Adjoint-Based Optimization through Reduced-Order Subdomain Surrogate Modelling

An Application in Reservoir Simulation

D.S. Jagt

TUDelft

Adjoint-Based Optimization through Reduced-Order Subdomain Surrogate Modelling

An Application in Reservoir Simulation

by

Declan Jagt

To obtain the degree of Master of Science in
Applied Mathematics at the Delft University of Technology,
to be defended Wednesday April 29th, at 11:00 AM.

Student number:	4725611	
Specialisation:	Computational Science and Engineering	
Project Supervisor:	Prof. Dr. Ir. A.W. Heemink	
Thesis Committee	Prof. Dr. Ir. A.W. Heemink	TU Delft
	Dr. J.W. van der Woude	TU Delft
	Dr. Ir. O. Leeuwenburgh	TU Delft, TNO

TU Delft

Preface

The following report presents my final research performed towards obtaining the degree of Master of Science in Applied Mathematics, at the Delft University of Technology. In conducting this research, codes from the Netherlands Organization of applied scientific research (TNO) have been used, contributing to a crucial part of the study. The research itself concerns the problem of optimization of oil production from a reservoir, exploring a novel method for doing so at a reduced cost.

The graduation committee that will assess my overall performance is comprised of Prof. Dr. Ir. A.W. Heemink, Dr. J. van der Woude, and Dr. Ir. O. Leeuwenburgh. I would like to thank each of the members of this committee for their support and insight throughout the project, guiding me towards the final product presented here. In addition, I would like to thank X. Cong, MSc, for his extensive help throughout my research, setting me up with the right software and sharing with me his knowledge and ideas. I would also like to thank Prof. Dr. Ir. H.X. Lin and Ir C.W.J. Lemmens for granting me access to a computer and the cluster on campus, and the secretaries of the Mathematical Physics group for opening the door for me in the mornings.

Finally, I would like to thank my family for their continuous support throughout my education, helping me get to where I am today.

Declan Jagt
Rotterdam, April 2020

Abstract

Although a transition to more sustainable energy production is necessary, fossil fuels will remain a crucial contributor to the world's energy production in the near future. In numerically maximizing the production of these fuels, as well as in many other optimization problems, an objective function has to be optimized on the basis of an input vector, related through some underlying model. Although the adjoint methodology generally allows for efficient gradient-based optimization of such a problem, it quickly becomes infeasible when the model comprises a large-scale system, or access to this model is prohibited. To resolve this, we consider the application of a model order reduction scheme, in combination with subdomain surrogate modeling, to perform the optimization using a reduced-order approximation of the model. In particular, we employ principal orthogonal decomposition (POD), efficiently reducing the size of the underlying system of equations, on the basis of a limited number of samples of the full-order model. Applying a domain decomposition, this number of samples can be reduced even further, although at a decreased efficiency. Next, radial basis function (RBF) interpolation is applied, using the samples to construct a trajectory piecewise linear approximation of the reduced-order model in each subdomain. Combining these different techniques, an optimization algorithm is constructed, applying the adjoint methodology to the reduced-order model in order to compute an approximate gradient. The accuracy of this approximation was found to be poor, but comparable to alternative sample-based methods. An improved accuracy could be achieved by reducing the number of input parameters, allowing for more efficient optimization when applying the algorithm to a particular reservoir model. For another model, however, the algorithm performed worse upon reducing the size of the input, as a result of the fewer degrees of freedom in the optimization procedure. Reacquiring this freedom during the optimization, improved results could be attained, but the number of iterations and thus the cost of the method increased drastically. Comparing the full input implementation to another sample-based method, specifically a straight gradient ensemble algorithm, it was found to produce comparable results. Each method was able to surpass the other, dependent on the particular situation, though the reduced-adjoint algorithm generally expended more effort to attain similar results. This suggests that further study is necessary, for example improving the RBF interpolation or input reduction techniques, to fully exploit the benefits of the POD-TPWL methodology.

Contents

Chapter 1

Introduction

1.1 Reservoir oil recovery

Since the start of the industrial revolution, energy consumption worldwide has increased tremendously. This has fuelled a surge in technological development, as well as a rapid population growth, still continuing to this day [36]. Accordingly, the total energy demand continues to grow each year, expected to increase with 50% by 2050 [12]. Currently, this energy is largely produced from fossil fuels, with oil and natural gas still accounting for more than half of the total energy production in 2018 [5]. As such, although combating climate change necessitates a reduction in the use of these carbon-rich fuels, they will remain crucial sources of energy throughout the transition towards less environmentally damaging production. Therefore, it is essential these products are obtained as efficiently as possible, both from an economical and ecological perspective.

Resources such as oil are generally extracted from underground reservoirs. In the primary stage of this extraction, the internal pressure naturally forces the oil out of such reservoirs, allowing it to be pumped up to the surface with little effort [6]. As this method allows only a small portion of the available oil to be extracted, however, a secondary recovery method is also used, called waterflooding. As part of this process, water is injected into the reservoir through injection wells, aiming to push the present oil towards production wells, as illustrated in figure 1.1. At these latter wells, any present phase can then be extracted, including both the water and the oil. Naturally, the goal is to maximize this oil production, whilst minimizing the water injection and production, expending as little effort as possible to recover the greatest amount of viable product.

In order to optimize the process of waterflooding, reservoir simulation is applied. To this end, information has to be gathered on the reservoir, for example performing seismic surveys or well test analyses. This already imposes an optimization problem in minimizing the cost associated with accumulating information, compared to the benefit of having this information, requiring so-called value-of-information analysis. An additional optimization problem arises in establishing parameters necessary to describe the flow in the reservoir, referred to as history matching. This problem involves minimizing the difference between simulation results and physical observations, seeking model parameters to best represent the dynamics within the reservoir. Once a model of the flow is constructed, production optimization can finally be performed, determining the necessary control exerted at the injection and production wells to maximize the amount of extracted oil. In many cases, geological uncertainty is still taken into account at this stage, seeking optimal controls maximizing average production under different model parameters.

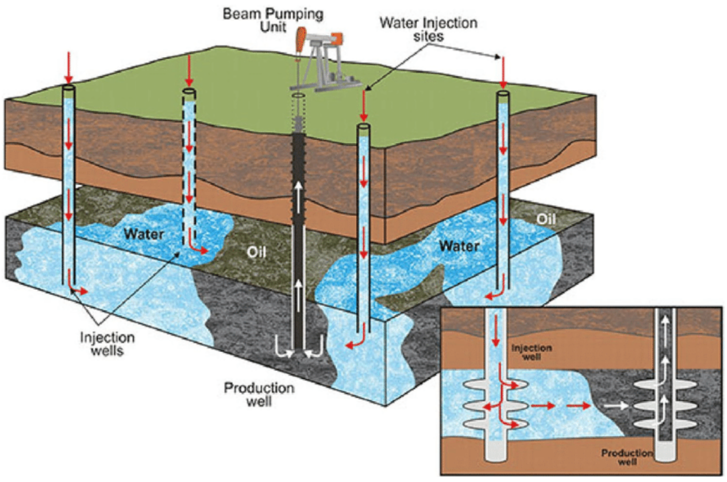


Figure 1.1: Waterflooding of an oil reservoir [22]

1.2 Adjoint-based optimization

For solving optimization problems such as those involved in oil recovery, a variety of techniques are available. These techniques operate by optimizing an objective function, which expresses the quality of a particular situation in terms of a single value. Establishing an optimum can then be done using the gradient of this function, which in turn can be computed through a multitude of methods.

A common technique used to construct the gradient of an objective function is the adjoint method. Through this method, the derivative of the objective function with respect to the control parameters is computed by directly implementing the model functions in the derivation [21]. This allows an exact gradient to be determined under any circumstances, requiring relatively little computational effort when considering small-scale models. For problems such as those involved in oil recovery, however, often an overwhelming number of variables have to be taken into account, increasing the computational cost tremendously. Moreover, access to the actual models is not always feasible, in which case the information necessary to employ an adjoint method is not available.

To address the various problems in applying an adjoint method to construct a gradient, reduction and reparameterization techniques have been investigated within this context. Reduced-order modelling techniques such as principal orthogonal decomposition (POD) have been shown to allow a significant reduction in the cost of the method [7][29]. Additionally applying surrogate modelling techniques such as trajectory piecewise linearization (TPWL), the complexity of the method can be further reduced [16][17][18]. Finally, the inherent intrusive property of the adjoint technique can be lifted using radial basis function (RBF) interpolation [24][42], which can be combined with a subdomain scheme to reduce the cost. Combining these different methods into a subdomain POD-TPWL algorithm, adjoint gradients can then be computed on the basis of a reduced-order linear approximation of the model, yielding promising results when applied to history matching [43]. At the time of writing of this thesis, however, this non-intrusive POD-TPWL methodology has not been tested on alternative optimization problems, raising the question of how it would perform for a problem such as production optimization.

1.3 Synopsis of the thesis

In this study, we investigate the performance of a non-intrusive, reduced-order linearization technique for optimization, which we will refer to as the POD-TPWL or reduced adjoint method. We will consider this methodology specifically in the context of production optimization in reservoir flow, implementing the algorithm on the basis of two reservoir models. For this situation, we optimize the performance of the algorithm, and compare it to a similar, more well-established ensemble based algorithm. In addition, we consider implementing methods for reducing the number of parameters to be optimized, as also done in the history matching application [43], and how these affect the algorithm. Overall, we will address three questions:

1. How do the different stages of the algorithm affect its overall performance?
2. How do different schemes reducing the number of parameters to be optimized influence the efficacy of the algorithm?
3. How does the POD-TPWL method compare to a similar ensemble-based method when applied to the problem of production optimization?

Before considering these questions, necessary background on reservoir simulation and optimization methods is introduced in chapters 2 and 3. Consecutively, chapters 4 and 5 provide the mathematical theory as well as numerical experiments on the core techniques applied in the POD-TPWL method, addressing the first research question. The obtained results also drive choices made in the implementation of the algorithm, described in chapter 7. Surrounding chapters 6 and 8 answer the second question, suggesting various schemes for adjusting the parameters to be optimized, and displaying the performance of the algorithm based on these schemes. Finally, chapter 9 provides a comparison between the algorithm and an ensemble-based method, answering the final question. The most important results of these chapters are summarized in the conclusion, additionally providing suggestions for further improvement.

Chapter 2

The Porous Media Model

In this chapter, the system of equations governing the flow of oil and water through a reservoir is derived. Spatial and temporal discretization are performed, describing the flow in terms of unknown state variables to be numerically computed at specific times. An objective function is introduced, assigning a monetary value to the situation corresponding to these states. Finally, two reservoir models are described, on the basis of which any numerical experiments throughout this thesis will be conducted.

2.1 System of equations

Throughout this thesis, the studied optimization techniques will be considered specifically within the context of optimization of oil production from a reservoir. This production will be performed using waterflooding, thus depending on the manner in which water is injected into the reservoir, as well as the flow of the water and oil through the reservoir. To describe this dependence, a model of the flow within the reservoir is used. This model computes the flow at any time by solving a system of equations which this flow must satisfy.

2.1.1 Conservation laws

To attain a general set of equations describing the flow of different phases through a reservoir, we focus on a small cubic volume within the reservoir. Considering the flow through this volume, the mass of any phase ℓ, such as water or oil, must be conserved. This means that any mass flowing into the volume, either naturally or as a result of external influence, must also exit the volume, or accumulate within it. Consequently, the flow of mass through the volume must satisfy:

$$\text{([rate in] - [rate out])} + \text{([rate injected] - [rate extracted])} = \text{[rate accumulated]} \tag{2.1}$$

In this equation, the first term denotes the mass flux, describing the rate at which mass flows into and out of the considered volume from the surroundings. This rate depends on the change in velocity of the flow $\mathbf{v}_\ell$ of the phase through the volume, where an increase in this velocity implies that mass leaves the volume more quickly than it enters. Naturally, this rate also depends on the density ρ_ℓ of the phase in the box, determining how much mass the flow actually carries. Based on these variables, the flux can be shown to be given as [33]:

$$\text{([rate in] - [rate out])} = -\left(\frac{\partial(\rho_\ell v_{\ell,x})}{\partial x} + \frac{\partial(\rho_\ell v_{\ell,y})}{\partial y} + \frac{\partial(\rho_\ell v_{\ell,z})}{\partial z}\right) = -\boldsymbol{\nabla}\cdot(\rho_\ell \mathbf{v}_\ell) \tag{2.2}$$

Aside from natural flow from the surroundings, we also have to account for flow of mass into the volume from external sources. In the setting of waterflooding, for example, this corresponds to the injected or extracted fluids at the locations of wells. We denote the rate of this imposed flow as q_ℓ, with a positive value corresponding to flow of phase ℓ into the volume. The rate at which mass enters the volume from outside the reservoir can then be given as:

$$\text{([rate injected] - [rate extracted])} = \rho_\ell q_\ell \tag{2.3}$$

Finally, we have to consider the mass that is stored within the volume. This depends on the porosity ϕ of the material through which the flow occurs, being defined as the ratio of the pore volume to the total volume of this material. As such, it is a measure of how well the medium can retain an arbitrary fluid, where a higher porosity corresponds to more available space for mass accumulation. However, in case of multiple phases, each phase ℓ can only occupy a fraction of this available space, which we denote by the saturation s_ℓ of this phase. The fraction of the total volume occupied by phase ℓ is then given by ϕs_ℓ, such that the rate of accumulation of mass in the volume is equal to:

$$\text{[rate accumulated]} = \frac{\partial(\rho_\ell \phi s_\ell)}{\partial t} \tag{2.4}$$

Combining these results, we find that conservation of mass for an arbitrary phase ℓ requires:

$$-\nabla \cdot (\rho_\ell \mathbf{v}_\ell) + \rho_\ell q_\ell = \frac{\partial(\rho_\ell \phi s_\ell)}{\partial t} \tag{2.5}$$

Imposing this equation, the flow of phase ℓ through the reservoir will conserve mass. Aside from conserving mass, however, the flow of this phase must also conserve momentum. When considering porous media, this is generally imposed through Darcy's law, requiring the velocity of phase ℓ to satisfy [19]:

$$\mathbf{v}_\ell = -\frac{\mathbf{K}}{\mu_\ell}\left(\boldsymbol{\nabla} p_\ell - \rho_\ell g \boldsymbol{\nabla} d\right) \tag{2.6}$$

In this equation the first term describes the effect of the pressure gradient $\boldsymbol{\nabla} p_\ell$ of phase ℓ on its velocity, with a larger pressure gradient also invoking a greater flow velocity. The second term in the equation corresponds to the effects of gravity, with g denoting the gravitational acceleration, and d the depth of the reservoir, dependent on the considered position. This term incorporates the effect of sloping at the bottom of the reservoir, where gravity will increase the velocity down along the slope. Both effects are scaled by a factor dependent on the viscosity μ_ℓ of the phase, where a higher viscosity diminishes the effects on the velocity.

In addition to the viscosity, the permeability $\mathbf{K}$ of the rock also affects the velocity of the phase. This tensor provides a measure of how well a phase can pass through the medium, where a higher permeability corresponds to less obstruction of the flow [2]. In doing so, it accounts for the effects of the pressure gradient along each direction on the velocity in each direction, thus forming a full 3×3 matrix. In general, however, a proper choice of coordinate system allows for this tensor to be written as a diagonal matrix, taking only the effect of the pressure gradients on flow in the same direction to be nonzero.

Substituting Darcy's law into the conservation of mass equation, we find that the flow of phase ℓ must obey:

$$\boldsymbol{\nabla} \cdot \left[\frac{\rho_\ell \mathbf{K}}{\mu_\ell}(\nabla p_\ell - \rho g \nabla d)\right] + \rho_\ell q_\ell = \frac{\partial(\rho_\ell \phi s_\ell)}{\partial t} \tag{2.7}$$

In this equation, the density of the phase and porosity of the rock are taken to depend only on the pressure, whilst the relative permeability depends only on the saturation of the present phases. The rate q_ℓ can be influenced externally, potentially also depending on the pressure, and being nonzero only at the location of wells. All other parameters are assumed to be known and constant, such that only the pressure and saturation are unknown variables.

2.1.2 Closure equations

We now assume to have N_ℓ phases in the reservoir. Each of these phases must satisfy a corresponding differential equation (2.7), yielding a total of N_ℓ equations to be solved. However, both the pressure and saturation in each of these equations are unknown, amounting to twice as many unknown variables. To assure a unique solution, therefore, closure equations are imposed on these variables. The first of these is straightforward, requiring all space in the reservoir to be occupied. By definition, then, the sum of the saturation of all phases must be equal to 1, imposing:

$$1 = \sum_{\ell=1}^{N_\ell} s_\ell \tag{2.8}$$

In the presence of a single phase, this condition implies the saturation of the phase to be equal to 1, leaving only the pressure to be computed. In a situation involving multiple phases, however, additional equations are required in order to attain a well-posed problem. These equations can be found considering the pressure difference at the interface between two phases. Specifically, for an arbitrary pair of phases ℓ_1, ℓ_2, the pressure difference between the non-wetting phase and the wetting phase can be given as a function of solely the saturation of the wetting phase [33]. Here, the wetting phase is the phase with the highest tendency to adhere to the surrounding rock, with the other phase denoted as non-wetting. Assuming the phases $\ell = 1, \dots, N_\ell$ to be ordered from most- to least-wetting, this yields $N_\ell - 1$ additional equations:

$$p_{\ell+1} - p_\ell = p_{c_{\ell+1,\ell}}(s_\ell) \qquad \ell = 1, \dots, N_\ell - 1 \tag{2.9}$$

Here, p_c denotes the capillary pressure between two phases, being some known function of the saturation of the wetting phase. Since the capillary pressure between two arbitrary phases can be given as a sum of those between consecutive phases, the equations (2.9) are sufficient to guarantee all capillary pressure requirements are imposed. In combination with the total saturation condition and the conservation equations, this gives $2N_\ell$ equations on the N_ℓ phases.

2.1.3 Two phase flow

Throughout this thesis, only a situation with water and oil is considered, involving just two phases. Of these two, water is the wetting phase, allowing for the water pressure to be expressed as:

$$p_\mathrm{w} = p_\mathrm{o} - p_\mathrm{c}(s_\mathrm{w}) \tag{2.10}$$

Similarly, the oil saturation can be expressed directly in terms of the water saturation as:

$$s_\mathrm{o} = 1 - s_\mathrm{w} \tag{2.11}$$

Finally, the presence of water will inhibit the flow of the oil, and vice versa. To take this into account, relative permeabilities $k_{r\mathrm{o}}$ and $k_{r\mathrm{w}}$ are introduced, describing the additional resistance to the flow of each phases as result of the presence of the other. Letting $\mathbf{K}$ describe the absolute permeability of the rock, the full permeability for phase ℓ then becomes:

$$\mathbf{K}_\ell = k_{r,\ell}\mathbf{K} \tag{2.12}$$

Substituting these results into the conservation equations, we can rephrase them in terms of only the oil pressure and water saturation, and with a relative permeability:

$$\begin{aligned} \boldsymbol{\nabla}\cdot\left[\frac{\rho_\mathrm{w} k_{r\mathrm{w}}}{\mu_\mathrm{w}}\mathbf{K}\left(\boldsymbol{\nabla} p_\mathrm{o} - \frac{\partial p_c}{\partial s_\mathrm{w}}\boldsymbol{\nabla} s_\mathrm{w} - g\boldsymbol{\nabla} d\right)\right] + \rho_\mathrm{w} q_\mathrm{w} &= \frac{\partial(\rho_\mathrm{w}\phi s_\mathrm{w})}{\partial t} \\ \boldsymbol{\nabla}\cdot\left[\frac{\rho_\mathrm{o} k_{r\mathrm{o}}}{\mu_\mathrm{o}}\mathbf{K}(\boldsymbol{\nabla} p_\mathrm{o} - g\boldsymbol{\nabla} d)\right] + \rho_\mathrm{o} q_\mathrm{o} &= \left[\frac{\partial(\rho_\mathrm{o}\phi)}{\partial t} - \frac{\partial(\rho_\mathrm{o}\phi s_\mathrm{w})}{\partial t}\right] \end{aligned} \tag{2.13}$$

To further simplify this system, we expand the accumulation derivatives of each phase ℓ, where the density and porosity are assumed only to depend on the corresponding pressure:

$$\frac{\partial(\rho_\ell\phi s_\mathrm{w})}{\partial t} = \phi s_\mathrm{w}\frac{\partial \rho_\ell}{\partial p_\ell}\frac{\partial p_\ell}{\partial t} + \rho_\ell s_\mathrm{w}\frac{\partial\phi}{\partial p_\ell}\frac{\partial p_\ell}{\partial t} + \rho_\ell\phi\frac{\partial s_\mathrm{w}}{\partial t} \tag{2.14}$$

In general, the density and porosity depend only weakly on the pressure. Assuming the temperature to be constant, the derivatives of these parameters can then be described in terms of corresponding compressibilities:

$$\begin{aligned} c_\ell(p_\ell) &:= \frac{1}{\rho_\ell}\frac{\partial\rho_\ell}{\partial p_\ell} \\ c_r(p_\ell) &:= \frac{1}{\phi}\frac{\partial\phi}{\partial p_\ell} \end{aligned} \tag{2.15}$$

These compressibilities provide a measure of the relative change in volume of each phase and the surrounding rock, in response to a change in pressure of a phase. Assuming the capillary effects to be small, all of these compressibilities can be expressed as functions of the oil pressure, allowing the accumulation term for phase ℓ to be given as:

$$\frac{\partial(\rho_\ell\phi s_\mathrm{w})}{\partial t} = \rho_\ell\phi s_\mathrm{w} c_\ell(p_\mathrm{o})\frac{\partial p_\mathrm{o}}{\partial t} + \rho_\ell\phi s_\mathrm{w} c_r(p_\mathrm{o})\frac{\partial p_\mathrm{o}}{\partial t} + \rho_\ell\phi\frac{\partial s_\mathrm{w}}{\partial t} \tag{2.16}$$

Finally, substituting this result into the system (2.13), we attain:

$$\begin{aligned} \boldsymbol{\nabla}\cdot\left[\frac{\rho_\mathrm{w} k_{r\mathrm{w}}}{\mu_\mathrm{w}}\mathbf{K}\left(\boldsymbol{\nabla} p_\mathrm{o} - \frac{\partial p_c}{\partial s_\mathrm{w}}\boldsymbol{\nabla} s_\mathrm{w} - \rho_\mathrm{w} g\boldsymbol{\nabla} d\right)\right] + \rho_\mathrm{w} q_\mathrm{w} &= \rho_\mathrm{w}\phi\left[s_\mathrm{w}(c_\mathrm{w}+c_r)\frac{\partial p_\mathrm{o}}{\partial t} + \frac{\partial s_\mathrm{w}}{\partial t}\right] \\ \boldsymbol{\nabla}\cdot\left[\frac{\rho_\mathrm{o} k_{r\mathrm{o}}}{\mu_\mathrm{o}}\mathbf{K}(\boldsymbol{\nabla} p_\mathrm{o} - \rho_\mathrm{o} g\boldsymbol{\nabla} d)\right] + \rho_\mathrm{o} q_\mathrm{o} &= \rho_\mathrm{o}\phi\left[(1-s_\mathrm{w})(c_\mathrm{o}+c_r)\frac{\partial p_\mathrm{o}}{\partial t} - \frac{\partial s_\mathrm{w}}{\partial t}\right] \end{aligned} \tag{2.17}$$

This is the set of nonlinear differential equations describing the flow of oil and water in a reservoir, as considered in this thesis. Since the pressure is always computed for the oil phase, whilst the saturation is always computed for the water phase, we will henceforth refer to these unknowns without explicitly mentioning the corresponding phase, also omitting the o and w subscripts in the equations.

2.2 State space representation

To solve the differential system (2.17), the numerical simulator *flow* will be used, which is part of the Open Porous Media initiative, available at [32]. This software computes the unknown values in a reservoir specified by the user, applying a time-implicit two-point flux finite volume scheme. We will broadly describe the techniques behind this method here, referring to the OPM documentation for more information.

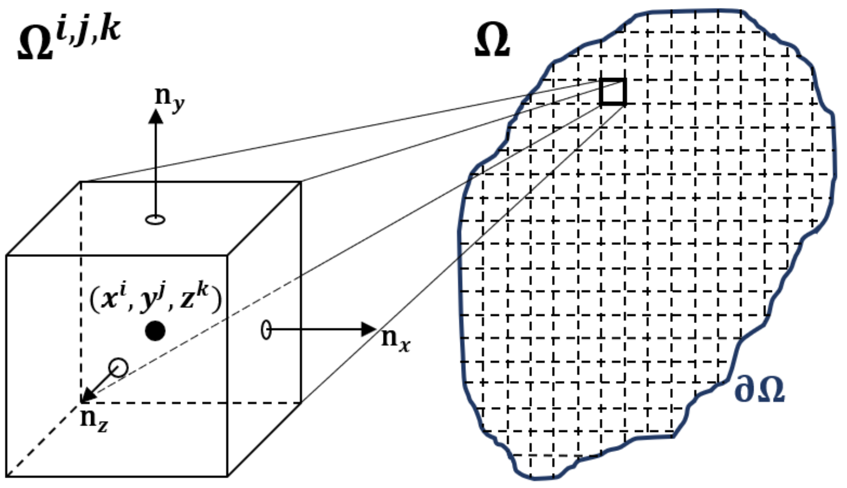


Figure 2.1: Spatial discretization of a domain Ω

2.2.1 Spatial discretization

We consider the problem of solving system (2.17) in some computational domain Ω, in our case denoting the full space occupied by the reservoir. We assume this domain to be bounded by a surface $\partial\Omega$, for example entailing some layer of rock that is impenetrable to the phases. In order to solve this problem, the finite volume method, as well as most alternative numerical methods, divides the computational domain into a large number of smaller grid blocks [14], as illustrated in figure 2.1. Using a Cartesian coordinate system, we could for example divide the domain into N_g cubic subdomains $\Omega^{i,j,k}$, with boundaries $\partial\Omega^{i,j,k}$, where indices (i, j, k) denote the position of the subdomain within the reservoir. Rather than determining the unknown value at an arbitrary position, then, these are only determined at the centers of each grid block, for example requiring them to assume a uniform value throughout each grid block. In our case, this amounts to determining N_g pressure and saturation values at any time t, which we will combine into a single state vector $\mathbf{x}(t) \in \mathbb{R}^{N_\mathrm{x}}$, with $N_\mathrm{x} := 2N_g$. In each subdomain (i, j, k), this state must then satisfy an equation of the form:

$$\boldsymbol{\nabla} \cdot \boldsymbol{F}\left(\mathbf{x}(t), \nabla\mathbf{x}(t)\right)\Big|^{ijk} + S(\mathbf{x}(t), \mathbf{u}(t))\Big|^{ijk} = \frac{\partial}{\partial t} A(\mathbf{x}(t))\Big|^{ijk} \tag{2.18}$$

In this system, $\boldsymbol{\nabla F}\left(\mathbf{x}(t), \nabla\mathbf{x}(t)\right)$ describes the mass flux, $S(\mathbf{x}(t), \mathbf{u}(t))$ the source effects, and $\frac{\partial}{\partial t}A(\mathbf{x}(t))$ the accumulation in grid block (i, j, k). Each of these terms depends on the state $\mathbf{x}(t)$

at the corresponding time, with the mass flux further depending on the gradient of the state, and the source effects also depending on some input $\mathbf{u}(t) \in \mathbb{R}^{N_\mathrm{w}}$, describing the control exerted at the wells. This control could entail any variable influencing the rate of flow $q^{i,j,k}$ in the considered domain, including simply the rate itself.

To solve system (2.18) in an arbitrary domain $\Omega^{i,j,k}$, we first integrate both sides over the entire domain:

$$\int_{\Omega^{i,j,k}} \left[\boldsymbol{\nabla} \cdot \boldsymbol{F}\left(\mathbf{x}(t), \nabla \mathbf{x}(t)\right) + S(\mathbf{x}(t), \mathbf{u}(t))\right] d\Omega = \int_{\Omega^{i,j,k}} \frac{\partial}{\partial t} A(\mathbf{x}(t)) d\partial\Omega \tag{2.19}$$

Using Gauss's law, the integral of the flux term can be transformed into:

$$\int_{\Omega^{i,j,k}} \boldsymbol{\nabla} \cdot \boldsymbol{F}\left(\mathbf{x}(t), \nabla \mathbf{x}(t)\right) d\Omega = \int_{\partial\Omega^{i,j,k}} \mathbf{n} \cdot \boldsymbol{F}\left(\mathbf{x}(t), \nabla \mathbf{x}(t)\right) d\partial\Omega \tag{2.20}$$

In this notation, the vector $\mathbf{n}$ at any point along the boundary $\partial\Omega$ is a unit vector pointing outward, perpendicular to this boundary. Since we are using cubic domains in a Cartesian grid, this vector will be uniform along each of the six faces along the domain, containing a single nonzero element equal to ± 1 in the direction normal to this face. Assuming further the pressure and saturation values to be uniform along each face, only their derivatives perpendicular to the face will be nonzero, allowing us to neglect the other elements. This implies that, considering for example the boundaries along the y-direction, we can express the integral as:

$$\begin{aligned}\int_{\partial\Omega^{i,j,k}} \mathbf{n}_y \cdot \boldsymbol{F}\left(\mathbf{x}(t), \boldsymbol{\nabla}\mathbf{x}(t)\right) d\partial\Omega &= \Delta_y F_y^{i,j,k}(\mathbf{x}(t), \boldsymbol{\nabla}\mathbf{x}(t)) \\ &:= (\Delta x \Delta z)\left[F_y\left(\mathbf{x}(t), \frac{\partial}{\partial y}\mathbf{x}(t)\right)\right]\Bigg|_{i,j-\frac{1}{2},k}^{i,j+\frac{1}{2},k}\end{aligned} \tag{2.21}$$

Here $\Delta x \Delta z$ denotes the area of the y-boundaries, assuming each subdomain to be of dimensions $\Delta x \times \Delta y \times \Delta z$. With this derivation, the divergence operator from the system is removed, but a dependence upon derivatives of the state vector remains. To remove this, a two-point finite difference approximation is used, for example replacing the derivative at boundary $(i, j + \frac{1}{2}, k)$ by:

$$\partial_y \mathrm{x}_{s,p}^{i,j+\frac{1}{2},k} \approx \frac{\mathrm{x}_{s,p}^{i,j+1,k} - \mathrm{x}_{s,p}^{i,j,k}}{\Delta y} \tag{2.22}$$

In this notation, subscripts s, p denote respectively the saturation and pressure elements of the state vector $\mathbf{x}$. Along the boundary $\partial\Omega$ of the reservoir, approximating the derivative in this manner requires implementing boundary conditions. This can be done by, for example, introducing virtual cells outside of the domain, in which we fix the saturation and pressure values to be predefined functions of those in neighboring cells within the domain. Substituting this function into the approximation (2.22), we can then express the gradient of the state as a (linear) operator acting on this state, lifting the dependency on the gradients $\boldsymbol{\nabla}\mathbf{x}$.

Since the pressure and saturation are assumed to be uniform throughout each cell, integrating the source and accumulation functions simply entails multiplying these with the volume $\Delta x \Delta y \Delta z$. Imposing this result in integral equation (2.19), as well as substituting the function (2.21) with the gradient dependence described through (2.22), this equation then becomes:

$$\sum_{\xi \in \{x,y,z\}} \Delta_\xi F_\xi^{i,j,k}(\mathbf{x}(t)) + (\Delta x \Delta y \Delta z) S^{i,j,k}(\mathbf{x}(t), \mathbf{u}(t)) = (\Delta x \Delta y \Delta z) \frac{\partial}{\partial t} A^{i,j,k}(\mathbf{x}(t)) \tag{2.23}$$

In this equation, $S^{i,j,k}$ and $A^{i,j,k}$ represent the source and accumulation effects at position (i, j, k) in the grid. Solving this system of equations at an arbitrary time t, approximate pressure and saturation values can then be computed at the grid points.

2.2.2 Temporal discretization

In system (2.23), all spatial derivatives have been removed using spatial discretization. To also remove the derivative in time, temporal discretization is applied. To this end, let t_n for $n = 1, \dots, N$ denote times at which we wish to compute the state vector, where we assume the state at time t_0 to be known. Using an implicit scheme, we then approximate the temporal derivative in the accumulation term as:

$$\frac{\partial}{\partial t} A^{i,j,k}(\mathbf{x}^{n+1}) \approx \frac{A^{i,j,k}(\mathbf{x}^{n+1}) - A^{i,j,k}(\mathbf{x}^{n})}{\Delta t} \tag{2.24}$$

In this approximation, $\mathbf{x}^n$ denotes the state at time step n, with Δt providing the time between consecutive time steps, assumed to be constant for ease of notation. Substituting this approximation into system (2.23), as well as dividing by the volume, we attain:

$$\frac{1}{\Delta x \Delta y \Delta z} \sum_{\xi \in \{x,y,z\}} \Delta_\xi F^{i,j,k}(\mathbf{x}^{n+1}) + S^{i,j,k}(\mathbf{x}^{n+1}, \mathbf{u}^{n+1}) = \frac{A^{i,j,k}(\mathbf{x}^{n+1}) - A^{i,j,k}(\mathbf{x}^{n})}{\Delta t} \tag{2.25}$$

In this system, all derivatives have been removed, leaving only functions directly dependent on the state vector $\mathbf{x}^n$ and input $\mathbf{u}^n$ at each time step. Combining the different functions, as well as merging the functions at each grid block, we can describe this system compactly as one of the form:

$$\mathbf{0} = \mathbf{g}^{n+1}(\mathbf{x}^n, \mathbf{x}^{n+1}, \mathbf{u}^{n+1}) \qquad n = 0, \dots, N-1 \tag{2.26}$$

This is the general system solved by the numerical simulator. At each time step $n+1 = 1, \dots, N$, this simulator will use the current input $\mathbf{u}^{n+1}$ and previous state $\mathbf{x}^n$ to compute a new state $\mathbf{x}^{n+1}$ satisfying (2.26), starting with a predefined state $\mathbf{x}^0 = \mathbf{x}^{\text{init}}$. In this manner, the pressure and saturation values at certain positions within a specified reservoir are determined at discrete times $t_1, \dots, t_N$, based on the inputs $\mathbf{u}^1, \dots, \mathbf{u}^N$ provided at each time.

2.2.3 Input-output system

When performing production optimization in reservoir flow, it is not directly the state vector that is used to quantify the performance. Instead, quantities such as the water injection and oil production rates at each of the wells are considered when optimizing two-phase flow. Therefore, it is useful to collect these values into a single vector $\mathbf{y} \in \mathbb{R}^{N_\mathrm{y}}$, and rewrite system (2.26) in terms of an input-output system. That is, we assume the vector $\mathbf{y}^n$ at an arbitrary simulation time t_n, to be given as a direct function $\mathbf{h}^n : \mathbb{R}^{N_\mathrm{x}} \times \mathbb{R}^{N_\mathrm{u}} \to \mathbb{R}^{N_\mathrm{y}}$ of the state and input at this time. Furthermore, by adding $\mathbf{x}^{n+1}$ to both sides of equality (2.26), we explicitly require the state $\mathbf{x}^{n+1}$ at a new time step to be equal to some function $\mathbf{f}^{n+1}(\mathbf{x}^n, \mathbf{x}^{n+1}, \mathbf{u}^{n+1})$. We can them write our system as:

$$\begin{aligned}
\mathbf{x}^0 &= \mathbf{x}^{\text{init}} \\
\mathbf{x}^{n+1} &= \mathbf{f}^{n+1}(\mathbf{x}^n, \mathbf{x}^{n+1}, \mathbf{u}^{n+1}) \\
\mathbf{y}^{n+1} &= \mathbf{h}^{n+1}(\mathbf{x}^{n+1}, \mathbf{u}^{n+1}) \qquad n = 0, \dots, N-1
\end{aligned} \tag{2.27}$$

Throughout this thesis, system (2.27) describes the model on the basis of which we wish to optimize a cost function. This model, denoted as the full-order model (FOM), is assumed to be expensive to run, and thus aimed to be executed as sparingly as possible in the numerical optimization schemes. Although we specifically consider this system in the context of reservoir flow, numerically solving equations such as (2.17), the POD-TPWL optimization procedure described in following chapters does not require any knowledge of the actual functions $\mathbf{f}, \mathbf{h}$, and can thus be easily adjusted to fit an arbitrary model.

2.3 Net Present Value

With the model described by equations (2.27), the goal is now to optimize the controls at the wells for the purpose of producing the most oil at the lowest cost. For this, an objective function is introduced, translating observed injection and production rates of water and oil into some monetary value. Specifically, we let $q_{\text{o,w}}^{\text{prod,inj}}(t)$ denote respectively the oil and water production and injection rates at some well at time t, corresponding to water injected at injection wells, and oil and water produced at production wells. We denote the monetary values of these injected and produced fluids by $r_{\text{o,w}}^{\text{prod,inj}}(t)$, such that the total amount of money earned between times t_0 and t_f can be given as:

$$J = \int_{t_0}^{t_f} \left[\sum_{\text{prod}} [r_{\text{o}}^{\text{prod}}(t) q_{\text{o}}^{\text{prod}}(t) - r_{\text{w}}^{\text{prod}}(t) q_{\text{w}}^{\text{prod}}(t)] - \sum_{\text{inj}} [r_{\text{w}}^{\text{inj}}(t) q_{\text{w}}^{\text{inj}}(t)] \right] dt \tag{2.28}$$

This is the objective function aimed to be maximized throughout this thesis, called the net present value. By summing over all production and injection wells, and integrating in time, this function provides a total monetary value corresponding to the observed rates of injection and extraction. Herein, the values corresponding to both water injection and production are subtracted from those corresponding to oil production, as any extracted oil will yield a profit, whereas production or injection of water will only cost money. The monetary values associated with each phase can change in time, generally assumed to decrease as:

$$\begin{aligned} r_{\text{o}}^{\text{prod}}(t) &= r_{\text{o}}^{\text{prod}}(t_0) f_{\text{o}}(t)(1+b_{\text{o}})^{-(t-t_0)/\text{year}} \\ r_{\text{w}}^{\text{prod}}(t) &= r_{\text{w}}^{\text{prod}}(t_0) f_{\text{w}}(t)(1+b_{\text{w}})^{-(t-t_0)/\text{year}} \\ r_{\text{w}}^{\text{inj}}(t) &= r_{\text{w}}^{\text{inj}}(t_0) f_{\text{w}}(t)(1+b_{\text{w}})^{-(t-t_0)/\text{year}} \end{aligned} \tag{2.29}$$

Here, $f_{\text{o,w}}(t)$ is the time-dependent price factor of the fluid, and $b_{o,w}$ denotes a yearly interest rate.

Since optimization will be performed on the basis of a reservoir model, computing results only at discrete time steps $n = 1, \ldots, N$, the integral in the function for the net present value can be replaced by a sum:

$$J = \sum_{n=1}^{N} \left[\sum_{\text{prod}} [r_{\text{o}}^{\text{prod}}(t^n) q_{\text{o}}^{\text{prod}}(t^n) - r_{\text{w}}^{\text{prod}}(t^n) q_{\text{w}}^{\text{prod}}(t^n)] - \sum_{\text{inj}} [r_{\text{w}}^{\text{inj}}(t^n) q_{\text{w}}^{\text{inj}}(t^n)] \right] \tag{2.30}$$

In this equation, the rates of flow observed at each well are taken to exactly comprise the output at an observation time:

$$\begin{pmatrix} \mathbf{y}_{\text{op}}^n \\ \mathbf{y}_{\text{wp}}^n \\ \mathbf{y}_{\text{wi}}^n \end{pmatrix} = \mathbf{y}^n = \mathbf{h}^n(\mathbf{x}^n, \mathbf{u}^n) = \begin{pmatrix} \mathbf{h}_{\text{op}}^n(\mathbf{x}^n, \mathbf{u}^n) \\ \mathbf{h}_{\text{wp}}^n(\mathbf{x}^n, \mathbf{u}^n) \\ \mathbf{h}_{\text{wi}}^n(\mathbf{x}^n, \mathbf{u}^n) \end{pmatrix} = \begin{pmatrix} \mathbf{q}_{\text{o}}^{\text{prod}}(t^n) \\ \mathbf{q}_{\text{w}}^{\text{prod}}(t^n) \\ \mathbf{q}_{\text{w}}^{\text{inj}}(t^n) \end{pmatrix} \qquad n = 1, \ldots, N \tag{2.31}$$

With the rates collected in vector $\mathbf{y}$, a corresponding cost vector can be introduced as:

$$\mathbf{c}^n = \begin{pmatrix} \mathbf{r}_{\text{op}}^n \\ -\mathbf{r}_{\text{wp}}^n \\ -\mathbf{r}_{\text{wi}}^n \end{pmatrix} := \begin{pmatrix} r_{\text{o}}^{\text{prod}}(t^n) \cdot \mathbf{1}^{\text{prod}} \\ -r_{\text{w}}^{\text{prod}}(t^n) \cdot \mathbf{1}^{\text{prod}} \\ -r_{\text{w}}^{\text{inj}}(t^n) \cdot \mathbf{1}^{\text{inj}} \end{pmatrix} \qquad n = 1, \dots, N \tag{2.32}$$

Here, the vectors $\mathbf{1}^{\text{prod}}, \mathbf{1}^{\text{inj}}$ are vectors of sizes corresponding to the number of production and injection wells, containing only elements equal to 1. With this cost vector, the net present value can be given simply as:

$$\begin{aligned} J &= \sum_{n=1}^{N} \left[\sum_{\text{prod}} [r_{\text{o}}^{\text{prod}}(t^n) q_{\text{o}}^{\text{prod}}(t^n) - r_{\text{w}}^{\text{prod}}(t^n) q_{\text{w}}^{\text{prod}}(t^n)] - \sum_{\text{inj}} [r_{\text{w}}^{\text{inj}}(t^n) q_{\text{w}}^{\text{inj}}(t^n)] \right] \\ &= \sum_{n=1}^{N} \left[[\mathbf{r}_{\text{op}}^n]^T \cdot \mathbf{h}_{\text{op}}^n(\mathbf{x}^n, \mathbf{u}^n) - [\mathbf{r}_{\text{wp}}^n]^T \cdot \mathbf{h}_{\text{wp}}^n(\mathbf{x}^n, \mathbf{u}^n) - [\mathbf{r}_{\text{wi}}^n]^T \cdot \mathbf{h}_{\text{wi}}(\mathbf{x}^n, \mathbf{u}^n) \right] \\ &= \sum_{n=1}^{N} [\mathbf{c}^n]^T \cdot \mathbf{h}^n(\mathbf{x}^n, \mathbf{u}^n) \end{aligned} \tag{2.33}$$

In this equation, the dependence of the cost function on the states and inputs at all times is directly displayed, indicating it to be of the form:

$$J = J(\mathbf{x}^1, \dots, \mathbf{x}^N, \mathbf{u}) \tag{2.34}$$

Here, $\mathbf{u} \in \mathbb{R}^{N_{\text{u}}}$ is a vector comprised of all the input variables at all times $n = 0, \dots, N-1$. In production optimization, this is the vector for which an optimum is sought, aiming to maximize the cost function (2.33).

In many cases, the dependence of the objective function on the used parameters is also considered, referred to as robust optimization. In that case, uncertainties in, for example, the permeability values throughout the reservoir are taken into account during the optimization procedure. Letting $\boldsymbol{\beta}$ denote these unknown parameters, optimization could then be performed considering an ensemble of parameter vectors $\boldsymbol{\beta}^{(1)}, \dots, \boldsymbol{\beta}^{(N_\beta)}$, maximizing the ensemble averaged net present value:

$$J = \frac{1}{N_\beta} \sum_{i=1}^{N_\beta} J(\mathbf{x}^1, \dots, \mathbf{x}^N, \mathbf{u}, \boldsymbol{\beta}^{(i)}) \tag{2.35}$$

Naturally, this robust approach to optimization comes with additional complications, as well as requiring a greater computational effort. For this reason, solely deterministic optimization has been considered for this thesis, assuming the chosen parameters $\boldsymbol{\beta}$ to be known with full certainty. For more detailed descriptions on robust optimization, or more recent developments in this field, we refer to [3],[21] and [35], [27].

2.4 Cases

In numerically testing the methods described in the next chapter, two distinct reservoir models are considered. These are the two-dimensional Kanaal model, providing a simple square reservoir of which all grid cells lie in the same plane, and the more complex three-dimensional Egg model. The most important parameters for each model are summarized in table 2.1.

2.4.1 Kanaal model

The Kanaal model is a model of a perfectly square oil reservoir. It covers an area of 990 m by 990 m in the horizontal plane, and has a uniform depth of just 5 m over this entire domain. Accordingly, it is discretized only along the x- and y-direction, dividing it into 99×99 grid blocks. This amounts to a total of 9801 blocks, each measuring $\Delta x \times \Delta y \times \Delta z = 10$ m $\times$ 10 m $\times$ 5 m. At these cells, the permeability $\mathbf{K}$ is assumed to be exactly known, and only assigned values in the horizontal directions, as there is no vertical flow. Moreover, these values are equal in the x- and y-direction, both assigned a predefined value k. Therefore, the permeability in each grid block i can be expressed as $\mathbf{K}^i = k\mathbf{I}_{2\times 2}$, with $\mathbf{I}_{2\times 2} \in \mathbb{R}^{2\times 2}$ denoting an identity matrix. The values of the permeability over the entire reservoir are plotted in figure 2.2.

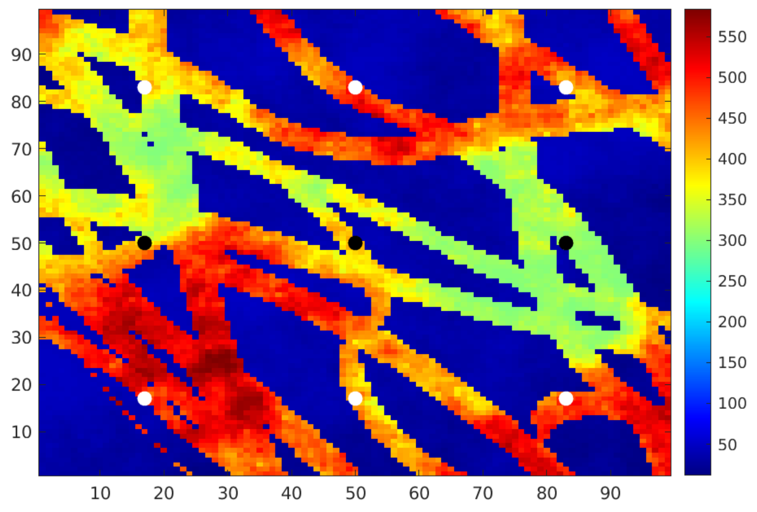


Figure 2.2: Permeability field of the Kanaal reservoir

In the Kanaal model, $N_w = 9$ wells are implemented, each injecting or extracting phases into or from a single grid block. These wells are divided uniformly along the domain, each being equidistant from its closest neighbors, as displayed by the circular markers in figure 2.2. In this figure, the six white markers denote injection wells, forming the northern and southern row of wells. These surround the three production wells at the center of the reservoir, indicated by the black markers in the figure. Organizing the wells in this manner, the aim is to force the oil from the northern and southern parts of the reservoir towards the center, to recover as much oil and as little water as possible.

For the Kanaal model, simulations always start on the first of January in the year 2000. During such a simulation, pressure and saturation values are computed in each gird block every $\Delta t = 30$ days, finishing on the 27th of April in the year 2012. This corresponds to $N = 150$ simulation times, at each of which the injection and production rates at the wells are also observed. During this period, however, control at the wells is only exerted every 300 days, amounting to 15 times at which the control is adjusted. Here, control is exerted through interval control valves. If these valves are fully open, implemented as an input value $\mathrm{u}_{\mathrm{max}} = 1$, flow can pass through the valves unhindered. Conversely, setting the input to $\mathrm{u}_{\mathrm{min}} = 0$, the valve is set to be completely closed, allowing no flow whatsoever. At the control times, the input at each well can be adjusted to be any value in this domain $[\mathrm{u}_{\mathrm{min}}, \mathrm{u}_{\mathrm{max}}]$, after which it will remain constant until the next control time. Fixing the inputs to be constantly equal to 0.5, this returns pressure and saturation fields evolving as displayed in figure 2.3. Here, the state variables are plotted in each grid block at time steps 1, 75 and 150, with the circular markers once more indicating the location of wells.

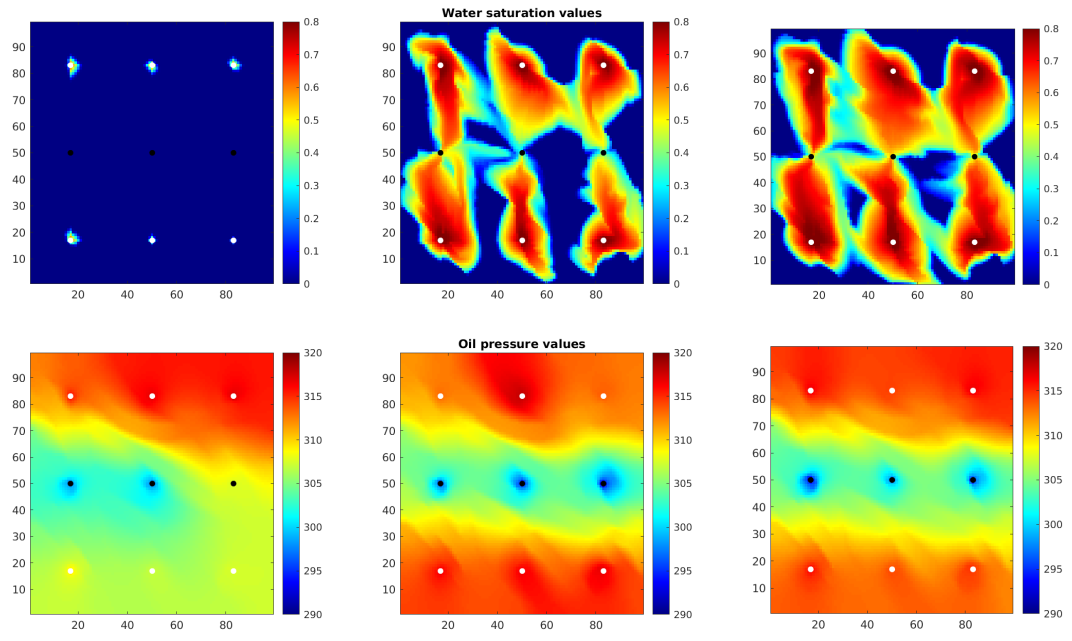


Figure 2.3: Evolution of the saturation and pressure values for the Kanaal model

At each of the simulation steps, the injected and produced rates of water and oil are observed at the wells. An example of these rates using the constant input $\mathrm{u} = 0.5$ at all the wells is given in figure 2.4. Herein, the oil production rates initially increase, corresponding to the injected water driving the oil towards he production wells. At this stage, the water injection and oil production rates are also equal, as the interval control valves at the corresponding wells are set to the same value. However, once water reaches the production wells, water is produced as well as oil. As a result, starting after approximately 30 time steps, the oil production rate decreases at the cost of the water production rate, displaying the difficulties of oil production through waterflooding.

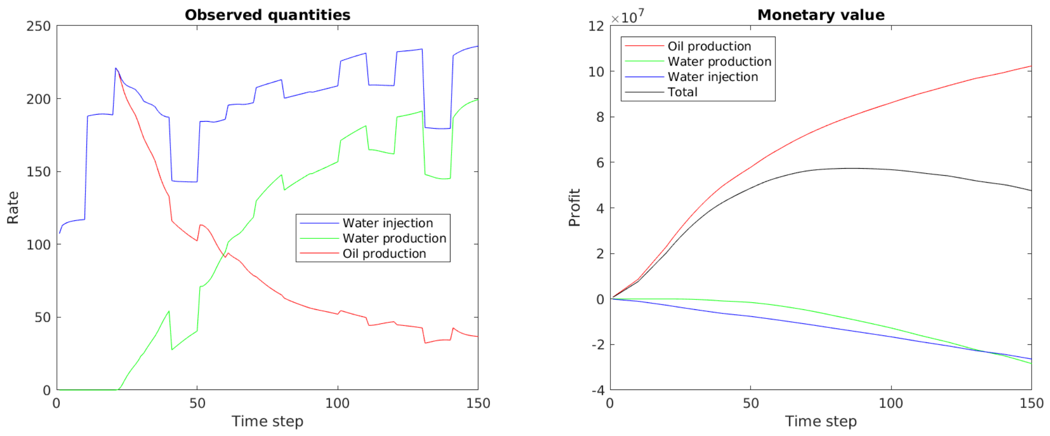


Figure 2.4: Observed injection and production rates and corresponding profit gained over time for the Kanaal model

In addition to the observed rates, the corresponding gained profit is also plotted in figure 2.4. For the Kanaal model as implemented throughout this thesis, monetary values of $r_{\mathrm{o}}^{\mathrm{prod}} = 252$, $r_{\mathrm{w}}^{\mathrm{prod}} = 60$ and $r_{\mathrm{w}}^{\mathrm{inj}} = 30$ were assigned to respectively the produced oil, produced water, and injected water. The profit made from the oil here is much larger than the cost of injecting or producing water, whilst producing water is set to cost more than injecting it, to account for the cost of separation of the produced phases. These values were also taken not to change in time, imposing a constant price factor $f_{\mathrm{o}}(t) = f_{\mathrm{w}}(t) = 1$, and a yearly interest rate of $b_{\mathrm{o}} = b_{\mathrm{w}} = 0$. Implementing these parameters, the profit made in the constant input case initially increases, as only oil is produced and the rate of this production increases. However, as the water production rates start exceeding the oil production rates, the scheme starts costing more money than it returns. As a result, profits start decreasing approximately halfway through the simulation. This shows that the constant input $\mathrm{u} = 0.5$ does not return a desirable net present value, and instead, more variable input functions are necessary.

2.4.2 Egg model

Compared to the Kanaal model, the Egg model appears significantly more often in studies on reservoir flow [20], recently appearing in [8], though varying parameter values are used. It is also more complex than the Kanaal model, pertaining a larger number of cells over a smaller reservoir. This reservoir is located in a domain of 480 m $\times$ 480 m $\times$ 28 m, which is divided into $60 \times 60 \times 7$ grid blocks, each measuring 8 m $\times$ 8 m $\times$ 4 m. However, as no flow is allowed outside the reservoir, pressure and saturation values are only computed in 18553 of the total 25200 cells, which we denote as the active cells. In each of these cells i, the permeability matrix $\mathbf{K}^i$ is diagonal, and assigned the same value k in each direction, as also done for the Kanaal model. For the Egg model, however, flow is also allowed in the vertical direction, such that the tensor is of the form $\mathbf{K}^i = k\mathbf{I}_{3\times 3}$. The values k for the central layer of the reservoir are displayed in figure 2.5.

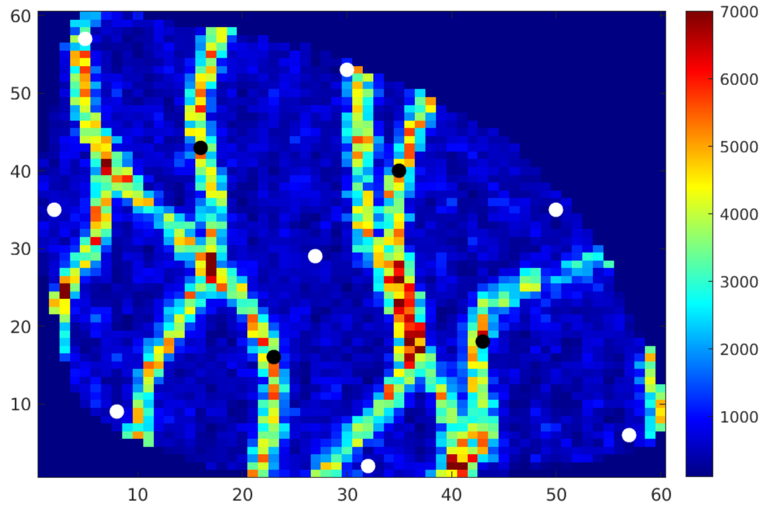


Figure 2.5: Permeability field of the Egg reservoir

As with the Kanaal model, the injection wells in the Egg model are placed to surround the production wells. In particular, seven of the $N_w = 8$ injection wells are distributed along the edge of the reservoir, with a final well located at the center, displayed as the white markers in figure 2.5. The production wells here are denoted by black markers, located halfway towards the edge of the reservoir, enclosed by the injection wells. At these injection wells, control can be exerted directly through the rate of injected flow, fixed to assume values between $\mathrm{u_{min}} = 0\ \mathrm{s}^{-1}$ and $\mathrm{u_{max}} = 79.5\ \mathrm{s}^{-1}$. At the production wells, no control can be exerted, with the rate of flow at these wells fixed at $\mathrm{u} = 39.75\ \mathrm{s}^{-1}$ at all times.

Similarly to the Kanaal model, the unknowns of the Egg model are computed over a period of approximately ten years, starting on the 15th of June in 2011, and finishing on the 23rd of March in 2021. During this time, the pressure and saturation values are computed every Δt=30 days, amounting to a total of $N = 120$ simulation times. Control, however, can only be exerted at $T_w = 40$ times, allowing the rates to be adjusted every third time step of the simulation. Keeping all the rates constant at $\mathrm{u} = 39.75\ \mathrm{s}^{-1}$, pressure and saturation values are obtained as displayed in figure 2.6, providing the state variables after $1, 60$ and 120 simulation steps.

As with the Kanaal model, at each of the simulation times of the Egg model, rates of flow are observed at the wells, yielding results such as those displayed in figure 2.7. In translating these rates to monetary values, the price of each phase is also kept constant in time, once more implementing $f_\mathrm{o}(t) = f_\mathrm{w}(t) = 1$ and $b_\mathrm{o} = b_\mathrm{w} = 0$. However, the actual prices are lower than those of the Kanaal model, fixing $r_\mathrm{o}^\mathrm{prod} = 126$, $r_\mathrm{w}^\mathrm{prod} = 19$ and $r_\mathrm{w}^\mathrm{inj} = 6$. This results in lower profits to be achieved with the Egg model, as also displayed in figure 2.7. Nevertheless, the emerging pattern upon implementing a constant input is similar, initially increasing the profits, but starting to decrease halfway through, as large quantities of water are lost to produce little oil. This illustrates the necessity of an optimization algorithm, allowing more complex input functions to be established, aiming to maximize the net present value.

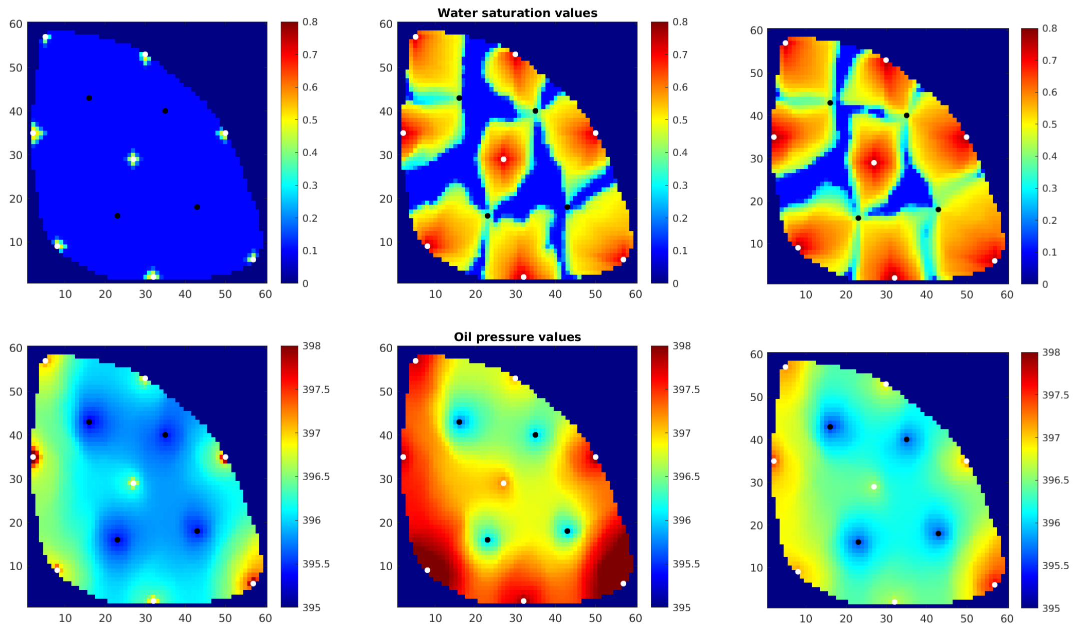


Figure 2.6: Evolution of the saturation and pressure values for the Egg model

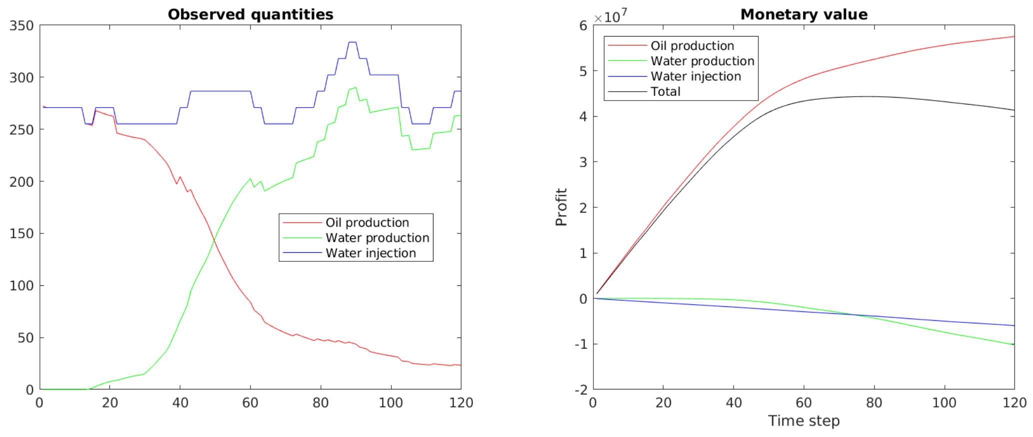


Figure 2.7: Observed injection and production rates and corresponding profit gained over time for the Egg model

		Kanaal model	Egg model
Number of active cells	(N_g)	9801	18553
Number of state variables	(N_x)	19802	37106
Number of time steps	(N)	150	120
Number of output variables	(N_y)	15	20
Number of controlled wells	(N_w)	9	8
Number of control times	(T_w)	15	40
Number of input variables	(N_u)	135	320
Lower bound on input values	$(\mathrm{u}_\mathrm{min})$	0	0
Upper bound on input values	$(\mathrm{u}_\mathrm{max})$	1	79.5
Oil production profit	$(r_\mathrm{o}^\mathrm{prod})$	252	126
Water production cost	$(r_\mathrm{w}^\mathrm{prod}$	60	19
Water injection cost	$(r_\mathrm{w}^\mathrm{inj})$	30	6
Oil price factor	$(f_\mathrm{o}(t))$	1	1
Water price factor	$(f_\mathrm{w}(t))$	1	1
Yearly oil interest rate	(b_o)	0	0
Yearly water interest rate	(b_w)	0	0

Table 2.1: Important parameters of the Kanaal and Egg model

Chapter 3

Optimization Methods

In this chapter, the different optimization schemes employed throughout this thesis are introduced. A procedure for establishing optimal arguments to an objective function is presented, utilizing the gradient of this function to iteratively establish new optima. Several techniques for computing this gradient are discussed, specifically describing the methods as implemented for this thesis.

3.1 Gradient-based optimization

We now consider the problem of constrained optimization, which we can describe as:

$$\begin{aligned} & \max_{\mathbf{u}\in\mathbb{R}^{N_u}} J(\mathbf{u}) && \\ \text{subject to} \quad & \mathrm{u}_i \geq \mathrm{u}_{\min} && i = 1, \ldots, N_u \\ & \mathrm{u}_i \leq \mathrm{u}_{\max} && i = 1, \ldots, N_u \end{aligned} \tag{3.1}$$

In our case, the objective function J is the net present value, with the vector $\mathbf{u}$ providing the inputs at all times and wells as described in the previous chapter. For such a situation, analytically solving the problem (3.1) is impossible, and we have to resort to numerical methods. Herein, we distinguish between the classes of gradient-based and gradient-free methods. The first of these pertains methods applying the gradient of the cost function with respect to the input vector to determine an optimum. These methods exploit the fact that the gradient of the objective function $\boldsymbol{\nabla} J(\mathbf{u})$, at any value of the input $\mathbf{u}$, will always point in the direction of maximal increase of $J(\mathbf{u})$ [38]. As such, by continuously moving in the direction of this gradient, a local optimum $\mathbf{u}^*$ can be found, for which $\boldsymbol{\nabla} J(\mathbf{u}^*) = \mathbf{0}$. However, as illustrated in figure 3.1, although such a vanishing gradient $\boldsymbol{\nabla} J(\mathbf{u}^*)$ would assure a local optimum, the established vector $\mathbf{u}^*$ might not necessarily correspond to the optimum over the entire domain.

The second class of optimization approaches concerns the gradient-free methods. Rather than relying on the gradient of the cost function, these methods use only the value of the function itself to determine an optimal input. As a result, they are generally more suitable for finding global optima than gradient-based methods. However, even obtaining an optimum through a gradient-free method, this solution is not guaranteed to be globally optimal. Moreover, in most cases, gradient-free methods are significantly more computationally demanding than gradient-free ones, requiring a large number of (often expensive) function evaluations before achieving a good result.

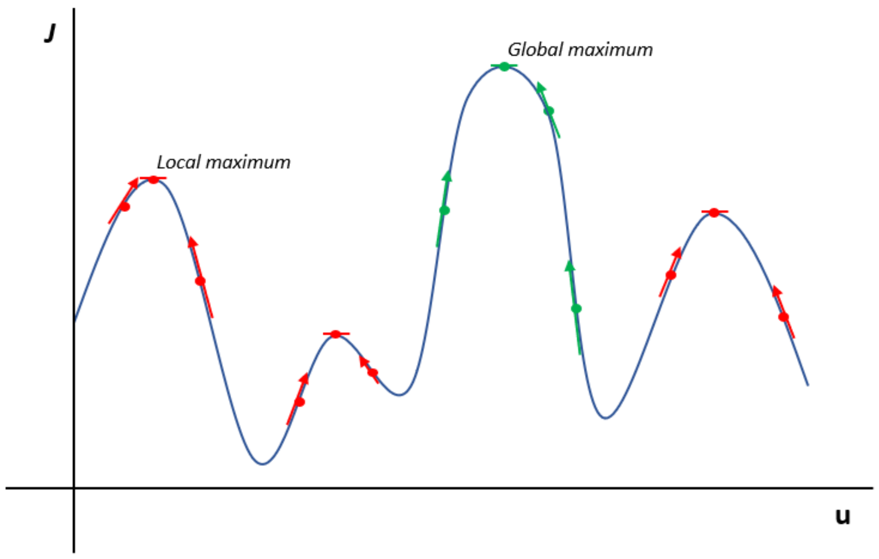


Figure 3.1: Global and local maxima of a curve

In this thesis, we investigate only gradient-based optimization, considering three different methods for computing this gradient. For more information on gradient-free methods, and recent developments in their applications in reservoir flow, we refer to [25] and respectively [8] and [23].

3.1.1 Iterative methods

Almost all numerical optimization methods work iteratively, establishing a new guess of the optimal input $\mathbf{u}^{(k+1)}$ based on an earlier guess $\mathbf{u}^{(k)}$, starting at a value $\mathbf{u}^{(0)}$. For a gradient-based optimization method, this new guess is computed simply as [31]:

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \boldsymbol{\rho}^{(k)} \tag{3.2}$$

In this notation, we define $\boldsymbol{\rho}^{(k)}$ to be the step taken at iteration $k+1$ of the optimization algorithm. The problem of optimizing the input then becomes that of optimizing the step at each iteration, solving:

$$\begin{aligned} & \max_{\boldsymbol{\rho}\in\mathbb{R}^{N_u}} \ J(\mathbf{u}^{(k)} + \boldsymbol{\rho}) & & \\ \text{subject to} \quad & \mathrm{u}_i^{(k)} + \rho_i \geq \mathrm{u}_{\min} & & i = 1, \dots, N_u \\ & \mathrm{u}_i^{(k)} + \rho_i \leq \mathrm{u}_{\max} & & i = 1, \dots, N_u \end{aligned} \tag{3.3}$$

To solve this problem, we consider two commonly used strategies, the first of which being the line search approach. This strategy decomposes the step $\boldsymbol{\rho}$ into a search direction $\mathbf{p}^{(k)}$ and step length α, such that $\boldsymbol{\rho} = \alpha\mathbf{p}^{(k)}$. Fixing the search direction $\mathbf{p}^{(k)}$ at the start of each iteration $k+1$, this approach then only optimizes the size of the step α taken along this direction, hence only aiming to maximize the cost function along a line at each iteration of the algorithm. Knowing that the gradient of the objective function always points in the direction of greatest increase, the search direction at each iteration is computed directly from this gradient:

$$\mathbf{p}^{(k)} = \mathbf{B}^{(k)}\boldsymbol{\nabla} J^{(k)} \tag{3.4}$$

Here $\boldsymbol{\nabla} J^{(k)} := \boldsymbol{\nabla} J(\mathbf{u}^{(k)})$ denotes the gradient of the objective function at the kth guess of the optimal input, and $\mathbf{B}^{(k)}$ is a matrix meant to improve the convergence rate of the algorithm. Popular choices for $\mathbf{B}$ include the inverse $(\boldsymbol{\nabla}^2 J^{(k)})^{-1}$ of the Hessian of the objective function at the corresponding input, yielding a Newton direction, or some approximation of this inverse Hessian, yielding a quasi-Newton direction. These directions are very reliable in case the objective function can be (locally) well approximated by its second-order Taylor polynomial, and an accurate Hessian can actually be computed. However, these methods are not applied in this thesis, and instead only a scaling is applied to the gradient. Specifically, at each iteration, the search direction is computed by dividing the gradient at the previous input by its L-infinity norm:

$$\mathbf{p}^{(k)} = \frac{\boldsymbol{\nabla} J^{(k)}}{\|\boldsymbol{\nabla} J^{(k)}\|_\infty} \tag{3.5}$$

This definition corresponds to a steepest ascent direction. By using an L-infinity norm, the step size α will correspond to the greatest change in the elements of $\mathbf{u}$, which is useful when imposing the physical restrictions on the input values.

The second strategy often used in solving problem (3.3) is the trust region approach. Rather than optimizing the objective function along a line, this strategy searches for an optimal step $\boldsymbol{\rho}^{(k)}$ within a specified area around the latest input $\mathbf{u}^{(k)}$, called a trust region. This region is usually specified by requiring the norm $\|\boldsymbol{\rho}\|$ of the step to be smaller than some constant $\Delta^{(k)}_{\text{trust}}$. In this region, we optimize the cost function $J(\mathbf{u}^k + \boldsymbol{\rho})$ by assuming it to closely match a model function $m^{(k)}(\boldsymbol{\rho})$, generally taken to be:

$$m^{(k)}(\boldsymbol{\rho}) = \mathbf{u}^{(k)} + \boldsymbol{\rho}^T \cdot \boldsymbol{\nabla} J^{(k)} + \frac{1}{2}\boldsymbol{\rho}^T \cdot \mathbf{B}^{(k)}\boldsymbol{\rho} \tag{3.6}$$

Here, once again, the matrix $\mathbf{B}^{(k)}$ is often chosen to be the Hessian of the objective function, or an approximation of it. For this thesis, however, this matrix is simply set equal to zero, approximating the cost function as a linear model:

$$m^{(k)}(\boldsymbol{\rho}) = \mathbf{u}^{(k)} + \boldsymbol{\rho}^T \cdot \boldsymbol{\nabla} J^{(k)} \tag{3.7}$$

This model increases most quickly along the direction of the gradient $\boldsymbol{\nabla} J^{(k)}$, growing with the distance α moved along this direction. Accordingly, the optimal step $\boldsymbol{\rho}$ under these conditions is simply $\alpha_{\max}\mathbf{p}^{(k)}$, with the search direction $\mathbf{p}^{(k)}$ being given by the normalized gradient (3.5), and $\alpha_{\max}$ denoting the maximal distance within the trust region that can be travelled in this direction. Taking this region to be bounded by an L-infinity norm, thereby comprising a hypercube around the current input $\mathbf{u}^{(k)}$, the maximal step size is simply equal to the size of the trust region. Consequently, the trust region method under these conditions is equivalent to a steepest ascent line search method, where the step size is taken equal to the size of the trust region. The remaining problem is then once more that of finding an appropriate step size to move in the computed search direction, establishing an optimal value along a line.

3.1.2 Wolfe condition

To determine an optimal step size $\alpha^{(k)}$ to move in the direction $\mathbf{p}^{(k)}$, an iterative procedure is applied. Such a method computes the value of the cost function for different estimates of an optimal input $\mathbf{u}^i = \mathbf{u}^{(k)} + \alpha^i \mathbf{p}^{(k)}$, adapting the step size α^i until a satisfactory result is attained. Naturally, this requires establishing criteria to indicate when an appropriate step size has been found. For this, we consider the first Wolfe condition, also known as the Armijo rule.

The Armijo rule, first introduced by Philip Wolfe in 1969 [40][41], provides a condition for establishing whether the increase in value of the objective function, corresponding to a certain step size in a line search method, is satisfactory. This condition exploits the fact that, approaching the input $\mathbf{u}^{(k)}$, the cost function will approximately match its first order Taylor approximation around this input:

$$J(\mathbf{u}^{(k)} + \alpha\mathbf{p}^{(k)}) \approx J(\mathbf{u}^{(k)}) + \alpha[\mathbf{p}^{(k)}]^T \cdot \boldsymbol{\nabla} J^{(k)} \qquad \text{for } \alpha \text{ sufficiently small} \tag{3.8}$$

As a result, for arbitrary $0 \leq \eta_{\mathrm{W}} < 1$, there exists a value α satisfying:

$$J(\mathbf{u}^{(k)} + \alpha\mathbf{p}^{(k)}) \geq J(\mathbf{u}^{(k)}) + \eta_{\mathrm{W}}\alpha[\mathbf{p}^{(k)}]^T \cdot \boldsymbol{\nabla} J^{(k)} \tag{3.9}$$

This is the first Wolfe condition, requiring the increase in value of the objective function to be greater than a fraction η_{W} of the expected increase $\alpha[\mathbf{p}^{(k)}]^T \cdot \boldsymbol{\nabla} J^{(k)}$. Imposing this condition assures not only that the new input improves the value of the cost function, but also that this improvement exceeds a certain threshold, the magnitude of which is determined by η_{W}. This value is generally taken to be of fairly small order, allowing for substantial freedom in the size of the step, whilst assuring reasonable improvements are achieved.

3.1.3 The search algorithm

Using the linear trust region method and employing the Armijo rule, we construct an algorithm for determining a new optimal input $\mathbf{u}^{(k+1)}$ based on some input $\mathbf{u}^{(k)}$ and corresponding search direction $\mathbf{p}^{(k)}$. For this input, we also assume a model function $m^{(k)}(\alpha\mathbf{p}^k)$ to be known, for example given by (3.7), assumed to be accurate within a trust region of size $\alpha_{\max}$. Denoting $\mathbf{u}^i = \mathbf{u}^{(k)} + \alpha^i \mathbf{p}^{(k)}$, we then iteratively adjust α^i until satisfying a Wolfe condition:

$$\frac{J(\mathbf{u}^i) - J(\mathbf{u}^{(k)})}{|m^{(k)}(\alpha^i\mathbf{p}^{(k)}) - m^{(k)}(\mathbf{0})|} \geq \eta_{\mathrm{W}} \tag{3.10}$$

This condition requires the increase in value of the objective function to be bigger than some fraction of the expected increase, being equivalent to the Armijo rule when using model (3.7). Assuming the search direction $\mathbf{p}^{(k)}$ to be pointing in some direction of increase of the objective function, and the model function to match the exact one at the input $\mathbf{u}^{(k)}$, this condition will always be satisfied for sufficiently small α. Accordingly, we can iteratively decrease our guess of the step size by a factor $\frac{1}{2}$ as $\alpha^{i+1} = \frac{1}{2}\alpha^i$ until condition (3.10) is satisfied. Here, we start with a guess $\alpha^1 = \alpha^{(k)}_{\max}$, equal to the greatest possible step in search direction $\mathbf{p}^{(k)}$ whilst remaining in the trust region.

To gain slightly more freedom in the optimization process, the step size is also allowed to increase under certain conditions. Specifically, we note that a model function such as (3.7) matches the exact cost function at $\mathbf{u}^{(k)}$ to first order. Therefore, if the value of the cost function exceeds that of the model function close to this input, the optimum will likely lie at a greater distance. As such, the step size will be increased by a factor $1\frac{1}{2}$ whenever an α^i is found satisfying:

$$\frac{J(\mathbf{u}^i) - J(\mathbf{u}^{(k)})}{|m^{(k)}(\alpha^i \mathbf{p}^{(k)}) - m^{(k)}(\mathbf{0})|} \geq 1 \tag{3.11}$$

Finally, constraints are imposed to assure additional iterations provide significant changes in the optimal input and corresponding cost function value, given as:

$$\frac{\|\mathbf{u}^{i+1} - \mathbf{u}^i\|}{\max\{1, \|\mathbf{u}^{i+1}\|\}} < \eta_{\mathrm{u}} \qquad\qquad \frac{\|J^{i+1} - J^i\|}{\max\{1, \|J^{i+1}\|\}} < \eta_J \tag{3.12}$$

If either of the conditions (3.10) or (3.12) is satisfied, or a maximum number N_{search} of search iterations has been performed, the input $\mathbf{u}^i = \mathbf{u}^{(k)} + \alpha^i \mathbf{p}^{(k)}$ yielding the greatest cost function value is retained. We summarize these steps in algorithm 1 below:

Algorithm 1: Line search optimization

```
    Input: Input $\mathbf{u}^{(k)}$, Cost value $J^{(k)} := J(\mathbf{u}^{(k)})$, Search direction $\mathbf{p}^{(k)}$, Model function
           $m^{(k)}(\boldsymbol{\rho})$, Trust region size $\alpha^{(k)}_{\max}$
    Result: New input $\mathbf{u}^{(k+1)}$
1   $\mathbf{u}^* = \mathbf{u}^{(k)}$
2   $J^* = J^{(k)}$
3   $\mathbf{u}^0 = \mathbf{u}^{(k)}$
4   $J^0 = J^{(k)}$
5   $i = 1;$
6   $\alpha^1 = \alpha^{(k)}_{\max}$
7   while $\left[i \leq N_{\text{search}}\right] \wedge \left[\frac{\|\boldsymbol{u}^i - \boldsymbol{u}^{i-1}\|}{\max\{1, \|\boldsymbol{u}^i\|\}} \geq \eta_{\mathrm{u}}\right] \wedge \left[\frac{\|J^i - J^{i-1}\|}{\max\{1, \|J^i\|\}} \geq \eta_J\right]$ do
8       $\mathbf{u}^i = \mathbf{u}^{(k)} + \alpha^i \mathbf{p}^{(k)}$
9       $J^i = J(\mathbf{u}^i);$
10      $m^i = m^{(k)}(\alpha^i \mathbf{p}^{(k)})$
11      if $J^{(i)} > J^*$ then
12          $\mathbf{u}^* = \mathbf{u}^i$
13          $J^* = J^i$
14      end
15      if $\frac{J^i - J^{(k)}}{|m^i - m^{(k)}(0)|} > 1$ then
16          $\alpha^{i+1} = 1\frac{1}{2}\alpha^i$
17      else if $\frac{J^i - J^{(k)}}{|m^i - m^{(k)}(0)|} < \eta_{\mathrm{W}}$ then
18          $\alpha^{i+1} = \frac{1}{2}\alpha^i$
19      else
20          Break
21      end
22  end
23  $\mathbf{u}^{(k+1)} = \mathbf{u}^*$
```

3.2 Gradient computation methods

Through algorithm 1, a locally optimal input can be established, improving an arbitrary guess of the input by moving in some direction of increase $\mathbf{p}$. This direction is based on the gradient $\boldsymbol{\nabla} J = \left[\frac{dJ}{d\mathbf{u}}\right]^T$ of the cost function at the previous input, using the fact that this gradient points in the direction of greatest increase. Computing this gradient, however, cannot always be achieved in a straightforward manner. Especially if the dependence of the gradient on the input is indirect, as is the case for the net present value described in the previous chapter, determining an exact gradient can be strenuous or even impossible.

In this section we consider three different techniques for computing a gradient of the objective function. The first of these, the adjoint method, shapes the basis of the POD-TPWL method that is developed in the following chapters. The other methods will be used solely for the purpose of comparison, and will therefore only be explained in a rudimentary manner. It is important to note that, numerically implementing either of these methods, the determined gradient at an arbitrary value of the input will only be an approximation to the actual gradient at this input. Nevertheless, assuming the angle between this approximation and the actual gradient to be less than 90^o, this approximation will still point in a direction of increase of the cost function. Therefore, implementing any such approximate gradient, the techniques from the previous section will still be applicable, albeit less effective than upon employing the exact gradient.

3.2.1 Adjoint method

When considering optimization on a system such as (2.27), the adjoint method is among the most popular for computing a gradient. This is because of the ability of this technique to directly take into account the complex dependence of the cost function on the input at each time, by incorporating the model explicitly into the cost function [21]. Specifically, let the cost function be of the form:

$$J = J(\mathbf{x}^1, \dots, \mathbf{x}^N, \mathbf{u}) \tag{3.13}$$

This implies that any variable on which the cost function depends, can be expressed solely in terms of the total input vector $\mathbf{u}$, and the states $\mathbf{x}^1, \dots, \mathbf{x}^N$. For the net present value described earlier, this assumption holds, as it can be described directly in terms of the outputs of system (2.27), which in turn depend only on the corresponding inputs and states. In particular, this dependence is expected to be imposed through a general set of equations:

$$\mathbf{0} = \mathbf{g}^{n+1}(\mathbf{x}^{n+1}, \mathbf{x}^n, \mathbf{u}) \qquad\qquad n = 0, \dots, N-1 \tag{3.14}$$

Requiring these equalities to be satisfied, we can introduce an equivalent cost function to (3.13) as:

$$\hat{J} := J + \sum_{n=1}^{N} [\boldsymbol{\lambda}^n]^T \mathbf{g}^n(\mathbf{x}^n, \mathbf{x}^{n-1}, \mathbf{u}) \tag{3.15}$$

In this function, $\boldsymbol{\lambda}^1, \dots, \boldsymbol{\lambda}^N \in \mathbb{R}^{N_x}$ are Lagrange multipliers, also denoted as adjoint vectors. Since the function $\mathbf{g}^{n+1}$ is required to be zero at each time step, the value of the adjusted cost function should be equal to that of the original cost function, independent of the value of the multipliers. Accordingly, the gradient of the original cost function will also match that of the adjusted one, for which the adjoint vectors can be suitably chosen. To determine this gradient, we consider a small perturbation $\delta\hat{J}$ in the value of the adjusted function. Since this function depends only on the input and the states, this perturbation too must be the result of one in either the input or the state. We find:

$$\begin{aligned}
\delta\hat{J} &= \delta J + \sum_{n=1}^{N} [\boldsymbol{\lambda}^n]^T \delta\mathbf{g}^n(\mathbf{x}^n, \mathbf{x}^{n-1}, \mathbf{u}) \\
&= \sum_{n=1}^{N} \frac{\partial J}{\partial \mathbf{x}^n}\delta\mathbf{x}^n + \frac{\partial J}{\partial \mathbf{u}}\delta\mathbf{u} + \sum_{n=1}^{N} [\boldsymbol{\lambda}^n]^T \left[\frac{\partial \mathbf{g}^n}{\partial \mathbf{x}^n}\delta\mathbf{x}^n + \frac{\partial \mathbf{g}^n}{\partial \mathbf{x}^{n-1}}\delta\mathbf{x}^{n-1} + \frac{\partial \mathbf{g}^n}{\partial \mathbf{u}}\delta\mathbf{u}\right] \\
&= \sum_{n=1}^{N} \frac{\partial J}{\partial \mathbf{x}^n}\delta\mathbf{x}^n + \frac{\partial J}{\partial \mathbf{u}}\delta\mathbf{u} + \sum_{n=1}^{N} [\boldsymbol{\lambda}^n]^T \frac{\partial \mathbf{g}^n}{\partial \mathbf{x}^n}\delta\mathbf{x}^n + \sum_{n=0}^{N-1} [\boldsymbol{\lambda}^{n+1}]^T \frac{\partial \mathbf{g}^{n+1}}{\partial \mathbf{x}^n}\delta\mathbf{x}^n + \sum_{n=1}^{N} [\boldsymbol{\lambda}^n]^T \frac{\partial \mathbf{g}^n}{\partial \mathbf{u}}\delta\mathbf{u} \\
&= \sum_{n=1}^{N-1} \left[\frac{\partial J}{\partial \mathbf{x}^n} + [\boldsymbol{\lambda}^n]^T \frac{\partial \mathbf{g}^n}{\partial \mathbf{x}^n} + [\boldsymbol{\lambda}^{n+1}]^T \frac{\partial \mathbf{g}^{n+1}}{\partial \mathbf{x}^n}\right]\delta\mathbf{x}^n + \left[\frac{\partial J}{\partial \mathbf{x}^N} + [\boldsymbol{\lambda}^N]^T \frac{\partial \mathbf{g}^N}{\partial \mathbf{x}^N}\right]\delta\mathbf{x}^N \\
&+ \left[[\boldsymbol{\lambda}^1]^T \frac{\partial \mathbf{g}^1}{\partial \mathbf{x}^0}\right]\delta\mathbf{x}^0 + \left[\frac{\partial J}{\partial \mathbf{u}} + \sum_{n=1}^{N} [\boldsymbol{\lambda}^n]^T \frac{\partial \mathbf{g}^n}{\partial \mathbf{u}}\right]\delta\mathbf{u} \qquad (3.16)
\end{aligned}$$

Since the initial state $\mathbf{x}^0 = \mathbf{x}^{\text{init}}$ is fixed, no perturbations in this state are possible, such that the term $\left[[\boldsymbol{\lambda}^1]^T \frac{\partial \mathbf{g}^1}{\partial \mathbf{x}^0}\right]\delta\mathbf{x}^0$ vanishes. This leaves only perturbations in the input, as well as in the states at remaining times $n = 1, \ldots, N$. Now, in order to obtain a full derivative of the cost function with respect to the input, we need to express the perturbation of the cost function solely in terms of that of the input. To this end, the other perturbations of the state are also set to zero, by requiring the Lagrange multipliers to satisfy an adjoint model:

$$\begin{aligned}
\left[\frac{\partial \mathbf{g}^n}{\partial \mathbf{x}^n}\right]^T \boldsymbol{\lambda}^n &= -\left[\frac{\partial \mathbf{g}^{n+1}}{\partial \mathbf{x}^n}\right]^T \boldsymbol{\lambda}^{n+1} - \left[\frac{\partial J}{\partial \mathbf{x}^n}\right]^T \qquad n = N-1, \ldots, 1 \\
\frac{\partial \mathbf{g}^N}{\partial \mathbf{x}^N}\boldsymbol{\lambda}^N &= -\left[\frac{\partial J}{\partial \mathbf{x}^N}\right]^T \qquad (3.17)
\end{aligned}$$

If the partial derivatives of $\mathbf{g}$ and J with respect to the state at each time are known, this model can be solved backwards in time. This provides the adjoint vectors $\boldsymbol{\lambda}^n$ such that the $\delta\mathbf{x}^n$ terms in the perturbed cost function vanish, allowing δJ to be expressed directly in terms of $\delta\mathbf{u}$:

$$\delta\hat{J} = \left[\frac{\partial J}{\partial \mathbf{u}} + \sum_{n=1}^{N} [\boldsymbol{\lambda}^n]^T \frac{\partial \mathbf{g}^n}{\partial \mathbf{u}}\right]\delta\mathbf{u} \qquad (3.18)$$

In this perturbation, any effects of the state vector on the objective function are incorporated through the adjoint vectors, required to satisfy the system (3.17). In this manner, proper choice of the Lagrange multipliers allows perturbations in the cost function to be expressed solely in terms of perturbations in the input vector. Letting the size of these perturbation approach zero, the total derivative of the cost function with respect to the inputs can then be seen to be:

$$\frac{dJ}{d\mathbf{u}} = \frac{d\hat{J}}{d\mathbf{u}} = \frac{\partial J}{\partial \mathbf{u}} + \sum_{n=1}^{N} \left[[\boldsymbol{\lambda}^n]^T \frac{\partial \mathbf{g}^n}{\partial \mathbf{u}}\right] \qquad (3.19)$$

This method thus allows us to describe the full derivative of the cost function with respect to the input, in terms of the partial derivative, and a partial derivative of the model function.

When applied to production optimization, we can express the cost function as a linear function of the outputs:

$$J = \sum_{n=1}^{N} [\mathbf{c}^n]^T \cdot \mathbf{h}^n(\mathbf{x}^n, \mathbf{u}^n) \tag{3.20}$$

Here, the function $\mathbf{h}$ provides the output to system (2.27), in terms of the states and inputs at each time. These states and inputs are required to satisfy an equality (3.21), where the function $\mathbf{g}^{n+1}(\mathbf{x}^{n+1}, \mathbf{x}^n, \mathbf{u})$ is of the form:

$$\mathbf{g}^{n+1}(\mathbf{x}^{n+1}, \mathbf{x}^n, \mathbf{u}) = \mathbf{x}^{n+1} - \mathbf{f}^{n+1}(\mathbf{x}^n, \mathbf{x}^{n+1}, \mathcal{K}^{n+1}(\mathbf{u})) \qquad n = 0, \dots, N-1 \tag{3.21}$$

Here, $\mathcal{K} : \mathbb{R}^{N \cdot N_\mathrm{u}} \rightarrow \mathbb{R}^{N_\mathrm{u}}$ is a function translating the full input vector $\mathbf{u}$ back into an individual input at each time step. Based on this definition, the partial derivatives are then given by:

$$\begin{aligned}
\frac{\partial \mathbf{g}^{n+1}}{\partial \mathbf{u}} &= -\frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{u}} & \frac{\partial J}{\partial \mathbf{u}} &= \sum_{n=1}^{N} [\mathbf{c}^n]^T \cdot \left[\frac{\partial \mathbf{h}^n}{\partial \mathbf{u}}\right] \\
\frac{\partial \mathbf{g}^{n+1}}{\partial \mathbf{x}^n} &= -\frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}^n} & \frac{\partial J}{\partial \mathbf{x}^n} &= [\mathbf{c}^n]^T \cdot \left[\frac{\partial \mathbf{h}^n}{\partial \mathbf{x}^n}\right] \\
\frac{\partial \mathbf{g}^n}{\partial \mathbf{x}^n} &= \mathbf{I} - \frac{\partial \mathbf{f}^n}{\partial \mathbf{x}^n}
\end{aligned} \tag{3.22}$$

In this last equation, $\mathbf{I} \in \mathbb{R}^{N_\mathrm{x} \times N_\mathrm{x}}$ denotes an identity matrix of appropriate size. Substituting these derivatives, the adjoint model in this case is found to be:

$$\begin{aligned}
\left[\mathbf{I} - \frac{\partial \mathbf{f}^n}{\partial \mathbf{x}^n}\right]^T \boldsymbol{\lambda}^n &= \left[\frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}^n}\right]^T \boldsymbol{\lambda}^{n+1} - \left[\frac{\partial \mathbf{h}^n}{\partial \mathbf{x}^n}\right]^T \cdot [\mathbf{c}^n] \qquad n = N-1, \dots, 1 \\
\left[\mathbf{I} - \frac{\partial \mathbf{f}^N}{\partial \mathbf{x}^N}\right]^T \boldsymbol{\lambda}^N &= -\left[\frac{\partial \mathbf{h}^N}{\partial \mathbf{x}^N}\right]^T \cdot [\mathbf{c}^N]
\end{aligned} \tag{3.23}$$

Running this system backwards in time, adjoint vectors can be determined, at which point a resulting gradient can be computed as:

$$\frac{dJ}{d\mathbf{u}} = \sum_{n=1}^{N} \left([\mathbf{c}^n]^T \cdot \left[\frac{\partial \mathbf{h}^n}{\partial \mathbf{u}}\right] - [\boldsymbol{\lambda}^n]^T \left[\frac{\partial \mathbf{f}^n}{\partial \mathbf{u}}\right] \right) \tag{3.24}$$

In case the analytical partial derivatives of the objective function and model equations are implemented, the gradient as computed using the adjoint method will be exact. In many cases, however, numerical approximations have to be made, as the exact derivatives or even the actual model equations are unknown. Additionally, if the size N_x of the state vector, and thus also adjoint vector, is large, storing the derivative matrices and adjoint states at each time step can be computationally infeasible. For these reasons, we consider the implementation of a reduced-order surrogate model, described in consecutive chapters.

3.2.2 Finite difference method

In the field of derivative approximations, the finite difference method is perhaps the best known and most commonly applied. It forms the basis of many derivative discretization schemes, including the porous media model as discussed in the previous chapter. This is mainly due to the high accuracy and ease of implementation of this method, allowing for practically any differential equation to be numerically approximated. However, for the purpose of optimization on the basis of a system such as (2.27), this method can be computationally very expensive, making it practically infeasible in many situations.

The concept of the finite difference approximation emerges directly in the definition of the derivative. Letting $J(\mathrm{u})$ be a function dependent on a single variable u, this derivative at value $\mathrm{u} = a$ is given by:

$$\left.\frac{dJ}{d\mathrm{u}}\right|_{u=a} = \lim_{\epsilon\to 0} \frac{J(a+\epsilon) - J(a)}{\epsilon} \tag{3.25}$$

From this definition, a straightforward approximation of the derivative follows by considering just a finite step $\epsilon > 0$:

$$\left.\frac{dJ}{d\mathrm{u}}\right|_{\mathrm{u}=a} \approx \frac{J(a+\epsilon) - J(a)}{\epsilon} \qquad \text{for } \epsilon > 0 \text{ small} \tag{3.26}$$

This is a one-sided (forward) finite difference approximation of the derivative of an arbitrary function. A centered equivalent of this method follows directly as:

$$\left.\frac{dJ}{d\mathrm{u}}\right|_{\mathrm{u}=a} \approx \frac{J(a+\epsilon) - J(a-\epsilon)}{2\epsilon} \qquad \text{for } \epsilon > 0 \text{ small} \tag{3.27}$$

When considering a higher dimensional input $\mathbf{u} \in \mathbb{R}^{N_\mathrm{u}}$, either of these approximations can be applied to compute individual derivatives $\frac{dJ}{d\mathrm{u}_i}$, which can then be combined to yield an approximate gradient $\boldsymbol{\nabla} J$. In our implementation, only the one-sided approximation will be used, noting that the centered scheme requires almost twice the number of evaluations of the cost function in higher dimensions.

The accuracy of this approximation generally increases as the size of the perturbation $\epsilon > 0$ decreases, approaching the limit by which the derivative is defined. Nevertheless, when aiming to maximize the function J, a larger step size can help establish a (more) global optimum, or speed up the method, making the perturbation size a crucial parameter in the optimization procedure.

3.2.3 Ensemble method

The final method for gradient computation considered in this thesis is the ensemble optimization (EnOpt) methodology. Similarly to the finite difference scheme, this method determines an approximate gradient based on samples of the inputs and the corresponding cost function. Unlike the finite difference scheme, however, only a relatively small number of samples are generally used for this approximation, independent of the size of the input vector. Therefore, the EnOpt method provides an excellent comparison for the POD-TPWL scheme developed in the next chapters, which also computes a gradient based on a limited number of samples.

The ensemble method is one of several variations on the simultaneous perturbation stochastic approximation (SPSA) methodology [11]. For such a method, rather than perturbing the input in a predefined manner, as done for the finite difference approximation, a random perturbation $\delta\mathbf{u}^j$ in the input is considered. In general, such a perturbation is generated from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{C}_{\mathrm{u}})$, with $\mathbf{C}_{\mathrm{u}}$ denoting some covariance matrix for the input perturbations. Starting at a current best guess $\mathbf{u}_b$ for an optimal input, we can consider a new sample input $\mathbf{u}^j := \mathbf{u}_b + c\delta\mathbf{u}^j$ as a perturbation of the best guess, where c denotes the perturbation size. Based on the first order Taylor polynomial around the current optimum $\mathbf{u}_b$, we can then approximate the cost function in the direction of perturbation $\delta\mathbf{u}^j$ as:

$$J(\mathbf{u}_b + c\delta\mathbf{u}^j) = J(\mathbf{u}_b) + c[\delta\mathbf{u}^j]^T \boldsymbol{\nabla} J(\mathbf{u}_b) + \mathcal{O}(c^2) \tag{3.28}$$

Dropping the higher order terms, we can thus approximate the gradient as:

$$\boldsymbol{\nabla} J(\mathbf{u}_b) \approx \frac{J(\mathbf{u}_b + c\delta\mathbf{u}^j) - J(\mathbf{u}_b)}{c[\delta\mathbf{u}^j]^T} \tag{3.29}$$

This is effectively the finite difference gradient, based on a single perturbation, where all elements of $\mathbf{u}_b$ are perturbed simultaneously. For the SPSA method, however, this gradient is smoothed using the covariance matrix $C_{\mathbf{u}}$, or more generally an estimate $\delta\mathbf{u}[\delta\mathbf{u}]^T$ of this matrix. Multiplying the gradient with this matrix returns a smoothed approximation:

$$C_{\mathbf{u}} \boldsymbol{\nabla} J(\mathbf{u}_b) \approx \left(\delta\mathbf{u}^j[\delta\mathbf{u}^j]^T\right) \frac{J(\mathbf{u}_b + c\delta\mathbf{u}^j) - J(\mathbf{u}_b)}{c[\delta\mathbf{u}^j]^T} = \delta\mathbf{u}^j \frac{J(\mathbf{u}_b + c\delta\mathbf{u}^j) - J(\mathbf{u}_b)}{c} \tag{3.30}$$

Naturally, computing the gradient based on the single perturbation $\delta\mathbf{u}^j$ will not produce a reliable result. Instead, a set of N_e perturbations $\delta\mathbf{u}^1, \ldots, \delta\mathbf{u}^{N_e}$ is generated, and a gradient vector is established as the average of the separate smoothed approximations:

$$C_{\mathbf{u}} \boldsymbol{\nabla} J(\mathbf{u}_b) \approx \sum_{j=1}^{N_e} \delta\mathbf{u}^j \frac{J(\mathbf{u}_b + c\delta\mathbf{u}^j) - J(\mathbf{u}_b)}{c} \tag{3.31}$$

This provides the singly-smoothed G-SPSA search direction [15]. Additional smoothing can be applied by once more pre-multiplying with the covariance $C_{\mathbf{u}}$ or $\delta\mathbf{u}[\delta\mathbf{u}]^T$, returning an estimate of the vector $C_{\mathbf{u}}^2 \boldsymbol{\nabla} J$ at the input $\mathbf{u}_b$.

For the EnOpt gradient, the constant c is taken equal to 1, and the factor $\frac{1}{N_e}$ is replaced by $\frac{1}{N_e - 1}$. This provides an unbiased estimate of the cross-covariance $C_{\mathbf{u},J}$ between the input and the cost function, as [11]:

$$C_{\mathbf{u},J} = \mathbb{E}\left[\frac{1}{N_e - 1} \sum_{j=1}^{N_e} \delta\mathbf{u}^j \left(J(\mathbf{u}_b + c\delta\mathbf{u}^j) - J(\mathbf{u}_b)\right)\right] \tag{3.32}$$

This is the singly-smoothed EnOpt search direction, once more providing an estimation of the smoothed gradient $C_{\mathbf{u}} \boldsymbol{\nabla} J$ at $\mathbf{u}_b$. As for the G-SPSA gradient, a doubly-smoothed version of the EnOpt search direction can also be constructed, pre-multiplying this vector with (an estimate of) the covariance $C_{\mathbf{u}}$. Alternatively, a straight gradient implementation can be used, estimating the actual gradient $\boldsymbol{\nabla} J$ by multiplying the search direction with an estimated inverse of $C_{\mathbf{u}}$:

$$\boldsymbol{\nabla} J \approx \left(\sum_{j=1}^{N_e} \delta\mathbf{u}^j[\delta\mathbf{u}^j]^T\right)^{-1} \sum_{j=1}^{N_e} \delta\mathbf{u}^j \left(J(\mathbf{u}_b + c\delta\mathbf{u}^j) - J(\mathbf{u}_b)\right) = (\mathbf{U}\mathbf{U}^T)^{-1}\mathbf{U}\mathbf{J} \tag{3.33}$$

In this notation, matrix $\mathbf{U} \in \mathbb{R}^{N_{\mathrm{u}} \times N_e}$ consists column-wise of the perturbations $\delta\mathbf{u}^j$, with the elements of vector $\mathbf{J} \in \mathbb{R}^{N_e}$ denoting the corresponding perturbations in the cost function. Throughout this thesis, this approximation is considered as the ensemble method gradient, and provides the search direction in the corresponding implementation.

Chapter 4

Principal Orthogonal Decomposition

In this chapter, the theory behind principal orthogonal decomposition is described. Various choices regarding the implementation of this method are specified, as well as the numerical experiments driving several of these choices. In addition, the domain decomposition method is introduced, as applied in the context of model order reduction.

4.1 Model order reduction

We consider once more an arbitrary constrained optimization problem:

$$\begin{aligned} & \max_{\mathbf{u}\in\mathbb{R}^{N_u}} J(\mathbf{u}) & & \\ \text{subject to} \quad & \mathrm{u}_i \geq \mathbf{u}_{\min} & & i = 1,\ldots,N_u \\ & \mathrm{u}_i \leq \mathbf{u}_{\max} & & i = 1,\ldots,N_u \end{aligned} \tag{4.1}$$

Using the adjoint method, an accurate gradient $\boldsymbol{\nabla} J = \left[\frac{dJ}{d\mathbf{u}}\right]^T$ of the cost function $J(\mathbf{u})$ can be computed at arbitrary values of the input $\mathbf{u}$, at which point a search algorithm can be implemented to determine an optimum. However, in order to establish a gradient through the adjoint method, the exact dependence of the cost function on the input must be considered, which we assume to be through a system of the form:

$$\begin{aligned} \mathbf{x}^{n+1} &= \mathbf{f}^{n+1}(\mathbf{x}^n, \mathbf{x}^{n+1}, \mathbf{u}^{n+1}) & & \\ \mathbf{y}^{n+1} &= \mathbf{h}^{n+1}(\mathbf{x}^{n+1}, \mathbf{u}^{n+1}) & & n = 0,\ldots,N-1 \end{aligned} \tag{4.2}$$

In computing the gradient of the cost function, the adjoint method requires computing and storing the derivative matrices $\frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}^n}, \frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}^{n+1}} \in \mathbb{R}^{N_\mathrm{x}\times N_\mathrm{x}}$ and $\frac{d\mathbf{h}^{n+1}}{d\mathbf{x}^{n+1}} \in \mathbb{R}^{N_\mathrm{y}\times N_\mathrm{x}}$ at each simulation step. However, as the sizes of the first of these already increase quadratically with the size N_x of the state vector, the computational storage requirements of this method increase rapidly with the dimensions of the model. Consequently, applying the adjoint method in its full form is not always feasible, particularly when considering large-scale systems such as those encountered in reservoir simulation. Instead, this technique is often combined with a model order reduction scheme, computing a gradient based on a lower-dimensional model.

Through model order reduction, a lower-dimensional approximation of a large-scale model is constructed, aiming to provide an accurate representation of the full system at a reduced computational cost. In general, this amounts to establishing an invertible mapping $\mathbf{\Pi} : \mathbb{R}^{N_\text{x}} \to \mathbb{R}^{\tilde{N}_\text{x}}$, transforming the states and corresponding functions to lower dimensional equivalents and back. A multitude of methods have been developed to this end, each aiming to minimize the size $\tilde{N}_\text{x}$ of the reduced-order states, whilst also minimizing the discrepancy between the original states and their reduced-order approximations. In the field of reservoir flow, proper orthogonal decomposition (POD) is by far the most popular of these methods, and therefore also the focus of this study. For more information on alternative methods for model order reduction, we refer to [37].

4.1.1 State projection

At its core, principal orthogonal decomposition is a projection technique, representing the state vector under the basis of some reduced subspace. Specifically, if we let $\mathbb{R}^{N_\text{x}}$ denote the space in which the states $\mathbf{x}$ exist, the POD method searches for an optimal subspace $\mathcal{S} \subset \mathbb{R}^{N_{tnx}}$, minimizing the distance between observed states $\mathbf{x}$ and their projections $\mathbf{\Pi}_\mathcal{S}\mathbf{x}$ onto this subspace [34]. The matrix $\mathbf{\Pi}_\mathcal{S} \in \mathbb{R}^{N_\text{x} \times N_\text{x}}$ here is a projection matrix onto $\mathcal{S}$, such that $\mathbf{\Pi}_\mathcal{S}\mathbf{x}$ provides the closest vector along $\mathcal{S}$ to the vector $\mathbf{x}$, as illustrated for $\mathcal{S} = \mathbb{R}^2$ in figure 4.1. Such a projection matrix must always satisfy $\mathbf{\Pi}_\mathcal{S}^2 = \mathbf{\Pi}_\mathcal{S}$, as applying the projection matrix twice should return the same vector, but for POD we further require this matrix to be symmetric, such that $\mathbf{\Pi}_\mathcal{S} = \mathbf{\Pi}_\mathcal{S}^T$. Imposing this condition, the range $\mathcal{R}(\mathcal{S})$ and kernel $\mathcal{N}(\mathcal{S})$ of the subspace $\mathcal{S}$ will be orthogonal, and we call $\mathbf{\Pi}_\mathcal{S}$ an orthogonal projection [30].

Considering an orthogonal projection $\mathbf{\Pi}_\mathcal{S}$ onto some subspace $\mathcal{S} \subset \mathbb{R}^{N_\text{x}}$, we can express the projection in terms of an orthonormal basis of this subspace. Indeed, if we let $\mathbf{\Phi} \in \mathbb{R}^{N_\text{x} \times \tilde{N}_\text{x}}$ denote a matrix whose columns are orthonormal and span $\mathcal{S}$, we find that $\boldsymbol{\psi} := \mathbf{\Phi}^T\mathbf{x} = \mathbf{\Phi}^{-1}\mathbf{x}$ yields the components of state $\mathbf{x}$ under basis vectors $\mathbf{\Phi}$. Multiplying these components with the corresponding basis vectors, we then attain the projection of $\mathbf{x}$ onto $\mathcal{S}$ as:

$$\mathbf{\Pi}_\mathcal{S}\mathbf{x} = \mathbf{\Phi}\boldsymbol{\psi} = \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x} \tag{4.3}$$

A projection of this form $\mathbf{\Pi}_\mathcal{S} := \Phi\Phi^T$ is called a Galerkin projection [1], and can be easily seen to satisfy both the projection property $\mathbf{\Pi}_\mathcal{S} = \mathbf{\Pi}_\mathcal{S}^2$ and the orthogonality property $\mathbf{\Pi}_\mathcal{S} = \mathbf{\Pi}_\mathcal{S}^T$.

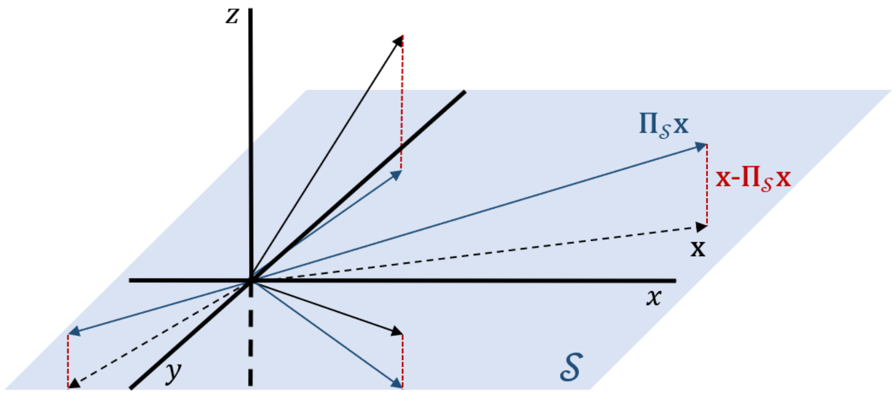

z
$\Pi_\mathcal{S}\mathbf{x}$
$\mathbf{x}$-$\Pi_\mathcal{S}\mathbf{x}$
$\mathbf{x}$
x
y
$\mathcal{S}$

Figure 4.1: Projection of vectors in $\mathbb{R}^3$ onto $\mathcal{S} = \mathbb{R}^2$

Splitting the projection in this manner, the problem of optimizing it becomes that of finding an optimal basis $\mathbf{\Phi} \in \mathbb{R}^{N_\text{x} \times \tilde{N}_\text{x}}$ of reduced size $\tilde{N}_\text{x} \leq N_\text{x}$, minimizing discrepancies between an arbitrary state $\mathbf{x}$ and its projection $\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}$.

4.1.2 Error minimization

To establish an optimal basis $\mathbf{\Phi}$, the POD method considers a set of N_S states $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N_S)}$. These states are collected by running the full-order model for a small number of different input vectors, and are assumed to be representative of the states the model generally produces. Accordingly, the squared error $\|\mathbf{x}^{(i)} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}^{(i)}\|_2^2$ in representing these states under a basis $\mathbf{\Phi}$, provides a measure of quality of this basis. Therefore, the problem of establishing an optimal basis $\mathbf{\Phi}$ can be given as a minimization problem:

$$\min \frac{1}{N_S} \sum_{i=1}^{N_S} \|\mathbf{x}^{(i)} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}^{(i)}\|_2^2 \tag{4.4}$$

This problem can be rewritten exploiting the orthogonality of the projection, assuring the projection $\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x} \in \mathcal{R}(\mathcal{S})$ and the remainder $\mathbf{x} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x} \in \mathcal{N}(\mathcal{S})$ to be orthogonal for arbitrary $\mathbf{x}$:

$$\begin{aligned}
\|\mathbf{x} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}\|_2^2 &= \left[\mathbf{x} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}\right]^T \left[\mathbf{x} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}\right] \\
&= \left[\mathbf{x}\right]^T \left[\mathbf{x} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}\right] - \left[\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}\right]^T \left[\mathbf{x} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}\right] \\
&= \left[\mathbf{x}\right]^T \left[\mathbf{x}\right] - \left[\mathbf{x}\right]^T \left[\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}\right] \\
&= \|\mathbf{x}\|_2^2 - \|\mathbf{\Phi}^T\mathbf{x}\|_2^2
\end{aligned} \tag{4.5}$$

Substituting this result, and removing the constant values $\|\mathbf{x}^{(i)}\|_2^2$, the problem (4.4) of determining an optimal basis becomes:

$$\max \sum_{i=1}^{N_S} \|\mathbf{\Phi}^T\mathbf{x}^{(i)}\|_2^2 \tag{4.6}$$

Note that, since $\mathbf{\Phi}^T\mathbf{x}^{(i)} = \mathbf{\Phi}^{-1}\mathbf{x}^{(i)}$ provides the components of vector $\mathbf{x}^{(i)}$ under basis $\mathbf{\Phi}$, this optimization problem suggests the basis must be chosen such that the norms of these component vectors are maximized, as would intuitively be expected.

To solve the maximization problem (4.6), we impose the additional condition that the basis vectors are normalized. Letting $\tilde{N}_\text{x}$ denote the size of the basis $\mathbf{\Phi}$, we then require $\|\boldsymbol{\phi}_j\|_2^2 = 1$ for $j = 1, \dots, \tilde{N}_\text{x}$, with $\boldsymbol{\phi}_j$ denoting column j of $\mathbf{\Phi}$. To enforce this condition, we introduce Lagrangian multipliers $\lambda_1, \dots, \lambda_{\tilde{N}_\text{x}}$, transforming the optimization problem into that of maximizing a cost function:

$$
\begin{aligned}
L(\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_{\tilde{N}_\mathrm{x}}) &:= \sum_{i=1}^{N_S} \|\boldsymbol{\Phi}^T \mathbf{x}^{(i)}\|_2^2 + \sum_{j=1}^{\tilde{N}_\mathrm{x}} \lambda_j (1 - \|\boldsymbol{\phi}_j\|_2^2) \\
&= \sum_{i=1}^{N_S} \sum_{j=1}^{\tilde{N}_\mathrm{x}} ([\mathbf{x}^{(i)}]^T \boldsymbol{\phi}_j)(\boldsymbol{\phi}_j^T \mathbf{x}^{(i)}) + \sum_{j=1}^{\tilde{N}_\mathrm{x}} \lambda_j - \lambda_j \boldsymbol{\phi}_j^T \boldsymbol{\phi}_j \\
&= \sum_{j=1}^{\tilde{N}_\mathrm{x}} \left[\sum_{i=1}^{N_S} (\boldsymbol{\phi}_j^T \mathbf{x}^{(i)})([\mathbf{x}^{(i)}]^T \boldsymbol{\phi}_j) - \lambda_j \boldsymbol{\phi}_j^T \boldsymbol{\phi}_j + \lambda_j \right] \\
&= \sum_{j=1}^{\tilde{N}_\mathrm{x}} \left[\boldsymbol{\phi}_j^T \mathbf{X}\mathbf{X}^T \boldsymbol{\phi}_j - \lambda_j \boldsymbol{\phi}_j^T \boldsymbol{\phi}_j + \lambda_j \right] \qquad (4.7)
\end{aligned}
$$

In this last step, we introduce the snapshot matrix $\mathbf{X} \in \mathbb{R}^{N_\mathrm{x} \times N_S}$, composed column-wise of the collected states $\mathbf{x}^{(i)}$. Maximizing the resulting functional with respect to some vector $\boldsymbol{\phi}_j$, the derivative with respect to this vector must be equal to zero:

$$
0 = \frac{\partial L}{\partial \boldsymbol{\phi}_j} = \boldsymbol{\phi}_j^T \mathbf{X}\mathbf{X}^T - \lambda_j \boldsymbol{\phi}_j^T \qquad j = 1, \dots, \tilde{N}_\mathrm{x} \qquad (4.8)
$$

For each basis vector, we then attain an eigenvalue problem:

$$
\mathbf{X}\mathbf{X}^T \boldsymbol{\phi}_j = \lambda_j \boldsymbol{\phi}_j \qquad j = 1, \dots, \tilde{N}_\mathrm{x} \qquad (4.9)
$$

This suggests that the optimal orthogonal space $\mathcal{S}$ to project the state vectors onto, is in fact that spanned by the eigenvectors of the matrix $\mathbf{X}\mathbf{X}^T$. Moreover, imposing this equality, the functional L is simply given by the sum of the eigenvalues:

$$
L(\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_{\tilde{N}_\mathrm{x}}) = \sum_{j=1}^{\tilde{N}_\mathrm{x}} \lambda_j \qquad (4.10)
$$

Therefore, determining an optimal orthogonal basis $\boldsymbol{\Phi} \in \mathbb{R}^{N_\mathrm{x} \times \tilde{N}_\mathrm{x}}$ of size $\tilde{N}_\mathrm{x}$ to project the state vectors, is equivalent to solving the eigenvalue problem (4.9), and retaining only the eigenvectors corresponding to the $\tilde{N}_\mathrm{x}$ largest eigenvalues.

4.1.3 Singular value decomposition

For large N_x, the matrix $\mathbf{X}\mathbf{X}^T \in \mathbb{R}^{N_\mathrm{x} \times N_\mathrm{x}}$ will be of considerable size, and directly solving the eigenvalue problem (4.9) is undesirable. Instead, we consider the singular value decomposition of the snapshot matrix $\mathbf{X}$, decomposing it as:

$$
\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \qquad (4.11)
$$

Here, $\mathbf{U} \in \mathbb{R}^{N_\mathrm{x} \times N_\mathrm{x}}$ and $\mathbf{V} \in \mathbb{R}^{N_S \times N_S}$ are orthonormal square matrices. The matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{N_\mathrm{x} \times N_S}$ is a pseudo-diagonal matrix, of which the diagonal elements σ_j for $j = 1, \dots, \min\{N_\mathrm{x}, N_S\}$ are denoted as the singular values of the matrix $\mathbf{X}$, and arranged in order of decreasing absolute value along the diagonal. Performing this decomposition, the matrix $\mathbf{X}\mathbf{X}^T$ can be given as:

$$
\mathbf{X}\mathbf{X}^T = [\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T][\mathbf{V}\boldsymbol{\Sigma}^T\mathbf{U}^T] = \mathbf{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T\mathbf{U}^T = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T \qquad (4.12)
$$

This yields a diagonalization of the matrix $\mathbf{X}\mathbf{X}^T$, with $\mathbf{U}$ comprising the eigenvectors of $\mathbf{X}\mathbf{X}^T$, and the diagonal matrix $\mathbf{\Lambda} \in \mathbb{R}^{\min\{N_\mathrm{x},N_S\}\times\min\{N_\mathrm{x},N_S\}}$ providing the corresponding eigenvalues. Here, the jth element $\lambda_j = \sigma_j^2$ along the diagonal of $\mathbf{\Lambda}$, gives the jth greatest eigenvalue of $\mathbf{X}\mathbf{X}^T$, with the column $\boldsymbol{u}_j$ of $\mathbf{U}$ providing the corresponding eigenvector. Since the diagonal $\boldsymbol{\sigma}$ of $\mathbf{\Sigma}$ contains only $\min\{N_\mathrm{x}, N_S\}$ elements, attaining a full eigenvalue spectrum for the matrix $\mathbf{X}\mathbf{X}^T$ will require the number of snapshots N_S to exceed the size N_x of the state. In that case, the matrix $\mathbf{U}$ provides an optimal orthonormal basis for representing the state vectors. Moreover, as the eigenvectors in $\mathbf{U}$ are ordered in accordance with the decreasing eigenvalues along the diagonal of $\mathbf{\Lambda}$, an optimal reduced basis for $\tilde{N}_\mathrm{x} \leq N_\mathrm{x}$ can be given as:

$$\tilde{\mathbf{\Phi}} = \tilde{\mathbf{U}} := [\boldsymbol{u}_1, \dots, \boldsymbol{u}_{\tilde{N}_\mathrm{x}}] \tag{4.13}$$

Here $\tilde{\mathbf{U}} \in \mathbb{R}^{N_\mathrm{x}\times\tilde{N}_\mathrm{x}}$ consists only of the first $\tilde{N}_\mathrm{x}$ columns of $\mathbf{U}$, corresponding to the eigenvectors with largest eigenvalues. In this manner, an optimal reduced orthonormal basis can be acquired, using the singular value decomposition of the snapshot matrix.

4.1.4 Energy criterion

In constructing a reduced basis in the manner described above, the desired size of this basis has to be established first. To choose such a size, we return to the least squares problem this basis should solve, which was found to be equivalent to:

$$\min \frac{1}{N_S}\sum_{i=1}^{N_S} \|\mathbf{x}^{(i)} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{x}^{(i)}\|_2^2 = \min \frac{1}{N_S}\sum_{i=1}^{N_S}\left[\|\mathbf{x}^{(i)}\|_2^2 - \|\tilde{\mathbf{\Phi}}^T\mathbf{x}^{(i)}\|_2^2\right] \tag{4.14}$$

Choosing the basis to consist of scaled eigenvectors of $\mathbf{X}\mathbf{X}^T$, the function to be minimized becomes:

$$\begin{aligned}
\sum_{i=1}^{N_S}\left[\|\mathbf{x}^{(i)}\|_2^2 - \|\tilde{\mathbf{\Phi}}^T\mathbf{x}^{(i)}\|_2^2\right] &= \sum_{i=1}^{N_S}\left[\|\mathbf{x}^{(i)}\|_2^2 - \sum_{j=1}^{\tilde{N}_\mathrm{x}}([\mathbf{x}^{(i)}]^T\tilde{\boldsymbol{\phi}}_j)(\tilde{\boldsymbol{\phi}}_j^T\mathbf{x}^{(i)})\right] \\
&= \sum_{i=1}^{N_S}\|\mathbf{x}^{(i)}\|_2^2 - \sum_{j=1}^{\tilde{N}_\mathrm{x}}\tilde{\boldsymbol{\phi}}_j^T\mathbf{X}\mathbf{X}^T\tilde{\boldsymbol{\phi}}_j \\
&= \sum_{i=1}^{N_S}\|\mathbf{x}^{(i)}\|_2^2 - \sum_{j=1}^{\tilde{N}_\mathrm{x}}\lambda_j \\
&= \sum_{i=1}^{N_S}\|\mathbf{x}^{(i)}\|_2^2 - \|\tilde{\boldsymbol{\sigma}}\|_2^2
\end{aligned} \tag{4.15}$$

In this last step, we define $\tilde{\boldsymbol{\sigma}}$ to contain only the $\tilde{N}_\mathrm{x}$ largest absolute singular values of matrix $\mathbf{X}\mathbf{X}^T$. In this notation, the error is expressed in terms of the difference between the total norm of these snapshots, and that of the eigenvalues. If the dimension $\tilde{N}_\mathrm{x}$ of the reduced space equals the dimension N_x of the full space, this error should be equal to zero, as in this case the columns of $\mathbf{\Phi} = \tilde{\mathbf{\Phi}} \in \mathbb{R}^{N_\mathrm{x}\times N_\mathrm{x}}$ will span the full space $\mathbb{R}^{N_\mathrm{x}}$. This suggests that the total norm $\|\boldsymbol{\sigma}\|_2^2$, when considering the full spectrum of $\mathbf{X}\mathbf{X}^T$, should be equal to total norm of the snapshots, which we denote as the total energy:

$$E_{\text{tot}} := \sum_{j=1}^{N_x} \sigma_j^2 = \|\boldsymbol{\sigma}\|_2^2 = \sum_{i=1}^{N_S} \|\mathbf{x}^{(i)}\|_2^2 \tag{4.16}$$

A measure of the quality of basis $\tilde{\boldsymbol{\Phi}}$ in representing the snapshots can then be given in terms of the energy retained by the spectrum, defined as:

$$E_{\tilde{N}_{\text{x}}} := \sum_{j=1}^{\tilde{N}_x} \sigma_j^2 = \|\tilde{\boldsymbol{\sigma}}\|_2^2 \tag{4.17}$$

The value of this retained energy increases with the dimension $\tilde{N}_{\text{x}}$ of the reduced space, matching the total energy when $\tilde{N}_{\text{x}} = N_{\text{x}}$. Determining the amount of basis vectors to retain can thus be translated into a requirement upon the relative energy to be preserved. In many cases, a small relative change in energy will already yield a significant reduction in the number of basis vectors [43], allowing for the major dynamics of the model to be captured through a much lower dimensional state.

4.2 Domain decomposition

In addition to the proper orthogonal decomposition, a domain decomposition is implemented. Through this method, the computational domain, pertaining the grid blocks in which the pressure and saturation variables are to be computed, is divided into a certain number of subdomains, each containing a portion of the grid blocks. In our implementation, this decomposition serves a dual purpose. Primarily, it aims to reduce the complexity of the algorithm, by adjusting the correlation between the unknowns at different grid blocks, based on the subdomain in which they are located. This is useful in constructing a surrogate model, as will be described in the next chapter. In addition, applying a domain decomposition in performing the POD method also offers benefits. For this purpose, rather than constructing a single basis for the entire domain, reduced bases aree constructed individually for each subdomain, to be combined afterwards.

Let Ω denote the full computational domain, consisting of a total of N_g grid blocks. In implementing the domain decomposition, these grid blocks are divided over N_D disjunct subdomains Ω^d, such that $\Omega = \bigcup_{d=1}^{N_D} \Omega^d$. This amounts to dividing the grid blocks into N_D sets, each containing N_g^d cells such that $N_g = \sum_{d=1}^{N_D} N_g^d$, and in such a manner that no two domains share a grid block. For our purposes, this decomposition will only be performed in the horizontal (x, y) directions, as the vertical dimension of many reservoirs is comparatively small.

In each domain d, we can consider the oil pressure and water saturation values at the encompassed grid blocks as the state vector $\mathbf{x}^{d,n} \in \mathbb{R}^{N_{\text{x}}^d}$ at each time step n. Accordingly, we can construct snapshots matrices $\mathbf{X}^d \in \mathbb{R}^{N_{\text{x}}^d \times N_S}$ for each domain d, by collecting the different rows of a full snapshot matrix $\mathbf{X} \in \mathbb{R}^{N_{\text{x}} \times N_S}$ corresponding to the cells in this domain. Based on these snapshot matrices, orthogonal bases $\boldsymbol{\Phi}^d \in \mathbb{R}^{N_{\text{x}}^d \times N_{\text{x}}^d}$ corresponding to singular values $\boldsymbol{\sigma}^d \in \mathbb{R}^{N_{\text{x}}^d}$ can then be constructed in each subdomain, which can in turn be reduced to bases $\tilde{\boldsymbol{\Phi}}^d \in \mathbb{R}^{N_{\text{x}}^d \times \tilde{N}_{\text{x}}^d}$ corresponding to singular values $\tilde{\boldsymbol{\sigma}}^d \in \mathbb{R}^{\tilde{N}_{\text{x}}^d}$ according to an energy criterion. In this manner, the POD method can be applied separately to the different subdomains, based on a snapshot matrix for the full domain.

Applying the POD method separately to the subdomains has an advantage in terms of the overall number of samples that need to be taken. Indeed, dividing the state variables over N_D subdomains, the state vector in each domain will be of a smaller size $N_{\mathrm{x}}^d < N_{\mathrm{x}}$. Performing principal orthogonal decomposition on such reduced states, the required number of snapshots to represent the states will thus also be lower. As each snapshot of the full domain provides a snapshot for all of the subdomains, this requires a lower total number of snapshots to be collected. Since snapshots have to be computed by running the full-order model, which is assumed to be computationally expensive, this method thus allows a reduction in the computational cost.

Naturally, performing POD separately in the subdomains also comes with a drawback. In particular, since the size of the singular vector $\boldsymbol{\sigma}^d$ in each domain reduces when more domains are used, the relative number of basis vectors that must be retained to meet the same energy criterion will increase with the number of domains. For example, in a situation where $N_D = N_g$ subdomains are implemented, the complete reduced basis will be equivalent to the full basis, offering no reduction of the state dimension. In this sense, the efficacy of the POD method decreases with the number of subdomains over which it is divided.

4.3 Numerical implementation

4.3.1 Affine subspace

In our application of the POD method, snapshots have always been computed by running the full-order model for N_{POD} inputs $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(N_{\mathrm{POD}})}$, collecting the resulting states at all times N to attain a total of $N_S = N \cdot N_{\mathrm{POD}}$ snapshots. These inputs in turn were always computed by perturbing some background input $\mathbf{u}_b$, such that in general, the sample inputs would not be centered around $\mathbf{0}$. For this reason, rather than implementing a projection onto a subspace $\mathcal{S}$, a projection onto an affine subspace $\hat{\mathbf{x}} + \mathcal{S}$ has always been performed, for some vector $\hat{\mathbf{x}} \in \mathbb{R}^{N_{\mathrm{x}}}$. Such a projection can be given in terms of the original one as $\hat{\mathbf{x}} + \boldsymbol{\Pi}_S(\mathbf{x} - \hat{\mathbf{x}})$, and can thus be achieved simply by performing the POD method with a shifted snapshot matrix:

$$\mathbf{X} = \left[\Delta\mathbf{x}^{(1)}, \ldots, \Delta\mathbf{x}^{(N_S)}\right] := \left[(\mathbf{x}^{(1)} - \hat{\mathbf{x}}), \ldots, (\mathbf{x}^{(N_S)} - \hat{\mathbf{x}})\right] \tag{4.18}$$

The aim of this shift is to reduce the norm of the considered snapshots, which is beneficial for the numerical implementation. As such, a natural choice for the shift is simply the sample mean:

$$\hat{\mathbf{x}} := \frac{1}{N_S} \sum_{i=1}^{N_S} \mathbf{x}^{(i)} \tag{4.19}$$

Henceforth, any considered snapshot matrix will always be (implicitly) shifted according to a corresponding mean vector $\hat{\mathbf{x}}$, placing the mean over the columns of any such matrix at $\mathbf{0}$.

4.3.2 Sampling

In performing the POD method, snapshots of the state vector are acquired by running the full-order model using samples $\mathbf{u}^{(j)}$ of the input. These snapshots are meant to provide a good representation of the overall states attainable, using input functions within the computational domain. In our implementation, the POD method was applied to construct a gradient $\boldsymbol{\nabla} J$ of the cost function at some background input $\mathbf{u}_b$. Therefore, the reduced basis had to be particularly accurate in representing states corresponding to inputs around $\mathbf{u}_b$. For this reason, the input samples $\mathbf{u}^{(j)}$ were always constructed as random perturbation $\epsilon_{\text{POD}}\delta\mathbf{u}^{(j)}$ to the background input $\mathbf{u}_b$, where ϵ_{POD} denotes the perturbation size.

In the computed input perturbations, the factor ϵ_{POD} describes the size of the domain for which the POD method is desired to provide an accurate basis. Indeed, a smaller value of ϵ_{POD} places the samples closer to the current optimum $\mathbf{u}_b$, thus providing a good representation of the states corresponding to inputs close to the background input. On the other hand, a greater perturbation size will allow for representing a larger variety of possible states, at the cost of accuracy close to the background input. Therefore, the perturbation size has always been adjusted according to the region in which the optimum was expected to be located. That is, a large perturbation size was implemented for the initial optimization, but this size would be reduced as the scale at which an optimum was sought decreased.

To construct the actual perturbations $\delta\mathbf{u}^{(j)}$, random numbers were generated in the domain $[-\frac{1}{2}, \frac{1}{2}]$, to shape each element of the perturbation. Aiming to attain some spread in the resulting samples over the desired domain, Latin hypercube sampling was implemented. For each element $i \in \{1, \dots, N_{\text{u}}\}$ of $\mathbf{u}_b$, this method divides the domain $[-\frac{1}{2}, \frac{1}{2}]$ into N_{POD} smaller domains of equal size $\frac{1}{N_{\text{POD}}}$, where N_{POD} denotes the desired number of perturbations. Next, a row vector of perturbations $[\delta\mathbf{u}]_i$ is constructed by generating a random number in each subdomain, and concatenating the resulting values in a random order. Repeating this process for each of the N_{u} input variables, N_{POD} random perturbations are obtained. Multiplying the attained vectors with the perturbation size, and adding these to the background input $\mathbf{u}_b$, near-random samples $\mathbf{u}^{(j)} := \mathbf{u}_b + \epsilon_{\text{POD}}\delta\mathbf{u}^{(j)}$ are then computed in the hypercube of size ϵ_{POD} around $\mathbf{u}_b$.

4.3.3 Pressure and saturation separation

In our implementation of the POD method, the considered states are composed of pressure and saturation information in the grid blocks. However, since these quantities display very different physical behaviour, constructing a single reduced basis $\tilde{\mathbf{\Phi}}$ to represent the state vectors might result in either the pressure or saturation variables to be underrepresented [39]. To avoid this, the orthogonal decomposition was performed separately for the pressure and state variables. To this end, snapshots $\mathbf{x}^{(i)} \in \mathbb{R}^{2N_g}$ of the full states were collected as before, but rather than combining these directly into a snapshot matrix $\mathbf{X} \in \mathbb{R}^{2N_g \times N_S}$, these were first divided into pressure and saturation snapshots $\mathbf{x}_p^{(i)}, \mathbf{x}_s^{(i)} \in \mathbb{R}^{N_g}$. Combining these into separate snapshot matrices $\mathbf{X}_p, \mathbf{X}_s \in \mathbb{R}^{N_g \times N_S}$, the POD method could then be performed to attain reduced pressure and saturation bases $\tilde{\mathbf{\Phi}}_p \in \mathbb{R}^{N_g \times \tilde{N}_p}$ and $\tilde{\mathbf{\Phi}}_s \in \mathbb{R}^{N_g \times \tilde{N}_s}$. Reducing the pressure and saturation states separately according to their respective bases, a complete reduced state could be attained recombining the results.

Since the pressure and saturation variables exhibit different variabilities, they will also require a different number of basis vectors to capture their dynamics. Accordingly, different energy criteria were implemented when establishing the size of each basis. To determine appropriate criteria, the eigenvalue spectra of the pressure and saturation states in the Kanaal model were considered. This was done using 100 input samples $\mathbf{u}^{(j)}$, for which the state was computed at each time step $n = 1, \dots, 150$, amounting to a total of 15000 snapshots. The resulting eigenvalue spectra were then computed as the squared singular values of the snapshot matrices $\mathbf{X}_p, \mathbf{X}_s$, displayed in figure 4.2. This figure also provides the corresponding cumulative relative energy for each of the variables, computed as the ratio of the partial energy (4.17) over the total energy (4.16).

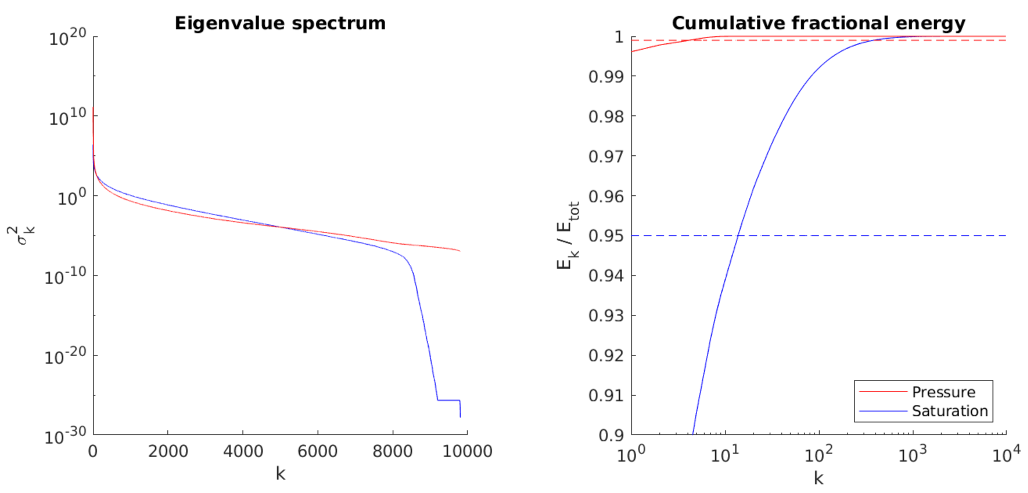


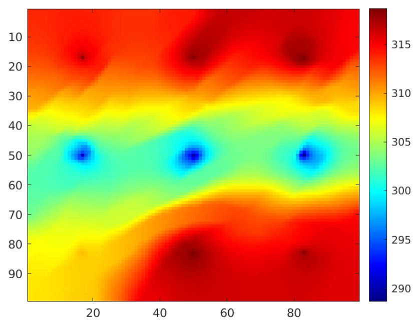
Figure 4.2: Eigenvalue spectrum and corresponding energy plot for pressure and saturation states based on 100 samples of the Kanaal model

The figure shows that the eigenvalues of the pressure state initially decrease more rapidly than those of the saturation state. In fact, the first pressure eigenvalue achieves a value of the order 10^{10}, but this order drops significantly with the consecutive values. Consequently, the total energy of the system is determined to an extreme extent by the initial eigenvalue, comprising already more than 99.5% of the total energy. This suggests a very low variability in the pressure values, allowing the states to be accurately represented with even a single basis vector. Nevertheless, to allow for slightly more freedom in the pressure values, retaining a few more basis vectors is desirable. Therefore, an energy condition of 99.9% was chosen for the pressure states, still requiring less than 10 basis vectors to represent 9801 variables.

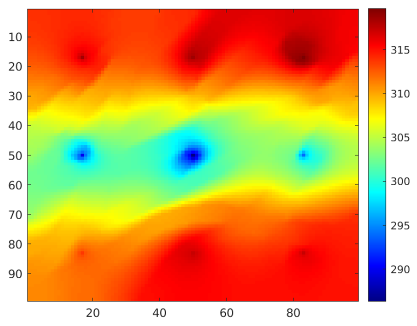
For the saturation state, the initial decrease in the eigenvalues occurs less rapidly than for the pressure state. This yields a more evenly divided contribution of the different eigenvalues to the total energy. Nevertheless, this contribution too does decrease, resulting in the cumulative energy to increase at a decreasing rate. Retaining an energy of 95%, the number of retained basis vectors would be of similar order to the pressure states, allowing for a significant reduction in the size of the state, whilst approximating the full saturation to a high accuracy.

4.3.4 Domain decomposition

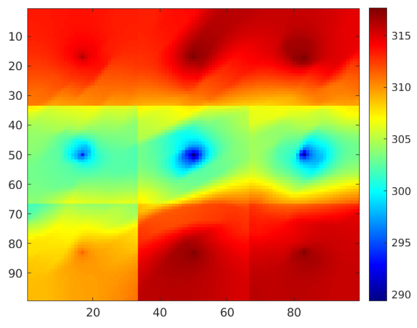
To investigate the effect of the domain decomposition on the accuracy, the POD method has been applied to the both the Kanaal and Egg model using 100 samples of the input, constructed using Latin hypercube sampling over the full domain of possible inputs. Herein, the vectors at each time step were used as snapshots, amounting to 15000 snapshots for the Kanaal model, and 12000 snapshots for the Egg model. In addition, a random sample $\mathbf{u}_{\text{ref}}$ of the input was used to attain reference states $\mathbf{x}_{\text{ref}}^n$ at each time step of each model. Then, the accuracy of the POD method using different domain decompositions could be tested, by comparing the reference states to their projected counterparts under the different implementations. A few of the results for the pressure states in the Kanaal model are illustrated in figure 4.3.



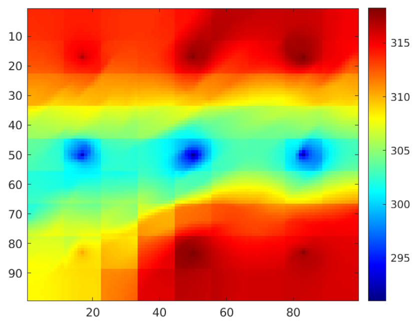
(a) True pressure field



(b) 1×1 domain projection



(c) 3×3 domain projection



(d) 9×9 domain projection

Figure 4.3: Sample of the pressure field and subdomain POD projections

The figure shows that, as would be expected, increasing the number of domains decreases the smoothness of the field. This is a result of the different domains being represented by different basis vectors, causing abrupt changes in the pressure and saturation values at the boundaries of the domains. Near the center of these domains, on the other hand, the decomposition provides a more accurate projection, as the same amount of snapshots is used to construct a basis for a smaller amount of variables. However, this comes at the cost of an increased number of basis vectors being retained, as described in table 4.1. In this table, results for a full projection of the Egg model variables are not presented, as the 12000 snapshots for this model are not sufficient to construct a full POD basis for the 18553 unknown variables.

As can be seen from the table, the total number of pressure and saturation basis vectors grows with the number of subdomains over which these are computed. This is a result of the same energy criterion being applied, resulting in similar numbers of vectors being retained in each subdomain as would originally be retained for the full space. This reduces the efficacy of the POD method when using a large number of subdomains, requiring a trade-off to be made between accuracy and computational cost when deciding upon the number of subdomains. This trade-off becomes even more relevant when implementing a surrogate model later on, for which an opposite effect occurs, decreasing both the accuracy and required computational effort with the number of domains. For this reason, only a limited number of subdomains has been implemented for this thesis, consistently using a 4×4 decomposition. In this manner, a significant reduction in the cost was achieved, whilst maintaining a high accuracy.

	Kanaal pressure	Kanaal saturation	Egg pressure	Egg saturation
Full state	9801	9801	18553	18553
1×1 projection	5	14	–	–
3×3 projection	12	48	58	40
5×5 projection	25	101	109	83
7×7 projection	49	158	176	136
9×9 projection	81	220	291	219

Table 4.1: Reduced number of POD state parameters upon implementing different domain decompositions

4.3.5 Number of snapshots

In constructing a reduced-order model, the POD method rests on the collected snapshots to provide an accurate representation of the general states acquired in the model. Intuitively, more of such snapshots will thus allow for a higher accuracy of the method, as these should provide a more accurate representation of the full range of possible states. However, taking more snapshots also comes at an increased computational cost, as this will require more full-order simulations to be run. To determine how many snapshots to take, an approach from [43] was considered, quantifying the accuracy of the method in terms of the singular value spectrum corresponding to the POD basis.

Let $\mathbf{X} \in \mathbb{R}^{N_x \times (N \cdot N_{\text{POD}})}$ be a snapshot matrix, constructed running the full simulation of N steps using N_{POD} different control vectors. From this snapshot matrix, an orthogonal basis $\mathbf{\Phi}$ with corresponding singular values $\boldsymbol{\sigma}$ is determined through a singular value decomposition. Next, an additional $N_{\text{POD}}^{\text{add}}$ inputs are randomly generated, assumed to differ significantly from the earlier inputs. Running the full-order model for each of these inputs, $N \cdot N_{\text{POD}}^{\text{add}}$ new snapshots are computed and added to the snapshot matrix, for which a singular value decomposition is then also performed. Now, if the original basis $\mathbf{\Phi}$ does not accurately represent the considered domain, and the new inputs differ sufficiently from the original ones, the addition of new snapshots should result in a significant adjustment of this basis and the corresponding spectrum $\boldsymbol{\sigma}$. Conversely, if the original basis provides a good representation of the considered space state, the change in spectrum upon addition of new snapshots would be minimal. Therefore, the change in the spectrum $\boldsymbol{\sigma}$ under addition of such snapshots can be taken as a measure of the quality of basis $\mathbf{\Phi}$ for the state reduction, where a smaller change implies a higher accuracy. In our implementation, this change was computed as:

$$\Delta_\sigma := \frac{\|\boldsymbol{\sigma}^{(i+1)} - \boldsymbol{\sigma}^{(i)}\|_2}{\|\boldsymbol{\sigma}^{(i+1)}\|_2} \tag{4.20}$$

In this notation, superscripts $(i+1)$ and (i) denote consecutive spectra computed before and after adding new snapshots to the matrix $\mathbf{X}$. By dividing by the norm of the new spectrum, the difference Δ_σ between the spectra would be computed relative to the current spectrum. The necessary number of snapshots could then be established by requiring this measure of consistency of the spectrum to be below a certain threshold ϵ_σ, adding samples until this criterion is met.

To investigate the suitability of a spectrum consistency criterion in determining the number of required samples, the change in spectrum (4.20) has been computed upon adding four new samples to the POD set, starting at various numbers of samples. These samples were always generated randomly in the domain of allowed inputs, generally assuring these new inputs to differ sufficiently from the precomputed samples. This was done for both the Kanaal and the Egg model, in a 4×4 subdomain framework. The results are displayed in figure 4.4, plotting the change in spectrum against the original size of the POD set.

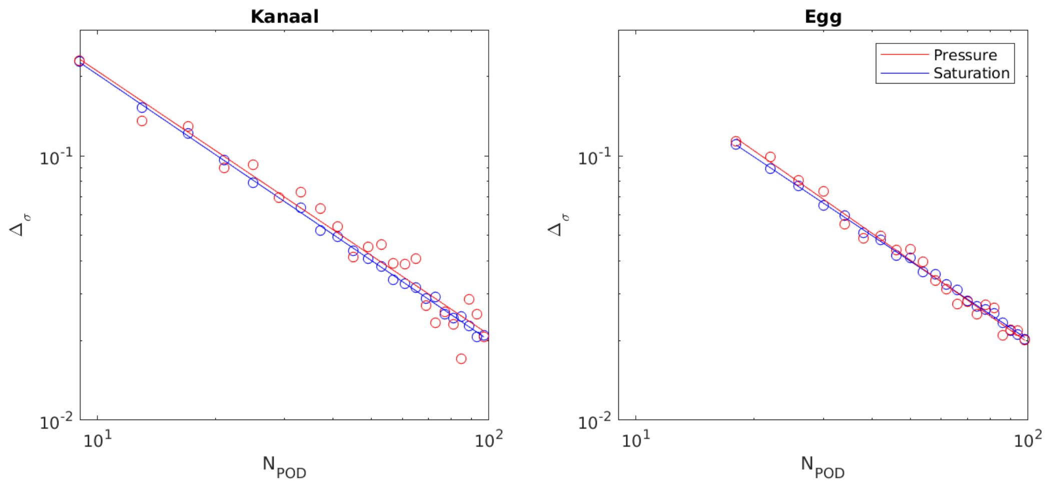


Figure 4.4: Relative change in spectrum upon increasing the number of POD samples for Kanaal and Egg model

The figure shows that, as the size of the snapshot matrix increases, the relative change in spectrum decreases linearly on a log-log scale, both for the pressure and saturation variables, and in both models. Here, each of the linear fits has a slope of approximately -1, suggesting the change in spectrum is inversely related to the number of samples used to generate it. Moreover, for both models, the number of samples required to achieve a certain consistency in the spectrum seems to be of the same order.

To establish how the spectrum consistency correlates to the accuracy of the reduction, this accuracy is measured using a random reference input $\mathbf{u}_{\text{ref}}$. Computing the states $\mathbf{x}^n_{\text{ref}}$ at each time step of the two models for this reference input, the POD projected state $\mathbf{\Pi}\mathbf{x}^i_{\text{ref}}$ could be compared to the actual state at each time. A measure of the accuracy of the basis could then be defined as the mean difference:

$$\varepsilon_{\text{ref}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\|\mathbf{x}^i_{\text{ref}} - \mathbf{\Pi}\mathbf{x}^i_{\text{ref}}\|_2}{\|\mathbf{x}^i_{\text{ref}}\|_2} \tag{4.21}$$

This mean was computed for each of the bases considered in figure 4.4, taking the pressure and saturation errors separately for both models. The results were plotted against the corresponding change in spectrum, as displayed in figure 4.5. In this figure, the spectrum consistency values were normalized according to an L-infinity norm, to allow the different situations to be plotted on the same scale.

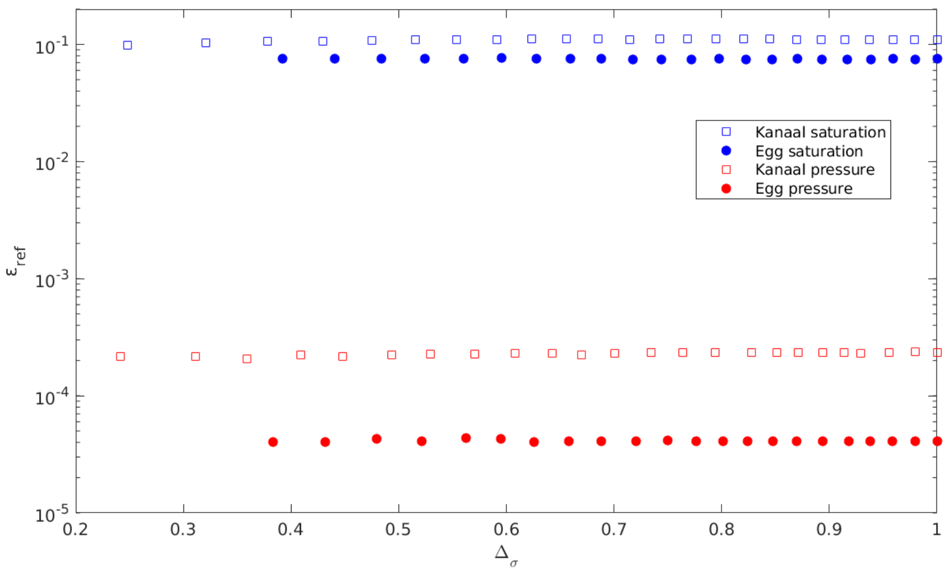


Figure 4.5: Accuracy of projection against scaled spectrum consistency value

The figure shows that, as the consistency of the spectrum decreases, the accuracy of the method remains of the same order. For both the Egg and Kanaal model, this accuracy is greater for the pressure states, likely due to the higher energy criterion. Nevertheless, both errors remain practically constant as the difference between consecutive spectra changes, suggesting no information on the accuracy of the method can be extracted from this difference as implemented here. Since also the dependence of this difference on the number of samples is the same for each model, there is no purpose in explicitly imposing a spectrum consistency condition, rather than directly fixing the number of POD samples. For this reason, the number of samples used to construct the reduced-order model has always been directly specified in the numerical experiments conducted for this thesis, rather than implementing a desired change in spectrum.

4.3.6 The subdomain POD algorithm

Algorithm 2 below describes the POD method for constructing an optimal reduced basis, performed in a subdomain framework, as implemented for the purposes of this thesis. Although in this algorithm the bases are constructed using a completely new set of samples, any precomputed snapshots can in theory be reused, though this will increase the computational cost.

Algorithm 2: Subdomain POD basis contruction

```
   Input: Background input $\mathbf{u}_b$, Number of perturbations $N_{\text{POD}}$, Size of perturbations
          $\epsilon_{\text{POD}}$, Domain decomposition $\Omega = \bigcup_{d=1}^{N_D} \Omega^d$
   Result: Reduced bases $\tilde{\mathbf{\Phi}}_{p,s}^d$ in each subdomain $\Omega^d$
 1 Construct $N_{\text{POD}}$ random perturbations $\delta\mathbf{u}^{(j)}$ in $[-\frac{1}{2}, \frac{1}{2}]$ using Latin hypercube sampling;
 2 Construct random input samples $\mathbf{u}^{(j)} := \mathbf{u}_b + \epsilon_{\text{POD}}\delta\mathbf{u}^{(j)} \in \Gamma_{\text{POD}}$;
 3 for $j \in \Gamma_{\text{POD}}$ do
 4     Run full-order model for $\mathbf{u}^{(j)}$;
 5     Collect states $\mathbf{x}^{(j),n}$ at all times $n = 1, \ldots, N$ in snapshot matrix $\mathbf{X}^{(j)}$;
 6 end
 7 Combine snapshot matrices into single matrix $\mathbf{X} = [\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N_{\text{POD}})}]$;
 8 for $d = 1, \ldots, N_D$ do
 9     Extract variables from $\mathbf{X}$ corresponding to cells in $\Omega^d$ in $\mathbf{X}^d$;
10     Divide $\mathbf{X}^d$ into pressure and saturation snapshots $\mathbf{X}_p^d, \mathbf{X}_s^d$;
11     Compute column-wise means $\hat{\mathbf{x}}_p^d, \hat{\mathbf{x}}_s^d$ of each matrix according to (4.19);
12     Redefine snapshots matrices as shifted by their means: $\mathbf{X}_{p,s}^d \to \mathbf{X}_{p,s}^d - \hat{\mathbf{x}}_{p,s}^d$;
13     Perform singular value decomposition on each matrix: $\mathbf{X}_{p,s}^d = \mathbf{U}_{p,s}^d \mathbf{\Sigma}_{p,s}^d [\mathbf{V}_{p,s}^d]^T$;
14     Determine minimal $\tilde{N}_p$ satisfying $\sum_{i=1}^{\tilde{N}_p^d} [\sigma_p^d]_i^2 \geq \frac{99.9}{100} \sum_{i=1}^{N_g^d} [\sigma_p^d]_i^2$;
15     Determine minimal $\tilde{N}_s$ satisfying $\sum_{i=1}^{\tilde{N}_s^d} [\sigma_s^d]_i^2 \geq \frac{95}{100} \sum_{i=1}^{N_g^d} [\sigma_s^d]_i^2$;
16     Retain only first $\tilde{N}_{p,s}$ columns of $\mathbf{\Sigma}_{p,s}^d$ in $\tilde{\mathbf{\Sigma}}_{p,s}^d$;
17     Construct reduced pressure and saturation bases: $\tilde{\mathbf{\Phi}}_{p,s}^d = \mathbf{U}_{p,s}^d \tilde{\mathbf{\Sigma}}_{p,s}^d$;
18 end
```

Running this algorithm, reduced bases for the state vectors can be constructed in each domain, for both the pressure and saturation variables separately. Based on these bases, an arbitrary state vector $\mathbf{x}^{d,n} \in \mathbb{R}^{N_x^d}$ in a domain d at a time step n, can be approximated through a corresponding reduced component vector $\tilde{\boldsymbol{\psi}}^{d,n} \in \mathbb{R}^{\tilde{N}_x^d}$ as:

$$\mathbf{x}^{d,n} = \begin{pmatrix} \mathbf{x}_p^{d,n} \\ \mathbf{x}_s^{d,n} \end{pmatrix} \approx \begin{pmatrix} \tilde{\mathbf{\Phi}}_p^d & \mathbf{O} \\ \mathbf{O} & \tilde{\mathbf{\Phi}}_s^d \end{pmatrix} \begin{pmatrix} \tilde{\boldsymbol{\psi}}_p^{d,n} \\ \tilde{\boldsymbol{\psi}}_s^{d,n} \end{pmatrix} = \tilde{\mathbf{\Phi}}^d \tilde{\boldsymbol{\psi}}^{d,n} \tag{4.22}$$

Here, in order to compute the components $\tilde{\boldsymbol{\psi}}^{(j),d,n}$ of a known state $\mathbf{x}^{(j),d,n}$, the pseudo-inverse of $\tilde{\mathbf{\Phi}}^d$ is used:

$$\tilde{\boldsymbol{\psi}}^{(j),d,n} = ([\tilde{\mathbf{\Phi}}^d]^T \tilde{\mathbf{\Phi}}^d)^{-1} [\tilde{\mathbf{\Phi}}^d]^T \mathbf{x}^{(j),d,n} \tag{4.23}$$

In this manner, any state can be reduced to a component state, and vice-versa, allowing for less demanding application of an adjoint methodolgy.

Chapter 5

Non-Intrusive Piecewise Linearization

In this chapter, the radial basis function technique is introduced, within the framework of an application towards trajectory piecewise linearization of the model. The implementation of the domain decomposition for this method is also discussed, as well as decisions on other stages of the method. Finally, the accuracy of this method is tested for several situations.

5.1 Surrogate modelling

We once more consider the problem of optimizing an objective function $J(\mathbf{u})$, based on a model of the form:

$$\begin{aligned} \mathbf{x}^0 &= \mathbf{x}^{\text{init}} \\ \mathbf{x}^{n+1} &= \mathbf{f}^{n+1}(\mathbf{x}^n, \mathbf{x}^{n+1}, \mathbf{u}^{n+1}) \\ \mathbf{y}^{n+1} &= \mathbf{h}^{n+1}(\mathbf{x}^{n+1}, \mathbf{u}^{n+1}) \qquad\qquad n = 0, \dots, N-1 \end{aligned} \tag{5.1}$$

In order to apply an adjoint method, principal orthogonal decomposition can be used to reduce the size of the necessary derivative matrices $\frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}^n}, \frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}^{n+1}} \in \mathbb{R}^{N_\mathrm{x} \times N_\mathrm{x}}$ and $\frac{d\mathbf{h}^{n+1}}{d\mathbf{x}^{n+1}} \in \mathbb{R}^{N_\mathrm{y} \times N_\mathrm{x}}$. However, an additional problem presents itself in computing these derivatives, as this requires knowledge on the different functions. This in turn demands access to the underlying model code, which in many cases is strenuous or even unfeasible. To be able to implement the adjoint method, then, surrogate modelling techniques are necessary.

Similarly to model order reduction, surrogate modelling involves establishing an approximation of a model, which is easier to work with, whilst still providing accurate results. The focus here, however, lies in reducing the complexity rather than the size of the model, aiming to establish an approximation for which the derivative functions can be readily computed. Such models are often constructed by interpolating between known values of the state vector, as for example done in four-dimensional variational schemes [4]. Alternatively, trajectory piecewise linearization (TPWL) can be applied, extrapolating information acquired by running the model for some sample inputs $\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(N_{\text{TPWL}})}$. When performing this method, the model is linearized around the results corresponding to the sample inputs, approximating system (5.1) as:

$$\begin{aligned}\left[\mathbf{I} - \frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}^n_{\text{tr}}}\right](\mathbf{x}^{n+1} - \mathbf{x}^{n+1}_{\text{tr}}) &\approx \frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}^n_{\text{tr}}}(\mathbf{x}^n - \mathbf{x}^n_{\text{tr}}) + \frac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{u}^{n+1}_{\text{tr}}}(\mathbf{u}^{n+1} - \mathbf{u}^{n+1}_{\text{tr}}) \\ (\mathbf{y}^{n+1} - \mathbf{y}^{n+1}_{\text{tr}}) &\approx \frac{\partial \mathbf{h}^{n+1}}{\partial \mathbf{x}^{n+1}_{\text{tr}}}(\mathbf{x}^{n+1} - \mathbf{x}^{n+1}_{\text{tr}}) + \frac{\partial \mathbf{h}^{n+1}}{\partial \mathbf{u}^{n+1}_{\text{tr}}}(\mathbf{u}^{n+1} - \mathbf{u}^{n+1}_{\text{tr}}) \quad n = 0, \dots, N-1\end{aligned} \tag{5.2}$$

In this approximation, states $\mathbf{x}^n_{\text{tr}}$ and outputs $\mathbf{y}^n_{\text{tr}}$ at each simulation step n, are those attained when running the full model based on input sample $\mathbf{u}_{\text{tr}}$. This input is selected from the set of input samples, chosen to be the one closest to the current input $\mathbf{u}$, minimizing the Euclidean distance $\|\mathbf{u} - \mathbf{u}^{(i)}\|_2$. In this manner, the piecewise linear approximation will linearly approximate the model around the samples, exactly matching them at the sample inputs. However, this does require knowing the derivatives of the model functions at each of these inputs, thereby relying on access to the model code. To circumvent this intrusive property of the TPWL method, it can be combined with radial basis function (RBF) interpolation, allowing function derivatives to be computed without requiring knowledge of the functions themselves.

5.2 Radial basis function interpolation

Radial basis function interpolation is, as the name suggests, a method for interpolating between provided data using radial basis functions. Accordingly, it can be used to directly construct a surrogate model based on results attained from a set of sample inputs, as has been done in works such as [24] and [42]. Nevertheless, the resulting models will remain non-linear, and will require a large number of samples to provide accurate results. Instead, we discuss RBF interpolation only for the purpose of constructing gradient matrices of the model, at which point the TPWL method will be applied to construct a surrogate.

In order to perform RBF interpolation, a certain set of samples must be collected. This is done running the full-order model for N_{RBF} sample inputs $\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(N_{\text{RBF}})}$, and collecting the resulting state vectors $\mathbf{x}^{(i),n}$ and outputs $\mathbf{y}^{(i),n}$ at all steps n, as well as the inputs $\mathbf{u}^{(i)}$ themselves, into a set Γ_{RBF}. For ease of notation, we will broadly refer to sample $i \in \Gamma_{\text{RBF}}$ as the full set of inputs and results attained when running the model using input vector $\mathbf{u}^{(i)}$. We divide this information into two sets of vectors, respectively describing the arguments for functions $\mathbf{f}$ and $\mathbf{h}$ in system (5.1) as:

$$\begin{aligned}[\boldsymbol{\chi}^{(i),n+1}]^T &:= \left([\mathbf{x}^{(i),n}]^T \quad [\mathbf{x}^{(i),n+1}]^T \quad [\mathbf{u}^{(i),n+1}]^T\right) \\ [\boldsymbol{\gamma}^{(i),n+1}]^T &:= \left([\mathbf{x}^{(i),n+1}]^T \quad [\mathbf{u}^{(i),n+1}]^T\right) \qquad n = 0, \dots, N-1\end{aligned} \tag{5.3}$$

Accordingly, we define full argument vectors $\boldsymbol{\chi}^{(i)}$ and $\boldsymbol{\gamma}^{(i)}$ as concatenations of the corresponding arguments at all time steps. Collecting this information on the different samples in set Γ_{RBF}, information regarding a new input $\mathbf{u}$ can be extrapolated based on the distance of this input to each of the samples. To this end, we first introduce functions describing the Euclidean distances between the collected samples:

$$\begin{aligned}r^{n+1}_{ij} &:= \|\boldsymbol{\chi}^{(i),n+1} - \boldsymbol{\chi}^{(j),n+1}\|_2 \\ \ell^{n+1}_{ij} &:= \|\boldsymbol{\gamma}^{(i),n+1} - \boldsymbol{\gamma}^{(j),n+1}\|_2 \qquad n = 0, \dots, N-1\end{aligned} \tag{5.4}$$

Similarly, letting $\boldsymbol{\chi}, \boldsymbol{\gamma}$ denote the argument vectors corresponding to the input $\mathbf{u}$, we introduce radial functions:

$$\begin{aligned} r_i^{n+1}(\boldsymbol{\chi}^{n+1}) &:= \|\boldsymbol{\chi}^{n+1} - \boldsymbol{\chi}^{(i),n+1}\|_2 \\ \ell_i^{n+1}(\boldsymbol{\gamma}^{n+1}) &:= \|\boldsymbol{\gamma}^{n+1} - \boldsymbol{\gamma}^{(i),n+1}\|_2 \qquad n = 0, \ldots, N-1 \end{aligned} \tag{5.5}$$

These functions express the distances between argument vectors corresponding to a new input $\mathbf{u}$, and those of the precomputed RBF samples. Based on these distances, interpolation can then be performed to compute the state and output at any time step. In particular, state weighting vectors $\boldsymbol{\omega}_i^{n+1} \in \mathbb{R}^{\mathrm{x}}$ and output weighting vectors $\boldsymbol{\varepsilon}_i^{n+1} \in \mathbb{R}^{N_{\mathrm{y}}}$ for $n = 0, \ldots, N-1$ are assigned to each sample $i \in \Gamma_{\mathrm{RBF}}$. Afterwards, assuming the state $\mathbf{x}^n$ and corresponding argument $\boldsymbol{\chi}^{n+1}$ to be known at time step n, an approximate state $\boldsymbol{x}^{n+1}$ at the next time step is computed as a linear combination of the weighting vectors, scaled according to their distance from the current input:

$$\boldsymbol{x}^{n+1} = \sum_{i \in \Gamma_{\mathrm{RBF}}} \boldsymbol{\omega}_i^{n+1} r_i^{n+1}(\boldsymbol{\chi}^{n+1}) \qquad n = 0, \ldots, N-1 \tag{5.6}$$

Computing the state at each step in this manner, each weighting vector is scaled in direct proportion to the distance between the new input and the RBF samples, corresponding to a linear spline method. From the resulting state, the next argument vector $\boldsymbol{\chi}^{n+2}$ can then be computed, allowing states at each simulation step to be approximated. Herein, the initial argument vector $\boldsymbol{\chi}^1$ can be constructed simply based on the initial state $\mathbf{x}^{\mathrm{init}}$, which is independent of the input $\mathbf{u}$. Generating approximate states at each time step in this manner, corresponding outputs can be estimated similarly as:

$$\boldsymbol{y}^{n+1} = \sum_{i \in \Gamma_{\mathrm{RBF}}} \boldsymbol{\varepsilon}_i^{n+1} \ell_i^{n+1}(\boldsymbol{\gamma}^{n+1}) \qquad n = 0, \ldots, N-1 \tag{5.7}$$

5.2.1 Multiquadratic basis function

Computing the states and outputs according to (5.6) and (5.7), the contribution of a sample i to the solution at an unknown input is taken to decrease linearly with the distance between the two. As a result, the state at each step will also be a piecewise linear function of the input, with an undefined gradient at each of the samples. To avoid this, the distances are usually transformed according to some non-linear basis function $\theta(r)$, aiming to attain a smoother interpolation. For this thesis, this has been done using a multiquadratic function, defined as:

$$\theta(r) := \sqrt{(\epsilon r)^2 + 1} \tag{5.8}$$

In this function, $\epsilon > 0$ denotes a shape parameter, determining how strongly the distances affect the correlation. Specifically, a small shape parameter increases the contribution of the added constant 1, resulting in an almost constant basis function $\theta(r)$. On the other hand, when using a large shape parameter, the contribution of the added value 1 quickly diminishes away from $r = 0$, resulting in an approximately linear function. Nevertheless, the added constant assures that the basis function remains bigger than 0 at $r = 0$, which is crucial to allow derivative functions to be computed at the sample points.

5.2.2 Computing weighting coefficients

Implementing a basis function $\theta(r)$ to transform the distances, we attain a surrogate model:

$$\begin{aligned} \boldsymbol{x}^{n+1} &= \boldsymbol{f}^{n+1}(\boldsymbol{\chi}^{n+1}) := \sum_{i\in\Gamma_{\mathrm{RBF}}} \boldsymbol{\omega}_i^{n+1}\theta(r_i^{n+1}(\boldsymbol{\chi}^{n+1})) \\ \boldsymbol{y}^{n+1} &= \boldsymbol{h}^{n+1}(\boldsymbol{\gamma}^{n+1}) := \sum_{i\in\Gamma_{\mathrm{RBF}}} \boldsymbol{\varepsilon}_i^{n+1}\theta(\ell_i^{n+1}(\boldsymbol{\gamma}^{n+1})) \qquad n = 0,\ldots,N-1 \end{aligned} \tag{5.9}$$

In this system, the weighting coefficient vectors $\boldsymbol{\omega}^{(i),n}, \boldsymbol{\varepsilon}^{(i),m}$ are chosen such that the surrogate model matches the original model at the computed RBF samples. At these samples, the states $\mathbf{x}^{(i),n}$ and outputs $\mathbf{y}^{(i),n}$ are known at all times $n = 1,\ldots,N$. Therefore, the weighting coefficients at an arbitrary sample $j \in \Gamma_{\mathrm{RBF}}$ can be computed solving the system:

$$\begin{aligned} \mathbf{x}^{(j),n+1} &= \sum_{i\in\Gamma_{\mathrm{RBF}}} \boldsymbol{\omega}_i^{n+1}\theta(r_{ij}^{n+1}) = \boldsymbol{\mathcal{W}}^{n+1}\boldsymbol{\theta}_j^{r,n+1} \\ \mathbf{y}^{(j),n+1} &= \sum_{i\in\Gamma_{\mathrm{RBF}}} \boldsymbol{\varepsilon}_i^{n+1}\theta(\ell_{ij}^{n+1}) = \boldsymbol{\mathcal{E}}^{n+1}\boldsymbol{\theta}_j^{\ell,n+1} \qquad n = 0,\ldots,N-1 \end{aligned} \tag{5.10}$$

Here, $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{N_{\mathrm{x}}\times N_{\mathrm{RBF}}}$ and $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{N_{\mathrm{y}}\times N_{\mathrm{RBF}}}$ are matrices composed column-wise of the state and output weighting vectors. Similarly, the vectors $\boldsymbol{\theta}_j^{r,n+1}, \boldsymbol{\theta}_j^{\ell,n+1} \in \mathbb{R}^{N_{\mathrm{RBF}}}$ contain the distances between sample $j \in \Gamma_{\mathrm{RBF}}$ and all other samples. Combining this system for each of the samples in Γ_{RBF}, we find a matrix equation:

$$\begin{aligned} \boldsymbol{X}^{n+1} &= \boldsymbol{\mathcal{W}}^{n+1}\boldsymbol{\Theta}^{r,n+1} \\ \boldsymbol{Y}^{n+1} &= \boldsymbol{\mathcal{E}}^{n+1}\boldsymbol{\Theta}^{\ell,n+1} \end{aligned} \tag{5.11}$$

In this system, $\boldsymbol{X}^{n+1} \in \mathbb{R}^{N_{\mathrm{x}}\times N_{\mathrm{RBF}}}$ and $\boldsymbol{Y}^{n+1} \in \mathbb{R}^{N_{\mathrm{y}}\times N_{\mathrm{RBF}}}$ are column-wise compositions of respectively the states and outputs at the different samples. The distance vectors $\boldsymbol{\theta}_j^{r,n+1}, \boldsymbol{\theta}_j^{\ell,n+1}$ have also been combined into matrices $\boldsymbol{\Theta}^{r,n+1}, \boldsymbol{\Theta}^{\ell,n+1} \in \mathbb{R}^{N_{\mathrm{RBF}}\times N_{\mathrm{RBF}}}$, yielding a linear system of $N_{\mathrm{x}} \times N_{\mathrm{RBF}}$ equations on an equal number of unknowns. This system can be solved efficiently using an arbitrary linear solver, allowing the weighting vectors at each time step to be computed.

5.2.3 Choice of shape parameter

In computing the weighting vectors, and for consecutively computing results for a new input, a shape parameter has to be chosen for the basis functions. As noted, this shape parameter will determine how strongly the distance from the samples affects the solution for an arbitrary input, and therefore a suitable value must be determined. A common method to do so is using "leave-one-out" cross validation (LOOCV) [13]. For this method, an estimate is made of the error in the interpolated solution, by comparing this interpolant to the actual solution at known samples. Since the interpolated solution is already implemented to exactly match the actual one at the RBF samples, this would demand additional samples to be taken. To avoid this extra computational cost, the LOOCV method instead computes the error at each RBF sample, when interpolating based only on the other samples. For sample $k \in \Gamma_{\mathrm{RBF}}$, then, this error is defined as:

$$\begin{aligned} L_{\mathrm{x}}^{(k),n+1}(\epsilon) &= \|\mathbf{x}^{(k),n+1} - \tilde{\boldsymbol{x}}^{(k),n+1}\|_2 \\ L_{\mathrm{y}}^{(k),n+1}(\epsilon) &= \|\mathbf{y}^{(k),n+1} - \tilde{\boldsymbol{y}}^{(k),n+1}\|_2 \qquad n = 0,\ldots,N-1 \end{aligned} \tag{5.12}$$

Here, vectors $\tilde{\boldsymbol{x}}^{(k),n+1} \in \mathbb{R}^{N_\mathrm{x}}$ and $\tilde{\boldsymbol{y}}^{(k),n+1} \in \mathbb{R}^{N_\mathrm{y}}$ are defined as the interpolated states and outputs at sample $k \in \Gamma_\mathrm{RBF}$, based on the remaining samples $i \in \Gamma_{RBF}\backslash\{k\}$:

$$\begin{aligned}
\tilde{\boldsymbol{x}}^{(k),n+1} &:= \sum_{i\in\Gamma_\mathrm{RBF}\backslash\{k\}} \tilde{\boldsymbol{\omega}}_i^{(k),n+1}\theta(r_i^{n+1}(\boldsymbol{\chi}^{(k),n+1})) \\
\tilde{\boldsymbol{y}}^{(k),n+1} &:= \sum_{i\in\Gamma_\mathrm{RBF}\backslash\{k\}} \tilde{\boldsymbol{\varepsilon}}_i^{(k),n+1}\theta(\ell_i^{n+1}(\boldsymbol{\gamma}^{(k),n+1})) \qquad n = 0,\dots,N-1
\end{aligned} \tag{5.13}$$

In this system, the weighting vectors at each sample are determined separately for the situation excluding each sample $k \in \Gamma_\mathrm{RBF}$, assuring the approximated vectors $\tilde{\boldsymbol{x}}^{(k),n+1}, \tilde{\boldsymbol{y}}^{(k),n+1}$ still match the exact results at the samples. Computing an error for each RBF sample this way, a total error corresponding to an arbitrary value $\epsilon > 0$ is given by the sum:

$$\begin{aligned}
L_\mathrm{x}^{n+1}(\epsilon) &= \sum_{k\in\Gamma_\mathrm{RBF}} L_\mathrm{x}^{(k),n+1} = \sum_{k\in\Gamma_\mathrm{RBF}} \|\mathbf{x}^{(k),n+1} - \tilde{\boldsymbol{x}}^{(k),n+1}\|_2 \\
L_\mathrm{y}^{n+1}(\epsilon) &= \sum_{k\in\Gamma_\mathrm{RBF}} L_\mathrm{y}^{(k),n+1} = \sum_{k\in\Gamma_\mathrm{RBF}} \|\mathbf{y}^{(k),n+1} - \tilde{\boldsymbol{y}}^{(k),n+1}\|_2 \qquad n = 0,\dots,N-1
\end{aligned} \tag{5.14}$$

Providing a single output on the basis of a single input, these functions can be efficiently minimized with respect to the parameter ϵ within a domain. In this manner, optimal values ϵ can be computed at each time step for both the state and output basis functions, minimizing the errors (5.14).

5.2.4 Derivative functions

With the optimal shape parameter at each time step, weighting vectors for each sample can be computed by solving system (5.11), allowing the model at any input $\mathbf{u}$ to be approximated through system (5.9). However, remaining fairly complex, this model is constructed solely for the purpose of attaining gradient information at the samples, at which point the TPWL methodology can be applied. To determine these gradients, the analytical derivatives at each sample $k \in \Gamma_\mathrm{RBF}$ are computed:

$$\begin{aligned}
\left.\frac{\partial \mathbf{f}^{n+1}}{\partial \boldsymbol{\chi}^{n+1}}\right|_{\boldsymbol{\chi}=\boldsymbol{\chi}^{(k)}} &\approx \left.\frac{\partial \boldsymbol{f}^{n+1}}{\partial \boldsymbol{\chi}^{n+1}}\right|_{\boldsymbol{\chi}=\boldsymbol{\chi}^{(k)}} = \sum_{i\in\Gamma_\mathrm{RBF}} \boldsymbol{\omega}_i^{n+1} \left.\frac{\partial\theta(r_i^{n+1}(\boldsymbol{\chi}^{n+1}))}{\partial \boldsymbol{\chi}^{n+1}}\right|_{\boldsymbol{\chi}=\boldsymbol{\chi}^{(k)}} \\
\left.\frac{\partial \mathbf{h}^{n+1}}{\partial \boldsymbol{\gamma}^{n+1}}\right|_{\boldsymbol{\gamma}=\boldsymbol{\gamma}^{(k)}} &\approx \left.\frac{\partial \boldsymbol{h}^{n+1}}{\partial \boldsymbol{\gamma}^{n+1}}\right|_{\boldsymbol{\gamma}=\boldsymbol{\gamma}^{(k)}} = \sum_{i\in\Gamma_\mathrm{RBF}} \boldsymbol{\varepsilon}_i^{n+1} \left.\frac{\partial\theta(\ell_i^{n+1}(\boldsymbol{\gamma}^{n+1}))}{\partial \boldsymbol{\gamma}^{n+1}}\right|_{\boldsymbol{\gamma}=\boldsymbol{\gamma}^{(k)}} \qquad n = 0,\dots,N-1
\end{aligned} \tag{5.15}$$

At each RBF sample k, these derivatives provide approximate gradients of the functions $\mathbf{f}$ and $\mathbf{h}$ in system (5.1), with respect to their arguments $\boldsymbol{\chi}$ and $\boldsymbol{\gamma}$, at each simulation step. These gradients only require taking the derivative of the basis vectors with respect to each argument, which can be done analytically. Indeed, considering an arbitrary sample $i \in \Gamma_\mathrm{RBF}$, the basis function derivative can be split using the chain rule:

$$\begin{aligned}
\frac{\partial\theta(r_i^{n+1}(\boldsymbol{\chi}^{n+1}))}{\partial \boldsymbol{\chi}^{n+1}} &= \frac{\partial\theta(r_i^{n+1})}{\partial r_i^{n+1}}\frac{\partial r_i^{n+1}(\boldsymbol{\chi}^{n+1})}{\partial \boldsymbol{\chi}^{n+1}} \\
\frac{\partial\theta(\ell_i^{n+1}(\boldsymbol{\gamma}^{n+1}))}{\partial \boldsymbol{\gamma}^{n+1}} &= \frac{\partial\theta(\ell_i^{n+1})}{\partial \ell_i^{n+1}}\frac{\partial \ell_i^{n+1}(\boldsymbol{\gamma}^{n+1})}{\partial \boldsymbol{\gamma}^{n+1}} \qquad n = 0,\dots,N-1
\end{aligned} \tag{5.16}$$

Considering now an arbitrary argument χ_j^{n+1} at an arbitrary time step in the first equation, we find the derivative of the distance function to be:

$$\frac{\partial r_i^{n+1}}{\partial \chi_j^{n+1}} = \frac{\partial}{\partial \chi_j} \|\boldsymbol{\chi}^{n+1} - \boldsymbol{\chi}^{(i),n+1}\|_2 = \frac{\chi_j^{n+1} - \chi_j^{(i),n+1}}{\|\boldsymbol{\chi}^{n+1} - \boldsymbol{\chi}^{(i),n+1}\|_2} = \frac{\chi_j^{n+1} - \chi_j^{(i),n+1}}{r_i^{n+1}(\boldsymbol{\chi}^{n+1})} \tag{5.17}$$

Since the gradients are computed at the training points, values $r_i = 0$ will have to be considered, leading to singularities in the above derivative. However, using the multi-quadratic function basis function, such singularities are avoided, as the derivative of this function depends linearly on the radial distance:

$$\frac{\partial \theta(r)}{\partial r} = \frac{\partial}{\partial r}\sqrt{(\epsilon r)^2 + 1} = \frac{\epsilon^2 r}{\sqrt{(\epsilon r)^2 + \epsilon^2}} = \frac{\epsilon^2}{\theta(r)} r \tag{5.18}$$

Substituting these results back into (5.16), and performing the same steps for the output derivative function, we find:

$$\begin{aligned} \frac{\partial \theta(r_i^{n+1}(\boldsymbol{\chi}^{n+1}))}{\partial \boldsymbol{\chi}^{n+1}} &= \epsilon^2 \frac{[\boldsymbol{\chi}^{n+1}]^T - [\boldsymbol{\chi}^{(i),n+1}]^T}{\theta(r_i^{n+1}(\boldsymbol{\chi}^{n+1}))} \\ \frac{\partial \theta(\ell_i^{n+1}(\boldsymbol{\gamma}^{n+1}))}{\partial \boldsymbol{\gamma}^{n+1}} &= \epsilon^2 \frac{[\boldsymbol{\gamma}^{n+1}]^T - [\boldsymbol{\gamma}^{(i),n+1}]^T}{\theta(\ell_i^{n+1}(\boldsymbol{\gamma}^{n+1}))} \qquad n = 0, \dots, N-1 \end{aligned} \tag{5.19}$$

With these functions, and the earlier computed weighting vectors, the derivatives (5.15) can be determined at each sample in Γ_{RBF}. Subsequently, a trajectory piecewise linear function can be constructed, providing state and output vectors for an arbitrary input $\mathbf{u} \in \mathbb{R}^{N_\text{u}}$ by linearly extrapolating results from the closest sample. That is, letting $\mathbf{u}_\text{tr}$ denote the sample input $\mathbf{u}^{(i)} \in \Gamma_{\text{RBF}}$ minimizing the distance $\|\mathbf{u} - \mathbf{u}^{(i)}\|$, we approximate the model at arbitrary $n \in \{0, \dots, N-1\}$ as:

$$\begin{aligned} \mathbf{x}^{n+1} - \mathbf{x}_\text{tr}^{n+1} &\approx \left.\frac{\partial \boldsymbol{f}^{n+1}}{\partial \boldsymbol{\chi}^{n+1}}\right|_{\boldsymbol{\chi}=\boldsymbol{\chi}_\text{tr}} (\boldsymbol{\chi}^{n+1} - \boldsymbol{\chi}_\text{tr}^{n+1}) \\ &= \left.\frac{\partial \boldsymbol{f}^{n+1}}{\partial \mathbf{x}^{n+1}}\right|_{\mathbf{x}=\mathbf{x}_\text{tr}} (\mathbf{x}^{n+1} - \mathbf{x}_\text{tr}^{n+1}) + \left.\frac{\partial \boldsymbol{f}^{n+1}}{\partial \mathbf{x}^{n}}\right|_{\mathbf{x}=\mathbf{x}_\text{tr}} (\mathbf{x}^{n} - \mathbf{x}_\text{tr}^{n}) + \left.\frac{\partial \boldsymbol{f}^{n+1}}{\partial \mathbf{u}^{n+1}}\right|_{\mathbf{u}=\mathbf{u}_\text{tr}} (\mathbf{u}^{n+1} - \mathbf{u}_\text{tr}^{n+1}) \\ \mathbf{y}^{n+1} - \mathbf{y}_\text{tr}^{n+1} &\approx + \left.\frac{\partial \boldsymbol{h}^{n+1}}{\partial \boldsymbol{\gamma}^{n+1}}\right|_{\boldsymbol{\gamma}=\boldsymbol{\gamma}_\text{tr}} (\boldsymbol{\gamma}^{n+1} - \boldsymbol{\gamma}_\text{tr}^{n+1}) \\ &= \left.\frac{\partial \boldsymbol{h}^{n+1}}{\partial \mathbf{x}^{n+1}}\right|_{\mathbf{x}=\mathbf{x}_\text{tr}} (\mathbf{x}^{n+1} - \mathbf{x}_\text{tr}^{n+1}) + \left.\frac{\partial \boldsymbol{h}^{n+1}}{\partial \mathbf{u}^{n+1}}\right|_{\mathbf{u}=\mathbf{u}_\text{tr}} (\mathbf{u}^{n+1} - \mathbf{u}_\text{tr}^{n+1}) \end{aligned} \tag{5.20}$$

This system provides a linear approximation of the original model, for which the adjoint method can be easily implemented. Constructing a gradient of the cost function on the basis of this model, a search direction can then be established for the optimization procedure, without requiring access to the model code.

5.3 Numerical implementation

5.3.1 Argument scaling

Using the RBF method, the value of the interpolated function $\mathbf{f}$ is computed for new arguments $\boldsymbol{\chi}$, based solely on the radial distance of these arguments from the sample arguments $\boldsymbol{\chi}^{(i)} \in \Gamma_{\text{RBF}}$. In our implementation, however, these argument vectors are comprised of input and state values, which in turn encompass pressure and saturation values. Each of these variables exist at different scales, where the input is bounded by the physical well limitations, the saturation must be a value between 0 and 1, and the pressure varies with the implemented units. Therefore, upon computing distances between different argument vectors, the contribution of each of these variables to the overall distance will not be equal, and thus neither will their contributions in computing the weighting vectors. To resolve this, the states and inputs were scaled whenever constructing the arguments:

$$\begin{aligned} [\boldsymbol{\chi}^{(i),n+1}]^T &:= \left(\tfrac{1}{c_p}[\mathbf{x}_p^{(i),n}]^T \quad \tfrac{1}{c_s}[\mathbf{x}_s^{(i),n}]^T \quad \tfrac{1}{c_p}[\mathbf{x}_p^{(i),n+1}]^T \quad \tfrac{1}{c_s}[\mathbf{x}_s^{(i),n+1}]^T \quad \tfrac{1}{c_{\mathrm{u}}}[\mathbf{u}^{(i),n+1}]\right) \\ [\boldsymbol{\gamma}^{(i),n+1}]^T &:= \left([\tfrac{1}{c_p}\mathbf{x}_p^{(i),n+1}]^T \quad \tfrac{1}{c_s}[\mathbf{x}_s^{(i),n+1}]^T \quad \tfrac{1}{c_{\mathrm{u}}}[\mathbf{u}^{(i),n+1}]^T\right) \qquad n = 0, \ldots, N-1 \end{aligned} \tag{5.21}$$

In this definition, the constants c_p, c_s and c_{u} are factors used to scale respectively the pressure, saturation, and input values. A natural choice for these would be the range covered by the corresponding values, which are readily available for the input and saturation from the physical bounds imposed on these. Unfortunately, such bounds are not applicable for the pressure, offering no natural scaling factor. Instead, for this thesis, all scaling factors were computed based on the samples used in the interpolation. Specifically, computing mean vectors $\hat{\mathbf{x}}_p^n, \hat{\mathbf{x}}_s^n$ and $\hat{\mathbf{u}}^n$ from the samples in Γ_{RBF}, constants were determined as the greatest distances between the samples and their means:

$$\begin{aligned} c_p &= \max_{i\in\Gamma}\left(\max_{n\in\{1,\ldots,N\}}\{\|\mathbf{x}_p^{(i),n} - \mathbf{x}_p^n\|_\infty\}\right) \\ c_s &= \max_{i\in\Gamma}\left(\max_{n\in\{1,\ldots,N\}}\{\|\mathbf{x}_s^{(i),n} - \mathbf{x}_s^n\|_\infty\}\right) \\ c_{\mathrm{u}} &= \max_{i\in\Gamma}\left(\max_{n\in\{1,\ldots,N\}}\{\|\mathbf{u}^{(i),n} - \mathbf{u}^n\|_\infty\}\right) \end{aligned} \tag{5.22}$$

Here, the means were computed according to equation (4.19). Based on these constants, the arguments were computed as (5.21), to which the interpolation techniques could then be applied as described before. Performing the RBF-TPWL method based on these arguments, the resulting derivatives would also be scaled, expressed in terms of the original arguments as:

$$\begin{aligned} \frac{\partial \mathbf{f}^{n+1}}{\partial \boldsymbol{\chi}^{n+1}} &= \left(c_p \tfrac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}_p^n} \quad c_s \tfrac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}_s^n} \quad c_p \tfrac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}_p^{n+1}} \quad c_s \tfrac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{x}_s^{n+1}} \quad c_{\mathrm{u}} \tfrac{\partial \mathbf{f}^{n+1}}{\partial \mathbf{u}^{n+1}}\right) \\ \frac{\partial \mathbf{h}^{n+1}}{\partial \boldsymbol{\gamma}^{n+1}} &= \left(c_p \tfrac{\partial \mathbf{h}^{n+1}}{\partial \mathbf{x}_p^{n+1}} \quad c_s \tfrac{\partial \mathbf{h}^{n+1}}{\partial \mathbf{x}_s^{n+1}} \quad c_{\mathrm{u}} \tfrac{\partial \mathbf{h}^{n+1}}{\partial \mathbf{u}^{n+1}}\right) \qquad n = 0, \ldots, N-1 \end{aligned} \tag{5.23}$$

From these derivatives, the individual derivatives with respect to the state and input at each time could be easily extracted.

5.3.2 Domain decomposition

Throughout the experiments conducted for this thesis, RBF interpolation has always been preceded by application of a subdomain POD method. Implementing this method, the model domain Ω is divided into N_D subdomains Ω^d, in each of which a reduced orthogonal basis $\tilde{\mathbf{\Phi}}^d \in \mathbb{R}^{N_{\mathrm{x}}^d \times \tilde{N}_{\mathrm{x}}^d}$ is constructed. Applying RBF interpolation based on the resulting reduced system, lower order surrogate models can then be constructed individually for each subdomain.

In performing RBF interpolation in each subdomain, the domain decomposition can be exploited to reduce the required computational effort, whilst potentially increasing the accuracy of the model. To illustrate this, we consider an arbitrary grid block $k \in \{1, \ldots, N_g\}$ at time step $n \in \{1, \ldots, N\}$, in a situation with an input $\mathbf{u} \in \mathbb{R}^{N_{\mathrm{u}}}$. When performing the full RBF interpolation as described before, the pressure and saturation values at this grid block will be extrapolated from the full states $\mathbf{x}^{(i),n-1} \in \mathbb{R}^{N_{\mathrm{x}}}$ corresponding to sample inputs $\mathbf{u}^{(i)}$. This enforces a correlation between the unknown variables at all the grid blocks throughout the domain, potentially introducing artificial correlations between otherwise (practically) independent variables [9]. Introducing a domain decomposition can help resolve this issue, as well as reducing the cost of the method.

In our implementation of the subdomain RBF interpolation, only grid blocks in neighboring domains were assumed to be correlated. That is, considering an arbitrary domain $d \in \{1, \ldots, N_D\}$, the state vector was assumed to satisfy a system of the form:

$$\mathbf{x}^{d,n+1} = \boldsymbol{\mathcal{L}}^{n+1}(\mathbf{x}^{d,n}, \mathbf{x}^{sd,n+1}, \mathbf{u}) \qquad n = 0, \ldots, N-1 \tag{5.24}$$

Here, vector $\mathbf{x}^{sd,n+1} \in \mathbb{R}^{N_{\mathrm{x}}^d}$ at each time step provides the state variables in the subdomains neighboring and including the domain d. For example for domain 6 in figure 5.1, this state would comprise only variables in domains $2, 5, 6, 7$ and 10 at the next time step, whilst for domain 12, only variables in domains $8, 11$ and 12 would be used. Collecting the arguments of function $\boldsymbol{\mathcal{L}}$, RBF interpolation can then be performed in each domain, based on an argument vector:

$$[\boldsymbol{\chi}^{d,n+1}]^T := \left([\mathbf{x}^{d,n}]^T \quad [\mathbf{x}^{sd,n+1}]^T \quad [\mathbf{u}]^T\right) \tag{5.25}$$

Constructing derivative matrices based on these arguments, only information from neighboring domains is considered, thus reducing the risk of introducing spurious correlations. Furthermore, excluding these large-distance correlations, implementing the RBF interpolation in this manner also reduces the required computational effort. This reduction becomes more significant when more subdomains are used, as this decreases the number of included correlations. However, this also introduces the risk of omitting correlations essential to the full-order model, thereby potentially decreasing the accuracy. For this reason, a 4×4 domain decomposition has always be implemented, amounting to a moderate value of 16 subdomains.

Similarly to the state vectors, the outputs can also be divided over the subdomains, and assumed to depend only on nearby results. In our case, the outputs are comprised of observed quantities at the different wells, suggesting only subdomains in which such a well is located to provide an output. For such a domain d including a well, the corresponding output was therefore assumed only to depend on the variables within this domain:

$$\mathbf{y}^{d,n+1} = \boldsymbol{\hbar}^{n+1}(\mathbf{x}^{d,n+1}, \mathbf{u}) \qquad n = 0, \ldots, N-1 \tag{5.26}$$

Including only these local state variables , an interpolated model for the output in each subdomain containing a well could then be constructed through the RBF method as before.

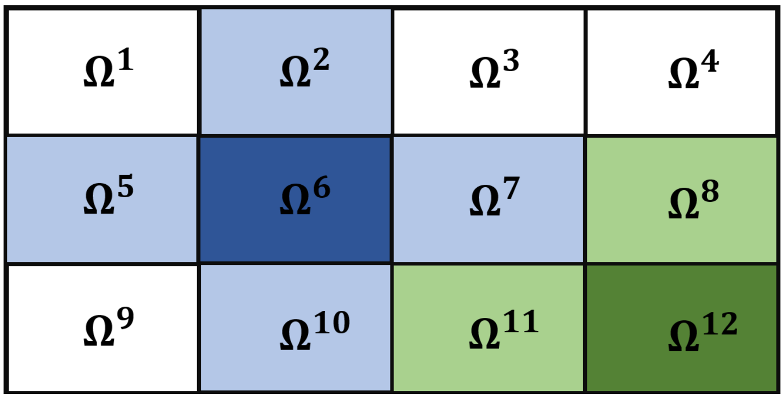


Figure 5.1: 3×4 domain decomposition

5.3.3 Sample computation

Constructing the TPWL approximation through RBF interpolation, derivative matrices of the model can be computed at different sample inputs $\mathbf{u}^{(i)} \in \Gamma_{\text{RBF}}$, allowing the full-order model at any input $\mathbf{u}$ to be approximated by extrapolating the results from the closest sample $\mathbf{u}_{\text{tr}} \in \Gamma_{\text{RBF}}$. In order to optimize an arbitrary cost function, the adjoint method can then be applied to the reduced-order model (5.20), to construct a gradient of the cost function at any input $\mathbf{u}$. Here, the derivatives of the model functions at such an input $\mathbf{u}$ depend only on which sample $\mathbf{u}^{(i)} \in \Gamma_{\text{RBF}}$ is closest. Since the adjoint gradient in turn is computed solely on the basis of these derivatives, this gradient too will depend only on the sample input closest to which it is being computed. The accuracy of the gradient will therefore depend strongly on the distance from each of the RBF samples.

In order to minimize computational effort, in our implementation, the same samples $\mathbf{u}^{(i)}$ used to construct the POD basis were also always used to perform the RBF interpolation. At any stage, these samples were determined considering a best guess $\mathbf{u}_b$ of the input, around which N_{POD} perturbed inputs $\mathbf{u}^{(i)} = \mathbf{u}_b + \epsilon_{\text{POD}}\delta\mathbf{u}^{(i)}$ were computed. Here, the perturbations $\epsilon_{\text{POD}}\delta\mathbf{u}^{(i)}$ were randomly generated to cover a hypercube of size ϵ_{POD} around the background. Using these samples to construct the TPWL approximation, approximate adjoint gradients of the cost function can be computed at the background input $\mathbf{u}_b$, as well as at the N_{POD} samples in the surrounding hypercube. However, considering an arbitrary input $\mathbf{u}$ not amongst these samples, neither of these approximations likely provides an accurate gradient.

To test the accuracy of the different gradients, an experiment has been conducted considering 16 random input samples $\mathbf{u}^{(k)}$ of the Kanaal model, of which the results were collected in a set Γ_{FD}. At each of these samples, the finite difference gradient (3.26) of the cost function (2.28) was computed, using a perturbation size $\epsilon_{\text{FD}} = 10^{-2}$. Next, choosing $\mathbf{u}_b = \mathbf{u}^{(k)}$ for $k \in \Gamma_{\text{FD}}$, the POD and consecutively the RBF method were performed around the background input $\mathbf{u}_b$. Herein, $N_{\text{POD}} = 25$ perturbations $\delta\mathbf{u}^{(i)}$ to the input $\mathbf{u}_b$ were used, scaled with factors $\epsilon_{\text{POD}} = 0.05$, $\epsilon_{\text{POD}} = 0.20$, and $\epsilon_{\text{POD}} = 0.5$. At each of these 25 POD samples, as well as at the background sample $\mathbf{u}_b$, an approximate gradient of the cost function (2.28) was then computed, applying the adjoint method to the reduced-order TPWL approximation. To compare these gradients to the finite difference results, the angle between them was computed as:

$$\theta = \arccos\left(\frac{[\boldsymbol{\nabla}_{\text{ROM}}J]^T \cdot [\boldsymbol{\nabla}_{\text{FD}}J]}{\|\boldsymbol{\nabla}_{\text{ROM}}J\|_2 \|\boldsymbol{\nabla}_{\text{FD}}J\|_2}\right) \tag{5.27}$$

Here, $\boldsymbol{\nabla}_{\text{ROM}}$ denotes the reduced-order model adjoint approximation of the gradient, with $\boldsymbol{\nabla}_{\text{FD}}$ denoting the finite difference approximation. Assuming the finite difference approximation to be highly accurate, this angle provides a measure of the accuracy of the ROM gradient, where a smaller angle corresponds to a better approximation. Computing this angle at the different finite difference samples, results were attained such as those displayed in figure 5.2.

In figure 5.2, the first sample from set Γ_{FD} is selected as background sample for the POD, such that $\mathbf{u}_b = \mathbf{u}^{(1)} \in \Gamma_{\text{FD}}$. In addition, the second POD sample is set to be equal to the second sample from this set, such that both the center sample and one POD perturbation coincide with a sample from Γ_{FD}. Each of the three graphs then displays the angle between the finite difference gradient at a sample in Γ_{FD}, compared to the adjoint gradient computed at each of the 26 POD samples, where sample 0 corresponds to the background input. These results are shown for each of the considered POD perturbation sizes, with the star $*$ indicating the POD sample closest to each finite difference sample.

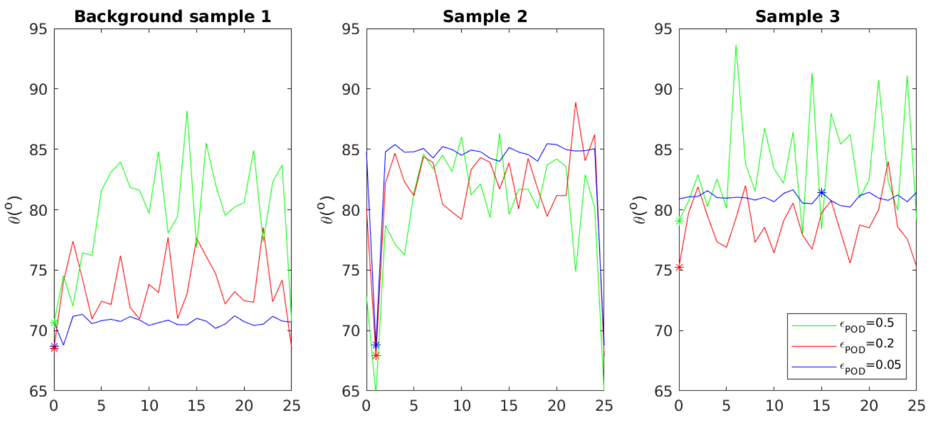


Figure 5.2: Angles between finite difference and reduced adjoint gradient at different inputs

The figure shows that, overall, the angle between the finite difference and reduced adjoint gradient is fairly large. When the input sample $\mathbf{u}^{(k)} \in \Gamma_{\text{FD}}$ does not coincide with a training sample $\mathbf{u}^{(k)} \in \Gamma_{\text{RBF}}$, the adjoint gradient is almost orthogonal to the finite difference one, independent of which training sample is used. This effect does not depend strongly on the size of the perturbation used, though a more moderate perturbation size does provide slightly lower angles at inputs away from the background sample. This is likely due to the fact that the greater perturbation size covers a higher variety of different inputs, and thus also provides a better estimate of gradients at unknown samples. Conversely, the accuracy of the gradient at the background sample seems to increase as the samples are placed closer together, as more information near this background is incorporated. This accuracy is also much higher than for samples not included in the POD perturbations, assuming a much lower angle at the background input than at sample 3. A similar result is observed at the second sample, which is also amongst the POD perturbations, and also displays significantly lower angles at the appropriate POD sample.

The results suggest that the accuracy of the RBF approximated gradients are highest at the actual RBF samples. This is unsurprising, as the accuracy of any RBF interpolation decreases with the distance from the samples. For this same reason, a moderate perturbation size will also provide better gradients at inputs away from the RBF samples, as a perturbation size that is too large or too small will generally increase the distance between these gradients and their closest sample. For this reason, the implemented optimization procedure has always been started using $\epsilon_{\text{POD}} = \frac{1}{4}(\text{u}_{\text{max}} - \text{u}_{\text{min}})$. Over the course of the algorithm, however, this perturbation size would be decreased, as the domain around $\mathbf{u}_b$ in which the optimum would be sought decreased as well.

5.3.4 Gradient recomputation

From figure 5.2, it follows that the gradient computed at perturbations of the background input $\mathbf{u}_b$, are significantly more accurate than those at arbitrary inputs. However, searching along the gradient of the background input in a high-dimensional setting $\mathbb{R}^{N_{\text{u}}}$, the gradient information from other samples is unlikely to become relevant. To avoid numerical inefficiency, therefore, the gradient at these perturbations was not computed during the implemented optimization procedure. Instead, we considered computing the gradient at a new input $\mathbf{u}$, without employing new samples around this input.

In establishing a new gradient, we can distinguish three levels of recomputation. The first of these simply computes new derivative functions (5.15) at the input $\mathbf{u}$, without adjusting the RBF weighting coefficients. Since the adjoint method can be applied directly on the basis of these derivatives, this only requires establishing the state and output vectors for this new input, and determining the distances of these results from the precomputed samples. A higher level of recomputation is established when also the weighting coefficients in (5.15) are redetermined, which requires adding the results from input $\mathbf{u}$ to the RBF sample set, and reperforming the RBF interpolation. In this manner, the new sample will belong to the RBF set, likely yielding a more accurate gradient, but also requiring a higher computational effort. Further increasing this effort, we can also add the results at input $\mathbf{u}$ to the POD snapshot matrices, and reconstruct the reduced bases. Implementing these bases, the RBF interpolation can also be performed using the new samples, constructing a gradient at $\mathbf{u}$ based on a new reduced-order approximation. However, since the accuracy of the POD method was not observed to increase with the number of samples, the effects of reperforming it are expected to be minimal.

To test the benefit of computing a new gradient, once more a random background input $\mathbf{u}_b$ was considered for the Kanal model, along with the corresponding finite difference gradient. Next, 15 samples $\mathbf{u}^{(k)} \in \Gamma_{\text{line}}$ along a line in the direction of this gradient were generated, and at each of these the finite difference gradient was computed. In addition, the ROM gradient at $\mathbf{u}_b$ was determined based on 25 samples around this input, using a perturbation size of $\epsilon_{\text{POD}} = 0.2$. The angle between this gradient and the finite difference gradient at each sample along the line was then computed according to equation (5.27), and plotted against the distance from the background as the black line in figure 5.3. In addition, the recomputed gradient was compared to the finite difference gradient at each input, given by the red line. The blue line and green markers provide this same comparison upon reperforming the RBF and respectively POD computation, after adding the input $\mathbf{u}$ along the line to the perturbation set. In this figure, a distance $\|\mathbf{u} - \mathbf{u}_b\|_\infty = 0.5$ corresponds to the input $\mathbf{u}$ reaching the boundary of the domain, at which point one of the elements of the input reaches its maximal absolute value.

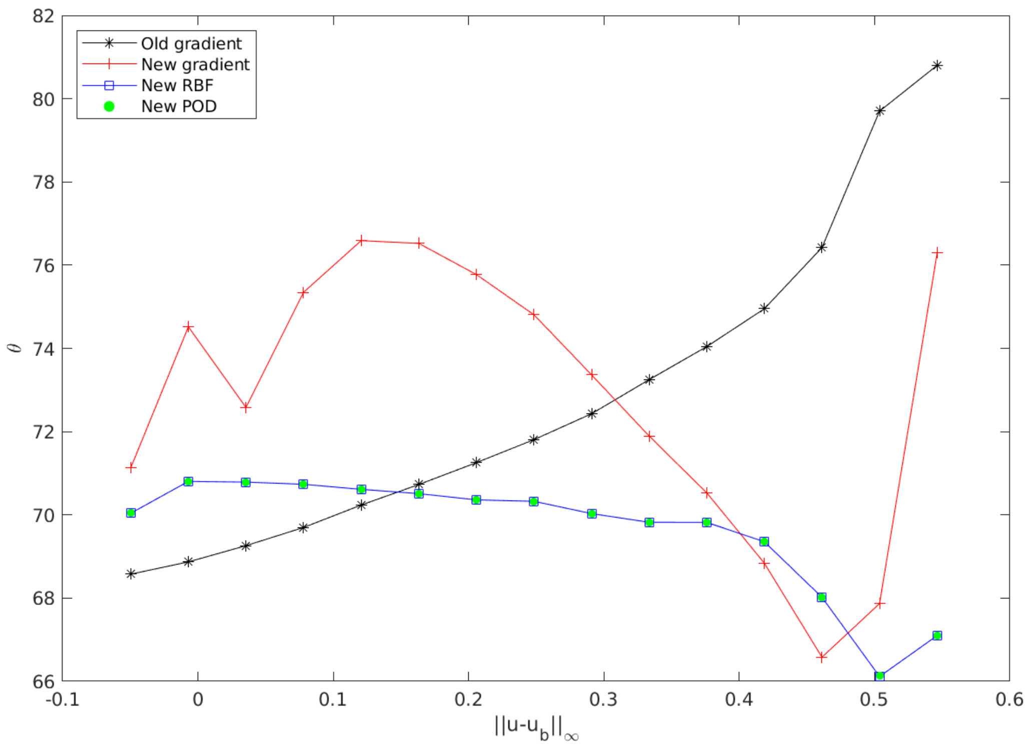


Figure 5.3: Angles between finite difference and reduced adjoint gradient upon reperforming computations at several inputs along a line

The figure shows that, as the distance $\|\mathbf{u} - \mathbf{u}_b\|_\infty$ between the input $\mathbf{u}$ and the background $\mathbf{u}_b$ increases, the mismatch between the finite difference gradient at this input and the adjoint gradient at the background also increases. Computing a new gradient using the same weighting coefficients, this mismatch can be decreased, although only at sufficient distance from the background input. This distance can be decreased further by recomputing the weighting vectors as well, yielding an accuracy exceeding the original one for larger values of $\|\mathbf{u}-\mathbf{u}_b\|_\infty$. Additionally reconstructing new POD bases, however, does not significantly affect the accuracy, practically matching the results attained when solely reperforming the RBF.

Overall, the RBF interpolated approximate gradient at an input $\mathbf{u}$ seems to improve when adding it to the RBF samples. Since performing the RBF interpolation is numerically expensive, however, this recomputation was only performed in the final method once a new optimal input $\mathbf{u}^*$ was established, resulting from a search algorithm starting in $\mathbf{u}_b$. If the resulting gradient did not provide a new optimum, new samples would be constructed as described for the POD method, and the process would be repeated.

5.3.5 Number of samples

Implementing the RBF method as described above, a gradient is initially established at some input $\mathbf{u}_b$ based on results of surrounding samples. Since the accuracy of this method at an arbitrary input depends on the distance from the RBF samples, this accuracy will depend strongly on the number of samples used. However, computing these samples requires running the full-order model, which is computationally expensive. Therefore, a trade-off between accuracy and cost must once again be considered.

To establish the effect of the number of samples on the accuracy of the method, a numerical experiment has been performed utilizing three arbitrary inputs $\mathbf{u}_b$ for the Kanaal model. At each input, the finite difference gradient was computed, and the POD method was applied using a perturbation size $\epsilon_{\text{POD}} = 0.25$, with $N_{\text{POD}} = 25$ samples. Performing the RBF interpolation on the basis of these same perturbations, gradients were established using the adjoint method as well, and the angles with the corresponding finite difference gradients were computed. Taking the mean of these angles, an accuracy of the method upon using 25 RBF samples was then established. Next, a single random perturbation was added to each RBF set, reconstructing the interpolations and the corresponding gradients, and recomputing the mean angle. Repeating this procedure until 100 RBF samples were considered, a measure of the accuracy of the method was attained upon using an increasing number of RBF samples, plotted in figure 5.4.

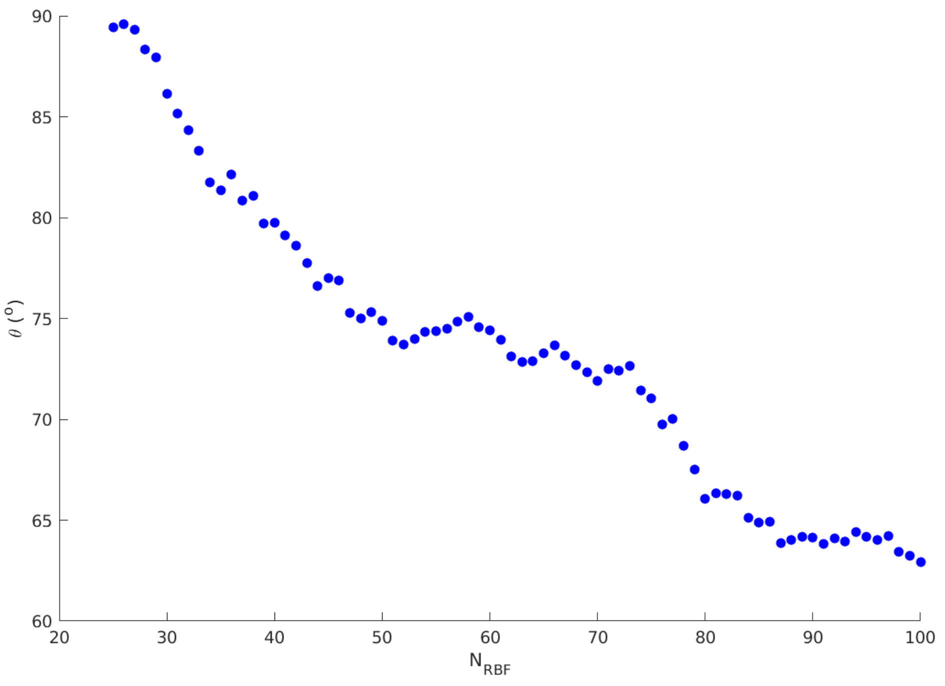


Figure 5.4: Angles between finite difference and reduced adjoint gradient based on an increasing number of samples

The figure shows that, as the number of samples is increased, the gradient at the background input $\mathbf{u}_b$ becomes more accurate. Here, the angle declines at a diminishing rate, achieving a decrease of 15^o upon adding the first 25 samples, but attaining less than a 5^o decrease upon adding an additional 25 samples. Even when using 100 samples, the accuracy of the method remains limited, with an angle of more than 60^o between the reduced adjoint and finite difference gradient.

Incorporating these results, the final algorithm for this thesis has only been implemented using 50 samples or less. This value was chosen as the computational effort increases rapidly with the number of samples, whereas the resulting increase in accuracy diminishes. Using such low number of samples, the reduced adjoint gradient will differ greatly from the finite difference gradient, potentially even being orthogonal. Nevertheless, so long as this angle remains smaller than 90^o, the approximate gradient will point in a direction of increase of the cost function. Moreover, similarly large angles are common for stochastic gradients [27], and tests with the specific EnOpt method implemented according to (3.33) also yields angles around 80^o when using only a small amount of samples.

5.3.6 The RBF derivative approximation algorithm

Algorithm 3 below summarizes the RBF method for constructing approximate derivative matrices of a model function $\mathbf{f}(\boldsymbol{\chi}_1, \ldots, \boldsymbol{\chi}_{N_\chi})$, at the location of background arguments $\boldsymbol{\chi}^{(b)}$, based on samples $\boldsymbol{\chi}^{(i)}$ with corresponding results $\mathbf{x}^{(i)} = \mathbf{f}(\boldsymbol{\chi}^{(i)})$. To compute the various derivative matrices of system (5.1), this algorithm can be applied to functions $\mathbf{f}^{n+1}, \mathbf{h}^{n+1}$ at each time step $n = 0, \ldots, N-1$, in which case the arguments would be given by $(\boldsymbol{\chi}_1, \boldsymbol{\chi}_2, \boldsymbol{\chi}_3) = (\mathbf{x}^n, \mathbf{x}^{n+1}, \mathbf{u}^{n+1})$ and $(\boldsymbol{\chi}_1, \boldsymbol{\chi}_2) = (\mathbf{x}^{n+1}, \mathbf{u}^{n+1})$ respectively. Alternatively, this algorithm can be executed in a subdomain framework, constructing derivative matrices of model functions $\boldsymbol{\mathcal{L}}^{d,n+1}, \boldsymbol{\hbar}^{d,n+1}$ as described by (5.24) and (5.26) respectively, running the algorithm separately in each domain $d \in \{1, \ldots, N_D\}$, and at each time step $n \in \{0, \ldots, N-1\}$. In doing so, arguments $(\boldsymbol{\chi}_1, \boldsymbol{\chi}_2, \boldsymbol{\chi}_3) = (\mathbf{x}^{d,n}, \mathbf{x}^{sd,n+1}, \mathbf{u})$ and $(\boldsymbol{\chi}_1, \boldsymbol{\chi}_2) = (\mathbf{x}^{d,n+1}, \mathbf{u})$ would have to be provided, with corresponding results given by the state vector $\mathbf{x}^{d,n+1}$ and output vector $\mathbf{y}^{d,n+1}$. For the final POD-TPWL method, algorithm 3 is performed on reduced vectors $(\tilde{\boldsymbol{\psi}}^{d,n}, \tilde{\boldsymbol{\psi}}^{sd,n+1}, \boldsymbol{\xi})$, where $\tilde{\boldsymbol{\psi}}$ comprises the components of a state vector under some POD basis $\tilde{\boldsymbol{\Phi}}^d$, and $\boldsymbol{\xi}$ denotes a potentially reduced input vector.

In computing the derivative matrices, algorithm 3 relies solely on samples of the arguments and corresponding results to provide information. In practice, any input $\mathbf{u}^{(i)}$ for which the full-order model results are known could be provided as such a sample in the algorithm. However, as established in section 5.3.3, the quality of the RBF computed gradient at an input $\mathbf{u}_b$ diminishes when the samples are placed further away from this input. Therefore, to minimize the computational cost, the provided samples were always the same as those used for the POD reduction. These samples always corresponded to simulation results for inputs around some background input $\mathbf{u}_b$, given by the optimal input guess at any stage of the final POD-TPWL algorithm. In this final algorithm, the background input $\mathbf{u}_b$, with corresponding state information, would also always be provided as background arguments $\boldsymbol{\chi}^{(b)}$ in algorithm 3. Through this algorithm, derivative matrices of the model functions at the optimal input $\mathbf{u}_b$ could then be approximated, allowing an adjoint methodology to be applied to construct an approximate gradient of the cost function at this input. In this manner, at each step of the POD-TPWL algorithm, a search direction could be computed at the current optimal input guess. Based on this search direction, a new guess could then be established, iteratively optimizing without requiring access to the model code.

Algorithm 3: Non-intrusibe RBF derivative approximation

```
**Input:** Background arguments $\boldsymbol{\chi}_1^{(b)}, \dots, \boldsymbol{\chi}_{N_\chi}^{(b)} \in \Gamma_{\mathrm{RBF}}$, Sample arguments
        $\boldsymbol{\chi}_1^{(i)}, \dots, \boldsymbol{\chi}_{N_\chi}^{(i)} \in \Gamma_{\mathrm{RBF}}$, Sample model results $\mathbf{x}^{(i)} \in \Gamma_{\mathrm{RBF}}$
**Result:** Approximate model derivatives $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\chi}_\ell}$ at the background arguments
1  Define multi-quadratic function $\theta(r,\epsilon) = \sqrt{(\epsilon r)^2 + 1}$;
2  Construct scaling factors $c_\ell = \max_{i \in \Gamma_{\mathrm{RBF}}} \Big( \max_{n \in \{1,\dots,N\}} \{ \|\boldsymbol{\chi}_\ell^{(i)} - \boldsymbol{\chi}_\ell^{(b)}\|_\infty \} \Big)$ for all
   $\ell = 1, \dots, N_\chi$;
3  Construct full argument vectors $[\boldsymbol{\chi}^{(i)}]^T = \Big[ \frac{1}{c_1} [\boldsymbol{\chi}_1^{(i)}]^T, \dots, \frac{1}{c_{N_\chi}} [\boldsymbol{\chi}_{N_\chi}^{(i)}]^T \Big]$
4  **for** $k \in \Gamma_{\mathrm{RBF}}$ **do**
5      Collect sample results $\mathbf{x}^{(i)} \in \Gamma_{\mathrm{RBF} \setminus \{k\}}$ column-wise in matrix $\tilde{\boldsymbol{X}}^{(k)}$;
6      Compute sample distances $r_{ij}^{(k)} = \|\boldsymbol{\chi}^{(i)} - \boldsymbol{\chi}^{(j)}\|_2$ for $i,j \in \Gamma_{\mathrm{RBF} \setminus \{k\}}$;
7      Collect distances in matrix $\tilde{\mathbf{R}}^{(k)}$;
8      Define $\tilde{\boldsymbol{\Theta}}^{(k)}(\epsilon) = \theta(\tilde{\mathbf{R}}^{(k)}, \epsilon)$ as function $\theta(r,\epsilon)$ applied element-wise to $\tilde{\mathbf{R}}^{(k)}$;
9      Define matrix $\big[\tilde{\boldsymbol{\mathcal{W}}}^{(k)}(\epsilon)\big]^T = \Big[ [\tilde{\boldsymbol{\Theta}}^{(k)}(\epsilon)]^T \Big]^{-1} [\tilde{\boldsymbol{X}}^{(k)}]^T$ as function of $\epsilon$;
10     Extract columns $\tilde{\boldsymbol{\omega}}_i^{(k)}(\epsilon)$ for $i \in \Gamma_{\mathrm{RBF} \setminus \{k\}}$ from $\tilde{\boldsymbol{\mathcal{W}}}^{(k)}(\epsilon)$;
11     Compute distances $r_i^{(k)} = \|\boldsymbol{\chi}^{(k)} - \boldsymbol{\chi}^{(i)}\|_2$ for $i \in \Gamma_{\mathrm{RBF} \setminus \{k\}}$;
12     Define $\tilde{\boldsymbol{x}}^{(k)}(\epsilon) = \sum_{i \in \Gamma_{\mathrm{RBF} \setminus \{k\}}} \tilde{\boldsymbol{\omega}}_i^{(k)}(\epsilon) \cdot \theta(r_i^{(k)}, \epsilon)$ as function of $\epsilon$;
13     Define $L^{(k)}(\epsilon) = \|\mathbf{x}^{(k)} - \tilde{\boldsymbol{x}}^{(k)}(\epsilon)\|_2$ as function of $\epsilon$;
14 **end**
15 Define $L(\epsilon) = \sum_{k \in \Gamma_{\mathrm{RBF}}} L^{(k)}(\epsilon)$;
16 Determine an $\epsilon^* \in \big[10^{-2}, 10^2\big]$ minimizing $L(\epsilon)$;
17 Define $\theta(r) = \sqrt{(\epsilon^* r)^2 + 1}$;
18 Collect sample results $\mathbf{x}^{(i)} \in \Gamma_{\mathrm{RBF}}$ column-wise in matrix $\boldsymbol{X}$;
19 Compute sample distances $r_{ij} = \|\boldsymbol{\chi}^{(i)} - \boldsymbol{\chi}^{(j)}\|_2$ for $i,j \in \Gamma_{\mathrm{RBF}}$;
20 Collect distances in matrix $\mathbf{R}$;
21 Compute $\boldsymbol{\Theta} = \theta(\mathbf{R})$ as function $\theta(r)$ applied element-wise to $\mathbf{R}$;
22 Compute coefficient matrix $\big[\boldsymbol{\mathcal{W}}\big]^T = \big[\boldsymbol{\Theta}\big]^{-1} \big[\boldsymbol{X}\big]^T$;
23 Extract columns $\boldsymbol{\omega}_i$ for $i \in \Gamma_{\mathrm{RBF}}$ from $\boldsymbol{\mathcal{W}}$;
24 Compute distances $r_i^{(b)} = \|\boldsymbol{\chi}^{(b)} - \boldsymbol{\chi}^{(i)}\|_2$ for $i \in \Gamma_{\mathrm{RBF}}$;
25 Compute derivative vectors $\frac{\partial \theta(r_i^{(b)})}{\partial \boldsymbol{\chi}_\ell^{(b)}} = c_\ell (\epsilon^*)^2 \frac{[\boldsymbol{\chi}_\ell^{(b)}]^T - [\boldsymbol{\chi}_\ell^{(i)}]^T}{\theta(r_i^{(b)})}$ for $\ell = 1, \dots, N_\chi$ and
   $i \in \Gamma_{\mathrm{RBF}}$;
26 Compute model derivatives $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\chi}_\ell}\Big|_{\boldsymbol{\chi}_\ell = \boldsymbol{\chi}_\ell^{(b)}} = \sum_{i \in \Gamma} \boldsymbol{\omega}_i \cdot \frac{\partial \theta(r_i^{(b)})}{\partial \boldsymbol{\chi}_\ell^{(b)}}$ for $\ell = 1, \dots, N_\chi$;
```

Chapter 6

Reduction of the Input Dimensionality

In this chapter, several methods for reducing the size of the input vector are discussed. Numerical experiments are conducted to test the benefits of adjusting the input, and a scheme is introduced for adapting the adjustments during the optimization. In addition, transformation functions are introduced to assure the input values remain within the physical bounds.

6.1 Dimensionality reduction

Using the methods described in the previous two chapters, a reduced-order linear approximation of any model can be constructed, allowing for more feasible application of an adjoint method. This in turn allows for computationally less demanding optimization of an arbitrary cost function $J(\mathbf{u})$ with respect to the input $\mathbf{u}$ of the model, as this input remains invariant under application of these methods. In many cases, however, the number of dimensions N_{u} spanned by the possible inputs is also considerable. For example in the field of reservoir flow, where we assume to have N_w wells which can be controlled at T_w times, the total number $N_u = N_w \cdot T_w$ of input variables can be very large when considering a large number of wells over an extended period of time. Therefore, an additional reduction in computational effort might be achieved addressing the dimensionality of the input.

Since the size of the gradient $\boldsymbol{\nabla} J$ depends directly on the size of the input $\mathbf{u}$, the numerical cost of any gradient-based optimization procedure will increase with the number of inputs. The one-sided finite difference approximation, for example, requires an additional evaluation of the cost function and thus the underlying model for every additional input parameter. Also for the adjoint method, relying on derivative matrices of the model functions with respect to $\mathbf{u}$, decreasing the size of this vector will allow for less expensive optimization. In addition, the optimization process is generally less complicated in a lower dimensional space. Indeed, reducing the number of parameters on which to optimize produces a more smooth computational domain, allowing faster and potentially more accurate optimization. For these reasons, several methods for reducing the dimensionality of the input have been incorporated in the final algorithm, adjusting the controls via a reparameterization scheme.

6.2 Input reparameterization

Let $\mathbf{u}^n \in \mathbb{R}^{N_w}$ denote the input vector at time step $n \in \{1, \ldots, N\}$, with N_w in our case corresponding to the number of wells over which control is exerted. We assume that this input can only be adjusted at T_w time steps $n_1, \ldots, n_{T_w}$, with the input remaining constant between any two such times. The total control $\mathbf{u} \in \mathbb{R}^{N_u}$ with $N_u := N_w \cdot T_w$ is then shaped by the concatenation of all these controls at each of the control times, providing the overall vector to be optimized.

When performing reduced-scale optimization, the vector $\mathbf{u} \in \mathbb{R}^{N_\mathrm{u}}$ is replaced by a lower dimensional reparameterization $\boldsymbol{\xi} \in \mathbb{R}^{N_c}$, with $N_c \leq N_u$. Rather than optimizing the full input vector $\mathbf{u}$, the reduced control vector $\boldsymbol{\xi}$ can then be optimized, and the result can be transformed back into a physical control. Naturally this requires introducing an injective mapping $\mathcal{K} : \mathbb{R}^{N_c} \to \mathbb{R}^{N_\mathrm{u}}$ translating each control vector into a unique input vector. For ease of notation, we also introduce N corresponding mappings $\mathcal{K}^n : \mathbb{R}^{N_c} \to \mathbb{R}^{N_w}$ for $n = 1, \ldots, N$, directly transforming the control vector $\boldsymbol{\xi}$ into an input vector at each time step of the model. As the input does not change between any consecutive control times n_i and n_{i+1}, this mapping must yield a constant input vector for any n satisfying $n_i \leq n < n_{i+1}$:

$$\mathcal{K}^n(\boldsymbol{\xi}) = \mathbf{u}^n = \mathbf{u}^{n_i} = \mathcal{K}^{n_i}(\boldsymbol{\xi}) \qquad \text{where} \qquad n_i = \max_{n_i \leq n}\{n_i\} \tag{6.1}$$

As a result, even when using an identity mapping, such that $\boldsymbol{\xi} = \mathbf{u}$, the input will still be a piecewise constant function.

Reducing the size of the input vector, any input reparameterization scheme will also reduce the number of degrees of freedom for optimization. Consequently, reparameterizing the input will exclude solutions from the system, potentially pertaining certain optima. A proper choice of reparameterization is therefore crucial, where the method should match desired or expected results of the optimal input. Additionally, a multiscale method can be applied, adjusting the number N_c of controlled parameters during the optimization procedure, and thereby altering the freedom allowed in searching for an optimum. This approach will be briefly discussed in the last section.

6.2.1 Step function

Perhaps the most straightforward approach to reducing the number of controls is a method that will be referred to as step function reparameterization. Given that there are N_w wells for which input can be defined at T_w times, this method defines the value of the input at all wells but only at $T_c \leq T_w$ times, yielding a total of $N_c = N_w \times T_c$ control parameters. The value of the input at other times is simply set equal to the value specified at the previous control time, such that for arbitrary $\boldsymbol{\xi}$:

$$\mathcal{K}^n(\boldsymbol{\xi}) = \mathcal{K}^{n_i^c}(\boldsymbol{\xi}) \qquad \text{where} \qquad n_i^c = \max_{n_i^c \leq n}(n_i^c) \tag{6.2}$$

Here $n = 1, \ldots, N$ denote the timesteps of the model, and n_i^c for $i = 1, \ldots, T_c$ denote the times for which the control vector $\boldsymbol{\xi} \in \mathbb{R}^{N_c}$ actually provides a value of the input. The resulting full input is a step function of $T_c - 1$ steps at each well. Setting $T_c = T_w$, this control method is equivalent to optimization using all input parameters.

Fixing the input to remain constant between predefined times, the step function method significantly decreases the variety of attainable input functions when considering only few control times $T_c << T_w$. Nevertheless, the ease of implementation and unambiguous transformation offered by this method amounts to an overall very attractive reparameterization scheme. Specifically when considering multiscale optimization, where a variable number of controls is desired, the clarity of this method makes it a suitable candidate. In the experiments performed for this thesis, it has also only been applied within such a multiscale framework, increasing the number of steps taken in this method during the optimization.

6.2.2 Cubic spline interpolation

The second method considered for reparameterizing the input is cubic spline interpolation. As for the step function reparameterization, this method takes the control vector $\boldsymbol{\xi}$ to contain the values of the input at all N_w wells, but only at $T_c \leq T_w$ times. The full input is then determined separately for each well, by interpolating the values at the control times using a cubic spline method.

Cubic spline interpolation, as the name suggests, is a spline interpolation method constructing a piecewise polynomial function on the provided data, with each separate polynomial being of third order. For this method, we assume to have T_c data points (n_j, ξ_j), ordered increasingly in terms of the times n_j. Between each pair of consecutive times n_j, n_{j+1}, a third degree polynomial function is then constructed as:

$$P_j(n) = a_j n^3 + b_j n^2 + c_j n + d_j \tag{6.3}$$

To establish the values of the different coefficients in this function, the polynomials are required to match to second order at the different data points:

$$\begin{aligned}
&P_j(n_j) = P_{j-1}(n_j) = \xi_j &\qquad &P_j(n_{j+1}) = P_{j+1}(n_{j+1}) = \xi_{j+1}\\
&\left.\frac{dP_j}{dn}\right|_{n_j} = \left.\frac{dP_{j-1}}{dn}\right|_{n_j} & &\left.\frac{dP_j}{dn}\right|_{n_{j+1}} = \left.\frac{dP_{j+1}}{dn}\right|_{n_{j+1}}\\
&\left.\frac{d^2P_j}{dn^2}\right|_{n_j} = \left.\frac{d^2P_{j-1}}{dn^2}\right|_{n_j} & &\left.\frac{d^2P_j}{dn^2}\right|_{n_{j+1}} = \left.\frac{d^2P_{j+1}}{dn^2}\right|_{n_{j+1}}
\end{aligned} \tag{6.4}$$

Imposing these constraints can be expressed as a linear system, which can be easily solved to compute the coefficients a_j, b_j, c_j, d_j for each segment $[n_j, n_{j+1}]$ [10]. Using the polynomial functions, then, the values of the input can be computed at all times, interpolated from the prescribed control values ξ_j.

Interpolating between control parameters will yield smooth inputs over time, with values equal to those assigned by $\boldsymbol{\xi}$ at the control times. This allows the construction of a wide variety of input functions, using a small number of control points. Furthermore, the imposed smoothness also results in functions that might be preferable from a physical perspective, at least when using limited control times. Even in this situation, however, the final input function at all times will remain piecewise constant, as the input can be adjusted only at discrete times $n_1, \ldots, n_{T_w}$, being constant between consecutive well times. If the cubic spline interpolation method is used with $T_c = T_w$, this control method is equivalent to an optimization method without reparameterization.

6.2.3 Principle component analysis

Through principal orthogonal decomposition, the size of state vectors can be reduced by expressing them in terms of basis vectors, which are derived from observed realizations of the state. Similarly, the size of inputs can also be reduced by expressing them in terms of certain basis vectors. Unlike for the state vectors, however, the dominant basis vectors for the input have to be artificially introduced. This is because the inputs are free to be chosen, and as such any shape of the input function is, in theory, equally probable. Nevertheless, some input functions might be more desirable than others, allowing common patterns to be established. We will refer to this method as principal component analysis (PCA), to distinguish it from the POD applied to the states, although the underlying theory is identical.

In our implementation of the PCA method, correlations were imposed between the input variables at the different wells and times. Herein, we assumed the input value at an arbitrary well and time to be most strongly related to inputs at nearby wells, as well as those at closer times. To this end, let $\mathbf{r}^{j,n}$ for $j = 1, \ldots, N_w$ and $n = 1, \ldots, T_w$ be a vector containing the spatial and temporal coordinates of the jth well at the nth time step. Then we take arbitrary inputs $[\mathbf{u}^n]_j$ and $[\mathbf{u}^{n'}]_{j'}$ to be correlated with a covariance given by:

$$\operatorname{cov}([\mathbf{u}^n]_j, [\mathbf{u}^{n'}]_{j'}) = \exp\left(\frac{\mathbf{r}^{j,n} - \mathbf{r}^{j',n'}}{2\sigma^2}\right) \tag{6.5}$$

Using this function, the covariance between inputs at different wells and times decreases as a Gaussian function with the corresponding spatial and temporal distance between these inputs. Here, the parameter σ^2 determines how quickly this decay occurs, imposing a stronger correlation when using a larger value. Based on this covariance then, random snapshots of the inputs can be generated, allowing for a reduced basis to be approximated. Alternatively, this basis can be constructed directly from the eigenvectors of the covariance matrix, retaining only those with the greatest eigenvalues using some energy criterion [43]. Letting $\boldsymbol{\Xi} \in \mathbb{R}^{N_u \times N_c}$ denote the matrix consisting column-wise of these eigenvectors, a control vector $\boldsymbol{\xi} \in \mathbb{R}^{N_c}$ can then be introduced as components of a corresponding input $\mathbf{u}$ under matrix $\boldsymbol{\Xi}$:

$$\mathbf{u} = \mathcal{K}(\boldsymbol{\xi}) = \boldsymbol{\Xi}\boldsymbol{\xi} \tag{6.6}$$

Reducing the input in this manner, effectively a smoothing is introduced in space and time. The strength of this smoothing is determined by the assigned covariance between the inputs, where a greater covariance will allow for a smoother function. An increase in this covariance can be achieved by increasing the variance σ^2, as well as by decreasing the distances between the inputs. These distances are determined by the positions $\mathbf{r}^{j,n}$ assigned to each well at each time, defined as:

$$[\mathbf{r}^{j,n}]^T = \begin{pmatrix}\alpha_t[r_t^n] & \alpha_x[\mathbf{r}_x^j]^T\end{pmatrix} = \begin{pmatrix}\alpha_t \frac{\Delta x}{\Delta t} n & \alpha_x j_x & \alpha_x j_y & \alpha_x j_z\end{pmatrix}$$

In this definition, j_x, j_y, j_z denote the indices of the grid block in which the jth well is situated in respectively the x-, y- and z-direction. The values Δx and Δt denote spatial and temporal discretization steps, used to scale the timestep n to be of comparable magnitudes to the spatial indices. Finally, the constants α_t and α_x are implemented to scale the distances between inputs, thereby adjusting the strength of the correlation between these inputs. In our implementation, the correlation between inputs at the same well was always assumed to be stronger than that between different wells, imposed by taking $\alpha_t = 0.5$ and $\alpha_x = 1$.

Implementing the PCA method in this manner, the input is reparameterized on the basis of how strongly the different variables are expected to be correlated. By adjusting the position vectors of the different inputs, the inputs can be assigned to be completely independent, or all determined by each other. Although this offers a natural reduction in the size of the state, where stronger correlations allow for representation using fewer basis vectors, the overall choice of this covariance remains difficult. For example, in introducing a correlation between the inputs at different wells, it would physically make sense to consider the flow between these wells. However, incorporating this is practically unfeasible, as the flow depends on the model, and thus on the prescribed inputs. This problem can be avoided by implementing solely a correlation between inputs at the same well, using $\alpha_t << \alpha_x$. However, in such a scenario, a CSI or step-function method is generally more appealing, offering input functions that are more comprehensible or physically more desirable. Therefore the PCA method has not been extensively considered in the numerical experiments.

6.2.4 Single step control

As a final reparameterization method, we consider single step control. Similarly to step function control, this method assumes the input in time to change like a step function. However, this scheme is restricted to just one change in the input value for each well, and instead, the timestep at which this change occurs is considered as control parameter. Fixing the initial and final inputs at the wells, this yields a type of bang-bang control, where the input switches between predefined values and the time of this switch is to be optimized. Such a control method is physically appealing, especially if the input at all the wells is taken to switch at the same time, as this requires very limited physical action. Computationally, optimization involving a single control would also be very cheap, allowing efficient application of a finite difference method as the optimum is sought (at discrete points) along a single line. However, the restrictions upon the input in this case are unreasonable, likely excluding many significantly better input functions, such that the attained optimum deviates a lot from the actual global optimum. Therefore, additional control has always been permitted, by allowing the input at each well to swap at a different time. Furthermore, the initial and final values of the input at each well were implemented to be optimized as well, such that an optimal step function would be sought at each well. This control method therefore required optimizing $N_c = 3N_w$ values, restricting the input function at each well to be a step function, but optimizing both the initial and final value, as well as the time of the step.

6.3 Constraint enforcement

When computing the full inputs $\mathbf{u}^n$ based an arbitrary reparameterization, care should be taken to assure the physical limitations are not violated. For the step control methods, this can be easily achieved by assuring the reduced controls to not exceed certain bounds. For the PCA or CSI reparameterization though, imposing such bounds might unnecessarily exclude certain inputs, or still yield inputs that are physically infeasible. To avoid this, we have employed transformation functions.

In this thesis, a transformation function will be denoted as any function $v : \mathbb{R} \to [[\mathrm{u}_{\min}]_i, [\mathrm{u}_{\max}]_i]$, mapping an arbitrary real number to one satisfying the input restrictions, assumed to be equal for each well. In implementing such a function, the reduced control was always centered around zero, and given a natural spread of $\frac{1}{2}$ in each direction. This means that, an arbitrary control parameter ξ_i was generally fixed to exist in the domain $[-\frac{1}{2}, \frac{1}{2}]$, although it was also possible to allow greater values. Along this domain, the full input variables u_j must then be allowed to attain any value in the domain $[\mathrm{u}_{\min}, \mathrm{u}_{\max}]$, which is most easily assured using a linear transformation function:

$$v_L(\mathrm{u}) := \begin{cases} \mathrm{u}_{\min} & \text{for u} < -\frac{1}{2} \\ \mathrm{u}_{\min} + (\mathrm{u}_{\max} - \mathrm{u}_{\min})(\mathrm{u} + \frac{1}{2}) & \text{for u} \in [-\frac{1}{2}, \frac{1}{2}] \\ \mathrm{u}_{\max} & \text{for u} > \frac{1}{2} \end{cases} \tag{6.7}$$

This function transforms each input directly into one satisfying the constraints, where only adjustments of u within the domain $[-\frac{1}{2}, \frac{1}{2}]$ will affect the final input. Since the slope of this function is also constant within this domain, the size of such an adjustment in the input will also be directly proportional to the adjustment before transformation. At the boundaries $\mathrm{u} = \pm\frac{1}{2}$ on the other hand, this slope changes abruptly to zero, such that increasing or decreasing the value of u beyond this boundary no longer influences the transformed input. As a result, gradients with respect to the input will be free to point towards a boundary up to arbitrarily small distances from it, but are abruptly fixed to point away from or along it once the boundary is reached. This introduces discontinuities in any gradient-based optimization procedure, which might limit its performance. To avoid this, a smoother transformation function has also been employed as a Sigmoid function:

$$v_L(\mathrm{u}) := \begin{cases} \mathrm{u}_{\min} & \text{for u} < -\frac{1}{2} \\ \mathrm{u}_{\min} + (\mathrm{u}_{\max} - \mathrm{u}_{\min})\frac{1+\exp(-\alpha_S)}{1+\exp(-2\alpha_S \mathrm{u})} & \text{for u} \in [-\frac{1}{2}, \frac{1}{2}] \\ \mathrm{u}_{\max} & \text{for u} > \frac{1}{2} \end{cases} \tag{6.8}$$

This function has a greater slope near the center of the domain, which decreases as the boundaries are approached. The value of this slope is determined by the constant α_S, where a greater value corresponds to a steeper increase at the center. Using such a function, the greatest changes in the input function will be placed at the center of the domain. This encourages the optimization procedure to establish optima using less extreme values, as adjusting the control near the boundaries will not significantly affect the input, and thus also not the cost function. In our implementation, a value $\alpha_S = 10$ has been used, of which the corresponding function is plotted in figure 6.1. Using this parameter, practically any input can be achieved within the domain $[-\frac{1}{4}, \frac{1}{4}]$, with the slope of the function being almost zero outside of this domain. When using this function, therefore, a perturbation size $\epsilon_{\mathrm{POD}} = \frac{1}{8}$ was always initially taken for the POD and thus also RBF perturbations, spreading the samples halfway through the domain $[-\frac{1}{4}, \frac{1}{4}]$. Accordingly, if a linear transformation function was implemented, this perturbation size was set to $\epsilon_{\mathrm{POD}} = \frac{1}{4}$, as covering the full spectrum of input values for this transformation function requires controls in the full domain $[-\frac{1}{2}, \frac{1}{2}]$.

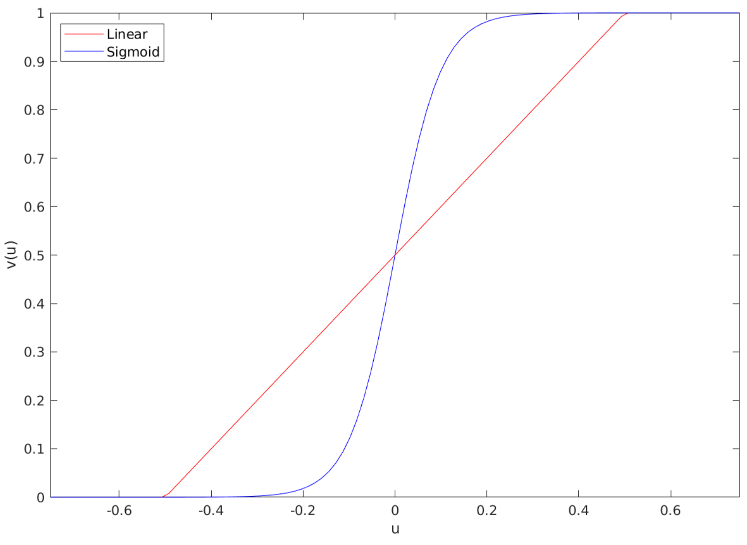


Figure 6.1: Linear and Sigmoid transformation function

6.4 Reduced input optimization

Using any of the described reparameterization schemes, the size of the input can be reduced to a desired extent, allowing for generally cheaper optimization. In addition, employing such a method might enhance the overall performance of the optimization method, as the difficulty of optimization is smaller when optimizing on fewer variables. For a gradient-based method, this improved performance lies in the construction of the gradient, which will be computationally cheaper as well as potentially more accurate as the size of the gradient decreases. For example, when constructing the gradient based on samples of the model, implementing a reduced control vector will cause these samples to be spread out over a lower-dimensional domain. Consequently, a small amount of samples should yield more accurate results when considering a reduced amount of input parameters, suggesting any sample based optimization scheme might benefit from reparameterizing the inputs.

To test whether a reduced control vector indeed improves the results for the reduced adjoint method described in the previous chapters, the CSI and step function reparameterization methods were implemented for the Kanaal model. For each, two control times were considered for all wells, reducing the number of degrees of freedom from $N_{\text{u}} = 135$ to $N_c = 18$. Next, 16 samples $\boldsymbol{\xi}^{(k)}$ of the reduced control vector were computed randomly in the computational domain for each scheme, and collected in a set Γ_{FD}. For each of these samples, the finite difference gradient was then computed, using a perturbation size $\epsilon_{\text{FD}} = 10^{-3}$. In addition, the POD-TPWL method was applied to compute a gradient at each sample $\boldsymbol{\xi}^{(k)}$, taking this sample as background control $\boldsymbol{\xi}_b = \boldsymbol{\xi}^{(k)}$. Herein, $N_{\text{POD}} = 25$ additional perturbations were consistently considered, using perturbation sizes $\epsilon_{\text{POD}} = 0.05$, $\epsilon_{\text{POD}} = 0.20$, and $\epsilon_{\text{POD}} = 0.5$. The angle between each of the resulting gradients was consecutively computed using (5.27). The mean of the resulting angles when using the first sample in Γ_{FD} as background are plotted in figure 6.2. In this figure, the results are displayed using both reparameterization schemes, as well as when using the full input.

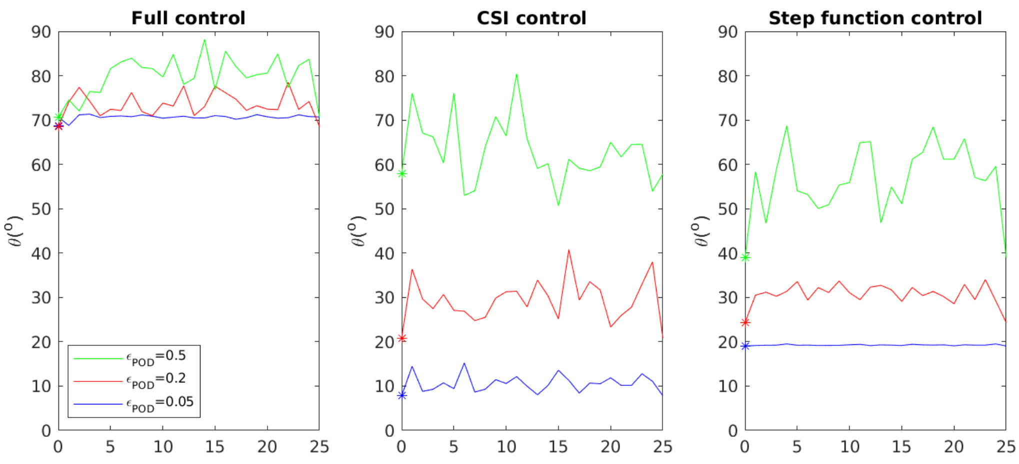


Figure 6.2: Angles between finite difference and reduced adjoint gradient using different reparameterization schemes

The figure shows that, reducing the number of parameters to optimize, the accuracy of the reduced adjoint gradient indeed increases. Both the CSI and step function reparameterization scheme return relatively small angles between the adjoint and finite difference gradient, displaying a significant decrease compared to the full control method. This effect is strongest for the smaller perturbation size, placing the RBF samples closer to the background. For these perturbations, the adjoint gradient at the background control also provides a more accurate approximation of that at the perturbations, as these perturbations lie very close to the background.

In addition to increasing the accuracy, implementing a reparameterization scheme also increases the discrepancy between results using different perturbation sizes. This is likely due to the fact that perturbations in the reduced control translate to even bigger perturbations in the full input vector. As a result, different perturbation sizes using smaller control vectors will yield increasingly different samples for the full input, in turn providing increasingly different gradients. This effect is stronger for the CSI method, where changes in the control will adjust the shape of the entire input function, whereas only two values are adjusted using step function control. Nevertheless, this complex adjustment does allow the CSI method to provide more accurate gradients than the step function method upon using a small perturbation size.

6.4.1 Multiscale methods

Implementing a reparameterization method, the overall efficiency of an optimization scheme can be greatly enhanced, as also shown to be the case for the introduced POD-TPWL method. However, this enhancement comes at the cost of a certain number of degrees of freedom, as fewer parameters are considered in the optimization procedure. As a result, a large portion of the physically possible inputs will be excluded when reparameterizing, potentially limiting the optimal objective function value that can be achieved. To avoid this, multiscale algorithms can be applied. These methods adjust the dimensionality of the control as part of the optimization process, optimizing based on varying degrees of freedom. Through this process, both an optimal reparameterization and a corresponding optimal control are established simultaneously. Many such methods have already been developed, using a variety of optimization methods, as well as considering different schemes to reduce the input dimensionality [8] [28]. These methods can, for example, adjust the reparameterization based on individual derivatives of the cost function with respect to each input parameter [26]. However, this still requires computing the full cost function gradient at each time step, which we aim to avoid through the reparameterization.

For this thesis, a rudimentary multiscale method has been applied, based on the cubic spline interpolation, and step function reparameterization. The aim of this method is to exploit the higher accuracy of the gradient in the reduced computational domain, without excluding more irregular inputs that are physically still allowable. To this end, the number of degrees of freedom would be increased between different loops of this optimization procedure. Herein, only a small amount of controls would initially be implemented, on the basis of which optimization would be performed. Once an optimal input was established, the control vector would be translated to a higher dimensional equivalent, and the optimization procedure was continued with more degrees of freedom. This could be repeated until the full input vector was optimized.

In optimizing in the manner described above, the more extreme behaviour of the cost function in the computational domain is mitigated, as contributions of different input variables to the cost function value are combined. Optimizing initially in this reduced space, the method should then guide the input to an area of higher objective function value. At this point, the dimensionality can be increased, allowing optimization at a finer scale, and the process can be repeated. However, fully optimizing the control vector at each scale will amount to a substantial computational effort. Moreover, the optimal control using fewer degrees of freedom will generally not correspond to an optimal control in the higher dimensional setting. For this reason, the dimensionality was always aimed to be increased before an actual optimum was established. To this end, the control at each level would be optimized until the gain of optimizing further was expected to be greater upon increasing the number of degrees of freedom. As criterion for this, we note that the gradient in each loop is based on perturbations around the current control. If the new control is still in the hypercube covered by the perturbations, this hypercube is expected to be the area in which the optimal control will lie. Therefore, once this situation would occur, the dimesionality would be increased, and the process could be repeated.

Chapter 7

The Subdomain POD-TPWL Adjoint Algorithm

In this chapter, the full sudomain POD-TPWL algorithm is outlined, implementing the techniques from the previous chapters. The iterative framework in which these techniques are applied is described, consisting of several nested loops, adjusting the search direction and the step size taken is this direction. Details regarding the exact application of each of the techniques are discussed, based on the results from earlier numerical experiments.

7.1 Initialization

Building on the results from previous chapters, we now construct the full algorithm implemented for this thesis. This algorithm aims to optimize some prespecified cost function $J(\mathbf{u})$, governed by a model of the form:

$$\begin{aligned} \mathbf{x}^0 &= \mathbf{x}^{\text{init}} \\ \mathbf{x}^{n+1} &= \mathbf{f}^{n+1}(\mathbf{x}^n, \mathbf{x}^{n+1}, \mathbf{u}^{n+1}) \\ \mathbf{y}^{n+1} &= \mathbf{h}^{n+1}(\mathbf{x}^{n+1}, \mathbf{u}^{n+1}) \qquad\qquad n = 0, \dots, N-1 \end{aligned} \tag{7.1}$$

Through the algorithm, a reduced-order linear surrogate of this model is constructed, to approximate a gradient of the objective function. Based on this gradient, an improved input is then established using the search algorithm described in chapter 3, at which point the entire process is repeated.

7.1.1 Input reparameterization

Before starting the optimization procedure, the algorithm performs an input reparameterization, obtaining a reduced control vector $\boldsymbol{\xi} \in \mathbb{R}^{N_c}$ to optimize. Here, any of the schemes described in the previous chapter can be used, although primarily the step function and cubic spline interpolation have been considered. The number N_c of controls can be set to any value smaller than or equal to the original number of inputs N_{u}, where usually a starting value of just 2 control times per well was chosen. This number of controls can also be increased during the algorithm, halting at the maximum number of controls.

In addition to the input reparameterization, a transformation function is also implemented, assuring the inputs to not exceed the imposed bounds. In general, a Sigmoid function was used to this end, stimulating the optimization procedure to remain more at the center of the domain. Letting $\mathcal{K}_{\text{bnd}}^n$ denote the reparameterization function at each time step after implementing this transformation, the system (7.1) becomes:

$$\begin{aligned}\mathbf{x}^{n+1} &= \mathbf{f}^{n+1}(\mathbf{x}^n, \mathbf{x}^{n+1}, \mathcal{K}_{\text{bnd}}^{n+1}(\boldsymbol{\xi})) \\ \mathbf{y}^{n+1} &= \mathbf{h}^{n+1}(\mathbf{x}^{n+1}, \mathcal{K}_{\text{bnd}}^{n+1}(\boldsymbol{\xi})) \qquad n = 0, \ldots, N-1\end{aligned} \tag{7.2}$$

7.1.2 Domain decomposition

With the reparameterization scheme chosen to reduce the size of the input, the algorithm starts the reduction of the state vector dimensions. The first step in this procedure is the domain decomposition, dividing the total grid Ω into N_D disjunct subdomains Ω^d. These subdomains are always implemented only along the horizontal plane, splitting the domain into $N_{D,x}$ and $N_{D,y}$ regions along the x- and y-direction respectively, where $N_D = N_{D,x} \cdot N_{D,y}$. Initially, these domains are sought to contain an equal number of grid blocks. Specifically, letting the grid be of dimensions $N_{g,x} \times N_{g,y} \times N_{g,z}$, such that $N_g = N_{g,x} \cdot N_{g,y} \cdot N_{g,z}$, each domain will be assigned dimensions:

$$\lfloor N_{g,x}/N_{D,x} \rfloor \times \lfloor N_{g,y}/N_{D,y} \rfloor \times N_{g,z} \tag{7.3}$$

However, the number of cells in each direction might not be divisible up to integer value by the corresponding desired number of subdomains. Therefore, if the remaining number of cells in either the x- or y-direction is less than or equal to 4, the algorithm adds these cells to the last subdomains. This means that, enumerating the subdomains according to their (x, y) position as (i, j), the remaining cells in x- and y-direction would be assigned to domains with $i = N_{D,x}$ and respectively $j = N_{D,y}$. Alternatively, if the extra number of cells in some direction exceeds 4, new subdomains are constructed to encompass these. In a situation with more than 4 additional grid blocks in both the x- and y-direction, this will thus amount to creating $N_{D,y} + N_{D,x} + 1$ new subdomains at the edge of the domain.

In our implementation, the domain has always been divided into 4 subdomains in each direction, with a potential extension to a fifth domain as described above. For each of these domains d, the algorithm collects corresponding pressure and saturation values into a matrix $\mathbf{x}^{d,n} \in \mathbb{R}^{2N_g^d}$ at each time step n, where N_g^d denotes the total number of grid blocks in this subdomain. In addition, the state variables of grid blocks in neighboring domains are added to $\mathbf{x}^{d,n}$ at each time, to construct a larger vector $\mathbf{x}^{sd,n}$. Herein, only subdomains sharing a face are assumed to be neighbors, such that any domain has a minimum of 2, and maximum of 4 neighboring domains, as displayed in figure 5.1. In performing the interpolation, the algorithm assumes the state variables in each subdomain to rely only on those in neighboring subdomains, yielding for arbitrary $d = 1, \ldots, N_D$ a system of the form:

$$\begin{aligned}\mathbf{x}^{d,n+1} &= \boldsymbol{\mathcal{F}}^{d,n+1}(\mathbf{x}^{d,n}, \mathbf{x}^{sd,n+1}, \boldsymbol{\xi}) \\ \mathbf{y}^{d,n+1} &= \boldsymbol{\mathcal{H}}^{d,n+1}(\mathbf{x}^{d,n+1}, \boldsymbol{\xi}) \qquad n = 0, \ldots, N-1\end{aligned} \tag{7.4}$$

7.1.3 Starting guess

Once an input reparameterization scheme is chosen, a corresponding initial guess $\boldsymbol{\xi}_b = \boldsymbol{\xi}^0 \in \mathbb{R}^{N_c}$ has to provided. This initial guess determines in what area the algorithm should start searching for an optimum, often being based on prior knowledge of the model or preceding optimization results. In our case, this initial guess was generally placed at the center of the computational domain, assuming no prior knowledge on the problem.

For the initial guess, the full-order model (7.2) is run, to compute the states $\mathbf{x}^n$ and outputs $\mathbf{y}^n$ at each time. These results, along with the corresponding control vector, are collected as the first sample in a set Γ. From these results, an initial value of the cost function is also computed.

7.2 Outer loops

During each outer loop of the algorithm, a new gradient of the cost function is computed at the current optimal guess $\boldsymbol{\xi}_b$ of the reduced control vector, reperforming both the POD method and RBF interpolation. Between outer loops, the size of this control vector might also be increased, adding more degrees of freedom for the optimization procedure.

7.2.1 Proper orthogonal decomposition

Each outer loop starts with reperforming the POD method, based on the new control $\boldsymbol{\xi}_b$. For this, a set of N_{POD} control vectors $\boldsymbol{\xi}^{(i)} \in \mathbb{R}^{N_c}$ are constructed, by randomly perturbing the background control according to a Latin hypercube sampling method. Herein, a perturbation size ϵ_{POD} is used, such that the samples are taken in a hypercube of corresponding size around the background control $\boldsymbol{\xi}_b$. Based on these perturbations, the full-order model (7.2) is then run, and the results are added to the sample set Γ. We will denote the set of solely POD samples as $\Gamma_{\text{POD}} \subseteq \Gamma$.

Using the samples collected in Γ, the POD method is applied. To this end, the average $\hat{\mathbf{x}}$ of the sample states $\mathbf{x}^{(j)} \in \Gamma$ is first computed according to (4.19), and subtracted from the states to attain shifted vectors $\Delta\mathbf{x}^{(j)} = \mathbf{x}^{(j)} - \hat{\mathbf{x}}$. These shifted vectors are then divided into pressure and saturation states $\Delta\mathbf{x}_{p,s}^{(j)}$ for the domains $d = 1, \ldots, N_D$, and combined into snapshot matrices as:

$$\begin{aligned} \mathbf{X}_p^d &= \begin{pmatrix} \Delta\mathbf{x}_p^{(1),d} & \ldots & \Delta\mathbf{x}_p^{(N_S),d} \end{pmatrix} \in \mathbb{R}^{N_g^d \times N_S} \\ \mathbf{X}_s^d &= \begin{pmatrix} \Delta\mathbf{x}_s^{(1),d} & \ldots & \Delta\mathbf{x}_s^{(N_S),d} \end{pmatrix} \in \mathbb{R}^{N_g^d \times N_S} \end{aligned} \tag{7.5}$$

Based on these snapshot matrices, the POD method is consecutively applied in each subdomain. For this, a singular value decomposition is performed for both matrices as:

$$\mathbf{X}_p^d = \mathbf{U}_p^d \boldsymbol{\Sigma}_p^d [\mathbf{V}_p^d]^T \qquad \mathbf{X}_s^d = \mathbf{U}_s^d \boldsymbol{\Sigma}_s^d [\mathbf{V}_s^d]^T \tag{7.6}$$

From these, bases for the pressure and saturation states are constructed as:

$$\boldsymbol{\Phi}_p^d = \mathbf{U}_p^d \tilde{\boldsymbol{\Sigma}}_p^d \qquad \boldsymbol{\Phi}_s^d = \mathbf{U}_s^d \tilde{\boldsymbol{\Sigma}}_s^d \tag{7.7}$$

Here, $\tilde{\boldsymbol{\Sigma}}_{p,s}^d$ denotes an adapted version of $\boldsymbol{\Sigma}_{p,s}^d$ containing solely its first N_g columns, removing the other columns containing only values equal to zero. These bases are then reduced, retaining only sufficient columns of each to satisfy a corresponding energy criterium. This was always taken to be 99.9% for the pressure variables, and 95% for the saturation values, maintaining the minimum amount $\tilde{N}_{p,s}$ of columns necessary to satisfy:

$$\sum_{i=1}^{\tilde{N}_p^d}[\sigma_p^d]_i^2 \geq \frac{99.9}{100}\sum_{i=1}^{N_p^d}[\sigma_p^d]_i^2 \qquad \sum_{i=1}^{\tilde{N}_s^d}[\sigma_s^d]_i^2 \geq \frac{95}{100}\sum_{i=1}^{N_s^d}[\sigma_s^d]_i^2 \tag{7.8}$$

Here, $N_{p,s}^d$ and $\tilde{N}_{p,s}^d$ denote respectively the full and reduced number of pressure and saturation variables in domain d. Retaining only the first $\tilde{N}_{p,s}$ columns of the corresponding basis $\mathbf{\Phi}_{p,s}^d$, reduced bases $\tilde{\mathbf{\Phi}}_p^d$ and $\tilde{\mathbf{\Phi}}_s^d$ are attained. Under these, the full state in domain d is approximated as:

$$\mathbf{x}^{d,n} = \begin{pmatrix}\mathbf{x}_p^{d,n}\\ \mathbf{x}_s^{d,n}\end{pmatrix} \approx \begin{pmatrix}\tilde{\mathbf{\Phi}}_p^d & \mathbf{O}\\ \mathbf{O} & \tilde{\mathbf{\Phi}}_s^d\end{pmatrix}\begin{pmatrix}\tilde{\boldsymbol{\psi}}_p^{d,n}\\ \tilde{\boldsymbol{\psi}}_s^{d,n}\end{pmatrix} = \tilde{\mathbf{\Phi}}^d\tilde{\boldsymbol{\psi}}^{d,n} \tag{7.9}$$

With these bases, optimization can be performed on the lower dimensional component states $\tilde{\boldsymbol{\psi}}^{d,n}$ in each domain, assumed to satisfy a corresponding set of equations:

$$\begin{aligned}\tilde{\boldsymbol{\psi}}^{d,n+1} &= \boldsymbol{\mathcal{L}}^{d,n+1}(\tilde{\boldsymbol{\psi}}^{d,n}, \tilde{\boldsymbol{\psi}}^{sd,n+1}, \boldsymbol{\xi})\\ \mathbf{y}^{d,n+1} &= \boldsymbol{\hbar}^{d,n+1}(\tilde{\boldsymbol{\psi}}^{d,n+1}, \boldsymbol{\xi}) \qquad n = 0, \dots, N-1\end{aligned} \tag{7.10}$$

In this system, the last equation is only relevant in domains where an output is produced, which excludes any domain that does not contain a well. Furthermore, the actual functions $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\hbar}$ are never established. Instead, a linear approximation of this model is constructed using the RBF-TPWL methodology.

7.3 Inner loops

During each inner loop of the algorithm, a new gradient is computed at the current optimal control $\boldsymbol{\xi}_b$. Afterwards, a new optimum is established through a search algorithm, using the determined approximate gradient.

7.3.1 RBF interpolation

To perform the RBF interpolation, the same set of samples Γ is used as applied in the POD method. To this end, a number of input samples $\boldsymbol{\xi}^{(i)}$ and corresponding states $\mathbf{x}^{(i),d,n}$ and outputs $\mathbf{y}^{(i),d,n}$ are extracted from Γ, for each domain and at each time step. We will denote these samples as $i = 1, \dots, N_{\mathrm{RBF}}$, where in our runs, all samples available in Γ, being all samples employed in the POD method, were reused. For the state vectors, the algorithm further extracts the state variables $\mathbf{x}^{(i),sd,n}$ of the domains surrounding d at each time. Then, the components of these sample states under the corresponding basis are computed at each time as:

$$\tilde{\boldsymbol{\psi}}^{i,d,n} = ([\tilde{\mathbf{\Phi}}^d]^T\tilde{\mathbf{\Phi}}^d)^{-1}[\tilde{\mathbf{\Phi}}^d]^T\mathbf{x}^{(i),d,n} \qquad \tilde{\boldsymbol{\psi}}^{(i),sd,n} = ([\tilde{\mathbf{\Phi}}^{sd}]^T\tilde{\mathbf{\Phi}}^{sd})^{-1}[\tilde{\mathbf{\Phi}}^{sd}]^T\mathbf{x}^{(i),sd,n} \tag{7.11}$$

Here, matrix $\tilde{\mathbf{\Phi}}^{sd}$ combines the bases $\tilde{\mathbf{\Phi}}^{d'}$ for all domains d' neighboring and including d. Next, the arguments of functions $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\hbar}$ in system (7.10) are collected into scaled argument vectors:

$$\begin{aligned}[\boldsymbol{\chi}^{(i),d,n+1}]^T &:= \begin{pmatrix}\frac{1}{c_\psi}[\boldsymbol{\psi}^{(i),d,n}]^T & \frac{1}{c_\psi}[\boldsymbol{\psi}^{(i),sd,n+1}]^T & \frac{1}{c_\xi}[\boldsymbol{\xi}^{(i)}]^T\end{pmatrix}\\ [\boldsymbol{\gamma}^{(i),d,n+1}]^T &:= \begin{pmatrix}\frac{1}{c_\psi}[\tilde{\boldsymbol{\psi}}^{(i),d,n+1}]^T & \frac{1}{c_\xi}[\boldsymbol{\xi}^{(i)}]^T\end{pmatrix} \qquad n = 0, \dots, N-1\end{aligned} \tag{7.12}$$

Here, the constants c_ψ and c_ξ are computed based on (5.22), as the maximal deviation of the global vectors $\boldsymbol{\psi}^{(i),n}$ and $\boldsymbol{\xi}^{(i)}$ from their sample mean, over the entire set Γ. Unlike the constants (5.22) though, no distinction is made here between pressure and saturation components, since these will exist at a similar scale.

Based on the arguments (7.12), radial distances between the samples in Γ are now computed, and transformed according to a multiquadratic basis function (5.8). Collecting these transformed distances in matrices $\boldsymbol{\Theta}^{r,d,n}, \boldsymbol{\Theta}^{\ell,d,n} \in \mathbb{R}^{N_{\text{RBF}} \times N_{\text{RBF}}}$ for the arguments $\boldsymbol{\chi}$ and $\boldsymbol{\gamma}$ respectively, weighting vectors at the samples are then computed as:

$$
\begin{aligned}
[\boldsymbol{\mathcal{W}}^{d,n+1}]^T &= [\boldsymbol{\Theta}^{r,d,n+1}]^{-1}[\boldsymbol{Z}^{d,n+1}]^T \\
[\boldsymbol{\mathcal{E}}^{d,n+1}]^T &= [\boldsymbol{\Theta}^{\ell,d,n+1}]^{-1}[\boldsymbol{Y}^{d,n+1}]^T \qquad n = 0, \dots, N-1
\end{aligned}
\tag{7.13}
$$

Here, the matrices $\boldsymbol{Z}^{d,n+1} \in \mathbb{R}^{\tilde{N}_x^d \times N_{\text{RBF}}}$ and $\boldsymbol{Y}^{d,n+1} \in \mathbb{R}^{\tilde{N}_y^d \times N_{\text{RBF}}}$ consist column-wise of the sample component states and outputs in the domain d. The resulting matrices $\boldsymbol{\mathcal{W}}^{d,n} \in \mathbb{R}^{\tilde{N}_x \times N_{\text{RBF}}}$ and $\boldsymbol{\mathcal{E}}^{d,n} \in \mathbb{R}^{N_y \times N_{\text{RBF}}}$ provide the RBF weighting coefficients for the reduced states and outputs in domain d, at each of the samples in Γ. These vectors depend on the choice of the shape parameter $\epsilon > 0$ in the multiquadratic function, for which an optimum is established between $\epsilon_{\min} = 10^{-2}$ and $\epsilon_{\max} = 100$ based on a "leave-one-out" cross validation error.

7.3.2 Trajectory piecewise linearization

After computing the RBF coefficient vectors, a piecewise linear model is constructed around the current optimum $\boldsymbol{\xi}_b$. For this, derivative matrices are computed at $\boldsymbol{\xi}_b$ as:

$$
\begin{aligned}
\mathbf{E}^{d,n+1}_{\tilde{\boldsymbol{\psi}}_b} &:= \left.\frac{\partial \boldsymbol{\mathcal{L}}^{d,n+1}}{\partial \tilde{\boldsymbol{\psi}}^{d,n}}\right|_{\tilde{\boldsymbol{\psi}}^{d,n}=\tilde{\boldsymbol{\psi}}^{d,n}_b} &&= \sum_{j\in\Gamma} \boldsymbol{\omega}_j^{d,n+1} \cdot \left.\frac{\partial\theta(r_j^{n+1}(\boldsymbol{\chi}^{d,n+1}))}{\partial\tilde{\boldsymbol{\psi}}^{d,n}}\right|_{\tilde{\boldsymbol{\psi}}^{d,n}=\tilde{\boldsymbol{\psi}}^{d,n}_b} \\
\mathbf{E}^{sd,n+1}_{\tilde{\boldsymbol{\psi}}_b} &:= \left.\frac{\partial \boldsymbol{\mathcal{L}}^{d,n+1}}{\partial \tilde{\boldsymbol{\psi}}^{sd,n+1}}\right|_{\tilde{\boldsymbol{\psi}}^{sd,n+1}=\tilde{\boldsymbol{\psi}}^{sd,n+1}_b} &&= \sum_{j\in\Gamma} \boldsymbol{\omega}_j^{d,n+1} \cdot \left.\frac{\partial\theta(r_j^{n+1}(\boldsymbol{\chi}^{d,n+1}))}{\partial\tilde{\boldsymbol{\psi}}^{sd,n+1}}\right|_{\tilde{\boldsymbol{\psi}}^{sd,n+1}=\tilde{\boldsymbol{\psi}}^{sd,n+1}_b} \\
\mathbf{G}^{d,n+1}_{\boldsymbol{\xi}_b} &:= \left.\frac{\partial \boldsymbol{\mathcal{L}}^{d,n+1}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\boldsymbol{\xi}_b} &&= \sum_{j\in\Gamma} \boldsymbol{\omega}_j^{d,n+1} \cdot \left.\frac{\partial\theta(r_j^{n+1}(\boldsymbol{\chi}^{d,n+1}))}{\partial\boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\boldsymbol{\xi}_b} \\
\mathbf{A}^{d,n+1}_{\tilde{\boldsymbol{\psi}}_b} &:= \left.\frac{\partial \boldsymbol{\hbar}^{d,n+1}}{\partial \tilde{\boldsymbol{\psi}}^{d,n+1}}\right|_{\tilde{\boldsymbol{\psi}}^{d,n+1}=\tilde{\boldsymbol{\psi}}^{d,n+1}_b} &&= \sum_{j\in\Gamma} \boldsymbol{\varepsilon}_j^{d,n+1} \cdot \left.\frac{\partial\theta(\ell_j^{n+1}(\boldsymbol{\gamma}^{d,n+1}))}{\partial\tilde{\boldsymbol{\psi}}^{d,n+1}}\right|_{\tilde{\boldsymbol{\psi}}^{d,n+1}=\tilde{\boldsymbol{\psi}}^{d,n+1}_b} \\
\mathbf{B}^{d,n+1}_{\boldsymbol{\xi}_b} &:= \left.\frac{\partial \boldsymbol{\hbar}^{d,n+1}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\boldsymbol{\xi}_b} &&= \sum_{j\in\Gamma} \boldsymbol{\varepsilon}_j^{d,n+1} \cdot \left.\frac{\partial\theta(\ell_j^{n+1}(\boldsymbol{\gamma}^{d,n+1}))}{\partial\boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\boldsymbol{\xi}_b}
\end{aligned}
\tag{7.14}
$$

With these derivatives, a linear reduced-order model is constructed:

$$
\begin{aligned}
\tilde{\boldsymbol{\psi}}^{d,n+1} &\approx \tilde{\boldsymbol{\psi}}^{d,n+1}_b + \mathbf{E}^{d,n+1}_{\tilde{\boldsymbol{\psi}}_b}(\tilde{\boldsymbol{\psi}}^{d,n} - \tilde{\boldsymbol{\psi}}^{d,n}_b) + \mathbf{E}^{sd,n+1}_{\tilde{\boldsymbol{\psi}}_b}(\tilde{\boldsymbol{\psi}}^{sd,n+1} - \tilde{\boldsymbol{\psi}}^{sd,n+1}_b) + \mathbf{G}^{d,n+1}_{\boldsymbol{\xi}_b}(\boldsymbol{\xi} - \boldsymbol{\xi}_b) \\
\mathbf{y}^{d,n+1} &\approx \mathbf{y}^{d,n+1}_b + \mathbf{A}^{d,n+1}_{\tilde{\boldsymbol{\psi}}_b}(\tilde{\boldsymbol{\psi}}^{d,n+1} - \tilde{\boldsymbol{\psi}}^{d,n+1}_b) + \mathbf{B}^{d,n+1}_{\boldsymbol{\xi}_b}(\boldsymbol{\xi} - \boldsymbol{\xi}_b)
\end{aligned}
\tag{7.15}
$$

For ease of notation, we let a subscript r denote the difference between a new solution and the current one, such that $\tilde{\boldsymbol{\psi}}_r := \tilde{\boldsymbol{\psi}} - \tilde{\boldsymbol{\psi}}_b$, $\boldsymbol{\xi}_r = \boldsymbol{\xi} - \boldsymbol{\xi}_b$, and $\mathbf{y}_r := \mathbf{y} - \mathbf{y}_b$. With this notation, the linearized system can be given as:

$$\begin{aligned}
\tilde{\boldsymbol{\psi}}_r^{d,n+1} &\approx \mathbf{E}_{\tilde{\boldsymbol{\psi}}_b}^{d,n+1}\tilde{\boldsymbol{\psi}}_r^{d,n} + \mathbf{E}_{\tilde{\boldsymbol{\psi}}_b}^{sd,n+1}\tilde{\boldsymbol{\psi}}_r^{sd,n+1} + \mathbf{G}_{\boldsymbol{\xi}_b}^{d,n+1}\boldsymbol{\xi}_r \\
\mathbf{y}_r^{d,n+1} &\approx \mathbf{A}_{\tilde{\boldsymbol{\psi}}_b}^{d,n+1}\tilde{\boldsymbol{\psi}}_r^{d,n+1} + \mathbf{B}_{\boldsymbol{\xi}_b}^{d,n+1}\boldsymbol{\xi}_r \qquad n = 0,\dots,N-1 \qquad (7.16)
\end{aligned}$$

7.3.3 Global system

To compute a gradient of the cost function with respect to the controls $\boldsymbol{\xi}$, the separate systems (7.16) in each domain d are first combined into a global system. For this system, the reduced state vector is given by the full vector of components:

$$[\tilde{\boldsymbol{\psi}}^n]^T = \left([\tilde{\boldsymbol{\psi}}^{1,n}]^T, \dots, [\tilde{\boldsymbol{\psi}}^{N_D,n}]^T\right) \in \mathbb{R}^{\tilde{N}_{\mathrm{x}}} \qquad n = 1,\dots,N \qquad (7.17)$$

This vector provides the components of the state under a reduced basis:

$$\tilde{\boldsymbol{\Phi}} = \begin{pmatrix} \tilde{\boldsymbol{\Phi}}_p^1 & \mathbf{O} & \dots & \dots & \tilde{\boldsymbol{\Phi}}_p^{N_D} & \mathbf{O} \\ \mathbf{O} & \tilde{\boldsymbol{\Phi}}_s^1 & \dots & \dots & \mathbf{O} & \tilde{\boldsymbol{\Phi}}_s^{N_D} \end{pmatrix} \in \mathbb{R}^{N_{\mathrm{x}} \times \tilde{N}_{\mathrm{x}}} \qquad (7.18)$$

With the component vector $\tilde{\boldsymbol{\psi}}$ and basis matrix $\tilde{\boldsymbol{\Phi}}$ defined in this manner, the full state at any time step can then be approximated by a vector:

$$\tilde{\mathbf{x}}^n = \left([\tilde{\mathbf{x}}_p^{1,n}]^T \quad [\tilde{\mathbf{x}}_s^{1,n}]^T \quad \dots \quad [\tilde{\mathbf{x}}_p^{N_D,n}]^T \quad [\tilde{\mathbf{x}}_s^{N_D,n}]^T\right)^T = \tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\psi}}^n \qquad (7.19)$$

This vector will deviate from the actual state vector $\mathbf{x}^n$ not only in that the it is merely an approximation, but also in that the elements might be ordered differently as a result of the domain decomposition.

To construct the global model, the gradient matrices in each domain are combined. For most gradients, construction of corresponding global versions is fairly straightforward, as these depend only on a single domain:

$$\mathbf{E}_{\tilde{\boldsymbol{\psi}}_b}^{D,n} = \begin{pmatrix} \mathbf{E}_{\tilde{\boldsymbol{\psi}}_b}^{1,n} & \mathbf{O} & \dots & \mathbf{O} \\ \mathbf{O} & \mathbf{E}_{\tilde{\boldsymbol{\psi}}_b}^{2,n} & \dots & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \dots & \mathbf{E}_{\tilde{\boldsymbol{\psi}}_b}^{N_D,n} \end{pmatrix} \qquad \mathbf{G}_{\boldsymbol{\xi}_b}^{n} = \begin{pmatrix} \mathbf{G}_{\boldsymbol{\xi}_b}^{1,n} \\ \mathbf{G}_{\boldsymbol{\xi}_b}^{2,n} \\ \vdots \\ \mathbf{G}_{\boldsymbol{\xi}_b}^{N_D,n} \end{pmatrix}$$

$$\mathbf{A}_{\tilde{\boldsymbol{\psi}}_b}^{n} = \begin{pmatrix} \mathbf{A}_{\tilde{\boldsymbol{\psi}}_b}^{1,n} & \mathbf{O} & \dots & \mathbf{O} \\ \mathbf{O} & \mathbf{A}_{\tilde{\boldsymbol{\psi}}_b}^{2,n} & \dots & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \dots & \mathbf{A}_{\tilde{\boldsymbol{\psi}}_b}^{N_D,n} \end{pmatrix} \qquad \mathbf{B}_{\boldsymbol{\xi}_b}^{n} = \begin{pmatrix} \mathbf{B}_{\boldsymbol{\xi}_b}^{1,n} \\ \mathbf{B}_{\boldsymbol{\xi}_b}^{2,n} \\ \vdots \\ \mathbf{B}_{\boldsymbol{\xi}_b}^{N_D,n} \end{pmatrix}$$

For the matrices $\mathbf{E}_{\tilde{\boldsymbol{\psi}}_b}^{sd,n}$ however, certain columns will correspond to derivatives of components across different domains. To account for this, we define:

$$\left[\mathbf{E}_{\tilde{\boldsymbol{\psi}}_b}^{sd,n}\right]_j := \begin{cases} \left.\dfrac{\partial \mathcal{L}^{d,n}}{\partial \tilde{\boldsymbol{\psi}}^{j,n}}\right|_{\tilde{\boldsymbol{\psi}}^{j,n} = \tilde{\boldsymbol{\psi}}_b^{j,n}} & j \in sd \\ \mathbf{O} & j \notin sd \end{cases} \qquad (7.20)$$

Thus, $[\mathbf{E}^{sd,n}_{\tilde{\boldsymbol{\psi}}_b}]_j$ gives the derivative of the component state $\boldsymbol{\mathcal{L}}^{d,n}$ with respect to $\tilde{\boldsymbol{\psi}}^{j,n}_b$, given that the domain j neighbors the domain d, returning a zero matrix otherwise. With this notation, the global gradient can be given as:

$$\mathbf{E}^{SD,n}_{\tilde{\boldsymbol{\psi}}_b} = \begin{pmatrix} \left[\mathbf{E}^{s1,n}_{\tilde{\boldsymbol{\psi}}_b}\right]_1 & \left[\mathbf{E}^{s1,n}_{\tilde{\boldsymbol{\psi}}_b}\right]_2 & \cdots & \left[\mathbf{E}^{s1,n}_{\tilde{\boldsymbol{\psi}}_b}\right]_{N_D} \\ \left[\mathbf{E}^{s2,n}_{\tilde{\boldsymbol{\psi}}_b}\right]_1 & \left[\mathbf{E}^{s2,n}_{\tilde{\boldsymbol{\psi}}_b}\right]_2 & \cdots & \left[\mathbf{E}^{s2,n}_{\tilde{\boldsymbol{\psi}}_b}\right]_{N_D} \\ \vdots & \vdots & \ddots & \vdots \\ \left[\mathbf{E}^{sN_D,n}_{\tilde{\boldsymbol{\psi}}_b}\right]_1 & \left[\mathbf{E}^{sN_D,n}_{\tilde{\boldsymbol{\psi}}_b}\right]_2 & \cdots & \left[\mathbf{E}^{sN_D,n}_{\tilde{\boldsymbol{\psi}}_b}\right]_{N_D} \end{pmatrix} \tag{7.21}$$

With these global gradient matrices, the reduced-order model becomes:

$$\begin{aligned} \tilde{\boldsymbol{\psi}}^{n+1}_r &\approx \mathbf{E}^{D,n+1}_{\tilde{\boldsymbol{\psi}}_b}\tilde{\boldsymbol{\psi}}^n_r + \mathbf{E}^{SD,n+1}_{\tilde{\boldsymbol{\psi}}_b}\tilde{\boldsymbol{\psi}}^{n+1}_r + \mathbf{G}^{n+1}_{\boldsymbol{\xi}_b}\boldsymbol{\xi}_r \\ \tilde{\mathbf{y}}^{n+1}_r &\approx \mathbf{A}^{n+1}_{\tilde{\boldsymbol{\psi}}_b}\tilde{\boldsymbol{\psi}}^{n+1}_r + \mathbf{B}^{n+1}_{\boldsymbol{\xi}_b}\boldsymbol{\xi}_r \qquad n = 0,\ldots,N-1 \end{aligned} \tag{7.22}$$

We will refer to this system as the reduced-order model (ROM). Using this model, the value of the component vectors and thus the actual pressure and saturation states can be approximated, given as a linear extrapolation around the background input $\boldsymbol{\xi}_b$.

For implementation of the adjoint method, we also introduce functions $\tilde{\mathbf{g}}^n$ and $\tilde{\mathbf{h}}^n$:

$$\begin{aligned} \tilde{\mathbf{g}}^{n+1}(\tilde{\boldsymbol{\psi}}^n, \tilde{\boldsymbol{\psi}}^{n+1}, \boldsymbol{\xi}) &:= \left[\mathbf{I} - \mathbf{E}^{SD,n+1}_{\tilde{\boldsymbol{\psi}}_b}\right]\tilde{\boldsymbol{\psi}}^{n+1}_r - \mathbf{E}^{D,n+1}_{\tilde{\boldsymbol{\psi}}_b}\tilde{\boldsymbol{\psi}}^n_r - \mathbf{G}^{n+1}_{\boldsymbol{\xi}_b}\boldsymbol{\xi}_r \\ \tilde{\mathbf{h}}^{n+1}(\tilde{\boldsymbol{\psi}}^{n+1}, \boldsymbol{\xi}) &:= \tilde{\mathbf{y}}^{n+1}_b + \mathbf{A}^{n+1}_{\tilde{\boldsymbol{\psi}}_b}\tilde{\boldsymbol{\psi}}^{n+1}_r + \mathbf{B}^{n+1}_{\boldsymbol{\xi}_b}\boldsymbol{\xi}_r \qquad n = 0,\ldots,N-1 \end{aligned} \tag{7.23}$$

Then we can denote the reduced-order model as:

$$\begin{aligned} 0 &\approx \tilde{\mathbf{g}}^{n+1}(\tilde{\boldsymbol{\psi}}^n, \tilde{\boldsymbol{\psi}}^{n+1}, \boldsymbol{\xi}) \\ \tilde{\mathbf{y}}^{n+1} &\approx \tilde{\mathbf{h}}^{n+1}(\tilde{\boldsymbol{\psi}}^{n+1}, \boldsymbol{\xi}) \qquad n = 0,\ldots,N-1 \end{aligned} \tag{7.24}$$

7.3.4 Adjoint method

With the linear reduced-order model given by (7.24), a gradient can be computed for the cost function using the adjoint method in a straightforward manner. For this, we introduce a cost function $\tilde{J}(\boldsymbol{\xi})$, computing the net present value from the control vectors as:

$$\tilde{J}(\tilde{\boldsymbol{\psi}}^1, \ldots, \tilde{\boldsymbol{\psi}}^N, \boldsymbol{\xi}) := \sum_{n=1}^{N} [\tilde{\mathbf{c}}^n]^T \cdot \tilde{\mathbf{h}}^n(\tilde{\boldsymbol{\psi}}^n, \boldsymbol{\xi}) \tag{7.25}$$

Here, the cost vector $\tilde{\mathbf{c}}$ contains the same elements as the original cost vector (2.32), only rearranged to match the order of the outputs after the domain decomposition. Clearly, the partial derivatives of this function with respect to its arguments are given by:

$$\begin{aligned} \frac{\partial \tilde{J}}{\partial \tilde{\boldsymbol{\psi}}^n} &= \sum_{n=1}^{N} [\tilde{\mathbf{c}}^n]^T \cdot \left[\frac{\partial \tilde{\mathbf{h}}^n}{\partial \tilde{\boldsymbol{\psi}}^n}\right] = [\tilde{\mathbf{c}}^n]^T \cdot \mathbf{A}^n_{\tilde{\boldsymbol{\psi}}_b} \\ \frac{\partial \tilde{J}}{\partial \boldsymbol{\xi}} &= \sum_{n=1}^{N} [\tilde{\mathbf{c}}^n]^T \cdot \left[\frac{\partial \tilde{\mathbf{h}}^n}{\partial \boldsymbol{\xi}}\right] = \sum_{n=1}^{N} [\tilde{\mathbf{c}}^n]^T \cdot \mathbf{B}^n_{\boldsymbol{\xi}_b} \end{aligned} \tag{7.26}$$

Similarly, considering the model in the form (7.24), derivatives of the evolution equation are seen to be:

$$\frac{\partial \mathbf{g}^{n+1}}{\partial \tilde{\boldsymbol{\psi}}^{n+1}} = \left[\mathbf{I} - \mathbf{E}^{SD,n+1}_{\tilde{\boldsymbol{\psi}}_b}\right] \qquad \frac{\partial \mathbf{g}^{n+1}}{\partial \tilde{\boldsymbol{\psi}}^{n}} = -\mathbf{E}^{D,n+1}_{\tilde{\boldsymbol{\psi}}_b}$$
$$\frac{\partial \mathbf{g}^{n+1}}{\partial \boldsymbol{\xi}} = -\mathbf{G}^{n+1}_{\boldsymbol{\xi}_b} \tag{7.27}$$

Substituting these derivatives into the system (3.17), we attain a reduced surrogate adjoint model as:

$$\left[\mathbf{I} - \mathbf{E}^{SD,n}_{\tilde{\boldsymbol{\psi}}_b}\right]^T \tilde{\boldsymbol{\lambda}}^n = \left[\mathbf{E}^{D,n+1}_{\tilde{\boldsymbol{\psi}}_b}\right]^T \cdot \tilde{\boldsymbol{\lambda}}^{n+1} - \left[\mathbf{A}^n_{\tilde{\boldsymbol{\psi}}_b}\right]^T \cdot \tilde{\mathbf{c}}^n \qquad n = N-1, \dots, 1$$
$$\left[\mathbf{I} - \mathbf{E}^{SD,N}_{\tilde{\boldsymbol{\psi}}_b}\right]^T \tilde{\boldsymbol{\lambda}}^N = -\left[\mathbf{A}^N_{\tilde{\boldsymbol{\psi}}_b}\right]^T \cdot \tilde{\mathbf{c}}^N \tag{7.28}$$

Running this model backwards, the algorithm constructs the adjoint vectors $\tilde{\boldsymbol{\lambda}}^n \in \mathbb{R}^{\mathrm{N}_x}$. From these, the cost function gradient is then computed according to (3.19), which yields for the reduced system:

$$[\boldsymbol{\nabla} J]^T = \frac{d\tilde{J}}{d\boldsymbol{\xi}} = \sum_{n=1}^{N} \left([\tilde{\mathbf{c}}^n]^T \cdot \mathbf{B}^n_{\boldsymbol{\xi}_b} - [\tilde{\boldsymbol{\lambda}}^n]^T \cdot \mathbf{G}^n_{\boldsymbol{\xi}_b} \right) \tag{7.29}$$

This provides a search direction for optimizing the control vector at the current control $\boldsymbol{\xi}_b$

7.3.5 Search iterations

After establishing a gradient $\boldsymbol{\nabla} J$ of the cost function, a new optimal control is sought in the direction of this gradient. This is done using the line search algorithm 1 described in chapter 3, trying several controls $\boldsymbol{\xi}^i$ along the search direction $\mathbf{p}_b$ until a Wolfe condition is satisfied. The search direction here is simply computed as the normalized gradient at the control:

$$\mathbf{p}_b = \left.\frac{\boldsymbol{\nabla}\tilde{J}}{\|\boldsymbol{\nabla}\tilde{J}\|_\infty}\right|_{\boldsymbol{\xi}=\boldsymbol{\xi}_b} \tag{7.30}$$

In case an element of control $\boldsymbol{\xi}_b$ lies on the boundary of the domain, and the corresponding element of the gradient points in the direction of this boundary, this element will be set to 0.

In algorithm 1, the step size α^i along the search direction is decreased, until the increase in the value of J exceeds a certain fraction η_W of the expected increased, as computed according to some model function $m(\boldsymbol{\xi})$. In our implementation, a value of $\eta_W = 10^{-4}$ was consistently used, requiring only a fairly small improvement with respect to the expected increase. Since the computed gradient is exactly that of the adjusted cost function $\tilde{J}(\boldsymbol{\xi})$, on the basis of the reduced system (7.22), the algorithm was also set to use this function as model cost $m(\boldsymbol{\xi})$ in the line search, with the exact value of the objective function calculated according to the full-order model (7.2). The initial step size $\alpha_{\max}$ in the algorithm is based on the size of the domain in which the model is assumed to be accurate, which we have taken to be $\alpha_{\max} = 1\frac{1}{2}\epsilon_{\mathrm{POD}}$ in our simulations. Finally, bounds $\eta_\xi = 10^{-3}$ and $\eta_J = 10^{-2}$ were imposed on the required change in the input and cost function values respectively. Providing these parameters, algorithm 1 will then search for a new control $\boldsymbol{\xi}^*$ and corresponding cost $J(\boldsymbol{\xi}^*)$, varying to a desired degree from the current values.

7.4 Reiteration

7.4.1 Inner loops

After establishing a new control $\boldsymbol{\xi}^*$, yielding a higher value of the cost function than the current control $\boldsymbol{\xi}_b$, this current control is replaced by the new optimal guess. Next, this control is added to the set Γ of samples, whilst another sample is removed from this set. Specifically, the sample $\boldsymbol{\xi}^{(j)} \in \Gamma$ which lies furthest from the new optimum $\boldsymbol{\xi}_b$, thus maximizing the distance $\|\boldsymbol{\xi}^{(j)} - \boldsymbol{\xi}_b\|_2$, is removed. In this manner, the size of set Γ does not increase during the inner loops, to avoid increasing the computational cost. Based on the new set Γ, a new inner loop is then initiated, reperforming the RBF interpolation and gradient computation around the new sample $\boldsymbol{\xi}_b$.

The inner loops as described above are repeated until a stopping criterion is satisfied. Here we have implemented three stopping criteria, the first of which imposes a maximum value N_{inner} on the number of inner iterations to be performed. This value was set to be $\lfloor \frac{3}{4} N_{\text{POD}} \rfloor$, assuring that no more than 75% of the POD samples computed during the outer loop were replaced, at which point the RBF interpolation was no longer expected to be reliable. In practice, however, this maximum number of iterations was never reached, as one of the alternative stopping criteria would already be satisfied. These criteria required the cost function and input to display a significant change between inner iterations, repeating an inner loop only if deemed effective. They were implemented to be the same conditions as for the search algorithm, requiring:

$$\begin{aligned} \frac{\|\boldsymbol{\xi}^k - \boldsymbol{\xi}^{k-1}\|}{\max\{1, \|\boldsymbol{\xi}^k\|\}} &\geq \eta_\xi \\ \frac{\|J^k - J^{k-1}\|}{\max\{1, \|J^k\|\}} &\geq \eta_J \end{aligned} \tag{7.31}$$

In these inequalities, the superscripts k denote the results of consecutive inner iterations, providing new estimates of an optimal input $\boldsymbol{\xi}^k$, and a corresponding cost function value $J^k := J(\boldsymbol{\xi}^k)$. In our implementation, values $\eta_\xi = 10^{-3}$ and $\eta_J = 10^{-2}$ have been used. Based on these values, the inner loops would be stopped whenever conditions (7.31) were no longer satisfied, or a maximum number of inner loops had been performed.

7.4.2 Outer loops

When the inner loops are terminated, a new optimal control $\boldsymbol{\xi}_b$ and sample set Γ have been generated, for which reperforming the RBF is not assumed to provide a useful gradient. Instead, at this stage, a new search direction is computed based on a new sample set, removing certain samples from Γ and adding N_{POD} new ones. To establish how many samples to retain, the distance of the new optimum from the old one is considered.

Let $\boldsymbol{\xi}^k$ denote the optimal sample at the start of outer loop k. As starting guess $\boldsymbol{\xi}^{k+1}$ for the new outer loop, the sample $\boldsymbol{\xi}_b \in \Gamma$ is taken yielding the greatest net present value, potentially being one of the POD samples in Γ_{POD}, though usually being the result of the inner iterations. Next, the distance $\|\boldsymbol{\xi}^{k+1} - \boldsymbol{\xi}^k\|_2$ between the current and old optimum is computed. This value is then compared to the radius of the space spanned by the POD samples, given by:

$$r^k_{\text{POD}} = \max_{\boldsymbol{\xi}^{(j)} \in \Gamma_{\text{POD}}} \{\|\boldsymbol{\xi}^{(j)} - \boldsymbol{\xi}^k\|_2\} \tag{7.32}$$

If the sample $\boldsymbol{\xi}^{k+1}$ falls outside of the space spanned by the POD samples, such that $r^k_{\text{POD}} \leq \|\boldsymbol{\xi}^{k+1} - \boldsymbol{\xi}^k\|_2$, the samples collected in Γ are assumed not to provide useful information for the next outer loop, and are thus discarded. On the other hand, if the new sample $\boldsymbol{\xi}^{k+1}$ is within the POD domain, a certain number of samples is retained, computed as:

$$N_{\text{retain}} = \frac{1}{2}\left\lfloor \left(1 - \frac{\|\boldsymbol{\xi}^{k+1} - \boldsymbol{\xi}^k\|_2}{r^k_{\text{POD}}}\right) \right\rfloor N_\Gamma \tag{7.33}$$

Using this function, the number of retained samples decreases linearly with the distance from the starting guess $\boldsymbol{\xi}^k$, retaining at most half of the samples N_Γ in set Γ, when the new input $\boldsymbol{\xi}^{k+1}$ is close to the old one. This function rests on the assumption that the samples in Γ are more relevant for inputs closer to the original $\boldsymbol{\xi}^k$, aiming to retain samples only if deemed useful. Here, no samples are retained when moving outside of the computational domain, as the required computational effort increases with the number of retained samples. The factor of $\frac{1}{2}$ has also been chosen to further abate this increase in computational cost, although any value between 0 and 1 can be implemented.

In addition to prescribing the number of retained samples, the distance $\|\boldsymbol{\xi}^{k+1} - \boldsymbol{\xi}^k\|_2$ also determines how the perturbation size ϵ_{POD} for the new samples is adjusted. Specifically, if this distance is less than r_{POD}, the optimum is expected to be sought at a smaller scale, and thus the perturbation size is multiplied with a factor $\frac{1}{2}$. This is only done if no input reparameterization is used. Otherwise the number of degrees of freedom is doubled, and the same initial perturbation size is taken, restarting the optimization in a higher-dimensional setting. On the other hand, if the distance $\|\boldsymbol{\xi}^{k+1} - \boldsymbol{\xi}^k\|_2$ exceeds r_{POD}, the perturbation size is simply kept constant, independent of whether reparameterization is used.

Adjusting the parameters as described, increasingly more accurate gradients are aimed to be computed if the control vector does not change sufficiently, using smaller perturbation sizes and retaining more samples. If this does not yield improved results, however, the algorithm should be terminated. To enforce this, we required the perturbation size to remain above a certain threshold, satisfying:

$$\epsilon_{\text{POD}} \geq \mu_{\text{POD}} \tag{7.34}$$

Here we used $\mu_{\text{POD}} = 0.005$, noting that the controls are generally scaled to assume values within the domain $[-0.5, 0.5]$. If this condition was not satisfied, the algorithm would terminate, as new outer loops were not expected to improve the results. In addition, a maximal number of N_{outer} outer loops was imposed, also terminating the algorithm if the number of outer loops became too big.

The full methodology as implemented for this thesis is summarized in algorithm 4 below. A schematic overview for a situation applying a fixed number of degrees of freedom is further provided in figure 7.1.

Algorithm 4: Subdomain POD-TPWL adjoint optimization algorithm

```
Input: Starting input guess $\mathbf{u}^{\text{init}}$, Number of perturbations $N_{\text{POD}}$, Size of perturbations
       $\epsilon_{\text{POD}}$, Domain decomposition $\Omega = \bigcup_{d=1}^{16} \Omega^d$, Number of outer iterations $N_{\text{outer}}$
       Optional: Set of precomputed samples $\Gamma$
Result: Estimate of optimal input $\mathbf{u}^{\text{opt}}$
1  Run the full-order model for input $\mathbf{u}^{\text{init}}$ and collect results in $\Gamma$;
2  Reduce the input $\mathbf{u}^{\text{init}}$ according to the reparameterization to $\boldsymbol{\xi}^{\text{init}}$;
3  Set $k = 1$ and $\boldsymbol{\xi}^0 = \boldsymbol{\xi}^{\text{init}}$;
4  while $\left[k \leq N_{\text{outer}}\right] \wedge \epsilon_{\text{POD}} > 0.005$ do
5      Construct $N_{\text{POD}}$ pseudo-random inputs in the hypercube of size $\epsilon_{\text{POD}}$ around $\boldsymbol{\xi}^{k-1}$;
6      Run full-order model for all samples and collect results in $\Gamma_{\text{POD}} \subseteq \Gamma$;
7      Construct POD basis $\tilde{\boldsymbol{\Phi}}^d$ based on the samples in $\Gamma$ for each subdomain $d = 1, \ldots, 16$;
8      Compute component states $\tilde{\boldsymbol{\psi}}^{(i),n}$ for all samples in $\Gamma$ under bases $\tilde{\boldsymbol{\Phi}}^d$;
9      Divide component states into $(\tilde{\boldsymbol{\psi}}^{(i),d,n}, \tilde{\boldsymbol{\psi}}^{(i),sd,n}) \in \Gamma$ in each subdomain $d = 1, \ldots, 16$;
10     Set $j = 1$ and $\boldsymbol{\xi}^{k,0} = \boldsymbol{\xi}^{k-1}$;
11     while $\left[j \leq \lfloor \frac{3}{4} N_{\text{POD}} \rfloor\right] \wedge \left[\frac{\|\boldsymbol{\xi}^{k,j} - \boldsymbol{\xi}^{k,j-1}\|}{\max\{1, \|\boldsymbol{\xi}^{k,j}\|\}} \geq 10^{-3}\right] \wedge \left[\frac{\|J^{k,j} - J^{k,j-1}\|}{\max\{1, \|J^{k,j}\|\}} \geq 10^{-2}\right]$ do
12         Compute RBF weighting vectors for reduced state and output functions based on
             samples in $\Gamma$;
13         Construct derivatives of reduced state and output functions using the weighting
             vectors;
14         Combine derivatives in each domain into global derivatives;
15         Construct reduced-order model based on global derivatives;
16         Perform adjoint method to reduced-order model to construct a gradient;
17         Determine a new optimum $\boldsymbol{\xi}^{k,j}$ in direction of gradient using algorithm 1;
18         Discard furthest sample $i$ from $\Gamma$ maximizing $\|\boldsymbol{\xi}^{(i)} - \boldsymbol{\xi}^{k,j}\|_2$;
19         Add $\boldsymbol{\xi}^{k,j}$ and corresponding reduced state and output information to $\Gamma$;
20         Set $j = j + 1$
21     end
22     Establish $\boldsymbol{\xi}^k = \arg\max_{\boldsymbol{\xi}^{(i)} \in \Gamma} \{J(\mathcal{K}_{\text{bnd}}(\boldsymbol{\xi}^{(i)}))\}$;
23     Determine size $r_{\text{POD}}^k = \max_{\boldsymbol{\xi}^{(i)} \in \Gamma_{\text{POD}}} \{\|\boldsymbol{\xi}^{(i)} - \boldsymbol{\xi}^{k-1}\|_2\}$ of POD domain;
24     Determine $N_{\text{retain}} = \min\left\{\frac{1}{2}\left\lfloor \left(1 - \frac{\|\boldsymbol{\xi}^k - \boldsymbol{\xi}^{k-1}\|_2}{r_{\text{POD}}^k}\right) \right\rfloor N_\Gamma, \frac{1}{2} N_\Gamma\right\}$;
25     Retain only $N_{\text{retain}}$ samples closest to $\boldsymbol{\xi}^k$ in $\Gamma$;
26     if $N_c < N_{\text{u}}$ then
27         Set $N_c \to \min\{2N_c, N_{\text{u}}\}$;
28         Adjust the reparameterization $\mathcal{K}$;
29         Adjust the sample controls $\boldsymbol{\xi}^{(i)} \in \Gamma$ according to the new reparameterization;
30     else if $\|\boldsymbol{\xi}^{(i)} - \boldsymbol{\xi}^{k-1}\|_2 < r_{\text{POD}}$ then
31         Reduce perturbation size $\epsilon_{\text{POD}} \to 0.5 \cdot \epsilon_{\text{POD}}$;
32     end
33     Set $k = k + 1$;
34 end
35 Compute optimal input $\mathbf{u}^{\text{opt}} = \mathcal{K}_{\text{bnd}}(\boldsymbol{\xi}^{k-1})$;
```

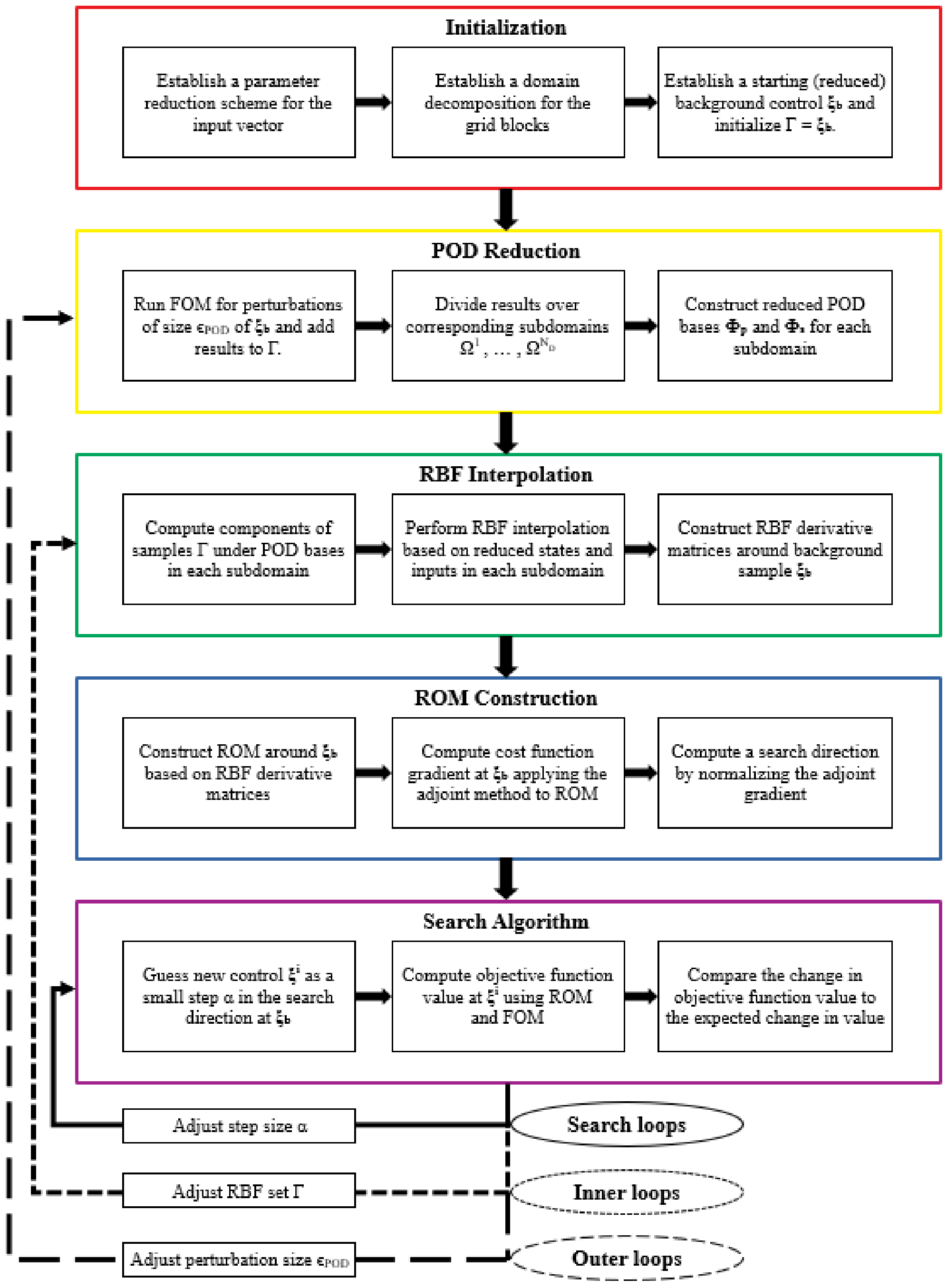

Initialization
Establish a parameter reduction scheme for the input vector
Establish a domain decomposition for the grid blocks
Establish a starting (reduced) background control ξb and initialize Γ = ξb.
POD Reduction
Run FOM for perturbations of size εPOD of ξb and add results to Γ.
Divide results over corresponding subdomains Ω1, … , ΩND
Construct reduced POD bases Φp and Φs for each subdomain
RBF Interpolation
Compute components of samples Γ under POD bases in each subdomain
Perform RBF interpolation based on reduced states and inputs in each subdomain
Construct RBF derivative matrices around background sample ξb
ROM Construction
Construct ROM around ξb based on RBF derivative matrices
Compute cost function gradient at ξb applying the adjoint method to ROM
Compute a search direction by normalizing the adjoint gradient
Search Algorithm
Guess new control ξi as a small step α in the search direction at ξb
Compute objective function value at ξi using ROM and FOM
Compare the change in objective function value to the expected change in value
Adjust step size α
Search loops
Adjust RBF set Γ
Inner loops
Adjust perturbation size εPOD
Outer loops

Figure 7.1: Schematic overview of the subdomain POD-TPWL algorithm using a fixed number of control parameters

Chapter 8

Numerical Testing of the Reduced Adjoint Algorithm

In this chapter, the performance of the POD-TPWL algorithm is tested, applying the different reparameterization schemes from chapter 6. Results are also discussed upon using the full input vector, for both of the reservoir models introduced in chapter 2. Finally, the performance is considered when increasing the number of input parameters during the optimization.

8.1 Experimental setup

Implementing the POD-TPWL algorithm as described in the previous chapter, an optimal input $\mathbf{u}$ can be established for an arbitrary cost function $J(\mathbf{u})$, on the basis of any model. The final execution of this algorithm depends on the parameters used at each stage, for example determining how many samples are used and retained during each iteration. For this thesis, the values of these parameters have been chosen on the basis of observations of the various techniques the algorithm relies on, described in earlier chapters. In this manner, although adjustments are easily implemented, the algorithm has been specifically tailored for the problem of production optimization. In testing its performance, the algorithm has also been specifically applied to this problem, considering the Kanaal and Egg reservoir models as described before.

In running the reduced adjoint algorithm, different input reparameterization schemes could be applied. The most straightforward of these involves no reduction of the size of the input vector, optimizing the full set of input parameters as allowed by the model. Since the POD-TPWL method does not require the size of the vectors to be reduced, and only a limited number of reparameterization schemes were considered for this thesis, the situation without reduction is also considered as the basis implementation. Running the algorithm under these conditions, the performance of the general method was tested. In addition, experiments were conducted running the algorithm upon reparameterizing the inputs according to each of the methods from chapter 6. In these experiments, the reduced number of controls was always kept low, aiming to maximize the benefit of implementing a reparameterization rather than optimizing the full input. Finally, the same experiments were run increasing the number of degrees of freedom between outer loops, as described in the previous chapter. For this, only the cubic spline interpolation and step function control methods were used.

8.2 Full control results

To test the general performance of the algorithm, it was run several times. In each case, only a limited number of $N_{\text{POD}} = 25$ samples was utilized to recompute search directions between outer loops, to maintain a short computational time. The full input vector was used, not reducing the number of input parameters, but transforming them through a Sigmoid function to satisfy the imposed bounds. Otherwise, the algorithm was implemented as described in the previous chapter, running a maximum of $N_{\text{outer}} = 10$ outer iterations, and $N_{\text{search}} = 50$ iterations in the search algorithm. In this manner, better inputs were established iteratively, as expected, increasing the net present value. An example of this increase is plotted in figure 8.1, for both the Kanaal and the Egg model. In this figure, each filled marker corresponds to the start of an outer loop, at which point N_{POD} new samples were computed and the POD and RBF steps were reperformed. Each empty marker corresponds to a new inner loop, at which point an RBF sample was replaced, and a new gradient was computed. In between, the net present value changes as new inputs are tested through the search algorithm.

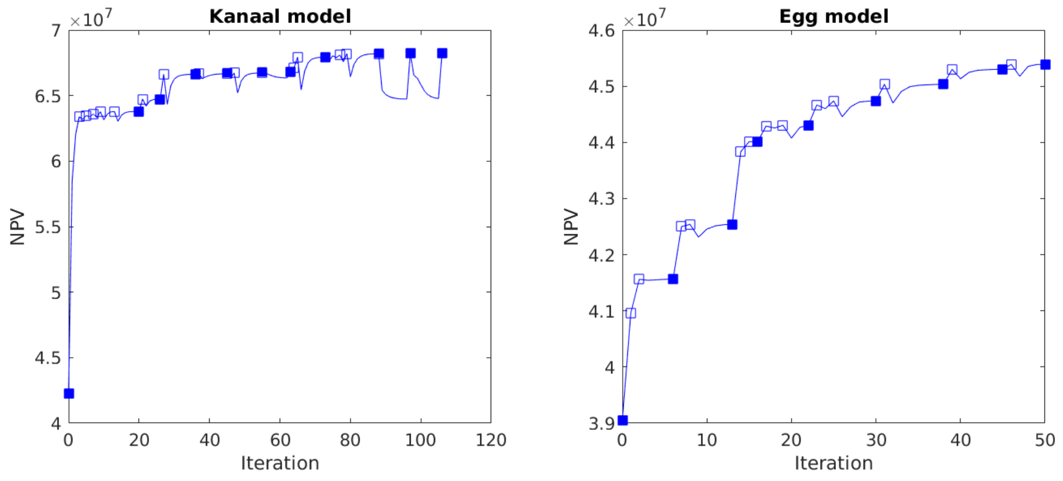


Figure 8.1: Example of increase of objective function for Kanaal and Egg model

Overall, the algorithm was found to achieve net present values of around $J = 6.85 \cdot 10^7$ for the Kanaal model, and $J = 4.55 \cdot 10^7$ for the Egg model. The large discrepancy in these value is a result of the different monetary parameters in two models, allowing a larger profit to be made for the Kanaal model when injecting and producing the same amounts of each phase. Aside from these different results attained at the end of the algorithm, the intermediate results building up to this optimum also differed between the models. In particular, the initial increase in net present value for the Kanaal model was consistently significantly greater than that of the Egg model, as also visible in figure 8.1. This was despite the fact that, at the end of the first outer loop, different runs of the algorithm for the same model produced varying estimates of the input. In consecutive outer loops, the increase in net present value was much less extreme for the Kanaal model, whereas more gradual improvements were observed for the Egg model.

An explanation for the disparity between the increase in objective function observed for the two models, could be the difference in the shape of this function over the domain of possible inputs. Likely, the cost function favors more extreme inputs for the Kanaal model, whereas more moderate input values are optimal for the Egg model. As a consequence, the initial step in the algorithm, moving away from the central inputs towards the boundaries of the domain, produces a greater improvement for the Kanaal model than for the Egg model. On the other hand, once this extreme input has been established, determining new inputs offers little improvement for the Kanaal model, whereas a lot of gain is still possible for the Egg model. As a result, the Kanaal model also tended to perform many more search iterations than the Egg model, as improvements for this model rapidly became more difficult to achieve.

Although the increase in net present value assumed different curves for the two models, the overall performance of the algorithm displayed very similar patterns. Specifically, the effect of the inner loops was found to be limited for both the Kanaal and Egg model, offering only minor improvements and in scarce cases. This suggests that, although a fairly accurate gradient can be computed at an arbitrary input by adding it to the RBF sample set, as seen in chapter 5, this new search direction does not yield extensive improvements in the considered implementation. Instead, the outer loops are necessary to provide a more significant increase in the objective function value, relying on information attained from new samples generated around the input. However, even between outer loops such improvements are not guaranteed, as especially for the Kanaal model, new outer loops were often performed without increasing the cost function value. Improvements could then be attained only once sufficient samples had been accumulated between such loops, and the perturbation size had sufficiently decreased. This too, however, would often take multiple outer loops, all the while expending a lot of effort without success.

8.3 Reparameterization experiments

8.3.1 Single step control

Implementing the single step control, varying results were attained. Specifically for the Kanaal model, a step function type control works very well, and the results of several tests are displayed in figure 8.2. In the first graph, the increase in net present value is plotted over several outer loops, computed as the mean of multiple runs starting with the same initial guess. This graph also provides the maximal and minimal values attained over these different runs, displaying a large variation from the mean. The value of $7.4 \cdot 10^7$ attained here is exceptionally high, and corresponds to the input displayed in the second subfigure. At wells 5 and 7, this input displays extreme values, whilst at the other wells milder input values were attained. Such an input is difficult to attain using alternative reparameterization schemes, as such schemes cannot mimic a step function of this form to a sufficient degree. However, the single step method itself is not able to consistently produce such desirable results either, often assuming more extreme values, or failing to escape local optima. Moreover applying this method to the Egg model, very poor results were attained, where often the method failed to improve the value of the objective function beyond that of the initial guess. This suggests that, a single step control method as applied here, should be considered only on the basis of prior knowledge or a desired input shape, and even then several runs are necessary to assure optimal results.

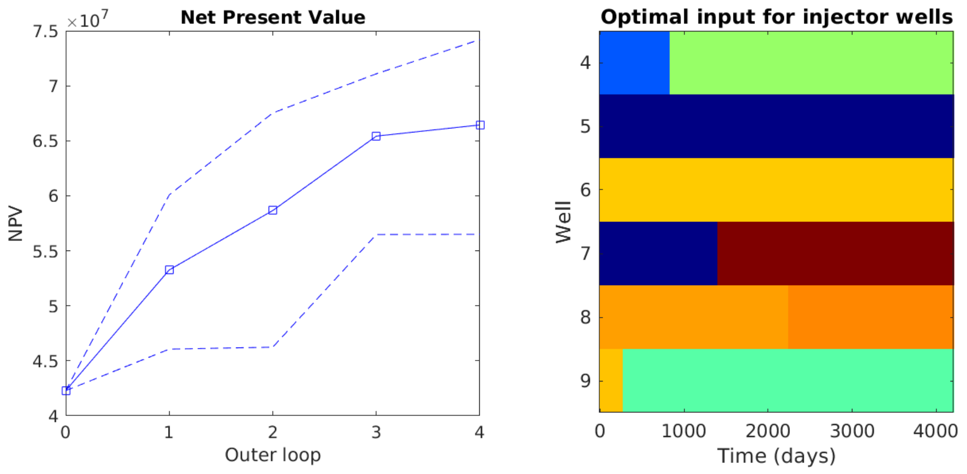


Figure 8.2: Increase of objective function and example of optimal inputs at injection wells attained using single step control

8.3.2 Principal component analysis

Using a PCA type reparameterization, poor results were attained. Some of these are displayed in figure 8.3, showing a minimal, mean, and maximal increase in net present value over several runs of the Kanaal model. In these simulations, values $\alpha_t = 0.5$, $\alpha_x = 1$, and $\sigma = 1$ were used, imposing a stronger temporal than spatial correlation. The required energy retained by the basis vectors was taken to be 95%, retaining a reduced basis of 21 vectors. The full control could then described as a linear combination of these vectors, with the scaling coefficients describing the control. This way, the full input would be described as a linear combination of smooth basis functions, of which a few at the first well are plotted in figure 8.3.

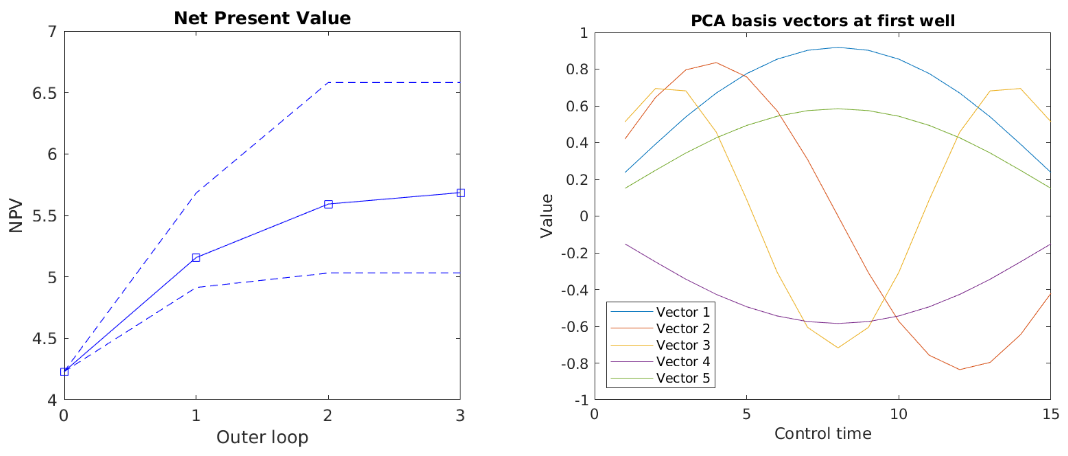


Figure 8.3: Increase of objective function and example basis functions using PCA control

The poor results attained using this method are due to the difficulties in combining the basis vectors to a full input. In particular, since each of the resulting basis vectors could assume values between -1 and 1, imposing bounds on the controls such that the resulting inputs would not violate the physical limitations was challenging. Indeed, any such restriction would exclude a large variety of potential input functions. Therefore, the controls were allowed to assume any real value, after which the computed input function would be transformed using a Sigmoid function. The resulting inputs, however, were not able to achieve net present values comparable to those using the full input vector, offering low values of the objective function.

8.3.3 Cubic spline interpolation

Of the different implemented reduction schemes, the cubic spline interpolation method provides the best results. Using an interpolation based on just a few times at each well, both for the Kanaal and Egg model the optimization procedure produces consistently reasonable results. Herein, the CSI method provides a smooth input that can assume a wide variety of shapes, allowing more irregular inputs upon using more control times. For the Kanaal model, two control times per well already produces good results, as the resulting inputs approximate step functions, which work well for this model. Results for such a situation are displayed in figure 8.4. The Egg model, however, requires more irregular inputs, and thus more input parameters are necessary to attain better results. Nevertheless, good results can still be attained at a reduced computational cost for both models, making this method more appealing than either the PCA or single step method.

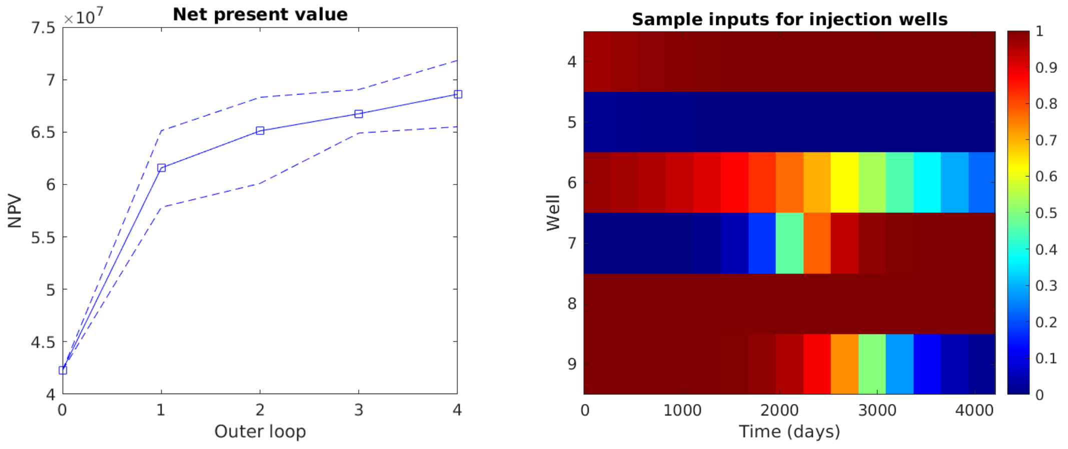


Figure 8.4: Increase of objective function and example of optimal inputs at injection wells attained using cubic spline interpolation

8.4 Dimensionality increase

8.4.1 CSI reparameterization

Using the CSI reparameterization, the algorithm was found to produce good results, based on a reduced control. In reducing the input, however, degrees of freedom in the optimization procedure are lost, thus also losing potentially better solutions. Increasing the dimensionality during the optimization, this problem is aimed to be avoided, exploiting the higher accuracy in lower dimensions, without the overall loss of freedom. To test this, the algorithm has been run several times using dimensionality increase. This was done for both the Kanaal and Egg model, implementing a CSI reparameterization. Herein, initially 2 inputs per well were used for the Kanaal model, whilst 4 inputs per well were taken for the Egg model, noting that this model has a greater total number of control parameters. The results of a single run for each model are given in figure 8.5. In this figure, each triangular marker corresponds to the start of an outer loop in which the number of degrees of freedom is doubled.

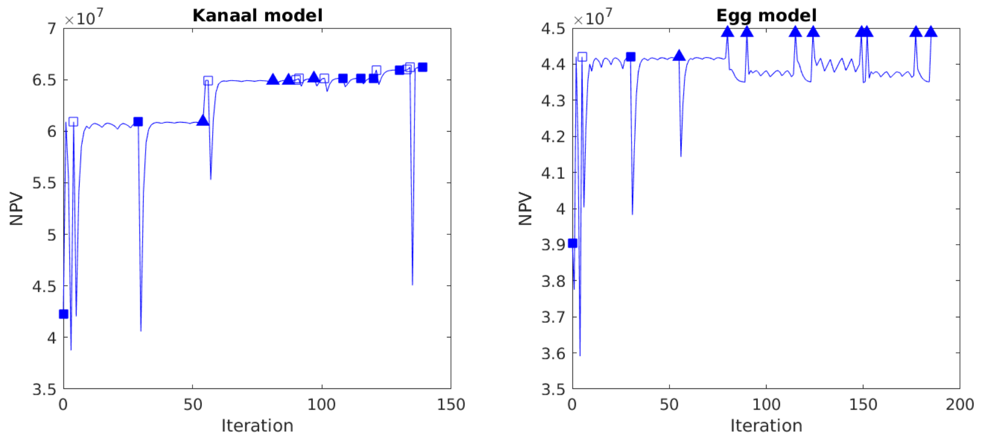


Figure 8.5: Increase of objective function using dimensionality increase with cubic spline interpolation

As the figure shows, increasing the number of input dimensions based on a CSI reparameterization, similar objective function values can be achieved as when considering the full input. For both methods, significant increase in this value is attained with the first outer loop, after which the increase becomes more limited. These later jumps in the objective function are in fact allowed by adding more degrees of freedom, though this does not guarantee improvements. In fact, an optimum was always observed to be established before all available degrees of freedom were exploited, though the exact number of necessary control parameters varied between runs.

Overall, the obtained cost function values using this scheme are comparable to those using the full input. However, achieving these values required a larger number of inner iterations. As also displayed in figure 8.5, the computed search direction at the start of outer loops would often be poor, at which point the search algorithm would expend a large number of iterations without yielding any improvement. This result is surprising, as the gradient of the cost function was observed to be more accurate upon using a reduced control. A potential reason for this is that, in the lower dimensional setting, a local optimum is established more quickly, at which point establishing a good search direction becomes more difficult.

8.4.2 Step function reparameterization

In addition to using the CSI reparameterization, tests were also performed increasing the dimensionality using a step function reparameterization. For this method, the same initial number of control parameters was considered, corresponding to a single step for the Kanaal model and two steps for the Egg model at each controlled well. This number of steps would be doubled between certain outer loops, splitting each constant segment into an additional step function. The results of two runs for the Kanaal model are displayed in figure 8.6.

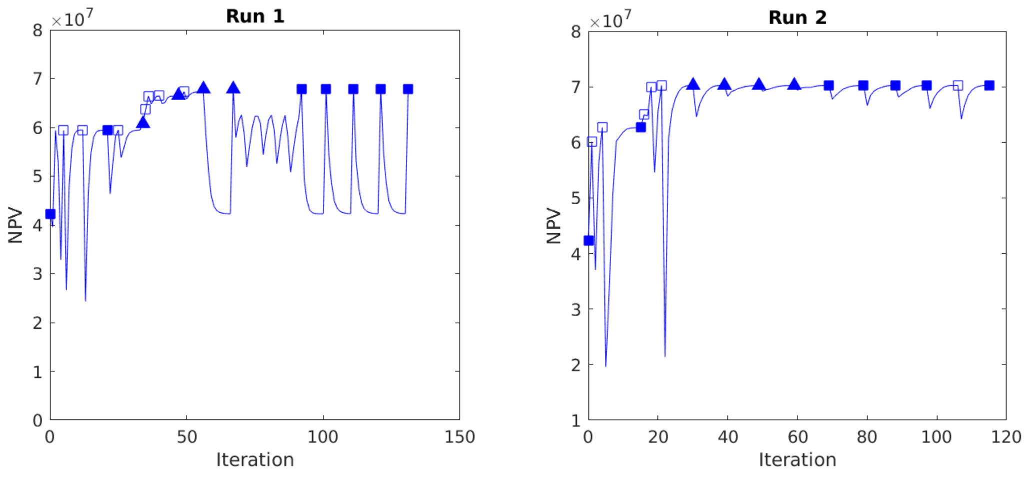


Figure 8.6: Increase of objective function using dimensionality increase with step function control

As can be seen from the figure, varying results were attained. These were largely similar to those using the cubic spline interpolation, attaining comparable results, and requiring a large number of inner iterations. Using a step function, however, the higher number of degrees of freedom was better exploited than for the CSI method, often yielding slightly better results. This can be seen in the first graph, where a new optimum is established almost every time the dimensionality is increased. This is likely because each increase in dimensionality translates more directly into additional freedom for the step function method than for the CSI method, splitting each constant input value into two independent values. In a situation where this additional freedom does not allow for better solutions, however, the step function algorithm will perform a large number of search iterations to establish a new optimum for the higher-dimensional input, without success. In this manner, even if an optimum is established in a few outer loops, a large number of iterations will be performed before the algorithm terminates. This can also be seen in the second figure, where a good objective function value is achieved using few control parameters, but a large number of search iterations are nevertheless performed, attempting to determine a better solution using the full input.

Overall, both dimensionality increase schemes provide good results, attaining similar values to the algorithm using all input parameters. Very often, these higher values are attained using few control parameters, and increasing the dimensionality offers only minor improvements, or no improvement at all. Occasionally, the obtained objective value in reduced space exceeds that generally achieved utilizing the full control vector, in particular for the Kanaal model, for which reduced control was already established to be well-suited. Nevertheless, even in this case, the overall number of iterations expended before terminating is significantly greater than upon starting with the full control vector. This is a consequence of the algorithm unsuccessfully seeking to improve the optimum using the additional freedom. As such, a multiscale implementation of the algorithm seems promising, but adjustments are necessary to reduce the computational effort.

Chapter 9

Numerical Comparison with the Ensemble Methodology

In this chapter, the performance of the POD-TPWL method as outlined in chapter 7, is compared to that of an ensemble method. Experiments are conducted using different sample sizes, starting with an uninformed guess of the initial input. In addition, an experiment is displayed upon starting with a random guess of the input, for each of the available reservoir models.

9.1 Ensemble optimization

With the reduced adjoint method functioning as desired, the question remains whether it is actually capable of improving upon already existing methods. In testing this, we have compared the algorithm to an ensemble method, specifically the straight gradient implementation as described by (3.33). This method was chosen as it computes an estimate of a non-smoothed gradient of the cost function, similarly to the adjoint method. Moreover, this method constructs such a gradient at an arbitary input $\mathbf{u}_b$, based solely on results from a limited number of perturbations to the input $\mathbf{u}_b$. This allows for a clear comparison between both the attained results and the computational cost of the two methods. Since the ensemble approach is also a popular method for efficient optimization in reservoir simulation, it provides a good indication of the state-of-the-art the POD-TPWL method is desired to match, or even exceed.

In implementing the EnOpt method, the algorithm described in chapter 7 was slightly adjusted, only altering the computation of the gradient in each outer loop, to assure a fair comparison between the methods. To this end, the same number N_{POD} of samples was also generated in each outer iteration as for the POD-TPWL method. However, rather than computing these samples according to a Latin hypercube scheme, they were generated from a standard normal distribution, as also done in most implementations of ensemble schemes. In addition, these samples were consistently scaled with a factor 0.5, rather than adjusting this factor between outer loops. Then, an ensemble gradient could be computed according to equation (3.33), which was normalized to provide a search direction for algorithm 1. Since the ensemble method does not allow a new gradient to be computed without generating new samples, only a single line search was conducted during each outer loop, corresponding to fixing $N_{\text{inner}} = 1$ in the algorithm. As also no reduced-order model is available for the ensemble method, the Taylor approximate linear model (3.7) was provided as model function for the search algorithm in each outer iteration.

To compare the performance of the two methods, several experiments have been conducted. In each, both algorithms were run for a fixed number of outer loops, starting with the same initial guess of the input, and adding the same number of samples in each outer iteration. For the first experiment, the initial guess was fixed at the center of the domain, assuming no prior information on the optimal input. In this experiment, the number of samples was limited to $N_{\mathrm{POD}} = 25$, to keep the computational cost low. As this number of samples affects the performance, however, an additional experiment was run with $N_{\mathrm{POD}} = 50$ samples, using the same initial guess. Finally, an experiment was run once more implementing $N_{\mathrm{POD}} = 25$, but starting at a random initial guess of the input, away from the center of the domain. Collecting the net present values attained using the different methods, as well as the number of full-order model simulations necessary to attain these values, the performance of the EnOpt and POD-TPWL methods could then be compared under different circumstances.

9.2 Uninformed initial guess

For the initial comparison of the EnOpt and POD-TPWL algorithms, each was run five times, collecting the results of the different runs. In these simulations, the initial guess of the input was always placed at the center of the computational domain, such that $\mathrm{u}_i = \frac{1}{2}(\mathrm{u}_{\min} + \mathrm{u}_{\max})$ for each input parameter u_i. Furthermore, $N_{\mathrm{POD}} = 25$ perturbations were consistently generated at each outer loop in order construct a new gradient. The mean of the results was taken over the five runs, to account for the stochastic variations induced by the random perturbations. This experiment was performed for both the Kanaal and Egg model, though fewer outer loops were implemented for the Egg model because of the higher cost.

9.2.1 Kanaal model

The average of the net present value at the end of each outer loop for the Kanaal model is plotted in figure 9.1.

The figure shows that, applying the POD-TPWL method to the Kanaal model as described, it generally achieves a greater value of the objective function than the EnOpt method. Especially in the first step, the reduced adjoint method greatly improves the net present value, increasing it to an extent unmatched by the ensemble method. In consecutive iterations, this difference decreases as the ensemble method improves the input more quickly than the adjoint method, though at no point achieving a greater value. Instead, both methods seem to achieve a local optimum within 8 iterations, offering no more improvements with additional outer loops. Notably, this optimal input differs greatly for the two methods, with an example given in figure 9.2.

As the figure shows, the optimal input parameters at the injection wells achieved through the EnOpt algorithm assume less extreme values than those computed using the POD-TPWL method. This is likely due to the different direction taken already during the first step, resulting in the optimal inputs of the two methods diverging, until both settle in different local optima. These local optima attained vary even between runs of the same algorithm, suggesting a highly irregular field where the net present value assumes local maxima at many different values of the input.

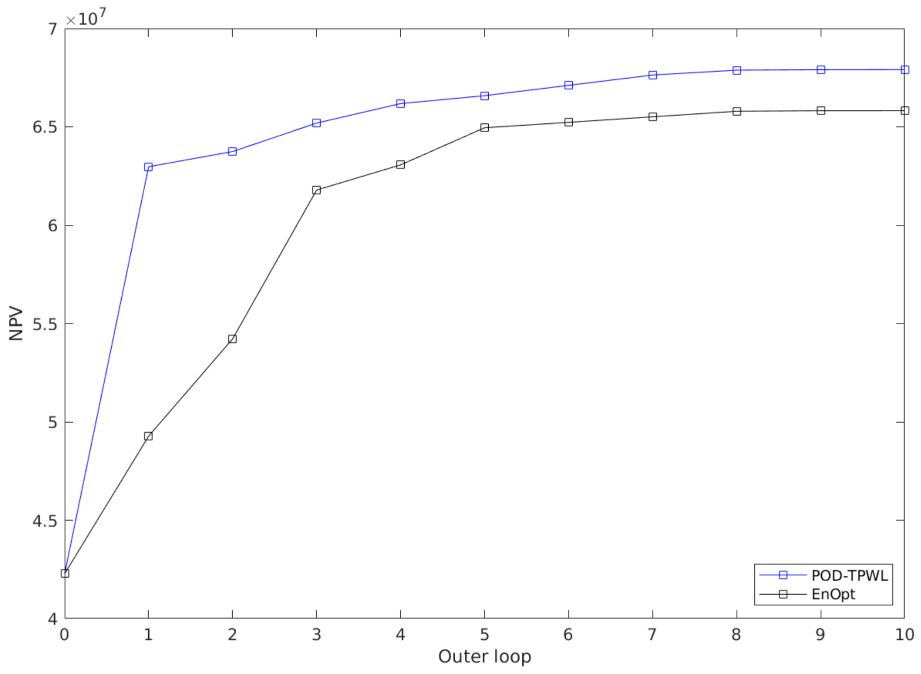


Figure 9.1: Objective function increase applying reduced adjoint and ensemble method to the Kanaal model

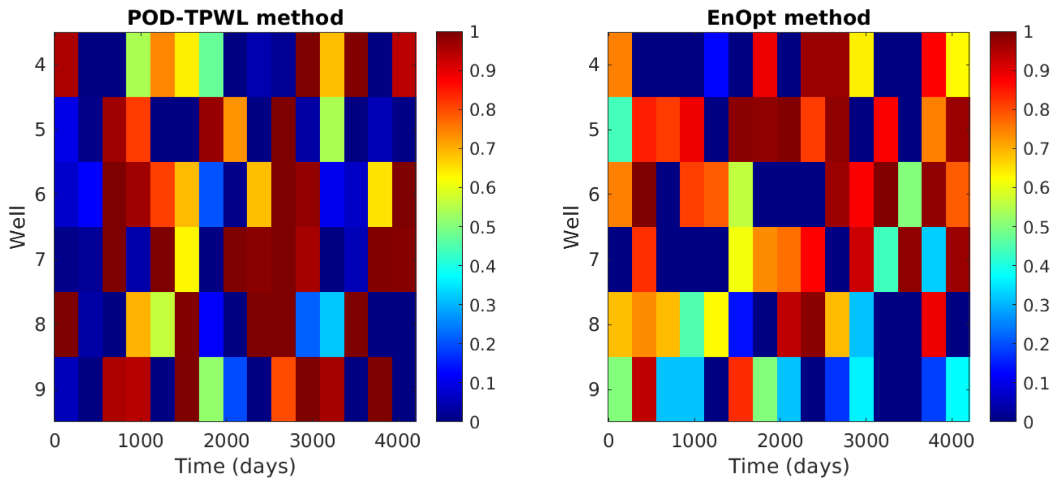


Figure 9.2: Example of input functions at injection wells after ten outer loops of the POD-TPWL and EnOpt algorithms for the Kanaal model

Although the final net present value is generally greater for the POD-TPWL method, this also comes at a higher computational cost. The EnOpt method applies a single search algorithm during each outer loop, but the reduced adjoint method can run multiple of these, computing a new gradient in between by replacing one of the RBF samples. As a result, the required number of full-order model runs also increases more rapidly for the POD-TPWL algorithm, as displayed in figure 9.3. This figure provides the mean cumulative number of full-order model runs at each outer loop of the two algorithms, based on the five different simulations. In this figure, the dashed line assumes recomputation of the weighting vectors and derivative matrices in each inner loop to be of comparable cost to a single full-order model run, adding these to the actual number of performed runs.

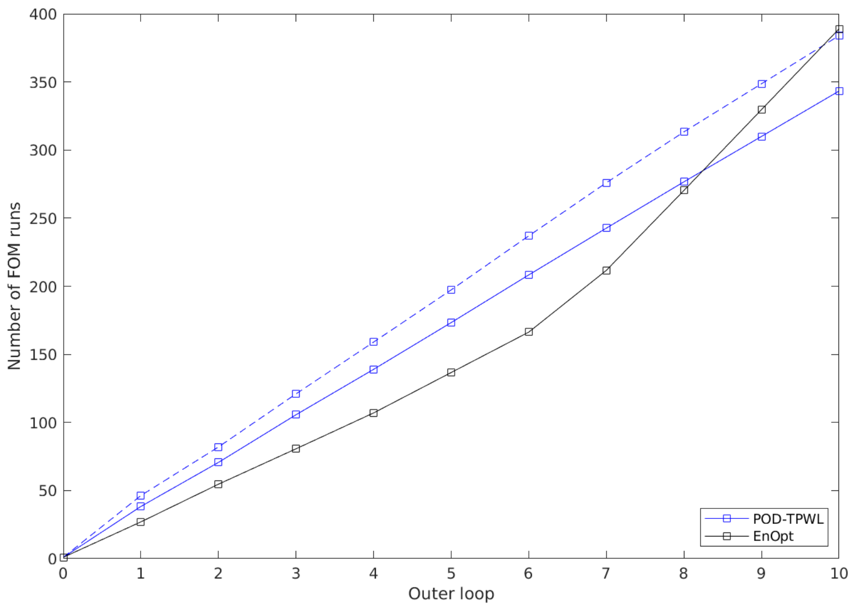


Figure 9.3: Cumulative number of full-order model runs applying the reduced adjoint and ensemble method to the Kanaal model with $N_{\mathrm{POD}} = 25$

As the figure shows, the required number of full-order model runs after the initial outer loops is higher for the POD-TPWL method than for the EnOpt method. The largest portion of these runs are due to the POD samples computed at the start of each loop, which are equal for both methods. Nevertheless, as the number of outer loops increases, the additional number of FOM runs required for the inner loops of the POD-TPWL method also grows fairly big. This makes the EnOpt method less expensive for the initial loops, though the EnOpt graph does surpass that of the POD-TPWL method, following an increase in the slope after six loops. This is the result of a local optimum having been established, but the algorithm being forced to continue running additional outer loops. As a result, the number of search iterations performed in each outer loop increases drastically, as the Wolfe condition remains unsatisfied. This suggests that, under alternative stopping criteria, the EnOpt algorithm will achieve an optimum relying on fewer runs of the full-order model, though this optimum is generally of lower value than that achieved using the POD-TPWL method.

9.2.2 Egg model

Figure 9.4 gives the mean net present values at the end of several outer loops for the Egg model, computed based on five runs of the ensemble and adjoint algorithm.

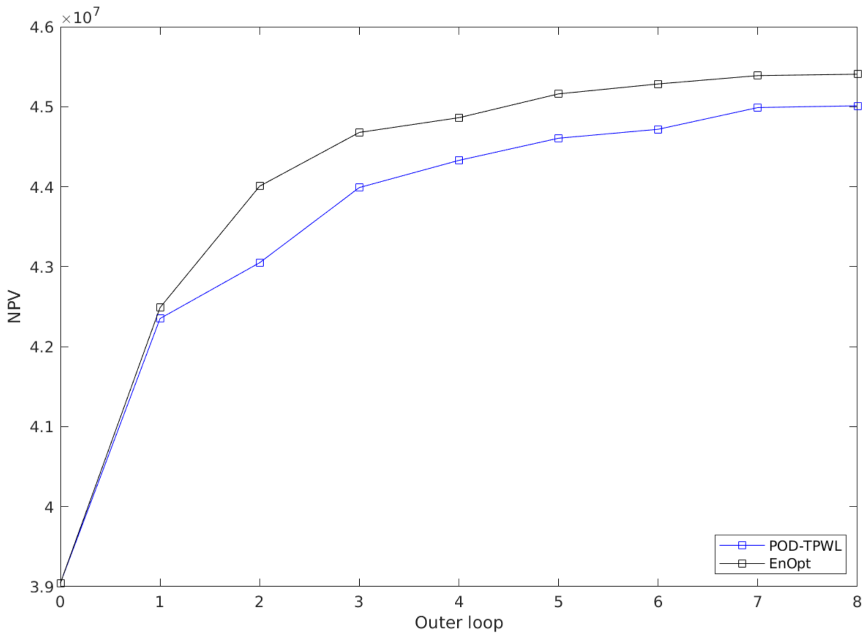


Figure 9.4: Objective function increase applying the reduced adjoint and ensemble method to the Egg model with $N_{\text{POD}} = 25$

The figure shows that, unlike for the Kanaal model, the EnOpt method performs better than the POD-TPWL method for the Egg model, yielding a greater average net present value at each of the outer loops. Surprisingly, this difference emerges starting only in the second outer loop, where the initial step yields an almost identical increase for both methods. As with the Kanaal model, however, this initial step already yields very different input functions, an example of which is displayed in figure 9.5. This suggest that, like for the Kanaal model, the net present value displays a highly non-convex dependence on the input, with many different inputs yielding the same net present value. This is also evident from the different input functions that occur after running the same algorithm multiple times, each run establishing a different local optimum.

In addition to producing a lower net present value, the reduced adjoint method also requires a greater computational effort to attain this value, as shown in figure 9.6. This is once more due to the various inner loops, allowing several line searches to be performed during each outer loop, adding to the required number of full-order model runs. Although these inner loops do occasionally yield new optima (as can be seen for example in figure 8.1), and do so at a lower cost than computing new perturbations, this is not sufficient to exceed or even match the results of the EnOpt method.

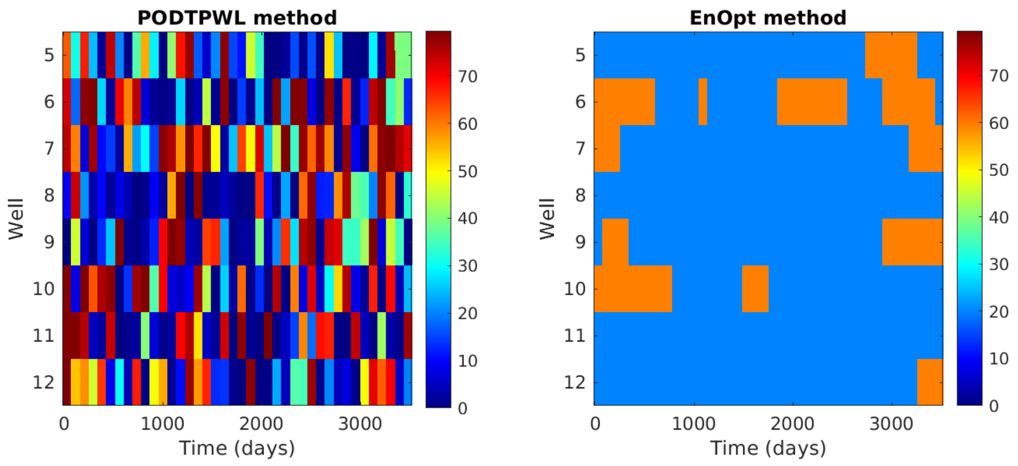


Figure 9.5: Example of input functions at injection wells after a single outer loop of the POD-TPWL and EnOpt algorithms for the Egg model

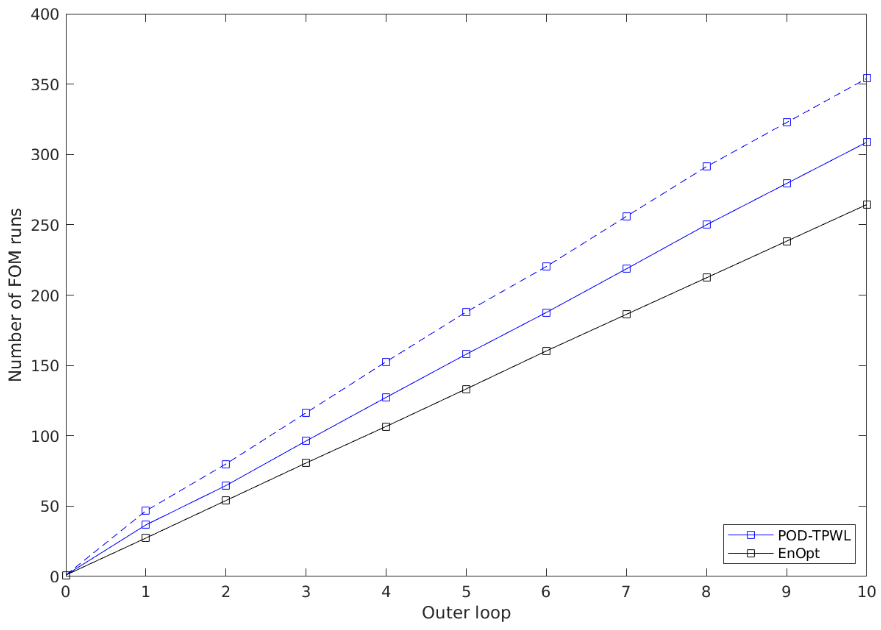


Figure 9.6: Cumulative number of ful-order model runs applying the reduced adjoint and ensemble method to the Egg model with $N_{\text{POD}} = 25$

Overall, the POD-TPWL method as applied here requires a greater computational effort than the EnOpt algorithm. Expending this effort, it achieves better results than the EnOpt method for the Kanaal model, but returns worse values for the Egg model. Since the Egg model is a more realistic model of a physical reservoir, the ensemble method remains preferable to the reduced adjoint method under these circumstances.

9.3 Increased sample size

Although the POD-TPWL method was found to produce worse results than the EnOpt method for the Egg model, this was based on a sample size of just $N_{\text{POD}} = 25$. Since the accuracy of the approximated gradient increases with the number of samples, an experiment was also conducted using a larger sample size of $N_{\text{POD}} = 50$. Computing 50 perturbations in each outer loop, the algorithm was run once again using both the reduced adjoint and ensemble method. However, as this doubles the computational effort required between each outer loop, the number of outer loops in this experiment was decreased. In addition, only three runs were considered for each model, of which the average was once more determined.

9.3.1 Kanaal model

The observed mean increase in objective function value for the Kanaal model is given in figure 9.7. This figure also provides the corresponding mean number of full-order model runs performed at the end of each outer loop.

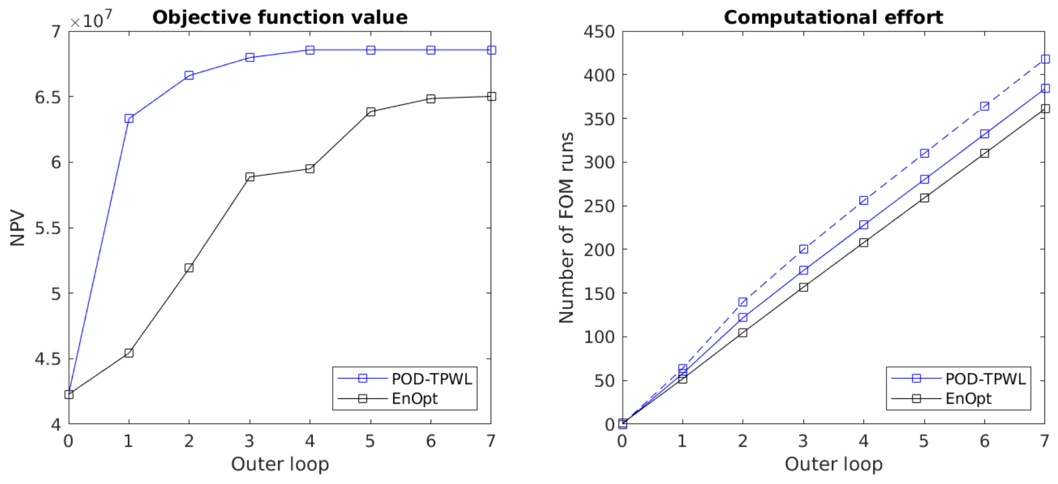


Figure 9.7: Performance of the algorithm applying the reduced adjoint and ensemble method to the Kanaal model with $N_{\text{POD}} = 50$

The figure shows that, using a larger set of POD samples for both the POD-TPWL and EnOpt algorithm, the POD-TPWL algorithm still performs better. In fact, a very similar increase is observed as in figure 9.1, where the initial step of the reduced adjoint algorithm is very large, whereas the ensemble method displays a more gradual increase. Surprisingly, neither algorithm performs significantly better upon increasing the number of samples, with the EnOpt method even yielding a slightly lower result. A potential explanation for this is the irregular dependence of the cost function on the input vector also discussed before, yielding a lot of local optima in the computational domain. As a result, the algorithm will easily guide the input to one of these optima, at which point even a more accurate approximation of the gradient is not sufficient to improve the cost function value.

Increasing the number of perturbed samples for each method, the required number of full-order model runs increases accordingly, as also seen in figure 9.1. This figure additionally shows that, under these circumstances, the POD-TPWL algorithm will still rely on more of such runs to improve the cost function value between outer loops. This effect is particularly clear upon including the RBF interpolation and derivative computation in the assessment. Nevertheless, the relative contribution of the inner iterations to the total number of full-order model runs still decreases, reducing the difference between the required effort for each algorithm. The relative performance of the POD-TPWL method compared to the EnOpt method thus improves with the number of perturbations, though the overall gain of using additional samples is minimal.

9.3.2 Egg model

Repeating the experiment for the Egg model, the observed mean increase and corresponding required number of full-order model runs are displayed in figure 9.8. The figure shows that, as also observed for the Kanaal model, increasing the number of samples computed in each outer loop does not yield a significant change in the performance of either algorithm. For both, the required number of full-order model runs increases as expected, but this does not provide an improved final net present value. Instead, the increase in this objective function value displays an almost identical shape to that in figure 9.4, with both algorithms achieving very similar results in the initial outer loops, but the POD-TPWL method still attaining a lower value than the EnOpt algorithm as more loops are performed.

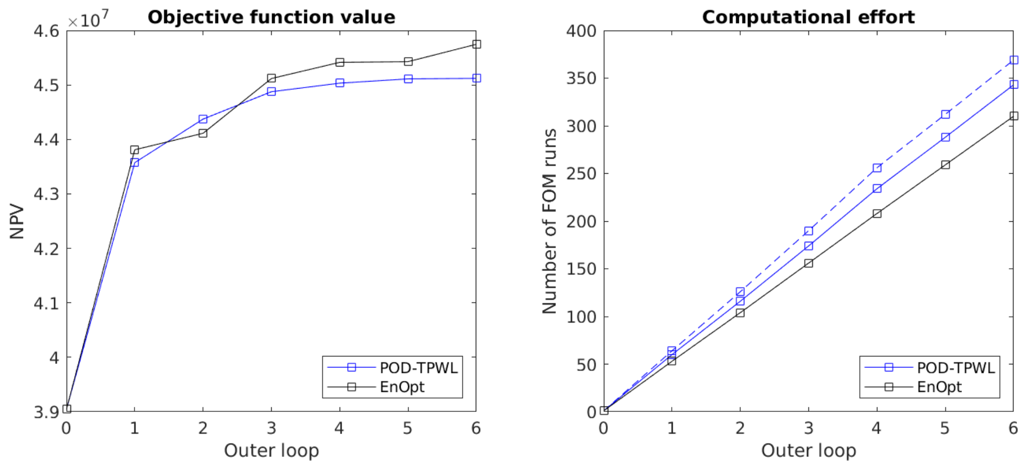


Figure 9.8: Performance of the algorithm applying the reduced adjoint and ensemble method to the Egg model with $N_{\text{POD}} = 50$

Overall, increasing the number of samples does not enhance the performance of either the POD-TPWL or EnOpt algorithm, only increasing the required computational effort. Since the accuracy of the gradients does generally increase with the number of samples, alternative implementations of these gradient computation schemes might be able to better exploit the greater sample size. However, the particular algorithms as implemented for this thesis fail to do so. Instead, using only a small number of samples for these allows for more efficient optimization, with the ensemble methodology generally performing better.

9.4 Random start

In the previous experiments, the initial guess of the input was always placed at the center of the computational domain, assuming no prior knowledge on where an optimum might be situated. However, if such knowledge is available, the initial guess might be taken in a region where the optimum is expected, or desired, to be situated. To mimic this, we performed a final experiment where the input was placed at a random location within the range of possible inputs. Running both the EnOpt and POD-TPWL algorithm starting at this input, we could then see whether the results attained before extend to situations with a different initial guess. The results of this experiment are displayed in figure 9.9 below. These graphs show the increase of the objective function for both methods applied to both models, starting at the same randomly computed initial input.

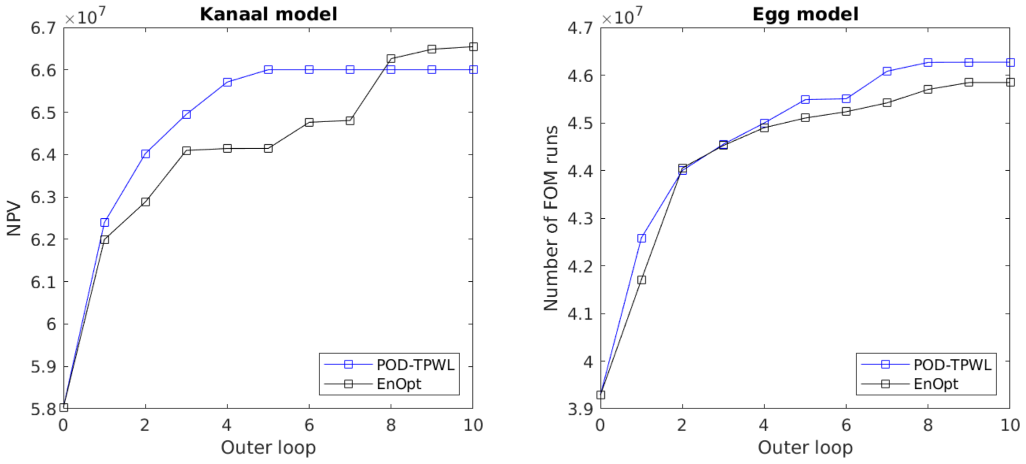


Figure 9.9: Objective function achieved applying the reduced adjoint and ensemble method to the Kanaal and Egg model with a random initial input

As the figure shows, for both models the starting net present value is higher using the random input than the center of the domain, though only for the Egg model this also translates into a greater final value. Remarkably, this value is bigger for the POD-TPWL method than the EnOpt method for the Egg model, whereas the opposite holds true for the Kanaal model, directly contradicting the results obtained earlier. This implies a more complex relation between the two methods, where the POD-TPWL method can in fact provide better results for both models, depending on the choice of initial value. Nevertheless, the overall cost of this method remains higher, and there is no guarantee this additional cost also yields a better input. In this sense, although the POD-TPWL method shows potential, the EnOpt algorithm at this point remains favorable.

Chapter 10

Conclusion and Discussion

Overall, the subdomain POD-TPWL method provides an elegant way to compute the gradient of an arbitrary cost function at a reduced cost. Through this method, the adjoint scheme for computing an exact gradient can be applied to a sample based approximation which, although less accurate, is significantly more numerically feasible. In particular, although the full adjoint method becomes computationally strenuous for large-scale systems, these systems can be addressed as well by applying principal orthogonal decomposition. This can be done using just a limited number of samples, which can additionally be employed to construct the necessary derivative matrices, without requiring access to the model code. The cost of this method can be further reduced by combining it with a domain decomposition, potentially also increasing the accuracy of the method. Finally, a reduced cost vector could also be used to further augment the efficiency.

Considering the three questions posed at the start of this thesis, the following was observed:

1. *How do the different stages of the algorithm affect its overall performance?*

   In applying the POD-TPWL methodology, the RBF interpolation for construction of the surrogate model exerts the greatest influence over the accuracy of the approximate gradients. In particular, this method depends strongly on the number of samples supplied for the interpolation, where a considerable amount of samples is necessary to yield a substantial increase in accuracy. This is in contrast to the POD method, for which the number of provided snapshots was not observed to significantly affect the accuracy of the projection. Instead, particularly upon application in a subdomain framework, a limited number of samples already allowed for significant reduction of the state vector size, whilst maintaining a high accuracy.

   Based on these results, improvements of the POD-TPWL methodology can likely be achieved focusing on the RBF interpoplation. This interpolation depends strongly on how the samples are chosen, being determined directly by the distances from the different samples. As such, alternative sampling schemes might be investigated, potentially improving the accuracy of the interpolation. In addition, alternate subdomain methods could also be considered, as the effects of this decomposition on the RBF interpolation has not been extensively studied for this thesis. Such methods could, for example, include correlations between diagonally neighboring domains, try to implement a more radial dependence between cells, or simply consider using more or fewer domains.

2. *How do different schemes reducing the number of parameters to be optimized influence the efficacy of the algorithm?*

   Implementing a reparameterization scheme, the POD-TPWL method yields a more accurate gradient upon using the same number of samples. However, this increased accuracy does not always translate into an improved performance, partially due to the complex dependence of the objective function on the input, displaying a large number of locally optimal solutions. In addition, each reparameterization scheme imposes a certain shape onto the full input function, resulting in a strongly varying performance when applied to different models. Optimal inputs for different models also require different numbers of parameters in the reduced control vector, displaying the need for a multiscale method. However, simply increasing the number of control parameters during the optimization does not achieve desirable results, as the algorithm has difficulty exploiting the increased freedom at each iteration, expending a lot of effort with little reward.

   To resolve the encountered issues, more advanced multiscale methods could be implemented. Such implementations could adjust the size of the reduced control in a smarter way, for example using gradient information as suggested in [26], to better exploit the benefits a reparmaterization of the input has to offer.

3. *How does the POD-TPWL method compare to a similar ensemble-based method when applied to the problem of production optimization?*

   Comparing the POD-TPWL algorithm to a similar EnOpt algorithm, the two are able to attain comparable values of the objective function. Herein, the initial guess of an optimal input most strongly affects the final result, with each method providing superior results to the other under different starting conditions. Surprisingly, an increased amount of samples does not strongly improve these results for either method, likely due to the large number of local optima allowed for this problem. As the gradients are computed on the basis of random perturbations, different runs of either algorithm also return different such optima, making it difficult to establish which method performs best under general conditions. Nevertheless, as the required cost in performing the POD-TPWL method tends to be higher, without guaranteeing a better result, the ensemble method remains preferable at this stage.

   To enhance the appeal of the POD-TPWL method, we note that, in order to compute the objective function gradient at some input, the ensemble method requires constructing a new set of samples around this input. In this aspect, the reduced adjoint methodology offers a significant computational advantage, as it allows gradients to be computed at any input in the computational domain, on the basis of earlier samples. Although the accuracy of such gradients would be poor for a general input, it was found to be fairly accurate if the input was amongst the samples used for the RBF interpolation. For this reason, the optimal input at the end of each inner loop was also added to the RBF samples, recomputing the weighting vectors and consecutively the gradient. Upon testing the algorithm, however, these inner iterations were found to provide only minor improvements in the objective function value, requiring more expensive outer iterations to achieve more significant changes.

To improve upon our results, future methods could consider alternative implementations of the intermediate gradient computation. For example, new criteria could be established for discarding RBF samples between inner loops, retaining only those sufficiently close to the new optimal input. In this manner, the RBF samples might provide a more accurate representation of the domain around this input, potentially resulting in a more accurate gradient. Performing adjustments like these, fewer samples might be required to achieve the same results, decreasing the necessary number of full-order model simulations. In this manner, the cost of the POD-TPWL algorithm could be reduced, thereby increasing its appeal compared to ensemble based algorithms.

Considering these different results and suggestions, it is important to note the vast freedom in constructing an algorithm based on the described techniques. The choice of search algorithm and gradient implementation, as well as the extensive number of parameters implemented at the different stages, each affect the overall performance of the final method. Consequently, even though the method as implemented for this thesis does not improve upon alternative schemes, a large variety of similar algorithms can be devised which might achieve this result. Significant improvements can likely be attained investigating the suggested areas, enhancing the results or reducing the computational effort. In this manner, further research can unlock the full potential of the subdomain POD-TPWL methodology, to transcend the performance of the currently available alternatives.

Bibliography

[1] Antoulas AC. 2005. Approximation of large-scale dynamical systems. Society for Industrial and Applied Mathematics.

[2] Bear J. 1972. Dynamics of fluids in porous media. Dover, New York.

[3] Ben-Tal A, Ghaoul LE, Nemlrovski AS. 2009. Robust optimization. Princeton University Press.

[4] Bishop CH, Frolov S, Allen DR, Kuhl DD, Hoppel K. 2016. The local ensemble tangent linear model: an enabler for coupled model 4D-Var. Quarterly Journal of the Royal Meteorological Society, 143(703): 1009-1020. doi: 10.1002/qj.2986.

[5] BP. 2019. *Statistical Review of World Energy 2019.* 68th edition. Available from: `https://www.bp.com/en/global/corporate/energy-economics/statistical-review-of-world-energy.html`

[6] Encyclopedia Britannica.

[7] Cardoso M, Durlofsky LJ. 2009. Developement and application of reduced-order modeling procedures for subsurface flow simulation. International Journal for Numerical Methods in Engineering, 77(9): 1322-1350. doi: 10.1002/nme.2453.

[8] Chen G, Chang K, Xue X, Zhang L, Jun Y, Sun H, Fan L, Yang, Y. 2019. Surrogate-assisted evolutionary algorithm with dimensionality reduction method for water flooding production optimization. Journal of Petroleum Science and Engineering, 185. doi: 10.1016/j.petrol.2019.106633.

[9] Chinchapatnam PP, Djidjeli K, Nair PB. 2007. Domain decomposition for time-dependent problems using radial basis meshless methods. Numerical Methods for Partial Differential Equations, 23(1): 38-59. doi: 10.1002/num.20171.

[10] De Boor C. 1978. A Practical Guide to Splines. Springer-Verlag, New York.

[11] Do ST, Reynolds AC. 2013. Theoretical connections between optimization algorithms based on an approximate gradient. Computational Geosciences, 17: 959–973. doi: 10.1007/s10596-013-9368-9.

[12] U.S. Energy Information Administration. 2019. International energy outlook 2019. Available from: `https://www.eia.gov/outlooks/ieo/`

[13] Fasshauer GE, Zhang JG. 2007. On choosing “optimal” shape parameters for RBF approximation. Numerical Algorithms, 45: 345-368.

[14] Ferziger JH, Perić M. 2002. Computational methods for fluid dynamics, 3rd edition. Springer.

[15] Fonseca R, Chen B, Jansen JD, Reynolds AC. 2016. A stochastic simplex approximate gradient (StoSAG) for optimization under uncertainty. International Journal for Numerical Methods in Engineering, 109(13): 1756-1776. doi: 10.1002/nme.5342.

[16] He J, Sætrom J, Durlofsky LJ. 2011. Enhanced linearized reduced-order models for subsurface flow simulation. Journal of Computational Physics, 230(23): 8313-8341.

[17] He J, Sarma P, Durlofsky LJ. 2013. Reduced-order flow modeling and geological parameterization for ensemble-based data assimilation. Computational Geosciences, 55: 54-69.

[18] He J, Durlofsky LJ. 2014. Reduced-order modeling for compositional simulation by use of trajectory piecewise linearization. Society of Petroleum Enigneers Journal, 19(5): 858-872. doi: 10.2118/163634-PA.

[19] Jansen JD. 2013. A system description of flow through porous media. Springer.

[20] Jansen JD, Fonseca RM, Kahrobaei M, Siraj MM, Van Essen GM, Van den Hof PMJ. 2014. The egg model - a geological ensemble for reservoir simulation. Geoscience Data Journal, 1(2): 192-195. doi: 10.1002/gdj3.21.

[21] Jansen JD. 2016. Gradient-based optimization of flow through porous media, version 3.

[22] Kanimozhi B, Prakash J, Pranesh RV, Mahalingam S. 2019. Numerical and experimental investigation on the effect of retrograde vaporization on fines migration and drift in porous oil reservoir: roles of phase change heat transfer and saturation. Journal of Petroleum Exploration and Production Technology, 9: 2953–2963. doi: 10.1007/s13202-019-0692-z.

[23] Kim J, Kang B, Jeong H, Choe J. 2019. Field development optimization using a cooperative micro-particle swarm optimization with parameter integration schemes. Journal of Petroleum Science and Engeneering, 183. doi: 10.1016/j.petrol.2019.106416.

[24] Klie H. 2013. Unlocking fast reservoir predictions via non-intrusive reduced-order models. From: SPE reservoir simulation symposium, The Woodlands, Texas, USA, 18-20 February 2013. doi: 10.2118/163584-MS.

[25] Liberti L, Maculan N. 2006. Global optimization: from theory to implementation. Springer.

[26] Lien ME, Brouwe DR, Mannseth T, Jansen JD. 2008. Multiscale regularization of flooding optimization for smart field management. Society of Petroleum Engineers Journal, 13(2): 195:204. doi: 10.2118/99728-PA.

[27] Liu Z, Reynolds AC. 2019. An SQP-filter algorithm with an improved stochastic gradient for robust life-cycle optimization problems with nonlinear constraints. From: SPE reservoir simulation conference, Galveston, Texas, USA, 10-11 April 2019. doi: 10.2118/193925-MS.

[28] Oliveira DFB, Reynolds AC, Jansen JD. 2015. An improved multiscale method for life-cycle production optimization. Computational Geosciences, 19: 1139–1157. doi: 10.1007/s10596-015-9530-7.

[29] Markovinović R, Jansen J. 2006. Accelerating iterative solution methods using reduced-order models as solution predictors. International Journal for Numerical Methods in Engineering. 68(5): 525-541. doi: 10.1002/nme.1721.

[30] Meyer CD. 2000. Matrix Analysis and Applied Linear Algebra. Society for Industrial and Applied Mathematics.

[31] Nocedal J, Wright SJ. 2006. Numerical optimization. Springer

[32] Open Porous Media initiative: `https://opm-project.org/`

[33] Peaceman DW. 1977. Fundamentals of numerical reservoir simulation. Elsevier Scientific Publishing Company, Amsterdam.

[34] Pearson K. 1901. On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2(11): 559-572.

[35] Pinto JWO, Afonso SMB, Willmersdorf RB. 2019. Robust optimization formulations for waterflooding management under geological uncertainties. Journal of the Brazilian Society of Mechanical Science and Engineering, 41:475. doi: 10.1007/s40430-019-1970-x.

[36] Roser M, Ritchie H, Ortiz-Ospina E. 2020. World Population Growth. Published online at OurWorldInData.org, 2013 [updated May 2019, accessed March 2020]. Retrieved from: `https://ourworldindata.org/world-population-growth`

[37] Schilders WHA, van der Vorst HA, Rommes J. 2008. Model order reduction: theory, research aspects and applications. Springer. doi: 10.1007/978-3-540-78841-6.

[38] Snyman JA. 2005. Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms. Springer.

[39] Van Doren JFM, Markovinović R, Jansen JD. 2006. Reduced-order optimal control of waterflooding using proper orthogonal decomposition. Computational Geosciences, 10: 137-158.

[40] Wolfe P. 1969. Convergence conditions for ascent methods. SIAM Review, 11(2): 226-235. doi: 10.1137/1011036.

[41] Wolfe P. 1971. Convergence conditions for ascent methods II: some corrections. SIAM Review, 13(2): 185-188. doi: 10.1137/1011036.

[42] Xiao D, Fang F, Pain C, Navon I, Muggeridge A. 2016. Non-intrusive reduced order modelling of waterflooding in geologically heterogeneous reservoirs. From: ECMOR XV-15th European conference on mathematics of oil recovery, August 2016. doi: 10.3997/2214-4609.201601854.

[43] Xiao C, Leeuwenburgh O, Lin HX, Heemink A. 2019. Non-intrusive subdomain POD-TPWL for reservoir history matching. Computational Geosciences, 23: 537-565. doi: 10.1007/s10596-018-9803-z.