Faculty of Mathematics and Computer Science
University of Münster

Master Thesis

A trust region-reduced basis-machine learning approach for parameter optimization with parabolic PDE constraints

by Benedikt Simon Klein,
born August 15, 1997 in Krefeld

December 21, 2023

Für meine Großeltern, die mich immer in meinem akademischen Streben unterstützt haben.

First supervisor: Prof. Dr. Mario Ohlberger
Second supervisor: Dr. Stephan Rave

Summary

This thesis introduces a novel algorithm that integrates trust-region strategies, reduced basis (RB) methods, and machine learning (ML) to address parameter optimization problems constrained by parabolic partial differential equations (PDEs). The aim is to expedite the optimization process by integrating machine learning methods with existing algorithms.

The proposed hybrid approach incorporates trust-region BFGS-like optimization, utilizing reduced basis methods to efficiently estimate solutions to the primal and dual forward problems, significantly reducing the computational demands compared to high-fidelity models. Originally presented by Yue and Meerbergen [51] and later applied to parabolic problems by Qian et al. [39], the approach involves enriching the RB-space whenever the error of the approximative objective functional becomes too large. This error can be calculated during the optimization by utilizing online-efficient a posteriori error estimators for the RB-solutions.

To further reduce the computational costs, this approach was augmented by data-driven machine-learning-based surrogates. These return approximative RB-solutions by employing kernel-models, which are adaptively trained by the Vectorial Kernel Orthogonal Greedy Algorithm (VKOGA)[44]. The inference time of these kernel-models is notably reduced compared to the one of the RB-models and VKOGA ensures rapid training. Therefore, there is the justified hope that integrating ML-models into the existing optimization scheme would provide a significant speed up. Since the ML-models operate within the reduced spaces, the RB-error estimators remain valid for the approximative solutions provided by them, thereby ensuring their certification and making a trust-region approach applicable.

The key idea is to replace the direct computation of solutions for the PDEs in the RB-space by inferring ML-based models trained on these solutions. These ML-models are used unless their error exceeds a predetermined threshold, triggering the use of RB-models instead. Moreover, a fallback procedure ensures the use of RB-models during BFGS-backtracking if the ML-models fail to reduce the objective functional and no local minimum is reached. The RB-solutions generated during this process will be used as training data for (re-)adapting the ML-models. To avoid inconsistencies, the training, when necessary, will be performed iteratively after completing the backtracking procedure.

For numerical validation, an optimization of the quadratic misfit from a target temperature profile in a building was performed. The temperature profiles were derived by solving a heat equation, parametrized by the capacities of heater and thermal conductivities of the building components. The experimental results show that the established method still outperforms the new algorithm. However, it was also demonstrated that the ML-model processes individual parameter inferences about 50 times faster than the RB-models. Despite this speed advantage, the integration of ML-surrogate models adversely impacts the optimization process in several ways. A key issue is that the ML-approximations exhibit larger errors compared to the RB-methods, resulting in more evaluations of parameters and more frequent costly expansions of the RB-space. This highlights the necessity for additional optimization and refinement of the algorithm.

Contents

Introduction 5

1. Machine learning & statistical learning theory 8
   1.1. Empirical Risk Minimization 8
   1.2. Vectorial Kernel Orthogonal Greedy Algorithm 12

2. Problem formulation & adjoint approach 15
   2.1. Constrained optimization problem 15
   2.2. Differentiability & dual problem 17

3. Trust region algorithm 20
   3.1. Trust region based optimization 20
   3.2. BFGS-TR-Algorithm 22

4. Adaptive hierarchical framework 26
   4.1. General considerations 26
   4.2. Model framework 27
   4.3. Adaptive fallback 30

5. Models & Generators 31
   5.1. Full order model 31
   5.2. Reduced basis model 33
   5.3. Reduced basis generator 35
   5.4. Machine learning ROM & generator 39
   5.5. Objective error estimator 41

6. Dual adaptive optimization 43
   6.1. General considerations 43
   6.2. Trust region-reduced basis-machine learning optimization 45
      6.2.1. Trust region-reduced basis optimization 45
      6.2.2. Adaptive machine learning surrogates 47
      6.2.3. Machine learning based optimization 48

7. Numerical experiments 52
   7.1. Implementation 52
   7.2. Temperature distribution in a building 53
   7.3. Optimal heater capacity 55
      7.3.1. Convergence & used models 56

7.3.2. Error estimator performance 59
7.3.3. Stepsize dependency 61

8. Conclusion & Outlook **64**

A. Appendix **66**
A.1. Tables from Chapter 7 66

Bibliography **68**

Glossary

AGP Approximate generalized Cauchy point.

ANN Artificial Neural Network.

BFGS Broyden, Fletcher, Goldfarb, Shanno.

CG Conjugate gradient.

DL Deep Learning.

DoF Degree of Freedom.

ERM Empirical Risk Minimization.

FE Finite Element.

FOM Full Order Model.

HaPOD Hierachical Approximate POD.

LLN Law of large numbers.

MGS modified gram-schimdt.

ML Machine Learning.

MOR Model Order Reduction.

MSE Mean squared error.

pBFGS projected BFGS.

PDE Partial differential equation.

PINN Physics-informed Neural Network.

POD Proper Orthogonal Decomposition.

pyMOR pyMOR.

RB Reduced Basis.

RKHS	Reproducing Kernel Hilbert Space.
ROM	Reduced Order Model.
SLT	Statistical Learning Theory.
SPD	strictly positive definite.
SVD	Singular Value Decomposition.
TR	Trust Region.
VKOGA	Vectorial Kernel Orthogonal Greedy Algorithm.

Introduction

Optimization problems with constraints governed by parametrized parabolic Partial differential equation (PDE)s arise in many fields, from natural science to engineer disciplines and beyond. Typical examples are optimal design or optimal control questions, such as determining the optimal layout of a heating element to achieve a uniform temperature distribution.

Many optimization algorithms require iterative solving of the PDE governing the constraints on the parameters during the approach to the optimum. A common approach to numerically handle these calculations is to approximate the PDE through high-dimensional models obtained from discretization method, for example finite element methods. However, a significant concern arises due to the high computational costs associated with the large number of degrees of freedom needed to achieve sufficiently accurate solutions.

One way to circumvent this limitation is introducing Model Order Reduction (MOR), replacing the high-fidelity models by cheaper surrogate models (Reduced Order Model (ROM)). An approach that has been particular promising for this are Reduced Basis (RB)-methods, approximating the PDE-solutions in low-dimensional linear spaces, that are constructed with high-fidelity solutions (snapshots) as basis. However, when employing MOR-methods, a common challenge is that efficient surrogates are typically only locally valid and may not be accurate enough for all parameters that need to be evaluated during the optimization process.

A well-studied approach to nonetheless utilize the reduced models are Trust Region (TR)-methods. These methods perform local optimization within a restricted area of the parameter domain, requiring that the surrogates used are only locally accurate rather than globally. If the error of the model becomes untenable for the optimization, it will be adapted to a new region, and the optimization is performed again. In this iterative process, the TR-method converges towards the global optimum by successively determining local optima. RB-methods prove to be a well-suited choice for this type of optimization. Firstly, since a posteriori error estimator can be derived for their approximations, allowing efficient evaluation whether the solutions are still accurate enough. Secondly, they can be easily integrated into the TR-scheme, performing the adaption to the new region by enlarging the basis of the approximative linear space.

However, depending on the problem, reduced models can still incur significant computational costs. A recent proposal to address such challenges is employing Machine Learning (ML)-based models as surrogates. ML-algorithms can provide approximations to some (unknown) function $\tilde{f} : \mathcal{X} \to \mathcal{Y}$ by processing a finite set of data points $(x, \tilde{f}(x))$ (training). There is justified hope that these methods can be used to approximate solutions to the PDE by utilizing other models for generating the required data. These ML-approximations can often be evaluated with relatively low computational cost. In this way, ML-based models could potentially replace or complement other reduced models during optimization, leading to improved temporal performance and an expanded pool of available methods, which offers more flexibility and the potential for further improvements.

In this thesis, we present a novel algorithm that extends a TR-RB-optimization method for quadratic objective functionals constrained by a parametric parabolic PDE, by incorporating ML-based reduced-order models. Our goal is twofold: firstly, we aim to demonstrate how ML-methods can be effectively utilized for this task, identify potential challenges, and propose solutions. Secondly, we hope that the addition of these methods can help reducing the computational time required to achieve the optimum compared to the original variants. A key aspect of this new algorithm is its adaptability, allowing for the use of a wide variety of ML-methods for optimization, provided certain technical requirements are met. We do not commit ourselves to a specific ML-method, offering the flexibility to test different models and choose the one that best fits the task at hand.

The original TR-RB-optimization method was presented by Qian et al. in [39] featuring gradient-based optimization methods, via an adjoint approach for efficiently calculating the gradient. We will use a version of this method introduced in [28], adapting it to problems with a simple bounded parameter domain. For the adjoint approach to be used, a system of two coupled PDEs has to be solved. To perform these calculations, both equations are discretized using the Finite Element (FE) method, followed by the construction of appropriate RB-models. Therefore, a more efficient variant of the classic Proper Orthogonal Decomposition (POD) will be employed. Leveraging the RB-models, we then create ROMs that utilize machine learning, approximating the RB-estimates. This design has the advantage that a posteriori error estimator for the RB-solutions can also be applied to the ML-approximations, providing the error measurement needed for applying the TR-method. A fallback and adaptive enrichment strategy presented in [20] will be used, selecting for each parameter the best model available, based on its error. Moreover, enacts this procedure efficient (re)training of the ML-ROMs adapting them constantly to new data and avoiding costly offline-training before engaging to the optimization. An additional advantage of this adaptive framework is that it allows an easy implementation of offline-preparation steps for the other components, especially the error estimator, thereby minimizing online computational costs during the optimization process.

We start in Chapter 1 by introducing ML-methods in a general and abstract notion, in particular highlighting the difficulties that arise in designing a capable ML-algorithm. A concrete implementation using kernel methods is given, showing how some of these obstacles can be evaded. Chapter 2 gives a formal definition for the class of problems we are interested in, establishing analytic properties necessary to apply the numerical procedures and to prove associated error bounds. Additionally, the formal mathematical justification for the adjoint approach will be given. Chapters 3 and 4 deal with introducing and adapting the key aspects from previous works to our problem. In Chapter 3 the idea of the TR-method in the sense of [51] is discussed in general and afterwards a concrete implementation in the form of a BFGS-TR on simple bounded domains, inspired by [28, 29], is deployed. This is followed, in Chapter 4, by introducing the adaptive scheme from [20] and discussing the benefits and challenges of using ML-models for PDE-inference. The next Chapter applies this scheme to the coupled system by employing FE-methods and RB-methods as and defining the interface for ML-models in a general way, retaining flexibility and freedom of choice. Thereafter, in Chapter 6, this system is integrated into the TR-algorithm, addressing the specific problems that arise in contrast to the more conservative implementations in [39] or [28]. Finally, in the last Chapter 7, some numerical experiments will be performed, using an implementation of this algorithm with a kernel-based ML-model. Based on the results remaining problems and potential staring points for further optimization will be highlighted.

1. Machine learning & statistical learning theory

1.1. Empirical Risk Minimization

For the following discussions it is advisable to introduce and clarify some basic concepts and heuristics regarding ML to sharpen the subject and scope of this thesis, laying the ground work for the argumentations and discussions in the next sections. Although started in the 1950s by scientists mainly in the USA [35, 43] and some early success in until the 1990s, ML-models gained much of their popularity in the last decades. Noteworthy examples are the progresses in image processing, especially since the introduction of Artificial Neural Network (ANN) [32, 33, 42] or more recently the raised interest in the large language model ChatGPT. Notice that despite historical and recent successes, there is a significant lack of an elaborate and comprehensive mathematical theory [30] for many crucial aspects, making precise predictions for the behaviour of ML-algorithms in practical scenarios difficult.

In fact, it is not easy to give an accurate definition of what ML itself is, Goodfellow and its co-authors define it as *a form of applied statistics with increased emphasis on the use of computers to statistically estimate functions.* [18, p. 95]. The mathematical foundation is mainly build by the Statistical Learning Theory (SLT) [50, 11, 48, 5], developed since the 1960s. Machine Learning combines ideas from mathematics, computer and data science to let computer *learn* concepts by themself, without programming explicit rules defining the underlying subject to be learned. Learning in this context is the process of inferring general rules by processing examples (*training data*) [50].

Learning paradigms are usually distinguished into *supervised* and *unsupervised.* In the first case training data x are annotated by a human or an upstream computer based model and the task is to find labels y for non-annotated data. Examples could be the response of physical systems to some controlled parameter or image classification. In the later case y is for example either one or zero indicating if a car (or an other object of interest) is present on image x, decoded by the color values of the pixel. In an unsupervised situation the system is left alone with the data and has to sort them by itself in some way, for example by finding clusters. In the following discussions we will focus on *supervised* tasks. To formalize the ambiguous application we introduce the general supervised learning task in Definition 1.1.1. The theoretical discussions in this section are mainly adapted from [50, 5, 3].

Definition 1.1.1 (Supervised Learning Task)**.** Let $\mathcal{X}, \mathcal{Y}$ and $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$ be measurable spaces. The goal of learning is to find for each finite set of *training data* $z := (z_i)_{i\in\{1,\ldots,m\}} \in \mathcal{Z}^m$ and a preselected set $\mathcal{F} \subset \mathcal{M}(\mathcal{X}, \mathcal{Y})$ of functions (*hypothesis set*), a measurable function $f^* \in \mathcal{F}$ that fits the training data in an optimal way. To get a measure how well a fit is a measurable *loss function*

$$\mathcal{L} : \mathcal{M}(\mathcal{X}, \mathcal{Y}) \times \mathcal{Z} \to \mathbb{R}_{\geq 0}$$

is defined. Additionally, the loss function might has to satisfy requirements like regularity, depending on the problem. A *(machine) learning algorithm* $\mathcal{A}$ with respect to $\mathcal{F}$ is a map of the form

$$\mathcal{A} : \bigcup_{m \in \mathbb{N}} \mathcal{Z}^m \to \mathcal{F},$$

returning a model $f_z := \mathcal{A}(z) \in \mathcal{F}$ achieved by $\mathcal{A}$ for given training data z. Moreover, we assume that a method $f_z \leftarrow \texttt{train}[z]$ can be implemented on a digital computer, returning f_z in finitely many steps.

Consider now the following situation: Assume that a measurable function $\tilde{f} : \mathcal{X} \to \mathcal{Y}$ exists, describing some relation of interest. Information about the system studied, are only accessible via the response to some input data $x \in \mathcal{X}$, meaning for x we can obtain some response $y \in \mathcal{Y}$ such that $y \approx \tilde{f}(x)$. Such situations may occur because $\tilde{f}$ is unknown or too expensive to evaluate for the whole domain or a subset of interest. The goal is to find a function f_z that acts as a *surrogate* for $\tilde{f}$ *for all* $x \in \mathcal{X}$, based only on finitely many tuple $(x, y) \in \mathcal{Z}$ by a learning algorithm $\mathcal{A}$. To model this situation, we assume that there exist a random variable $Z : \Omega \to \mathcal{Z}$ with an unknown distribution $\mathbb{P}_Z$ encoding $\tilde{f}$. The training data $(z_i)_{i \in \{1,\ldots,m\}} \in \mathcal{Z}^m$ are modelled as *independent* realizations of $S := (Z_i)_{i \in \{1,\ldots,m\}} \sim \mathbb{P}_Z^m$, with Z_i independent and identically distributed. A measure of how well a function $f \in \mathcal{F}$ fits the law $\mathbb{P}_Z$ can be provided by the theoretical risk.

Definition 1.1.2 (Theoretical risk)**.** The theoretical risk for $f \in \mathcal{F}$ regarding $\mathbb{P}_Z$ is given by

$$\mathcal{R}(f) := \mathbb{E}\left[\mathcal{L}(f, Z)\right] = \int_{\mathcal{Z}} \mathcal{L}(f, (x, y)) \mathrm{d}\mathbb{P}_Z((x, y)), \tag{1.1}$$

where $\mathcal{L}$ is the loss function. The global minimizer f^* of the theoretical risk over all available functions, i.e.

$$\mathcal{R}^* := \mathcal{R}(f^*) := \inf_{f \in \mathcal{M}(\mathcal{X}, \mathcal{Y})} \mathcal{R}(f),$$

is called *Bayes-optimal function.*

The loss function is usually chosen such that $\mathcal{L}(f, (x, y)) = 0$ for all $x \in \mathcal{X}$, if $y = f(x)$ and greater than zero otherwise. The risk will therefore be minimized by a good approximation to all realizations of Z, weighted by the probability. If $f_z := \mathcal{A}(z)$ achieves a small risk with respect to $\mathbb{P}_Z$ for all possible training data z, we say that $\mathcal{A}$ has a good *generalization* performance. This can be modelled by the random variable $\mathcal{R}(f_S) := \mathbb{E}\left[\mathcal{L}(\mathcal{A}(S), Z) | S\right]$. A good learning algorithm $\mathcal{A}$ might be defined as an algorithm such that the achieved model minimizes $\mathcal{R}(f_S)$.

The obvious problem with this approach is the inaccessibility of the theoretical risk (1.1), due to the unknown distribution $\mathbb{P}_Z$. A common evasion of this problem is to replace the theoretical risk with its empirical counterpart (1.2), utilizing the convergence for an increasing number of training samples given by the Law of large numbers (LLN).

Definition 1.1.3 (Empirical risk)**.** The *empirical risk* of a function $f \in \mathcal{M}(\mathcal{X}, \mathcal{Y})$ regrading given training data $z = (z_i)_{i \in \{1,\dots,m\}}$ is defined as

$$\widehat{\mathcal{R}}_z(f) := \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f, z_i). \tag{1.2}$$

In practice it is a usual technique to expand the empirical risk by a regulation term $\lambda L(f)$, with $L : \mathcal{M}(\mathcal{X}, \mathcal{Y}) \to \mathbb{R}_{\geq 0}$ and $\lambda \in \mathbb{R}_{\geq 0}$, giving an additional degree of freedom to control the trade-off between fit and generalization. Consequentially, learning algorithms are often designed as Empirical Risk Minimization (ERM).

Definition 1.1.4 (ERM learning algorithm)**.** An ERM learning algorithm $\mathcal{A}^{\text{ERM}}$ is an algorithm in the sense of Definition 1.1.1 returning for training data z an (approximate) minimizer $\hat{f}_z$ to the according empirical risk $\widehat{\mathcal{R}}_z(\cdot)$ plus a regularization term, i.e.

$$\hat{f}_z := \mathcal{A}^{\text{ERM}}(z) \in \operatorname*{argmin}_{f \in \mathcal{F}} \left(\widehat{\mathcal{R}}_z(f) + \lambda L(f) \right). \tag{1.3}$$

Remark that in general $\hat{f}_z$ does not coincide with the Bayes-optimal function $f^* \in \mathcal{M}(\mathcal{X}, \mathcal{Y})$, since f^* might not be an element of $\mathcal{F}$. This, in fact, raises the urgent question how accurate a ERM-algorithm can become. The error in terms of risk between the Bayes-minimizer and an estimate returned by such an algorithm can be decomposed, by

$$\begin{aligned}\mathcal{R}(f_z) - \mathcal{R}^* &= \underbrace{\mathcal{R}(f_z) - \widehat{\mathcal{R}}_z(f_z)}_{(3)} + \underbrace{\widehat{\mathcal{R}}_z(f_z) - \widehat{\mathcal{R}}_z(f^*_{\mathcal{F}})}_{(1)} + \underbrace{\widehat{\mathcal{R}}_z(f^*_{\mathcal{F}}) - \mathcal{R}(f^*_{\mathcal{F}})}_{(3)} + \underbrace{\mathcal{R}(f^*_{\mathcal{F}}) - \mathcal{R}^*}_{(2)} \\ &\leq \epsilon_{\text{opt}} + \epsilon_{\text{approx}} + 2\epsilon_{\text{gen}}\end{aligned}$$

with $f^*_{\mathcal{F}} \in \operatorname{argmin}_{f \in \mathcal{F}} \mathcal{R}(f)$ and

(1) the optimization error $\widehat{\mathcal{R}}_z(f_z) - \widehat{\mathcal{R}}_z(f^*_{\mathcal{F}}) \leq \widehat{\mathcal{R}}_z(f_z) - \widehat{\mathcal{R}}_z(\hat{f}_z) := \epsilon_{\text{opt}}$, measuring the error with respect to the empirical risk between the actual minimizer of the empirical risk in $\mathcal{F}$ and the minimizer supposed by a learning algorithm,

(2) the approximating error $\mathcal{R}(f^*_{\mathcal{F}}) - \mathcal{R}^* := \epsilon_{\text{approx}}$, measuring the error with respect to the theoretical risk between the global minimizer and a minimizer in $\mathcal{F}$, and

(3) the generalization error $\sup_{f \in \mathcal{F}} |\widehat{\mathcal{R}}_z(f) - \mathcal{R}(f)| := \epsilon_{\text{gen}}$, measuring the maximal possible error for functions in $\mathcal{F}$ made by approximating the theoretical with the empirical risk.

For simplicity we have assumed here $\lambda = 0$. The following discussions assume that learning is performed by ERM, but can be generalized in main aspects. The most obvious sources of errors in the ERM-approach are (numerical) imperfections of the minimization method applied to (1.3). In general it is not clear, if an (unique) optimizer exists. Even if a minimizer exists, the algorithm might not converge (with arbitrary accuracy). In practice the converges depends heavily on the problem and the algorithm used. Further aspects like the amount of training data available or their distribution inside the domain $\mathcal{X}$, could have significant impact on the performance too. To make matters worse, if the model is used in a complex environment and maybe has to adapt to new data, these properties might vary and are a priori not predictable.

Another, little more subtile, problem is the choice of the hypothesis set. As already mentioned, in general it is not given that f^* is an element of $\mathcal{F}$. Thus even the best possible approximation given $\mathcal{F}$ could show notable deviation from the expected function $\tilde{f}$. This becomes even more a problem if the surrogates should reproduce properties like smoothness or regularity from $\tilde{f}$. Such points have to be considered in the choice of $\mathcal{F}$ (and the loss function). Therefore, the hypothesis set should not be chosen too "simple", since otherwise the available functions could not approximate $\tilde{f}$ properly. There are different definitions describing the capacity of hypothesis sets. We will not discuses them here and refer the reader to [30, 5].

However, simply extending $\mathcal{F}$ to meet these demands is usually not recommend either, because this could raise the generalization error, making $\mathcal{F}$ too "complex", diminishing the generalization by *overfitting* to the training data. Overfitting refers to the fact that the model tries to minimize the error on the training input $x := (x_i)_{i\in\{1,\dots,m\}}$, which becomes easier with a more rich hypothesis set, but this in general simultaneously reduces the fit quality on the rest of the domain, not emulating $\tilde{f}$ properly there. This effect becomes increasingly worse in regions with a low training data density. Combining these arguments suggest, that there exists an optimal complexity of $\mathcal{F}$, balancing both effects and giving an overall error minimization. This heuristic is known as *bias-variance-dilemma*. In fact, such behaviour can be observed for many models, but it could also be significantly more complex, especially for ANNs [1, 2, 30].

A measure against overfitting is the aforementioned regularization by adding an extra loss that not depends on the training data. Then an optimal function not only has to fit them well but has also to satisfy additional (global) properties. This results in a trade off between (training) data fitting and generalization, which could become a problem in scenarios, where a good reproduction of the training data (interpolation) as well as a good generalization are simultaneously necessary. The same holds for the bias-variance trade off in general.

It should become clear by now that the choice of an appropriate loss, hypothesis set and regularization is a crucial and highly non-trivial task. Less so if additionally aspects, like time-complexity of the training or integration into a complex system / pipeline have to be taken into account. In general, the information of the problem that is encoded in the data alone is not sufficient to get a good model by a ML-algorithm. A more accurate picture is that this information as well as some pre-knowledge about the system, encoded by these variables, is needed.

In summary, it can be said that no golden way exists leading to a ML-model covering all possible application scenarios. In fact it can, e.g. for classification tasks, be proven that no algorithm exists that works arbitrary well on any training set [30, Theorem 1.13]. As a consequence an extensive collection of protocols, families of methods and rules of good practices exist, that were developed over the last decades and adjusted to different applications. For reasons of comprehensibility, we will only discuss one of them, namely kernel methods, in further detail. For a more detailed introduction to the applied aspects, see e.g. [37, 10, 18].

One family that certainly got much prominence in last years from the academical and business world alike are ANN. They have proven to be exceptional capable for a wide range of application cases and environments. Among other things also for the purpose of PDE evaluation [21, 16, 41, 22, 24], with some success. However, we will not use them for optimization and use kernel methods instead for reason that will become clear later while unrolling the concepts and technicalities in the next chapters.

1.2. Vectorial Kernel Orthogonal Greedy Algorithm

After the abstract introduction of machine learning frameworks and terminology, we now briefly introduce kernel methods for surrogate modelling of functions in a so called Reproducing Kernel Hilbert Space (RKHS) and an algorithm associated to them that fits into Definition 1.1.1, defining a ML-model. This is just one of many different well-studied kernel methods [9, 8, 45, 44], which by themself are merely a fraction of the ML-approaches invented in the last decades. In fact, a quite heterogeneous family of methods and algorithms exist, that have been intensely studied.

Definition 1.2.1 (Reproducing Kernel Hilbert Space; [44, Definitions 15, 16])**.** Let $\mathcal{P}$ be a nonempty subset of $\mathbb{R}^d$ for some $d \in \mathbb{N}$. A *kernel* is a symmetric bivariant function $K : \mathcal{P} \times \mathcal{P} \to \mathbb{R}^{q \times q}$, i.e. $K(x, y) = K(y, x)^T$ for all $x, y \in \mathcal{P}$, with $q \in \mathbb{N}$. For a finite set of distinct points $(x_i)_{i \in \{1,\ldots,m\}} \subset \mathcal{P}^m$, the matrix $A := (K(x_i, x_j))_{(i,j)} \in \mathbb{R}^{qm \times qm}$ is called *kernel matrix*. We say that K is strictly positive definite (SPD), if A is positive definite. A Hilbert space $\mathcal{H}$ of functions $f : \mathcal{P} \to \mathbb{R}^q$ is called Reproducing Kernel Hilbert Space, if a kernel K exists such that

1. $K(\cdot, x)v \in \mathcal{H}$ for all $x \in \mathcal{P}$, $v \in \mathbb{R}^q$, and
2. $\langle f, K(\cdot, x)v\rangle = f(x)^T v$ for all $x \in \mathcal{P}$, $v \in \mathbb{R}^q$ and $f \in \mathcal{H}$.

In this case K is called *reproducing kernel.* A simple way to construct such matrix-valued kernel is to use an one-dimensional kernel $\tilde{K}$, by setting $K(x, y) = \tilde{K}(x, y)\mathrm{Id}_{\mathbb{R}^q}$.

It can be shown that for each SPD kernel K a unique RKHS exists, with K as reproducing kernel [44, Theorem 12].

We now use RKHS for defining a well-suited learning task in the sense of Definition 1.1.1. By reiterating the notation from the last section, we set $\mathcal{X} := \mathcal{P}$ as source set, $\mathcal{Y} := \mathbb{R}^q$ as target set and the hypothesis set as $\mathcal{F} := \mathcal{H}$ as given by a RKHS for some SPD kernel K. An important theoretical result in this context is the *Representer Theorem.*

Theorem 1.2.2 (Representer Theorem; [44, Theorem 17])**.** Let $\mathcal{P} \neq \emptyset$, K a SPD kernel on $\mathcal{P}$ and $\lambda \geq 0$. Further let a finite training data set $z := ((x_1, y_1), \ldots, (x_m, y_m))$ be given, with $x_i \in x$, $y_i \in y$, where $x := (x_i)_{i \in \{1,\ldots,m\}} \subset (\mathcal{P})^m$ consists of pairwise distinct training datapoints and $y := (y_i)_{i \in \{1,\ldots,m\}} \subset (\mathbb{R}^q)^m$. If the loss function is defined by $\mathcal{L}(f, (x_i, y_i)) := \|f(x_i) - y_i\|^2_{\mathbb{R}^q}$ (Mean squared error (MSE)), then the ERM-problem regularized with $\lambda \geq 0$ and the norm of the target function, i.e.

$$\underset{f \in \mathcal{H}}{\operatorname{argmin}} \left(\frac{1}{m} \sum_{i=1}^{m} \|f(x_i) - y_i\|^2_{\mathbb{R}^q} + \lambda \|f\|^2_{\mathcal{H}} \right)$$

has a unique finite dimensional solution

$$\hat{f}_z(\cdot) := \sum_{i=1}^{m} \alpha_i K(\cdot, x_i) \in \operatorname{span}\{vK(\cdot, x_i) \,|\, i \in \{1, \ldots, m\},\, v \in \mathbb{R}^q\} \subset \mathcal{H}$$

for suitable coefficients $\alpha_1, \ldots, \alpha_m \in \mathbb{R}^q$.

Proof. See [44, Theorem 17]. $\square$

The Representer Theorem asserts the existence of an ERM-solution for our learning task. As explained in [44, Corollary 19], for kernels of the form $\tilde{K}(\cdot,\cdot)\mathrm{Id}_{\mathbb{R}^q}$ and the MSE-loss, the coefficients α_i are given as solution to the linear problem

$$(A + \lambda I)\alpha = y, \text{ with } \alpha := (\alpha_1, \ldots, \alpha_m) \text{ and } y := (y_1, \ldots, y_m). \tag{1.4}$$

Note that if $\lambda = 0$ holds, $\hat{f}_z$ is an interpolant of the training data, i.e. $y_i = \hat{f}_z(x_i)$ for $i \in \{1, \ldots, m\}$, enforcing a perfect fit to the data. Is $\lambda > 0$, this restriction is weakened to allow better generalization but the interpolation property gets lost. From this argumentation follows that the ERM-problem has a unique solution, which is accessible by a numerical procedure in finitely many steps, e.g. by matrix-inversion. However, this matrix-inversion would, especially for a large amount of training data, not be feasible, due to performance reasons. To avoid this problem different techniques has been invented. One of them is a greedy-approach implemented by the Vectorial Kernel Orthogonal Greedy Algorithm (VKOGA)

$$\mathcal{A}_{\mathrm{VKOGA}} : \bigcup_{m \in \mathbb{N}} (\mathcal{P} \times \mathbb{R}^q)^m \to \mathcal{H},$$

providing the routine $\texttt{train}_{\mathrm{VKOGA}}[\cdot]$. We will only sketch the idea here, for details we refer to [44].

The key point of the VKOGA is to reduce problem (1.4) by selecting a subset $x_M := (x_i)_{i \in I_M}$ of the training points $(x_i)_{i \in \{1,\ldots,m\}}$ with I_M depending on the data z and to solve the reduced problem

$$(A_M + \lambda I)\alpha_i = y_i, \text{ for all } i \in I_M, \text{ with } A_M := (K(x_i, x_j))_{(i,j) \in I_M \times I_M} \tag{1.5}$$

instead, with $M := |I_M|$, getting an approximative solution

$$s_z(\cdot) := \sum_{i \in I_M} \alpha_i K(\cdot, x_i). \tag{1.6}$$

The set $(x_i)_{i \in I_M}$ is constructed iteratively, by initializing $I_0 := \emptyset$ and successively extending it until the error of $s_z(\cdot)$ with respect to some loss is below a given threshold. The extention is performed by adding one training point x_i per iteration, $I_{M+1} := I_M \cup \{i_{M+1}\}$. This point is selected with respect to some pointwise error measure $\nu_M : \mathcal{P} \to \mathbb{R}_{\geq 0}$ as

$$i_{M+1} := \underset{i \in \{1,\ldots,m\} \setminus I_M}{\arg\max}\ \nu_M(x_i),$$

i.e. it is the point with the greatest remaining error and thus with the greatest potential to improve the model selected. A second important feature of the VKOGA is that solving (1.5) can be performed efficiently by using solutions from previous iterations (see Propositions 24 and 25 and Remark 28 in [44]). This is possible, due to a Cholesky-like decomposition of $A_M + \lambda I$, employing a Newton basis $\{v_j\}_{j\in\{1,\dots,M\}}$ of $\{K_\lambda(\cdot,x_i)\}_{i\in I_M}$, with $K_\lambda(\cdot,x_i) := K(\cdot,x_i) + \lambda\delta_{x_i}(\cdot)$ and

$$\begin{aligned}
v_1(x) &:= \frac{K_\lambda(x,x_{i_1})}{\|K_\lambda(\cdot,x_{i_1})\|_{\mathcal{H}}},\\
\tilde{v}_k(x) &:= K_\lambda(x,x_{i_k}) - \sum_{j=1}^{k-1} v_j(x_{i_k})v_j(x),\\
v_k(x) &:= \frac{\tilde{v}_k(x)}{\|\tilde{v}_k(x)\|_{\mathcal{H}}}, \text{ for } 1 < k \le M.
\end{aligned}$$

The VKOGA-model will build the core of the ML-model used in the numerical experiments from the last chapter. There we will use the P-greedy approach, setting

$$\nu_M(x) := P_M(x) = \left(K_\lambda(x,x) - \sum_{j=1}^{M} v_j(x)^2\right)^{\frac{1}{2}} \quad \text{for all } x \in \mathcal{P}.$$

The following chapters outline how ML-algorithms can be applied to optimization problems with parabolic PDE constraints and discussing their benefits and problems in this context. We start by making a formal problem definition and deriving primal and dual problems.

2. Problem formulation & adjoint approach

2.1. Constrained optimization problem

Consider a bounded domain $\Omega \subset \mathbb{R}^d$ with Lipschitz continuous boundary and a Hilbert space V such that $H_0^1(\Omega) \subset V \subset H^1(\Omega) \subset L^2(\Omega) \subset V'$, where V' denotes the dual space. We are interested in finding an parameter $\mu \in \mathcal{P}$ minimizing an objective (cost) functional of the form

$$\hat{J}(\mu) := \int_0^T \|\mathcal{L}(u(t,\mu)) - g_{\text{ref}}(t)\|_{\mathbb{D}}^2 dt + \lambda \mathcal{R}(\mu), \tag{2.1}$$

where the state $u(\mu) \in L^2(0,T;V)$ is constrained by a parabolic parametric PDE. The parameter domain is a compact finite dimensional set $\mathcal{P} \subset \mathbb{R}^P$ and $T \in (0,\infty)$ is a finite end-time. The functional $\mathcal{L} : V \to \mathbb{D}$ is linear, continuous and maps elements from the solution space V to some suitable feature Hilbert space $\mathbb{D}$ and $g_{\text{ref}} \in L^2(0,T;\mathbb{D})$ a reference value, for example obtained by some physical measurement. The $C^2(\mathcal{P})$-differentiable term $\mathcal{R} : \mathcal{P} \to \mathbb{R}$, weighted by $\lambda \in \mathbb{R}_{\geq 0}$, provides regularization. The parabolic constraint is given by

$$\begin{aligned} \left(\overline{\partial_t u(t,\mu)}, v\right)_{L^2(\Omega)} + a(u(t,\mu), v; \mu) &= b(t) f(v;\mu) \qquad \text{for all } v \in V \text{ and } t \in [0,T], \\ u(0,\mu) &= u_0(\mu) \qquad \text{for } u_0(\mu) \in V, \end{aligned} \tag{2.2}$$

assuming $u(\mu) \in L^2(0,T;V)$ and $\partial_t u(\mu) \in L^2(0,T;V')$, while $\overline{\partial_t u(t,\mu)} \in L^2(\Omega)$ denotes the Riesz-representative of the weak derivative $\partial_t u(t,\mu)$. The parametric operator $f(\cdot\,;\mu) : V \to \mathbb{R}$ on the right hand side is linear, V-continuous for each $\mu \in \mathcal{P}$ and will be multiplied by a time-dependent forcing input $b : [0,T] \to \mathbb{R}$. The elliptic part $a(\cdot,\cdot\,;\mu) : V \times V \to \mathbb{R}$ is modelled by a symmetric, continuous and coercive bilinear form, i.e. for all $\mu \in \mathcal{P}$ holds

$$0 < \gamma_a(\mu) := \sup_{w \in V\setminus\{0\}} \sup_{v \in V\setminus\{0\}} \frac{a(w,v;\mu)}{\|w\|_V \|v\|_V} \leq \gamma_0^a < \infty \text{ and} \tag{2.3}$$

$$0 < \alpha_0^a \leq \alpha(\mu) := \inf_{v \in V\setminus\{0\}} \frac{a(v,v;\mu)}{\|v\|_V^2}. \tag{2.4}$$

This induces an energy product on V by setting $\langle \cdot,\cdot \rangle_V := a(\cdot,\cdot\,;\bar{\mu})$ for a fixed $\bar{\mu} \in \mathcal{P}$. Moreover, we assume that this bilinear form is twice continuously differentiable for all components of μ and that its first derivatives are also bilinear, i.e. $a_{\mu_i} := \partial_{\mu_i} a(\cdot,\cdot\,;\mu) : V \times V \to \mathbb{R}$ is for all $i \in \{1,\dots,P\}$ bilinear and V-continuous such that

$$0 < \gamma_{a_{\mu_i}}(\mu) := \sup_{w \in V\setminus\{0\}} \sup_{v \in V\setminus\{0\}} \frac{a_{\mu_i}(w,v;\mu)}{\|w\|_V \|v\|_V} \leq \gamma_0^{a_{\mu_i}} < \infty. \tag{2.5}$$

Likewise, f is chosen to be twice continuously differentiable in μ, with its first derivatives being also linear and continuous, i.e. $\partial_{\mu_i} f(\cdot\,;\mu) \in V'$ for all $i \in \{1, \dots, P\}$. Additionally, we assume that continuity and coercivity with respect to $\|\cdot\|_V$ is also given for the bilinear form $(\cdot,\cdot)_{L^2(\Omega)}$. Let further the operators a and f be *affinely decomposable*, i.e. they can be split into parameter-depended and parameter-independent components, such that for all $w, v \in V$ and $\mu \in \mathcal{P}$ hold

$$a(w, v; \mu) = \sum_{q=1}^{Q_a} \Theta_a^q(\mu) a^q(w, v) \text{ and } f(v; \mu) = \sum_{\tilde{q}=1}^{Q_f} \Theta_f^{\tilde{q}}(\mu) f^{\tilde{q}}(v), \tag{2.6}$$

where Q_a and Q_f are some integer, the functions $\Theta_a^q, \Theta_f^q : \mathcal{P} \to \mathbb{R}$ twice continuously differentiable and $a^q : V \times V \to \mathbb{R}$, $f^q : V \to \mathbb{R}$ continuous (bi)linear forms, independent of μ.

For the construction of an a posteriori error estimator, we require that the coercivity constant of a can be bounded from below and the continuity constants of the derivatives from above by some positive constants $\alpha_{\mathrm{LB}}(\mu)$ and $\gamma_{a_{\mu_i}}^{UB}(\mu)$, i.e. for all $\mu \in \mathcal{P}$ holds that

$$\begin{gathered} 0 < \alpha_0^a \leq \alpha_{\mathrm{LB}}(\mu) \leq \alpha(\mu) \text{ and} \\ \gamma_{a_{\mu_i}}^{\mathrm{UB}}(\mu) \geq \gamma_{a_{\mu_i}}. \end{gathered}$$

For simplicity we argue in the following with a fixed temporal grid $\mathcal{G}_{\Delta t}^{\mathrm{pr}} := \{0 =: t^0 < \cdots < t^K := T\}$ of $[0, T]$, with $K < \infty$ and $\Delta t := t^{i+1} - t^i$ for all $i \in \{0, \dots, K-1\}$. Furthermore, we demand for the analytic discussion that the functional is zero at the initial value. Therefore, trajectories are considered to be in the Bochner-type Hilbert space

$$Q_{\Delta t}^{\mathrm{pr}}(0, T; V) := \Big\{ f : \mathcal{G}_{\Delta t}^{\mathrm{pr}} \to V \,|\, f(t^0) = 0 \text{ and } \|f\|_{Q_{\Delta t}^{\mathrm{pr}}(0,T;V)} < \infty \Big\} \cong V^K, \tag{2.7}$$

with inner product $\langle f, g \rangle_{Q_{\Delta t}^{\mathrm{pr}}(0,T;V)} := \sum_{k=1}^{K} \langle f(t^k), g(t^k) \rangle_V$ for $f, g \in Q_{\Delta t}^{\mathrm{pr}}(0, T; V)$. This space can be interpreted as a subspace of $L^2(0, T; V)$, its dual space is denoted by $Q_{\Delta t}^{\mathrm{pr}}(0, T; V)'$. For a more convenient notation, we will expand the quadratic term and introduce (bi)linear, continuous operators

$$d(w, v) := (\mathcal{L}(w), \mathcal{L}(v))_{\mathbb{D}} \text{ and } l^k(v) := -2 \Big(\mathcal{L}(v), g_{\mathrm{ref}}(t^k) \Big)_{\mathbb{D}} \text{ for } v, w \in V,$$

with continuity constants γ_d and γ_l. Therefore, the actual optimization problem we are interested in, using the method of lines, can be stated as

Problem 2.1.1. Consider the minimization problem

$$\operatorname*{argmin}_{\mu \in \mathcal{P}} \tilde{J}(u(\mu), \mu), \tag{2.8}$$

such that $u(\mu) := (u^k(\mu))_{k \in \{0,\dots,K\}} \in Q_{\Delta t}^{\mathrm{pr}}(0, T; V)$ satisfying

$$\frac{1}{\Delta t}(u^k(\mu) - u^{k-1}(\mu), v)_{L^2(\Omega)} + a(u^k(\mu), v; \mu) = b(t^k) f(v; \mu) \;\; \forall v \in V, \; k \in \{1, \dots, K\}, \tag{2.9}$$

with initial condition $u^0(\mu) = 0$ and $\tilde{J} : Q^{\mathrm{pr}}_{\Delta t}(0,T;V) \times \mathcal{P} \to \mathbb{R}$ defined by

$$(v,\mu) \mapsto \tilde{J}(v;\mu) := \Delta t \sum_{k=1}^{K} \Big[d(v^k, v^k) + l^k(v^k) + \Big(g_{\mathrm{ref}}(t^k), g_{\mathrm{ref}}(t^k) \Big)_{\mathbb{D}} \Big] + \lambda \mathcal{R}(\mu). \tag{2.10}$$

The linear problem (2.9) will be called the *primal problem.*

2.2. Differentiability & dual problem

Our goal is to solve Problem 2.1.1 with a gradient-based optimization method. To do this in a mathematical strict way, we first have to discuss some properties of this problem. Remark that under the assumptions made in Section 2.1, the linear problem (2.9) has for all $\mu \in \mathcal{P}$ a unique solution in $Q^{\mathrm{pr}}_{\Delta t}(0,T;V)$, which is easy to verify by iteratively applying the Lax-Milgram theorem [13, Theorem 4.3]. Therefore, a well-defined (primal) state-evaluation operator

$$\begin{aligned} A^{\mathrm{pr}} : \mathcal{P} &\to Q^{\mathrm{pr}}_{\Delta t}(0,T;V); \\ \mu &\mapsto u(\mu), \end{aligned} \tag{2.11}$$

returning the solution $u(\mu)$ to (2.9), exists. However, the existence of a unique optimizer for $\mu \mapsto \mathcal{J}(\mu) := \tilde{J}(A^{\mathrm{pr}}(\mu), \mu)$ can not be asserted, since this functional is in general non-convex. In the following we will discuss, if not mentioned otherwise, differentiability as Fréchet differentiability. It is easy to prove that the functional (2.10) is twice continuously (Fréchet) differentiable with respect to both arguments, due to the (bi)linearity of the terms and the C^2-regularity of the regularization. The derivatives at $(v;\mu) \in Q^{\mathrm{pr}}_{\Delta t}(0,T;V) \times \mathcal{P}$ for the respective arguments are linear bounded operator of the form

$$\partial_v \tilde{J}(v;\mu)[\,\cdot\,] \in Q^{\mathrm{pr}}_{\Delta t}(0,T;V)' \text{ and } \partial_\mu \tilde{J}(v;\mu)[\,\cdot\,] \in \left(\mathbb{R}^P\right)'.$$

The derivative along a direction ν in the respective vector space are given by $\partial_v \tilde{J}(v;\mu)[\nu]$ and $\partial_\mu \tilde{J}(v;\mu)[\nu]$. For using classic optimization methods like gradient descent or Broyden, Fletcher, Goldfarb, Shanno (BFGS) as presented e.g. in [51, 29], existence and feasible computational access to the parametric gradient $\nabla_\mu \mathcal{J}$ is required. Its directional derivatives are accessible by the following Lemma.

Lemma 2.2.1. [26, Section 1.6] Let the assumptions from Section 2.1 be in force and let the primal residuum be given as

$$\begin{aligned} r_{\mathrm{pr}} : Q^{\mathrm{pr}}_{\Delta t}(0,T;V) \times \mathcal{P} &\to (V')^K \cong (V^K)'; \\ (u,\mu) =: (u^0, \dots, u^K, \mu) &\mapsto (r^1_{\mathrm{pr}}(u,\cdot\,;\mu), \dots, r^K_{\mathrm{pr}}(u,\cdot\,;\mu)) \end{aligned}$$

with

$$r^k_{\mathrm{pr}}(u,\cdot\,;\mu) := \Delta t b(t^k) f(\,\cdot\,;\mu) - \Delta t a(u^k,\,\cdot\,;\mu) - (u^k - u^{k-1},\,\cdot\,)_{L^2(\Omega)}$$

for $k \in \{1, \dots, K\}$. The residuum r_{pr} and the state-evaluation operator A^{pr} are differentiable. Moreover, are the derivatives of $d_\nu A^{\mathrm{pr}}(\mu) := \partial_\mu A^{\mathrm{pr}}(\mu))[\nu]$ in direction $\nu \in \mathbb{R}^P$ given by a solution of the linear problem

$$\partial_u r_{\mathrm{pr}}(A^{\mathrm{pr}}(\mu);\mu)[d_\nu A^{\mathrm{pr}}(\mu)][v] = \partial_\mu r_{\mathrm{pr}}(A^{\mathrm{pr}}(\mu);\mu)[\nu][v]. \tag{2.12}$$

This problem has a unique solution.

Proof. The differentiability of r^{pr} follows directly from the assumptions made on $a(\cdot,\cdot;\mu)$ and $f(\cdot;\mu)$. For the evaluation operator (2.11) this follows from applying the implicit function theorem [26, Theorem 1.45]. For details we refer to the argumentation in [26, Section 1.6]. The existence of a solution to (2.12) is asserted by the Lax-Milgram theorem. □

The derivative of $\mathcal{J}(\mu)$ can for all $\nu \in \mathbb{R}^P$ be calculated, using the chain rule, getting

$$\begin{aligned}\partial_\mu \mathcal{J}(\mu)[\nu] &= \partial_u \tilde{J}(A^{\mathrm{pr}}(\mu),\mu)[\partial_\mu A^{\mathrm{pr}}(\mu)][\nu] + \partial_\mu \tilde{J}(A^{\mathrm{pr}}(\mu),\mu)[\nu] \\ &= (\partial_\mu A^{\mathrm{pr}}(\mu))'[\partial_u \tilde{J}(A^{\mathrm{pr}}(\mu),\mu)][\nu] + \partial_\mu \tilde{J}(A^{\mathrm{pr}}(\mu),\mu)[\nu].\end{aligned} \tag{2.13}$$

The operator $(\partial_\mu A^{\mathrm{pr}}(\mu))' : Q^{\mathrm{pr}}_{\Delta t}(0,T;V)' \to \left(\mathbb{R}^P\right)'$ denotes the dual operator to $\partial_\mu A^{\mathrm{pr}}(\mu)$ who can be obtained by solving the linear problem (2.12) for $\nu := e_1,\dots,e_P$, with e_i being the canonical basis vectors for $\mathbb{R}^P$. Solving (2.12) P-times is a major bottleneck for the calculation. Fortunately, can these calculations be carried out in a far more efficient way [6, 26]. From inserting (2.12) to (2.13) follows

$$(\partial_\mu A^{\mathrm{pr}}(\mu))'[\partial_u \tilde{J}(A^{\mathrm{pr}}(\mu),\mu)][\nu] = (\partial_\mu r_{\mathrm{pr}}(A^{\mathrm{pr}}(\mu);\mu))'[p(\mu)][\nu]$$

for all $\nu \in \mathbb{R}^P$, where $p(\mu) \in ((V')^K)' \cong V^K$ is called the *dual solution* to (2.9), set as solution of the *adjoint or dual problem*

$$(\partial_u r_{\mathrm{pr}}(A^{\mathrm{pr}}(\mu);\mu))'[p(\mu)][v] = -\partial_u \tilde{J}(A^{\mathrm{pr}}(\mu),\mu)[v] \;\; \forall v \in Q^{\mathrm{pr}}_{\Delta t}(0,T;V). \tag{2.14}$$

Remark that V^K is equipped with an inner product making it homeomorph to $Q^{\mathrm{pr}}_{\Delta t}(0,T;V)$. Therefore, the solution step can be reduced to solving the linear problem (2.14). In the situation presented here, we can reformulate this problem to the initial value problem

$$\begin{gathered}\frac{1}{\Delta t}(v, p^k(\mu) - p^{k+1}(\mu))_{L^2(\Omega)} + a(v, p^k(\mu);\mu) = 2d(u^k(\mu), v) + l^k(v), \\ p^{K+1}(\mu) = 0,\end{gathered} \tag{2.15}$$

for all $k \in \{1,\dots,K\}$ and $v \in V$. The gradient of $\mathcal{J}$ is therefore given by

$$\frac{\partial \mathcal{J}(\mu)}{\partial \mu_i} = \frac{\partial \tilde{J}(u(\mu),\mu)}{\partial \mu_i} = \Delta t \sum_{k=1}^{K} \Big[b(t^k) f_{\mu_i}(p^k(\mu);\mu) - a_{\mu_i}(u^k(\mu), p^k(\mu);\mu) \Big] + \lambda \frac{\partial \mathcal{R}(\mu)}{\partial \mu_i},$$

for $i \in \{1,\dots,P\}$. According to the Lax-Milgram theorem the dual problem (2.14) has a unique solution. This allows us to define the dual state-evaluation operator

$$\begin{aligned} A^{\mathrm{du}} : \mathcal{P} &\to Q^{\mathrm{du}}_{\Delta t}(0,T;V); \\ \mu &\mapsto p(\mu), \end{aligned}$$

where $p(\mu)$ is the solution to (2.15). This operator is well-defined and, by reiterating the arguments in Theorem 2.2.1, differentiable. Note that, in contrast to the primal case, the dual trajectory $p(\mu) := (p^k(\mu))_{k\in\{1,\dots,K+1\}}$ evolves backwards in time. We therefore define the Bochner-type space $Q^{\mathrm{du}}_{\Delta t}(0,T;V)$ similar to $Q^{\mathrm{pr}}_{\Delta t}(0,T;V)$ in (2.7) as

$$Q^{\mathrm{du}}_{\Delta t}(0,T;V) := \Big\{ f : \mathcal{G}^{\mathrm{du}}_{\Delta t} \to V \mid f(t^{K+1}) = 0 \text{ and } \|f\|_{Q^{\mathrm{du}}_{\Delta t}(0,T;V)} < \infty \Big\} \cong V^K,$$

with the fixed temporal grid $\mathcal{G}^{\mathrm{du}}_{\Delta t} := \{\Delta t =: t^1 < \cdots < t^{K+1} := T + \Delta t\}$. The inner product is defined analogue to $Q^{\mathrm{pr}}_{\Delta t}(0,T;V)$. It is easy to see, due to the continuity of the solution map, that $p(\mu) \in Q^{\mathrm{du}}_{\Delta t}(0,T;V)$.

We define an operator returning an estimate of the gradient for given primal and dual trajectories

$$\begin{aligned} \nabla_\mu \tilde{J} : Q^{\mathrm{pr}}_{\Delta t}(0,T;V) \times Q^{\mathrm{du}}_{\Delta t}(0,T;V) \times \mathcal{P} &\to \mathbb{R}^P \\ (v,w,\mu) \mapsto (\nabla_\mu \tilde{J})_i(v,w;\mu) := \Delta t \sum_{k=1}^{K} \Big[b(t^k) f_{\mu_i}(w^k;\mu) - a_{\mu_i}(v^k,w^k;\mu) \Big] &+ \lambda \frac{\partial \mathcal{R}(\mu)}{\partial \mu_i} \end{aligned} \tag{2.16}$$

for $i \in \{1,\dots,P\}$.

In a case where higher order derivatives of $\mathcal{J}(\mu)$ are required, additional calculations have to be carried out. For our purposes it is enough to mention, that the Hessian $\mathcal{H}$ of $\mathcal{J}(\mu)$ exists, we will use it in the BFGS-Algorithm, which requires existence but not direct access of $\mathcal{H}$.

The discussion above reveals that the objective functional and its gradient are completely determined by the primal and dual problem. For a gradient-based optimization we therefore need to solve both problems along the decay trajectory multiple times for different parameter. To perform accurate numerical calculations, spatial discretization will be employed, approximating them by finite dimensional problems. However, for maintaining sufficient precision for all parameters evaluated, the dimension of these *high fidelity model*s has usually to be so high that the computational effort raises to uneconomical expense, prohibiting efficient calculations. To address this issue different Model Order Reduction methods can be applied, to reduce the computational cost. A crucial aspect thereby is to assert that the approximation error stays in an acceptable range, for all parameters along the trajectory and simultaneously keeping the complexity of the reduced model manageable. Trust Region-methods are a suitable choice, for this purpose. They can achieve both targets by iteratively updating the reduced models if the error becomes untenable. In the next Chapter such a TR-method, inspired by [51, 39], will be introduced, using a variant of the BFGS-Algorithm for optimization. We assume there for the moment that high fidelity and reduced order models for $\tilde{J}$ and its gradient are available, fitting into the requirements of the Trust Region-method. Their construction and the integration of ML-algorithms into them will be discussed in Chapters 4 to 6.

3. Trust region algorithm

3.1. Trust region based optimization

Assume that we have a sufficiently smooth function $\hat{J} : \mathcal{P} \to \mathbb{R}$ on a finite dimensional domain $\mathcal{P} \subset \mathbb{R}^P$ and a *high fidelity approximation* J to this and its derivative $\nabla_\mu J$ given, that are accessible by some *high fidelity model* M. Our goal is to find a global optimizer

$$\operatorname*{argmin}_{\mu \in \mathcal{P}} J(\mu).$$

For the TR-approach, an initial guess $\mu^{(0)}$ is chosen and a *local model* $M^{(0)}$ is build, providing access to (local) approximations $J^{(0)}$, $\nabla_\mu J^{(0)}$ near $\mu^{(0)}$, for example by Taylor expansion. Such models can usually only be trusted in some region $T^{(0)} \subset \mathcal{P}$, with $\mu^{(0)} \in T^{(0)}$, i.e. the approximation error is only in $T^{(0)}$ acceptable small. Thus, the optimization problem can be reasonable solved only in $T^{(0)}$, returning an local optimizer $\mu^{(1)}$. This solution is then used to build a new model $M^{(1)}$ with a new trust region $T^{(1)}$. In this iterative fashion a sequence of subproblems

$$\operatorname*{argmin}_{\mu \in \mathcal{P}} J^{(i)}(\mu) \text{ s.t. } \mu \in T^{(i)} \tag{3.1}$$

emerges. Usually the trust regions are controlled by some tolerance $\epsilon^{(i)}$ limiting the error of the models, e.g. by euclidean distance to the development point $T^{(i)} := \{\mu \in \mathcal{P} \mid \|\mu^{(i)} - \mu\| \leq \epsilon^{(i)}\}$. The truncation of $\mathcal{P}$ by $T^{(i)}$ does not necessarily guarantees sufficient accuracy within the whole trust region. Therefore, for each solution $\mu^{(i+1)}$ of (3.1) must be decided if it can be trusted. A common technique for this is determining the trustworthiness

$$\rho^{(i)} = \frac{J(\mu^{(i)}) - J(\mu^{(i+1)})}{J^{(i)}(\mu^{(i)}) - J^{(i)}(\mu^{(i+1)})}. \tag{3.2}$$

Based on this ratio, the guess $\mu^{(i+1)}$ will either be accepted or rejected else. If $\mu^{(i+1)}$ is accepted, $\epsilon^{(i+1)}$ will be kept or enlarged compared to $\epsilon^{(i)}$ and $\mu^{(i+1)}$ will be used as the starting point for the $(i+1)$-th subproblem. If on the other hand $\mu^{(i+1)}$ is rejected, $\epsilon^{(i)}$ will be reduced and the i-th problem be solved again with the updated trust region, until the solution $\mu^{(i+1)}$ is acceptable. The algorithm stops if some global termination criteria is reached.

For this algorithm to work properly, it is necessary that the approximation error in the trust region approaches zero if the the trust region is reduced to $\{\mu^{(i)}\}$. Furthermore, we want to avoid direct evaluations of M. These high fidelity evaluations are usually too expensive and will slow down the algorithm significantly. For these reasons, the *relaxed first-order condition* is introduced formalizing the error conditions surrogate models have to meet and providing the basis to replace (3.2) with an efficient error calculation.

Condition 3.1.1 (relaxed first-order condition)**.**

1. Error bounds $\Delta^{J,(i)}(\cdot)$ and $\Delta^{\nabla J,(i)}(\cdot)$ are available for both $J^{(i)}(\cdot)$ and $\nabla_\mu J^{(i)}(\cdot)$, i.e. for $\mu \in \mathcal{P}$ holds

$$\left|J^{(i)}(\mu) - J(\mu)\right| \leq \Delta^{J,(i)}(\mu) \text{ and } \|\nabla_\mu J^{(i)}(\mu) - \nabla_\mu J(\mu)\| \leq \Delta^{\nabla J,(i)}(\mu). \tag{3.3}$$

   Furthermore, let these bounds be *online-efficent*, i.e. their calculation is cheap for new μ, at least after an initialisation procedure.

2. For any given $\epsilon^{(i)} > 0$ and $\mu^{(i)}$ a surrogate model $M^{(i)}$ can be found such that

$$\frac{\Delta^{J,(i)}(\mu^{(i)})}{\left|J^{(i)}(\mu^{(i)})\right|} < \epsilon^{(i)} \text{ and } \frac{\Delta^{\nabla J,(i)}(\mu^{(i)})}{\|\nabla_\mu J^{(i)}(\mu^{(i)})\|} < \epsilon^{(i)}. \tag{3.4}$$

   I.e. the models can be *infinitely refined.*

We say a model $M^{(i)}$ satisfies this condition if the first condition and (3.4) hold, for a given $\epsilon^{(i)} > 0$.

The Problem (3.1) is usually be solved by iteratively determining a descendent trajectory $(\mu^{(i,l)})_{l\in\{0,\dots,L^{(i)}\}}$ minimizing $J^{(i)}(\mu^{(i,l)})$ and starting at $\mu^{(i,0)} := \mu^{(i)}$. The index i refers to the subproblem solved (outer iteration) and l to the inner iteration. The iterative procedure is stopped if a sufficient termination criteria is satisfied. Then the last iteration point $\mu^{(i+1)} := \mu^{(i,L^{(i)})}$ is returned as solution of the i-th sub problem.

It is important to point out here that solving the subproblems is not sufficient to assert a decrease of the objective functional, because the models might be untrustworthy at the edge of the trust region. To assert decay (or at least stagnation) along the solution trajectory and to replace condition (3.2) the *error-aware sufficient decrease condition*

$$J^{(i+1)}(\mu^{(i+1)}) \leq J^{(i)}(\mu^{(i)}_{\text{AGC}}) \tag{EASDC$^{(i)}$}$$

is introduced. $\mu^{(i)}_{\text{AGC}}$ denotes the *approximate generalized Cauchy point* (AGP), set as the result of the first optimizing step, i.e. $\mu^{(i)}_{\text{AGC}} := \mu^{(i,1)}$. If condition (EASDC$^{(i)}$) is satisfied, the sequence $(J^{(i)}(\mu^{(i)}))_{i\in\mathbb{N}_0}$ is decreasing and therefore the solution $\mu^{(i+1)}$ will be accepted. Otherwise, the trust region is reduced and the problem solved again, as outlined above.

In (EASDC$^{(i)}$), the model $J^{(i+1)}$ is constructed using $\mu^{(i+1)}$. However, this is not optimal, because the solution might be rejected and the model $J^{(i+1)}$ will therefore not be used for the next subproblem. To circumvent this, a sufficient and necessary condition for (EASDC$^{(i)}$) can be given, both depending only on the i-th models and $\mu^{(i+1)}$. It is easy to verify (see [51, Lemma 3.5]) that a suitable sufficient condition is

$$J^{(i)}(\mu^{(i+1)}) + \Delta^{J,(i)}(\mu^{(i+1)}) < J^{(i)}(\mu^{(i)}_{\text{AGC}}) \tag{SC$^{(i)}$}$$

and

$$J^{(i)}(\mu^{(i+1)}) - \Delta^{J,(i)}(\mu^{(i+1)}) \leq J^{(i)}(\mu^{(i)}_{\text{AGC}}) \tag{NC$^{(i)}$}$$

is a necessary one. In this way, an algorithm can be formulated first checking those two conditions, accepting $\mu^{(i+1)}$ if it satisfies ($\mathrm{SC}^{(i)}$) and rejecting if it does not satisfy ($\mathrm{NC}^{(i)}$). In cases where neither condition holds, ($\mathrm{EASDC}^{(i)}$) have to be checked directly by updating the surrogates at $\mu^{(i+1)}$. We want to use in the following a trust region defined by the relative error of the approximative objective functional $J^{(i)}$, assuming that the (3.3) and (3.4) hold for the models used. Therefore, is $T^{(i)}$ given by

$$T^{(i)} := \{\mu \in \mathcal{P} \,|\, b^{(i)} := \Delta^{J,(i)}(\mu)/J^{(i)}(\mu) \leq \epsilon^{(i)}\}. \tag{3.5}$$

Is $\Delta^{J,(i)}(\mu) \leq \epsilon_{\mathrm{m}}$ we set $b^{(i)} = 0$. Is on the other hand $J^{(i)}(\mu) \leq \epsilon_{\mathrm{m}}$ and $\Delta^{J,(i)}(\mu) > \epsilon_{\mathrm{m}}$ we set $b^{(i)} = \infty$, where ϵ_{m} is the machine precision $2.22 \cdot 10^{-16}$. Following the trust region scheme, we either shrink the trust region, setting $\epsilon^{(i)} = \kappa_{tr}\epsilon^{(i)}$ for some $\kappa_{tr} \in (0, 1)$ or start solving the $(i + 1)$-th subproblem (3.1) at $\mu^{(i+1)}$. These steps are summarized by Algorithm 1.

Algorithm 1 Trust region optimization [51, 39]

```
Choose $\epsilon_L > 0, \kappa_{tr} \in (0,1)$, $\mu^{(0)} \in \mathcal{P}$ and a global_termination_criteria.
Set $i := 0$, $\epsilon^{(0)} := \epsilon_L$.
Construct $M^{(0)}$, satisfying condition 3.1.1.
while global_termination_criteria is not satisfied or $i = 0$ do
    Compute $\mu^{(i+1)}$ as solution of (3.1) with a convenient termination criteria.
    if the sufficient condition ($\mathrm{SC}^{(i)}$) holds then
        Accept $\mu^{(i+1)}$.
        Use $\mu^{(i+1)}$ to update the model $M^{(i)}$, such that condition 3.1.1 holds for $M^{(i+1)}$.
    else if condition ($\mathrm{NC}^{(i)}$) fails then
        Reject $\mu^{(i+1)}$ and update $\epsilon^{(i)}$ by $\kappa_{tr}\epsilon^{(i)}$.
    else
        Use $\mu^{(i+1)}$ to update the model $M^{(i)}$, such that condition 3.1.1 holds for $M^{(i+1)}$.
        if ($\mathrm{EASDC}^{(i)}$) holds then
            Accept $\mu^{(i+1)}$.
        else
            Reject $\mu^{(i+1)}$ and update $\epsilon^{(i)}$ by $\kappa_{tr}\epsilon^{(i)}$.
        end if
    end if
    $i \leftarrow i + 1$
end while
```

3.2. BFGS-TR-Algorithm

We will now specify the algorithm from the last section, by employing a variant of the BFGS-algorithm to solve the subproblems (3.1) and discuss some aspects regarding convergence and termination. First we restrict ourselves to problems with a simple bounded parameter domain $\mathcal{P}$. A domain $\mathcal{P}$ is simple bounded if it is rectangular, i.e. given by

$$\mathcal{P} := \{x \in \mathbb{R}^P \,|\, L_j \leq x_j \leq U_j,\ j = 1, \dots, P\},$$

where $-\infty < L_j < U_j < \infty$. We say the j-th constraint is *active* if $(x)_j = L_j$ or $(x)_j = U_j$ and *inactive* else for $x \in \mathcal{P}$. The sets of (in)active indices will be denoted by $\mathcal{I}(x)$ and $\mathcal{A}(x)$. We define the operator $P : \mathbb{R}^P \to \mathcal{P}$, mapping a potential overshot back into the domain by

$$P(\mu)_j := \begin{cases} L_j, & \text{if } (\mu)_j \leq L_j \\ (\mu)_j, & \text{if } L_j \leq (\mu)_j \leq U_j \\ U_j, & \text{if } (\mu)_j \geq U_j \end{cases}$$

and the operator $P_{\mathcal{S}} : \mathbb{R}^P \to \mathbb{R}^P$ for $\mathcal{S} \subset \{1, \ldots, P\}$ by

$$P_{\mathcal{S}}(\mu)_j := \begin{cases} (x)_j, & \text{if } j \in \mathcal{S} \\ 0, & \text{if } j \notin \mathcal{S}. \end{cases}$$

The optimization itself is, inspired by [28, 29], performed with a projected BFGS-algorithm (pBFGS), a variant of the standard BFGS-algorithm for simple bounded domains, returning an optimizing sequence $\{\mu^{(i,l)}\}_{l=0}^{L^{(i)}}$ for each subproblem (3.1), starting at $\mu^{(i,0)}$. For $l \in \mathbb{N}_0$ the parameter $\mu^{(i,l+1)}$ is chosen by going along a search direction d, beginning at $\mu^{(i,l)} + d$, and reducing the distance to $\mu^{(i,l)}$ until a termination criteria is met. This technique is called *backtracking.*

As a quasi-Newton method, the search direction in the pBFGS-method is determined for each $\mu^{(i,l)}$ based on an approximation $\tilde{\mathcal{H}}^{(i,l)}$ of the inverse Hessian $(\mathcal{H}_J(\mu^{(i,l)}))^{-1}$ of J. We follow the algorithm presented in [29, Section 5.5.3], constructing $\tilde{\mathcal{H}}^{(i,l)}$ iteratively, starting with $\tilde{\mathcal{H}}^{(i,0)} := \mathrm{Id}_{\mathbb{R}^P}$ and $\mu^{(i,0)} := \mu^{(i)}$ setting for $l \in \mathbb{N}_0$

$$\begin{aligned} y^{\#} &:= P_{\mathcal{I}(\mu^{(i,l+1)})}\left(\nabla_\mu J^{(i)}(\mu^{(i,l+1)}) - \nabla_\mu J^{(i)}(\mu^{(i,l)})\right), \\ s^{\#} &:= P_{\mathcal{I}(\mu^{(i,l+1)})}\left(\mu^{(i,l+1)} - \mu^{(i,l)}\right). \end{aligned} \tag{3.6}$$

The update step is performed by

$$\tilde{\mathcal{H}}^{(i,l)} := \begin{cases} P_{\mathcal{A}(\mu^{(i,l)})} + A^{(i,l)}, & \text{if } {y^{\#}}^T s^{\#} \geq \epsilon_{\mathrm{m}} \\ \mathrm{Id}_{\mathbb{R}^P}, & \text{else} \end{cases} \tag{3.7}$$

for $l \in \mathbb{N}$. The matrix $A^{(i,l)}$ is constructed by

$$A^{(i,l+1)} := \left(\mathrm{Id}_{\mathbb{R}^P} - \frac{s^{\#}{y^{\#}}^T}{{y^{\#}}^T s^{\#}}\right) P_{\mathcal{I}(\mu^{(i,l+1)})} A^{(i,l)} P_{\mathcal{I}(\mu^{(i,l+1)})} \left(\mathrm{Id}_{\mathbb{R}^P} - \frac{y^{\#}{s^{\#}}^T}{{y^{\#}}^T s^{\#}}\right) + \frac{s^{\#}{s^{\#}}^T}{{y^{\#}}^T s^{\#}}. \tag{3.8}$$

The search direction is set as

$$d^{(i,l)} := \begin{cases} -\tilde{\mathcal{H}}^{(i,l)} \nabla_\mu J^{(i)}(\mu^{(i,l)}), & \text{if } -\nabla_\mu J^{(i)}(\mu^{(i,l)})^T \tilde{\mathcal{H}}^{(i,l)} \nabla_\mu J^{(i)}(\mu^{(i,l)}) < 0 \\ -\nabla_\mu J^{(i)}(\mu^{(i,l)}), & \text{else} \end{cases} \tag{3.9}$$

asserting $\nabla_\mu J^{(i)}(\mu^{(i,l)})^T d^{(i,l)} < 0$ if $\nabla_\mu J^{(i)}(\mu^{(i,l)}) \neq 0$. Does the later case in (3.9) occur, $\tilde{\mathcal{H}}^{(i,l)}$

is reset to $\mathrm{Id}_{\mathbb{R}^P}$. The minimizing sequence is obtained by starting at $\mu^{(i,0)} := \mu^{(i)}$ and updating by

$$\mu^{(i,l)}(k) := P\left(\mu^{(i,l)} + \alpha_0 \eta^k \tilde{d}^{(i,l)}\right) \text{ for a } k \in \mathbb{N}_0, \tag{3.10}$$

where $\tilde{d}^{(i,l)}$ denotes the normalized search direction if $\|d^{(i,l)}\| \neq 0$ and zero else. $\alpha_0 > 0$ is the initial stepsize and $\eta \in (0,1)$ a factor controlling the speed of the backtracking procedure. The exponent k is increased, enforcing an Armijo-type decrease condition and preventing violations of the trust region. Precisely, k is chosen as the first natural number (starting from zero) such that the both conditions

$$J^{(i)}(\mu^{(i,l)}) - J^{(i)}(\mu^{(i,l)}(k)) \geq \frac{\alpha_{\mathrm{arm}}}{\alpha_0 \eta^k} \|\mu^{(i,l)} - \mu^{(i,l)}(k)\|^2 \text{ and} \tag{3.11}$$

$$\mu^{(i,l)}(k) \in T^{(i)} \tag{3.12}$$

hold, with $\alpha_{\mathrm{arm}} = 10^{-7}$. The next parameter is then set as $\mu^{(i,l+1)} := \mu^{(i,l)}(k)$. Condition (3.11) indeed asserts, if satisfied, a decay or stagnation of $J^{(i)}$. Using the argumentation in [29, Theorem 5.4.5], we can state the following Lemma regarding the termination of the backtracking procedure.

Lemma 3.2.1. Let $\nabla_\mu J$ be Lipschitz continuous with respect to μ and $i \in \mathbb{N}_0$ be chosen arbitrarily. Is $\mu^{(i,0)} \in \mathcal{P}$ an inner point of $T^{(i)}$, then holds that a $k \in \mathbb{N}_0$ exists such that $\mu^{(i,0)}(k)$ satisfies (3.11) and (3.12).

Proof. See [25, Theorem 5.4.5] and [28, Lemma 3.2]. □

From this follows that the first backtracking procedure in a subproblem terminates. However, termination is not guaranteed in general. Therefore, the optimization might stagnate at a parameter for that no sufficient criteria of a first order critical point is met, because the algorithm is trapped in an infinite loop. Moreover, neither is $J^{(i)}(\mu^{(i)}_{\mathrm{AGC}}) < J^{(i)}(\mu^{(i)})$ nor $J^{(i+1)}(\mu^{(i+1)}) \leq J^{(i)}(\mu^{(i)}_{\mathrm{AGC}})$ asserted. So, convergence to an optimum is not guaranteed by Algorithm 1.

With the trust-region as defined in (3.5), the BFGS-iteration terminates and returns $\mu^{(i+1)} := \mu^{(i,l)}$ if either

$$\|\mu^{(i,l)} - P_{\mathcal{P}}(\mu^{(i,l)} - \nabla_\mu J^{(i)}(\mu^{(i,l)}))\| \leq \tau_{\mathrm{sub}} \tag{3.13}$$

for some $\tau_{\mathrm{sub}} \in (0,1)$ or

$$\beta \epsilon^{(i)} \leq b^{(i)} = \frac{\Delta^{J,(i)}(\mu^{(i,l)})}{\left|J^{(i)}(\mu^{(i,l)})\right|} \leq \epsilon^{(i)} \tag{3.14}$$

for some $\beta > 0$ is satisfied. Analogue to above, we set $b^{(i)} = \infty$ if $J^{(i)}(\mu) \leq \epsilon_{\mathrm{m}}$ and $\Delta^{J,(i)}(\mu) > \epsilon_{\mathrm{m}}$, and $b^{(i)} = 0$ if $\Delta^{J,(i)}(\mu) \leq \epsilon_{\mathrm{m}}$. The criteria (3.13) and (3.14) are analogue to the ones used in [51, 28, 39]. Condition (3.13) is the usual criteria for an local optimum in a setting with simple bounded domain. On the other hand, (3.14) stops the iteration if $\mu^{(i,l)}$ gets too close to the boundary of $T^{(i)}$. If one of these conditions is met, the algorithm returns to the main loop (Algorithm 1) and either updates the model and keeps the trust region or rejects the result and enacts recalculation. Remark that in general a termination is not asserted.

A reasonable `global_termination_criteria` for Algorithm 1 is given by

$$\|\mu^{(i+1)} - P_{\mathcal{P}}(\mu^{(i+1)} - \nabla_\mu J^{(i+1)}(\mu^{(i+1)}))\| + \Delta^{\nabla J,(i+1)}(\mu^{(i+1)}) \leq \tau, \tag{3.15}$$

where $0 < \tau_{\text{sub}} \leq \tau$ and $\mu^{(i+1)}$ is the solution to the i-th subproblem.

To run Algorithm 1 with the BFGS-solver the abstract notion of the models $M^{(i)}$ have to be filled with a concert implementation, satisfying Condition 3.1.1. The next chapters outline how such models can be build for the objective (2.10), by solving the primal and dual problem. In particular, will the integration of machine learning algorithms be discussed.

4. Adaptive hierarchical framework

4.1. General considerations

As outlined in Section 1.1, ML-models are, at least in theory, capable of providing sufficient approximations for a wide variety of functions. There is justified hope that these models can be used for the solving of parametrized PDEs by learning the solution for a given parameter. Indeed different algorithm designs including ML-components have been tested for this purpose with success [34, 20, 47, 27, 12, 17, 4, 46, 24, 21, 16, 41, 22]. A usual design pattern in literature is a purely data driven approach. Hereby, the ML-model is build with a generic hypothesis set, with the physical information completely encoded in the training data. Our goal is to use this and integrate ML-methods into a MOR-pipeline, by learning approximative solutions from the results provided by other models.

The main benefit from using the ML-based approaches compared to classic algorithms is that these have the potential for *faster evaluation*, thus making the evaluation of large parameter batches more feasible, which is crucial in tasks like Problem 2.1.1. Such speed ups can for example result from a lower computational complexity of the ML-models or because the algorithms can be build in a more efficient way. One example for this could be parallelization, by performing the calculation of $u(\mu)$ for each timepoint simultaneously. The gain that actually can be achieved however depends significantly on the ML-algorithm used and the overhead created by substituting a classical model with an ML-based.

Despite these promising prospects, ML-surrogates come with some technical difficulties. Among other things, these are *training efficiency, error management and certification.* While the evaluation is often quiet fast, the generation of appropriate training data as well as the training itself are often bottlenecks, reducing efficiency. Training requires a sufficient amount of training data $(\mu, u(\mu))$ that are typically generated by the a classic algorithm to get the solution $u(\mu)$ of the PDE-constraint. This can take significant computational effort.

A second problem is the quality of the approximation attained by the ML-models. Although there is theoretical justification [31] showing that such models can act as good estimators, the approximation results in real world implementations often have significant errors. In order to be able to work with them nevertheless sophisticated algorithms or data optimization have to be applied for obtaining an error in an acceptable range is required. Sometimes even more problematic, for example for a TR-approach, is that for many of such models no rigours a posteriori error estimation exists. We say in such cases the models are *not certified.*

In this chapter, we will present a model framework, introduced in [20]. This framework makes it possible to feature different models for solving the primal and dual problems. In particular, a wide range of ML-algorithms can be used to define appropriate models, as long as some technical requirements are met. Moreover, this framework allows to formulate a fallback procedure, choosing for each new parameter the most efficient yet accurate model to solve these problems and adaptively updating the others. With this procedure the mentioned issues of training data generation and certification can be handled in an efficient general way. The concepts and notations established here will be used in Section 6.2 to define the models M and $M^{(i)}$ used in the Trust Region-algorithm.

4.2. Model framework

We will present first the abstract definition of the models as introduced in [20]. The notation matches the situation of the primal problem but the arguments can be made analogously for the dual setting or other problems, as long as the abstract definitions are satisfied. For a parameter domain $\mathcal{P} \subset \mathbb{R}^P$ and a finite end-time $T > 0$, we are interested in efficiently and accurately approximating some state $u(\mu) \in L^2(0,T;V)$, that solves

$$M_\mu u(\mu) = f_\mu, \text{ with } \mu \in \mathcal{P},\ M_\mu : L^2(0,T;V) \to W \text{ and } f_\mu \in W, \tag{4.1}$$

where V and W are real-valued Hilbert spaces. We define the *abstract state-evaluation operator*, returning a solution to (4.1) for a given $\mu \in \mathcal{P}$ by

$$\begin{aligned} A : \mathcal{P} &\to L^2(0,T;V); \\ \mu &\mapsto u(\mu). \end{aligned}$$

As mentioned in Section 2.1, we focus on fixed temporal discritization, therefore we estimate the trajectories in $Q^{\text{pr}}_{\Delta t}(0,T;V) \subset L^2(0,T;V)$. Specifying the minimal mathematical and algorithmic requirements to provide a solution to (4.1) leads to the abstract definition of a *state-based model.* This will be the base definition from which all models will emerge from. The index $*$ is a placeholder distinguishing different models.

Definition 4.2.1 (State-based model)**.** A state-based model $M_* := \text{MODEL}[V_*, A_*]$ of order N_* is given by

(i) a Hilbert space $(V_*, (\cdot,\cdot)_{V_*})$ with $V_* \subset V$ and $N_* := \dim V_*$ and

(ii) an approximate state-evalution operator $A_* : \mathcal{P} \to Q^{\text{pr}}_{\Delta t}(0,T;V_*),\ \mu \mapsto u_*(\mu)$, approximating $A(\mu)$.

For algorithmic use, M_* has the routine

(iii) $u_*(\mu) \leftarrow M_*$`.eval_state`$[\mu]$, returning $A_*(\mu)$.

As mentioned above, high fidelity models give accurate approximations but are usually too costly for many real world applications. To give a more formal definition, we say a state-based model $M_h := \text{MODEL}[V_h, A_h]$ is a Full Order Model (FOM) if the following assumptions hold.

Assumption 4.2.2 (Full Order Model (FOM))**.**

(i) For a (possible unknown) subset of the parameter domain $\tilde{\mathcal{P}} \subset \mathcal{P}$ the approximative solutions $u_h(\mu)$ of (4.1), given by M_h`.eval_state`$[\mu]$, are sufficiently accurate for all $\mu \in \tilde{\mathcal{P}}$.

(ii) The evaluation of M_h, by calling M_*`.eval_state`$[\mu]$, is too costly for our application, with respect to the time- and/or computational constrains given to us.

To address the issue that follows from the second assumption, different methods exist to reduce the evaluation effort. Noteworthy here are Reduced Basis-methods [40], constructing a subspace V_* to V_h by determining a *reduced basis* and projecting the problem on them. Inspired by this, a definition of a Reduced Order Model (ROM) can be stated, generalizing this procedure. Additionally, an *a posteriori error estimate operator* E is introduced, quantifying the approximation error made by projecting the problem.

Definition 4.2.3 (Reduced Order Model (ROM))**.** Let M_h be a FOM as outlined in Assumption 4.2.2 and let further

(i) a subspace $V_* \subset V_h$, called *reduced state-space*

(ii) and accordingly a *reduced state-evaluation* $A_* : \mathcal{P} \to Q^{\mathrm{pr}}_{\Delta t}(0, T; V_*)$

be given, such that $M_* := \mathrm{MODEL}[V_*, A_*]$ is a state-based model. This model is called *reduced order model of order* $N_* := \dim V_*$, *approximating* M_h, abbreviated by $M_* = \mathrm{ROM}\,[V_*, A_*]$. We call a $\mathrm{ROM}\,[V_*, A_*]$ *certified* if we can provide a *state-error estimate operator*

$$E_* : Q^{\mathrm{pr}}_{\Delta t}(0, T; V_*) \times \mathcal{P} \to \mathbb{R},$$

such that

$$\|u_h(\mu) - v_*\| \leq E_*(v_*, \mu) \text{ for all } v_* \in Q^{\mathrm{pr}}_{\Delta t}(0, T; V_*), \mu \in \mathcal{P},$$

for some suitable norm $\|\cdot\|$ on $L^2([0, T], V_*)$. $\|\cdot\|$ is usually the Bochner-norm. We denote a certified ROM by $\mathrm{ROM}\,[V_*, A_*, E_*]$. Additional to `eval_state`, inherited from M_h, a certified ROM has the routines

(iii) $\Delta^u_*(\mu) \leftarrow M_*$`.est_state`$[\mu]$, returning $E_*(A_*(\mu), \mu)$ and

(iv) $(u_*(\mu), \Delta^u_*(\mu)) \leftarrow M_*$`.eval_and_est_state`$[\mu]$, returning both $A_*(\mu)$ and $E_*(A_*(\mu), \mu)$.

These ROMs have the potential to achieve a gain in computational speed, due to the reduced dimension of the state space. To exploit this potential, we formalize additional assumptions.

Assumption 4.2.4.

(i) The dimension of V_* is far smaller than the one of V_h ($N_* \ll N_h$), but still ensuring good approximation properties, in terms of $\| \cdot \|$.

(ii) It is possible to carry out precomputing to prepare the routines, such that the computational complexity depends on N_* and not on N_h (*offline/online-splitting*). Parameter independent calculations will be performed once in advance (*offline-phase*) and their results will be used during the evaluation (*online-phase*).

The introduction of an error estimator in Definition 4.2.3 is an important step. This asserts error-control during the online-phase *for all* vectors $v \in V_*$, not only for solutions yielded by M_*`.eval_state`. Hence, the operator A_* can be replaced by another suitable operator $A_\diamond$ with better characteristics, mapping into the same target space, keeping certification. We can therefore define a new certified ROM, denoted by $\mathrm{ROM}\,[V_*, A_\diamond, E_*]$. This scheme will be used to introduce ML-based surrogates A_{ML} for A_*, achieving a *certified ML-ROM*, see Section 5.4 for details.

However, to ensure training and to enforce the iterative updating demanded by the TR-method in an efficient way, we have to extend the current framework by introducing *adaptive enrichment*. The general idea is that the ROMs M_* are not fixed during the run, but can be adapted to new $\check{\mu} \in \mathcal{P}$, providing good (local) approximation. For the adaptation process data will be collected, for example by solving another model, and used to improve M_*. An example for this is updating A_{ML} by training with an ML-algorithm, using data generated from $\mathrm{ROM}\,[V_*, A_*, E_*]$. Furthermore, this allows the adaptive enrichment to start with relative coarse models and refining them only if needed, making them only so rich as needed. Analogue to the scheme above, we will provide an general abstract notion by introducing the *ROM generator*.

Definition 4.2.5 (ROM generator)**.** Let a state-based model M_* be given. A ROM generator $G_\diamond := \mathrm{ROMGEN}_\diamond\,[M_*]$ is constructed by

(i) an initially empty set $\mathcal{P}_\diamond$ of parameters,

(ii) a routine $G_\diamond$`.extend`$[\mu]$, enlarging the parameter set $\mathcal{P}_\diamond \leftarrow \mathcal{P}_\diamond \cup \{\mu\}$ and collecting all data associated to μ (e.g. by evaluating M_*) required for `pre_compute`, and

(iii) a routine $M_\diamond \leftarrow G_\diamond$`.pre_compute`$[]$, yielding a certified ROM $M_\diamond := \mathrm{ROM}\,[V_\diamond, A_\diamond, E_\diamond]$ based on the current state of $\mathcal{P}_\diamond$.

Optionally, a generator may also provide

(iv) a routine $G_\bullet \leftarrow G_\diamond$`.prolong`$[M_\bullet]$, yielding a new ROM generator $G_\bullet := \mathrm{ROMGEN}_\bullet\,[M_\bullet]$ by extending the state space to $V_\bullet \supset V_*$ and prolonging all so far collected data to V_*. Here $M_\bullet$ is a new state-based model with $M_\bullet := \mathrm{MODEL}[V_\bullet, A_\bullet, S_\bullet]$.

The `precompute` method carries out the offline-calculations necessary for preparing the ROM. An implicit assumption made here is that an extension is either a rather rare occurrence or is not too computational expensive compared to evaluations. Otherwise, the effort of precomputing and extension will nullify the gain from the offline/online-splitting. All these building blocks can be combined to an adaptive framework.

Definition 4.2.6 (Adaptive Framework)**.** Let M_h be a costly FOM as outlined in Assumption 4.2.2. An adaptive framework consists of the ROM generator $G_* := \mathrm{ROMGEN}_*[M_h]$ and $G_\diamond := \mathrm{ROMGEN}_\diamond[M_*]$, where the later supports `prolong`ation and M_* is the state-based model obtained by $G_\diamond$`.pre_compute`$[]$. This framework will be denoted by $\mathrm{AF}\,[G_*, G_\diamond]$.

4.3. Adaptive fallback

We will now discuss the adaptive fallback procedure choosing the most efficient yet accurate model available and updating the others during inference. The framework from above induces a hierarchy of models based on increasing efficiency, typically accompanied by decreasing accuracy. This hierarchy can be stated as follows:

1. FOMs $M_h := \text{MODEL}[V_h, A_h]$ approximate the analytic solutions to a given problem (4.1). They are accurate enough for the given task, but in general too costly to evaluate for multiple parameters, thus satisfying Assumption 4.2.2.
2. Certified ROMs $M_* := \text{ROM}\,[V_*, A_*, E_*]$ approximate the solutions u_h obtained from M_h, but are still costly, even after precomputation.
3. Derived certified ROMs $M_\diamond := \text{ROM}\,[V_*, A_\diamond, E_*]$ approximate the solutions u_* obtained from M_*. They are defined analogue to M_*, however the state-evaulation operator will be replaced by a proper operator $A_\diamond$. By inheriting the error estimator, these models remain certified and are for a suitable choice of $A_\diamond$ more efficient than M_*.

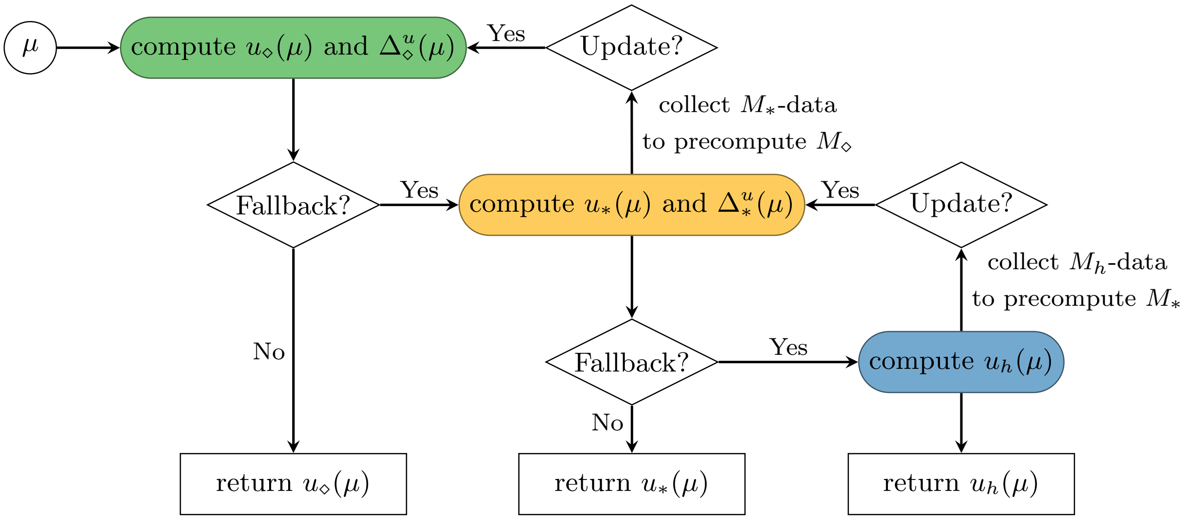


Figure 4.1.: Flowchart of the general adaptive scheme. Derived from [20].

The goal is to evaluate whenever possible the model lowest in the hierarchy and only go a step up if needed. The general evaluation procedure, as shown in Figure 4.1, is therefore as follows: For each new μ the derived ROM $M_\diamond$ will be evaluated first, calculating $u_\diamond$ and $\Delta^u_\diamond$. Based on these results, a condition is checked if a fallback to the next more accurate model M_* is necessary. One simple example of such a condition is checking if the error is below a fixed threshold, but other choices are also possible. If this condition is not satisfied, M_* will be evaluated in the same way as $M_\diamond$. The procedure is applied again to, checking if a fallback to the FOM is needed, based on a second fallback condition. After each fallback the newly obtained solutions are collected by the `extend`-methods of the model next higher in the hierarchy and it is checked if they should be updated, using `pre_compute`. The updates are again controlled by some condition. This scheme allows to update the models and performing productive evaluations simultaneously.

5. Models & Generators

In this chapter, we will adapt the framework outlined in Section 4.2 to primal-dual system of parabolic PDEs, using a Reduced Basis-method as MOR-technique to defined certified RB-ROMs, following the argumentation in [39]. Furthermore, ML-algorithms will be integrated into the MOR-context by defining ML-ROMs, derived from the RB-ROMs. Based on these definitions, error estimator, for the objective functional and its gradient, will be defined, which can be used for the TR-method presented above.

5.1. Full order model

Our goal is to solve Problem 2.1.1 using Algorithm 1 with ML-based surrogate models. First we need to define a method to find accurate approximations to u solving (2.9) and building thereon a FOM. We will follow the standard finite element theory, as for example outlined in Chapters 3 & 4 of [13], introducing spatial approximation and asserting convergence to the analytic solution for sufficient regular functions.

The spatial approximation of the domain Ω will be introduced by a triangulation $\mathcal{T}$, giving a finite basis $\{\phi_i\}_{i\in\{1,\dots,N_h\}} \subset V^{N_h}$, $N_h < \infty$. Let the parameter $h(\mathcal{T}) > 0$ be the maximal diameter of elements in $\mathcal{T}$ and let the state-space V be approximated by the N_h-dimensional subspace $V_h := \operatorname{span}\{\phi_i,\, i = 1,\dots,N_h\} \subset V$. An inner product $\langle\cdot,\cdot\rangle_{V_h}$ on this space can be achieved by restricting $\langle\cdot,\cdot\rangle_V$ to V_h, making it a Hilbert space. The equation (2.9) will be approximated by applying the Ritz-Galerkin projection, getting the initial value problem

$$\begin{aligned} \frac{1}{\Delta t}(u_h^k(\mu) - u_h^{k-1}(\mu), v)_{L^2(\Omega)} + a(u_h^k(\mu), v; \mu) &= b(t^k) f(v;\mu), \\ u_h^0(\mu) &= 0, \end{aligned} \tag{5.1}$$

for all $v \in V_h$ and $k \in \{1,\dots,K\}$. The existence and uniqueness of a solution $u_h(\mu) := (u_h^k(\mu))_{k\in\{0,\dots,K\}}$ in V_h is asserted by the Lax-Milgram theorem. This solution can numerically be approximated with arbitrary accuracy in finitely many steps, for example using the Conjugate gradient (CG)-method [38, Theorem 5.1]. According to the classical FE-theory, the approximation can be performed, such that the error approaches zero for $h \to 0$, as for example shown by the following theorem.

Theorem 5.1.1. [13] Let $u(\mu)$ be the solution to (2.9) and $u_h(\mu)$ its approximation, defined as solution to (5.1). For a sufficient triangulation $\mathcal{T}$ of Ω with piecewise linear basis functions and $u(\mu) \in (H^2(\Omega))^K$ the interpolation error is bounded by

$$\|u^k(\mu) - u_h^k(\mu)\|_{H^1(\Omega)} \leq C h(\mathcal{T}) \|u^k(\mu)\|_{H^2(\Omega)},$$

for all $k \in \{0, \dots, K\}$ and $\mu \in \mathcal{P}$, with constant $C > 0$ depending on $\mathcal{P}$ and $\mathcal{T}$. For higher regularity of $u(\mu)$ higher convergence rates are tangible.

Proof. See [13, Example 3.32 and following discussion]. $\square$

Thus for an appropriate choice of basis functions and sufficient regularity of the analytic solution the approximation error with respect to $\|\cdot\|_{H^1(\Omega)}$ can be made arbitrary small for all $\mu \in \mathcal{P}$ by refining the spatial gird. This proves that part one of Assumption 4.2.2 is satisfied for the solution $u_h(\mu)$ if the grid is fine enough.

However, there is a trade-off between accuracy and computational complexity. The solution of (5.1) with the CG-method requires matrix-vector-multiplication in $\mathbb{R}^{N_h}$. These have a computational complexity that is usually not feasible, because reducing h by refining the grid results in an increase of the dimension N_h and thus in an significant increase of computational cost. We are therefore in a situation described by Assumption 4.2.2. Thus the Model 5.1.2 is indeed a FOM in the sense of Section 4.2.

Model 5.1.2 (Primal FOM)**.** Let the FOM Hilbert space be given by the FE-space V_h and the approximate state-evaluation operator with $\mathcal{P}^{\mathrm{pr}} := \mathcal{P}$ by

$$\begin{aligned} A_h^{\mathrm{pr}} : \mathcal{P}^{\mathrm{pr}} &\to Q_{\Delta t}^{\mathrm{pr}}(0, T; V_h^{\mathrm{pr}}); \\ \mu &\mapsto u_h(\mu), \end{aligned} \tag{5.2}$$

where $u_h(\mu)$ is the solution to (5.1), that can be obtained up to machine accuracy by some numerical procedure `eval_state`$[\mu]$. This FOM to the primal problem is therefore set as the state-based model $M_h^{\mathrm{pr}} := \mathrm{MODEL}[V_h, A_h^{\mathrm{pr}}]$.

Analogue to M_h^{pr}, we can build a Full Order Model M_h^{du} solving the dual problem, returning $p(\tilde{\mu}) \in Q_{\Delta t}^{\mathrm{du}}(0, T; V_h)$ and depending on a trajectory $y \in Q_{\Delta t}^{\mathrm{pr}}(0, T; V_h)$. The parameter $\tilde{\mu}$ is set as $(\mu, y) \in \mathcal{P}^{\mathrm{du}} := \mathcal{P} \times Q_{\Delta t}^{\mathrm{pr}}(0, T; V_h)$. Hence, to match the abstract definition of a state-based model the parameter space of M_h^{du} is chosen as $\mathcal{P}^{\mathrm{du}}$. We use the same FE-space V_h and solve analogue to the primal situation. The problem is then stated as

$$\begin{aligned} \frac{1}{\Delta t}(v, p_h^k(\tilde{\mu}) - p_h^{k+1}(\tilde{\mu}))_{L^2(\Omega)} + a(v, p_h^k(\tilde{\mu}); \mu) &= 2d(y, v) + l^k(v), \\ p_h^{K+1}(\tilde{\mu}) &= 0, \end{aligned} \tag{5.3}$$

for all $v \in V_h$ and $k \in \{1, \dots, K\}$. Hence the dual FOM is defined similar to the primal one 5.1.2.

Model 5.1.3 (Dual FOM)**.** Let the definitions from Model 5.1.2 be in force. The dual $M_h^{\mathrm{du}} := \mathrm{MODEL}[V_h, A_h^{\mathrm{du}}]$ is defined analogue by replacing A_h^{pr} with

$$\begin{aligned} A_h^{\mathrm{du}} : \mathcal{P}^{\mathrm{du}} &\to Q_{\Delta t}^{\mathrm{du}}(0, T; V_h); \\ (\mu, A_h^{\mathrm{pr}}(\mu)) =: \tilde{\mu} &\mapsto p_h(\tilde{\mu}), \end{aligned}$$

returning the solution to (5.3), for $y = A_h^{\mathrm{pr}}(\mu)$.

5.2. Reduced basis model

The most straight forward way to employ ML-methods for solving Problem 2.1.1 would be to choose suitable models to learn a function mapping μ to the solutions of (5.1) and (5.3), getting approximative trajectories $u_{\mathrm{ML}}(\mu)$ and $p_{\mathrm{ML}}((\mu, u_{\mathrm{ML}}(\mu)))$, but this would cause the problems discussed so far. Moreover, the relaxed first order condition 3.1.1 is in general not met by the ML-solutions, so that they can not be directly applied to Algorithm 1. Hence and to maintain the framework layout from the last chapter, we must take the extra step of defining certified ROMs for both full order models ($M_{\mathrm{RB}}^{\mathrm{pr}}$ and $M_{\mathrm{RB}}^{\mathrm{du}}$). This involves utilizing error bounds for trajectories in the RB-ROM-spaces and establishing an adaptive procedure for both branches. As mentioned, we will use the well-studied RB-methods for this [40, 19].

Suppose that two sets of V_h-orthonormal vectors $\{\psi_i^{\mathrm{pr}},\, i = 1, \ldots, N_{\mathrm{RB}}^{\mathrm{pr}}\}$ and $\{\psi_i^{\mathrm{du}},\, i = 1, \ldots, N_{\mathrm{RB}}^{\mathrm{du}}\}$, are given. With these sets as *reduced basis*, new reduced vectors spaces

$$\begin{aligned}
V_{\mathrm{RB}}^{\mathrm{pr}} &:= \operatorname{span}\{\psi_i^{\mathrm{pr}},\, i = 1, \ldots, N_{\mathrm{RB}}^{\mathrm{pr}}\} \subset V_h \text{ and} \\
V_{\mathrm{RB}}^{\mathrm{du}} &:= \operatorname{span}\{\psi_i^{\mathrm{du}},\, i = 1, \ldots, N_{\mathrm{RB}}^{\mathrm{du}}\} \subset V_h
\end{aligned}$$

will be constructed. We will choose the bases such that $V_{\mathrm{RB}}^{\mathrm{pr}} = V_{\mathrm{RB}}^{\mathrm{du}}$, see Definition 5.3.3. The importance of this will become clear in the further discussion. To provide an optimal approximation to the FOM-spaces, while maintaining a reasonable small dimension, the bases have to been carefully chosen. We will use a variant of the POD-algorithm [49] for selecting them. The details of the basis-generation will be discussed in the Section 5.3.

The primal RB-ROM is given by projecting (5.1) to $V_{\mathrm{RB}}^{\mathrm{pr}}$. The reduced primal state-evaluation operator

$$\begin{aligned}
A_{\mathrm{RB}}^{\mathrm{pr}} : \mathcal{P}^{\mathrm{pr}} &\to Q_{\Delta t}^{\mathrm{pr}}(0, T; V_{\mathrm{RB}}^{\mathrm{pr}}); \\
&\mu \mapsto u_{\mathrm{RB}}(\mu) := (u_{\mathrm{RB}}^k(\mu))_{k \in \{0, \ldots, K\}}
\end{aligned}$$

returns the solution to

$$\begin{aligned}
\frac{1}{\Delta t}(u_{\mathrm{RB}}^k(\mu) - u_{\mathrm{RB}}^{k-1}(\mu), v)_{L^2(\Omega)} + a(u_{\mathrm{RB}}^k(\mu), v; \mu) &= b(t^k) f(v; \mu), \\
u_{\mathrm{RB}}^0(\mu) &= 0,
\end{aligned} \tag{5.4}$$

for all $v \in V_{\mathrm{RB}}^{\mathrm{pr}}$ and $k \in \{1, \ldots, K\}$. Solving of the problem will again be carried out with the CG-method.

The dual RB-ROM is provided largely analogue by a Galerkin projection of (5.3), but the primal trajectory $y \in Q_{\Delta t}^{\mathrm{pr}}(0, T; V_h)$, determining the right hand side, has to be projected itself to $V_{\mathrm{RB}}^{\mathrm{pr}}$. Thus for $\tilde{\mu} := (\mu, y) \in \mathcal{P}^{\mathrm{du}}$ we have to solve

$$\begin{aligned}
\frac{1}{\Delta t}(v, p_{\mathrm{RB}}^k(\tilde{\mu}) - p_{\mathrm{RB}}^{k+1}(\tilde{\mu}))_{L^2(\Omega)} + a(v, p_{\mathrm{RB}}^k(\tilde{\mu}); \mu) &= 2d(\Pi_{V_{\mathrm{RB}}^{\mathrm{pr}}}(y), v) + l^k(v), \\
p_{\mathrm{RB}}^{K+1}(\tilde{\mu}) &= 0,
\end{aligned} \tag{5.5}$$

for all $v \in V_{\mathrm{RB}}^{\mathrm{du}}$ and $k \in \{1, \ldots, K\}$. The operator $\Pi_{V_{\mathrm{RB}}^{\mathrm{pr}}} : V_h \to V_{\mathrm{RB}}^{\mathrm{pr}}$ is the canonical projection.

Thus the reduced dual state-evaluation operator is given by

$$\begin{aligned} A_{\mathrm{RB}}^{\mathrm{du}} &: \mathcal{P}^{\mathrm{du}} \to Q_{\Delta t}^{\mathrm{du}}(0,T;V_{\mathrm{RB}}^{\mathrm{du}}); \\ (\mu, y) =: \tilde{\mu} &\mapsto p_{\mathrm{RB}}(\tilde{\mu}) := (p_{\mathrm{RB}}^{k}(\tilde{\mu}))_{k\in\{1,\dots,K+1\}}. \end{aligned}$$

Well-definedness of the state-evaluation operator is again assured by the Lax-Milgram theorem. The errors made by the projections into the reduced spaces will be given in terms of the *spatiotemporal energy norms*, measuring the deviation along the trajectory with respect to the left hand sides.

Definition 5.2.1 (Energy norm). The primal and dual spatiotemporal energy norms for $\mu \in \mathcal{P}$ are defined as

$$\|v\|_{\mu}^{\mathrm{pr}} := \left[(v^K, v^K)_{L^2(\Omega)} + \Delta t \sum_{k=1}^{K} a(v^k, v^k; \mu) \right]^{\frac{1}{2}} \quad \forall v \in Q_{\Delta t}^{\mathrm{pr}}(0,T;V_h)$$

and

$$\|v\|_{\mu}^{\mathrm{du}} := \left[(v^1, v^1)_{L^2(\Omega)} + \Delta t \sum_{k=1}^{K} a(v^k, v^k; \mu) \right]^{\frac{1}{2}} \quad \forall v \in Q_{\Delta t}^{\mathrm{du}}(0,T;V_h).$$

To achieve certification, we first introduce the *primal and dual residual operators*, expressing the error of a trajectory for $\mu \in \mathcal{P}^{\mathrm{pr}}$ or $\tilde{\mu} \in \mathcal{P}^{\mathrm{du}}$ respectively according to a given test vector.

Definition 5.2.2 (Residuals). Let the residual operators for $k \in \{1,\dots,K\}$ be (re)defined as

$$\begin{aligned} r_{\mathrm{pr}}^{k} &: Q_{\Delta t}^{\mathrm{pr}}(0,T;V_h) \times V_h \times \mathcal{P}^{\mathrm{pr}} \to \mathbb{R}; \\ (y, v; \mu) &\mapsto r_{\mathrm{pr}}^{k}(y, v; \mu) := b(t^k) f(v; \mu) - a(y^k, v; \mu) - \frac{1}{\Delta t}(y^k - y^{k-1}, v)_{L^2(\Omega)} \end{aligned}$$

and

$$\begin{aligned} r_{\mathrm{du}}^{k} &: Q_{\Delta t}^{\mathrm{du}}(0,T;V_h) \times V_h \times \mathcal{P}^{\mathrm{du}} \to \mathbb{R}; \\ (z, v; (\mu, y)) &\mapsto r_{\mathrm{du}}^{k}(z, v; (\mu, y)) := 2d(y^k, v) + l^k(v) - a(v, z^k; \mu) - \frac{1}{\Delta t}(v, z^k - z^{k+1})_{L^2(\Omega)}. \end{aligned}$$

Following the proof of Lemma 8 in [39] we define the *primal and dual error bound operators.* These extend the residual operators to obtain a posteriori error bounds for the RB-models with respect to the spatiotemporal norms, utilizing the assumptions on continuity and coercivity made in Section 2.1.

Definition 5.2.3 (Error bound operator). The error bound operators are defined by

$$\begin{aligned} E_{\mathrm{RB}}^{\mathrm{pr}} &: Q_{\Delta t}^{\mathrm{pr}}(0,T;V_{\mathrm{RB}}^{\mathrm{pr}}) \times \mathcal{P}^{\mathrm{pr}} \to \mathbb{R}_{\geq 0}; \\ (y, \mu) &\mapsto E_{\mathrm{RB}}^{\mathrm{pr}}(y; \mu) := \left(\frac{\Delta t}{\alpha_{\mathrm{LB}}(\mu)} \sum_{k=1}^{K} \|r_{\mathrm{pr}}^{k}(y, \cdot\,; \mu)\|_{V_h'}^{2} \right)^{\frac{1}{2}} \end{aligned}$$

and

$$E^{\mathrm{du}}_{\mathrm{RB}} : Q^{\mathrm{du}}_{\Delta t}(0,T;V^{\mathrm{pr}}_{\mathrm{RB}}) \times \mathcal{P}^{\mathrm{du}} \to \mathbb{R}_{\geq 0};$$

$$(z,(\mu,y)) \mapsto E^{\mathrm{du}}_{\mathrm{RB}}(z;(\mu,y)) := \left(8\gamma_d^2 \left(\frac{E^{\mathrm{pr}}_{\mathrm{RB}}(y;\mu)}{\alpha_{\mathrm{LB}}(\mu)} \right)^2 + \frac{\Delta t}{\alpha_{\mathrm{LB}}(\mu)} \sum_{k=1}^{K} \| r^k_{\mathrm{du}}(z,\cdot\,;(\mu,y))\|^2_{V_h'} \right)^{\frac{1}{2}}.$$

With these definitions it is possible to give a posterori error bounds *for all reduced solutions.*

Lemma 5.2.4 (A posterori error bounds; [39, Lemma 8])**.** For $\mu \in \mathcal{P}$ let $u_h(\mu) \in Q^{\mathrm{pr}}_{\Delta t}(0,T;V_h)$ be the solution to (5.1) and $p_h(\tilde{\mu}) \in Q^{\mathrm{du}}_{\Delta t}(0,T;V_h)$ to (5.3), with $\tilde{\mu} := (\mu, u_h(\mu))$. Let further the vectors $y \in Q^{\mathrm{pr}}_{\Delta t}(0,T;V^{\mathrm{pr}}_{\mathrm{RB}})$ and $z \in Q^{\mathrm{du}}_{\Delta t}(0,T;V^{\mathrm{du}}_{\mathrm{RB}})$ be given. Then the error bounds

$$\|u_h(\mu) - y\|^{\mathrm{pr}}_{\mu} \leq E^{\mathrm{pr}}_{\mathrm{RB}}(y;\mu) \text{ and } \|p_h(\tilde{\mu}) - z\|^{\mathrm{du}}_{\mu} \leq E^{\mathrm{du}}_{\mathrm{RB}}(z;(\mu,y))$$

are given.

Proof. See [39, Lemma 8]. □

With these error bounds we can define the certified RB-ROMs.

Model 5.2.5 (RB-ROMs)**.** With the argumentation above in place the primal $M^{\mathrm{pr}}_{\mathrm{RB}}$ and dual $M^{\mathrm{du}}_{\mathrm{RB}}$ RB-ROM are given by following Definition 4.2.3, as

$$M^{\mathrm{pr}}_{\mathrm{RB}} := \mathrm{ROM}\left[V^{\mathrm{pr}}_{\mathrm{RB}}, A^{\mathrm{pr}}_{\mathrm{RB}}, E^{\mathrm{pr}}_{\mathrm{RB}}\right] \text{ and } M^{\mathrm{du}}_{\mathrm{RB}} := \mathrm{ROM}\left[V^{\mathrm{du}}_{\mathrm{RB}}, A^{\mathrm{du}}_{\mathrm{RB}}, E^{\mathrm{du}}_{\mathrm{RB}}\right].$$

The routines `eval_state` can be implemented by some suitable numerical procedure.

In the following we will denote these models with M^{*}_{RB}, if we do not need to distinguish between the primal and the dual branch.

5.3. Reduced basis generator

It remains to discuss how the reduced bases $\{\psi^*_i\}_{i\in\{1,\dots,N^*_{\mathrm{RB}}\}}$ will be constructed. As mentioned above we will use a more economical variant of the well-studied Proper Orthogonal Decomposition (POD)-method [25, 40, 49]. The POD can be used to construct l^2-optimal subspaces $V_r \subset V_h$ to a finite dimensional Hilbert space V_h, using a (truncated) Singular Value Decomposition (SVD).

Definition 5.3.1 (Proper Orthogonal Decomposition (POD); [25, Definition 2.1])**.** Let $\mathcal{S} := \{s_1, \dots, s_{N_s}\} \subset V_h^{N_s}$, with $N_s \ll N_h$ be a finite set. The vectors s_i are called *snapshots.* The POD returns the (ordered by decreasing singular value) $\langle\cdot,\cdot\rangle_{V_h}$-orthonormal left singular vectors $\varphi_1, \dots, \varphi_{N_s}$ and associated left singular values $\sigma_1, \dots, \sigma_{N_s}$ of the linear map

$$\underline{\mathcal{S}} : \mathbb{R}^{N_s} \to V_h, \;\; e_i \mapsto \underline{\mathcal{S}}(e_i) := s_i, \;\; i \in \{1, \dots, N_s\}.$$

For $r \leq N_s$ the subspace $V_r := \mathrm{span}\{\varphi_1, \dots, \varphi_r\}$ provides an l^2-optimal r-dimensional approximation to $\mathcal{S}$, in the sense of the following theorem.

Theorem 5.3.2. [25, Theorem 2.4] Let the set $\mathcal{S}$ be as in Definition 5.3.1 and $1 \leq r \leq N_s$. Let $\varphi_1, \ldots, \varphi_{N_s}$ and $\sigma_1, \ldots, \sigma_{N_s}$ be returned by the according POD. Then it holds for $V_r := \operatorname{span}\{\varphi_1, \ldots, \varphi_r\}$ and $\mathcal{X}_r := \{X \subset V_h \mid \text{linear subspace and } \dim(X) = r\}$ that

$$I(r) := \sum_{s \in \mathcal{S}} \|s - \Pi_{V_r} s\|_{V_h}^2 = \min_{X \in \mathcal{X}_r} \sum_{s \in \mathcal{S}} \|s - \Pi_X s\|_{V_h}^2 = \sum_{m=r+1}^{N_s} \sigma_m^2.$$

Proof. See [25, Theorem 2.4]. ☐

Obviously, there exist for all $\epsilon_{\text{POD}} > 0$ a $r \in \mathbb{N}$ such that $I(r) \leq \epsilon_{\text{POD}}$.

For the algorithmic implementation we will use the in [25] presented enhanced variant Hierachical Approximate POD (HaPOD) in its incremental version, with equalsized (except the last) chunks. The idea behind this method is to split up the calculation by running multiple PODs with reduced snapshot sets each by hierachical splitting $\mathcal{S}$ into smaller chunks. The actual calculation of the singular vectors will be handled by the *methods of snapshots* [49]. Thereby eigenvalue decomposition of the Gramian matrix $(\langle s_i, s_j \rangle_V)_{(i,j)} \in \mathbb{R}^{N_s \times N_s}$ is performed, instead of solving the more demanding N_h-dimensional eigenvalue problem. This reduces the computational effort, due to $N_s \ll N_h$. The use of HaPOD instead of POD results in better performance characteristics, such as memory use and computational complexity (see [25] for details). Moreover, the splitting of the POD results in greater numerical stability, which is a particular issue of the methods of snapshots. Note that a similar version to Theorem 5.3.2 holds for the HaPOD-method [25].

For asserting (3.4) in the relaxed first-order condition 3.1.1 and the assumptions made in Section 4.2, we have to construct the reduced bases by adaptive enrichment. This allows us to build the iterative structure necessary for the trust region method. The adaptive fashion will simultaneously enable efficient calculations by maintaining reasonable small dimensions of V_{RB}^*, yet maintaining sufficient accurate RB-solution. Therefore, consider that the snapshots will be constructed as the trajectory solving the high fidelity problems (5.1) and (5.3) for $\mu \in \mathcal{P}^{\text{pr}}$ or $\tilde{\mu} \in \mathcal{P}^{\text{du}}$. Theorem 5.3.2 shows that the error between the full order solution $u_h(\mu)$ to (5.1) and reduced order solution $u_{\text{RB}}(\mu)$ to (5.4) can be made smaller than an arbitrary $\epsilon_{\text{RB}}^{\text{pr}} > 0$ by reducing ϵ_{POD}, i.e. holds

$$\|u_h(\mu) - u_{\text{RB}}(\mu)\|_{Q_{\Delta t}^{\text{pr}}(0,T;V_h)} < \epsilon_{\text{RB}}^{\text{pr}},$$

if $\varphi_1, \ldots, \varphi_r \in V_{\text{RB}}^{\text{pr}}$ (for the dual case analogue). We will actually use a light variation of this procedure in the RB generator 5.3.3.

To assert online-efficiency it is advisable to also perform precompuation preparing the models 5.2.5. What precompuation methods are actually applicable vary a lot and depend significantly on the problem studied. For our situation we will enact two computational procedures here. The first one utilizes that the operators a and f are affinely decomposable, i.e. they can be decomposed as shown in (2.6). This enables us to setup an offline-projection of these operators to $V_{\text{RB}}^* \subset V_h$. For the basis $\Psi_{\text{RB}}^* := \{\psi_i^*,\, i = 1, \ldots, N_{\text{RB}}^*\}$ of V_{RB}^* we calculate the matrices $A^q(\Psi_{\text{RB}}^*) := (a^q(\psi_i^*, \psi_j^*))_{(i,j)}$ and $f^{\tilde{q}}(\Psi_{\text{RB}}^*) := (f^{\tilde{q}}(\psi_i^*))_{(i)}$ in advance during the offline-phase and store them. During the online-pahse the operator can for $v, w \in V^*_{\mathrm{RB}}$ be expressed by

$$a(v,w;\mu) = \sum_{q=1}^{Q_a} \Theta_a^q(\mu)\underline{v}^T A^q(\Psi^*_{\mathrm{RB}})\underline{w} \text{ and } f(v;\mu) = \sum_{\tilde{q}=1}^{Q_f} \Theta_f^{\tilde{q}}(\mu) f^{\tilde{q}}(\Psi^*_{\mathrm{RB}})^T \underline{v}, \tag{5.6}$$

where $\underline{v}, \underline{w}$ are the Degree of Freedom (DoF)-vectors to v, w with respect to Ψ^*_{RB}. Thus, during the online-phase only this linear combination and low-dimensional ($N_{\mathrm{RB}} \ll N_h$) matrix-vector-multiplications have to be performed.

A second precomputation, that we want to exploit, is regarding the error estimate operators E^*_{RB}. For both error estimators the dual norm of the residual functionals in V_h' needs to be calculated. These are accessible via the Riesz-representatives of the residual operators $\mathsf{R}(r^k_*(x,\cdot\,;\mu)) \in V_h$, exploiting the canonical isometry $\|r^k_*(x,\cdot\,;\mu)\|_{V_h'} = \|\mathsf{R}(r^k_*(x,\cdot\,;\mu))\|_{V_h}$, for a primal or dual trajectory $x \in Q^*_{\Delta t}(0,T;V^*_{\mathrm{RB}})$. The Riesz-representatives can be calculated as solution of the high-dimensional problem

$$r^k_*(x,\phi_i;\mu) = \langle \mathsf{R}(r^k_*(x,\cdot\,;\mu)), \phi_i\rangle_V = \left(\langle\phi_1,\phi_i\rangle_V, \dots, \langle\phi_{N_h},\phi_i\rangle_V\right)^T \underline{r}^k_*$$

for all $i = 1, \dots, N_h$, where $\{\phi_i\}_{i\in\{1,\dots,N_h\}}$ is the basis of V_h. The DoF-vector $\underline{r}^k_*$ is therefore given by $\underline{r}^k_* = E^k_*(x,\mu)$, with

$$\begin{aligned} E^k_* : Q^*_{\Delta t}(0,T;V_h) \times \mathcal{P}^* &\to \mathbb{R}^{N_h} && (5.7)\\ (x,\mu) &\mapsto G^{-1}\left(r^k_*(x,\phi_1;\mu), \dots, r^k_*(x,\phi_{N_h};\mu)\right)^T, && (5.8)\end{aligned}$$

where G is the Gramian matrix, i.e. $\left(\langle\phi_i,\phi_j\rangle_V\right)_{(i,j)} \in \mathbb{R}^{N_h\times N_h}$. To reduce the online-cost, we follow the argumentation in [7] by constructing an orthonormal basis (with respect to $\langle\cdot,\cdot\rangle_V$) of the space $W^* := \langle \cup_{k=1}^K \mathrm{Im}(E^k_*)\rangle$ and perform a projection. The affine structure of $a(\cdot,\cdot\,;\mu)$ and $f(\cdot\,;\mu)$ allows us to decomposition E^k_* and perform calculations on the parts separately. For a RB-basis $\Psi^*_{\mathrm{RB}} \subset V_h$ we define for $q \in \{1,\dots,Q_a\}$, $\tilde{q} \in \{1,\dots,Q_f\}$ and $k \in \{1,\dots,K\}$

$$w^{\mathrm{pr}}(\Psi^{\mathrm{pr}}_{\mathrm{RB}}) := \left\{ \left((\psi,\phi_i)_{L^2(\Omega)}\right)^T_{(i)}, \left(a^q(\psi,\phi_i)\right)^T_{(i)}, \left(b(t^k)f^{\tilde{q}}(\phi_i)\right)^T_{(i)} \,\middle|\, \psi \in \Psi^{\mathrm{pr}}_{\mathrm{RB}} \right\} \subset \mathbb{R}^{N_h}$$

and

$$w^{\mathrm{du}}(\Psi^{\mathrm{du}}_{\mathrm{RB}}) := \left\{ \left((\psi,\phi_i)_{L^2(\Omega)}\right)^T_{(i)}, \left(a^q(\psi,\phi_i)\right)^T_{(i)}, \left(d(\psi,\phi_i)\right)^T_{(i)}, \left(l^k(\phi_i)\right)^T_{(i)} \,\middle|\, \psi \in \Psi^{\mathrm{du}}_{\mathrm{RB}} \right\} \subset \mathbb{R}^{N_h}.$$

A I^*-dimensional basis $\mathcal{B}^* := \{\varphi^*_i\}_{i\in\{1,\dots,I^*\}}$ of W^* can be constructed by applying G^{-1} to $w^*(\Psi^*_{\mathrm{RB}})$, followed by a POD. The operator E^k_* are then projected onto Ψ^*_{RB} as basis of the source space and $\mathcal{B}^*$ for the image space. Analogue to (5.6) will $\mathcal{B}^*$ be calculated and E^k_* assembled in the offline-phase. These operator then return the DoF-vector $\underline{r}^k_* \in \mathbb{R}^{I^*}$ with respect to $\mathcal{B}^*$.

In general, it holds that $I^* \ll N_h$, thus the number of calculations during the online-phase can be significantly reduced by additionally projecting the operator E^k_* to $\mathcal{B}^*$ in the images space. In the described manner all N_h-multiplication will be carried out in the offline-phase, making the online-calculations (relative) low-dimensional. This decomposition also makes the operator numerical more stable [7]. Furthermore, the orthogonalization reduces the (online-)effort for determining the norm of $\underline{r}_*^k$ by effectively avoiding matrix-vector-multiplications, giving the dual norm by

$$\|r_*^k(x,\cdot\,;\mu)\|_{V_h'} = \|\underline{r}_*^k\|_{V_h} = \left(\sum_{i,j=1}^{I^*} \underline{r}_{*,i}^k \underline{r}_{*,j}^k \underbrace{\langle \varphi_i^*, \varphi_j^* \rangle_V}_{\delta_{ij}}\right)^{\frac{1}{2}} = \sqrt{\underline{r}_*^{kT} \underline{r}_*^k}.$$

To formalize these different aspects and to integrate them into the framework from Section 4.2, we are introducing RB-ROMgenerator G_{RB}^* in the sense of Definition 4.2.5 to the model M_{RB}^*.

Definition 5.3.3 (RB-ROM generator)**.** Let a FOM M_h^* and an error-tolerance $\epsilon_{\text{RB}}^* > 0$ be given. The space V_{RB}^* is initialized with an empty basis, i.e. $V_{\text{RB}}^* := \langle \Psi_{\text{RB}}^{*,0} \rangle$ with $\Psi_{\text{RB}}^{*,0} := \emptyset$. Let further be the set of collected parameters $\mathcal{P}_{\text{RB}}^{*,0}$ and the image-basis $\mathcal{B}^{*,0}$ initially also be set as empty. A RB generator $G_{\text{RB}}^* := \text{ROMGEN}_{\text{RB}}[M_h^*]$ associated to M_h^*, can be obtained by defining the following routines:

(i) G_{RB}^*`.extend`$[\mu]$: For a new $\mu \in \mathcal{P}^*$ the parameter set will be enlarged, i.e. $\mathcal{P}_{\text{RB}}^{*,n+1} := \mathcal{P}_{\text{RB}}^{*,n} \cup \{\mu\}$, with n being the number of parameters collected up to this point in time. The FOM will either be evaluated for the new parameter μ, i.e. $u_h(\mu) \leftarrow M_h^*$`.eval_state`$[\mu]$ or cached solutions for μ are used. The snapshots $\mathcal{S}$ are defined by the residuum of the states along the resulting trajectory to their projections into the current reduced space V_{RB}^*, i.e.

$$\mathcal{S} := \left[u_h^1(\mu) - \Pi_{V_{\text{RB}}^{*,n}} u_h^1(\mu) \,\middle|\, \dots \,\middle|\, u_h^K(\mu) - \Pi_{V_{\text{RB}}^{*,n}} u_h^K(\mu)\right] \in \mathbb{R}^{N_h \times K}.$$

The Matrix $\mathcal{S}$ is compressed by the `HaPOD`-algorithm and $\Psi_{\text{RB}}^{*,n}$ extended by the new basis vectors, followed by re-orthogonalisation with respect to $\langle \cdot, \cdot \rangle_{V_h}$ by a modified gram-schimdt-algorithm. The reduced space V_{RB}^* will be redefined by

$$\Psi_{\text{RB}}^{*,n+1} := \texttt{MGS}\left(\Psi_{\text{RB}}^{*,n} \cup \texttt{HaPOD}\left(\mathcal{S}, \epsilon_{\text{POD}}\right)\right) \text{ and } V_{\text{RB}}^* \leftarrow \langle \Psi_{\text{RB}}^{*,n+1} \rangle. \tag{5.9}$$

The tolerance $\epsilon_{\text{POD}} > 0$ has to be chosen such that the trajectory $u_{\text{RB}}^*(\mu)$ at μ is sufficiently accurate, i.e. $E_{\text{RB}}^*(u_{\text{RB}}^*(\mu), \mu) < \epsilon_{\text{RB}}^*$, after extending V_{RB}^*. This can be done for example by setting ϵ_{POD} to the machine accuracy and argumenting with the continuity of the residual operators. As a final step we assure that the reduced spaces coincide. Therefore after the reset of $V_{\text{RB}}^{\text{pr}}$ in (5.9) is performed, $V_{\text{RB}}^{\text{du}}$ is set equal to $V_{\text{RB}}^{\text{pr}}$. For the dual case vice-versa.

(ii) $M_{\text{RB}}^* \leftarrow G_{\text{RB}}^*$`.precompute`$[]$:

Operator Projection: If $\Psi_{\text{RB}}^{*,n} = \emptyset$ set $a \equiv 0$ and $f \equiv 0$. If $\Psi_{\text{RB}}^{*,n} \neq \emptyset$, calculate $A^q(\Psi_{\text{RB}}^{*,n})$ and $f^{\tilde{q}}(\Psi_{\text{RB}}^{*,n})$ for all $q \in \{1, \dots, Q_a\}$ and $\tilde{q} \in \{1, \dots, Q_f\}$. For better performance the matrices $A^q(\Psi_{\text{RB}}^{*,n-1})$ and $f^{\tilde{q}}(\Psi_{\text{RB}}^{*,n-1})$ will be reused, avoiding redundant calculations. Assemble a and f as operators on $\mathbb{R}^{N_{\text{RB}}^*} \times \mathcal{P}^*$ as set in (5.6).

Error estimator: Extend the Image-basis $\mathcal{B}^*$, if necessary, using `HaPOD` with a sufficient tolerance ϵ_{POD}. I.e. perform

$$\mathcal{B}^{*,n+1} := \mathtt{HaPOD}\left(\mathcal{B}^{*,n} \cup G^{-1}(w^*(\Psi_{\mathrm{RB}}^{*,n+1} \setminus \Psi_{\mathrm{RB}}^{*,n})), \epsilon_{\mathrm{POD}}\right) \text{ and } \mathcal{B}^* \leftarrow \mathcal{B}^{*,n+1}.$$

Assemble E_*^k as operator on $\mathbb{R}^{KN_{\mathrm{RB}}^*} \times \mathcal{P}^*$ as outlined above.

The `extend`-method asserts that $V_{\mathrm{RB}}^{\mathrm{pr}}$ and $V_{\mathrm{RB}}^{\mathrm{du}}$ coincide. This is necessary, because calculating the gradient via the dual problem returns only accurate results if the state-spaces of the primal and dual problem coincide. Otherwise, the results might become inaccurate. There are other methods to reformulate Problem 2.1.1 in fashions that allow different spaces, e.g. presented in [28]. However, we will restrict ourselves for the moment, to a situation with equal reduced spaces.

5.4. Machine learning ROM & generator

As already mentioned in Section 4.2, can the definition of the RB-ROMs easily be extended to ML-ROMs by replacing the solution operator $A_{\mathrm{RB}}^* : \mathcal{P}^* \to V_{\mathrm{RB}}^*$ with a suitable ML-operator A_{ML}^* and inheriting the error estimators. The centerpieces for defining the operators A_{ML}^* are trainable maps T^*, learned by an ML-algorithm $\mathcal{A}^*$. Hence, we assume that a method $T^* \leftarrow \mathtt{train}^*[z^*]$ is available, taking some set of training data $z^* \in (\mathcal{Z}^*)^m := (\mathcal{X}^* \times \mathcal{Y}^*)^m$, with $m \in \mathbb{N}$ data points, and returning an operator T^* estimating the solution $u_{\mathrm{ML}}^*(\mu)$ to the problems (5.4) or (5.5) respectively.

The performance of many learning algorithms, and thus the quality of the achieved surrogates, often depend on some implicit assumptions to the (training) data. For example, the accuracy may suffer if the range of the inputs is too large. To address this and provide more flexibility let some operator $\Phi^* : \mathcal{X}^* \to \mathcal{X}^*$ be defined, transforming the training data $z^* = ((x_i, y_i))_{i \in \{1,\dots,m\}}$ to $\tilde{z}^* := ((\Phi^*(x_i), y_i))_{i \in \{1,\dots,m\}} \in (\mathcal{Z}^*)^m$. Using such a transformation means that the training is actually performed as $\mathtt{train}^*[\tilde{z}^*]$. Moreover, during the inference we have to correct the input data accordingly (s. below). Similarly, a transformation to the output data could be applied. Note that in such cases, the mapping has to be bijective to allow backtransformation during the inference. The actual definition of T^* and the other components depend on the algorithm used. Here we show the *time-vectorized* variant from [20].

Let therefore $T^* : \mathbb{R}^{P^*} \to \mathbb{R}^{KN_{\mathrm{RB}}^*}$ be trained such that this operator predicts the ROM-coefficients for all times at once. We thus obtain $u_{\mathrm{ML}}(\mu)$ by setting the DoF-vector as

$$\underline{u_{\mathrm{ML}}^{*,k}(\mu)}_n := (T^* \circ \Phi^*)(\mu)_{(k-1)N_{\mathrm{RB}}^*+n} \text{ for } 1 \leq k \leq K \text{ and } 1 \leq n \leq N_{\mathrm{RB}}^*.$$

The training set consists therefore of tuples of the form $(\mu, v) \in \mathcal{P} \times \mathbb{R}^{KN_{\mathrm{RB}}^*} =: \mathcal{X}^* \times \mathcal{Y}^*$. Additionally, to meet the initial conditions from Section 5.1, we need to extend the DoF-trajectories by $\underline{u_{\mathrm{ML}}^{\mathrm{pr},0}(\mu)} := 0$ for the primal case or $\underline{u_{\mathrm{ML}}^{\mathrm{du},K+1}(\mu)} := 0$ for the dual one. To enhance performance only parameters μ will be used as input. I.e. in contrast to the RB-ROM, depends the dual-ML-estimate not explicitly on the primal solution. Hence, the ML-state-evaluation operator is given as

$$\begin{aligned} A_{\mathrm{ML}}^* &: \mathcal{P} \to Q_{\Delta t}^*(0,T;V_{\mathrm{RB}}^*); \\ \mu &\mapsto u_{\mathrm{ML}}(\mu) := \left(\sum_{n=1}^{N_{\mathrm{RB}}^*} \underline{u_{\mathrm{ML}}^{*,k}(\mu)}_n \psi_{\mathrm{RB}}^* \right)_{k \in \mathcal{K}}, \end{aligned} \tag{5.10}$$

with $\mathcal{K} = \{0, \dots, K\}$ for the primal case and $\mathcal{K} = \{1, \dots, K+1\}$ for the dual. Remark that $\underline{u_{\mathrm{ML}}^{*,k}(\mu)}_n$ is not explicitly determined by $\underline{u_{\mathrm{ML}}^{*,k-1}(\mu)}_n$. This allows us in theory to parallelize the evaluations of the time steps, gaining an additional speed up. In contrast to the RB-ROM, for a general ML-ROM it *can not* be asserted that the retrained model is accurate. The accuracy (even on the training data) depends on the model and its configuration.

Model 5.4.1 (Certified ML-ROM)**.** With the definitions above in place, the primal $M_{\mathrm{ML}}^{\mathrm{pr}}$ and dual $M_{\mathrm{ML}}^{\mathrm{du}}$ ML-ROM are given by

$$M_{\mathrm{ML}}^{\mathrm{pr}} := \mathrm{ROM}\left[V_{\mathrm{RB}}^{\mathrm{pr}}, A_{\mathrm{ML}}^{\mathrm{pr}}, E_{\mathrm{RB}}^{\mathrm{pr}}\right] \text{ and } M_{\mathrm{ML}}^{\mathrm{du}} := \mathrm{ROM}\left[V_{\mathrm{RB}}^{\mathrm{du}}, A_{\mathrm{ML}}^{\mathrm{du}}, E_{\mathrm{RB}}^{\mathrm{du}}\right],$$

with $A_{\mathrm{ML}}^* : \mathcal{P}^* \to V_{\mathrm{RB}}^*$ being for the primal situation defined by (5.10). For the dual case however (5.10) has to be composed with a projection, mapping $(\mu, y) \in \mathcal{P}^{\mathrm{du}}$ to $\mu \in \mathcal{P}$. The error estimators E_{RB}^* are inherited from the RB-ROM, i.e. given by Definition 5.2.3. The routines `eval_state` are completely defined by evaluating $T^* \circ \Phi^*$.

For integrating these ML-ROM into the adaptive scheme and especially to (re)train, we need to define generators G_{ML}^*. To provide flexibility in the choice of the ML-algorithm, we define a general ML-ROM generator. A concrete implementation of such models and generators will be presented for an example in Section 7.2.

Definition 5.4.2 (General ML-ROM generator)**.** Let $M_{\mathrm{RB}}^* = \mathrm{ROM}[V_{\mathrm{RB}}^*, A_{\mathrm{RB}}^*, E_{\mathrm{RB}}^*]$ be the primal or dual RB-ROM. Assume that we have a model $M_{\mathrm{ML}}^* := \mathrm{ROM}[V_{\mathrm{RB}}^*, A_{\mathrm{ML}}^*, E_{\mathrm{RB}}^*]$ derived from M_{RB}^* by substituting A_{RB}^* with an evaluation operator A_{ML}^*, constructed by with a ML-Algorithm $\mathcal{A}^*$, as defined above. The generator $G_{\mathrm{ML}}^* := \mathrm{ROMGEN}[M_{\mathrm{RB}}^*]$ is given by initializing an empty training data tuple $z^* := \emptyset$ and defining the following methods:

(i) G_{ML}^*`.extend`$[\mu]$: We want to learn approximative solutions to the problems (5.4) and (5.5). Therefore, to extend z^* for a new $\mu \in \mathcal{P}$ the methods M_{RB}^*`.eval_state`$[]$ have to be evaluated. For the primal case this returns the trajectory $(u_{\mathrm{RB}}^k(\mu))_{k \in \{0,\dots,K\}}$. The tuple z^* is therefore extended at the end by $(\mu, (\underline{u_{\mathrm{RB}}^k(\mu)})_{k \in \{0,\dots,K\}})$. For the dual case $M_{\mathrm{RB}}^{\mathrm{du}}$`.eval_state`$[\tilde{\mu}]$ returns $(p_{\mathrm{RB}}^k(\tilde{\mu}))_{k \in \{1,\dots,K+1\}})$, with $\tilde{\mu} := (\mu, y)$ for some $y \in Q_{\Delta t}^{\mathrm{pr}}(0,T;V_h)$. The training data is therefore extend in the same way by $(\mu, (\underline{p_{\mathrm{RB}}^k(\tilde{\mu})})_{k \in \{1,\dots,K+1\}})$.

    Remark that for performance reasons each time M_{RB}^*`.eval_state`$[]$ was called the DoF-vectors are stored, such that they can be loaded for training later. An evaluation of M_{RB}^* only occurs if the solution for μ have not been determined yet. Moreover, can also the full order solutions be used for training, by projecting them on to V_{RB}^*.

(ii) $M_{\mathrm{ML}}^* \leftarrow G_{\mathrm{RB}}^*$`.precompute`$[]$: Precomutation for ML-ROM is in essence just the training by $\mathcal{A}^*$ with the collected training data z^*. Before the training is performed an additional transformation $\Lambda^* : \mathcal{Z}^* \to \mathcal{Z}^*$ might be needed, preparing the data. One transformation Λ^*, that we will always apply is the elimination of duplicates from the training set, i.e. deleting a training tuple for μ if another one already exists for the same parameter. Then the `train`-routine is called accordingly, i.e. `train`$^*[\Lambda^*(\tilde{z}^*))]$ and A^*_{ML} reset, by (5.10).

(iii) $G^*_{\mathrm{ML}} \leftarrow G^*_{\mathrm{ML}}$`.prolong`$[M^*_{\mathrm{RB}}]$: This method allows the handling of vector space extension, adapting it to M^*_{ML}. If V^*_{RB} has been extended by G^*_{RB}`.extend`[], the dimension of the stored DoF-vectors does not fit any more the one of V^*_{RB}. It is disadvantageous to delete the stored training data, because this would entail costly recalculations. As a workaround the DoF-vectors are instead prolonged by padding. Remark that is not unproblematic because it suggests that the new entries in the basis do not contribute to the solutions for the μ collected so far, which is usually not the case. Finally G^*_{ML} is reset to the new V^*_{RB} and the model retrained to redefined the image-space, by calling G^*_{ML}`.precompute`[].

5.5. Objective error estimator

Before we can use the adaptive framework in an TR-algorithm, we have to show that a posteriori error estimator for the approximative objective and the approximative gradient exist, which is a precondition to formulate the trust-region, as presented in Section 3.1. We will adapt the argumentation in [39], defining *objective error operators.*

Definition 5.5.1 (Objective error operator)**.** Let the objective error operator be defined by

$$\begin{aligned}
E^J : Q^{\mathrm{pr}}_{\Delta t}(0,T;V^{\mathrm{pr}}_{\mathrm{RB}}) \times Q^{\mathrm{du}}_{\Delta t}(0,T;V^{\mathrm{du}}_{\mathrm{RB}}) \times \mathcal{P} \to \mathbb{R}_{\geq 0};& \\
(y,z,\mu) \mapsto E^J(y,z;\mu) := \left(\Delta t \sum_{k=1}^{K} \|r^k_{\mathrm{du}}(z,\cdot\,;(\mu,y))\|^2_{V_h'}\right)^{\frac{1}{2}} \frac{E^{\mathrm{pr}}_{\mathrm{RB}}(y;\mu)}{\sqrt{\alpha_{\mathrm{LB}}(\mu)}}& \\
+ \frac{\gamma_d}{\alpha_{\mathrm{LB}}(\mu)} \left(E^{\mathrm{pr}}_{\mathrm{RB}}(y;\mu)\right)^2 + \Delta t \left|\sum_{k=1}^{K} r^k_{\mathrm{pr}}(y,z^k;\mu)\right|&
\end{aligned}$$

and

$$\begin{aligned}
E^{\nabla_{\mu_i} J} : Q^{\mathrm{pr}}_{\Delta t}(0,T;V^{\mathrm{pr}}_{\mathrm{RB}}) \times Q^{\mathrm{du}}_{\Delta t}(0,T;V^{\mathrm{du}}_{\mathrm{RB}}) \times \mathcal{P} \to \mathbb{R}_{\geq 0};& \\
(y,z,\mu) \mapsto E^{\nabla_{\mu_i} J}(y,z;\mu) := \left(\Delta t \sum_{k=1}^{K} \|f_{\mu_i}(\cdot\,;\mu)\|^2_{V_h'}\right)^{\frac{1}{2}} \frac{E^{\mathrm{du}}_{\mathrm{RB}}(y,z;\mu)}{\sqrt{\alpha_{\mathrm{LB}}(\mu)}}& \\
+ \frac{\gamma^{\mathrm{UB}}_{a_{\mu_i}}(\mu)}{\alpha_{\mathrm{LB}}(\mu)} E^{\mathrm{pr}}_{\mathrm{RB}}(y;\mu) E^{\mathrm{du}}_{\mathrm{RB}}(y,z;\mu) + \frac{\gamma^{\mathrm{UB}}_{a_{\mu_i}}(\mu)}{\sqrt{\alpha_{\mathrm{LB}}(\mu)}} E^{\mathrm{pr}}_{\mathrm{RB}}(y;\mu) \left(\Delta t \sum_{k=1}^{K} \|z^k\|^2_{V_h}\right)^{\frac{1}{2}}& \\
+ \frac{\gamma^{\mathrm{UB}}_{a_{\mu_i}}(\mu)}{\sqrt{\alpha_{\mathrm{LB}}(\mu)}} E^{\mathrm{du}}_{\mathrm{RB}}(y,z;\mu) \left(\Delta t \sum_{k=1}^{K} \|y^k\|^2_{V_h}\right)^{\frac{1}{2}}.&
\end{aligned}$$

We define the objective functional J by concatenating the time-discretized functional $\tilde{J}$, defined by (2.10), with the high-fidelity state-evaluation operator (5.2). Its gradient $\nabla_\mu J$ can be calculated by applying the argumentation from Section 2.2, replacing V by V_h and enforcing spatial discretizaiton. With the definitions so far it is obvious that $p_h(\tilde{\mu}) := A_h^{\mathrm{du}}(\tilde{\mu})$ with $\tilde{\mu} := (\mu, u_h(\mu))$ is the dual solution to $u_h(\mu) := A_h^{\mathrm{pr}}(\mu)$. Thus, $\nabla_\mu J$ is given by (2.16), using the discrete versions $u_h(\mu)$ and $p_h(\tilde{\mu})$ of the trajectories $u(\mu)$ and $p(\mu)$. We set

$$J(\mu) := \tilde{J}(u_h(\mu); \mu) = \tilde{J}(A_h^{\mathrm{pr}}(\mu); \mu), \tag{5.11}$$

implying

$$\nabla_\mu J(\mu) = \nabla_\mu \tilde{J}(u_h(\mu), p_h(\tilde{\mu}); \mu) = \nabla_\mu \tilde{J}(A_h^{\mathrm{pr}}(\mu), A_h^{\mathrm{du}}(\tilde{\mu}); \mu). \tag{5.12}$$

Theorems 9 and 10 stated in [39] show that error bounds can be formulated if the state-space of both adaptive solutions is the respective RB-spaces.

Theorem 5.5.2 (A posteriori objective error estimates; [39, Theorems 9 and 10])**.** Let the same definitions as in Lemma 5.2.4 be given, remark that $y \in Q_{\Delta t}^{\mathrm{pr}}(0, T; V_{\mathrm{RB}}^{\mathrm{pr}})$ and $z \in Q_{\Delta t}^{\mathrm{du}}(0, T; V_{\mathrm{RB}}^{\mathrm{du}})$. For $i \in \{1, \ldots, P\}$ it holds that

$$|J(\mu) - \tilde{J}(y; \mu)| \leq E^J(y, z; \mu) \text{ and}$$
$$|(\nabla_\mu J)_i - (\nabla_\mu \tilde{J})_i(y, z; \mu)| \leq E^{\nabla_{\mu_i} J}(y, z; \mu),$$

giving access to a posteriori error estimates for $\tilde{J}(y; \mu)$ and $(\nabla_\mu \tilde{J})(y, z; \mu)$.

Proof. See [39, Theorems 9 and 10]. □

Before going to the optimization we have to do some remarks. Firstly, it holds, due to the argumentation in [39] and [14], that

$$J(\mu) \text{ and } \nabla_\mu J(\mu)$$

and for $u_{\mathrm{RB}}(\mu) = A_{\mathrm{RB}}^{\mathrm{pr}}(\mu)$ and $p_{\mathrm{RB}}(\tilde{\mu}) = A_{\mathrm{RB}}^{\mathrm{du}}(\mu, u_{\mathrm{RB}}(\mu))$ as well

$$\tilde{J}(u_{\mathrm{RB}}(\mu); \mu) \text{ and } \nabla_\mu \tilde{J}(u_{\mathrm{RB}}(\mu), p_{\mathrm{RB}}(\tilde{\mu}); \mu),$$

are Lipschitz continuous with respect to μ. However, for the ML-based solution this might not be the case anymore. Secondly, does the real gradient of $\tilde{J}(y; \mu)$ in general not coincide with $\nabla_\mu \tilde{J}(y, z; \mu)$, if z is not the dual solution associated to y or if $V_{\mathrm{RB}}^{\mathrm{pr}} \neq V_{\mathrm{RB}}^{\mathrm{du}}$. Otherwise, (2.16) provides only a (maybe arbitrary insuffіcent) approximation to $\nabla_\mu J$. For the RB-ROM, constructed by the generator defined in Definition 5.3, both conditions are fulfilled and equality holds. However, this is not the case if we use the ML-ROM to solve the dual problem. The problem here is that we only get an approximation to $p_{\mathrm{RB}}((\mu, y))$, but not the accurate solution, resulting in

$$\nabla_\mu(\tilde{J}(A_*^{\mathrm{pr}}(\mu))) \neq \nabla_\mu \tilde{J}(A_*^{\mathrm{pr}}(\mu), p_{\mathrm{ML}}((\mu, A_*^{\mathrm{pr}}(\mu))); \mu).$$

This is important to keep in mind, because it could lead to erroneous search directions during the optimization.

6. Dual adaptive optimization

6.1. General considerations

With the discussions above we now have the tools at hand to apply the adaptive scheme to Problem 2.1.1, using the primal-dual system for an efficient gradient calculation with a BFGS-TR-model performing the optimization. However, before we can go to the actual implementation we have to explain the general heuristics we want exploit to achieve optimal results from the new surrogates. Moreover, extending the more classical way, presented in [39], by ML-ROMs raises some peculiarities, especially during the backtracking procedure, that have to be taken into account while designing the algorithm.

For an efficient evaluation, we want the ML-ROMs to learn solutions for the next optimization steps from data generated by travelling the trajectory so far. Thereby, the training data is generated, while simultaneously the optimization is performed, eliminating an offline-training phase (for the ML-ROMs). Remark that other designs have been published [34], using ML-ROMs in an adaptive way. Such a layout enforces two important paradigms. The first one is that the models should and could only learn the solutions in a narrow localized region in $\mathcal{P}$, resulting (in general) in a very limited region, where these can be accurate. Nonetheless, the ML-ROMs needed to be able of achieving good results along the trajectory with a relative small amount of these localized data (I). Otherwise, a significant portion of the steps needed to reach the optimum have to be used for training, making the algorithm less efficient than a variant not employing ML-ROM.

A more intricate consequence is that we are constantly evaluating outside of the training distribution. Consider that a model is trained with data for some parameters $(\mu_1, \ldots, \mu_I)$ collected along the trajectory. The next parameter to evaluate μ_{I+1} is in general in a region we do not have "travelled trough" already. So μ_{I+1} lies outside of the distribution of training data and the ML-ROMs have to be evaluated in an "unknown" region of $\mathcal{P}$. This is an important point to remember in the choice of the ML-algorithm and regularization used as well as for the algorithm design in general. We can not realistically expect that the ML-ROM, under these conditions, will learn the quite complex behaviour along the trajectory well for regions far ahead of the path already travelled. The difficulties here are the restrictive amount of training data as well as the fact that we are evaluating in regions without training data at all. Moreover, the general trend of behaviour of the solution manifold might change, while we progressing down to the optimum and we have to adapt.

For these reasons, the main mechanism we want to exploit here, and this is the second paradigm, is to assume that the ML-ROM are only accurate for a limited region ahead the trajectory. Only in this region, they can be used sensibly; thereafter, they have to be retrained, making them accurate on the next segment. This makes constant retraining a necessity. Obviously, this could cause significant overhead to the evaluation procedure. Thus the ML-ROMs used should be able to be (re)trained quickly (II). Moreover, we need to assume good interpolation properties (III) of the ML-ROMs and continuity of the surrogates (IV). Both combined guarantees that the difference between the ML- and RB-solution reduces, while shrinking the distance to the RB-training data points, achieving proper approximation of the RB-solution by the ML-ROM near the training data. Unfortunately, as discussed in Section 1.1, does the interpolation property and generalization affect each other. Therefore, the model and its regularization have to be chosen carefully. Otherwise, the ML-ROMs could not be able to deliver satisfactorial results. In the context of Algorithm 1 the necessity to retrain locally means we have to update the ML-ROMs during solving subproblem (3.1), i.e. in the inner loop (line 4 of Algorithm 1). This however causes problems regarding *consistency and convergence*. There are two main issues here we have to address.

The first one is that after retraining and without further restrictions, we can not expect from a ML-ROM that the predicted values for the same input parameter coincide before and after training. This inconsistency directly affects the convergence of the backtracking procedure. If an ML-ROM is trained during the backtracking, the value of $J^{(i)}(\mu^{(i,l)})$ might not coincide anymore with the value before the training, which could it make impossible to satisfy (3.11) and lead to an infinite iteration and stagnation away from the optimum. Further aggravating is the fact, that this also prevents the ML-ROMs from improving, because no new training data can be generated, if the ROMs will not be called during this backtracking. Additionally to these problems comes, as argued in Section 3.2, that termination of the backtracking is in general not asserted.

The second fundamental problem is that ML-based estimates generally do not meet the interpolation property, i.e. for training data $(\mu^{(i)}, u_{\mathrm{RB}}(\mu^{(i)}))$ it can not be asserted that $u_{\mathrm{RB}}(\mu^{(i)}) = u_{\mathrm{ML}}(\mu^{(i)})$ holds. In fact, regularization usually prevents interpolation of the training data, causing a better global approximation by diminishing accuracy on the training data. This however makes ML-based models for $J^{(i)}$ and its gradient *not infinitely refinable* trough training, thus directly violating the relaxed first-order condition 3.1.1. Consequently, Algorithm 1 can not be directly applicable with ML-based models.

We will, in the following, present an advanced algorithm for finding a local minimizer (3.1), addressing these problems and making the ML-ROM feasible for minimization with a backtracking-like method (s. Algorithm 5). For this optimization we will parallel introduce models for M and $M^{(i)}$ with notations similar to Section 4.2 for the ML-RB-setting outlined in Chapter 5. However, the ideas can be generalized to other ROMs, satisfying the relaxed first order condition 3.1.1. Linchpin of this new algorithm is that the RB-ROM will be approximated by a sequence of iteratively updated ML-ROMs. In this way, the ML-ROMs can be utilized on the one hand for increasing the evaluation speed, while we can always fallback back to the RB-ROM performing more accurate calculations, compared to the ML-ROMs, if necessary.

6.2. Trust region-reduced basis-machine learning optimization

Based upon the prior arguments, we now have to discuss how the models and concepts introduced so far can be fitted into the general setting optimization algorithm. Starting by defining the adaptive frameworks

$$F^{\mathrm{pr}} := \mathrm{AF}\left[G_{\mathrm{RB}}^{\mathrm{pr}}, G_{\mathrm{ML}}^{\mathrm{pr}}\right] \text{ and } F^{\mathrm{du}} := \mathrm{AF}\left[G_{\mathrm{RB}}^{\mathrm{du}}, G_{\mathrm{ML}}^{\mathrm{du}}\right] \tag{6.1}$$

with $G_{\mathrm{RB}}^{*} := \mathrm{ROMGEN}_{\mathrm{RB}}^{*}[M_h^{*}]$ and $G_{\mathrm{ML}}^{*} := \mathrm{ROMGEN}_{\mathrm{ML}}^{*}[M_{\mathrm{RB}}^{*}]$, matching Definitions 5.3.3 and 5.4.2, we get access to the primal and dual trajectories independently. As shown before, does the objective functional (2.10) only depend on the primal solution. Its gradient however depends on both primal and dual solutions for a given μ, where the later one itself is based on the primal trajectory. We therefore need to *couple* the models of the primal and dual branch into one model, giving access to both.

6.2.1. Trust region-reduced basis optimization

We want to employ Algorithm 1, using the projected BFGS-optimization with a trust region given by (3.5). Following the line of argument in Section 3.1, it is sensible to use the FOMs as engine for the high fidelity model M and the RB-ROM and ML-ROM as substitute in $M^{(i)}$. It was also considered to use FOMs in $M^{(i)}$, permitting fallbacks of the RB-ROMs up to them. This approach however would lead to a couple of issues. Foremost, the difficulties in setting up efficient fallback conditions. In the procedure discussed in Section 4.3, these conditions must be based upon the primal and dual solutions individually and not on J, which is in general less efficient, since this might result in costly and unnecessary FOM-evaluations. Consider for example that the error for a primal RB-solution is above a given tolerance but the error for the J based on this solution is still below the according tolerance. Following the fallback idea this would lead to an evaluation of the primal FOM, despite J is still sufficiently accurate. In fact even both FOMs have to be evaluated in this situation for asserting $V_{\mathrm{RB}}^{\mathrm{pr}} = V_{\mathrm{RB}}^{\mathrm{du}}$. Therefore, this idea was rejected and the high fidelity model M and its surrogates $M^{(i)}$ were defined accordingly, such that the FOMs are evaluated in the outer loop only.

Definition 6.2.1. Let the frameworks defined in (6.1) and an objective functional J of the form (5.11) be given. The high fidelity model M is defined by the routines

(i) $(u_h(\mu), p_h(\tilde{\mu})) \leftarrow M$`.eval_state`$[\mu]$: Evaluating the FOMs by calling M_h^{pr}`.eval_state`$[\mu]$ and M_h^{du}`.eval_state`$[\tilde{\mu}]$ with $\tilde{\mu} := (\mu, u_h(\mu))$, returning the primal and according dual trajectories.

(ii) $(J(\mu), \nabla_\mu J(\mu)) \leftarrow M$`.eval_objective`$[\mu]$: Returning the objective functional and its gradient, as defined by (5.11) and (5.12), using the results of M`.eval_state`$[\mu]$.

Taking these considerations into account, the next question is how the RB-ROM and ML-ROM should be evaluated and updated, enforcing the adaptive scheme and circumventing the problems outlined above. Updating the RB-ROM is straightforward, utilizing FOM-solutions from the outer loop and constructing enriched RB-ROMs, with the methods provided by the generator G^{*}_{RB}. Remark that the primal and dual RB-ROM will be updated by the same set of basis vectors, requiring to evaluate both FOMs in tandem. (s. Definition 5.3.3). Following this procedures, both RB-ROMs stay unchanged during the optimization run of the subproblem and thus are not causing issues regarding inconsistencies. As a starting point, we define the RB-ROM-based local surrogates $M^{(i)}$ to M and than discuss the necessary extensions to this model and the TR-algorithm to use the ML-ROM adaptively.

Definition 6.2.2. Let the same assumptions as in Definition 6.2.1 be in force. Let further $u_h(\mu^{(i)}), p_h((\mu^{(i)}, u_h(\mu^{(i)})) \in V_{\mathrm{RB}}^{\mathrm{pr}} = V_{\mathrm{RB}}^{\mathrm{du}}$ be given. The model $M^{(i)}$ in the sense of Chapter 3 is defined by the routines

(i) $(u_{\mathrm{RB}}(\mu), \Delta^{u}_{\mathrm{RB}}(\mu), p_{\mathrm{RB}}(\tilde{\mu}), \Delta^{p}_{\mathrm{RB}}(\tilde{\mu})) \leftarrow M^{(i)}.\texttt{eval\_and\_est\_state}[\mu]$: Evaluating the RB-ROMs and returning the primal and according dual trajectories as well as the error estimates, obtained by Procedure 2.

(ii) $(J^{(i)}(\mu), \Delta^{J^{(i)}}(\mu), \nabla_{\mu} J^{(i)}(\mu), \Delta^{\nabla_{\mu} J^{(i)}}(\mu)) \leftarrow M^{(i)}.\texttt{eval\_and\_est\_objective}[\mu]$: Returning the approximate objective functional and its gradient, defined by

$$J^{(i)}(\mu) := \tilde{J}(u_{\mathrm{RB}}(\mu), \mu) \text{ and } \nabla_{\mu} J^{(i)}(\mu) := \nabla_{\mu} \tilde{J}(\mu)(u_{\mathrm{RB}}(\mu), p_{\mathrm{RB}}(\tilde{\mu}); \mu)$$

as well as the respective a posteriori error estimates

$$\Delta^{J^{(i)}}(\mu) := E^{J}(u_{\mathrm{RB}}(\mu), p_{\mathrm{RB}}(\tilde{\mu}); \mu) \text{ and } (\Delta^{\nabla_{\mu} J^{(i)}}(\mu))_j := E^{\nabla_{\mu_j} J}(u_{\mathrm{RB}}(\mu), p_{\mathrm{RB}}(\tilde{\mu}); \mu),$$

with $j \in \{1, \ldots, P\}$.

To increase the performance, the results of $M^{(i)}.\texttt{eval\_and\_est\_state}[\mu]$ will be cached and reused for other methods.

Procedure 2 $(u_{\mathrm{RB}}(\mu), \Delta^{u}_{\mathrm{RB}}(\mu), p_{\mathrm{RB}}(\tilde{\mu}), \Delta^{p}_{\mathrm{RB}}(\tilde{\mu})) \leftarrow M^{(i)}.\texttt{eval\_and\_est\_state}[\mu]$

```
$(u_{\mathrm{RB}}(\mu), \Delta^{u}_{\mathrm{RB}}(\mu)) \leftarrow M^{\mathrm{pr}}_{\mathrm{RB}}.\texttt{eval\_and\_est\_state}[\mu]$
$\tilde{\mu} \leftarrow (\mu, u_{\mathrm{RB}}(\mu))$
$(p_{\mathrm{RB}}(\tilde{\mu}), \Delta^{p}_{\mathrm{RB}}(\tilde{\mu})) \leftarrow M^{\mathrm{du}}_{\mathrm{RB}}.\texttt{eval\_and\_est\_state}[\tilde{\mu}]$
return $(u_{\mathrm{RB}}(\mu), \Delta^{u}_{\mathrm{RB}}(\mu), p_{\mathrm{RB}}(\tilde{\mu}), \Delta^{p}_{\mathrm{RB}}(\tilde{\mu}))$
```

The TR-algorithm constructs the models $(M^{(i)})_{i \in \mathbb{N}_0}$ iteratively, maintaining local approximations, starting with the model $M^{(0)}$ for an initial guess $\mu^{(0)}$. Definition 6.2.2 shows that for a given $M^{(i)}$ with $i \in \mathbb{N}_0$ the next model $M^{(i+1)}$ can simply be obtained by updating the backbone adaptive frameworks, i.e. extending the RB-spaces by the generator and prolonging the ML-training data. The formal update procedure, called at lines 7 and 11 in Algorithm 1 is therefore given by Procedure 3. From Theorem 5.5.2 and the discussion in Section 5.3 follows that the relaxed first-order condition 3.1.1 will be fulfilled by these models if ϵ_{POD} is sufficiently small, making them a vital cornerstone and critical link to bring ML-ROMs into Algorithm 1, as we will see in the next section.

Procedure 3 Update $M^{(i)}$

Let for $i \in \mathbb{N}_0$, $\mu^{(i+1)}$, $u_h(\mu^{(i+1)})$ a solution to (5.1) for $\mu^{(i+1)}$ and $M^{(i)}$ with F^{pr}, F^{du} be given.

```
$G_{\text{RB}}^{\text{pr}}$.extend[$\mu^{(i+1)}$] and $G_{\text{RB}}^{\text{du}}$.extend[$(\mu^{(i+1)}, u_h(\mu^{(i+1)}))$]
$M_{\text{RB}}^{*} \leftarrow G_{\text{RB}}^{*}$.pre_compute[]
$G_{\text{ML}}^{*}$.prolong[$M_{\text{RB}}^{*}$]
Return $M^{(i+1)}$ as given by Definition 6.2.2 using $F^{\text{pr}}$ and $F^{\text{du}}$.
```

6.2.2. Adaptive machine learning surrogates

For evaluating the ML-ROMs a basic coupled fallback procedure will be used. In general the primal ML-ROM will be tried first and its relative estimated errors calculated. In cases where this error exceeds a given tolerance ϵ^{pr} the algorithm falls back to the associated RB-ROM and this solution will always be accepted. In both cases, the error estimates will be used determining $\Delta^{J^{(i)}}$ and $\Delta^{\nabla_\mu J^{(i)}}$ for assessing the trust region and the termination criteria. The dual solution, based on the primal one, is obtained by applying the adaptive step again with constant ϵ^{du} and primal trajectory get from the first fallback. Additionally to this stepwise procedure, we force the model to use the RB-ROM, if it has not been evaluated for the last `max_mlm`* evaluations. This asserts the generation of new training data and updates the ML-ROM in regular intervals. In the notation of Section 4.2 this procedure is formally defined below in Procedure 4.

Procedure 4 Coupled adaptive fallback

Let F^{pr}, F^{du}, μ and $\epsilon^{\text{pr}}, \epsilon^{\text{du}} > 0$ as well as $\tilde{k}, \tilde{l} \in \mathbb{N}_0$ be given. Define $\Delta^y(\mu) = \Delta^z(\tilde{\mu}) := \infty$.

```
$\tilde{k} \leftarrow \tilde{k} + 1$
if $\tilde{k} <$ max_mlm$^{\text{pr}}$ then
    $(y(\mu), \Delta^y(\mu)) \leftarrow M_{\text{ML}}^{\text{pr}}$.eval_and_est_state[$\mu$]
end if
if $\Delta^y(\mu)/\|y\|_\mu^{\text{pr}} > \epsilon^{\text{pr}}$ or $\tilde{k} =$ max_mlm$^{\text{pr}}$ then
    $(y(\mu), \Delta^y(\mu)) \leftarrow M_{\text{RB}}^{\text{pr}}$.eval_and_est_state[$\mu$]
    $G_{\text{ML}}^{\text{pr}}$.extend[$\mu$]
    $\tilde{k} \leftarrow 0$
    $\tilde{\mu} \leftarrow (\mu, y)$
end if
$\tilde{l} \leftarrow \tilde{l} + 1$
if $\tilde{l} <$ max_mlm$^{\text{du}}$ then
    $(z, \Delta^z(\tilde{\mu})) \leftarrow M_{\text{ML}}^{\text{du}}$.eval_and_est_state[$\tilde{\mu}$]
end if
if $\Delta^z(\tilde{\mu})/\|z\|_\mu^{\text{du}} > \epsilon^{\text{du}}$ or $\tilde{l} =$ max_mlm$^{\text{du}}$ then
    $(z, \Delta^z(\tilde{\mu})) \leftarrow M_{\text{RB}}^{\text{du}}$.eval_and_est_state[$\tilde{\mu}$]
    $G_{\text{ML}}^{\text{du}}$.extend[$\tilde{\mu}$]
    $\tilde{l} \leftarrow 0$
end if
return $(y(\mu), \Delta^y(\mu), z(\tilde{\mu}), \Delta^z(\tilde{\mu}))$
```

As discussed is the ML-RB-fallback based on the errors of the primal and dual solution individually. We have chosen this layout to allow greater flexibility and use ML-methods even if one of the ML-ROM is not sufficiently accurate. On the other hand is the RB-FOM-fallback (to the main loop) controlled by the error $\Delta^{J^{(i)}}$ via the TR-condition (3.14) and this fallback enacts the evaluation of both FOMs and the recreation of both RB-ROMs. This is more efficient than individual fallbacks, because both RB-ROM have to be recreated simultaneously to assert $V_{\mathrm{RB}}^{\mathrm{pr}} = V_{\mathrm{RB}}^{\mathrm{du}}$. The extension is a relative costly step. Therefore and for the reasons outlined in the last subsection, it is more economical to perform the evaluation of both FOMs and use the solutions of both the recreate the reduced spaces.

6.2.3. Machine learning based optimization

This established and the problems discussed above in mind, the next question is how the ML-ROMs should be updated. It was decided to work with two different setups for a reduced model. For the i-th subproblem we first have the already defined $M^{(i)}$, using only the RB-ROMs. Our goal is to build a sequence of surrogates $(M^{(i,l)})_{l\in\{0,\dots,L^{(i)}\}}$ to $M^{(i)}$, evaluating RB-ROMs and ML-ROM. These models are constructed very similar to $M^{(i)}$, but Procedure 2 will be replaced by the coupled fallback outlined in Procedure 4. They will be iteratively constructed starting with $M^{(i,0)} := M^{(i)}$. $M^{(i,l)}$ is used for calculating the search direction $d^{(i,l)}$ by the BFGS-algorithm presented in Section 3.2 and during the backtracking loop. Thereafter, the next model $M^{(i,l+1)}$ is constructed by updating $M^{(i,l)}$ trough retraining the ML-ROMs. In this way consistency can be obtained during one backtracking loop. If no new training data was generated since the last update, hold $M^{(i,l+1)} = M^{(i,l)}$.

Definition 6.2.3. Let the same assumptions as in Definition 6.2.2 as well as $\epsilon^{\mathrm{pr}}, \epsilon^{\mathrm{du}} > 0$ be given and $\tilde{k}, \tilde{l} \in \mathbb{N}_0$ initialized as zero. The model $M^{(i,l)}$ is defined by the routines

(i) $(y(\mu), \Delta^y(\mu), z(\tilde{\mu}), \Delta^z(\tilde{\mu})) \leftarrow M^{(i,l)}$`.eval_and_est_state`$[\mu]$: Performing the evaluation by Procedure 4.

(ii) $(J^{(i,l)}(\mu), \Delta^{J^{(i,l)}}(\mu), \nabla_\mu J^{(i,l)}(\mu), \Delta^{\nabla_\mu J^{(i,l)}}(\mu)) \leftarrow M^{(i,l)}$`.eval_and_est_objective`$[\mu]$: Returning the approximate objective functional and its gradient defined by

$$J^{(i,l)}(\mu) := \tilde{J}(y(\mu), \mu) \text{ and } \nabla_\mu J^{(i,l)}(\mu) := \nabla_\mu \tilde{J}(\mu)(y(\mu), z(\tilde{\mu}); \mu)$$

as well as the respective a posteriori error estimates

$$\Delta^{J^{(i,l)}}(\mu) := E^J(y(\mu), z(\tilde{\mu}); \mu) \text{ and } (\Delta^{\nabla_\mu J^{(i,l)}}(\mu))_j := E^{\nabla_{\mu_j} J}(y(\mu), z(\tilde{\mu}); \mu),$$

with $j \in \{1, \dots, P\}$.

(iii) $M^{(i,l+1)} \leftarrow M^{(i,l)}$`.update`: Retraining the ML- ROMs M_{ML}^* by calling G_{ML}^*`.pre_compute[]`, defining $M^{(i,l+1)}$.

We now have all definitions to expand the optimization of the subproblem (3.1) presented in Section 3.2 by ML-based surrogates. As a first enhancement, we add a third termination criteria to the backtracking, asserting a minimal stepsize. More precisely, the backtracking loop terminates and $\mu^{(i,l+1)}$ is rejected if

$$\|\mu^{(i,l)}(k) - \mu^{(i,l)}\|_{\mathbb{R}^P} < \epsilon_{\text{cutoff}}, \tag{6.2}$$

for a threshold ϵ_{cutoff} (s. Algorithm 5, line 9). If this happens the backtracking will be redone (line 17) using the model $M^{(i)}$, starting again at $\mu^{(i,l)}$. The search direction $d^{(i,l)}$ is also recalculated using the more accurate model $M^{(i)}$. However, remark that the inverse Hessian $\tilde{\mathcal{H}}^{(i,l)}$ will not be reset. If $\mu^{(i,l+1)}$ was rejected, $M^{(i,l)}$ is not updated after the backtracking; an update is only performed if $\mu^{(i,l+1)}$ was accepted, i.e. (3.11) and (3.12) hold. The termination criteria (3.11), (3.12), (3.13) and (3.14) will be calculated based on either $M^{(i,l)}$ or $M^{(i)}$, depending on the situation.

Can by $M^{(i)}$ still no sufficient $\mu^{(i,l+1)}$ be found, satisfying (3.11) and (3.12) but not (6.2), the last accepted parameter $\mu^{(i,l)}$ will be returned as $\mu^{(i+1)}$ to the main loop and the RB-ROMs rebuild. If on the other hand $\mu^{(i,l)}$ will be accepted, the approximative inverse Hessian and the search direction will be updated and the next backtracking starts at $\mu^{(i,l+1)}$. One exception to this layout is the first backtracking for each subproblem. Here $M^{(i)}$ is used directly and no minimal stepsize is demanded. Remark that for the RB-ROMs the initial point $\mu^{(i,0)}$ is by construction an inner point of the trust region, if ϵ_{POD} is small enough. Together with the comments made in Section 5.5 we can apply Lemma 3.2.1, asserting termination for the initial backtracking with arbitrary small stepsize. This procedure avoids stagnation of the convergence caused by (6.2) and generates new, non-prolonged training data after extending the RB-spaces, allowing a better adaption of the ML-ROM to the extended space.

Adding this approach to the general backtracking algorithm, inconsistencies and infinite iterations during the backtracking are avoided by keeping $M^{(i,l)}$ constant and the backtracking procedure is forced to terminate. The reiteration of the backtracking with the RB-ROMs on the other hand provides the possibility for further optimization, even if the ML-ROMs are not sufficient. Remark that at no point we demand the ML-based models to satisfy part two of the relaxed first-order condition. The models could be arbitrarily inaccurate, even in $\mu^{(i)}$. If they are to inaccurate the backtracking is performed by falling back to RB-models, which satisfy Condition 3.1.1 and thus avoiding stagnation and undefined behaviour in the described manner. Remark further, that this does not guarantee convergence of the Algorithm 1 to a first-order critical point, as explained in Section 3.2.

The discussions culminate in Algorithm 5 for solving the i-th subproblem (Algorithm 1, line 4). $O(\mu)$ denotes the output-tuple storing the results from $M^{(i)}$`.eval_and_est_objective`$[\mu]$ or $M^{(i,l)}$`.eval_and_est_objective`$[\mu]$, depending on the model used. This procedure allows use to of the ML-based models for calculation of the trajectories for $(\mu^{(i,l)})_{l\in\{0,\dots,L^{(i)}\}}$ in the inner loop. But importantly we will use them only in the inner loop. In the outer loop the RB-ROM-based model $M^{(i)}$ will be used exclusively, otherwise similar problems as in the inner loop regarding consistency would occur and the necessary (NC$^{(i)}$) and sufficient (SC$^{(i)}$) condition, would not be applicable anymore.

One final note we have to make, before going to the numerical experiments, is that this algorithm still has a potential issue with inconsistencies caused by training the ML-ROMs.

There is the possibility that we achieve $J^{(i,l)}(\mu^{(i,l+1)}) \leq J^{(i,l)}(\mu^{(i,l)})$ by (3.11), but retraining could causes a discontinuity resulting in $J^{(i,l+1)}(\mu^{(i,l+1)}) \geq J^{(i,l)}(\mu^{(i,l)})$. This might in general not be a problem, as long as the objective decays in the long run. And even if this is not the case and using ML-ROMs would result in the situation $J^{(i+1)}(\mu^{(i+1)}) > J^{(i)}(\mu^{(i)})$, the guess $\mu^{(i+1)}$ would be rejected by the conditions in the outer loop. Therefore, we do not handle this case by a particular subroutine.

In the next chapter this algorithm is implemented and applied to a relatively simple constrained optimisation problem, for studying its capabilities and potential further problems.

Algorithm 5 Inner loop

Let a starting point $\mu^{(i)}$ and the local surrogate models $M^{(i)}$ and $M^{(i,0)}$ be given.

```
1: Set $l := 0$, $\mu^{(i,l)} := \mu^{(i)}$, $\tilde{\mathcal{H}}^{(i,l)} := \mathrm{Id}_{\mathbb{R}^P}$, $d^{(i,l)} := -\tilde{\mathcal{H}}^{(i,l)} \nabla_\mu J^{(i,l)}(\mu^{(i,l)})$ and $\tilde{M} := M^{(i)}$.
2: Set no_progress $\leftarrow$ false.
3: $O(\mu^{(i,0)}) \leftarrow \tilde{M}$.eval_and_est_objective$[\mu^{(i,0)}]$
4: while (3.13) or (3.14) is not satisfied or $l = 0$ do
5:     Update $\alpha_0$ (if needed) and set $k := 0$.
6:     Get $\mu^{(i,l)}(k)$ by (3.10).
7:     $O(\mu^{(i,l)}(k)) \leftarrow \tilde{M}$.eval_and_est_objective$[\mu^{(i,l)}(k)]$
8:     while (3.11) and (3.12) are not satisfied do
9:         if $(\|\mu^{(i,l)}(k) - \mu^{(i,l)}\| < \epsilon_{\mathrm{cutoff}})$ and $l > 0$ then
10:            no_progress $\leftarrow$ true
11:            break
12:        end if
13:        Get $\mu^{(i,l)}(k)$ by (3.10).
14:        $O(\mu^{(i,l)}(k)) \leftarrow \tilde{M}$.eval_and_est_objective$[\mu^{(i,l)}(k)]$
15:        $k \leftarrow k + 1$
16:    end while
17:    if no_progress then
18:        if $\tilde{M}$ equal $M^{(i)}$ then
19:            Go to line 34.
20:        end if
21:        $\tilde{M} \leftarrow M^{(i)}$
22:        $O(\mu^{(i,l)}) \leftarrow \tilde{M}$.eval_and_est_objective$[\mu^{(i,l)}]$
23:        Reset $d^{(i,l)}$ by (3.9).
24:    else
25:        no_progress $\leftarrow$ false
26:        $\mu^{(i,l+1)} := \mu^{(i,l)}(k)$
27:        Get $\tilde{\mathcal{H}}^{(i,l+1)}$ by (3.7), (3.8) and (3.6).
28:        Get $d^{(i,l+1)}$ by (3.9).
29:        $M^{(i,l+1)} \leftarrow M^{(i,l)}$.update
30:        $\tilde{M} \leftarrow M^{(i,l+1)}$
31:        $l \leftarrow l + 1$
32:    end if
33: end while
34: $\mu^{(i+1)} := \mu^{(i,l)}$
35: return $\mu^{(i+1)}$
```

7. Numerical experiments

7.1. Implementation

The implementation of the optimizer is based on code[1] built for the adaptive model presented in [20]. The basis of this software is the freely available Python-based library pyMOR[2] [36].

The FOMs are build using `discretize_stationary_cg` from the pyMOR-library, returning an instance of the class `InstationaryModel`. These match the definition of a state-based model 4.2.1, using the method of lines as in the Models 5.1.2 and 5.1.3. Solving these problems is performed by the `LSMR`-algortihm [15] in pyMOR. The procedure is similar for the reduced models. The RB-ROM generator are implemented by the pyMOR-class `ParabolicRBReductor`, executing precomputation in combination with the `inc_vectorarray_hapod`-method, performing the HaPOD-reduction. The image-bases generation for the error estimators are build on the pyMOR-method `estimate_image_hierarchical`. For the dual equation occur the problem, that the reduced dual problem (5.5) needs to be overwritten each time the primal model was updated. This is necessary for ensuring efficient calculations of $d(y, v)$ in the right hand side of (5.5). Furthermore, parallel to the projection of (2.9) and (2.15), the functionals (2.10) and (2.16) are projected to the reduced spaces, to improve the calculation performance. The optimization Algorithms 1 and 5 themselves are also purely Python-based, using the `numpy`-library[3] [23] in version 1.22.0, performing the linear algebra operations.

As discussed, the error estimators depend on the parameter-depended lower coercive $\alpha_{\mathrm{LB}}(\mu)$ and the upper continuity bound $\gamma^{\mathrm{UB}}_{a_{\mu_i}}(\mu)$ of $a(\cdot,\cdot\,;\mu)$ and its derivatives $a_{\mu_i}(\cdot,\cdot\,;\mu)$ as well as the continuity constant γ_d of the bilinear form $d(\cdot,\cdot)$. Hence, we need to provide methods for calculating these factors. An estimation for γ_d can be made by solving the eigenvalue problem

$$\gamma_d^2 = \sup_{v \in V_h} \frac{\langle \mathsf{R}(v), \mathsf{R}(v)\rangle_{V_h}}{\langle v, v\rangle_{V_h}},$$

with $\mathsf{R}(v)$ being the Riesz-representative of $d(v,\cdot)$, i.e. $\langle \mathsf{R}(v), w\rangle_{V_h} = d(v, w)$ for all $w \in V_h$, see [39] for details. The parameter dependence of the first two poses the challenge that we need online-efficient estimator for both. Therefore, we will use the *min-theta* approach as introduced in [19]. Recall that $a(\cdot,\cdot\,;\mu)$ was assumed to be symmetric, affine decomposable and coercive. Additionally, we assume that $\Theta^q_a(\mu) > 0$ and $a^q(v, v) \geq 0$ for all $q \in \{1, \ldots, Q_a\}$ and $v \in V_h$. We say $a(\cdot,\cdot\,;\mu)$ is *parametrically coercive*, allowing us to set

$$\alpha_{\mathrm{LB}}(\mu) = \min_{q \in \{1,\ldots,Q_a\}} \frac{\Theta^q_a(\mu)}{\Theta^q_a(\bar{\mu})} \text{ for all } \mu \in \mathcal{P},$$

---

[1] https://github.com/ftschindler/paper-2022-certified-adaptive-RB-ML-ROM-hierarchy

[2] https://pymor.org/

[3] https://numpy.org/

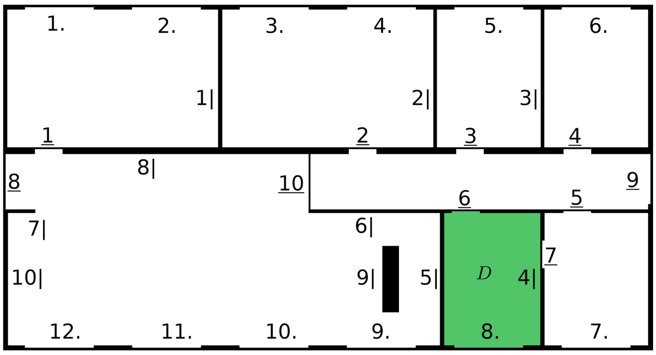


Figure 7.1: On the floor with domain $\Omega := (0,2) \times (0,1)$ heaters below the windows $i.$, doors $\underline{i}$ and walls $i|$ are places. Each has its own parameter μ_* describing the power of the heaters or respectively the heat conductivity of the components. Everywhere else the heat conductivity is fixed as $\mu_{\text{air}} = 0.5$.

with $\bar{\mu} \in \mathcal{P}$ specifying the inner product. In a similar fashion, $\gamma^{\text{UB}}_{a_{\mu_i}}(\mu)$ can be calculated if the bilinear derivatives are parametrically coercive, using the analogue *max-theta* approach.

7.2. Temperature distribution in a building

We will test the new algorithm with a relatively simple problem to discuss fundamental capabilities and limitations of the adaptive ML-RB-optimization. Our goal is to minimize the quadratic misfit between the temperature distribution $u(\mu)$ on a building floor (as shown in Figure 7.1) and a desired state. Even for this relatively simple example complications arise, that have to be addressed before testing more complicated problems. The temperature distribution is governed by the instationary heat equation, which is a parabolic PDE of the form (2.2), with

$$a(u, v; \mu) := \int_\Omega \kappa(\mu) \nabla u(x) \nabla v(x) dx \text{ and } f(v; \mu) := \int_\Omega \eta(\mu) v(x) dx$$

for all $v \in V$. We assume homogeneous Dirichlet condition on the boundary $\partial\Omega$ and set $u^0(\mu) \equiv 0$. The parametric parts $\kappa(\mu)$ and $\eta(\mu)$ are modelling the components in the floor as linear combinations of indicator function with the respective parameter, making them affine decomposable. The diffusion term a is parametrized by the heat conductivity of walls and doors and the source term f by the capacity of the heaters and is factored with $b(t) := \min\{2t, 1\}$, modelling the time the heaters need to rise to their full heating power.

We want to minimize the difference of the temperature on the floor to some reference distribution $g_{\text{ref}} \in Q^{\text{pr}}_{\Delta t}(0, T; V)$. Therefore, we set $\mathcal{L} : V \to \mathbb{D}$ as the identity map on $\mathbb{D} := L^2(\Omega)$. The space $\mathbb{D}$ will be equipped with the inner product $(\cdot,\cdot)_{\mathbb{D}} := \frac{10^4}{2}(\cdot,\cdot)_{L^2(\Omega)}$. A FOM M_h, as defined in Section 5.1, is given by a triangular grid on Ω, with $\dim V_h = 19401$ degrees of freedom. The temporal grid consists of 501 grid points ($K = 500$), with uniform distances of $\Delta t = 1/10$. The reference g_{ref} is a solution of (5.1) for a fixed parameter tuple $\hat{\mu} \in \mathcal{P}$, determined by M_h, i.e. $g_{\text{ref}} := u_h(\hat{\mu})$. The misfit of g_{ref} to $u_h(\mu) := A^{\text{pr}}_h(\mu)$ will be regularized with quadratic distance to $\hat{\mu}$. Thus the objective functional is given by

$$J(\mu) = \Delta t \frac{10^4}{2} \sum_{k=1}^{K} \|u^k_h(\mu) - g^k_{\text{ref}}\|^2_{L^2(\Omega)} + \kappa \sum_{i=1}^{P} \|\mu_i - \hat{\mu}_i\|^2_{\mathbb{R}^P}.$$

The convexity of this functional ensures the existence of a unique minimum at $\hat{\mu}$.

As ML-model we will use the aforementioned kernel-based models from Section 1.2, with

VKOGA in a P-greedy approach as the learning algorithm, implemented by the Python-based VKOGA-library[4]. Both for primal and dual, the models will be defined in the exact same way. The VKOGA-models are constructed using a time-vectorized layout from Definition 5.4.2, i.e. T^* returns for a given parameter the DoF-vectors for all timesteps simultaneously. We will study the behaviour of two different well known matrix-valued kernel-functions each with different parameter and input-transformation Φ^*. Our focus is to investigate how these methods perform on this task and how this affects the optimization. Assuming that a RB-space with $\dim V^*_{\mathrm{RB}} = N^*_{\mathrm{RB}}$ and K timesteps are given, a linear and a Gaussian kernel are given by

$$K_{\mathrm{Lin}}(x,y) = x^T y \mathrm{Id}_{\mathbb{R}^{KN^*_{\mathrm{RB}}}} \text{ and } K_{\mathrm{Gauss},\sigma}(x,y) = \exp\bigl(-\sigma^2 \|x-y\|^2_{\mathbb{R}^P}\bigr)\mathrm{Id}_{\mathbb{R}^{KN^*_{\mathrm{RB}}}}.$$

It is easy to verify, that both are strictly positive definite kernels and thus define a unique RKHS. The VKOGA-algorithm returns a linear combination T^* of the form (1.6) with $(\mu_i)_{i\in I_M} \subset \mathcal{P}$ selected from a training tuple $z^* \subset \mathcal{P} \times \mathbb{R}^{KN^*_{\mathrm{RB}}}$. The solution operator is therefore given for the primal case by

$$\begin{aligned} A^*_{\mathrm{ML}} : \mathcal{P} \to Q^*_{\Delta t}(0,T;V^*_{\mathrm{RB}});& \\ \mu \mapsto u^*_{\mathrm{ML}}(\mu) &:= (u^{*,k}_{\mathrm{ML}}(\mu))_{k\in\{0,\dots,K\}}, \end{aligned}$$

with the state-vector $u^{*,k}_{\mathrm{ML}}(\mu)$ at time t^k completely defined by the DoF-vector

$$\underline{u^{*,k}_{\mathrm{ML}}(\mu)}_n = \begin{cases} \Bigl(\sum_{i\in I_M} \alpha_i K_\diamond(\Phi^*(\mu), \Phi^*(\mu_i))\Bigr)_{(k-1)N^*_{\mathrm{RB}}+n}, & \text{if } k \geq 1 \\ 0, & \text{else,} \end{cases} \tag{7.1}$$

with $\alpha_j \in \mathbb{R}^{KN^*_{\mathrm{RB}}}$ and a suitable input-transformation Φ^*. The dual case is analogue. For Φ^* we investigate data-normalization to the range $[-1,1]$, i.e. a parameter $\mu \in \mathcal{P}$ is mapped to $2(\mu_i - L_i)/(U_i - L_i - 1) \in [-1,1]$ for $i \in \{1,\dots,P\}$ and the identity on $\mathcal{P}$. The bounds L_i and U_i are the one from Section 3.2. A ML-ROM generator adapting Definition 5.4.2 to VKOGA can be obtained by the following definition.

Definition 7.2.1. The VKOGA generator is given by the methods

1. G^*_{ML}`.extend`$[\mu]$: The training data will be collected as outlined in Definition 5.3.3.

2. $M^*_{\mathrm{ML}} \leftarrow G^*_{\mathrm{ML}}$`.precompute`$[]$: As explained in Section 5.4, the key part of precomputation for a ML-ROM is the (re)training by $\texttt{train}_{\mathrm{VKOGA}}[\Lambda^*(\tilde{Z}^*)]$. The input-transformations for the different models are set as above. The data-transformation Λ^* however eliminates duplicates, as outlined in Definition 5.4.2, followed by swapping the order of the training data. This places the last recorded data at the top of the training tuple. The rationale behind this measure is that VKOGA preferably picks the first data points for I_M. Swapping them forces VKOGA to use the points nearest to our current position first. After applying Λ^* and Φ^*, the `train`-routine selects the indices I_M of the most important points, by a P-greedy approach [44, p. 19], obtaining optimal coefficients $\alpha := \{\alpha_i\}_{i\in I_M}$ for (7.1).

---

[4] https://gitlab.mathematik.uni-stuttgart.de/pub/ians-anm/vkoga

3. $G^*_{\mathrm{ML}} \leftarrow G^*_{\mathrm{ML}}.\texttt{prolong}[M^*_{\mathrm{RB}}]$: Prolongation is realized, as in Definition 5.3.3, via padding of the DoF-vectors.

To investigate the capabilities of the ML-ROMs, their behaviour will be compared to an analogue optimization employing only the RB-ROM. For this run some minor adjustments are made to the algorithm Firstly, $M^{(i,l)}$ is set equal to the RB-based model $M^{(i)}$ for all indices. Secondly, the line 29 in Algorithm 5 is skipped. This optimization will be referenced as RB-O.

Before delving into the numerical experiments, let us briefly discuss why VKOGA-based models are a suitable choice, considering the requirements (I) - (IV) outlined in Section 6.1. The kernel-based models have a relatively low number of trainable parameters, making it more likely to achieve good results with fewer data. This and the iterative procuedure outlined in Section 1.2 allows relatively fast retraining. The surrogates are continuous, due to the continuous kernels. Finally, if no regularization is performed, the returned surrogate interpolates on the training point selected by VKOGA. However, if regularization is applied, this statement does in general not hold true. Furthermore, it is not asserted that the numerical algorithm returns the solution (1.5), for example due to numerical errors. These properties, particularly the first two, make kernel-based models advantages against e.g. ANNs for this task.

7.3. Optimal heater capacity

The optimization problem we are considering is to find optimal heater settings while keeping the heat conductivity of the other structural facilities fixed. To solve this problem, we are employing the BFGS-TR algorithm from Section 3.2. The subproblem there is solved by Algorithm 5.

The heat conductivity of the outer walls and doors $(\mu_{7|}, \mu_{10|}, \mu_{\underline{8}}, \mu_{\underline{9}})$ is set to $2.7 \cdot 10^{-2}$. On the remaining walls, it is set to 10^{-2}, and on the remaining doors to $2.5 \cdot 10^{-3}$. For simplicity, the capacities for all heaters on one side of the building $(\mu_{1.}, \ldots, \mu_{6.})$ are controlled by one parameter and all heaters on the other side $(\mu_{7.}, \ldots, \mu_{12.})$ by another, each with a value in the range $[0, 100]$. Thus, we are studying a two-dimensional optimization problem $\mathcal{P} := [0, 100] \times [0, 100] \subset \mathbb{R}^2$ with two sets of heaters and fixed components otherwise.

The energy product is constructed with $\bar{\mu} = (50, 50)$ and the desired state is determined for $\hat{\mu} = (50, 50)$. It was verified, by using the FOMs, that $\hat{\mu}$ is indeed a first-order critical point. The objective functional is regularized with the constant $\kappa = 5 \cdot 10^{-5}$ and the VKOGA-models with $\lambda^{\mathrm{pr}} = 10^{-11}$ for the primal and $\lambda^{\mathrm{du}} = 10^{-9}$ for the dual models. The l^2-tolerance ϵ_{POD} used in the RB-ROM generator is set to 10^{-13} and thus above machine precision. This reduces the problem that the image-vectors of the error estimator, calculated during the precomputation of the RB-models (s. Def. 5.3.3), become almost linear dependent, resulting in costly orthonormalization. We demand that the approximative minimizer $\mu^{(\iota)}$ satisfies the termination criteria for $\tau = 10^{-3}$ and $\tau_{\mathrm{sub}} = 5 \cdot 10^{-4}$. The other remaining constants are displayed in the table below.

Parameter	ϵ_L	β	κ_{tr}	$\epsilon_{\mathrm{cutoff}}$	ϵ^{pr}	ϵ^{du}	`max_mlm`*
Value	0.01	0.95	0.5	10^{-5}	10^{-2}	10^{-4}	25

Table 7.1.: Parameters used for the optimization of the heater capacity.

For backtracking, the initial stepsize α_0 is reset after excepting $\mu^{(i,l)}$, depending on the current gradient, given by

$$\alpha_0 := \begin{cases} \alpha_{\min}, & \text{if } \|J^{(i,l)}(\mu^{(i,l)})\| \leq \alpha_{\min} \\ \|J^{(i,l)}(\mu^{(i,l)})\|, & \text{if } \alpha_{\min} \leq \|J^{(i,l)}(\mu^{(i,l)})\| \leq \alpha_{\max} \\ \alpha_{\max}, & \text{if } \alpha_{\max} \leq \|J^{(i,l)}(\mu^{(i,l)})\|, \end{cases}$$

with $0 < \alpha_{\min} \leq \alpha_{\max}$. All calculations were run on a custom PC with AMD Ryzen 7 3700X 8-Core Processor with 16 cores and 32GB RAM.

7.3.1. Convergence & used models

The primary objective of this experiment is to compare the algorithm's performance for different kernels and transformations. Our main focus is on determining whether the algorithm converges to $\hat{\mu}$ and, if so analysing the time required in comparison to RB-O. Additionally, we aim to identify the models that were utilized in the process. The settings chosen are shown in the following table:

Experiment	Kernel	Constants	Φ^{pr}	Φ^{du}
I	Gauss	$\sigma^2 = 5$	Normalize	Normalize
II	Gauss	$\sigma^2 = 0.5$	Normalize	Normalize
III	Gauss	$\sigma^2 = 0.05$	Normalize	Normalize
IV	Linear	–	$\text{Id}_{\mathcal{P}}$	$\text{Id}_{\mathcal{P}}$
V	Linear	–	Normalize	Normalize
VI	Linear	–	$\text{Id}_{\mathcal{P}}$	N

Table 7.2.: Overview over the different kernels and transformations used in the heater capacity experiments.

The constants for determining the stepsize were set as $\alpha_{\min} = 0.1$ and $\alpha_{\max} = 0.7$. Ten random initial guesses $\mu^{(0)}$ are uniformly sampled from $\mathcal{P}$ and for each experiment ten optimizations were performed, starting at these initial guesses, by Algorithm 1. The initial guesses remain the same for each experiment. The most relevant results are shown in Tables A.1 and A.2 in the appendix. For the described experiments, we observe from Table A.1 that all runs converge to $\hat{\mu}$ up to a tolerance and the last recorded gradient satisfies the global termination criteria (3.15) with $\tau = 10^{-3}$. In some runs the last local termination criteria (3.13) is not triggered for $\tau_{\text{sub}} = 5 \cdot 10^{-4}$, this results from the termination of the inner loop by line 19 in Algorithm 5.

However, we observe also that only the experiments with a linear kernel (IV - VI) have a shorter runtime than RB-O, but the speed up there is relatively small. To understand this behaviour, we have to analyse which models were used during the optimizations. Table A.2 shows the average evaluation number per model (M_h^*, M_{RB}^* and M_{ML}^*) for each experiment. We see there that all experiments need more total evaluations than the RB-O to reach the optimum. In particular, the number of evaluations of the RB-ROMs are only for some linear kernel significantly lower. The increased number of models evaluated obviously affects the performance negatively, since more calculations need to be performed.

As presented in Chapter 6 two different fallback mechanisms are active, increasing the number of model evaluations. Firstly, in Procedure 4, if the relative estimated error is above a given tolerance ϵ^*. Secondly, in Algorithm 5, due to the cutoff-condition in line 9, forcing the use of the RB-ROMs for the next backtracking. An analysis of the data reveals that for our setting the first one is far more important[5]. As an example we present in Figure 7.2 the runtimes and models used during the optimization starting at $\mu^{(0)} = (2.0584, 96.9910)$ for experiments II (left) and VI (right). Each vertical line represents a call of a model M, $M^{(i)}$ or $M^{(i,l)}$ along the optimization trajectory. The times used for the individual submodels M^*_{ML}, M^*_{RB} and M^*_h are color-coded.

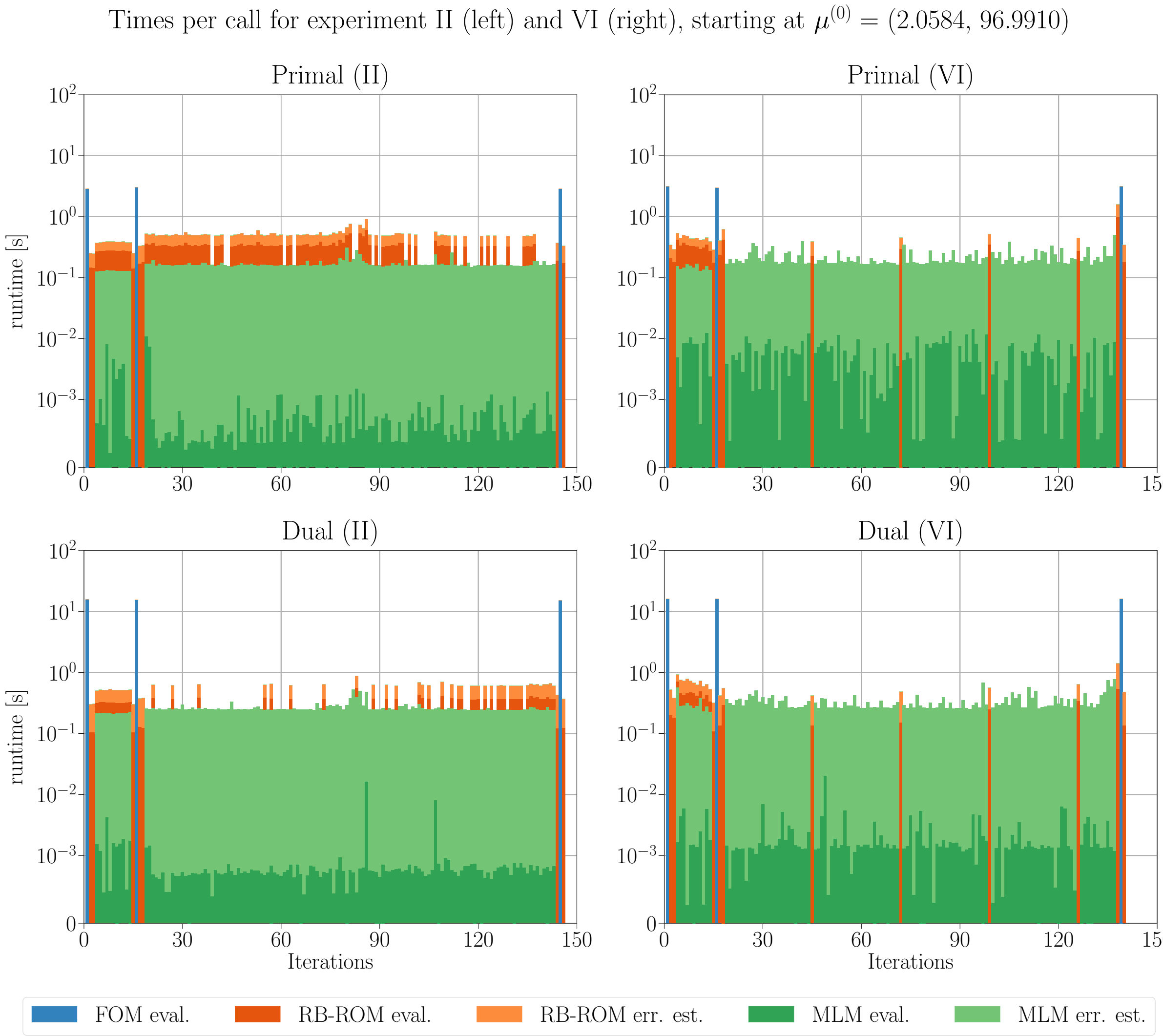


Figure 7.2.: Times per call for estimating state (darker colors) and errors (lighter colors) along the trajectory starting at initial guess $\mu^{(0)} = (2.0584, 96.9910)$. In each iteration (x-axis) one of the models M, $M^{(i)}$ or $M^{(i,l)}$ was evaluated. The color indicates which of the three stages (M^*_h, M^*_{RB} and M^*_{ML}) was used, FOM (blue), RB-ROM (orange), ML-ROM (green). Plotted are experiments II (left) and VI (right). For each the primal evaluations (top) and dual evaluations (bottom) are shown.

---

[5] The second mechanism becomes usually more important, if smaller tolerances for τ and τ_{sub} are chosen, due to the higher accuracy the ML-ROMs need to reach near the optimum.

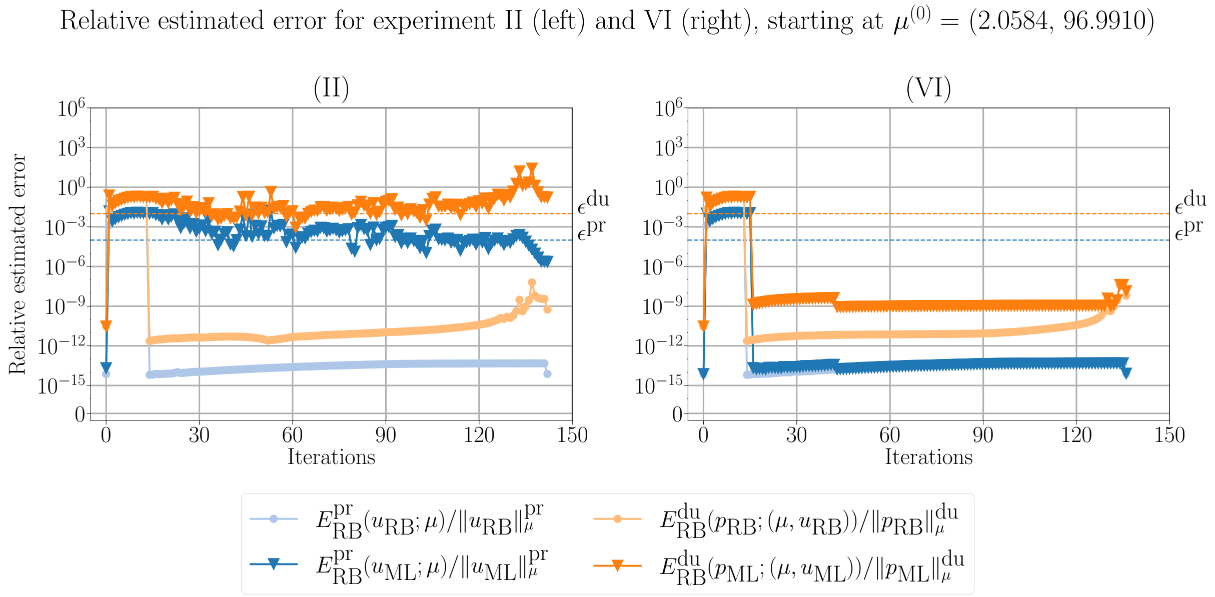


Figure 7.3.: Relative estimated errors for the RB-ROM M^{*}_{RB} (light) and the ML-ROM M^{*}_{ML}(dark) for the optimization starting at $\mu^{(0)} = (2.0584, 96.9910)$. These models are evaluated for all parameter used during the optimization, after each call of $\tilde{M}$`.eval_and_est_objective` in Algorithm 5. For both models the primal (blue) and dual (orange) errors are shown. Plotted are experiments II (left) and VI (right). The error tolerances $\epsilon^{\mathrm{pr}} = 10^{-4}$ and $\epsilon^{\mathrm{du}} = 10^{-2}$ are shown as dashed lines.

We see there that for a significant portion of the evaluations in experiment II the primal ML-ROM was evaluated first and a fallback to the RB-ROM occur thereafter, induced by Procedure 4. For experiment VI on the other hand, all solutions of the ML-ROM are excepted after the second FOM-evaluation. Obviously, the primal ML-ROM for experiment II was not capable of achieving relative errors below $\epsilon^{\mathrm{pr}} = 10^{-4}$. The estimated errors for both runs are shown in Figure 7.3[6], there we see this behaviour reflected. The estimated errors of the both ML-ROMs for experiment II are several orders of magnitude larger than for the RB-ROM. For experiment VI the difference is more modest, particularly for the primal model. The results for the other experiments are comparable. In general, the linear kernel perform depending on the transformation and the initial guess, better than the Gaussian kernel.

This discussion shows that for the setting used the presented algorithm converges, returning a correct optimizer and utilizes the ML-ROMs. However, for most of the ML-ROMs used, their approximations have a significantly larger error with respect to the FOM-solutions than the ones of the RB-ROMs and are not sufficient with respect to the given error tolerances ϵ^{*}. The algorithm therefore falls for most of the evaluations back to the RB-ROMs and utilizes their solutions for the optimization. This reduces the efficiency drastically due to the induced overhead by evaluating insufficient models.

In general it is not clear how accurate the ML-ROMs need to be to achieve converges. Moreover, this might also change during the optimization and the tolerances ϵ^{*} should be adapted (see below). But obviously, could the high errors compared to the RB-ROMs cause some fundamental problems, since the ML-ROMs can not achieve the same approximation quality than the

---

[6] In Figure 7.3 the errors of the dual ML-ROM are based on solutions of the primal ML-ROM. Therefore, the errors do not match exactly with the ones actually calculated during the optimization, but the general trend becomes clear.

RB-ROMs, which, however, might be necessary for for replacing them during the optimization. Thus the quality of the ML-approximation is an important factor determining the efficiency, and the ML-algorithms and their parameters have to be chosen carefully. For our examples, we see that the Gaussian-kernels are not a suitable choice for this algorithm; we will comment in Subsection 7.3.3 possible reason for this behaviour, but a satisfactory explanation can not be given yet.

We have to make two additional comments here. Firstly, in Figure 7.3 we see that for the initial iterations the errors of the RB-ROMs and ML-ROMs are in the same order of magnitude, but above the fallback tolerances ϵ^*. It is likely that the solutions of the ML-ROMs were sufficiently accurate but following Procedure 4, they were rejected. This shows that fallback conditions with fixed tolerances ϵ^* are not optimal, since this could lead to over-rejecting sufficient solutions, as seen here, or under-rejecting insufficient ones, reducing the efficiency in both cases. A better choice would be to adapt ϵ^* during the optimization. This could for example be done either by coupling them to the errors of the RB-ROMs or by increasing respectively reducing them iteratively if the ML-solution is accepted or rejected.

The second point refers back to Table A.2. We see there that, during the optimization, the FOMs are on average more often evaluated in experiments I - III than for RB-O. This increased calling rate can be traced back to the, on average, higher errors of the ML-estimates, raising also the estimated errors of $J^{(i)}$ and $\nabla_\mu J^{(i)}$. Therefore, ML-based solutions reach the boundary of the trust region, given by (3.5), faster than for the RB-ROMs, forcing a return to the main loop and thus making it more likely to evaluate the FOMs. This also has a significant negative impact on the total runtime. It should therefore be considered to tighten the termination criteria (3.13) and (3.14), preventing unnecessary termination due to ML-solutions.

The points highlighted in this section alone do not explain why the speed up for experiments IV to VI are relatively meagre. An additional factor reducing the efficiency result from the error estimator. We will discuss this in the next subsection.

7.3.2. Error estimator performance

As already visible from Figure 7.2, contribute the error estimation of the reduced models mainly to the overall calculation time. This diminishes the potential gain in computational speed drastically because the RB-ROMs and the ML-ROMs use the same error estimator. Table 7.3 presents the times needed to perform an evaluation and error estimation averaged over the ten runs for experiment VI. The proportions are comparable for the other experiments. The speed ups are calculated relative to the model next higher in the hierarchy from Section 4.3, i.e. ML-ROM relative to RB-ROM and RB-ROM to FOM.

We can observe that the speed ups for evaluating the ML-ROMs compared to the RB-ROMs are in the order of magnitude 10^1 to 10^2 (45.82 (primal) and 55.94 (dual)), showing that ML-models are indeed capable of providing a significant performance increase. However, the impact of this is drastically diminished, if the (for every evaluation mandatory) error estimation is factored in. As expected, the error estimation runtime for ML- and RB-ROMs are comparable, but by a factor of approximate 10^2 slower than the evaluation, making the error estimation a major bottleneck. Calculating the speed ups between the RB-ROMs and the ML-ROMs based

Primal	eval. [s]	speed up	err. est. [s]	speed up	total [s]	speed up	build [s]
ML-ROM	$4.50 \cdot 10^{-3}$	45.82	$1.91 \cdot 10^{-1}$	0.70	$1.95 \cdot 10^{-1}$	1.74	$3.72 \cdot 10^{-3}$
RB-ROM	$2.06 \cdot 10^{-1}$	14.63	$1.34 \cdot 10^{-1}$	–	$3.40 \cdot 10^{-1}$	8.87	$5.09 \cdot 10^{1}$
FOM	$3.02 \cdot 10^{0}$	–	–	–	$3.02 \cdot 10^{0}$	–	–
Dual							
ML-ROM	$2.78 \cdot 10^{-3}$	55.94	$3.14 \cdot 10^{-1}$	0.80	$3.16 \cdot 10^{-1}$	1.29	$6.39 \cdot 10^{-2}$
RB-ROM	$1.56 \cdot 10^{-1}$	103.31	$2.52 \cdot 10^{-1}$	–	$4.08 \cdot 10^{-1}$	39.41	$5.09 \cdot 10^{1}$
FOM	$1.61 \cdot 10^{1}$	–	–	–	$1.61 \cdot 10^{1}$	–	–

Table 7.3.: Average times over all ten test runs for model evaluation, error estimation and model building, separated by models and branch (primal/dual). The total time refers to the average time for evaluation and error estimation combined. Building times for the ML-ROMs are identical to the training times. For the RB-ROMs this is the time used to extend V_{RB}^* and projecting the operators to the new basis, reconstructing both error estimator and proloning the training data. The reduced models $M_{\mathrm{RB}}^{\mathrm{pr}}$ and $M_{\mathrm{RB}}^{\mathrm{du}}$ are constructed with the same vector space, so only one extension is actually performed and the build of both RB-ROMs is run as one block.

on times for evaluation and error estimation combined (total) we get 1.74 and 1.29, respectively.

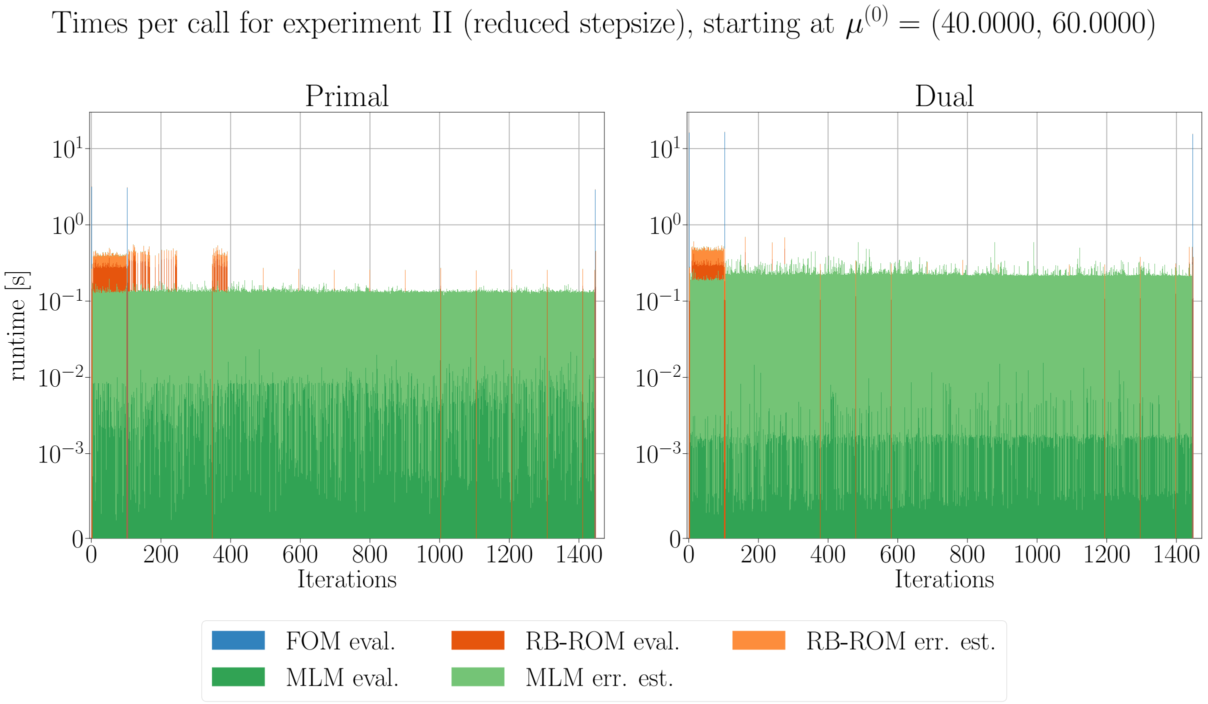
The significant difference in computational speed between evaluating with ML-ROMs and error estimations becomes clear by comparing the calculations carried out. The evaluation of the kernel $K_\diamond(\Phi^*(\mu), \Phi^*(\mu_i))$ for $i \in I_M$ is in general not time-critical, in particular, because the number of selected data points $|I_M|$ for our setting is usually small (≤ 100). This is followed by a linear combination of vectors with KN_{RB}^* elements. The error estimation, on the other hand, performs for each time step several N_{RB}^*- and $|\mathcal{B}^*|$-sized vector-operators, resulting in a higher computational complexity than the evaluation.

Another point, we have to briefly discuss here are the building times of the models, i.e. the times to set up the models and make them operational. For the ML-ROMs, these correspond largely to training time. As discussed above, one reason for choosing the VKOGA-engine for the ML-ROMs are the short training times. Indeed, we can observe that the training takes on average less time than one evaluation of a RB-ROM and is therefore reasonably fast. Building the RB-ROMs includes all necessary steps to extend the models and error estimators with new FOM-solution. This is the most expensive single contribution to the overall calculation time.

The relatively high error estimation time is a tremendous limiting factor of this certified ML-model. This reduces not only the gain that can be expected from the ML-ROMs, but it reduces the also leeway of the algorithm. Remember that the usage of ML-ROMs comes with some overhead compared to a RB-only implementations. This overhead is either inherently given by the ML-ROMs or caused by the problems discussed in Section 7.3.1. To be more efficient, the ML-ROMs have to compensate this by lower evaluation time. The high cost in error estimation, however, shrinks the overhead that can be taken before becoming less efficient than a classical variant. It should be considered either to increase the performance of the error estimator of the ML-ROMs or to redesign the optimization algorithm such that error estimation of the ML-ROMs is not required. A possible measure to increase the performance could be for example to parallelize them, calculating the errors for each timestep simultaneously.

7.3.3. Stepsize dependency

We want at last discuss one further point. As argued in Chapter 6, the continuity and (approximative) interpolation propriety of the kernel surroagte should assert that the errors of the ML-ROMs reduce if the distance to the nearest training data point is reduced. For testing this we use the same configuration as in experiment II but choosing the stepsize by $\alpha_{\text{min}} = 0.001$ and $\alpha_{\text{max}} = 0.01$ and taking the initial guess $\mu^{(0)} = (40.0000, 60.0000)$. In Figure 7.4, runtimes for the primal (left) and dual (right) problems are plotted, analogue to Figure 7.2. Additionally, are the relative estimated error for the primal and dual models are shown in Figure 7.5[7].



Times per call for experiment II (reduced stepsize), starting at $\mu^{(0)} = (40.0000, 60.0000)$

Primal

Dual

Figure 7.4.: Times per call for estimating state (darker colors) and errors (lighter colors) along the trajectory starting at initial guess $\mu^{(0)} = (40.0000, 60.0000)$. In each iteration (x-axis) one of the models M, $M^{(i)}$ or $M^{(i,l)}$ was evaluated. The color indicates which of the three stages (M^*_{ML}, M^*_{RB} and M^*_h) was used, FOM (blue), RB-ROM (orange), ML-ROM (green). Plotted is experiment II with reduced stepsize. The primal evaluations are shown on the left and dual evaluations on the right.

The algorithm converges to $\hat{\mu}$ up to a tolerance of $7.12 \cdot 10^{-3}$ and we see that after the second FOM-evaluation the fallback in Procedure 4 was enacted only a few times. The degree of rejected ML-estimates is far lower than for experiment II. From Figure 7.5 follows that the smaller stepsize in combination with the forced RB-ROM-calls after each 25 iterations ensure that the relative estimated error are below the given thresholds ϵ^*. Each time the RB-ROMs were called and the ML-ROMs were retrained the error drops as expected. Due to the (approximative) interpolation propriety the ML-solutions sufficiently approximate the RB-solutions for a μ in the training data selected by VKOGA. The swap of the data asserts that the latest recorded data point is selected by VKOGA. The reduced stepsize and the continuity of (7.1) asserts that the errors rise more modest compared to Subsection 7.3.1 and stay below ϵ^*. This is an indication that the heuristic outlined in Section 6.1 applies and that the ML-ROMs have indeed be retrained at a relative

---

[7] The dual errors of ML-ROM are based on primal solutions and therefore the errors do not match exactly with the ones actual calculated during the optimization, but the general trend becomes clear.

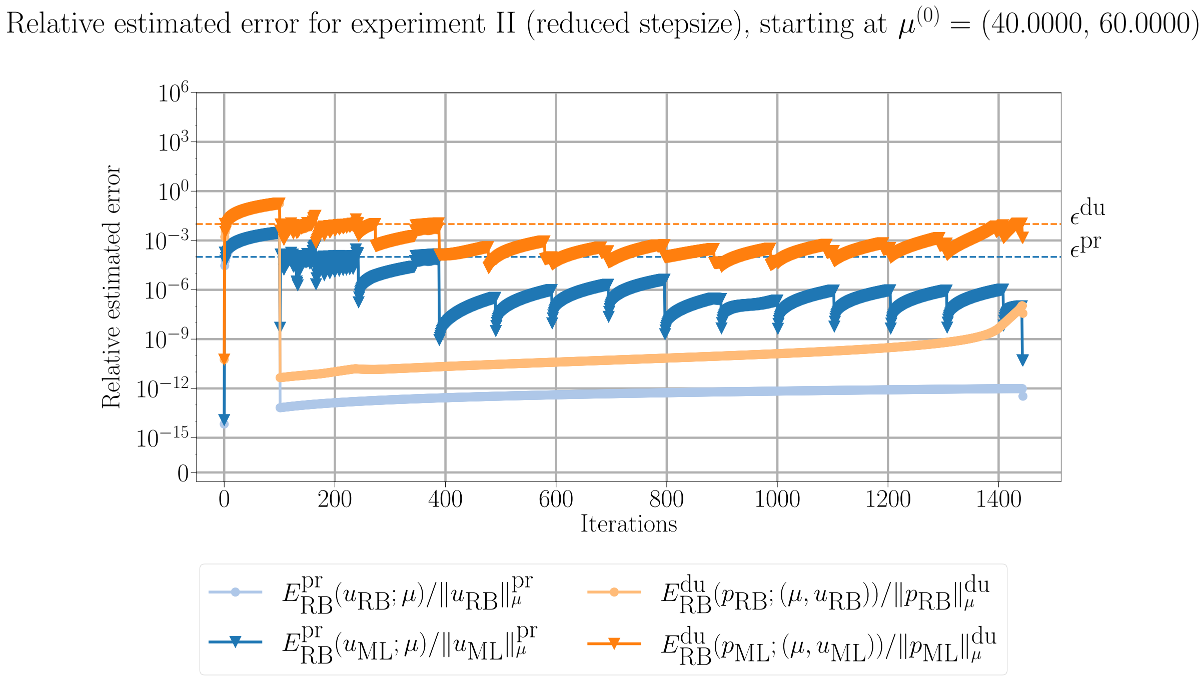


Figure 7.5.: Relative estimated errors for the RB-ROM (light) and the ML-ROM (dark) for the optimization starting at $\mu^{(0)} = (40.0000, 60.0000)$. These models are evaluated for all parameter used during the optimization, after each call of $\tilde{M}$`.eval_and_est_objective` in Algorithm 5. For both models the primal (blue) and dual (orange) errors are shown. Plotted is experiments II with reduced stepsize. The error tolerances $\epsilon^{\text{pr}} = 10^{-4}$ and $\epsilon^{\text{du}} = 10^{-2}$ are shown as dashed lines.

high frequency, for ensuring good local approximation properties outside the training dataset, as outlined in Section 6.1.

However, due to the reduced stepsize, the total number of evaluation increases drastically. As a consequence, the total runtime is 1224.11 seconds, and thus significantly higher than the runtimes needed in the optimizations before. This points to another issue in using the kernel-methods. The range ahead along the optimization trajectory, where the ML-ROMs are accurate enough, is for the configuration used rather limited such that the stepsize have to be chosen accordingly small. Therefore, the stepsize that can effectively be used after extending/training is much smaller than for the RB-ROMs, which is a very limiting factor for efficient optimization. This also highlights that the effective quality of the ML-ROMs depend heavily on the parametrization of the optimization algorithm and not alone on the ML-algorithm used.

As shown in the experiments from the Section 7.3.1, the effect of the stepsize cannot be compensated for by changing the kernel width, at least not for the configurations tested. A satisfactory explanation for this can again not be given yet. However, it is important to notice that altering the kernel may introduce secondary effects that could potentially degrade the quality of the ML-ROMs and might nullify any a positive effect. For instance, it has been observed that the condition number of the matrix A_M in (1.5) is significantly influenced by the choice of kernel and input-transformation. A high condition number, however, can diminish the quality of the approximation due to numerical imperfections. Additionally, other factors such as the distance between training data points, controlled among other things by the stepsize, may also play a role.

These discussions show that choosing suitable parametrization for the ML-ROMs and the algorithm that leads to increased performance is a non-trivial task due to the different effects of the learning algorithm and data generator during the optimization that play together. Further research is required for getting a better understanding of this situation, allowing us to make the ML-based optimization more reliable.

8. Conclusion & Outlook

This thesis was aimed at making machine learning models applicable to optimization problems constrained by parabolic PDEs, by incorporating them into an established Trust Region-based optimization, which employs Reduced Basis-methods for model order reduction. An essential aspect of this design was its adaptability to different ML-methods, providing a high level of flexibility.

The algorithm we have developed uses the hierarchical scheme and fallback procedure presented in [20] as vital link to utilize ML-methods into the algorithms introduced in [27, 39]. This procedure does not enact specific assumptions on the ML-algorithm used, bringing the freedom of choosing an algorithm most appropriate for the task. Moreover, this framework allows to extend the scope of the error estimators, developed for the RB-method, to reduced models build with ML-algorithms mapping into the same state-space as the RB-ROM. This is an important step, since TR-methods with error based regions, as in [39], require the existence of efficient error estimator. Furthermore, this hierarchical framework directly implements an adaptive scheme making the integration into the TR-optimization as well as the inclusion of offline-preparations for the models straightforward. This also eliminates the need for a costly offline-training phase, as the ML-ROMs can be trained on demand using data generated during the optimization process. The evaluation for new parameters of the models is performed by an error-based fallback procedure using the most efficient, yet accurate model available.

Although this framework was easy to adapt into the primal-dual system used for calculating the gradient, some additional challenges arose when combining backtracking and machine learning. These challenges stem from the iterative retraining process conducted while solving localized subproblems, potentially leading to stagnation away from the optimum. However, the quality of the ML-approximation depends critically on this retraining. Therefore, the TR-algorithm is extended by a secondary adoption step, performing only uncritical updates, and an additional fallback step to assure termination of the backtracking procedure and to allow further updates of the models.

We have tested this novel algorithm with a relatively simple problem, determining an optimal heater setting defined by a pre-set state and with a constraint governed by a heat equation. The machine learning model was implemented using kernel-based surrogates with different kernel-functions and input-transformations applied to it and training was performed by a greedy algorithm. For all ML-models and initial guesses tested, convergence to the desired parameters (up to a tolerance) was observed. This shows that this algorithm is able to perform optimization, employing machine learning-based reduced order models. However, the portion of evaluations for them the ML-solutions are actually considered to be sufficiently accurate depends heavily on the parametrization of the ML- and TR-algorithm. This behaviour has an direct impact onto the second question studied in this thesis.

One of the main points of interest in adding ML-methods to the TR-RB-optimization is that this could significantly reduce the runtime needed to perform the optimization. The experiments however have shown that for the problem studied here no notable speed up was achieved. The analysis of the data have revealed that the evaluation performed by kernel-based ROMs is significant faster than the with RB-ROM, but the overall speed up that could be achieved from this for the optimization was compensated by several factors.

The most important one of them are the, compared to the RB-ROM, large estimated errors of the ML-ROM and the high computational costs of the error estimator used. In particular, the increased estimated errors affect the efficiency negatively in different ways. Firstly, due to the overhead induced by performing evaluations and rejecting the solutions afterwards. Secondly, as a consequence, approach the ML-solutions the edge of the trusted region faster, after resetting the model, than the RB-solutions and thus inducing premature costly rebuilds of the RB-ROMs.

Another efficiency-reducing factor is the relatively small parameter domain, where the ML-ROMs can effectively be used, limiting the maximal stepsize available. Furthermore, had we seen that the choice of fixed fallbacks is not optimal, since this can result in over-rejecting solutions. Beside these problems, the experiments have revealed a more practical issue. Choosing a suitable configuration for the ML-models is a crucial but complicated task for obtaining good results. Therefore, before this algorithm can be applied to an unknown problem, initial experiments have to be performed for determining an appropriate setting.

These discussions have highlighted that the algorithm developed in this thesis still holds significant potential for improvement. Further research is required to enhance its capabilities and positioning machine learning in trust region-reduced basis-based optimization as a substantial and promising addition. The most pressing questions is whether the observed behaviour in this thesis is representative of other constrained problems and objective functionals. It is imperative to test more various machine learning algorithms to obtain a comprehensive picture of capabilities and challenges of this algorithm. Furthermore, efforts should be directed towards developing a theoretical understanding of machine learning within the context of trust region-based optimization and to achieve quantitative statements on the expected quality of approximation from the ML-ROM.

The integration of machine learning into model order reduction processes is an active area of research. We hope that the work done in this thesis will be a valuable contribution to this effort and a step towards in making machine learning models a suitable addition to the established methods for solving parameter optimization problems constrained by parabolic partial differential equations.

A. Appendix

A.1. Tables from Chapter 7

Experiment	$\lVert\mu^{(\iota)} - \hat{\mu}\rVert$ av.	$\lVert\mu^{(\iota)} - \hat{\mu}\rVert$ max.	$\lVert\nabla_\mu J^{(\iota)}(\mu^{(\iota)})\rVert$ av.	$\lVert\nabla_\mu J^{(\iota)}(\mu^{(\iota)})\rVert$ max.
RB-O	$3.08 \cdot 10^{-3}$	$5.46 \cdot 10^{-3}$	$2.38 \cdot 10^{-4}$	$4.37 \cdot 10^{-4}$
I	$4.21 \cdot 10^{-3}$	$1.00 \cdot 10^{-2}$	$3.32 \cdot 10^{-4}$	$8.58 \cdot 10^{-4}$
II	$5.23 \cdot 10^{-3}$	$1.16 \cdot 10^{-2}$	$3.96 \cdot 10^{-4}$	$9.94 \cdot 10^{-4}$
III	$3.61 \cdot 10^{-3}$	$7.20 \cdot 10^{-3}$	$2.68 \cdot 10^{-4}$	$4.88 \cdot 10^{-4}$
IV	$3.63 \cdot 10^{-3}$	$9.94 \cdot 10^{-3}$	$2.69 \cdot 10^{-4}$	$6.79 \cdot 10^{-4}$
V	$3.47 \cdot 10^{-3}$	$6.49 \cdot 10^{-3}$	$2.68 \cdot 10^{-4}$	$4.91 \cdot 10^{-4}$
VI	$3.85 \cdot 10^{-3}$	$1.02 \cdot 10^{-2}$	$2.89 \cdot 10^{-4}$	$7.02 \cdot 10^{-4}$

Experiment	$J^{(\iota)}(\mu^{(\iota)})$ av.	runtime [s] av.	speed up av.
RB-O	$4.92 \cdot 10^{-7}$	$3.78 \cdot 10^{2}$	–
I	$9.52 \cdot 10^{-7}$	$5.51 \cdot 10^{2}$	0.69
II	$1.50 \cdot 10^{-6}$	$4.80 \cdot 10^{2}$	0.79
III	$6.47 \cdot 10^{-7}$	$5.15 \cdot 10^{2}$	0.73
IV	$7.63 \cdot 10^{-7}$	$3.49 \cdot 10^{2}$	1.08
V	$6.26 \cdot 10^{-7}$	$3.56 \cdot 10^{2}$	1.06
VI	$8.28 \cdot 10^{-7}$	$3.22 \cdot 10^{2}$	1.17

Table A.1.: Average key figures over all ten initial guesses for each experiment. Shown are the euclidean distances for the final optimizer step $\mu^{(\iota)}$ to $\hat{\mu}$, the last recorded objective $J^{(\iota)}(\mu^{(\iota)})$, its gradient-norm $\lVert\nabla_\mu J^{(\iota)}(\mu^{(\iota)})\rVert$ at $\mu^{(\iota)}$ and the total runtime with speed up relative to RB-O. For the euclidean distance and $\lVert\nabla_\mu J^{(\iota)}(\mu^{(\iota)})\rVert$, additionally the maximal recorded values are shown.

Primal	# FOM evals.	# RB-ROM evals.	# ML-ROM evals.	# total evals.
RB-O	3.00	99.70	0.00	102.70
I	3.70	82.90	120.50	207.10
II	3.60	76.70	123.10	203.40
III	3.80	65.60	116.20	185.60
IV	3.00	28.90	90.80	122.70
V	3.00	99.20	92.20	194.40
VI	3.00	28.90	90.50	122.40
Dual				
RB-O	3.00	99.70	0.00	102.70
I	3.70	90.50	120.50	214.70
II	3.60	87.10	123.20	213.90
III	3.80	76.30	117.00	197.10
IV	3.00	64.00	92.50	159.50
V	3.00	26.30	90.00	119.30
VI	3.00	28.60	90.50	122.10

Table A.2.: Average number of evaluations per model and runtime over all ten initial guesses for each experiment.

Bibliography

[1] M. Belkin, D. Hsu, S. Ma, and S. Mandal, *Reconciling modern machine-learning practice and the classical bias–variance trade-off*, Proceedings of the National Academy of Sciences, 116 (2019), pp. 15849–15854.

[2] M. Belkin, A. Rakhlin, and A. B. Tsybakov, *Does data interpolation contradict statistical optimality?*, in The 22nd International Conference on Artificial Intelligence and Statistics, PMLR, 2019, pp. 1611–1619.

[3] J. Berner, P. Grohs, G. Kutyniok, and P. Petersen, *The modern mathematics of deep learning*, arXiv preprint arXiv:2105.04026, (2021), pp. 86–114.

[4] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart, *Model reduction and neural networks for parametric pdes*, The SMAI journal of computational mathematics, 7 (2021), pp. 121–157.

[5] O. Bousquet, S. Boucheron, and G. Lugosi, *Introduction to statistical learning theory*, (2003), pp. 169–207.

[6] A. M. Bradley, *Pde-constrained optimization and the adjoint method. 2019*, URL https://cs. stanford. edu/˜ ambrad/adjoint_tutorial. pdf, (2021).

[7] A. Buhr, C. Engwer, M. Ohlberger, and S. Rave, *A numerically stable a posteriori error estimator for reduced basis approximations of elliptic equations*, (2014).

[8] F. Cucker and S. Smale, *On the mathematical foundations of learning*, Bulletin of the American mathematical society, 39 (2002), pp. 1–49.

[9] F. Cucker and D. X. Zhou, *Learning theory: an approximation theory viewpoint*, vol. 24, Cambridge University Press, 2007.

[10] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for machine learning*, Cambridge University Press, 2020.

[11] L. Devroye, L. Györfi, and G. Lugosi, *A probabilistic theory of pattern recognition*, vol. 31, Springer Science & Business Media, 2013.

[12] S. Dittmer, T. Kluth, P. Maass, and D. Otero Baguer, *Regularization by architecture: A deep prior approach for inverse problems*, Journal of Mathematical Imaging and Vision, 62 (2020), pp. 456–470.

[13] G. Dziuk, *Theorie und Numerik partieller Differentialgleichungen*, de Gruyter, 2010.

[14] J. L. Eftang, A. T. Patera, and E. M. Rønquist, *An" hp" certified reduced basis method for parametrized elliptic partial differential equations*, SIAM Journal on Scientific Computing, 32 (2010), pp. 3170–3200.

[15] D. Fong and M. Saunders, *Lsmr: An iterative algorithm for sparse least-squares problems*, (2011).

[16] S. Fresca and A. Manzoni, *Pod-dl-rom: Enhancing deep learning-based reduced order models for nonlinear parametrized pdes by proper orthogonal decomposition*, Computer Methods in Applied Mechanics and Engineering, 388 (2022), p. 114181.

[17] M. Geist, P. Petersen, M. Raslan, R. Schneider, and G. Kutyniok, *Numerical solution of the parametric diffusion equation by deep neural networks*, Journal of Scientific Computing, 88 (2021), p. 22.

[18] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[19] B. Haasdonk, *Chapter 2: Reduced Basis Methods for Parametrized PDEs—A Tutorial Introduction for Stationary and Instationary Problems*, pp. 65–136.

[20] B. Haasdonk, H. Kleikamp, M. Ohlberger, F. Schindler, and T. Wenzel, *A new certified hierarchical and adaptive rb-ml-rom surrogate model for parametrized pdes*, (2022).

[21] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proceedings of the National Academy of Sciences, 115 (2018), pp. 8505–8510.

[22] J. Han, L. Zhang, and E. Weinan, *Solving many-electron schrödinger equation using deep neural networks*, Journal of Computational Physics, 399 (2019), p. 108929.

[23] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, *Array programming with NumPy*, Nature, 585 (2020), pp. 357–362.

[24] J. S. Hesthaven and S. Ubbiali, *Non-intrusive reduced order modeling of nonlinear problems using neural networks*, Journal of Computational Physics, 363 (2018), pp. 55–78.

[25] C. Himpe, T. Leibner, and S. Rave, *Hierarchical approximate proper orthogonal decomposition*, SIAM Journal on Scientific Computing, 40 (2018), pp. A3267–A3292.

[26] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich, *Optimization with PDE constraints*, vol. 23, Springer Science & Business Media, 2009.

[27] T. Keil, H. Kleikamp, R. J. Lorentzen, M. B. Oguntola, and M. Ohlberger, *Adaptive machine learning-based surrogate modeling to accelerate pde-constrained optimization in enhanced oil recovery*, Advances in Computational Mathematics, 48 (2022), p. 73.

[28] T. Keil, L. Mechelli, M. Ohlberger, F. Schindler, and S. Volkwein, *A non-conforming dual approach for adaptive trust-region reduced basis approximation of pde-constrained parameter optimization*, ESAIM: Mathematical Modelling and Numerical Analysis, 55 (2021), p. 1239–1269.

[29] C. T. Kelley, *Iterative methods for optimization*, SIAM, 1999.

[30] G. Kutyniok, *The mathematics of artificial intelligence*, arXiv preprint arXiv:2203.08890, (2022).

[31] G. Kutyniok, P. Petersen, M. Raslan, and R. Schneider, *A theoretical analysis of deep neural networks and parametric pdes*, Constructive Approximation, 55 (2022), pp. 73–125.

[32] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, nature, 521 (2015), pp. 436–444.

[33] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation, 1 (1989), pp. 541–551.

[34] K. O. Lye, S. Mishra, D. Ray, and P. Chandrashekar, *Iterative surrogate model optimization (ismo): An active learning algorithm for pde constrained optimization with deep neural networks*, Computer Methods in Applied Mechanics and Engineering, 374 (2021), p. 113575.

[35] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, *A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955*, AI magazine, 27 (2006), pp. 12–12.

[36] R. Milk, S. Rave, and F. Schindler, *pyMOR – generic algorithms and interfaces for model order reduction*, SIAM Journal on Scientific Computing, 38 (2016), pp. S194–S216.

[37] M. Mitchell, *Artificial intelligence: A guide for thinking humans*, Penguin UK, 2019.

[38] J. Nocedal and S. J. Wright, *Conjugate gradient methods*, Numerical optimization, (2006), pp. 101–134.

[39] E. Qian, M. Grepl, K. Veroy, and K. Willcox, *A certified trust region reduced basis approach to pde-constrained optimization*, SIAM Journal on Scientific Computing, 39 (2017), pp. S434–S460.

[40] A. Quarteroni, A. Manzoni, and F. Negri, *Reduced basis methods for partial differential equations: an introduction*, vol. 92, Springer, 2015.

[41] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational physics, 378 (2019), pp. 686–707.

[42] D. Reis, J. Kupec, J. Hong, and A. Daoudi, *Real-time flying object detection with yolov8*, arXiv preprint arXiv:2305.09972, (2023).

[43] F. Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological review, 65 (1958), p. 386.

[44] G. Santin and B. Haasdonk, *Kernel methods for surrogate modeling*, Model Order Reduction, 1 (2019), pp. 311–354.

[45] R. Schaback and H. Wendland, *Kernel techniques: from machine learning to meshless methods*, Acta numerica, 15 (2006), pp. 543–639.

[46] D. N. Tanyu, J. Ning, T. Freudenberg, N. Heilenkötter, A. Rademacher, U. Iben, and P. Maass, *Deep learning methods for partial differential equations and related parameter identification problems*, Inverse Problems, 39 (2023), p. 103001.

[47] V. Tavakkoli, J. C. Chedjou, and K. Kyamakya, *A novel recurrent neural network-based ultra-fast, robust, and scalable solver for inverting a "time-varying matrix"*, Sensors, 19 (2019), p. 4002.

[48] V. Vapnik, *The nature of statistical learning theory*, Springer science & business media, 1999.

[49] S. Volkwein, *Proper orthogonal decomposition: Theory and reduced-order modelling*, Lecture Notes, University of Konstanz, 4 (2013), pp. 1–29.

[50] U. von Luxburg and B. Schoelkopf, *Statistical learning theory: Models, concepts, and results*, 2008.

[51] Y. Yue and K. Meerbergen, *Accelerating optimization of parametric linear systems by model order reduction*, SIAM Journal on Optimization, 23 (2013), pp. 1344–1370.

List of Figures

4.1. Flowchart of the general adaptive scheme. Derived from [20]. 30

7.1. On the floor with domain $\Omega := (0,2) \times (0,1)$ heaters below the windows $i.$, doors $\underline{i}$ and walls $i|$ are places. Each has its own parameter μ_* describing the power of the heaters or respectively the heat conductivity of the components. Everywhere else the heat conductivity is fixed as $\mu_{\text{air}} = 0.5$. 53

7.2. Times per call for estimating state (darker colors) and errors (lighter colors) along the trajectory starting at initial guess $\mu^{(0)} = (2.0584, 96.9910)$. In each iteration (x-axis) one of the models M, $M^{(i)}$ or $M^{(i,l)}$ was evaluated. The color indicates which of the three stages (M_h^*, M_{RB}^* and M_{ML}^*) was used, FOM (blue), RB-ROM (orange), ML-ROM (green). Plotted are experiments II (left) and VI (right). For each the primal evaluations (top) and dual evaluations (bottom) are shown. . . . 57

7.3. Relative estimated errors for the RB-ROM M_{RB}^* (light) and the ML-ROM M_{ML}^*(dark) for the optimization starting at $\mu^{(0)} = (2.0584, 96.9910)$. These models are evaluated for all parameter used during the optimization, after each call of $\tilde{M}$`.eval_and_est_objective` in Algorithm 5. For both models the primal (blue) and dual (orange) errors are shown. Plotted are experiments II (left) and VI (right). The error tolerances $\epsilon^{\text{pr}} = 10^{-4}$ and $\epsilon^{\text{du}} = 10^{-2}$ are shown as dashed lines. 58

7.4. Times per call for estimating state (darker colors) and errors (lighter colors) along the trajectory starting at initial guess $\mu^{(0)} = (40.0000, 60.0000)$. In each iteration (x-axis) one of the models M, $M^{(i)}$ or $M^{(i,l)}$ was evaluated. The color indicates which of the three stages (M_{ML}^*, M_{RB}^* and M_h^*) was used, FOM (blue), RB-ROM (orange), ML-ROM (green). Plotted is experiment II with reduced stepsize. The primal evaluations are shown on the left and dual evaluations on the right. 61

7.5. Relative estimated errors for the RB-ROM (light) and the ML-ROM (dark) for the optimization starting at $\mu^{(0)} = (40.0000, 60.0000)$. These models are evaluated for all parameter used during the optimization, after each call of $\tilde{M}$`.eval_and_est_objective` in Algorithm 5. For both models the primal (blue) and dual (orange) errors are shown. Plotted is experiments II with reduced stepsize. The error tolerances $\epsilon^{\text{pr}} = 10^{-4}$ and $\epsilon^{\text{du}} = 10^{-2}$ are shown as dashed lines. 62

List of Tables

7.1. Parameters used for the optimization of the heater capacity. 55

7.2. Overview over the different kernels and transformations used in the heater capacity experiments. 56

7.3. Average times over all ten test runs for model evaluation, error estimation and model building, separated by models and branch (primal/dual). The total time refers to the average time for evaluation and error estimation combined. Building times for the ML-ROMs are identical to the training times. For the RB-ROMs this is the time used to extend V_{RB}^{*} and projecting the operators to the new basis, reconstructing both error estimator and proloning the training data. The reduced models $M_{\mathrm{RB}}^{\mathrm{pr}}$ and $M_{\mathrm{RB}}^{\mathrm{du}}$ are constructed with the same vector space, so only one extension is actually performed and the build of both RB-ROMs is run as one block. 60

A.1. Average key figures over all ten initial guesses for each experiment. Shown are the euclidean distances for the final optimizer step $\mu^{(\iota)}$ to $\hat{\mu}$, the last recorded objective $J^{(\iota)}(\mu^{(\iota)})$, its gradient-norm $\|\nabla_{\mu} J^{(\iota)}(\mu^{(\iota)})\|$ at $\mu^{(\iota)}$ and the total runtime with speed up relative to RB-O. For the euclidean distance and $\|\nabla_{\mu} J^{(\iota)}(\mu^{(\iota)})\|$, additionally the maximal recorded values are shown. 66

A.2. Average number of evaluations per model and runtime over all ten initial guesses for each experiment. 67

Declaration of Academic Integrity

I hereby confirm that this thesis on A trust region-reduced basis-machine learning approach for parameter optimization with parabolic PDE constraints is solely my own work and that I have used no sources or aids other than the ones stated. All passages in my thesis for which other sources, including electronic media, have been used, be it direct quotes or content references, have been acknowledged as such and the sources cited.

(date and signature of student)

I agree to have my thesis checked in order to rule out potential similarities with other works and to have my thesis stored in a database for this purpose.

(date and signature of student)

Plagiatserklärung der / des Studierenden

Hiermit versichere ich, dass die vorliegende Arbeit über A trust region-reduced basis-machine learning approach for parameter optimization with parabolic PDE constraints selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

____
(Datum, Unterschrift)

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

____
(Datum, Unterschrift)