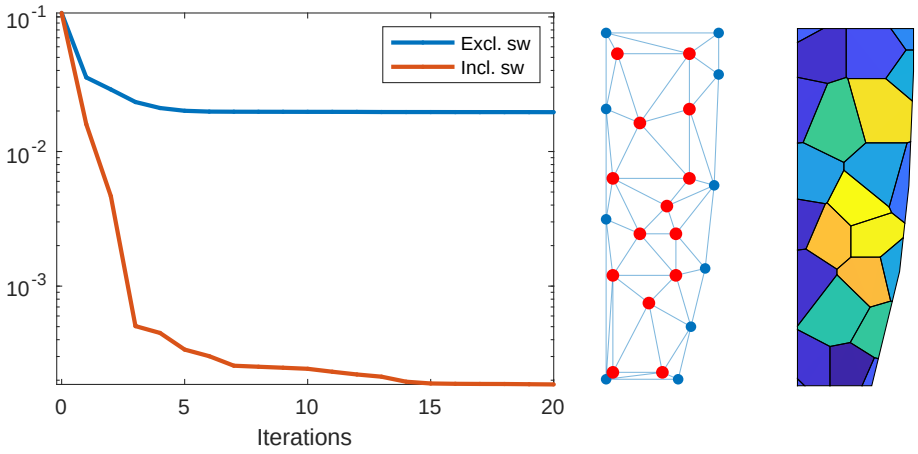


Figure 6.20: Mismatch (left) when calibrating the SAIGUP TriNet model (middle), with and without initial water saturation S_w^0 as a tunable parameter. The rightmost plot shows the calibrated initial oil saturation $(1 - S_w^0)$.

and calibrate it with and without including the initial water saturation S_w^0 as a tunable parameter. The TriNet model is constructed using DistMesh with a relative edge factor $\alpha_0 = 0.25$; see `saigupCalibration_sw.m`. This gives a coarse model with only 24 nodes and 59 edges. Pore volumes, transmissibilities and well indices are included in both cases, with a combined total of $24 + 59 + 14 = 97$ tunable parameters. When including the initial saturation S_w^0 as well, the total number of parameters is 121.

Figure 6.20 shows the TriNet model and the resulting mismatch reductions in the two cases. The results largely confirm our suspicions. The model using the poor initial saturation guess calibrates slowly and stagnates at a large mismatch value. On the other hand, the model allowing calibration of saturation calibrates both fast and to a small mismatch value. This experiment highlights the importance of providing a sufficiently accurate initial state. For partition-based models, we are able to do this to some extent a priori using upscaling. However, for triangulation-based models, we have not implemented a sophisticated mapping from the fine model. Then, calibrating the initial saturation seems to be a good and needed alternative.

6.4.2 Stacked TriNet

Next, we compare the performance of a stacked 2.5D TriNet model to a 2D TriNet of similar size. To construct the stacked model, we use two copies of the TriNet from the previous experiment, placed at the top and bottom of the domain. Here, we identify the bottom and top by finding the maximum and minimum z coordinate of the centroids in the original model. The resulting stacked model has a total of 48 nodes and 142 edges. To get a two-dimensional model of comparable granularity, we use DistMesh with maximum relative edge length $\alpha_0 = 0.1$, which gives a graph with 51 nodes and 133 edges. If we include pore volumes, transmissibilities, well indices and intial water saturation as tunable parameters, this gives a total of 266 parameters for the stacked model, and 249 for the two-dimensional model. Both models are calibrated for 20 iterations, and Figure 6.21 shows the model networks and resulting mismatch reduction. We also include a calibration of the stacked model including the gravitational parameter $g \cdot \texttt{grad}(z)$ as a tunable parameter. Then, the total number of parameters is 408. The script for this experiment is `saigupCalibration_stack.m`.

In this case, the two models perform similarly, both calibrating rapidly and to a small mismatch magnitude. The multi-layer topology does not give significantly improved results here. Moreover, adding the gravitational parameter in tuning does not improve the result, but instead gives a slightly worse calibration. However, this is perhaps not that surprising considering that this model has its variability mostly in the horizontal direction and is not particularly gravity-driven. We can expect that this parameter may be more important in cases where gravity has a larger effect, such as in models including gas.

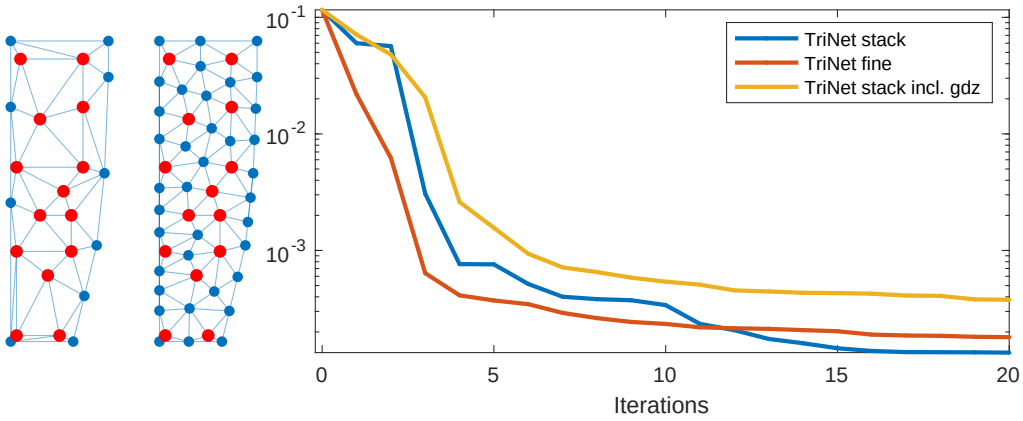


Figure 6.21: Stacked TriNet (top view) and finer two-dimensional TriNet for SAIGUP, and the resulting mismatch reduction when calibrating the models.

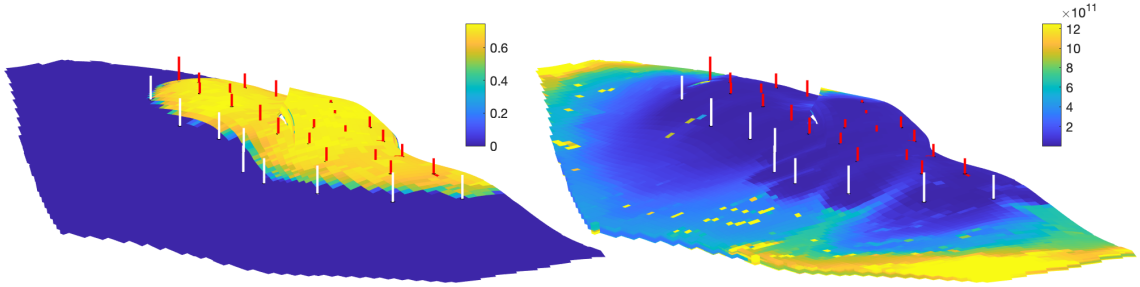
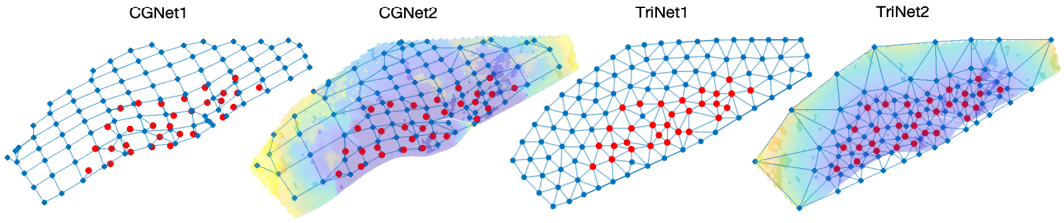


Figure 6.22: Initial oil saturation (left) and residence times (right) for the Brugge model. Injectors in white and producers in red.

6.5 The Brugge model

For our final example, we will use the Brugge benchmark model [39], which is an interesting case due to its concentration of oil and wells in a small part of the reservoir. This is illustrated in the left plot of Figure 6.22, which shows the initial oil saturation. The strong heterogeneity encourages the introduction of non-uniform models. Moreover, the initial saturation resembles that of the SAIGUP model, for which we demonstrated the importance of calibrating the initial saturation (Section 6.4.1).

The original Brugge grid has 43 474 cells. The reservoir has 10 injectors and 20 producers, and we simulate for a duration of 10 years, divided into 130 time steps.



Model	Nodes	Edges	Parameters	Description
CGNet1	122	205	1211	$13 \times 8 \times 1$ uniform partition + well blocks.
CGNet2	125	234	1264	$9 \times 5 \times 1$ uniform partition + split high-flow blocks + well blocks.
TriNet1	111	305	1223	DistMesh, $\alpha_0 = 0.055$
TriNet2	113	321	1255	DistMesh, well-adapted ($\alpha_0 = 0.05$., $h_{min} = 0.075$, $h_{max} = 0.5$)
TriNet3	47	125	531	DistMesh, $\alpha_0 = 0.1$.

Figure 6.23: The suggested network models for Brugge.

The models we will compare are shown in Figure 6.23, which also gives their dimensions and short descriptions. The first model, CGNet1, is based on a uniform $13 \times 8 \times 1$ partition, with separate nodes for the wells. Next, we have the flow-adapted CGNet2. This is constructed using residence times, as described in subsection 5.1.4. The right plot in Figure 6.22 shows the computed residence-time field. To set up the non-uniform model, we start from a $9 \times 5 \times 1$ coarse uniform partition. The residence-time field originally holds one value per fine cell, which we map to the coarse blocks by simply summing up to get our indicator. Then, we select the blocks where the indicator is above the median value, and split these blocks in four via a $18 \times 10 \times 1$ partition. Finally, we add separate well nodes.

We also include two triangulation-based models. The first, TriNet1, is constructed from DistMesh with initial edge length chosen to give a comparable number of parameters to the CGNets. The other, TriNet2, is constructed using our qualified guess flow indicator that measures distance to wells, as explained in Section 5.2.4. Below the model plots in Figure 6.23 is a table stating the dimensions of each model. We note that the models are of comparable size.

Finally, we include the coarse TriNet3, which will be the initial graph in an automatic refinement. This model has only 47 nodes initially, and less than half as many parameters as the four other models.

6.5.1 Calibration

We now calibrate each of the four fine models for 20 iterations. As our tunable parameters, we will use pore volumes, transmissibilities, well indices, six relative permeability scaling parameters (exponent and endpoint coordinates for each of the phases), and initial water saturation. This gives approximately 1200 parameter values in total for each model. Moreover, we run an automatic refinement process for TriNet3, refining 10% of the triangles at each graph update. The code for this experiment is found in `bruggeCalibration.m`.

For the CGNets, we consider two cases. In the first, we use the upscaled information from the fine model, including, e.g., a decent initial guess for saturations. To get a more fair comparison with the purely data-driven TriNets, we also include a CGNet tuning where we discard this information, using the same average-based initial guess as for TriNet.

Figure 6.24 shows the mismatch during calibration of the models. The CGNets using upscaled information calibrate better than the TriNets, and the difference between the uniform and non-uniform versions is small, with the flow-adapted model being slightly better. The non-upscaled

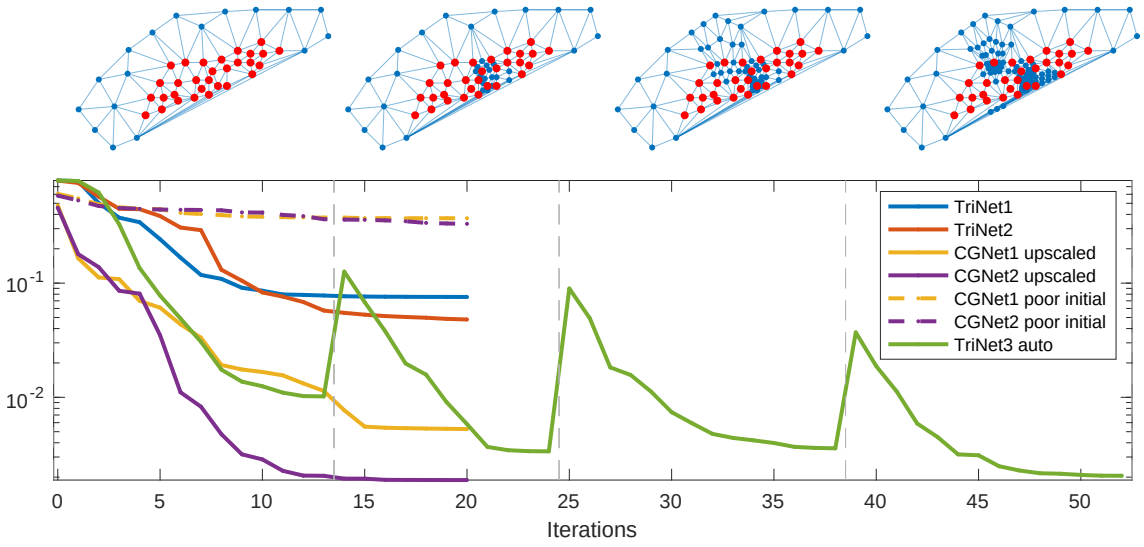


Figure 6.24: Calibration results for Brugge. Above the mismatch plot are the four graphs G_1-G_4 in automatic refinement of TriNet3.

Table 6.2: Model dimensions and final mismatch value in calibration of the different network models for Brugge.

	$\|V\|$	$\|E\|$	$\|\theta\|$	M
TriNet1	111	305	1223	7.6e-2
TriNet2	113	321	1255	4.8e-2
CGNet1 (upscaled)	122	205	1211	5.3e-3
CGNet2 (upscaled)	125	234	1264	8.9e-3
CGNet1 (poor initial)	122	205	1211	0.37
CGNet2 (poor initial)	125	234	1264	0.33
TriNet3 auto				
G_1	47	125	531	1.0e-2
G_2	63	173	707	3.4e-3
G_3	86	241	959	3.6e-3
G_4	120	343	1333	2.1e-3

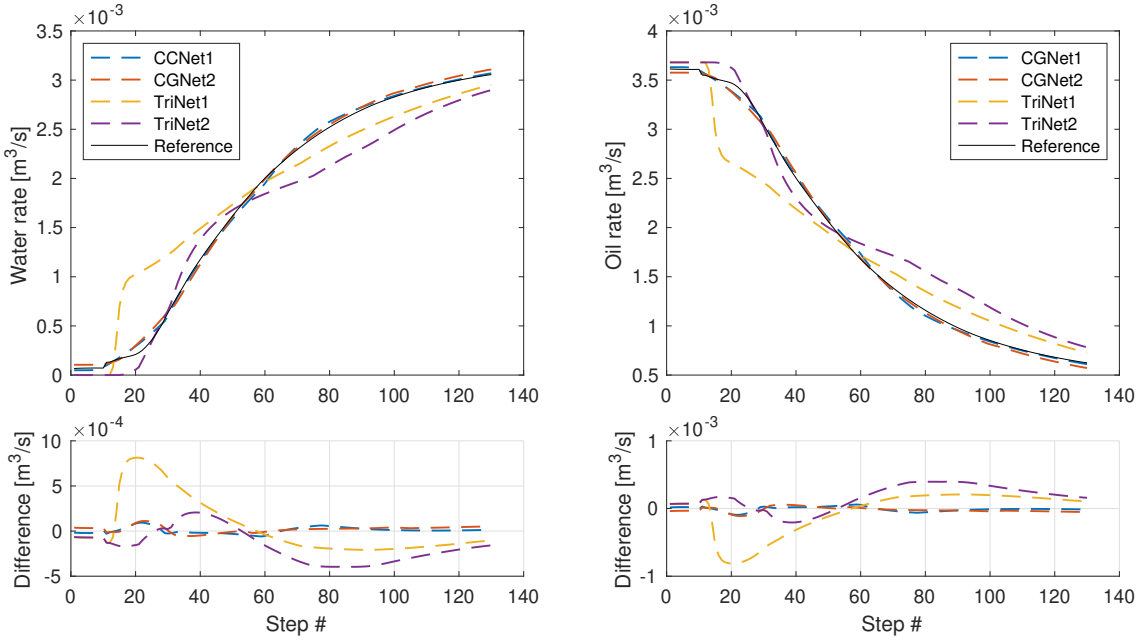


Figure 6.25: Water and oil rate in producer P5 predicted by the calibrated network models, compared to the fine-scale reference. The CGNets here are the upscaled versions.

CGNets work very poorly in this case, and significantly worse than the TriNets. The TriNet3 automatic refinement works rather well in this case. The mismatch is quite small already on the first graph, and for the fourth graph, which is comparable in granularity to the other models, the mismatch is as good as for the upscaled CGNet2. Although the automatic refinement does not give a better final result than the upscaled CGNet2, it has the advantage of not depending on the user's understanding and chosen model configuration.

The mismatch reduction should correspond to a good agreement between the fine-scale reference and calibrated well responses. Figure 6.25 illustrates this for the production well $P5$. The calibrated curves for the upscaled CGNets are qualitatively very close to the reference, whereas the TriNet curves are visibly further away.

In this case, the TriNet models and CGNets with poor initial guess, use an initial guess $S_w = 0.8577$, $S_o = 0.1423$ for the initial saturations, which comes from averaging over the original model. The water saturation, and thus implicitly the oil saturation, are then calibrated within a $[0, 1]$ interval. We can investigate which values they are calibrated to. Figure 6.26 shows the tuned oil saturation, plotted on the dual Voronoi grids. The saturations are interestingly qualitatively similar to those in the original model (Figure 6.22).

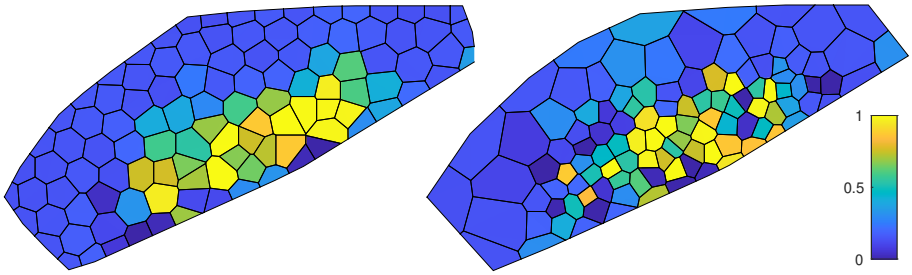


Figure 6.26: Calibrated initial oil saturation for TriNet1 and TriNet2, plotted on their dual Voronoi grids.

Table 6.3: Bounds used in control optimization for Brugge.

	Lower	Upper
Injector rate [m^3/day]	10	1000
Producer liquid rate [m^3/day]	10	500
Injector BHP [bar]	160	180
Producer BHP [bar]	50	120

We remark that when calibrating the TriNet models with initial saturations as purely tunable parameters, an illuminating problem occured. Letting the initial saturations vary freely within the unit interval enabled the optimizer to pick values in such a way that the simulation was inconsistent and the optimization crashed after a few iterations. This was due to the Brugge model having capillary pressures, in which case it is problematic to tune the initial water saturation to values below the residual water saturation. We therefore ended up removing the capillary pressures from all models. One could even argue that in a data-driven setting, this information would not be available, so calibrating models without imposing such physical effects a priori may be more realistic.

6.5.2 Control optimization

So far, we have mostly discussed efficiency and accuracy in calibrating the network models, and briefly tested their generality and predictive abilities. As a final experiment, we will demonstrate one of the motivating applications of the coarse models, namely control optimization. We choose the CGNet2 model (upscaled version), which demonstrated the best calibration result. The coarse nature of CGNet2 makes it a well-suited proxy model.

Control optimization in this case involves finding well controls that maximize the net-present-value (NPV) of the reservoir. In the NPV function, we use an oil revenue of 50 USD per STB (stock tank barrel), water injection cost of 3 USD per STB, water production cost of 3 USD per STB, and a yearly discount rate of 10%. Moreover, we set some bounds on the controls, given in Table 6.3.

The resulting NPVs are given in Table 6.4, and Figure 6.27 shows the obtained optimal controls. First, observe that the predicted NPV on the original schedule is similar for the original and reduced CGNet2 model. This confirms that the network model does in fact represent the original model, as the small mismatch value in calibration indicates. The NPV using optimized controls is

Table 6.4: Net-present-value for Brugge, with original and optimized controls, for the fine-scale and CGNet2 models.

Model	Controls	NPV (10^{10} \$)
CGNet2	Original	1.840
CGNet2	Optimized	2.404
Fine	Original	1.837
Fine	Optimized	2.117

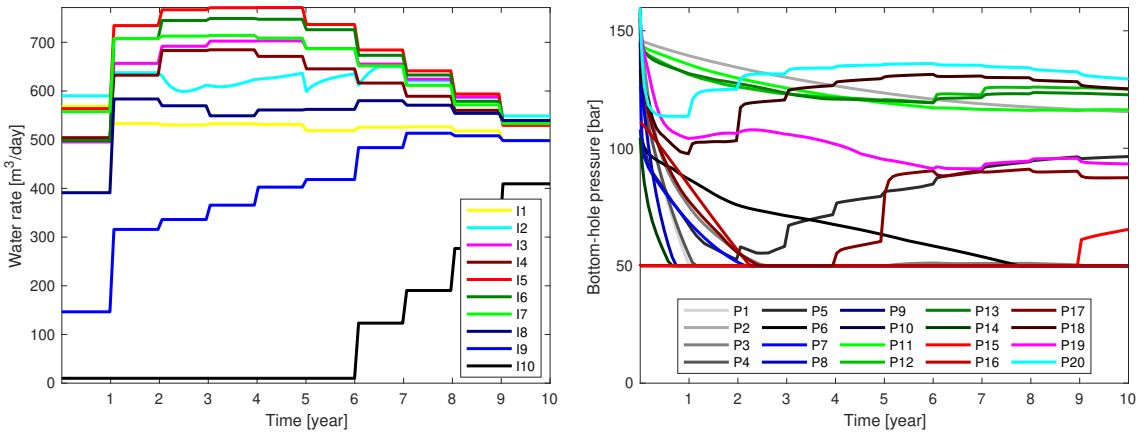


Figure 6.27: Optimized well controls for Brugge, using the calibrated CGNet2 as a proxy.

more deviant, and significantly larger for CGNet2. This could indicate that the controls that are optimal for CGNet2 are not optimal for the original model. However, we do obtain a significant NPV increase (2.8 bill. USD) compared to the original controls, so our optimization does have an effect.

The network models are trained on limited data, and their generality is thus likely equally limited. In the control optimization setting, we may end up with optimized controls that deviate significantly from those we started with. In other words, the controls give new reservoir responses that are far from the training data, and it can not be guaranteed that the calibrated models give accurate predictions. To circumvent this, it is possible to run a retraining; see [11]. Once the controls are optimized using the calibrated network model, a new fine-scale simulation is run using the obtained optimal controls to produce a new set of training data. The coarse model can then be retrained, a new control optimization can be run, and so on.

Chapter 7

Conclusions and outlook

We have demonstrated two main approaches to construct coarse, graph-based reservoir models. The models offer a richer graph topology and more tunable parameters than pure interwell network models and are easy and fast to evaluate in a standard finite-volume reservoir simulator. As we keep the underlying mathematical model, we prevent predicting unphysical states, giving an advantage over purely data-driven machine-learning models. Moreover, the coarse nature of the models allows computer-intensive applications like parameter calibration and production optimization.

The numerical examples have demonstrated that even very coarse models with less than a hundred nodes can be effectively calibrated to give predictions closely resembling those of the original fine-scale model. Perturbations of well controls also indicated that the models were able to give quite accurate predictions for data outside the range they were trained on, with the CGNet appearing to be more general than TriNet.

The initialization of both initial states and parameters had a significant effect on calibration. We saw that a reasonable initial guess for pore volumes was crucial for good calibration. Moreover, the SAIGUP case demonstrated the importance of an accurate initial saturation. For a partition-based network model, reasonably accurate values can easily be obtained via upscaling. If we do not have such an immediate mapping, e.g., like we assumed was the case for the triangulation-based models herein, adding initial saturation as a tunable parameter appears to work well. The saturation was even calibrated to at least qualitatively resemble that in the original model. It could also be worth an attempt to map more information from the fine model to a TriNet model, including initial saturations. This can be based on the dual Voronoi grid and some upscaling procedure, similar to how it was done for partition-based CGNets. Pore volumes, for instance, could be scaled according to the size of the corresponding Voronoi cell and initial saturations could just be volume-averaged.

Including the gravitational effect parameter in calibration did not improve the results in the examples tested here. This should be tested on cases where gravitation plays a more important role, such as models including gas, e.g., a carbon storage simulation.

Different methods for constructing non-uniform models were compared, including a priori flow-adapted models based on well proximity and residence times, and an automatic refinement procedure. These gave variable results, generally not significantly better than a uniform model. The automatic refinement algorithm has several weaknesses that should be adressed to improve results, including the parameter mapping between graphs. It may, for instance, be worth testing a fully local redistribution of pore volumes at the triangle level. Still, this approach offers the advantage of starting from a very coarse model, and not being dependent on the user choosing a suitable model topology a priori.

In this thesis, automatic graph refinement was only tested for triangulation-based models. A similar procedure can be applied to partition-based models; see [11]. The algorithm could even be expanded to go both ways. Using nested partitions, we can easily go both finer and coarser from a given graph.

With their limited degrees of freedom, the models enable speedy evaluations, and are well-suited as proxies in applications like production optimization. This was demonstrated for the Brugge model. Using a calibrated coarse CGNet, the controls were successfully adjusted to significantly improve the net-present-value objective function. An iterative retraining, as suggested in [11], could circumvent the lacking generalization of the calibrated network models, when the optimized controls end up being far from the initial training data, thus giving a more accurate optimization.

One of the main motivations behind the coarse, graph-based models were in fact their speedy simulations, and thus suitability in an optimization context. In the examples in this thesis, all parameters have had the same granularity as the network. For example, if there are 100 nodes in the graph, we calibrate 100 pore volume values. However, it is possible to reduce the parameter space without decreasing the granularity of the graph itself. We can use different resolutions for different parameters, for example by defining regions where the parameter value is constant. Considering the relative permeability parameters, of which there are six (for a Corey-type two-phase fluid model), it is evident that there are significant costs in calibrating these at every single node. It should be investigated if calibrating one or a few values instead gives good enough results. More parameters may require more training data, so a smaller parameter space may give a model that is easier to calibrate and less likely to be overfitted to the training data. Moreover, we have always calibrated all parameters simultaneously. It should also be investigated if it is better to split the calibration, e.g., first calibrating pore volumes, transmissibilities and well indices, and then fluid/permeability parameters.

Despite some of the weaknesses identified herein, we conclude that the methods hold great promise as a hybrid approach to reservoir modeling, utilizing data without discarding our knowledge of the flow physics. Finally, we remark that the models and principles used in this project can be readily applied to other contexts, and are most immediately extended to geothermal energy and carbon storage applications.

References

[1] S. Ghassemzadeh et al. «A data-driven reservoir simulation for natural gas reservoirs». In: *Neural Computing and Applications* 33.18 (2021), pp. 11777–11798. DOI: 10.1007/s00521-021-05886-y.

[2] L. A. N. Costa, C. Maschio, and D. José Schiozer. «Application of artificial neural networks in a history matching process». In: *Journal of Petroleum Science and Engineering*. Neural network applications to reservoirs: Physics-based models and data models 123 (2014), pp. 30–45. DOI: 10.1016/j.petrol.2014.06.004.

[3] M. M. Almajid and M. O. Abu-Alsaud. «Prediction of Fluid Flow in Porous Media using Physics Informed Neural Networks». In: *Abu Dhabi International Petroleum Exhibition & Conference*. Abu Dhabi, UAE, Nov. 2020. DOI: 10.2118/203033-MS.

[4] C. G. Fraces and H. Tchelepi. «Physics Informed Deep Learning for Flow and Transport in Porous Media». In: *SPE Reservoir Simulation Conference*. On-Demand, Oct. 2021. DOI: 10.2118/203934-MS.

[5] G. Ren et al. «Implementation of Physics-Based Data-Driven Models With a Commercial Simulator». In: *SPE Reservoir Simulation Conference* (Galveston, Texas, USA). Society of Petroleum Engineers, Apr. 2019. DOI: 10.2118/193855-MS.

[6] G. Lutidze. «StellNet – physics-based data-driven general model for closed-loop reservoir management». PhD thesis. The University of Tulsa, 2018.

[7] H. Zhao et al. «INSIM: A Data-Driven Model for History Matching and Prediction for Waterflooding Monitoring and Management with a Field Application». In: *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers, 2015.

[8] Z. Guo, A. C. Reynolds, and H. Zhao. «A Physics-Based Data-Driven Model for History Matching, Prediction, and Characterization of Waterflooding Performance». In: *SPE Journal* 23.02 (2018), pp. 367–395. DOI: 10.2118/182660-PA.

[9] Z. Guo, A. C. Reynolds, and H. Zhao. «Waterflooding optimization with the INSIM-FT data-driven model». In: *Computational Geosciences* 22.3 (2018), pp. 745–761. DOI: 10.1007/s10596-018-9723-y.

[10] K.-A. Lie and S. Krogstad. «Data-Driven Modelling with Coarse-Grid Network Models». In: *ECMOR 2022*. The Hague, Netherlands / Online: European Association of Geoscientists & Engineers, 2022. DOI: 10.3997/2214-4609.202244065.

[11] S. Krogstad, Ø. A. Klemetsdal, and K.-A. Lie. «Efficient Adaptation and Calibration of Adjoint-Based Reduced-Order Coarse-Grid Network Models». In: *SPE Reservoir Simulation Conference*. Galveston, Texas, USA: Society of Petroleum Engineers, Mar. 2023. DOI: 10.2118/212207-MS.

[12] K.-A. Lie and S. Krogstad. «Comparison of two different types of reduced graph-based reservoir models: Interwell networks (GPSNet) versus aggregated coarse-grid networks (CGNet)». In: *Geoenergy Science and Engineering* 221 (2023), p. 111266. DOI: 10.1016/j.petrol.2022.111266.

[13] V. L. Hauge, K.-A. Lie, and J. R. Natvig. «Flow-based coarsening for multiscale simulation of transport in porous media». In: *Computational Geosciences* 16.2 (2012), pp. 391–408. DOI: 10.1007/s10596-011-9230-x.

[14] I. S. Devold. «Graph-based Methods for Data-driven and Reduced-order Reservoir Modeling». Specialization Project (unpublished work). NTNU Trondheim, Dec. 2022.

[15] A. Yapparova, S. Matthäi, and T. Driesner. «Realistic simulation of an aquifer thermal energy storage: Effects of injection temperature, well placement and groundwater flow». In: *Energy* 76 (2014), pp. 1011–1018. DOI: 10.1016/j.energy.2014.09.018.

[16] M. Collignon, Ø. S. Klemetsdal, and O. Møyner. «Simulation of Geothermal Systems Using MRST». In: *Advanced Modeling with the MATLAB Reservoir Simulation Toolbox*. Ed. by K.-A. Lie and O. Møyner. Cambridge University Press, 2021, pp. 491–514. DOI: 10.1017/9781009019781.018.

[17] Ø. Klemetsdal et al. «Modeling and Optimization of Shallow Geothermal Heat Storage». In: *ECMOR 2022*. European Association of Geoscientists & Engineers, Sept. 2022. DOI: 10.3997/2214-4609.202244109.

[18] J. M. Nordbotten and M. A. Celia. *Geological Storage of CO2: Modeling Approaches for Large-Scale Simulation*. John Wiley & Sons, 2011.

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

[20] Ø. S. Klemetsdal. «Efficient Solvers for Field-Scale Simulation of Flow and Transport in Porous Media». PhD thesis. NTNU, 2019.

[21] D. W. Peaceman. «Interpretation of Well-Block Pressures in Numerical Reservoir Simulation With Nonsquare Grid Blocks and Anisotropic Permeability». In: *Society of Petroleum Engineers Journal* 23.03 (1983), pp. 531–543. DOI: 10.2118/10528-PA.

[22] SINTEF Computational Geosciences. *The MATLAB Reservoir Simulation Toolbox (MRST)*. Version 2023b. 2023. URL: https://www.sintef.no/projectweb/mrst/.

[23] A. Quarteroni. *Numerical Models for Differential Problems*. 2nd ed. Milano: Springer Milan, 2014. DOI: 10.1007/978-88-470-5522-3.

[24] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. 2nd ed. Vol. 37. Texts in Applied Mathematics. Berlin, Germany: Springer, 2007. DOI: 10.1007/b98885.

[25] K.-A. Lie and O. Møyner, eds. *Advanced Modeling with the MATLAB Reservoir Simulation Toolbox*. 1st ed. Cambridge University Press, Nov. 2021. DOI: 10.1017/9781009019781.

[26] K.-A. Lie et al. «Open-source MATLAB implementation of consistent discretisations on complex grids». In: *Computational Geosciences* 16.2 (2012), pp. 297–322. DOI: 10.1007/s10596-011-9244-4.

[27] S. Krogstad et al. «MRST-AD – an Open-Source Framework for Rapid Prototyping and Evaluation of Reservoir Simulation Problems». In: *SPE Reservoir Simulation Symposium*. Houston, Texas, USA: Society of Petroleum Engineers, 2015.

[28] O. Møyner. «Faster Simulation with Optimized Automatic Differentiation and Compiled Linear Solvers». In: *Advanced Modeling with the MATLAB Reservoir Simulation Toolbox*. Ed. by K.-A. Lie and O. Møyner. Cambridge University Press, 2021, pp. 200–254. DOI: 10.1017/9781009019781.011.

[29] D. S. Oliver and Y. Chen. «Recent progress on reservoir history matching: a review». In: *Computational Geosciences* 15.1 (2011), pp. 185–221. DOI: 10.1007/s10596-010-9194-2.

[30] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2nd ed. New York, NY, USA: Springer, 2006. DOI: 10.1007/978-0-387-40065-5.

[31] K.-A. Lie and S. Krogstad. «Data-driven Modelling With Coarse-grid Network Models». In: *European Conference on the Mathematics of Geological Reservoirs 2022* (The Hague, Netherlands). EAGE, Sept. 2022. DOI: 10.3997/2214-4609.202244065.

[32] S. Krogstad. «Introduction to adjoint methods for time-dependent systems». Presented at the Geilo Winter School (online). 2022.

[33] R. March et al. «A Unified Framework for Flow Simulation in Fractured Reservoirs». In: *Advanced Modeling with the MATLAB Reservoir Simulation Toolbox*. Ed. by K.-A. Lie and O. Møyner. Cambridge University Press, Nov. 2021, pp. 454–490. DOI: 10.1017/9781009019781.017.

[34] A. Kiærr et al. «Evaluation of A Data-Driven Flow Network Model (FlowNet) for Reservoir Prediction and Optimization». In: *ECMOR XVII – 17th European Conference on the Mathematics of Oil Recovery*. European Association of Geoscientists & Engineers, 2020. DOI: 10.3997/2214-4609.202035099.

[35] O. Leeuwenburgh et al. «Application of Coupled Flow Network and Machine Learning Models for Data-Driven Forecasting of Reservoir Souring». In: *ECMOR 2022*. European Association of Geoscientists & Engineers, Sept. 2022. DOI: 10.3997/2214-4609.202244046.

[36] Z. Guo, S. Sankaran, and W. Sun. «Reservoir Modeling, History Matching, and Characterization with a Reservoir Graph Network Model». In: *SPE Reservoir Evaluation & Engineering* (2023), pp. 1–13. DOI: 10.2118/209337-PA.

[37] S. Nnozuba. «A dual mesh and network model for closed-loop reservoir management». Master's thesis. The University of Tulsa, 2020.

[38] 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: 10.1002/gdj3.21.

[39] E. Peters et al. «Extended Brugge benchmark case for history matching and water flooding optimization». In: *Computers & Geosciences*. Benchmark problems, datasets and methodologies for the computational geosciences 50 (Jan. 2013), pp. 16–24. DOI: 10.1016/j.cageo.2012.07.018.

[40] P.-O. Persson and G. Strang. «A Simple Mesh Generator in MATLAB». In: *SIAM Review* 46.2 (2004), pp. 329–345. DOI: 10.1137/S0036144503429121.

[41] R. L. Berge, Ø. S. Klemetsdal, and K.-A. Lie. «Unstructured Voronoi grids conforming to lower dimensional objects». In: *Computational Geosciences* 23.1 (2019), pp. 169–188. DOI: 10.1007/s10596-018-9790-0.

[42] I. S. Devold. *Graph-based methods*. 2023. URL: https://bitbucket.org/ingvilddevold/graph-based-methods/.

[43] O. Møyner et al. *sintefmath/JutulDarcy.jl: v0.2.6*. Version v0.2.6. June 2023. DOI: 10.5281/zenodo.8013240. URL: https://github.com/sintefmath/JutulDarcy.jl.

[44] The Open Porous Media (OPM) Initiative. *The Norne dataset*. URL: https://github.com/OPM/opm-data.

[45] T. Manzocchi et al. «Sensitivity of the impact of geological uncertainty on production from faulted and unfaulted shallow-marine oil reservoirs: objectives and methods». In: *Petroleum Geoscience* 14.1 (2008), pp. 3–15. DOI: 10.1144/1354-079307-790.

[46] R. Diestel. *Graph Theory*. 5th ed. Vol. 173. Graduate Texts in Mathematics. Berlin, Germany: Springer, 2017. DOI: 10.1007/978-3-662-53622-3.

Appendix A

Basic Graph Theory*

In its most general form, a graph is a highly versatile tool that can be used to represent a wide range of systems. Abstractly speaking, we could say that a graph models pairwise relations between objects. We are free to choose both what the objects represent and what a relation between two objects means. The generality of graphs becomes apparent in their many of applications, including classical problems like shortest path or maximum flow in a network, the chemical model of a molecule, and even sociograms in social sciences.

The purpose of this chapter is to introduce some fundamental definitions and notation in graph theory. Different matrix representations for use in a programming setting are presented, and a brief introduction to MATLAB's graph framework is given.

A.1 Definitions

A *graph* is a pair of sets $G = (V, E)$, where the elements of V are called nodes or vertices, and the elements of E are called edges [46]. An edge connects two nodes, such that $E \subseteq [V]^2$. We denote an edge $e \in E$ between the nodes $v_1, v_2 \in V$ by $e = (v_1, v_2)$, and call v_1 and v_2 *incident* with the edge e. For directed graphs, the edge e goes *from* v_1 *to* v_2. For our purpose, undirected graphs are sufficient, in which case the edges are bidirectional and direction does not play a role. Figure A.1 illustrates an undirected graph.

In many applications, each edge and/or node in the graph is assigned a numerical value, a *weight*, which can represent some additional information. Figure A.2 illustrates a simple example of a

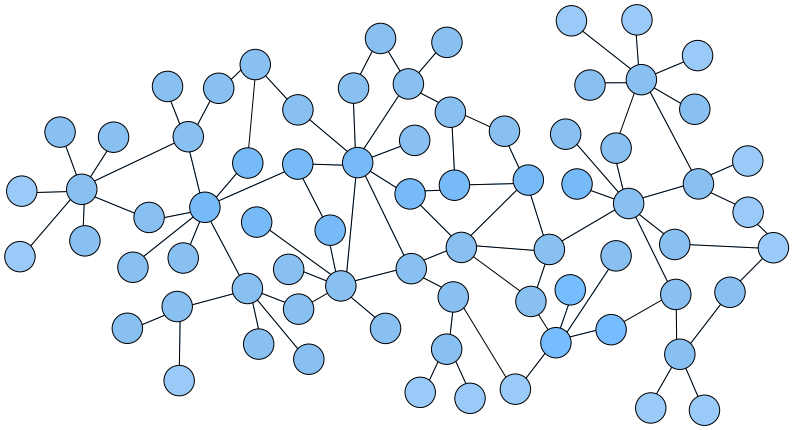


Figure A.1: A generic undirected graph. The blue discs represent the nodes, which are connected by the edges.

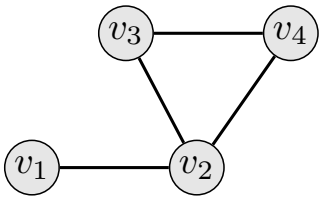


Figure A.2: A simple graph.

graph $G = (V, E)$ with nodes $V = \{v_1, \ldots, v_4\}$ and edges $E = \{(v_1, v_2), (v_2, v_3), (v_2, v_4), (v_3, v_4)\}$.

We call two nodes *adjacent* if there is an edge between them. Moreover, we let the neighborhood $N(v)$ of a node v denote the set of nodes adjacent to v. Similarly, two edges are adjacent if they have an end-node in common, and we let the neighborhood $N(e)$ of e be the set of all edges adjacent to e.

A.2 Matrix representation

In a programming setting, there are a number of ways to represent a graph. Let $G = (V, E)$ be a graph with nodes $V = \{v_1, \ldots, v_n\}$ and edges $E = \{e_1, \ldots, e_m\}$.

The *adjacency matrix* $A = (a_{ij})_{n \times n}$ has entries

$$a_{ij} \coloneqq \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

For the graph in Figure A.2, the adjacency matrix is

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

Having an adjacency matrix representation, it is trivial to find the neighborhood of a node v_i. We simply find the nonzero entries in row/column i, $N(v_i) = \{j \mid a_{ij} = a_{ji} = 1\}$. Moreover, the adjacency matrix provides a simple way of finding the *extended* neighborhood, $\hat{N}(v_i)$, the neighbors of the neighbors. The matrix $A^T A = (\hat{a}_{ij})$ will have nonzero entries on the diagonal, and for nodes i, j that share a neighbor. For undirected graphs, $A^T = A$, so we can skip the transpose and look at A^2 instead. Then, $\hat{N}(v_i) = \{j \mid \hat{a}_{ij} = 1, j \neq i\}$.

The *incidence matrix* $B = (b_{ij})_{n \times m}$ has elements

$$b_{ij} \coloneqq \begin{cases} 1 & \text{if } v_i \in e_j \\ 0 & \text{otherwise.} \end{cases}$$

The incidence matrix of the graph in Figure A.2 is

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

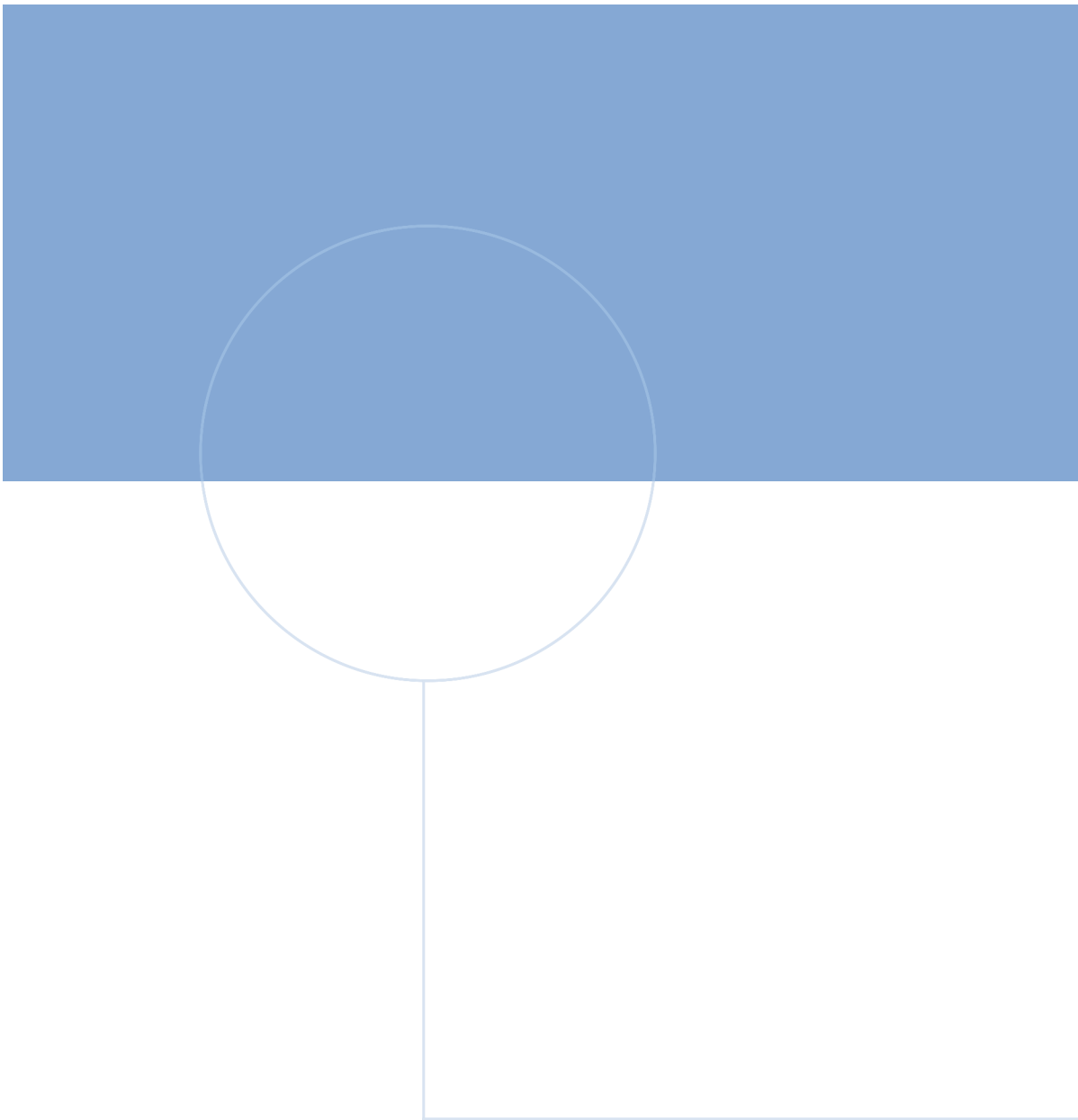
Finally, a graph can be represented by an edge list $C = (c_{ij})_{m \times 2}$, where the rows are edges in the

graph, $C = [e_1, e_2, \ldots, e_m]^T$. For our simple example graph, this gives

$$C = \begin{pmatrix} v_1 & v_2 \\ v_2 & v_3 \\ v_3 & v_4 \\ v_2 & v_4 \end{pmatrix}.$$

A.3 The MATLAB graph

MATLAB has a built-in graph object for undirected graphs. In its most basic form, such a graph has a set of nodes and a set of edges, and is defined by an edge list. The nodes and edges are represented by each their table. While additional properties on a graph are usually restricted to weights, the MATLAB graph is flexible, supporting assignment of custom properties to both nodes and edges. A new property is simply added as a column in the corresponding table. The graph object comes with a set of basic methods, such as accessing the adjacency and incidence matrix, or adding and removing nodes or edges.



NTNU

Norwegian University of
Science and Technology