Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Optimised Nonparametric Probabilistic Approach for Uncertainty Quantification for Operator Inference Based Reduced Order Models

Felix Michael Sievers

Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Optimised Nonparametric Probabilistic Approach for Uncertainty Quantification for Operator Inference Based Reduced Order Models

Author:	Felix Michael Sievers
Examiner:	Prof. Dr. Hans-Joachim Bungartz
Advisors:	Dr. Felix Dietrich
	Dr. Dirk Hartmann (Siemens)
	Dr. Lukas Failer (Siemens)
Submission Date:	July 15th, 2021

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

F. Sievers

July 15th, 2021 Felix Michael Sievers

Acknowledgments

First of all, I would like to thank Prof. Dr. Hans-Joachim Bungartz and Dr. Christoph Heinrich for allowing me to write my master's thesis in cooperation with Siemens. Furthermore, I owe a huge thank you to my supervisor from TUM Dr. Felix Dietrich and my supervisors from Siemens Dr. Dirk Hartmann and Dr. Lukas Failer. Thank you for listening every Monday over the last eight months to my progress, and for continuously giving me helpful ideas on possible next steps, and for the constructive feedback on the work that I have done. Not only have you been supporting me every Monday but also in some extra sessions in case I had an urgent question or I got stuck somewhere. Thanks a lot!

Especially in the beginning of my thesis, it was tough to get a hold of the topic and to understand every detail of the methods I had to use. In this phase of my thesis, I was in email contact with Christian Soize who helped me to gain confidence in what I was planing and helped me to better understand the nonparametric probabilistic approach for uncertainty quantification proposed by Soize and Farhat. Lots of thanks for that.

I also want to thank Dr. Juan Lorenzi and Qinyu Zhuang, who shared with me and explained the data sets I analysed in this thesis. An other thanks goes to Diana Manvelyan and Christoph Ludwig who kindly took some time during the start of my thesis to introduce me to the provided code base.

One very pleasant part was to write my thesis together with Shubham Khatri. We always had fruitful discussions after our weekly meeting and sometimes even met during the week to discuss our topics and give a little moral support in form of beer or whiskey. Thanks a lot for going with me through all of that. I also do not want to keep Nicole Höfer unmentioned. I owe you a lot for going through the trouble of proof reading my thesis.

Furthermore, I want to acknowledge my parents who always belief in me and support me in whatever I do. I could not have done anything I achieved without you. Thank you for everything.

Last but not least, I want to thank my girlfriend Misaki Yahata for always being with me and cheering me up in times of bad mood or frustration.

"The beauty of life lies in its uncertainty. Embrace it."

-Maniteja Kamireddy

Abstract

The speedup of reduced order models (ROMs) compare to full order model (FOM) always comes with a trade off in accuracy. To still make well-founded decisions based on the ROM, it is necessary to quantify the uncertainties of the ROM, where it is often impossible to cover all error sources. In this thesis, we use a nonparametric probabilistic approach for uncertainty quantification (UQ) to describe the uncertainties of μ-parametric operator inference (OI)-based ROMs, which is a non-intrusive approach that allows modellers to build ROMs purely based on data gathered, for example, from a black box solver, where the operators of the discretised problem is unknown. This UQ approach works via indirectly perturbing the data from the FOM that the ROM is built on. The FOM solution is therefore not deterministic anymore, but defined in a stochastic solution space. The ROM, built based on the stochastic solution space, is then called a stochastic reduced order model (SROM). The solution, and therefore the ROM, can be tuned with a set of parameters $\boldsymbol{\alpha}$, which have to be optimised via solving an inverse stochastic problem according to a cost functional to make sure that the SROM is covering all the errors of the deterministic ROM in comparison to the FOM. In this work, we improved the optimisation procedure of the SROM, which is done via a quasi Newton version of the interior point optimisation method with a finite difference (FD) gradient approximation, by introducing an analytical gradient via solving the corresponding adjoint problem. Additionally, we introduced the negative log marginal likelihood (NLML) cost function used to optimise the SROM parameters. This resulted in the desired accuracy of the confidence intervals of 99.7%. The SROM methodology was demonstrated in four computational experiments. At first, we used two static problems to investigate the basic approach and to test the changes we made to the procedure. Finally, in two transient problems with OI-based ROMs we have shown that the UQ approach is constructing confidence intervals covering most errors between FOM and ROM.

Contents

Notation

A real, deterministic variable is denoted by a lower case letter such as y.
A real, deterministic vector is denoted by a boldface, lower case letter such as in $\mathbf{y} = (y_1, ..., y_N)$.
A real, random variable is denoted by an upper case letter such as Y.
A real, random vector is denoted by a boldface, upper case letter such as in $\mathbf{Y} = (Y_1, ..., Y_N)$.
A real, deterministic matrix is denoted by an upper (or lower) case letter between brackets such as $[A]$ (or $[a]$).
A real, random matrix is denoted by a boldface, upper case letter between brackets such as $[\mathbf{A}]$.

N denotes the dimension of the full order model (FOM), and n the dimension of the reduced order model (ROM) if not stated otherwise.
$\|\mathbf{y}\|$ designates the Euclidean norm of vector $\mathbf{y}$.
$< \mathbf{y}, \mathbf{z} >$ designates the Euclidean inner product of $\mathbf{y}$ and $\mathbf{z}$.
E designates the mathematical expectation.
$\mathbb{M}_{N,n}$ denotes the set of $N \times n$ real matrices.
$\mathbb{M}_n$ denotes the set of square $n \times n$ real matrices.
$\mathbb{M}_N^+$ denotes the set of Symmetric Positive-Definite (SPD) $N \times N$ real matrices.
$\mathbb{M}_N^{\mathrm{u}}$ denotes the set of upper triangular $N \times N$ real matrices with strictly positive diagonal entries.
A_{jk} designates the entry $[A]_{jk}$ of matrix $[A]$.
$\mathrm{tr}\{[A]\}$ designates the trace of matrix $[A]$.
$[A]^\top$ designates the transpose of matrix $[A]$.
$\|A\|_F$ designates the Frobenius norm of matrix $[A]$, with $\|A\|_F^2 = \mathrm{tr}\left\{[A]^T[A]\right\}$.
$\|A\|_M$ is defined by $\|A\|_M^2 = \mathrm{tr}\left\{[A]^T[M][A]\right\}$, where $[M]$ is a positive definite matrix.
$[I_N]$ denotes the identity matrix in $\mathbb{M}_N$.
$[\mathbb{1}_{N,n}]$ denotes a matrix with all entries equal to 1 in $\mathbb{M}_{N,n}$.
$\mathbb{1}_n$ denotes a vector with all entries equal to 1 in $\mathbb{R}^n$.
$[0_{N,n}]$ denotes the zero matrix in $\mathbb{M}_{N,n}$.
δ_{jk} denotes Kronecker's symbol and therefore verifies $\delta_{jk} = 0$ if $j \neq k$ and $\delta_{jk} = 1$ if $j = k$.

1. Introduction

Mathematical models are widely used in plenty of different application fields like in biology for protein folding [11] or the spreading of infectious diseases like COVID-19 [3], in chemistry for chemical reaction dynamics [21] or in electrical engineering to check the stability of electric circuits [40], just to name a few. In this thesis we want to look at μ-parametric high fidelity Partial Differential Equation (PDE)-based computational models, with $\boldsymbol{\mu} = (\mu_1, ..., \mu_{N_p})$ belonging to some parameter space $\mathcal{C}_\mu$. PDEs play an important role in science and engineering, as all kinds of physical phenomena like heat, diffusion, electrostatics, electrodynamics, fluid dynamics and elasticity [8, 9, 5, 2, 1] can be modelled with them. The according computational models are in general high-dimensional because of their spatial discretisation. Hence, they either take a long time or a lot of compute power to simulate, which makes them rarely applicable to real time applications or use cases with a high number of model evaluation. Hence, approximate models are needed with a shorter evaluation time. However, as they are only approximations of the real solution, it is crucial to have uncertainty quantification (UQ) to certify the decisions that are taken based on the approximate solutions.

There has been a lot of research regarding UQ in model order reduction (MOR) [26, 23, 37, 33]. To the best of our knowledge, still an open research topic is to quantify the uncertainty of an operator inference (OI)-based reduced order model (ROM) over a parameter space. OI is a non-intrusive method of reduced order modelling, thus usable without the exact operators of the spatially discretised PDE. This method only requires knowledge about the type of operators of the semi-discretised PDE to be able to infer them and to build a ROM. However, it is restricted to semi-discretised PDEs that can be approximated with a PDE involving the state variable only as polynomial of degree at most two. The discretisation is often not available directly, for example in industrial problems where engineers work with simulation software like Siemens NX as black box solver, where they can not directly access the operators. For the OI, the full order model (FOM) is sampled for different parameter settings to get snapshots of the state vector of dimension N at certain points in time. Those snapshots are assembled to a snapshot matrix that can be used to build a reduced order basis (ROB) $[V] \in \mathbb{M}_{N,n}$ out of the condensed information of this matrix. The ROB maps the high dimensional state vector to a lower dimensional subspace of dimension n with $n \ll N$ that covers the dominant part of the behaviour of the system. The sampled snapshot matrix is then reduced to the low dimensional space and used to infer the operators of interest via linear least squares (LS) optimisation. There are multiple error sources in this process: a reduction error from reducing the state vectors to a lower dimensional space, a sampling error as the ROB is based on only a few samples of the

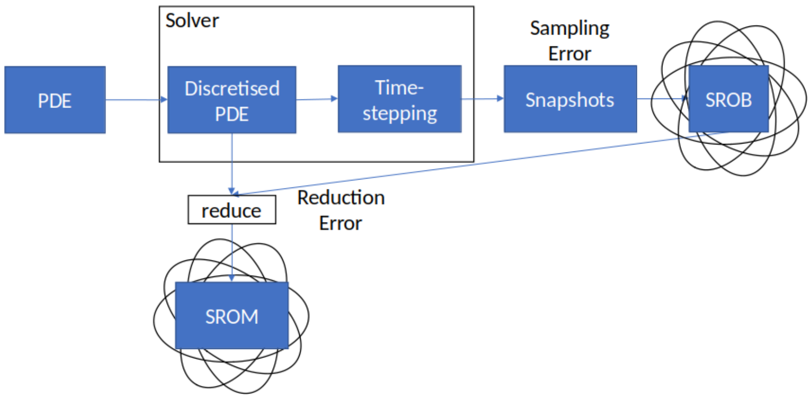


Figure 1.1.: Flow chart of generating a SROM for a projection based ROM. Using the operators of the spatial-discretised PDE allows the SROM to account for possible sampling errors and for reduction errors. Sampling errors occur, because the samples are not a good representation of the rest of the parameter space. The reduction error occurs, because we do not use all of the modes from the ROB, thus we cover not all of the energy from the system. The model can account for possible modelling errors by optimising the parameters of the SROB not over snapshots from the solver but over experimentally gathered snapshot of the real system, where a snapshot is the values of the state space of a system at a certain time.

parameter space and a truncation error in case the semi-discretised PDE can not be fully described in a form with a quadratic leading term.

It is not trivial to account for all of those errors in UQ. In recent research Soize and Farhat proposed a non-parametric probabilistic approach for projection-based MOR that is capable of accounting for reduction errors, sampling errors and modelling errors of the FOM in comparison to experimentally collected data [36] and the idea of this method can also be used for OI-based ROM. In the following we will discuss why. In comparison to OI, projection-based MOR is an intrusive approach. It uses a ROB to directly project the known operators of the semi-discretised Proper Orthogonal Decomposition (POD) to generate the ROM. The main idea of the proposed approach from Soize and Farhat is to substitute the ROB $[V]$ with a stochastic counterpart $[\mathbf{W}]$ that varies around $[V]$, fulfils certain conditions to be a valid ROB, and depends on a parameter set $\boldsymbol{\alpha}$. Then, instead of building a ROM, we build a stochastic reduced order model (SROM) which covers all the errors of the ROM in its variance. The generation of a SROM is shown in Figure 1.1 . To ensure that the variance covers all the errors, $\boldsymbol{\alpha}$ has to be optimised by solving a statistical inverse problem according to a cost functional. This is done by an algorithm based on multiple interior point optimisation algorithms using a quasi-Newton approach with a finite difference (FD) gradient approximation. One major drawback of this algorithm is

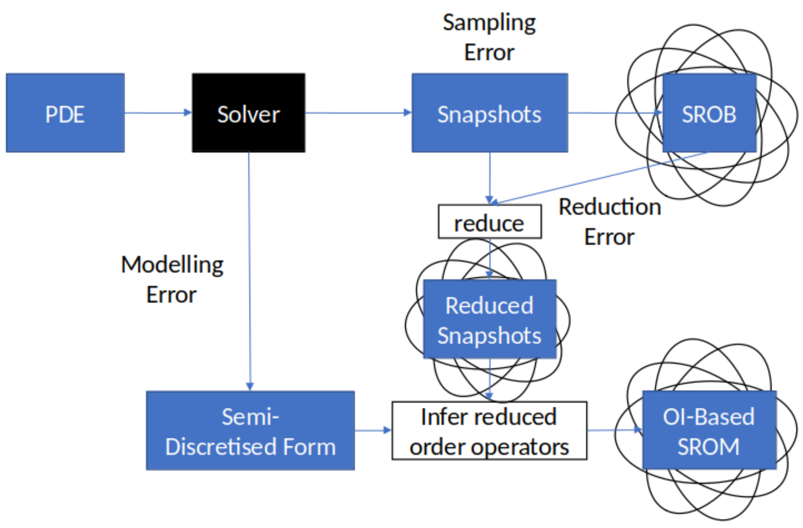


Figure 1.2.: Flow chart of building a SROM for an OI-based ROM. As we do not have any knowledge about the solver as in the case of a projection-based ROM, we can not account for any sampling errors. However, the SROM can learn modelling and reduction errors, introduced by the assumption of a quadratic leading term of the OI and the dimension reduction via the SROB. This is possible, as we learn the parameters based on the unreduced snapshots we collected, which are free of any modelling errors from the OI.

that the cost of the gradient scales with the number of optimisation parameters. The overall approach is shown in the context of a nonlinear Finite Element (FE) structural dynamics model, but is in principle applicable to any μ-parametric, nonlinear, computational model. The idea of this thesis was to use this method for OI-based ROMs, as it is also based on a ROB $[V]$. Furthermore, the truncation error of the OI can be seen as modelling error in comparison to the real data, which is therefore accounted for. The flow of building a SROM for an OI-based ROM is shown in Figure 1.2. Another objective was the improvement of the optimisation algorithm in a way that the gradient is not dependent on the number of optimisation parameters anymore. As ultimate goal we wanted confidence intervals which are valid in a range of three standard deviations and above, thus at least 99.7% of the full order model (FOM) solution lies withing the confidence interval to provide a solid foundation for aiding simulation based decisions.

In this thesis, we introduce a framework for the non-parametric probabilistic approach for projection-based and for OI-based ROMs, with an alternative optimisation algorithm for α that uses an analytical gradient which is not dependent on the number of parameters anymore and gives therefore a significant speedup to the overall optimisation. The gradient is computed via solving the corresponding adjoint problem, which is described in Section 3.2.3. Additionally, we introduce the negative log marginal likelihood (NLML) as an alternative cost function which showed an improved coverage of the FOM in the confidence interval.

The thesis is structured as follows. In Chapter 2, the state of the art is presented. We briefly introduce the POD for static and dynamic problems as choice for the SROB in this thesis, give a more thorough introduction to the nonparametric probabilistic approach for uncertainty quantification from Soize and Farhat, and introduce important topics regarding the simulation algorithm, namely the interior point optimisation algorithm, the epsilon approximation of the FD scheme, and the adjoint method to compute the analytical gradient of the cost function. Chapter 3 is divided into four sections. First of all, we introduce the developed tool to build SROMs, state its modular structure and provide information on how to further extend the tool. In Section 3.2 we uses a generic static problem to introduce the nonparametric probabilistic method. Furthermore, we replaced the FD gradient with an analytical gradient which caused a speedup of a factor up to 42 and introduce an alternative optimisation algorithm which led to a maximum speedup factor of 80 in comparison to the original algorithm with a FD gradient. Additionally, we investigated the confidence interval from the SROM in terms of validity and show that the proposed cost function from Soize and Farhat did not lead to a confidence interval valid for a range of three standard deviations. Hence, we propose optimising the SROM over the NLML function, which led to the desired outcome. In Section 3.3 we show how the developed tool can be used with models from one of the commonly used MOR python packages called pyMOR and we show that the impact of not using the grid information for the SROB, which is normally used to make sure that points close in space correlate with each other, led to underconfident confidence intervals, but still gave reasonable results. This is of interest, as the grid information was not given for the final two problems. The last section shows two SROMs for OI-based ROMs for a conduction dominated problem, the heat sink, and for a problem with an additional radiation term, referred to as gap radiation. The gap radiation problem introduces lifting, which enables to apply OI to a broader range of PDEs, as described in 2.1.2. The SROMs managed to give a valid confidence interval in the range of four standard deviations for the training data. However, only the SROM of the gap radiation problem gave valid intervals over the whole parameter space. The heat sink problem got slightly overconfident intervals, which came from a non-optimal sampling of the training points. As last experiment, we checked for the gap radiation problem the validity of the confidence intervals under extrapolation of a short period in time, which works as long as the state vectors stays within the range represented by the training set. In the end, we summaries all results and give a short outlook into possible improvements and future research directions.

2. State of the Art

In this chapter we introduce all important methods and theories that are needed to follow the argumentation in this thesis. In the first section, we describe the POD for static and dynamic systems, which was used for every ROM of the numerical experiments in Chapter 3. Furthermore, we describe OI which is the MOR method we want to compute the uncertainties for. The uncertainty quantification is done via a nonparametric probabilistic approach introduced in Section 2.2. In the last section of this chapter we explain the theory behind all optimisation related topics in this thesis. First of all, we introduce the interior point optimisation. Subsequently, we explain the FD scheme for the gradient approximation, and the adjoint method for computing the analytical gradient.

2.1. Model Order Reduction

MOR is a sophisticated method that was initially developed in 1967 where Lumley used it to describe the mechanisms and intensity of turbulence in fluid flow problems [4]. A lot has happened since. One major breakthrough was reduced basis methods, which are briefly described using the POD in this chapter [31, 39]. In the second subsection, we cover OI which is a non-intrusive reduced order method developed rather recently in 2016 [28].

2.1.1. Proper Orthogonal Decomposition for Model Order Reduction

The Proper Orthogonal Decomposition is a method for generating an orthogonal basis representing a given set of snapshots in a way that minimises the least-squares error between the projected data and the original data according to a certain inner product. This enables us to find an L^2-optimal lower-dimensional approximation of a solution space, which can speed up numerical computations tremendously. The POD has originally been introduced by Sirovich in 1987 [35]. However, the POD is just another name for Principal Component Analysis (PCA), which has already been developed in 1901 from Karl Pearson [27]. In other fields of application it is also known as discrete Karhunen–Loève transform (signal processing), Hotelling transform (multivariate quality control), singular value decomposition or eigenvalue decomposition (linear algebra), just to name a few. Its application in model order reduction started after the work from Kunisch and Volkweind about reduced-order Galerkin methods for PDEs [18]. It is heavily used in numerical computation of fluid dynamics and turbulence analysis for generating a ROM for the Navier-Stokes equations [6, 19, 20]. But it is also used in other fields like structural analysis to reduce the complexity of compute intensive simulations [17].

The rest of this section is a brief summary of PODs for reduced order models according to S. Volkwein [39], given a snapshot matrix $[Y] = [\mathbf{y}_1, ..., \mathbf{y}_k]$ of $\mathbb{M}_{N,k}$, with k snapshots of dimension of N, where a snapshot $\mathbf{y}_i$ is the state vector of a system at a particular point in time. If the system depends on a set of parameters $\boldsymbol{\mu}$, the snapshots can also differ in terms of $\boldsymbol{\mu}$. For the gap radiation problem in Section 3.4.2 for example, the POD is created out of snapshots of multiple simulations for different sets of $\boldsymbol{\mu}$.

Static System

In case of a static system, the different snapshots arise only from a different parameter selection $\boldsymbol{\mu}$. The POD constitutes the collection of vectors $u_1^*, \ldots, u_n^*$ that solve the following minimisation problem

$$\begin{aligned} &\min_{\boldsymbol{u}_1,...,\boldsymbol{u}_n \in \mathbb{R}^N} \left\| \mathbf{y} - \sum_{\ell=1}^{n} \langle \mathbf{y}, \boldsymbol{u}_\ell \rangle_{[W]} \boldsymbol{u}_\ell \right\|_{[W]}^2 , \\ &\text{s.t. } \langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{[W]} = \delta_{ij}, 1 \leq i,j \leq n, \end{aligned} \tag{2.1}$$

where $\{\boldsymbol{u}_1, ..., \boldsymbol{u}_n\}$ is an orthonormal basis with respect to the inner product $< \cdot, \cdot >_{[W]}$, with $[W] \in \mathbb{M}_N$. The POD that solves Equation [2.1] can be calculated via the singular value decomposition (SVD) of $[\hat{Y}] = [W]^{\frac{1}{2}}[Y]$,

$$[\hat{Y}] = [\hat{U}][\Sigma][\hat{V}]^\top, \tag{2.2}$$

where $[\hat{U}] \in \mathbb{M}_N$ is a matrix with the left singular-vectors in its columns, $[\Sigma] \in \mathbb{M}_{N,k}$ is a rectangular diagonal matrix with the singular values λ_ℓ on its diagonal with $\lambda_\ell > \lambda_{\ell+1}$, and $[\hat{V}] \in \mathbb{M}_k$ is a matrix with the right singular-vectors in its columns.

The columns of the optimal POD are then defined by

$$\boldsymbol{u}_\ell^* = [W]^{-\frac{1}{2}} \hat{\boldsymbol{u}}_\ell, \quad \text{for } i = 1, ..., n, \tag{2.3}$$

which gives us $[U] \in \mathbb{M}_{N,n}$ as POD with a ratio of modelled to total energy of

$$\mathcal{E}(n) = \frac{\sum_{\ell=1}^{n} \lambda_\ell}{\sum_{\ell=1}^{d} \lambda_\ell} \tag{2.4}$$

with $d = \min(N, k)$. Usually, most of the energy is covered within the first few modes, thus they result in a rather accurate approximation of the system, as can be seen, for example, later in Figure 3.24 .

Dynamic System

For dynamic systems the procedure is quite similar. However, the minimisation objective is different as the snapshots result not only from changes in parameters, but also from measurements over time

$$\begin{aligned} &\min_{\boldsymbol{u}_1,\dots,\boldsymbol{u}_n \in \mathbb{R}^N} \int_{t_0}^{t_f} \left\| \mathbf{y}(t) - \sum_{\ell=1}^{n} \langle \mathbf{y}(t), \boldsymbol{u}_\ell \rangle_{[W]} \boldsymbol{u}_\ell \right\|_{[W]}^2 \mathrm{d}t, \\ &\text{s.t. } \langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{[W]} = \delta_{ij}, 1 \le i,j \le n. \end{aligned} \tag{2.5}$$

Approximating the integral via the trapezoidal rule results in

$$\begin{aligned} &\min_{\boldsymbol{u}_1,\dots,\boldsymbol{u}_n \in \mathbb{R}^N} \sum_{j=1}^{k} a_j \left\| \mathbf{y}_j - \sum_{\ell=1}^{n} \langle \mathbf{y}_j, \boldsymbol{u}_\ell \rangle_{[W]} \boldsymbol{u}_\ell \right\|_{[W]}^2 \mathrm{d}t, \\ &\text{s.t. } \langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_{[W]} = \delta_{ij}, 1 \le i,j \le n, \end{aligned} \tag{2.6}$$

where the weights a_j have to be chosen as

$$a_1 = \frac{t_1 - t_0}{2}, \quad a_j = \frac{t_{j+1} - t_{j-1}}{2} \quad \text{for } 2 \le j \le k-1, \quad a_k = \frac{t_k - t_{k-1}}{2}.$$

Since the objective function has changed, we also have to change the matrix for the SVD to $[\bar{Y}] = [W]^{\frac{1}{2}}[Y][D]^{\frac{1}{2}}$, with $[D] = \text{diag}(a_1, .., a_k) \in \mathbb{M}_k$

$$[\bar{Y}] = [\bar{U}][\Sigma][\bar{V}]^\top, \tag{2.7}$$

where the components are defined as above. From here on, the procedure is equivalent to the static system. The columns of the optimal POD are again defined by

$$\boldsymbol{u}_\ell^* = [W]^{-\frac{1}{2}} \bar{\boldsymbol{u}}_\ell, \quad \text{for } i = 1, ..., n, \tag{2.8}$$

which gives us $[U] \in \mathbb{M}_{N,n}$ as POD with the same modelled energy approximation as for the static case.

For models discretised via a Finite Elements method, $[W]$ is the mass matrix, but as the time dependent reduced order models in this thesis are non-intrusive, the mass matrix is unknown and $[W]$ is considered to be the identity matrix.

2.1.2. Operator Inference ROM

Operator Inference is a non-intrusive method to build a ROM for nonlinear partial differential equations. The approach has been introduced by B. Peherstorfer and K. Willcox in 2016 [28]. In recent years, it found more and more attention as the method achieved very good results in complex systems, like the single-injector combustion process [24].

Basic Operator Inference

The method is based on the assumption that the ROM has the same structure as the spatial discretised PDE of the full order model (FOM). The method is only applicable to problems with a leading quadratic term [24]. If not given, it is in some cases possible to get this property via a lifting transformation that has to be done before the reduction [30]. An example for the methodology of lifting is shown in Section 3.4.2.

We assume that the FOM as well as the ROM can be approximated in the following way

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{y}(t) = \mathbf{c} + [A]\mathbf{y}(t) + [H](\mathbf{y}(t) \otimes \mathbf{y}(t)) + [B]\mathbf{u}(t), \quad \mathbf{y}(t_0) = \mathbf{y}_0, \quad t \in [t_0, t_f], \tag{2.9}$$

where $\mathbf{y}(t)$ in $\mathbb{R}^N$ is the state vector at time t, $\boldsymbol{u}(t)$ in $\mathbb{R}^m$ is the input vector at time t, $\mathbf{y}_0$ is the initial condition and t_0 and t_f are the initial time and the termination time. The operator "$\otimes$" multiplies each element of one vector with each element of the other vector. Therefore, the "$\otimes$" operator returns, with an input of size N, a vector of size $\frac{N}{2}(N-1)$. Furthermore, we have the operators $\mathbf{c} \in \mathbb{R}^N$, $[A] \in \mathbb{M}_N$, $[H] \in \mathbb{M}_{N,\frac{N}{2}(N-1)}$ and $[B] \in \mathbb{M}_{N,m}$, which do not need to be explicitly known for this method. For the ROM we have to replace the dimension of the FOM N with the dimension of the ROM n in the dimension of the operators. The lower dimensional operators are then going to be inferred.

We assume the data to be given discretised in N nodes in space and k steps in time $[Y] = [\mathbf{y}_0, ..., \mathbf{y}_{k-1}] \in \mathbb{M}_{N,k}$. In the following, we compute the POD $[V] \in \mathbb{M}_{N,n}$ of the given snapshot matrix $[Y]$ and use it to get the reduced snapshot matrix $[Q] = [V]^\top [Y] = [\mathbf{q}_0, ..., \mathbf{q}_{k-1}] \in \mathbb{M}_{n,k}$, where the $\mathbf{q}_i \in \mathbb{R}^n$ are the corresponding reduced vectors to $\mathbf{y}_i$. The operators of the ROM are inferred by solving the minimisation problem

$$\min_{\mathbf{c},[A],[H],[B]} \sum_{j=0}^{k-1} \|\mathbf{c} + [A]\mathbf{q}_j + [H](\mathbf{q}_j \otimes \mathbf{q}_j) + [B]\mathbf{u}_j - \dot{\mathbf{q}}_j\|_2^2 . \tag{2.10}$$

This equation can be rewritten in matrix form, which is going to be used in this thesis

$$\min_{[OP]} \left\| [D][OP]^\top - [R]^\top \right\|_F^2 , \tag{2.11}$$

where $\|\cdot\|_F$ is the Frobenius norm and

$$\begin{aligned}
[OP] &= \begin{bmatrix} \mathbf{c} & [A] & [H] & [B] \end{bmatrix} \in \mathbb{M}_{n,d(n,m)}, \quad &&\text{(unknown operators)},\\
[D] &= \begin{bmatrix} \mathbf{1}_k & [Q]^\top & ([Q]^\top \otimes [Q]^\top) & [U]^\top \end{bmatrix} \in \mathbb{M}_{k,d(n,m)}, \quad &&\text{(known data)},\\
[R] &= \begin{bmatrix} \dot{\mathbf{q}}_0 & \dot{\mathbf{q}}_1 & \cdots & \dot{\mathbf{q}}_{k-1} \end{bmatrix} \in \mathbb{M}_{n,k}, \quad &&\text{(time derivatives)},\\
[U] &= \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \cdots & \mathbf{u}_{k-1} \end{bmatrix} \in \mathbb{M}_{m,k}, \quad &&\text{(inputs)},
\end{aligned}$$

where $d(n,m) = 1 + n + \frac{1}{2}n(n+1) + m$.

The inferred operators allow us to compute an approximate solution to Equation [2.9] for new data points. A newly chosen initial state of interest $y_0 \in \mathbb{R}^N$ has to be projected into the lower dimensional space via the same POD used for the operator inference. The vector $q_0 = [V]^\top \mathbf{y}_0 \in \mathbb{R}^n$ defines with the ROM an initial value problem, which can be solved with any time stepping scheme. The solution is a time series of reduced order state vectors, which have to be projected back via $[V]$ to the high dimensional space to get an approximate solution to the FOM.

Operator Inference with Regularisation

The ROM computed via Equation [2.11] is going to have a rather poor predictive performance, as the model tries to perfectly fit the data. The data, however, includes errors due to the numerically estimated time derivatives $[R]$, modelling errors, in case the system is not truly quadratic, and is missing information from the reduction. In [24] McQuarrie introduced L^2-regularisation into the model to prevent overfitting, which results in a better performance. The new minimisation problem is

$$\min_{[OP]} \left\| [D][OP]^\top - [R]^\top \right\|_F^2 + \left\| [\Gamma][OP]^\top \right\|_F^2 = \min_{[OP]} \left\| \begin{bmatrix} [D] \\ [\Gamma] \end{bmatrix} [OP]^\top - \begin{bmatrix} [R]^\top \\ [0] \end{bmatrix} \right\|_F^2 \tag{2.12}$$

$$= \min_{[OP]} \left\| [\hat{D}][OP]^\top - [\hat{R}]^\top \right\|_F^2 \tag{2.13}$$

where $[\Gamma]$ is defined as $\lambda[I_{d(n,m)}]$. The regularisation parameter λ results from minimising the residual

$$\min_{\lambda} \|[Y] - [\hat{Y}]\|_2, \tag{2.14}$$

between the original state data $[Y]$ and the solution from the initial value problem $[\hat{Y}]$, defined by the ROM and the initial states from $[Y]$. For the numerical examples in Section 3.4 we used a second order approximation of the time derivative to infer the operators. Therefore, we solved the initial value problems with OI models via Heun, a second order time stepping scheme.

2.2. Nonparametric Probabilistic Approach For Quantifying Uncertainties

This section is going to describe a nonparametric probabilistic approach for quantifying uncertainties for reduced order models. In 2017, Soize and Farhat introduced a novel approach of predicting uncertainties [36]. Their method is not only capable of predicting uncertainties regarding the parameters but also regarding modelling errors of any μ-parametric, nonlinear, computational model. Soize and Farhat use a nonlinear projection-based ROM to introduce their method, but the idea is applicable to any ROM based on a ROB. This enables us to extend this approach to OI models in Section 3.4.

The basic idea of this method is to add some statistical fluctuation to a ROB, which is then called a stochastic reduced order basis (SROB). The fluctuations depend on a set of hyperparameters $\boldsymbol{\alpha} \in \mathcal{C}_{\boldsymbol{\alpha}} \subset \mathbb{R}^{m_\alpha}$, where m_α is the number of hyperparameters. Based on samples from the SROB, the ROM is rebuild as many times as it takes to get sufficient statistic of the quantity of interest $\mathbf{O}(t;\boldsymbol{\mu}) = (O_1(t;\boldsymbol{\mu}), \ldots, O_{m_0}(t;\boldsymbol{\mu}))$ in $\mathbb{R}^{m_0}$ computed by every ROM. The uncertainty is then described by the distribution of the resulting values. Therefore, the overall goal of this approach is to cover the difference of the ROM to the FOM with the variance of the SROM. This can be achieved by optimising $\boldsymbol{\alpha}$ using a maximum likelihood method or a nonlinear Least-Squares (LS) method with a given set of state values. The states values are obtained from running the FOM for different samples from a defined parameter space $\mathcal{C}_{\boldsymbol{\mu}} \subset \mathbb{R}^{m_\mu}$, where m_μ is the number of parameters.

In the following, we sketch the FOM and the ROM of the problem Soize and Farhat used in their paper to introduce the SROM. Subsequently, in Section 2.2.2, we used this problem to build a SROM upon and explain in Section 2.2.4 how to optimise the hyperparameters of the SROB to create a SROM which covers the uncertainties of the ROM.

2.2.1. Example Problem

In the following we quickly describe the FOM and the ROM of the nonlinear transient structural mechanics problem used to introduce the SROM [36, 38].

Full Order Model

The problem is defined on a domain $\Omega \in \mathbb{R}^d$. Semi-discretised by a Finite Element (FE) method, a large class of nonlinear transient structural mechanic problems can be written as

$$[M]\ddot{\mathbf{y}}(t) + \mathbf{g}(\mathbf{y}(t), \dot{\mathbf{y}}(t); \boldsymbol{\mu}) = \mathbf{f}(t;\boldsymbol{\mu}), \quad t \in]t_0, t_f]\,, \tag{2.15}$$

where $\mathbf{y}(t) \in \mathbb{R}^N$ is the state vector at time t, $\mathbf{f}(t;\boldsymbol{\mu}) \in \mathbb{R}^N$ the external force vector for a certain time t and a parameter set $\boldsymbol{\mu}$, $[M] \in \mathbb{M}_N^+$ the FE mass matrix and $\mathbf{g}(\mathbf{y}(t), \dot{\mathbf{y}}(t); \boldsymbol{\mu}) \in \mathbb{R}^N$ represents the internal force at time t for a given parameter set $\boldsymbol{\mu}$. The boundary conditions and linear constrains of this system can be written in the form

$$[B]^\top \mathbf{y}(t) = \mathbf{0}_{N_{CD}}, \quad t \in [t_0, t_f], \tag{2.16}$$

where $N_{CD} < N$ is the number of constrains and $[B]$ is a matrix in $\mathbb{M}_{N,N_{CD}}$, satisfying

$$[B]^\top [B] = [I_{N_{CD}}]. \tag{2.17}$$

Additionally, we define the number of nodes N_o and the dimension m of the field discretised by the Finite Element method. The total number of degrees of freedom (DOF) is then $N = m\ N_o$.

Projection-Based Reduced-Order Model

The ROM is built based on a ROB $[V] \in \mathbb{M}_{N,n}$ with $n << N$, introduced in Section 2.1.1. This basis is satisfying both, the constraint equation

$$[B]^\top [V] = [0_{N_{\text{CD}},n}] \,, \tag{2.18}$$

and the orthonormality condition

$$[V]^\top [V] = [I_n]. \tag{2.19}$$

We can now use $[V]$ for building the ROM corresponding to Equation [2.15]

$$\mathbf{y}^{(n)}(t) = [V]\mathbf{q}(t), \quad t \in [t_0, t_f], \tag{2.20}$$

$$[V]^\top [M][V]\ddot{\mathbf{q}}(t) + [V]^\top \mathbf{g}([V]\mathbf{q}(t), [V]\dot{\mathbf{q}}(t); \boldsymbol{\mu}) = [V]^\top \mathbf{f}(t; \boldsymbol{\mu}), \quad t \in]t_0, t_f]. \tag{2.21}$$

2.2.2. Stochastic Reduced-Order Model

For building the SROM, the ROB $[V]$ has to be substituted by a SROB $[\mathbf{W}] \in \mathbb{M}_{N,n}$. The SROB is a random variable defined on the probability space $(\Theta, \mathcal{T}, \mathcal{P})$. It meets, like its deterministic predecessor, the constraint equation [2.18] and the orthonormality condition [2.19]. The probability distribution $P_{[\mathbf{W}]}$ of $[\mathbf{W}]$ depends on $\boldsymbol{\alpha}$. The SROM can then be written like

$$\begin{aligned} \mathbf{Y}^{(n)}(t) &= [\mathbf{W}]\mathbf{Q}(t), \quad t \in [t_0, t_f], \\ [\mathbf{W}]^\top [M][\mathbf{W}]\ddot{\mathbf{Q}}(t) + [\mathbf{W}]^\top \mathbf{g}([\mathbf{W}]\mathbf{Q}(t), [\mathbf{W}]\dot{\mathbf{Q}}(t)) &= [\mathbf{W}]^\top \mathbf{f}(t), \quad t \in]t_0, t_f]. \end{aligned} \tag{2.22}$$

2.2.3. Construction of the Stochastic Reduced-Order Basis

The SROB is built based on the deterministic ROB $[V]$ living on the compact Stiefel manifold $\mathbb{S}_{N,n}$ defined as

$$\mathbb{S}_{N,n} = \{[\tilde{V}] \in \mathbb{M}_{N,n} : [\tilde{V}]^\top [\tilde{V}] = [I_n], [B]^\top [\tilde{V}] = [0_{N_{CD}}, n]\} \subset \mathbb{M}_{N,n}, \tag{2.23}$$

and the hyperparameter vector $\boldsymbol{\alpha} = (s, \beta, [\sigma])$, where s and β are real scalar values, and $\sigma \in \mathbb{M}_n^u$ is an upper triangular matrix with positive entries. The vector belongs to the admissible set

$$\begin{aligned} \mathcal{C}_\alpha = \{ & 0 \leq s_l \leq s \leq s_u \leq 1, \\ & 0 \leq \beta_l \leq \beta \leq \beta_u, \\ & \sigma_l \leq [\sigma]_{11}, \ldots, [\sigma]_{nn} \leq \sigma_u, [\sigma]_{kk'} \in \mathbb{R}, k < k' \}, \end{aligned} \tag{2.24}$$

where $s_l, s_u, \beta_l, \beta_u, \sigma_l$ and σ_u represent lower and upper boundaries for the parameters and can be considered as given. The size of $\boldsymbol{\alpha}$ and, therefore, the number of hyperparameters is $m_\alpha = 2 + \frac{n}{2}(n+1)$. The usage of the values in $\boldsymbol{\alpha}$ is going to be described in the course of this section. Equations [2.25]-[2.30] describe the whole construction of the SROB. Afterwards, each step is explained in detail.

$$[\mathbf{U}] = [\mathbf{G}(\beta)][\sigma], \tag{2.25}$$

$$[\mathbf{A}] = [\mathbf{U}] - [B]([B]^\top[\mathbf{U}]), \tag{2.26}$$

$$[\mathbf{D}] = \Big([V]^\top[\mathbf{A}] + [\mathbf{A}]^\top[V]\Big)/2, \tag{2.27}$$

$$[\mathbf{Z}] = [\mathbf{A}] - [V][\mathbf{D}], \tag{2.28}$$

$$[H_s([\mathbf{Z}])] = \Big([I_n] + s^2[\mathbf{Z}]^\top[\mathbf{Z}]\Big)^{-1/2}, \tag{2.29}$$

$$[\mathbf{W}] = ([V] + s[\mathbf{Z}])\,[H_s([\mathbf{Z}])]\,. \tag{2.30}$$

Before Equation [2.25] , we start with generating the second-order, centred random matrix $[\mathbf{G}(\beta)] \in \mathbb{M}_{N,n}$. For constructing $[\mathbf{G}(\beta)]$, we define a Gaussian random field $\{\mathcal{Q}(\mathbf{x}), \mathbf{x} \in \mathbb{R}^d\}$ with a spatial correlation length of

$$\mathcal{L}_i = \beta \max_{(\mathbf{x},\mathbf{x}')\in\Omega\times\Omega} |x_i - x_i'|\,, \tag{2.31}$$

The matrix $[\mathbf{G}(\beta)]$ is then built out of independent copies of $\mathcal{Q}$. More precisely, for all $\ell = 1,\ldots,m$, for all $j_o = 1,\ldots,N_o$, and for all $k = 1,\ldots,n$,

$$[\mathbf{G}(\beta)]_{jk} = \left[\mathcal{G}\left(\mathbf{x}^{j_o}\right)\right]_{\ell k} \quad , \quad j = (\ell, j_o) \in \{1,\ldots,N\},$$

are the random fields $\{[\mathcal{G}(\mathbf{x})]_{\ell k}, \mathbf{x} \in \mathbb{R}^d\}$ copies of $\mathcal{Q}$. The exact construction is explained in more detail in the work of Soize and Farhat [36]. In summary, the input parameter β controls the statistical correlations between the components of the columns of $[\mathbf{G}(\beta)]$, as the spatial correlation length is linearly dependent on it. Furthermore, it can be shown that the probability distribution of $[\mathbf{G}(\beta)]$ is not Gaussian, but is such that

$$p_{[\mathbf{G}(\beta)]}(-[G]) = p_{[\mathbf{G}(\beta)]}([G]) \quad , \quad \forall [G] \in \mathbb{M}_{N,n}. \tag{2.32}$$

Equation [2.25] is the construction of the centred random matrix $[\mathbf{U}] \in \mathbb{M}_{N,n}$ with

$$E\{[\mathbf{U}]\} = [0_{N,n}]. \tag{2.33}$$

For the purpose of optimising a cost function that depends indirectly on $[\mathbf{U}]$, Soize and Farhat introduced a parameter efficient way of controlling the correlations in each column of $[\mathbf{U}]$ via $[\mathbf{G}(\beta)]$ with only one hyperparameter β. The correlation between the columns of $[\mathbf{U}]$ is controlled via a matrix $[\sigma]$ with $\frac{n}{2}(n+1)$ hyperparameters. The alternative would have been to parameterise the fourth-order symmetric correlation tensor with $\frac{Nn}{2}(Nn+1)$ hyperparameters, where N can be very large. Having $[\mathbf{U}]$, we can enforce the boundary conditions via Equation [2.26] .

The next logical step is to map $[\mathbf{A}]$ onto the to $\mathbb{S}_{N,n}$ tangential vector space $T_V\mathbb{S}_{N,n}$. From $[V]^\top[V] = [I_n]$, we take the derivative of $[V]$ and get $[V]^\top[dV] + [dV]^\top[V] = [0_{n,n}]$, which leads to the tangent vector space

$$T_V\mathbb{S}_{N,n} = \left\{[\tilde{Z}] \in \mathbb{M}_{N,n} : [V]^\top[\tilde{Z}] + [\tilde{Z}]^\top[V] = [0_{n,n}]\right\}. \tag{2.34}$$

Then, in Equation [2.30] , we multiply the random matrix $[\mathbf{Z}]$ from$T_V\mathbb{S}_{N,n}$ with the last hyperparameter s, add it to the original basis $[V]$ and normalise the result with $[\mathbf{H}_s]$ calculated in Equation [2.29] . The scalar value s is used to control the amplitude of the fluctuations. Adding a matrix from $T_V\mathbb{S}_{N,n}$ ensures that $[\mathbf{W}]$ fulfils the constrain equation 2.19, and as we have already enforced the boundary conditions on $[\mathbf{U}]$ in Equation [2.26] , $[\mathbf{W}]$ also fulfils Equation [2.18] and is therefore in $\mathbb{S}_{N,n}$.

2.2.4. Optimisation of SROB Hyperparameters

To eventually get a SROM that covers the error of the ROM to the FOM correctly, the hyperparameters still have to be optimised. Soize proposed a nonlinear LS method with a cost function of the form

$$J(\boldsymbol{\alpha}) = w_J J_{\text{mean}}(\boldsymbol{\alpha}) + (1 - w_J)\, J_{\text{std}}(\boldsymbol{\alpha}), \tag{2.35}$$

where w_J is the weight between zero and one, balancing the contribution of $J_{\text{mean}}(\boldsymbol{\alpha})$, the term which represents the error between the expectation value of the quantity of interest (QoI) from the SROM $E\left\{\mathbf{O}^{(n)}(\boldsymbol{\alpha},\boldsymbol{\mu})\right\}$, and the reference values of the QoI from the FOM $\mathbf{o}^{\text{ref}}$, and $J_{\text{std}}(\boldsymbol{\alpha})$, the term representing how well the error of the ROM to the FOM model is represented in the standard deviation of the SROM. The two contributing terms are defined as follows

$$J_{\text{mean}}\left(\boldsymbol{\alpha}\right) = \frac{1}{c_{\text{mean}}\left(\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_{m_\mu}\right)} \sum_{i=1}^{m_\mu} \int_{t_0}^{t_f} \left\| \mathbf{o}^{\text{ref}}\left(t;\boldsymbol{\mu}_i\right) - E\left\{\mathbf{O}^{(n)}\left(t;\boldsymbol{\mu}_i,\boldsymbol{\alpha}\right)\right\}\right\|^2 dt, \tag{2.36}$$

$$J_{\text{std}}(\boldsymbol{\alpha}) = \frac{1}{c_{\text{std}}\left(\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_{m_\mu}\right)} \sum_{i=1}^{m_\mu} \int_{t_0}^{t_f} \left\| \mathbf{v}^{(\text{ref},n)}\left(t;\boldsymbol{\mu}_i\right) - \mathbf{v}^{(n)}\left(t;\boldsymbol{\mu}_i,\boldsymbol{\alpha}\right)\right\|^2 dt, \tag{2.37}$$

where

$$c_{\text{mean}}\left(\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_{m_\mu}\right) = \sum_{i=1}^{m_\mu} \int_{t_0}^{t_f} \left\| \mathbf{o}^{\text{ref}}\left(t;\boldsymbol{\mu}_i\right)\right\|^2 dt, \tag{2.38}$$

$$c_{\text{std}}\left(\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_{m_\mu}\right) = \sum_{i=1}^{m_\mu} \int_{t_0}^{t_f} \left\| \mathbf{v}^{(\text{ref},n)}\left(t;\boldsymbol{\mu}_i\right)\right\|^2 dt, \tag{2.39}$$

are constant normalisation factors. Here $o^{(n)}(t;\boldsymbol{\mu})$ is the deterministic solution of the ROM for a certain parameter set $\boldsymbol{\mu}$ and time t, and

$$\mathbf{v}^{(\mathrm{ref},n)}(t;\boldsymbol{\mu}) = \gamma \left| \mathbf{o}^{\mathrm{ref}}(t;\boldsymbol{\mu}) - o^{(n)}(t;\boldsymbol{\mu}) \right|, \tag{2.40}$$

$$\mathbf{v}^{(n)}(t;\boldsymbol{\mu},\boldsymbol{\alpha}) = \left(E\left\{ O^{(n)}(t;\boldsymbol{\mu},\boldsymbol{\alpha})^2 \right\} - \left(E\left\{ O^{(n)}(t;\boldsymbol{\mu},\boldsymbol{\alpha}) \right\} \right)^2 \right)^{1/2}. \tag{2.41}$$

The parameter $\gamma > 0$ defines a target value of how much of the difference between FOM and ROM shall be covered by the standard deviation of the SROM. This results in a statistical inverse non-convex optimisation problem,

$$\boldsymbol{\alpha}^{\mathrm{opt}} = \min_{\boldsymbol{\alpha}\in\mathcal{C}_a} J(\boldsymbol{\alpha}), \tag{2.42}$$

subject to Equation [2.22] . This problem can be solved via the interior-point algorithm or via probabilistic techniques as they are used in genetic algorithms. However, the deterministic interior-point method outperforms various genetic algorithms for the given problem type [38]. Originally, proposed in [36], the problem was solved via a four stage optimisation algorithm based on the interior-point method. The gradient has been calculated in a finite difference fashion, which made it quite expensive. Therefore, the optimisation has been split up to get an approximate solution on stages with few parameters and thus spent less steps on stages with more parameters. This algorithm has been used for testing the developed tool in this thesis, and is explained in detail in the Appendix A.2.

For the cost function evaluation we have to evaluate the SROM for all the m_μ samples of the parameter space we are training on, which takes m_μ times ν_s evaluations of a ROM, where ν_s is the number of samples drawn from the SROB for the SROM. The number of samples ν_s is considered to be 1,000 in this thesis. This makes the cost function evaluation one of the most expensive parts in the optimisation. Therefore, different methods were tried to overcome this difficulty. An intrusive version was developed by Charbel Farhat. He reformulated the problem around the concept of hyper-reduction and achieved on their machinery a speedup of approximately 30. He used the mesh of the FOM to hyper-reduce the SROM [10].

Soize proposed a diffusion map based algorithm which works fully non-intrusive and achieved a speedup up to 56 to the original problem formulation [38]. It uses a predictor-corrector approach with an interior-point method in the corrector. The main improvement originates from a diffusion map that locally approximates the SROM close to a minimum of the cost function. The construction is done based on only a few SROM evaluations, approximately the same number as happening during two cost function evaluations. The diffusion map can then be sampled for evaluating the cost function [38].

2.3. Optimisation

Since a basic understanding of the interior-point optimisation algorithm is going to be important for future steps in this thesis, it will be explained in the following.

2.3.1. Interior Point Optimisation

The IPopt library [16] was used to solve the optimisation problems in the numerical experiments in Chapter 3. In the following we introduce the primal-dual barrier approach from the IPopt implementation paper [41] and describe the parameters that have to be chosen. The general form of problems that the IPopt library is able to tackle, can be written as

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}), \tag{2.43}$$

$$\text{s.t. } c(\boldsymbol{x}) = 0, \tag{2.44}$$

$$\boldsymbol{x}_L \leq \boldsymbol{x} \leq \boldsymbol{x}_U, \tag{2.45}$$

where $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is the objective function, $c(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$, where $m \leq n$ is the number of constrains. The third equation shows linear inequality constrains for $\boldsymbol{x}$ with upper and lower bounds $\boldsymbol{x}_L$ and $\boldsymbol{x}_U$ that have to be set by the user.

The interior point algorithm computes approximate solutions for a sequence of barrier problems

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \varphi_{\mu_j}(\boldsymbol{x}) = f(\boldsymbol{x}) - \mu \sum_{i\in I_L} \ln\left(x^{(i)} - x_L^{(i)}\right) - \mu \sum_{i\in I_U} \ln\left(x_U^{(i)} - x^{(i)}\right), \tag{2.46}$$

$$\text{s.t. } c(\boldsymbol{x}) = 0, \tag{2.47}$$

for μ converging to zero, and where $I_L = \left\{i : x_L^{(i)} \neq -\infty\right\}$ and $I_U = \left\{i : x_U^{(i)} \neq \infty\right\}$. Setting the upper and lower bound is optional. In the unbounded case the barrier is set to infinity. This problem is equivalent to applying a homotopy method to the primal-dual equations,

$$\nabla f(\boldsymbol{x}) + \nabla c(\boldsymbol{x})\boldsymbol{\lambda} - \boldsymbol{z}_L + \boldsymbol{z}_U = 0, \tag{2.48}$$

$$c(\boldsymbol{x}) = 0, \tag{2.49}$$

$$(\boldsymbol{x}_L - \boldsymbol{x}) \circ \boldsymbol{z}_L + (\boldsymbol{x} - \boldsymbol{x}_U) \circ \boldsymbol{z}_U - \mu\boldsymbol{e} = 0, \tag{2.50}$$

where $\boldsymbol{e}$ is a vector with only ones as entries, μ is the homotopy parameter which goes to zero [7], $\lambda \in \mathbb{R}^m$ are the Lagrangian multipliers for the equality constrains in Equation [2.44] , $\boldsymbol{z}_L, \boldsymbol{z}_U \in \mathbb{R}^n$ are the Lagrangian multipliers for the inequality constrains in Equation [2.45] , and $\circ$ stands for the Hadamard product (element wise multiplication). Equation [2.48] is the gradient condition which has to be satisfied for an optimum. It can be seen that for $\mu = 0, \boldsymbol{x} \geq 0$ and for $\boldsymbol{z}_L, \boldsymbol{z}_U \geq 0$ the primal-dual equations are equivalent to the first order optimality Karush-Kuhn-Tucker (KKT) conditions. Equation [2.49] is the first condition for primal feasibility, and Equation [2.50] is the complementary slackness condition. We assume that the inequality constrains are fulfilled for primal feasibility. With the primal dual equations, the optimality error is defined as

$$E_\mu(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{z}) := \max \left\{ \frac{\|\nabla f(\boldsymbol{x}) + \nabla c(\boldsymbol{x})\boldsymbol{\lambda} - \boldsymbol{z}_L + \boldsymbol{z}_U\|_\infty}{s_d}, \|c(\boldsymbol{x})\|_\infty, \right. \\ \left. \frac{\|(\boldsymbol{x}_L - \boldsymbol{x}) \circ \boldsymbol{z}_L + (\boldsymbol{x} - \boldsymbol{x}_U) \circ \boldsymbol{z}_U - \mu \boldsymbol{e}\|_\infty}{s_c} \right\}, \tag{2.51}$$

with

$$s_d = \max \left\{ s_{\max}, \frac{\|\boldsymbol{\lambda}\|_1 + \|\boldsymbol{z}_L\|_1 + \|\boldsymbol{z}_U\|_1}{(m+2n)} \right\} / s_{\max}, \tag{2.52}$$

$$s_c = \max \left\{ s_{\max}, \frac{\|\boldsymbol{z}_L\|_1 + \|\boldsymbol{z}_U\|_1}{2n} \right\} / s_{\max} \tag{2.53}$$

where $s_{max} \geq 1$. The scalar s_{max} is set to 1000 in the IPopt library.

This method iteratively computes an approximate solution for the barrier problem in Equation [2.46] and Equation [2.47] . The barrier-parameter gets updated and the optimality error is calculated in every iteration. In case an approximate solution $\left(\tilde{\boldsymbol{x}}_*, \tilde{\boldsymbol{\lambda}}_*, \tilde{\boldsymbol{z}}_*\right)$ fulfils

$$E_0 \left(\tilde{\boldsymbol{x}}_*, \tilde{\boldsymbol{\lambda}}_*, \tilde{\boldsymbol{z}}_*\right) \leq \epsilon_{\text{tol}}, \tag{2.54}$$

the algorithm terminates with $\tilde{x}_*$ as optimal solution. The tolerance ϵ_{tol} has to be set manually, which is also covered in Section 3.2.2. However, this is not the only exit criteria used in this thesis. Not computing the Hessian, which was not done in this thesis, can lead to problems with bringing down the dual infeasibility. Therefore, in case the algorithm was not converging, we either used the option of setting a threshold for an acceptable change of the objective function scaled by $\max(1, |f(\boldsymbol{x})|)$, or we chose an acceptable tolerance above the actual tolerance ϵ_{tol}. There are multiple thresholds that a solution has to fulfil to be acceptable. Therefore, for making sure that one specific thresholds is the one which is going to decide whether the solution is acceptable, all the other criteria have to be set to values which are always met. This is the case for the standard values of all of the thresholds, expect for the acceptable tolerance where the default value is 10^{-6}. Once all the values are below their acceptable thresholds, this state has to be held for a certain number of optimisation iterations to cause the termination of the algorithm. This number can be chosen by the user.

The exact algorithm with all its settable options can be looked up in [41]. The problem is solved via a filter line-search algorithm. The interactions from the algorithm with computational expensive parts of the user defined problem are calculating the objective function and its gradient. For optimising the hyperparameters $\boldsymbol{\alpha}$ of a SROM, there have already been developed approaches to minimise the computational cost of the cost function evaluation, as stated in Section 2.2.4. However, to the best of my knowledge it has not been tried yet to optimise the gradient evaluation, which is done in this thesis. Since the original algorithm, described in Appendix A.2, calculates the gradient in a FD fashion and is therefore dependent on the size of the hyperparameter vector $\boldsymbol{\alpha}$, an analytical gradient would be quite an improvement. It is not only more accurate, but also not directly dependent on the size of $\boldsymbol{\alpha}$. The FD gradient approximation and the analytical gradient computation via solving the adjoint problem is explained in the following.

2.3.2. Gradient Approximation

One of the main improvements to the original algorithm developed in this thesis arises from the analytical gradient. Therefore, the originally used FD approximation and the gradient computation via solving the adjoint problem are introduced in this section to be able to compare them in Chapter 3.

Finite Differences Approximation

The FD approximation has been done with a central difference scheme which is described for a scalar input x, but can easily be extended to a vector input,

$$f(x)' \approx \frac{f(x+h) - f(x-h)}{2h}, \tag{2.55}$$

where h is a shift of the input to approximate the change of the output. The function f has to be evaluated twice for every parameter of interest. Hence, the cost of the gradient is linearly increasing with the number of parameters, which makes the evaluation quite expensive for a large number of parameters. Furthermore, a bad choice for h causes rather large errors. Therefore, this value has to be chosen in a way that minimises them.

The occurring errors are determined in the course of this section. Calculating the derivative in a FD way results in two type of errors: a truncation error ϵ_t and a round off error ϵ_r. The first, can easily be computed as it results from terms of higher order in the Taylor expansion,

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(x) + \cdots . \tag{2.56}$$

Writing Equation [2.55] in that fashion yields

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{1}{6}h^2 f'''(x) + \cdots , \tag{2.57}$$

which shows that the truncation error is roughly $\epsilon_t \sim h^2 f'''$.

The round off error can have two reasons. The first contribution to the round off error could be a value of h that is not exactly representable in the binary format, like $0.1_{10} \triangleq 0.0\overline{0011}_2$. For double precision, the machine precision is $\epsilon_m \sim 10^{-16}$. This would imply a fractional error of $\sim \epsilon_m x/h$. This error can be nullified by making sure to use an exactly binary representable number for h:

$$\begin{aligned} temp &= x + h, \\ h &= temp - x. \end{aligned} \tag{2.58}$$

In a compiler language, one has to make sure that Equation [2.58] in the code is not deleted in the optimisation procedure. This problem is usually not given in an interpreted language.

Once h is an exactly representable number, the round off error is $\epsilon_r \sim \epsilon_f |f(x)/h|$, where ϵ_f is the fractional accuracy f is computed with. For simple functions it is roughly machine precision, but increases for more complex functions. Even if the system we are working with in this thesis is rather complex, we consider $\epsilon_f \approx \epsilon_m$ as rough estimate.

The total error is then $\epsilon_r + \epsilon_t$. By taking the derivative with respect to h and setting it to zero we get a rough estimate of the optimal h

$$h \sim \left(\frac{\epsilon_f f}{f'''}\right)^{1/3} \approx (\epsilon_f)^{1/3} x_c \approx (\epsilon_m)^{1/3} x \tag{2.59}$$

where $x_c \equiv (f/f'')^{1/2}$ is the characteristic scale of the function f. If x_c is not given, we assume $x_c = x$. If x is equal to zero, we set $h = (\epsilon_m)^{1/3}$. There are other ways to chose the optimal h, but they usually include additional function evaluations and are therefore not considered in this thesis [29].

Adjoint Problem

The gradient of nonlinear function can be computed in an analytical way by solving the adjoint problem. In the following, we would like to introduce the idea of the adjoint method based on a simple example from [12] and [32].

A variable $\boldsymbol{u}(\boldsymbol{p})$ depending on a collection of parameters $\boldsymbol{p} \in \mathbb{R}^m$ is considered. This variable is the input to a scalar function $g(\boldsymbol{u}(\boldsymbol{p}))$, which we want to optimise and therefore calculate its gradient regarding $\boldsymbol{p}$. The first step is to form the Lagrangian to that problem, where intermediate variables, such as $\boldsymbol{u}$ are considered as independent. To account for the missing dependencies we add them as constrains with Lagrange multipliers. For the current example the Lagrangian looks like

$$\mathcal{L} = g(\bar{\boldsymbol{u}}) - \boldsymbol{\lambda}^\top (\bar{\boldsymbol{u}} - \boldsymbol{u}(\boldsymbol{p})) = g(\boldsymbol{u}(\boldsymbol{p})), \tag{2.60}$$

where $g(\boldsymbol{u})$ is the function we want to optimise and $\boldsymbol{\lambda} \in \mathbb{R}^n$ is a vector of Lagrange multipliers for the constrain equation. The vector $\boldsymbol{\lambda}$ can be chosen freely as the constrain equation is always zero. The fact that the constrain is always zero results in $\mathcal{L} = g(\boldsymbol{u}(\boldsymbol{p})))$ and therefore,

$$\frac{\mathrm{d}g(\boldsymbol{u}(\boldsymbol{p}))}{\mathrm{d}\boldsymbol{p}} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{p}} = \boldsymbol{\lambda}^\top \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}\boldsymbol{p}}, \tag{2.61}$$

which is the adjoint equation. However, as we want $\mathcal{L}$ to be only dependent on $\boldsymbol{p}$, we take the derivative regarding $\bar{\boldsymbol{u}}$ and force it to be zero

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{u}} = \frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{u}} - \boldsymbol{\lambda} = 0. \tag{2.62}$$

This results in

$$\boldsymbol{\lambda} = \frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{u}}. \tag{2.63}$$

Substituting the calculated lambda into the adjoint equation 2.61

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{p}} = \boldsymbol{\lambda}^{\top}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}\boldsymbol{p}} = \frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{u}}^{\top}\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}\boldsymbol{p}} = \frac{\mathrm{d}g(\boldsymbol{u}(\boldsymbol{p}))}{\mathrm{d}\boldsymbol{p}}, \tag{2.64}$$

leads to the total derivative of $g(\boldsymbol{u}(\boldsymbol{p}))$ regarding $\boldsymbol{p}$, which is the prove for correctness. The advantage of this approach is that the gradient of complicated systems can be computed in the form of Equation [2.61] with only partial derivatives of $\boldsymbol{p}$ and without ever calculating $\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}\boldsymbol{p}}$, which is often quite difficult in case the function is given only implicitly as the solution of a nonlinear equation, like in this thesis.

One of the drawbacks of this approach is that once the problem gets more complicated and more constrains are added, intermediate results are needed for the calculation of the different Lagrangian multipliers and therefore have to be stored during the forward calculation. This can be seen later in Section 3.2.3. A naive implementation storing all intermediate results would require a tremendous amount of memory which makes it infeasible. Hence, the “reverse mode” automatic differentiation is not an option [13]. In this thesis we take on the challenge of computing the analytical gradient ”by hand”, thus obtaining a memory efficient way to optimise the cost function.

3. Optimised Nonparametric Probabilistic Approach for Uncertainty Quantification for Operator Inference Based Reduced Order Models

In the main part of this thesis, we start with describing the code structure of the implemented tool for SROMs. We introduce the most important objects, how they have to be combined and how they can be replaced or extended. In Section 3.2, we use the generic static model from the paper form Soize and Farhat [36] to show the basic procedure of building and training a SROM and to introduce the improvements we made to the algorithm in form of the analytical gradient and an alternative optimisation algorithm. In the end of that section, we show that the confidence intervals gained from optimising the SROM over the nonlinear LS cost function proposed from Soize and Farhat had problems with covering values close to the boundaries and explain why those problems could be solved with optimising over the NLML function. In the third Section 3.3 we show how to apply the in Section 3.1 introduced framework to models implemented with the Python library pyMOR and show that not including the grid points for building the ROB led to more confident confidence intervals, but the SROB also worked without using this information. Afterwards, in Section 3.4 we discuss two OI-based ROM and show that the nonparametric probabilistic approach for UQ is capable of describing the error of OI-based ROM regarding the FOM.

3.1. Implementation

This section describes the in this work developed software for building a SROM and for optimisation in a condensed matter, thus not all of the classes, functions and parameters are stated. Developing this software was necessary as the methodology of UQ based on a SROM is not implemented in any other commonly known software package yet. As implementation language we decided to use Python, as it is easy to use and a common language at Siemens and the UQ community. Furthermore, it has to be stated that this is not a code which can run in production. It is mainly for research purposes. The main goal is to provide a code base which is easily understandable and extendable. The code base can be found in [34].

The main task of the software developed in this thesis is to build a SROM and to optimise the parameters of the SROB in a way that the variance of the SROM covers the error from the ROM to the FOM. We tried to implement this tool to be modular and easily extendable according to future needs, facilitating adjustability, maintainability and testing. Most of the code is already covered by tests, which also can be looked at to get a better understanding of the structure. For the purpose of modularity we structured the SROM in a way that the model itself is completely separate from its optimisation, however it has to know its contribution to the adjoint problem regarding the SROB parameters. In this section, we describe at first the structure of the SROM and afterwards in Section 3.1.2 additional classes related to the optimisation of the SROB parameters.

3.1.1. Structure of the SROM

For the SROM we have to sample the SROB multiple times and generate the corresponding ROMs. The SROM does not directly have to know how to generate a ROM or how to sample a new ROB. Therefore, we distinguish in terms of classes between the SROM, SROB, and a factory which has all the information needed to generate a ROM. The ROM is in our case the reduced order problem (ROP) as it is able to solve the whole problem including integration over a certain time frame. The ROP-factory is problem specific and has to be provided by the user for every new system. For instance by using supportive structures provided in the code that are introduced later in this section.

The objects the user has to provide to generate a SROP are shown in Figure 3.1 . The first one is the reference solution which includes the state values for the POD, the input values and the time stamps in case of a dynamic problem. The state values are stored in three dimensions. The first dimension corresponds to different samples from the parameter space, the second stands for the DOF and the third dimension is time. The input values are stored equivalently. This class is not a pure data class as it is able to reduce its state values based on a ROB. It can also return an input parameter list which consists of all necessary information needed to run the simulations in every given point of the parameter space. The only object that has to be provided is the “SROBInput”-object which stores the things needed to generate a SROB. First of all, the only mandatory property that is stored is the ROB which is in our case the POD. Second, the ROB can additionally use the boundary matrix, the coordinates of the nodes and a mapping of each value on every node to the state vector, which are then used to restrict space that the SROB can vary in. They help to set additional bounds to the function space of the SROM, which can lead to more confident, means smaller, confidence intervals. However, the SROM works also without that information as shown in Section 3.3.

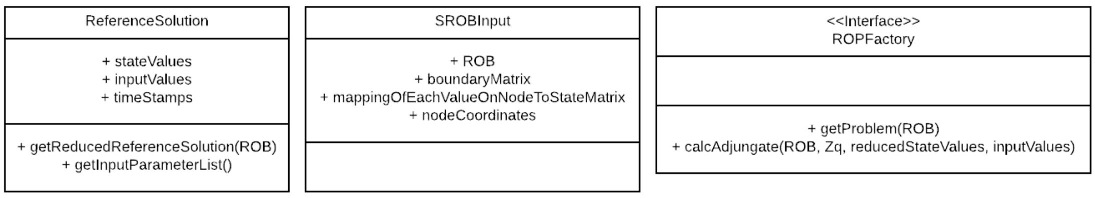
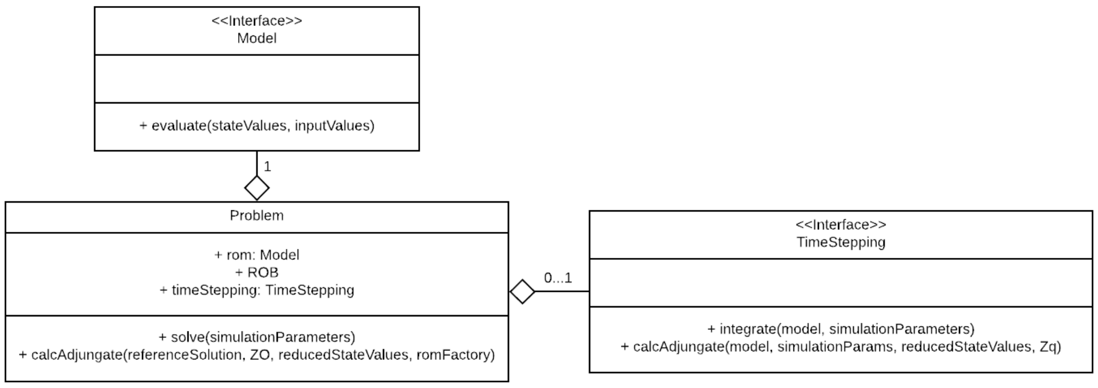

ReferenceSolution
+ stateValues
+ inputValues
+ timeStamps
+ getReducedReferenceSolution(ROB)
+ getInputParameterList()
SROBInput
+ ROB
+ boundaryMatrix
+ mappingOfEachValueOnNodeToStateMatrix
+ nodeCoordinates
<<Interface>>
ROPFactory
+ getProblem(ROB)
+ calcAdjungate(ROB, Zq, reducedStateValues, inputValues)

Figure 3.1.: Objects that have to be defined by the user to create a SROM for a certain problem. The "ReferenceSolution"-object which holds the snapshots of the FOM, the input values and the time stamps. This information is needed to build the POD and train the SROM. The second object is the "SROBInput"-object which is handed to the SROB-object later on. It holds the mandatory ROB and optional values like the boundary Matrix, the mapping of each value on every node to the state vector and the node coordinates. The third object is a ROP-factory, where the user has to implement its own factory with the "ROPFactory" interface.

The last mandatory object is the ROP-factory, which has to be implemented by the user. This class has two tasks. The first one is to produce a ROP based on a ROB. The second task is to compute the adjoint problem regarding each generated ROP, as the data used to generate those ROP is needed to do so, but a ROP does not need to know about it. This is not only design related but also memory related, as the data from the FOM is usually rather large and can not be stored in every instance of a ROP. The code base has already implemented three example factories, one for the generic static projection based model in Section 3.2, one for the pyMOR static problem in Section 3.3, and one for the operator inference models used in Section 3.4. Where the factories for the two static problems generate projection based ROMs, the factory for the dynamic problems in this thesis returns OI-ROMs. The inference-based factory can be used for any problem. The projection based factories, however, are problem specific and have to be rewritten for a new system. A general version of a projection based factory is an open issue.

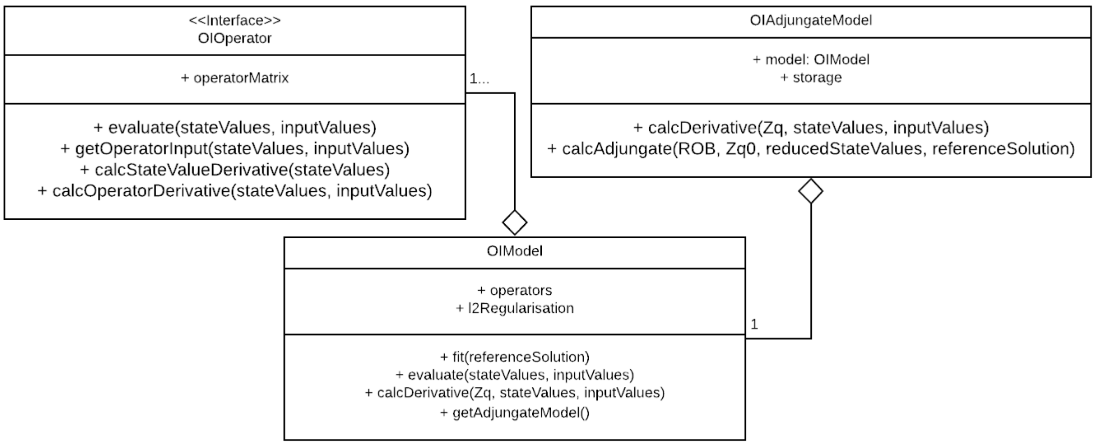
The problem returned by the ROP-factory has to be of the form shown in Figure 3.2 , which is the supportive structure we have already mentioned. The problem class holds at least the sample from the SROB and the ROP generated from that sample. If the problem is dynamic, it needs a time stepping scheme in addition. In the code itself we distinguish between static and dynamic problems. For reasons of simplicity, this distinction is neglected in the following. Each object is aware of its contribution to the adjoint problem. However, for a "Model"-object this is only the case, if its operators depend on the reduced state data, like for the OI model. For a projection based model this is not true. Therefore, the adjoint problem is handled by the factory. Regarding the time stepping scheme, the only implemented time stepping scheme is Heun, which is described in Appendix A.1.3 with its contribution to the adjoint problem of Section 3.2.3.



<<Interface>>
Model
+ evaluate(stateValues, inputValues)
1
Problem
+ rom: Model
+ ROB
+ timeStepping: TimeStepping
+ solve(simulationParameters)
+ calcAdjungate(referenceSolution, ZO, reducedStateValues, romFactory)
0...1
<<Interface>>
TimeStepping
+ integrate(model, simulationParameters)
+ calcAdjungate(model, simulationParams, reducedStateValues, Zq)

Figure 3.2.: UML diagram of the problem-class with its "has-a" relations. The problem-class holds the ROB and the corresponding ROM, to be able to map back the reduced order result from the ROM to the high dimensional space and return the approximate solution of the FOM. For dynamic problems it holds additionally a time stepping scheme. Furthermore, the diagram shows the interface for a model and a time stepping scheme. A model has to implement at least an evaluate function that returns a result for a state vector and a vector of input values. A time stepping scheme hast to implement at least an integrate method which integrates a certain model based on simulation parameters, namely timestamps, initial values and input values.

The model used for the dynamic problems in this thesis is the OI model, shown in Figure 3.3 . It is generated by the OI-factory, which is provided in the code, and can be technically used for any data set of a dynamic system. The model is defined by a given dataset, a string that defines the operators that are going to be inferred, and the ROB used to reduce the state values. In this thesis, all OI models have L^2-regularisation, but the code allows to turn it off. Based on the operator string given by the factory, we initialise the chosen operators and define them by inference via the fit function. There is a constant, linear, quadratic and input operator implemented. New operators can be added by simply creating an operator according to the "OIOperator" interface. We had to introduce the "OIAdjungateModel"-object for solving the adjoint problem in a decoupled way, as the integration scheme can call the model arbitrarily often but the model has to be aware of that as we need the corresponding Lagrangian variable to every evaluation of the model to compute the next Lagrangian variable in the adjoint problem computation, see Appendix A.1.4. The "OIAdjungateModel"-object is used as storage container to decouple the time stepping scheme from the model. To be able to compute the Lagrangian multipliers of the model without knowing how often it is called, the Lagrangian multipliers regarding each model evaluation are stored during every invocation of the model derivative. This allows us to use a different time stepping scheme without changing the rest of the code.



<<Interface>>
OIOperator
+ operatorMatrix
+ evaluate(stateValues, inputValues)
+ getOperatorInput(stateValues, inputValues)
+ calcStateValueDerivative(stateValues)
+ calcOperatorDerivative(stateValues, inputValues)
1...
OIAdjungateModel
+ model: OIModel
+ storage
+ calcDerivative(Zq, stateValues, inputValues)
+ calcAdjungate(ROB, Zq0, reducedStateValues, referenceSolution)
OIModel
+ operators
+ l2Regularisation
+ fit(referenceSolution)
+ evaluate(stateValues, inputValues)
+ calcDerivative(Zq, stateValues, inputValues)
+ getAdjungateModel()
1

Figure 3.3.: The UML diagram shows the class for OI-models, which consist of multiple operators and an L^2-regularisation. Each operator implements the “OIOperator”-interface. The “evaluate”-function is already implemented and multiplies the operator matrix with the result of the “getOperatorInput”-function, which must be implemented. Additionally, one must implement the derivative of the operator regarding the state vector and regarding the operator matrix to be able to compute the analytical gradient. To store the previously mentioned Lagrangian mutlipliers we introduced the “OIAdjungateModel”-object which functions as wrapper around the OI-model. With the stored Lagrangian multipliers it can compute the Lagrangian multipliers regarding the model without knowing how often the model was invoked.

With the SROB and the ROP-factory we can then build the SROP. It is initialised by sampling ν_s samples from the SROB and generating the corresponding ROP. Before that, a seed has to be set to get reproducible results and a valid gradient, as we resample the SROB multiple times during the optimisation, which is based on a random number generator. If we do not set a seed, we would get a non deterministic gradient. The models gained from each sample of the SROB are stored in the problem list. Evaluating the SROP means to loop over this list and evaluate each of the models. The statistics over all individual solutions yields the final result. The object provides functions to evaluate the expectation value and expectation of the squared value, which is needed to compute the variance. Furthermore, it can compute the deterministic solution from the ROP based on the ROB, and the confidence Intervals of a certain confidence value. Those functions always take simulation parameter as input, which include the initial values, the input values and the time frame of the simulation. In case lifting is used, the user has to implement a lifting model corresponding to the interface shown in Figure 3.4 and hand it to the SROP as well. Lifting is introduced in Section 2.1.2. It has to implement a “lift”- and “unlift”-

function which are then invoked by the "transform"- and "backTransform"-function. This model can now generate confidence intervals according to a certain SROB which has to be optimised to give reasonable results.

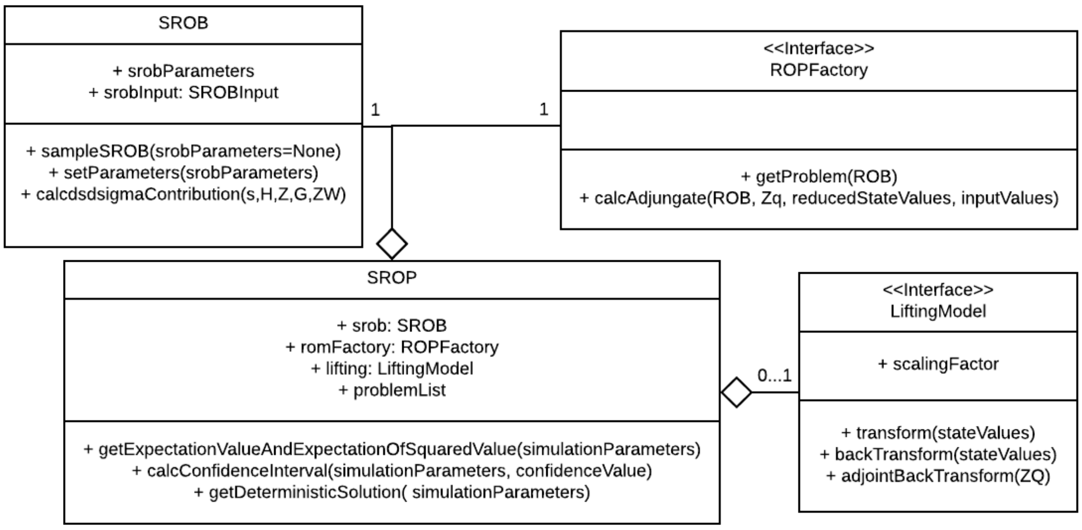

SROB
+ srobParameters
+ srobInput: SROBInput
+ sampleSROB(srobParameters=None)
+ setParameters(srobParameters)
+ calcdsdsigmaContribution(s,H,Z,G,ZW)
1
1
<<Interface>>
ROPFactory
+ getProblem(ROB)
+ calcAdjungate(ROB, Zq, reducedStateValues, inputValues)
SROP
+ srob: SROB
+ romFactory: ROPFactory
+ lifting: LiftingModel
+ problemList
+ getExpectationValueAndExpectationOfSquaredValue(simulationParameters)
+ calcConfidenceInterval(simulationParameters, confidenceValue)
+ getDeterministicSolution(simulationParameters)
0...1
<<Interface>>
LiftingModel
+ scalingFactor
+ transform(stateValues)
+ backTransform(stateValues)
+ adjointBackTransform(ZQ)

Figure 3.4.: The UML diagram in this figure shows the SROP-class and directly related classes. The SROP holds an instance of a SROB-class and an instance of a "ROPFactory"-object, which was already introduced. The "SROB"-object holds one instance of a "srobInput"-object with information regarding the ROB and optionally the grid coordinates and the boundary matrix. The object has a function to sample an instance of the through the "srobParameters"-object defined SROB and one to compute the contribution to the gradients regarding β and $[\sigma]$. In case lifting is used, the SROP-object must hold an instance of a "LiftingModel"-object to be able to map the result from the problem back to the unlifted space. The most important functions of the SROP are the computation of the confidence interval regarding a certain confidence and a set of simulation parameter, and the computation of the deterministic solution from the ROM based on the ROB.

3.1.2. Structure of the Optimisation

For the optimisation we first of all have to define a cost function. The main components of that class are shown in Figure 3.5 . It needs a SROP and a corresponding reference solution. Additionally, we define another wrapper, storing the intermediate results of the SROB and the end results of the ROM, which are needed to compute the gradient regarding the SROB parameters. There are already three different cost functions implemented in the code. Additional functions can be added by inheriting the cost object and implementing the “calcCost”-function, the “calcZO”-function, which compute the Lagrange multiplier corresponding to the full order state of every problem in the problem list, and the “prepareGradientLoop”-function, which can do pre-computations for the “calcZO”-function, to save compute time.

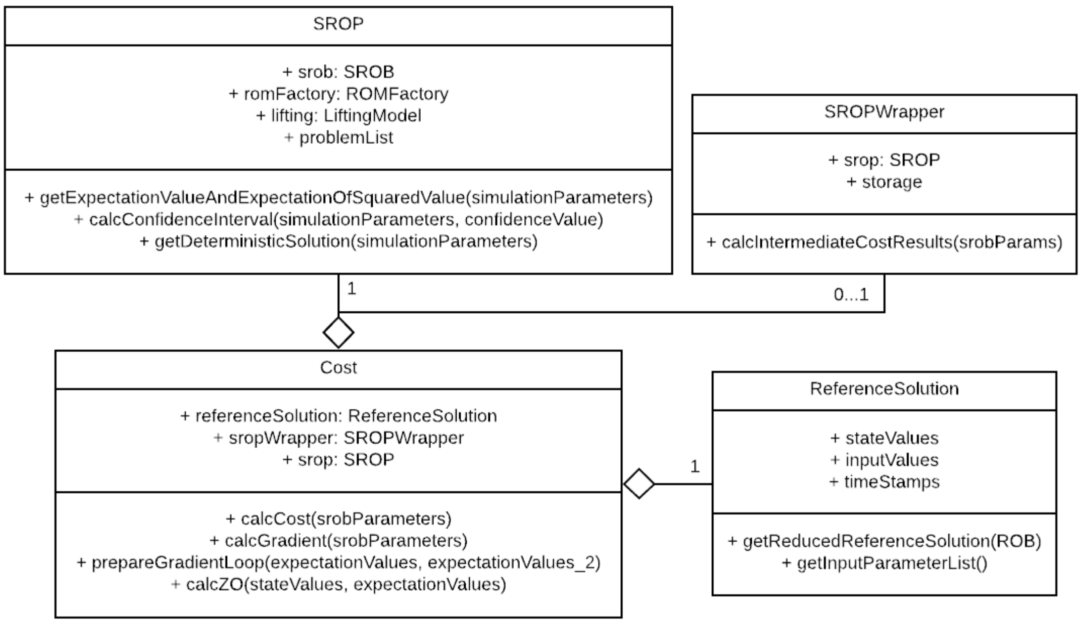

SROP
+ srob: SROB
+ romFactory: ROMFactory
+ lifting: LiftingModel
+ problemList
+ getExpectationValueAndExpectationOfSquaredValue(simulationParameters)
+ calcConfidenceInterval(simulationParameters, confidenceValue)
+ getDeterministicSolution(simulationParameters)
SROPWrapper
+ srop: SROP
+ storage
+ calcIntermediateCostResults(srobParams)
1
0...1
Cost
+ referenceSolution: ReferenceSolution
+ sropWrapper: SROPWrapper
+ srop: SROP
+ calcCost(srobParameters)
+ calcGradient(srobParameters)
+ prepareGradientLoop(expectationValues, expectationValues_2)
+ calcZO(stateValues, expectationValues)
1
ReferenceSolution
+ stateValues
+ inputValues
+ timeStamps
+ getReducedReferenceSolution(ROB)
+ getInputParameterList()

Figure 3.5.: The UML diagram shows the abstract “Cost”-class, where the functions “calcCost”, “prepareGradientLoop” and “calcZO” have to be implemented by the user. The object holds once instance of the reference solution that is used for the optimisation, an instance of a SROP which we want to optimise and a wrapper for the SROP that stores some intermediate results for the gradient computation.

The optimisation process can then be started after defining the cost function. There are two algorithms implemented, the original one from Soize and the one stage algorithm proposed in this thesis. They are implemented as functions with a cost object and initial SROB parameters as input. Furthermore, it takes a parameter object, defining the boundaries and termination criteria, as shown in Figure 3.6 . On termination, the algorithm returns a set of locally optimal parameters for the SROB. This set can then be used to simulate the system within the parameter space spanned by the given reference solution.

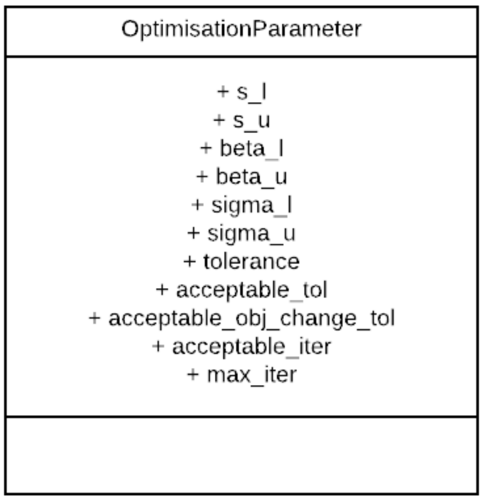


Figure 3.6.: The data class for the optimisation parameters. It includes the upper and lower bounds for s, β and $[\sigma]$. Furthermore, it holds the value for the tolerance and the maximum number of iterations. In case the algorithm is not converging, there is the option to set acceptable values for tolerance and objective function change that must be met over a number of acceptable_iter for the algorithm to terminate.

In this section we have introduced the main classes of the SROM-tool, how they are connected to each other and how they can be replaced or extended with new components. We have also seen that the structure of the SROM, including a list of problems which have to be executed, exposes a highly parallelisable structure which can be exploited in future work.

3.2. Generic Linear Static Problem

In this chapter we will look at a generic linear static problem. The problem is rather simple in terms of complexity, thus, easy to grasp and fast to compute, which makes it perfect for testing the setup and experimenting with hyperparameters. In Section 3.2.1 we define the problem. Afterwards, in Section 3.2.2, we show how to set up the optimisation algorithm and how the runtime can be enhanced by calculating the gradient via solving the adjoint problem. Additionally, we define the range of validity of the confidence intervals gained form the SROM and introduce the NLML as alternative cost function which enables a faster evaluation of the SROM in Section 3.2.5.

3.2.1. Problem Description

This problem is a linear static problem, which was taken from the paper from Soize [36]. In the following we define the FOM and the corresponding ROM.

Full Order Model

The full order model has $N = 1000$ DOF and is of the following form

$$[K]\mathbf{y} = \mathbf{f}, \tag{3.1}$$

where $\mathbf{y}$ is the displacement vector in $\mathbb{R}^N$, $[K]$ is a matrix in $\mathbb{M}_N^+$ and $\mathbf{f}$ is the force vector in $\mathbb{R}^N$. All components are dimensionless. The right hand side $\mathbf{f}$ is constructed from a set of the eigenvectors ϕ of $[K]$

$$\mathbf{f} = \frac{1}{0.27702}(0.1\phi^2 + 0.4\phi^4 + 0.6\phi^8 + 2.5(\phi^{29} + \phi^{30} + \phi^{31})). \tag{3.2}$$

The first and last value of $\mathbf{f}$ is equal to zero by construction, as shown in the detailed construction of $[K]$ in Appendix A.3. However, in our code we set those values explicitly to zero, as those values are due to numerical errors only close to zero. On the boundaries we apply Dirichlet boundary conditions ($y_1 = y_N = 0$). The complete model takes 18 ± 1 ms to evaluate. For building the SROM we additionally generate the boundary matrix $[B]$, a matrix of $\mathbb{M}_{N,N_c}$, with $N_c = 2$ which satisfies

$$[B]^\top \mathbf{y} = \mathbf{0}_{N_{CD}} \quad \text{, and } [B]^\top [B] = [I_N]. \tag{3.3}$$

The exact construction of all components is described in Appendix E of [36] and in Appendix A.3 in this thesis as well.

Reduced Order Model

For this problem we build a projection-based ROM

$$[V]^\top \mathbf{y}^{(n)} = \mathbf{q}, \tag{3.4}$$

$$[V]^\top [K][V]\mathbf{q} = [V]^\top \mathbf{f}, \tag{3.5}$$

where $[V]$ consist of the the first n eigenvectors $[\phi_1....\phi_n]$ of the matrix $[K]$ corresponding to the n biggest eigenvalues $0 < \lambda_1 < ... < \lambda_n$. From the definition of $\mathbf{f}$ in Equation [3.2] we can see that we need the first 31 modes to cover all the modes represented in $\mathbf{f}$. Equation [3.5] solves the system in the reduced order space and in Equation [3.4] the solution gets projected back to the high dimensional space. The resulting displacements $\mathbf{y}$ of Equation [3.1] and the resulting displacements $\mathbf{y}^{(n)}$ of Equation [3.4] , for the case of $n = 20$, are shown in Figure 3.7 . The model evaluation takes $47 \pm 15\mu s$, which is 380 ± 120 times faster than the FOM.

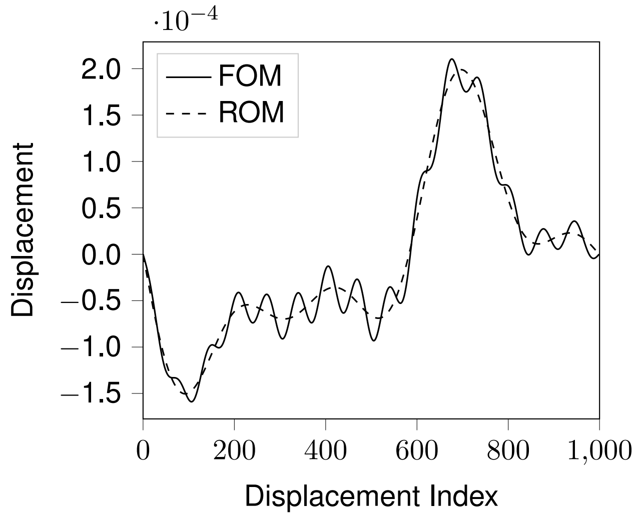


Figure 3.7.: Displacements computed from the HDM and from the ROM(n=20) of the linear static problem. It can be seen that the ROM properly describes the general trend but can not capture the smaller deflections. UQ is needed for the movements of the FOM not covered by the ROM.

3.2.2. Calibration of a SROM

In this section we state the SROM of the generic static problem and show how initial conditions and the tolerance of the interior point optimisation effect the termination time and the result.

With the just defined ROM we can build the corresponding SROM

$$\begin{aligned} \mathbf{Y}^{(n)} &= [\mathbf{W}]\mathbf{Q}, \\ [\mathbf{W}]^\top [K][\mathbf{W}]\mathbf{Q} &= [\mathbf{W}]^\top \mathbf{f}, \end{aligned} \tag{3.6}$$

subject to

$$[\mathbf{U}] = [\mathbf{G}(\beta)][\sigma], \tag{3.7}$$

$$[\mathbf{A}] = [\mathbf{U}] - [B]([B]^\top [\mathbf{U}]), \tag{3.8}$$

$$[\mathbf{D}] = \left([V]^\top [\mathbf{A}] + [\mathbf{A}]^\top [V]\right)/2, \tag{3.9}$$

$$[\mathbf{Z}] = [\mathbf{A}] - [V][\mathbf{D}], \tag{3.10}$$

$$[H_s([\mathbf{Z}])] = \left([I_n] + s^2 [\mathbf{Z}]^\top [\mathbf{Z}]\right)^{-1/2}, \tag{3.11}$$

$$[\mathbf{W}] = ([V] + s[\mathbf{Z}])\,[H_s([\mathbf{Z}])]\,. \tag{3.12}$$

To get a meaningful confidence interval from the SROM we still have to optimise its hyperparameters $\boldsymbol{\alpha}$, which is done by optimising the cost function

$$J(\boldsymbol{\alpha}) = w_J J_{\text{mean}}(\boldsymbol{\alpha}) + (1 - w_J)\, J_{\text{std}}(\boldsymbol{\alpha}), \tag{3.13}$$

where J_{mean} and J_{std} simplifies to

$$J_{\text{mean}}(\boldsymbol{\alpha}) = \frac{1}{\|\mathbf{o}^{\text{ref}}\|^2} \left\| \mathbf{o}^{\text{ref}} - E\left\{\mathbf{O}^{(n)}(\boldsymbol{\alpha})\right\}\right\|^2, \tag{3.14}$$

$$J_{\text{std}}(\boldsymbol{\alpha}) = \frac{1}{\left\|\mathbf{v}^{(\text{ref},n)}\right\|^2} \left\| \mathbf{v}^{(\text{ref},n)} - \mathbf{v}^{(n)}(\boldsymbol{\alpha}) \right\|^2, \tag{3.15}$$

with

$$\mathbf{v}^{(\text{ref},n)} = \gamma \left| \mathbf{o}^{\text{ref}} - o^{(n)} \right|, \tag{3.16}$$

and the quantity of interest $\mathbf{O}^{(n)}$, which is the displacement $\mathbf{Y}^{(n)}$.

Calibrating a SROM for a new problem demands to run several tests to find a parameter set for which the optimisation algorithm converges to a solution of sufficient accuracy. This parameter set can differs from problem to problem. The used procedure is described in more detail in this section, but for the other problems we skip this part. The following calibration was done by using the central difference gradient approximation.

Check Initial Values

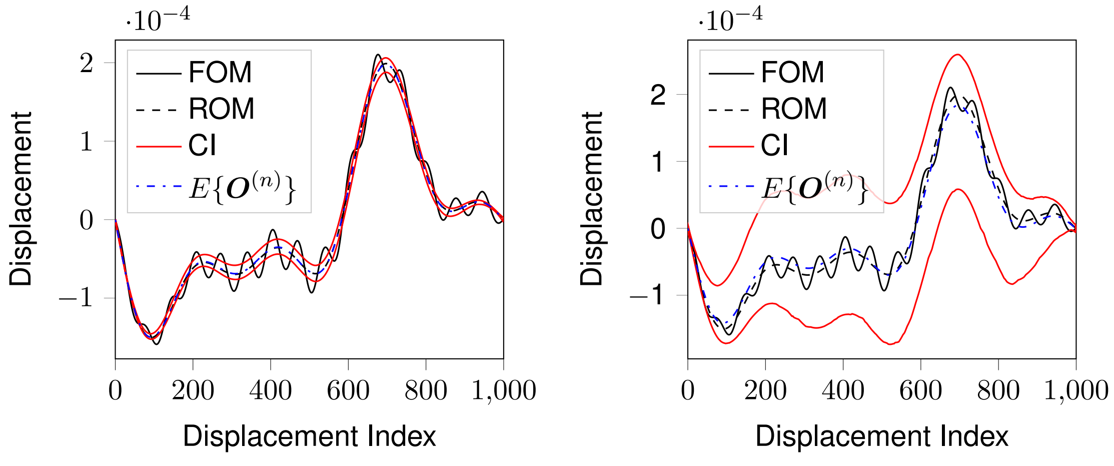
Two important things that have to be set for the optimisation are the initial conditions $\boldsymbol{\alpha}_0$ and the boundaries for the parameters we optimise for. It was proven useful to initially plot the confidence intervals and the expectation value of the QoI $E\{\mathbf{O}\}$ for a chosen $\boldsymbol{\alpha}_0$, and add the FOM and the ROM to that plot. The expectation value of the SROM has to be close to the FOM, and thus usually very close to the ROM. The confidence intervals have to be in a reasonable range, which means neither are they that big that the movement of the actual model is not recognisable anymore, as is was for the problem in Section 3.4.2 for $\boldsymbol{\alpha}_{01}$, nor are they as close to the expectation value of the SROM that they are not distinguishable anymore. If the plot does not look as expected, s can be adjusted as a first step, β and $[\sigma]$ should only be changed if the result is still not to ones satisfaction. However, one should not spend too much time on finding the perfect initial conditions, as an optimisation algorithm is almost certainly faster as soon as the initial parameters are close to a minimum. Once a proper $\boldsymbol{\alpha}_0$ is chosen, the boundaries for s can be set around the chosen initial values. The other boundaries are kept constant over all experiments. For demonstration purposes regarding compute time we are going to show two different initial conditions

$$\boldsymbol{\alpha}_{01} = \{s_{01} = 0.05, \quad \beta_{01} = 0.2, \quad [\sigma_{01}] = [I_n]\}, \tag{3.17}$$

and

$$\boldsymbol{\alpha}_{02} = \{s_{02} = 0.4, \quad \beta_{02} = 0.1, \quad [\sigma_{02}] = [I_n]\}. \tag{3.18}$$

The confidence intervals are shown in Figure 3.8 .



(a) $\boldsymbol{\alpha}_{01}$ (b) $\boldsymbol{\alpha}_{02}$

Figure 3.8.: These two plots show the confidence interval and the expectation value of the displacement of the SROM for the generic static problem with $\boldsymbol{\alpha}_{01}$ on the left and $\boldsymbol{\alpha}_{02}$ on the right. The FOM and the ROM are shown in each graph as well. Those plots show two different starting positions for the optimisation which are both valid starting position, however, result in a significantly different termination time of the optimisation.

Figure 3.8a shows a rather thin confidence interval. The expectation value of the SROM is very close to the ROM, which is positive. The confidence interval, however, covers only a small amount of the FOM. Figure 3.8b shows a confidence interval that covers some area without any FOM data points. Here we can already see that, despite the confidence interval is highly underconfident, it is still not able to cover a few points of the FOM close to the right boundary. Additionally, it can be seen that the expectation value of the SROM is not centred around the FOM anymore. Despite that, both conditions are valid initial conditions which result in differences in optimisation as shown in the course of this section.

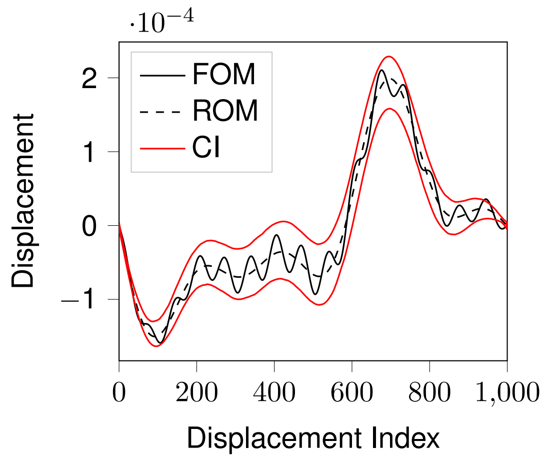
Tuning of Optimisation Algorithm

As next step, we calibrated the optimisation algorithm for this specific example. For the standard tolerance of ϵ_{tol}, which is 10^{-8}, we did not observe convergence. Therefore, we varied its value until the algorithm converged. We wanted to compare the results for values from 10^{-1} to 10^{-3} while holding all the other parameters constant. The parameters regarding the cost function were set to $\omega_j = 0.9$ and $\gamma = 0.8$. Furthermore, the boundaries for the optimisation regarding $\boldsymbol{\alpha}$ were chosen to be as in the paper from Soize and Farhat

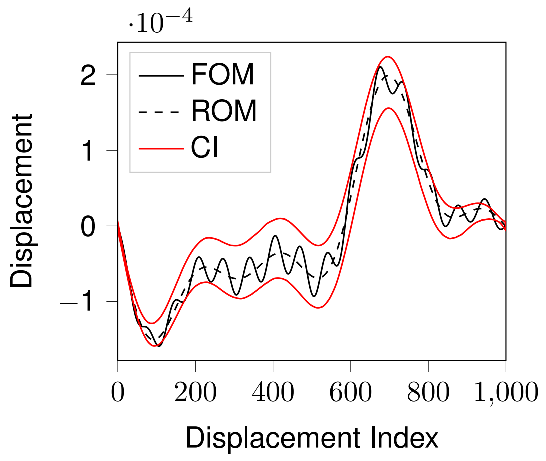
$$\begin{aligned} &\{0.01 \leq s \leq 1, \\ &\ 0.01 \leq \beta \leq 0.3, \\ &\ 0.01 \leq [\sigma]_{11}, \dots, [\sigma]_{nn} \leq 20, [\sigma]_{kk'} \in \mathbb{R}, k < k'\}. \end{aligned} \tag{3.19}$$

The off diagonal elements of $[\sigma]$ are optimised without any boundaries. We kept the limits for β and $[\sigma]$ the same for all problems. Only the boundaries regarding s are varied. The results shown in the following, were gained via the optimisation algorithm defined in Appendix A.2. The basic idea of this algorithm is to split up the optimisation to four different stages with a different number of optimisation parameters. This way, the algorithms comes close to a minimal solution by optimising only a few parameters and has to do only fine tuning with a larger set of parameters in a later stage. This algorithm is going to be referred to as the four stage algorithm (FSA) in the course of this thesis.

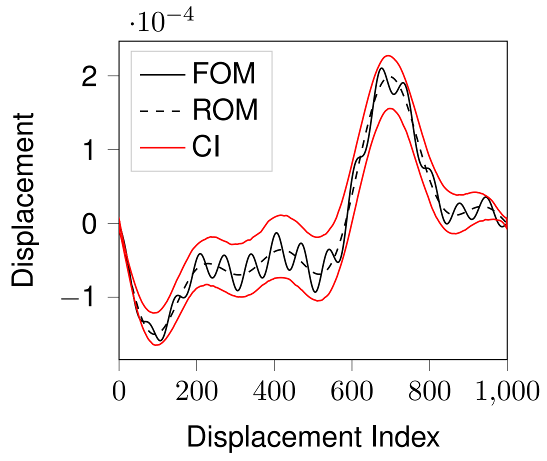
We computed the results for the two different initial values $\boldsymbol{\alpha}_{01}$ and $\boldsymbol{\alpha}_{02}$, shown in the left and right side of Figure 3.9 . The figures show 98% confidence intervals of the optimised SROMs. The intervals changed only slightly between different tolerances. Therefore, we plotted only the intervals for a tolerance of 10^{-1} and 10^{-3}. An important part to notice is that the cost functional for $\boldsymbol{\alpha}_{01}$ barely changed over different tolerances, which is shown in Table 3.1. For a large tolerance the algorithm reached a slightly less optimal minimum for $\boldsymbol{\alpha}_{02}$ at $J_{min} = 4.324 \cdot 10^{-2}$ where the algorithm reached $J_{min} = 4.190 \cdot 10^{-2}$ for $\boldsymbol{\alpha}_{01}$. Only after reducing the tolerance to $\epsilon_{tol} = 10^{-3}$ we got a similar result to the one from $\boldsymbol{\alpha}_{01}$. The slightly higher value in the cost function is for example reflected on the very right side of the interval, were we can see in comparison to $\boldsymbol{\alpha}_{01}$ that the variance covers more of the FOM, than the variance for $\boldsymbol{\alpha}_{02}$ on the right. However, one can not neglect that the termination time of the algorithm is significantly lower $\boldsymbol{\alpha}_{02}$, which might be a reason to go for $\boldsymbol{\alpha}_{02}$. However, initially we started with $\boldsymbol{\alpha}_{01}$ and conducted the experiment with $\boldsymbol{\alpha}_{02}$ rather in the end of the project. This is why we continued our experiments with $\boldsymbol{\alpha}_{01}$ and set $\epsilon_{tol} = 10^{-2}$ as a trade off between accuracy and time.



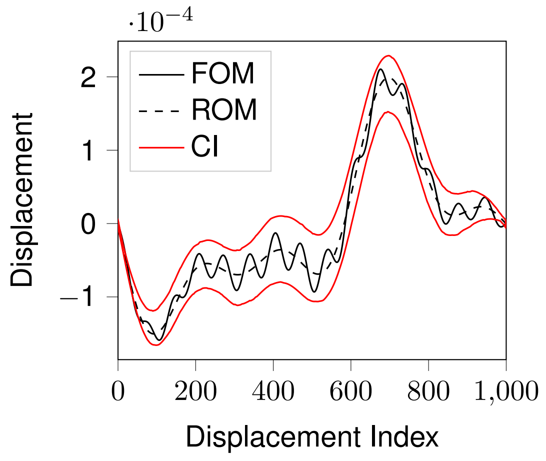
(a) $\epsilon_{tol} = 10^{-1}$, $\boldsymbol{\alpha}_{01}$



(b) $\epsilon_{tol} = 10^{-1}$, $\boldsymbol{\alpha}_{02}$



(c) $\epsilon_{tol} = 10^{-3}$, $\boldsymbol{\alpha}_{01}$



(d) $\epsilon_{tol} = 10^{-3}$, $\boldsymbol{\alpha}_{02}$

Figure 3.9.: Confidence intervals of 98% regarding the SROM for the generic static problem for different tolerances of the IPopt optimisation for initial values $\boldsymbol{\alpha}_{01}$ on the left and $\boldsymbol{\alpha}_{02}$ on the right. We can see that despite they start from different positions they both manage to achieve a similar result. However, we can see slight differences in terms of coverage close to the boundaries for different starting positions and for different tolerances.

Tolerance	Init. Value	Cost Functional	Time
10^{-1}	$\boldsymbol{\alpha}_{01}$	$4.190 \cdot 10^{-2}$	312.3 min
	$\boldsymbol{\alpha}_{02}$	$4.324 \cdot 10^{-2}$	227.2 min
10^{-2}	$\boldsymbol{\alpha}_{01}$	$4.138 \cdot 10^{-2}$	716.8 min
	$\boldsymbol{\alpha}_{02}$	$4.320 \cdot 10^{-2}$	263.54 min
10^{-3}	$\boldsymbol{\alpha}_{01}$	$4.128 \cdot 10^{-2}$	1399.5 min
	$\boldsymbol{\alpha}_{02}$	$4.133 \cdot 10^{-2}$	446.4 min

Table 3.1.: The optimised values of the nonlinear LS cost function for different tolerances and initial conditions $\boldsymbol{\alpha}_{01}$ and $\boldsymbol{\alpha}_{02}$. The time, the optimisation took to terminate is additionally shown. The time approximately doubles for every tolerance reduction.

For later reference we also state the number of evaluations of the objective function and the gradient for the optimisation $\boldsymbol{\alpha}_{01}$ in Table 3.2. Additionally, Table 3.3 shows the value of the cost function after every optimisation step. The optimisation itself has been done on a machine with 45GB DIMM RAM and ten Intel XEON processors. This machine was used for all the following optimisations as well.

	Stage	1	2	3	4
Tol.	Tuned Parameters Count m_α	22	190	1	211
10^{-1}	Cost eval.	37	1	2	79
	Gradient eval.	15	1	2	5
	Total Work Load [Cost eval.]	697	381	6	2189
10^{-2}	Cost eval.	111	1	2	316
	Gradient eval.	26	1	2	16
	Total Work Load [Cost eval.]	1255	381	6	7067
10^{-3}	Cost eval.	231	1	2	583
	Gradient eval.	48	1	2	34
	Total Work Load [Cost eval.]	2343	381	6	14931

Table 3.2.: This table shows the number evaluations of the objective function and the gradient for every stage of the FSA. Additionally, we state the total workload in terms of objective function evaluations under consideration that the gradient is calculated via a central difference scheme and needs therefore $2\,m_\alpha$ evaluations of the cost function, where m_α is the number of optimisation parameters in each stage. Due to the high number of optimisation parameters it can be seen that the workload at stage four is significantly higher than for the other stages.

Stage	1	2	3	4
Tolerance	Cost Functional			
10^{-1}	$4.19912 \cdot 10^{-2}$	$4.19912 \cdot 10^{-2}$	$4.19031 \cdot 10^{-2}$	$4.18981 \cdot 10^{-2}$
10^{-2}	$4.13840 \cdot 10^{-2}$	$4.13840 \cdot 10^{-2}$	$4.13838 \cdot 10^{-2}$	$4.13836 \cdot 10^{-2}$
10^{-3}	$4.12799 \cdot 10^{-2}$	$4.12799 \cdot 10^{-2}$	$4.12799 \cdot 10^{-2}$	$4.12794 \cdot 10^{-2}$

Table 3.3.: The values of the objective function after every stage of the FSA with the central difference gradient approximation. The table shows that the first stage got already very close to the end result and the other three stages did only some fine tuning.

It can be seen that the time approximately doubled for every digit of accuracy. For a tolerance of 10^{-3} it already took approximately one day until the optimisation algorithm reached the required tolerance. Furthermore, we can see based on the number of objective function evaluations that most of the work was done in the first and fourth stage. This is due to the small impact of the off-diagonal elements of $[\sigma]$ in this problem, which let the algorithm in stage two directly terminate. Even if the interior point optimisation directly terminates, there is one evaluation of the cost function and the gradient as they are needed to initially check the tolerance. In stage three the optimisation is rather simple as the algorithm has to optimise for only one variable. Hence, it terminates quite fast. The algorithm spent a lot of time in stage four, the resulting reduction of the objective function however was only very little. Considering the long computation time and the waste of time in stage two to four, we want to introduce the adjoint method, which makes it possible to calculate the gradient with constant computational cost, thus independent of m_α.

3.2.3. Adjoint Problem

In this subsection we are going to introduce the adjoint problem to the SROM for the generic static model and compare it to the central finite difference scheme. This adjoint problem is then extended in the course of this thesis.

General SROB Adjoint Problem Definition

As it was already described in Section 2.3.2, the adjoint problem is set up by defining the Lagrangian to the problem defined in Equations [3.6] to [3.13]. For the regarding example this results in

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{\alpha}, \underbrace{[\mathbf{W}]_i, [\mathbf{H}_s]_i, [\mathbf{Z}]_i, [\mathbf{D}]_i, [\mathbf{A}]_i, [\mathbf{U}]_i, \mathbf{q}_i, \mathbf{O}_i}_{\mathcal{U}}, \\
&\underbrace{[\mathbf{ZW}]_i, [\mathbf{ZH}]_i, [\mathbf{ZZ}]_i, [\mathbf{ZD}]_i, [\mathbf{ZA}]_i, [\mathbf{ZU}]_i, \mathbf{Zq}_i, \mathbf{ZO}_i}_{\mathcal{Z}}) = \\
&J(\mathbf{O}) \\
&-[\mathbf{ZU}]_i \cdot \Big([\mathbf{U}]_i - [\mathbf{G}(\beta)]_i[\sigma]\Big) \\
&-[\mathbf{ZA}]_i \cdot \Big([\mathbf{A}]_i - [\mathbf{U}]_i + [B]\left\{[B]^T[\mathbf{U}]_i\right\}\Big) \\
&-[\mathbf{ZD}]_i \cdot \Big([\mathbf{D}]_i - \left([V]^T[\mathbf{A}]_i + [\mathbf{A}]_i^T[V]\right)/2\Big) \\
&-[\mathbf{ZZ}]_i \cdot \Big([\mathbf{Z}]_i - [\mathbf{A}]_i + [V][\mathbf{D}]_i\Big) \\
&-[\mathbf{ZH}]_i \cdot \Big(\left([I_n] + s^2[\mathbf{Z}]_i^T[\mathbf{Z}]_i\right)[\mathbf{H}_s]_i[\mathbf{H}_s]_i - [I_n]\Big) \\
&-[\mathbf{ZW}]_i \cdot \Big([\mathbf{W}]_i - ([V] + s[\mathbf{Z}]_i)[\mathbf{H}_s]_i\Big) \\
&-\mathbf{Zq}_i^\top \Big(\mathbf{q}_i - \boldsymbol{f}_{\text{model}}\Big) \\
&-\mathbf{ZO}_i^\top \Big(\mathbf{O}_i - [\mathbf{W}]_i\mathbf{q}_i\Big),
\end{aligned} \tag{3.20}$$

where "$\cdot$" is the Frobenius inner product, $\mathcal{Z}$ are the Lagrangian multipliers, $\mathcal{U}$ are the helper variables that are defined to introduce the necessary equality constrains and J is the cost function. This formulation can not be used to calculate the derivative regarding β, as $[\mathbf{G}(\beta)]$ is introduced as given. This is intended and will be further elaborated in the coarse of this section. With Equation [3.20] we can now calculate the derivatives regarding the SROB hyperparameters $\boldsymbol{\alpha}$. Under the assumption that $\mathcal{U}$ is a solution of the SROM which corresponds to

$$\mathcal{L}'_{\mathcal{U}}(\boldsymbol{\alpha}, \mathcal{U}, \mathcal{Z})(\delta\mathcal{U}) = 0 \quad \forall \delta U \tag{3.21}$$

and $\mathcal{Z}$ is a solution of the adjoint problem

$$\mathcal{L}'_{\mathcal{Z}}(\boldsymbol{\alpha}, \mathcal{U}, \mathcal{Z})(\delta\mathcal{Z}) = 0 \quad \forall \delta Z, \tag{3.22}$$

we can compute the gradient of the cost function J via

$$[J'(\boldsymbol{\alpha})](\delta\boldsymbol{\alpha}) = \mathcal{L}'_{\boldsymbol{\alpha}}(\boldsymbol{\alpha}, \mathcal{U}, \mathcal{Z})(\delta\boldsymbol{\alpha}). \tag{3.23}$$

The derivatives regarding s and $[\sigma]$ are then

$$[J'_\sigma](\boldsymbol{\alpha})(\delta\sigma) = \sum_i^{\nu_s} -[\mathbf{ZU}]_i \cdot \Big(-[\mathbf{G}(\beta)]_i[\delta\sigma]\Big) = [\delta\sigma] \cdot \sum_i^{\nu_s} [\mathbf{G}(\beta)]_i^\top [\mathbf{ZU}]_i, \tag{3.24}$$

$$[J'_s](\boldsymbol{\alpha})(\delta s) = \Big(\sum_i^{\nu_s} [\mathbf{ZW}]_i \cdot \Big([\mathbf{Z}]_i[\mathbf{H}_s]_i\Big) - 2s[\mathbf{ZH}]_i \cdot \Big([\mathbf{Z}]_i^T[\mathbf{Z}]_i[\mathbf{H}_s]_i[\mathbf{H}_s]_i\Big)\Big)\delta s, \tag{3.25}$$

where we need the Lagrangian multipliers $[\mathbf{ZU}]$, $[\mathbf{ZW}]$ and $[\mathbf{ZH}]$. A step by step calculation for this specific case is shown in Appendix A.1.

Memory Considerations

To calculate the missing Lagrangian multipliers, we have to store the intermediate steps from the calculation of the cost function. In case we save all intermediate steps for the generic static model with double precision, we need to store

$$\left(4Nn + 2n^2 + (n+N)m_\mu k\right) \nu_s \cdot 8 \text{ Byte}, \tag{3.26}$$

five matrices of $\mathbb{M}_{N,n}$, two matrices of $\mathbb{M}_n$ plus the reduced order solution of size n and the full order solution of size N for all samples of the parameter space m_μ and all time steps k. All of this has to be stored for all ν_s realisations of the SROB. For the generic static problem, where $k = 1$, $m_\mu = 1$, $n = 20$ and $N = 1000$, this gives 654.56 MB. However, we do not have to save the high dimensional solution for every realisation, as we have to store the reduced order basis anyway and can therefore just use it to get the high dimensional solution from the reduced order solution. This does not seem that much of an improvement for the static generic problem as the intermediate steps of the SROB take way more memory, but for later examples it will have a great impact once we look at simulations for different parameters and over time. The memory usage for this problem is then

$$(4Nn + 2n^2 + n \cdot m_\mu k)\nu_s \cdot 8 \text{ Byte}, \tag{3.27}$$

which is 646.56 MB for the generic static problem with $n = 20$. This memory usage does not make any problem for modern computers and it will not grow to an extend that it can not be dealt with anymore once we look at dynamic problems or at problems over a parameter space. Furthermore, a Valgrind profiling for the optimisation algorithm over the static generic example has shown that 63.7% of the time was spent sampling the SROB, which is an other reason for storing the intermediate results and not compute it again. If we wanted to set up the adjoint problem for the gradient regarding β we have to store all the realisations of $\mathcal{Q}$ and the intermediate results during the generation of $[\mathbf{G}(\beta)]_i$, which results in a memory usage of around 4.5 GB for the static problem, which is significantly more than without the gradient for β. Furthermore, the storage requirements grow linearly with N and even if the usage for the linear static example can be handled by most of the machines nowadays, for the sake of this thesis we left it out to keep the memory usage for later problems with N up to 7413 in a range it could be handled by a laptop with 8 GB RAM. For this reason we computed the entry in the gradient $\nabla_\alpha J$ for β via a finite difference scheme.

Model Contribution

The model contribution to the adjoint problem is calculated in the following way: At first, we have to write down its contribution to the Lagrangian function of the overall cost function

$$-\mathbf{Zq}^\top\Big([\mathbf{W}]^\top[K][\mathbf{W}]\mathbf{q}-[\mathbf{W}]^\top\mathbf{f}\Big), \tag{3.28}$$

where $\mathbf{Zq}$ is the vector shaped adjoint variable in $\mathbb{R}^n$ and $[\mathbf{W}]$ is a basis in $\mathbb{M}_{N,n}$, sampled from a SROB, used for projecting the HDM to the reduced order space. Then we have to take the derivative regarding $[\mathbf{W}]$

$$-\mathbf{Zq}^\top\Big([\mathbf{W}]^\top[K][\delta\mathbf{W}]\mathbf{q}+[\delta\mathbf{W}]^\top[K][\mathbf{W}]\mathbf{q}-[\delta\mathbf{W}]^\top\mathbf{f}\Big), \tag{3.29}$$

and after moving $[\delta\mathbf{W}]$ out of the brackets we get

$$[\delta\mathbf{W}]\cdot\Big(-[K]^\top[\mathbf{W}]\mathbf{Zq}\mathbf{q}^\top-[K][\mathbf{W}]\mathbf{q}\mathbf{Zq}^\top+\mathbf{f}\mathbf{Zq}^\top\Big). \tag{3.30}$$

The resulting term in brackets is the contribution to the calculation of $[\mathbf{ZW}]$. However, to be able to compute that term, we are still missing the adjoint variable $\mathbf{Zq}$. Hence, we have to take the partial derivative of the whole Lagrangian regarding the reduced state values q and set it to zero

$$\mathbf{Zq}^\top\Big([\mathbf{W}]^\top[K][\mathbf{W}]\delta\mathbf{q}\Big)+\mathbf{ZO}^\top\Big(-[\mathbf{W}]\delta\mathbf{q}\Big)=0\quad\forall\delta\mathbf{q}. \tag{3.31}$$

This equation has to be true for any q. Moving $\delta\mathbf{q}$ out of the brackets and solving for $\mathbf{Zq}$ gives

$$[\mathbf{W}]^\top[K]^\top[\mathbf{W}]\mathbf{Zq}=[\mathbf{W}]^\top\mathbf{ZO}, \tag{3.32}$$

where $\mathbf{ZO}$ is the adjoint variable of the high dimensional state. For the calculation of the model contribution $\mathbf{ZO}$ can be seen as given, as we get it out of the partial derivative of the cost function regarding $\mathbf{O}$, as shown in Appendix A.1. Therefore, we computed all Lagrangian multipliers to compute the contribution to $[\mathbf{ZW}]$. This procedure can be applied to every new problem. In case, the structure is more complex, it can be helpful to write down the whole Lagrangian and redo the calculations step by step.

Correctness

To get an indication for correctness of the analytical gradient we compared it to the central difference approximation and check for second order convergence. The error of the derivative $f'(\alpha)$ of a function $f(\alpha)$ calculated via central difference scheme is

$$\left|\frac{f(\alpha+\varepsilon)-f(\alpha-\varepsilon)}{2\varepsilon}-f'(\alpha)\right|=c\varepsilon^2, \tag{3.33}$$

where $\varepsilon\in\mathbb{R}^+$ and c is a constant. The exponent of ε on the right side of the equation defines the order of the scheme. We compute this exponent and check if we see a convergence rate of two. This is done by subtracting the analytical gradient from the finite difference gradient once with ε and once with 0.1ε and divide one by the other

$$\frac{|\frac{f(\alpha+\varepsilon)-f(\alpha-\varepsilon)}{2\varepsilon} - f'(\alpha)|}{|\frac{f(\alpha+0.1\varepsilon)-f(\alpha-0.1\varepsilon)}{2\cdot 0.1\varepsilon} - f'(\alpha)|} = \frac{\varepsilon^a}{(0.1\varepsilon)^a}, \tag{3.34}$$

where a is the order of the FD scheme that we are interested in. By taking the log of both sides and rearranging the equation, we get an expression for a

$$a = \frac{1}{\log(10)} \cdot \log\left(\frac{|\frac{f(\alpha+\varepsilon)-f(\alpha-\varepsilon)}{2\varepsilon} - f'(\alpha)|}{|\frac{f(\alpha+0.1\varepsilon)-f(\alpha-0.1\varepsilon)}{2\cdot 0.1\varepsilon} - f'(\alpha)|}\right). \tag{3.35}$$

We expect the order of the scheme to be around two for high ϵ. With decreasing ϵ the truncation error decreases, whereas the round off error increases. At some point, when the round off error grows faster then the truncation error decreases, the gradient approximation turns worse and the order drops below zero, as can be derived from Section 2.3.2.

For the generic static problem and $\boldsymbol{\alpha}_{01}$, defined in Equation [3.17] , the graph is shown in Figure 3.10 , which shows exactly what we expected. On the right, we can see the order of two and once the round off error gets too big with smaller epsilon, the order drops to minus one. This is a strong indicator for a correct analytical gradient.

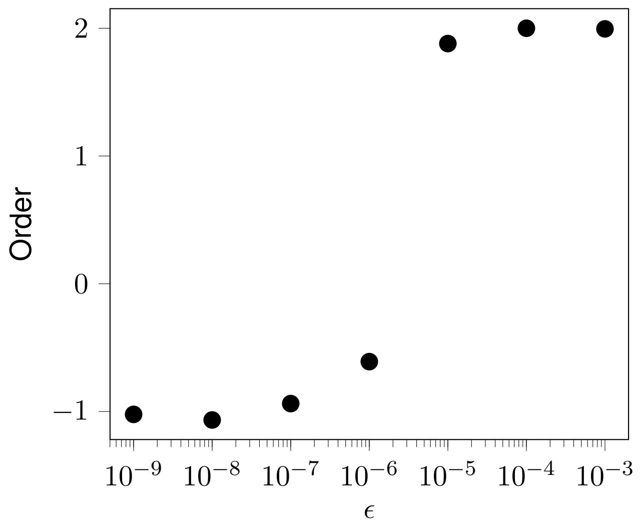


Figure 3.10.: Order of the central difference scheme for the derivative of the cost function regarding the SROM hyperparameters $\boldsymbol{\alpha}$ for the generic static problem. The approximation has been taken at $\boldsymbol{\alpha}_{01}$. The figure clearly shows a second order convergence for high ϵ and falls down to -1 when the round off error increases faster than the truncation error decreases.

When the order is around zero, the argument of the $\log$ in Equation [3.35] is one. This means that the error between FD approximation and analytical gradient is not decreasing anymore, but staying the same and decreasing ϵ any further would make it worse. Hence, the optimal epsilon is where the order is zero, which is around 10^{-5}. The approximated optimal epsilons used in Section 3.2.2 have been in between 10^{-5} to 10^{-6}, which confirms our parameter choice.

Comparison to Finite Difference Scheme

The speedup achieved by computing the analytical gradient depends on the optimisation problem, as the work load for the adjoint problem is always the same but the workload for the FD gradient grows with the number of variables. The analytical gradient has to compute one forward pass to gather all the matrices for the adjoint problem, plus the backward pass, which is usually a little less expensive than the forward pass. This gives the workload of approximately two cost function evaluations. Additionally, we have to compute the FD gradient for β in case it is needed, which are another two evaluations. In the central finite difference approximation the cost function is evaluated twice for every variable of the optimisation step. This gives a workload reduction of approximately a factor of

$$\text{load reduction} \approx \begin{cases} m_\alpha & \text{if } \nabla_\beta J \text{ is not evaluated} \\ 1 & \text{if only } \nabla_\beta J \text{ is evaluated} \\ \frac{m_\alpha}{2} & \text{if } \nabla_\beta J \text{ is evaluated} \end{cases}, \tag{3.36}$$

where m_α is the number of variables in the optimisation step.

To compute the real speedup we measured the gradient evaluation time in each stage of the FSA, shown in Table 3.4. The measurements were taken on a machine with 16 GB DDR4 RAM and an Intel Core i7-7500U CPU. The error was calculated out of five samples and the 95% confidence interval of the t-distribution.

Stage	1	2	3	4
Tuned Parameters Count m_α	22	190	1	211
Relative Speedup	$\times(11.3 \pm 0.6)$	$\times(203.6 \pm 6.4)$	$\times 1$	$\times(228.0 \pm 5.0)$

Table 3.4.: Relative speedup of the cost function gradient evaluation regarding the SROM for the generic static problem. The relative speedup was measured for the four stages of the FSA individually. The speedup factors where computed out of five samples and the 95% confidence interval of the t-distribution.

We can see that the estimated load reduction factor, which is 11, 190 and 211 for stage one, two and four, fits quite well to the measured speedups. In stage three, we optimise only for β. Hence, there is no change in compute time or workload.

As a next step, we look at the overall speedup over the optimisation algorithm, which is shown in Table 3.5 with the change of the final cost function value. We can see that replacing the FD gradient with an analytical gradient had barely an effect on the result. However, it got up to 42 times faster than before.

Tol.	Cost Functional	Rel. Change in Cost Functional Compared to FSA with FD Grad.	Rel. Speedup Compared to FSA with FD Grad.
10^{-1}	$4.1898 \cdot 10^{-2}$	$+0\%$	x35
10^{-2}	$4.1385 \cdot 10^{-2}$	$+0.002\%$	x41
10^{-3}	$4.1271 \cdot 10^{-2}$	-0.02%	x42

Table 3.5.: Comparison of the final value of the cost function and the execution time between the FSA with and without an analytical gradient. The shows a maximum relative speedup factor of 42 for a tolerance of 10^{-3} with a barely changing value of the cost functional.

Tol.	Stage	1	2	3	4
10^{-1}	Cost eval.	37 (+0)	1 (+0)	2 (+0)	14 (-65)
	Gradient eval.	15 (+0)	1 (+0)	2 (+0)	3 (-2)
10^{-2}	Cost eval.	111 (+0)	1 (+0)	2 (+0)	15 (-301)
	Gradient eval.	26 (+0)	1 (+0)	2 (+0)	3 (-13)
10^{-3}	Cost eval.	231 (+0)	1 (+0)	2 (+0)	15 (-586)
	Gradient eval.	48 (+0)	1 (+0)	2 (+0)	3 (-31)

Table 3.6.: The evaluations of the objective function and the analytical gradient for every stage of the FSA and the overall time. It shows a significant reduction in objective function evaluations and gradient evaluation in stage four.

Table 3.6 shows that switching to the analytical gradient caused only a slight change in the number of cost function and gradient evaluations, which demonstrates that the FD gradient approximation for s, β and the diagonal elements of $[\sigma]$ was accurate enough for the chosen tolerance. Only the fourth stage has significantly less evaluations of objective function and gradient. In comparison to Table 3.2, there were 583 objective function evaluations with the FD gradient but only 15 with the analytical one. This indicates a more accurate search direction which comes from an advanced gradient for the off diagonal elements of $[\sigma]$.

This section has shown that replacing the central difference gradient approximation with an analytical gradient leads to different speedup factors for the four stages in the FSA, dependent on the number of optimisation parameters. The analytical gradient is 228 times faster than the FD gradient in the fourth stage which previously took the longest to execute. On the whole algorithm we got speedups up to a factor of 42.

3.2.4. Alternative Optimisation Algorithm

In this subsection we would like to introduce an alternative optimisation algorithm. The FSA was chosen under the consideration of the increasing cost of the FD gradient approximation scheme with an increasing number of variables. That way, the algorithm can get close to a minimum with only a few variables and do fine adjustments with almost all of the variables afterwards. However, as the evaluation cost of the gradient computed via solving the adjoint problem is not dependent on the number of variables anymore, we can also directly optimise for all of the parameters. This saves the cost of three initialisations of the IPopt algorithm. From now on we will refer to the alternative algorithm as the one stage algorithm (OSA). For comparison, we deducted the same experiment over different tolerances again with the initial values $\boldsymbol{\alpha}_{01}$ defined in Equation [3.17] and the boundaries defined in Equation [3.19] . The results are shown in Table 3.7.

Tol.	Cost evals.	Gradient evals.	Cost Functional	Time	Rel. Speedup Compared to FSA with Anal. Gradient
10^{-1}	14	9	$4.252 \cdot 10^{-2}$	3.9 min	$\times 2.3$
10^{-2}	735	96	$4.063 \cdot 10^{-2}$	82.0 min	$\times 0.2$
	87	22	$4.142 \cdot 10^{-2}$	12.0 min	$\times 1.4$
10^{-3}	2399	278	$4.052 \cdot 10^{-2}$	257.6 min	$\times 0.13$

Table 3.7.: This table shows the number of cost function and gradient evaluation with the achieved cost functional for the OSA. The overall time for different tolerances is stated additionally. In the last column we compare the runtime to the FSA with the analytical gradient. The table shows that the OSA reaches a similar value for the cost functional in a shorter time.

Optimising over all of the parameters at once resulted in an overall relative speedup compared to the FSA with the central difference gradient of approximately 80 for a tolerance of 0.1. This is 2.3 times faster than the FSA with the analytical gradient from Section 3.2.3. However, the OSA also achieved a less optimal solution. The OSA achieved a minimal cost of $4.190 \cdot 10^{-2}$. Once we go down to smaller tolerances we can see that in the first lines of 10^{-2} and 10^{-3} in Table 3.7 take in comparison five to ten times longer, but reach a lower minimum. In the second run of the tolerance 10^{-2} we set the acceptable tolerance to 10^{-1} and the number of acceptable iteration to ten, which results in a faster termination. Consequently, the OSA is capable of reaching a similar solution to the FSA in a shorter amount of time. Therefore, the OSA has been used in the course of this thesis.

3.2.5. Confidence Intervals

In the last section of this section, we want to asses the quality and the evaluation time of the achieved confidence intervals and introduce the NLML as an alternative cost function for improving the SROM compute time and the coverage of the FOM close to the boundaries.

Soize LS Cost Function

For this evaluation we used the $\boldsymbol{\alpha}_{opt}$ gained from the optimisation with $\boldsymbol{\alpha}_{01}$ as initial value, a tolerance of 10^{-3} and the analytical gradient. The confidence interval is calculated by sorting the values of the SROM for every node and cutting off (100% - p_c)/2 from the the top and the bottom, where p_c is the percentage of the confidence. The overall evaluation of the SROM, which includes $\nu_s = 1000$ ROM evaluations and sorting of the values for every node, takes 115 ± 2 ms which is approximately 10 times longer than the evaluation of the FOM. Figure 3.11 shows the percentage of the covered FOM points over the percentage of the confidence interval. We can clearly see that the interval gained from the SROM is overconfident most of the times, except for a small range from 90% to 95%. Furthermore, it is not able to cover all of the data points from the FOM with its 100% interval and this behaviour does not change with increasing γ in Equation [3.16] . As we are looking for confidence intervals that are valid in the range above 99.7% this is not acceptable and we have to look for an alternative cost function.

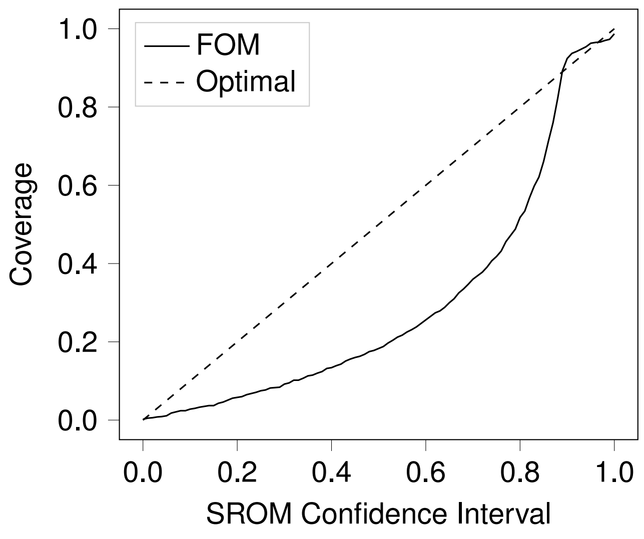


Figure 3.11.: This figures shows the coverage of the FOM over the percentage of the confidence intervals of the SROM optimised over the nonlinear LS cost function for the generic static problem. It can be seen that the model is overconfident most of the time, but is valid in a small range between 90%-95%.

Negative Log Marginal Likelihood

As an alternative cost function, we propose the Negative Log Marginal Likelihood (NLML), which is commonly used to fit a Gaussian Process (GP) [22]. In this case we minimise the NLML over the error of the expectation of the SROM, where we assume that the values of the SROM are Gaussian distributed. The NLML is defined as the negative of the logarithm of the probability of an N-dimensional normal distribution with covariance $[K(\boldsymbol{\alpha})]$,

$$\mathcal{NLML}(\boldsymbol{\alpha}) = \frac{1}{2}\mathbf{y}(\boldsymbol{\alpha})^T[K(\boldsymbol{\alpha})]^{-1}\mathbf{y}(\boldsymbol{\alpha}) + \frac{1}{2}\log|[K(\boldsymbol{\alpha})]| + \frac{N}{2}\log(2\pi), \tag{3.37}$$

where we choose $\mathbf{y}$ to be the difference of the FOM to the expectation of the SROM

$$\mathbf{y}(\boldsymbol{\alpha}) = \mathbf{o}^{\text{ref}} - E\left\{\mathbf{O}^{(n)}(\boldsymbol{\alpha})\right\}, \tag{3.38}$$

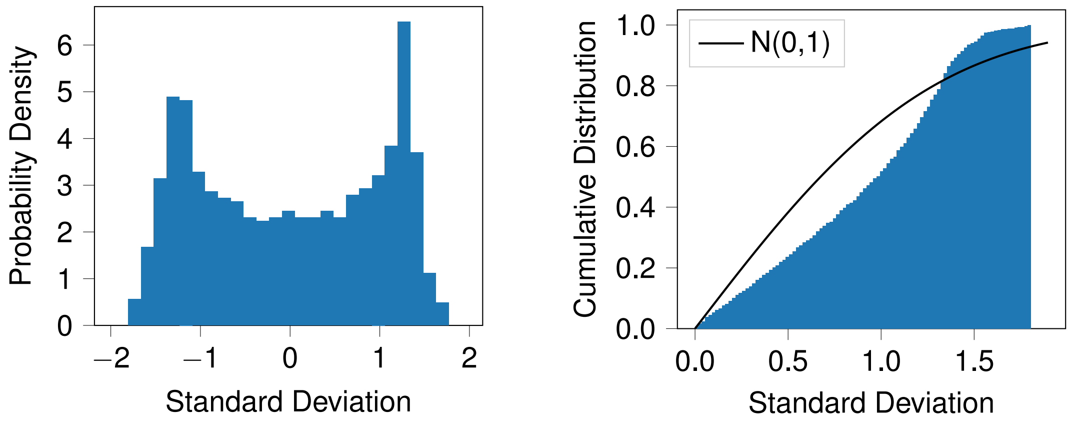
and the kernel $[K(\boldsymbol{\alpha})]$ to be a diagonal matrix of $\mathbb{M}_N$ with the variance of the SROM as diagonal elements

$$[K(\boldsymbol{\alpha})] = \sum_{ii}\left(E\left\{O^{(n)}(\boldsymbol{\alpha})^2\right\} - \left(E\left\{O^{(n)}(\boldsymbol{\alpha})\right\}\right)^2\right)_i, \tag{3.39}$$

N is the total number of points in one simulation. This models a GP with the expectation of the SROM as the mean value and noise of the size of the standard deviation of the SROM. This approach is valid if most points of the FOM are within a close range in terms of standard deviations to the expectation value of the SROM after optimising over the NLML. Otherwise, the GP is overly confident, which can be costly in certain applications. In general, we lose information from the assumption that the distributions are Gaussian, which is not necessarily true. In case the optimisation algorithm tries to fit highly skewed distributions with a Gaussian, it potentially results into cutting off the tail on one side and including space that would not be covered by the SROM on the other side. However, the NLML has some significant advantage over the nonlinear LS cost function. First of all, it does not have any parameters that we have to optimise manually, whereas we have to choose ω_j and γ for the nonlinear LS cost function. Second, we do not have to sort the values from the SROM evaluation to plot the confidence interval, but can just use the standard deviation. This replaces the sorting of $O(N\log(N))$ with the computation of the standard deviation of $O(N)$. Another advantage is that the points in the NLML are weighted by the inverse of the variance, which results in a higher weight on areas where the ROM is close to the FOM, and therefore results in a better coverage of the FOM close to the boundaries.

In the following, we show the result for the SROM for the genetic static model optimised over the NLML cost function. As optimisation algorithm we used the OSA with an analytical gradient. The derivative of the NLML for the adjoint problem can be found in Section A.1.2. The tolerance was set to 10^{-2} with an acceptable tolerance of 10^{-1}, a number of acceptable iterations of five and the same initial values $\boldsymbol{\alpha}_{01}$ and boundaries as in Section

3.2.2. The optimisation reached the tolerance after 1632 cost function evaluations and 204 gradient evaluations. The cost is $-9.813 \cdot 10^3$ and it took 183.3 minutes, which is again significantly longer then the 12 minutes it took for the nonlinear LS cost optimisation in Section 3.2.4. However, the gained confidence intervals finally reached a coverage of above 99.7% of the FOM data points.

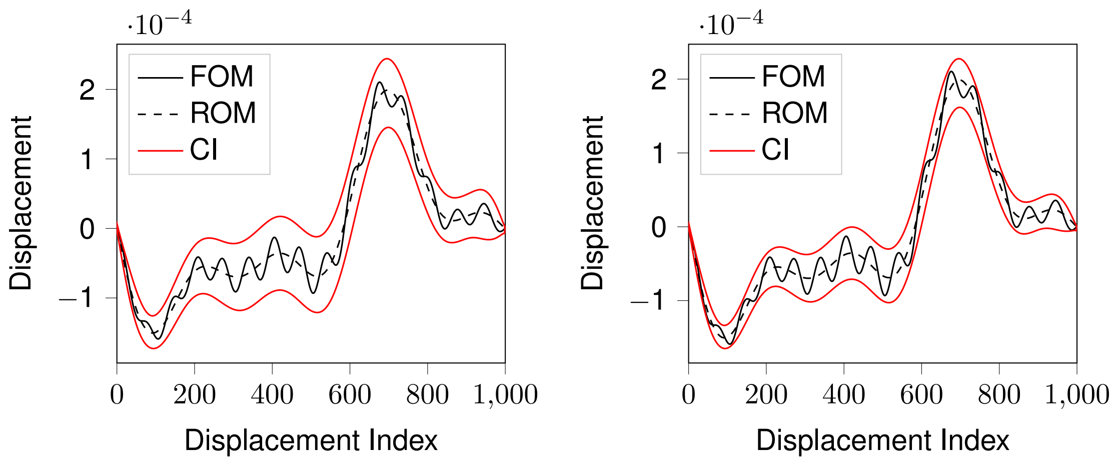


(a) Probability Density (b) Cumulative Distribution

Figure 3.12.: The probability distribution and the cumulative distribution of the FOM of the generic static model around the expectation value of the SROM standardised by the standard deviation is shown as blue bars. The cumulative distribution function of a normal distribution is plotted as solid black line to asses the valid range of the confidence interval. We can see that the FOM was placed symmetrical around the expectation value of the SROM and that a little less than two standard deviations cover all of the FOM data points.

In the left plot in Figure 3.12 we can see the probability distribution of the FOM data points over the distribution of the SROM, standardised by the standard deviation of every DOF. On the right, the cumulative distribution with the origin in the centre of each Gaussian is shown. In both figures it becomes clear that the optimisation managed to place all of the FOM data points in a close range in comparison the standard deviation to the expectation value of the SROM. However, the FOM data points are not Gaussian distributed, which leads to an overestimation of the covered data points in the range up to 1.3 standard deviations and therefore an overly confident confidence interval. When the cumulative distribution of the FOM over the SROM crosses the cumulative distribution function (CDF) of the standard normal distribution, we cover as much of the data as expected and more than that, which makes the confidence interval valid above 1.3 standard deviations.

In Figure 3.13 we plotted the confidence interval over three standard deviations, which corresponds to 99.7% coverage, on the left and over two standard deviations, which corresponds to 95.4%, on the right. Both of the intervals cover all of the data points, as can already be seen from Figure 3.12 , and originates from weighting the data points by its inverse standard deviation, which puts more value on the points close to the boundaries. However, for this specific example we get underconfident confidence intervals which is also not the desired outcome. The evaluation of the SROM takes without sorting 72.3 ± 6 ms, which is a speedup of approximately 1.59 ± 0.03 to the SROM evaluation via sorting. This is still slower than directly evaluating the FOM. However, this changes for more complex models.



(a) Three standard deviations interval

(b) Two standard deviations interval

Figure 3.13.: The two figures show the 99.7% one the left and the 95.4% confidence interval of SROM for the generic static example optimised over the NLML, on the right. Both intervals cover 100% which shows that the model is slightly underconfident for the given data.

Optimising the SROM for the generic static problem over the NLML has proven to result in a confidence interval which is valid for a confidence of 99.7% and above. Furthermore, we have seen that evaluating the confidence interval, under the assumption that the distribution of the points from the SROM is Gaussian, results in a 1.59 faster compute time.

3.3. PyMOR Thermal Block

PyMOR is a python package for building model order reduction software [25]. It provides a large variety of different MOR methods, which result in a ROM with uncertainty in comparison to the FOM. In this section, the application of pyMOR models in the setting of this thesis will be described. We use the thermal block problem, which is already implemented in pyMOR. This problem is also a good opportunity to show the difference between a known and an unknown grid, as the grid is not available for the problems in Section 3.4. In addition, we use this problem to introduce a parameter space the SROM is then optimised for.

The thermal block problem solves the stationary heat equation of the form

$$-\nabla[d(\boldsymbol{x},\boldsymbol{\mu})\nabla u(\boldsymbol{x},\boldsymbol{\mu})] = 1, \quad \text{for } \boldsymbol{x} \in \Omega := [0,1]^2, \tag{3.40}$$

$$u(\boldsymbol{x},\boldsymbol{\mu}) = 0, \quad \text{for } \boldsymbol{x} \in \delta\Omega. \tag{3.41}$$

where u is the heat in Joule [J]. Furthermore, the domain is split into blocks of different thermal conductivity $d(\boldsymbol{x},\boldsymbol{\mu})$. In this thesis we use a grid of 2x2 equally sized blocks. The value for the thermal conductivity μ of each block is in the interval of $[0.1, 1]\ \frac{\mathrm{W}}{\mathrm{m\,K}}$, which gives us a four dimensional parameter space of $[0.1, 1]^4\ \frac{\mathrm{W}}{\mathrm{m\,K}}$. In summary, we are simulating four blocks with different material properties that get uniformly heated while the heat at the boundaries is kept constant at zero.

3.3.1. Full Order Model

After discretising the domain, we have a finite element triangular discretisation with $m_0 = 5101$ nodes and $N = m_0$, as heat is the only value we are measuring on each node. The pyMOR discretisation method returns two objects, the FOM and an additional object that contains the point coordinates from the grid and information regarding the boundaries, from which the boundary matrix can be extracted from. The coordinates and boundary matrix can be passed on to the SROB to reduce the possible function space of the SROM. The extraction is explained in detail in Appendix A.4. The FOM after discretisation has the form of

$$\left(\sum_{i}^{2}\sum_{j}^{2} a(\boldsymbol{\mu})L_{ij}(\boldsymbol{\mu})\right)\boldsymbol{u} = c\,\mathbf{f}, \tag{3.42}$$

where each matrix $L_{ij}(\boldsymbol{\mu}) \in \mathbb{M}_N$ corresponds to one of the four blocks from the domain, a and c are coefficients, $\mathbf{f}$ is the right hand side vector in $\mathbb{R}^N$ and $\boldsymbol{u}$ is the heat vector in $\mathbb{R}^N$. The FOM takes 12 ± 0.08 ms to evaluate.

3.3.2. Reduced Order Model

Here we have, very similar to the linear static problem, a projection-based ROM of the form

$$\boldsymbol{u}^{(n)} = [V]\mathbf{q}, \tag{3.43}$$

$$(\sum_{i}^{2}\sum_{j}^{2} a(\boldsymbol{\mu})[V]^T[L_{ij}(\boldsymbol{\mu})][V])\mathbf{q} = c\,[V]^T\mathbf{f}. \tag{3.44}$$

For the construction of $[V]$ we uniformly sample the parameter space twice per parameter at 0.1 and at 1.0, which gives a total of 16 different parameter combinations, and generate the POD from the covered solution space. We chose the first nine POD modes, which cover 99.8% of the energy. For testing purposes, we draw another 30 samples randomly from the parameters pace. The ROM takes 340 ± 3 μs to evaluate which is a speed up of factor 35.3 ± 0.4. Figure 3.14 shows the infinity norm of the error of the ROM over the parameter space in comparison to the FOM. It can be seen that the largest errors are within our training set.

The solution of the FOM can be seen in Figure 3.15a . The thermal conductivity was set to 0.1 $\frac{\mathrm{W}}{\mathrm{m\,K}}$ for the right blocks and to 1.0 $\frac{\mathrm{W}}{\mathrm{m\,K}}$ for the left blocks. We can see that, as expected, the blocks with a low value were capable of isolating the heat, whereas the two blocks with a high value transported the heat faster towards the boundaries, which resulted in a lower equilibrium. The corresponding ROM result based on nine POD modes can be seen in Figure 3.15b .

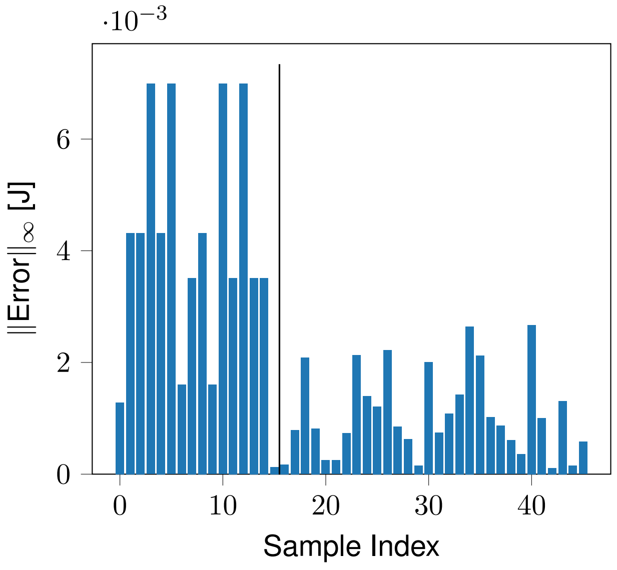
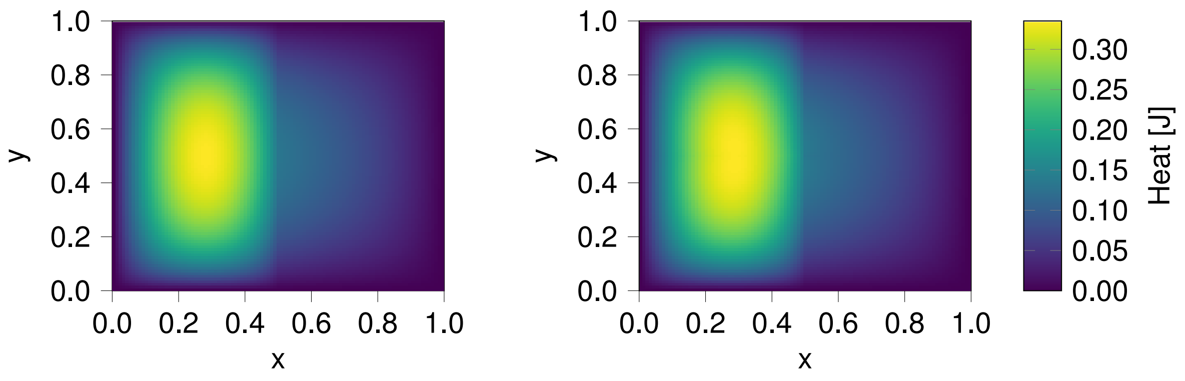


Figure 3.14.: We can see the infinity norm of errors of the ROM for every sample from the parameter space. The first 16 samples are the samples used for the POD the other 30 samples, separated by the black vertical line, are randomly drawn from the parameter space. The training set, representing the edges of the boundary have shown the largest errors.



(a) FOM (b) ROM

Figure 3.15.: Example solution of the FOM and the ROM with nine POD modes for stationary heat equation. The ROM gives an approximate solution which is already very close to the reference solution of the FOM.

3.3.3. Stochastic Reduced Order Model

The SROM corresponding to the static heat equation can be written as

$$\boldsymbol{U}^{(n)} = [\mathbf{W}]\mathbf{Q}, \tag{3.45}$$

$$(\sum_{i}^{2}\sum_{j}^{2} a(\boldsymbol{\mu})[\mathbf{W}]^{T}[L_{ij}(\boldsymbol{\mu})][\mathbf{W}])\mathbf{Q} = c\,[\mathbf{W}]^{T}\mathbf{f}, \tag{3.46}$$

where $\boldsymbol{U}^{(n)}$ is the stochastic counterpart to $\boldsymbol{u}^{(n)}$. Before we are able to optimise the SROM, we have to solve the adjoint equation for this problem, which is done in the next section.

Adjoint Problem

Solving the adjoint problem is the same as for the generic static model in Section 3.2.3. First, we write down the contribution to the Lagrangian and take the derivative regarding the sample of the SROB $[\mathbf{W}]$. Second, we take the derivative of the whole Lagrangian regarding $\mathbf{q}$ and set it to zero, to get $\mathbf{Zq}$. The first step results in

$$[\delta\mathbf{W}] \cdot \left(\sum_{i}^{2}\sum_{j}^{2} -a(\boldsymbol{\mu})([L_{ij}(\boldsymbol{\mu})]^{T}[\mathbf{W}]\mathbf{Zq}\mathbf{q}^{T} + [L_{ij}(\boldsymbol{\mu})][\mathbf{W}]\mathbf{q}\mathbf{Zq}^{T}) + c\,\mathbf{f}\mathbf{Zq}^{T} \right), \tag{3.47}$$

which is just a more general form of Equation [3.30] . The same holds for the calculation of $\mathbf{Zq}$

$$\sum_{i}^{2}\sum_{j}^{2} -a(\boldsymbol{\mu})[\mathbf{W}]^T[L_{ij}(\boldsymbol{\mu})]^T[\mathbf{W}]\mathbf{Zq} = [\mathbf{W}]^T\mathbf{ZO}. \tag{3.48}$$

After solving the adjoint method we validated the gradient via checking the convergence rate, analogous to Section 3.2.3. The memory required by the adjoint method for this problem is 1.4 GB.

SROM With and Without Discretisation

In this paragraph we are going to show the impact of not including the discretisation into the SROB on the example of the PyMOR thermal block SROM optimised over the nonlinear LS cost function from Equation [2.35] . We use this cost function, as we have not investigated yet, how well it leads to a generalised SROM over a parameter space and because the effect of having grid information was more significant for the nonlinear LS function than for the NLML function. Eventually, we will show the results for the SROM optimised over the NLML cost function, which resulted in a full coverage of the parameter space.

First of all, we want to repeat how the grid is included in the SROB. It is used for the correlation of the Gaussian random field, as mentioned in Section 2.2. This ensures that values of nodes close to each other correlate. With grid information, each node, excluded the boundaries, has two direct neighbours for a one dimensional example, eight direct neighbours for a two dimensional example and 23 direct neighbours for a three dimensional example. If the grid is not available, we consider the problem as one dimensional on a domain of [0,1], thus values close in the state vector correlate with each other. This means that not having the grid information for two or three dimensional problems leads to a more flexible model, as the nodes have less direct neighbours they correlate with. However, we might also include some correlation that do not make sense, dependent on the node numbering in the state vector. In summary, this means that we use less prior information which results in a more flexible model and we expect it to yield less confident confidence intervals.

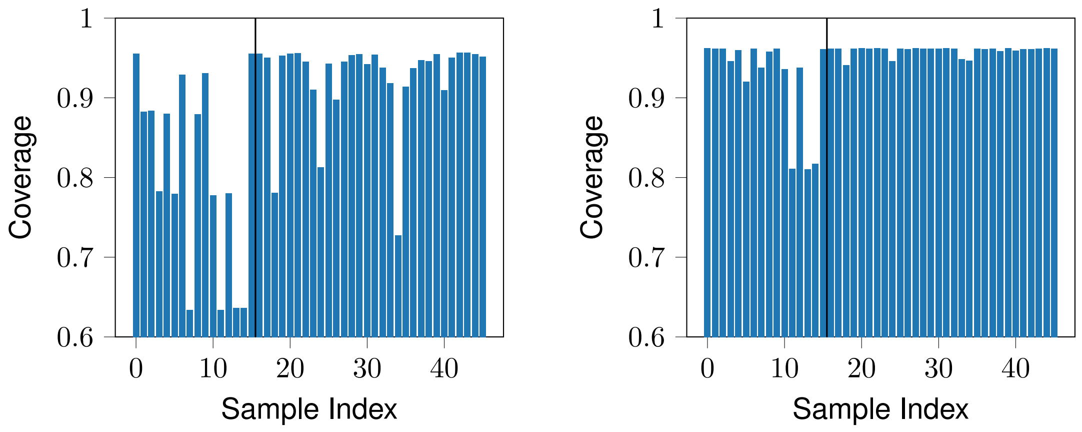
The given problem is two dimensional, thus we reduce the number of direct neighbours of a node from eight to two. We analyse the difference by looking at the coverage of the FOM by the SROM and by plotting the confidence intervals. The initial values were checked analogously to Section 3.2.2 and yielded after some iterations:

$$\boldsymbol{\alpha}_{03} = \{s_{03} = 0.5, \quad \beta_{03} = 0.2, \quad [\sigma_{03}] = 10 \cdot [I_n]\}. \tag{3.49}$$

The boundaries were set to

$$\begin{aligned}
\{&0.01 \leq s \leq 1,\\
&0.01 \leq \beta \leq 0.3,\\
&0.01 \leq [\sigma]_{11}, \dots, [\sigma]_{nn} \leq 20, [\sigma]_{kk'} \in \mathbb{R}, k < k'\}.
\end{aligned} \tag{3.50}$$

Furthermore, we kept using $\omega_j = 0.9$, as in the previous example, but we increased γ to 1.0 to be able to capture most of the variance between FOM and ROM. The SROM were optimised with and without including the information regarding the grid in the SROB. The optimisation with grid information via the OSA took 26 optimisation steps to reach a tolerance of 10^{-2}. The optimisation procedure terminated after 116.7 minutes and reached a value of $7.71 \cdot 10^{-2}$ for the nonlinear LS cost function. The version without the grid ran into the set maximum number of iterations of 100 after 647.54 minutes and a value of the cost functional of 7.59. As the value of the cost function was still continuously decreasing after 100 optimisation steps, we restarted the algorithm with the $\boldsymbol{\alpha}$ from the previous run and an acceptable tolerance of 100 to ensure convergence. After seven refinements of the acceptable tolerance, the algorithm reached an acceptable tolerance of 1.0. All runs together took 347 optimisation steps over 2243 min. It reached a minimal value of $1.49 \cdot 10^{-1}$ which is slightly higher than what we have achieved with considering grid information. Figure 3.16 shows the coverage of each of the SROMs over the parameter space. Both of the models do not reach a coverage of above 95%. However, the model which is not using the grid information, shown on the right, covers in average 94.8% of the FOM data point, which is significantly more than the 88.5% of the SROM which is using the grid data, shown on the left.



(a) With grid (b) Without grid

Figure 3.16.: Coverage of the FOM by the SROM of the PyMOR thermal block example optimised with the information regarding the grid on the left and without on the right. The training set is separated by a vertical black line to the test set. The left plot shows significantly less coverage within the training samples.

We have a closer look on the confidence interval for the sample with index 10, out of the training set. Specifically, we look at the slice with the maximum error of the ROM, which is the maximum error in the whole parameter space. We can see in Figures 3.17-3.18 that the confidence interval is a lot thinner for the model trained with the grid which supports our assumption. Furthermore, it has to be noticed that the SROM with grid information has an upper bound very close to the ROM without covering any point from the peak of the FOM. This emerges from the two terms of the nonlinear LS cost function. The mean term J_{mean}, which keeps the expectation of the SROM close to the FOM, and the standard deviation term J_{std}, which tries to cover the error between FOM and ROM in the variance of the SROM. As the distribution of the SROM in this area is highly skewed towards the bottom, the expectation value is still close and due to the tail of the distribution towards the bottom it might be that the variance is as large as the error between ROM and FOM thus, J_{std} is also close to a minimum. Even if the method performed rather poor in terms of coverage, it might be that we just reached a sub-optimal local minimum and there are other local minimums of the cost function which lead to better confidence intervals and therefore a better coverage. The SROM without grid information, however, managed to converge to a different minimum where the upper bound covers most of the peak from the FOM. The lower bound drops far below the FOM which is due to a highly skewed distribution from the SROM. This can originate from a more flexible model or from unintentionally introduced correlations between grid values.

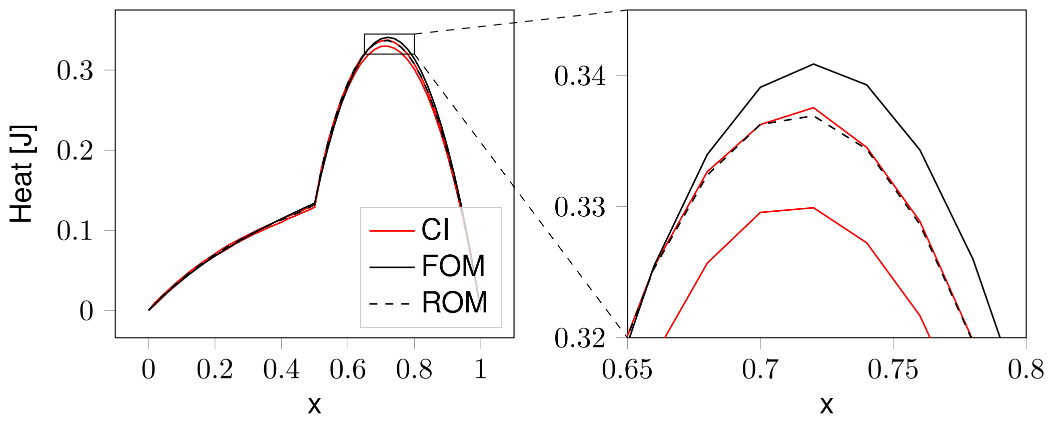


Figure 3.17.: This plot shows the slice of the domain with the maximum error between the ROM and the FOM over the parameter space. The confidence interval is from a SROM that uses the grid information to ensure the correlation between the values of grid points that are close in space. The zoomed frame shows that the computed interval does not cover any data point of the peak from the FOM, but the upper bound is very close to the ROM.

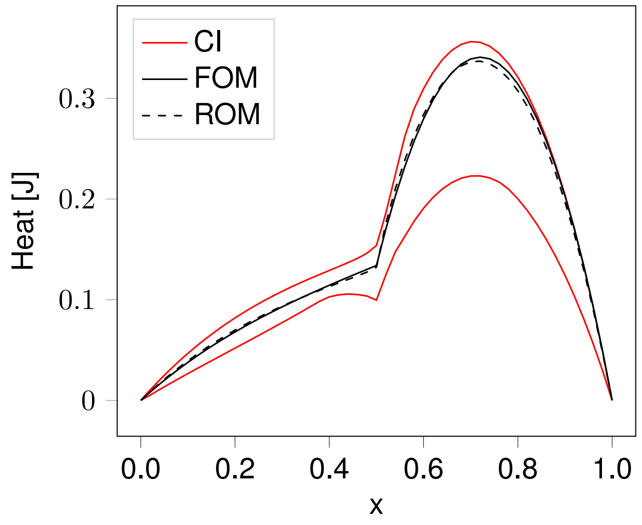
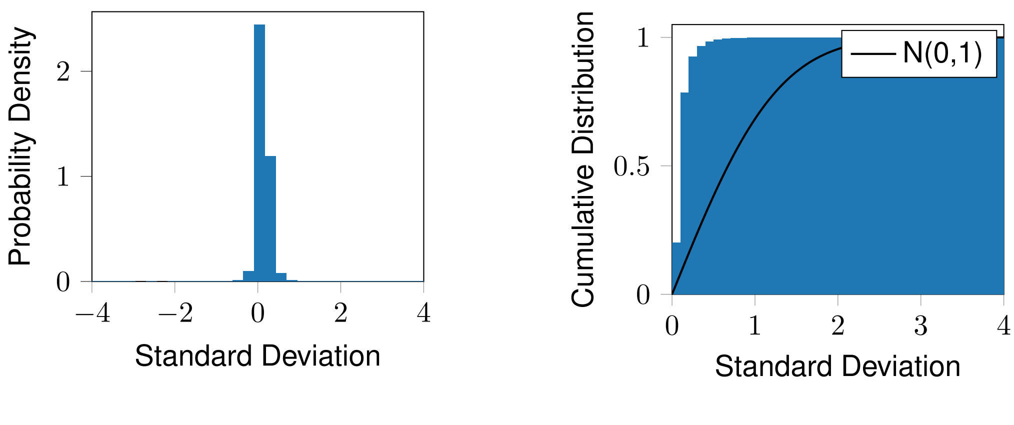


Figure 3.18.: This plot shows the slice of the domain with the maximum error between the ROM and the FOM over the parameter space. The confidence interval is from the SROM without considering the grid information which results in a broader confidence interval that covers most of the peak but is underconfident.

What we have seen in this paragraph supports our assumption that using grid data leads to more confident intervals and less flexible models.

Negative Log Marginal Likelihood

Using the NLML, the fact that the distribution might be skewed to the wrong side of the expectation value and therefore might not cover the FOM, is no longer a problem. With the NLML we optimise the standard deviation in a way that the FOM data points are covered under the assumption of a Gaussian distribution, thus the FOM is covered even if it is not covered by the actual distribution of the SROM. Using the same initial values and boundaries as for the optimisation via the nonlinear LS cost function, we achieved a minimum coverage per sample, with and without using the grid, of 99.8% over the whole parameter space for a 99.7% confidence interval, which does not violate the promised 99.7%. The optimisation without grid information reached the acceptable tolerance of 10,000 and held it for two iterations after ten iterations. The optimisation took 46.6 minutes. The final value for the NLML was $-3.27638 \cdot 10^5$. The acceptable tolerance had to be chosen that high as the IPopt algorithm checks the tolerance with Equation [2.51] which includes the gradient of the cost function which is supposed to go to zero. However, for a cost function with a value in the region of 10^5 an acceptable tolerance of 10,000 is approximately the same as the 10^{-2} for the nonlinear LS cost function.



(a) Probability Density

(b) Cumulative Distribution

Figure 3.19.: The probability distribution and the cumulative distribution of the FOM around the expectation value of the SROM for the pyMOR thermal block example in a standardised form. The grid information has been used for the SROM and the CDF of the normal distribution is plotted for comparison as solid black line. The values of the FOM are closely distributed around the expectation value of the SROM. However, there are are some outliers which could not be mapped into a range of four standard deviations. The coverage with a range of three standard deviations is exactly 99.7% of the data points.

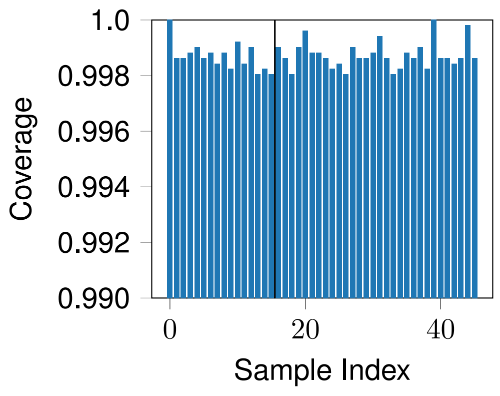
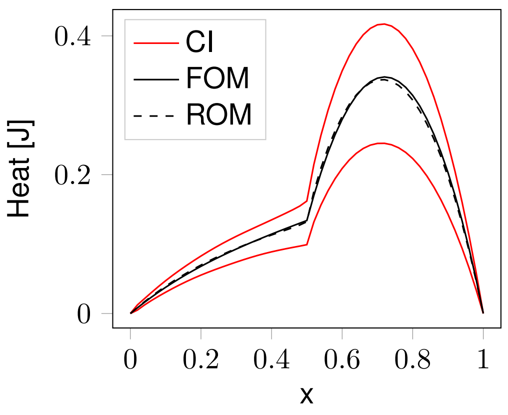
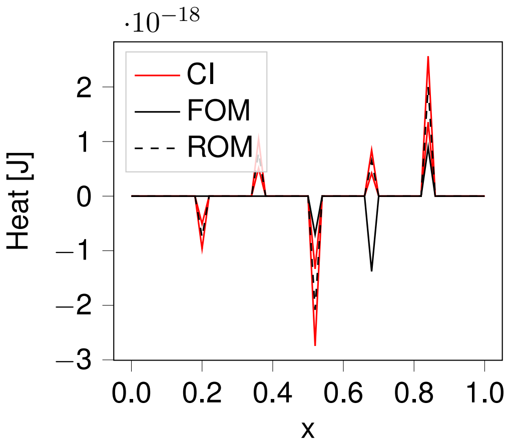


Figure 3.20.: This figure shows the coverage over the parameter space of the FOM by three standard deviations of the SROM with grid information optimised over the NLML cost function. Here it has to be noticed that the y-axis goes from 99% to 100% and that all samples have a coverage above 99.7%. The training set on the left is separated from the test set via the vertical black line.

Figure 3.19 shows that the optimisation algorithm was able to place most of the FOM data points very close to the expectation value of the SROM. However, there are some outliers in a distance above four standard deviations to the expectation value, which is why not all samples in the parameter space are covered with 100%, as shown in Figure 3.20 .



(a) Worst case in terms of the ROM error.



(b) Worst case in terms of coverage.

Figure 3.21.: 99.7% interval of worst case scenarios in terms of coverage on the right and in terms of ROM error on the left for the NLML optimised model with grid information. The confidence interval is underconfident. For the worst case of coverage we see the line of grid points next to the domain boundary with Dirichlet boundary conditions. There are some values of the FOM which look like and have the size of numerical errors. They were neither manged to cover by the ROM nor by the SROM, but contribute to the NLML cost function.

Figure 3.21a shows a 99.7% interval for the worst case in terms of the ROM error for the sample with index 10. we can see that the interval is highly underconfident, as most of the data points are within the range of one standard deviation, however, to get a 99.7% coverage one has to look at the full interval of three standard deviations, because of the outliers. On the right side, Figure 3.21b shows the slice of the domain, which is one grid row away from the boundary and represents the worst case in terms of coverage. The values are still almost zero as we have Dirichlet boundary conditions, but there are some values different from zero which originate from numerical errors as there are even negative values in the FOM which are nonphysical. Because those values are insignificant in their value in comparison to the values on the rest of the domain, they are not fully covered within the the first few modes of the POD. Hence, the ROM is not capable of accurately representing them. This is a problem as in the NLML the error of the ROM gets weighted by the standard deviation of the SROM, thus small values like this significantly contribute to the overall cost. Where this feature was one of the reasons why we haven chosen the NLML as cost function in the first place, it is rather a drawback in this case, as the algorithm tries to also cover the values corresponding to numerical errors which resulted then in other areas in an underconfident interval.

In terms of evaluation time, the SROM took for one sample of the parameter space 366 ± 3 ms which is by a factor of $(32.8\pm0.5)\cdot10^{-3}$ slower than the FOM, which is needed for later reference. This section has shown us, that including information regarding the grid can lead to thinner confidence intervals. Furthermore, we have seen that optimising the SROM over the nonlinear LS cost function led again to a maximum coverage of approximately 95%, which is not sufficient for our demand. In the end of this section, we have shown that the NLML cost function managed again to yield valid 99.7% intervals.

3.4. Operator Inference Based Transient Problems

In this chapter we describe two transient problems where the FOM was not available to us. We had only the state values of the FEM simulations from Siemens NX. Those values where taken directly from the solver, thus nodes with elements of higher order have multiple entries in the state vector and as we have not had the discretisation, we were not able to map the DOFs of the state space to individual nodes of the grid. This did not allow us to use the grid information for the SROM. Despite the missing grid information, the results support the assertion that the SROM also works without grid information. The simulations have been done on a machine with an Intel Xeon E5-2640 CPU and 48 GB memory. This machine was not available to us. Hence, we can not accurately compare the compute times from the FOM to the compute time of the ROM and the SROM we computed. However, an approximate comparison is still possible as our machine has, in comparison to the machine where the FOM ran on, a slightly high clock time (2.7 GHz to 2.5 GHz), but less cores (2 to 6) [14, 15] and as the SROM-framework of this thesis is mostly sequential and only partly parallelised over external libraries, the number of cores has only a small impact on the runtime of our tool, which lets us assume that the runtime would be approximately the same.

At first we will look at the heat sink problem which is a conduction dominated problem. Therefore, it is linear and suitable for the OI approach. The second problem, the gap radiation, has an additional radiation term with the temperature to the power of four. Hence, it requires a transformation to expose a form with a quadratic leading term. With those two problems we show that the nonparametric probabilistic approach can be used to generate valid confidence intervals for OI-based ROM which also generalise over a parameter space. Eventually, we show in the last paragraph that this method does not results in valid confidence intervals when we extrapolate in time and the temperature values reach values that are not covered by the training data.

3.4.1. Heat Sink

The heat sink problem is visualised in Figure 3.22 . The simulation consist of two parts. The first part, in red colour, is a copper chip with a thermal capacity of $385\frac{\mathrm{J}}{\mathrm{kg{\cdot}K}}$, a thermal conductivity of $387\frac{\mathrm{W}}{\mathrm{m{\cdot}K}}$, and a volumetric heat source of $0.05\frac{W}{mm^3}$ to $0.15\frac{W}{mm^3}$, which is the first dimension of the parameter space of this problem. The second part, in green colour, is an attached aluminium heat sink, with a thermal capacity of $963\frac{\mathrm{J}}{\mathrm{kg{\cdot}K}}$ and a thermal conductivity of $151\frac{\mathrm{W}}{\mathrm{m{\cdot}K}}$. The second dimension of the parameter space is the initial temperature. It can vary from $20°$C up to $50°$C. The phenomena can be described as a heat flow from the chip into the fins of the heat sink, where we have a very large surface which supports the dissipation into the air [42].

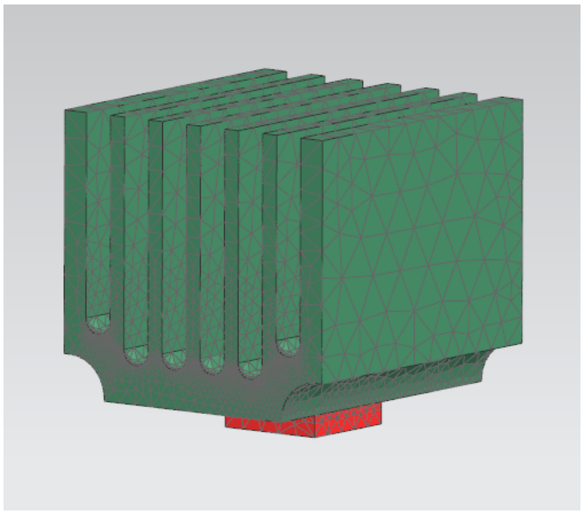


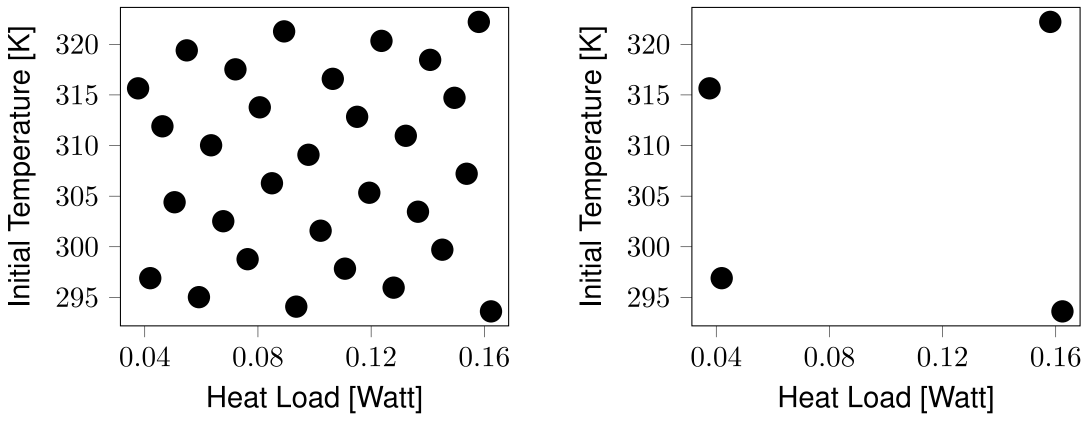
Figure 3.22.: Heat sink model [42]. The domain is split into two parts. The chip which acts as heat source is shown in red and the heat sink is shown in green.

Full Order Model

In this simulation we assume that the material properties are not temperature dependent, which results in a system governed by the linear heat transfer equation

$$[C_p]\dot{\boldsymbol{y}}(t) = [K]\boldsymbol{y}(t) + [B]\boldsymbol{u}, \tag{3.51}$$

with $[C_p]$ as thermal capacity matrix, $[K]$ as thermal conductivity Matrix, $[B]$ as load matrix, $\mathbf{y}(t)$ as temperature state vector at time t, and $\boldsymbol{u}$ as input vector. The matrices of the FOM are unknown, but we have a dataset of 30 simulations over 500 seconds with 100 snapshots, which where directly written out of the solver from a Siemens NX simulation. Every simulation is based on a different sample from the parameter space, shown in Figure 3.23a . The state values have a dimension of $N = 27413$ and are given in degree Celsius, which we had to transform to Kelvin (+273.15). Since we only have the result of the FOM, we neither have insight on the boundary conditions nor on the mapping from the DOFs of the state vector to the node coordinates. However, as we have already seen in Section 3.3 the SROM can handle missing grid coordinates. It is the same for the the boundary matrix, which again restricts the space of the POD, which can lead to faster convergence and thinner confidence intervals, especially at the boundaries. The FOM takes 87 seconds to execute. As we have not had excess to the machine the FOM was executed on, we could not conduct enough results to approximate the error.



(a) All samples (b) Used for POD

Figure 3.23.: This figure shows all the sampled initial temperatures (y-axis) and heat loads (x-axis) for the heat sink simulation that are within the provided dataset on the left, where the dots mark the parameter combination. The right figure shows only the four samples, which approximately span the parameter space, selected for the training set and the POD.

Reduced Order Model

For the ROM, we project the data onto a lower dimensional subspace via a ROB $[V]$ and used operator inference (OI) with L^2-regularisation to build the ROM. OI is a non intrusive method to build ROM. The method was introduced in Section 2.1.2. It infers a defined set of operators via a linear LS optimisation based on given snapshots from a system of interest. According to the FOM in Equation [3.51] , we infer a linear- and an input-operator. The ROM is then of the following form

$$\mathbf{y}^{(n)}(t) = [V]\mathbf{q}(t), \tag{3.52}$$

$$\dot{\mathbf{q}}(t) = [K]\mathbf{q}(t) + [B]\boldsymbol{u}, \tag{3.53}$$

where we assumed that $[C_p]$ is invertible and therefore implicitly included in the inferred operators $[K]$ and $[B]$. As we want to describe the parameter space with the least samples possible, we picked only the four samples from the corners of the parameter space as training set, shown in Figure 3.23b , which resulted in a very accurate model for the chosen samples. The training data was then used to built the POD and to infer the operators for the OI model as can be seen in Figure 3.25b . Furthermore, we dropped the initial state values of every simulation, as they are constant and can not be covered by the POD, leading to a high initial error from projecting the constant state to the lower dimensional space. The eigenvalues of the POD are shown in Figure 3.24 . We selected the first five modes for the reduced order basis, which cover 99.9999% of the energy.

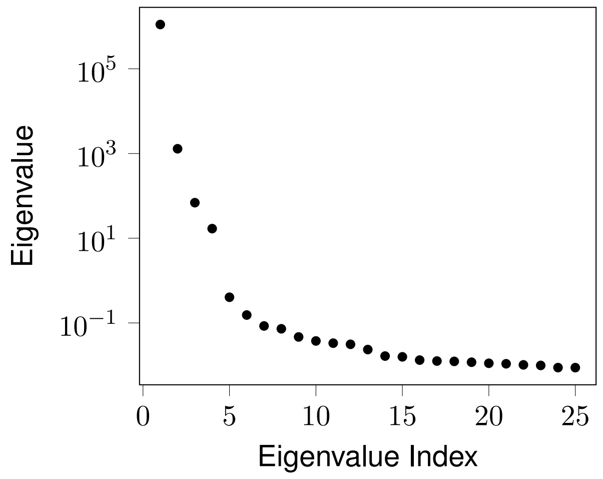
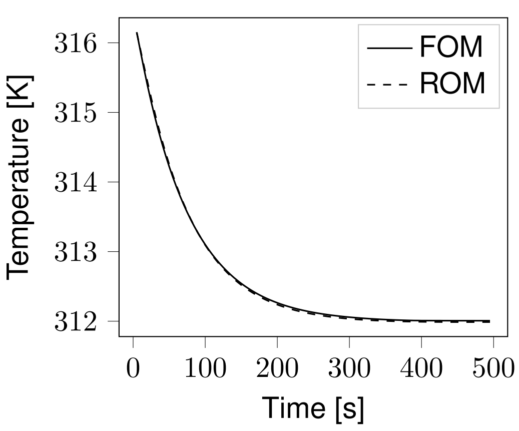
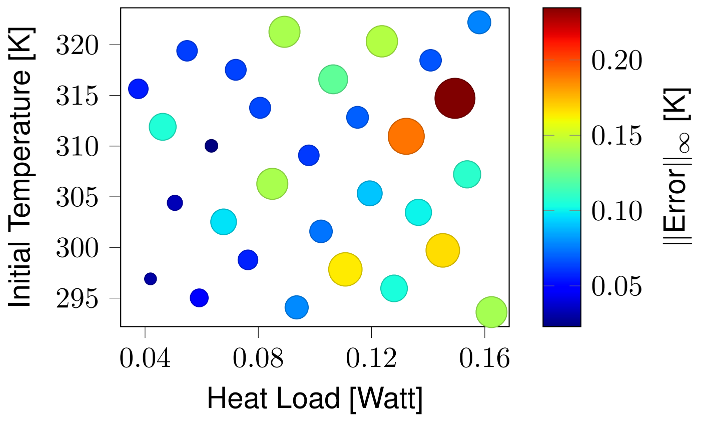


Figure 3.24.: The figure shows the eigenvalues (EW) from the POD of the selected training data for the heat sink problem in descending order on a log scale. We can see that most of the energy is covered within the first five EW.

The ROM simulation is then an initial value problem, which we solved via the Heun time stepping scheme. Using higher order time stepping schemes does not increase accuracy, as the OI model is built up on a second order time derivative approximation and the L^2-regularisation is optimised over solutions gained via the Heun time stepping scheme. With the inferred model and the chosen time stepping scheme we achieved an infinity error of $0.139K$ on the training data and an infinity error of $0.235K$ on the whole dataset as can be seen in Figure 3.25b . As an example, Figure 3.25a shows the 99th DOF of the simulation corresponding to the upper left sample in Figure 3.23a . The initial temperature is relatively high at a low heat load, which results in a convergence value below the initial temperature. For higher heat loads or lower temperature we can observe an increasing temperature like it can be seen later in 3.27.



(a) The vector state value of the 99th DOF of the first training simulation over time.



(b) Infinity error over all samples, where the value is indicated by colour and size of the circle.

Figure 3.25.: The 99th DOF of the first training simulation over time on the left and the infinity error over all samples of the parameter space on the right. Both figures show that the ROM describes the FOM very accurately with an infinity error of 0.2 Kelvin.

The ROM took 8.6 ± 0.6 ms to evaluate on our machine which is roughly 10,000 times faster than the evaluation on the FOM, which comes from the drastic reduction of DOF from 27413 to 5.

Stochastic Reduced Order Model

The corresponding SROM is of the form

$$\boldsymbol{Y}^{(n)}(t) = [V]\boldsymbol{Q}(t), \tag{3.54}$$

$$\dot{\boldsymbol{Q}}(t) = [\boldsymbol{K}]\boldsymbol{Q}(t) + [\boldsymbol{B}]\boldsymbol{u}. \tag{3.55}$$

There are two main differences to the static problems in Section 3.2 and 3.3. The first, difference is the time dependency. Hence, we had to include the time stepping scheme in the adjoint problem, which is shown in Appendix A.1.3. The adjoint method of this problem has a memory consumption of 4.4 GB due to the high dimension of the FOM of 27413. To be able to cover a dimension of this size it was necessary to recompute the high dimensional state values as stated in Section 3.2.3. Without this adjustment, the memory usage would have been 92 GB. The second difference is that we do not have a projection based ROM anymore, but the operators are inferred via a linear LS minimisation thus, the adjoint problem has to be recompute for OI-base ROMs which is shown in Appendix A.1.4.

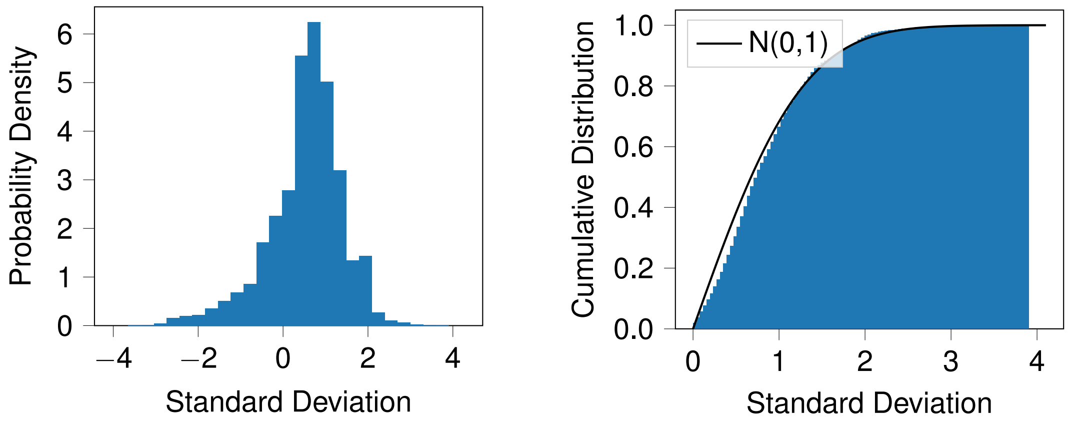
For the optimisation we chose the same initial values $\boldsymbol{\alpha}_{01}$ as in Section 3.2.2

$$\boldsymbol{\alpha}_{01} = \{s_{03} = 0.05, \quad \beta_{01} = 0.2, \quad [\sigma_{01}] = [I_n]\}. \tag{3.56}$$

However, we reduced the lower bound for s, as the ROM was quite accurate and only little fluctuation was needed to cover the error to the FOM. Therefore, we set the boundaries to

$$\begin{aligned}&\{10^{-4} \leq s \leq 1,\\ &0.01 \leq \beta \leq 0.3,\\ &0.01 \leq [\sigma]_{11}, \ldots, [\sigma]_{nn} \leq 20, [\sigma]_{kk'} \in \mathbb{R}, k < k'\}.\end{aligned} \tag{3.57}$$

Analogously to the problems in Section 3.2 and 3.3 optimising the SROM over the non-linear LS cost function did not reach the level of confidence we would like to have and neither did it for this problem. With $\omega_j = 0.8$ and $\gamma = 1.0$ we reached only an average coverage of 70% with a maximum coverage of 99.0% and a minimum coverage of 7.6% for the sample with the highest ROM error. The optimisation via the OSA took 163.9 minutes to reach a tolerance of 10^{-2} after 11 optimisation steps.



(a) Probability Density (b) Cumulative Distribution

Figure 3.26.: The probability distribution and the cumulative distribution of the FOM around the expectation value of the SROM for the heat sink problem normed by its standard deviation. The optimisation has been done over the NLML cost function. The CDF of the normal distribution is plotted as solid black line to indicate valid areas of the confidence intervals, which is in this case above 1.5 standard deviations. Furthermore, it can be seen that the whole FOM training set lays within four standard deviations.

Optimising the SROM over the NLML function via the OSA, with $\boldsymbol{\alpha}_{01}$ as initial conditions and the limits defined above, reached an acceptable tolerance of 10.000 over 2 acceptable iterations after 15 optimisation steps and 312.9 minutes. The result shows an improved coverage for a 99.7% interval, but it does not cover the FOM over the parameter space up to the expected level of 99.7%. This can origin from the fact that the maximum error of the ROM over the parameter space was not included in the training set. In Figure 3.26a we can see the difference of FOM to the expectation value of the SROM in a standardised form. The optimisation managed to bring all the FOM data points from the training data within a range of four standard deviations which compares to a confidence of 99.994%. Figure 3.26b shows that the NLML optimisation results in a SROM with a valid confidence interval from 1.5 standard deviations on, if this coverage is true for each of the training samples. This is not the case for the 99.7% confidence interval, therefore, we have to use the 99.994% interval.

The coverage of the 99.994% interval from the SROM over the whole parameter space is shown in Figure 3.27 . The average coverage is 88.2% with a maximum coverage of 100% on the training data and a minimum coverage of 20.6% of the sample with the highest SROM error. This is already better than the result from the optimisation over the nonlinear LS cost function, but by far not accurate enough. The evaluation of the SROM took 13.07 ± 0.15 seconds which is 6.6 times faster than the the execution of the FOM.

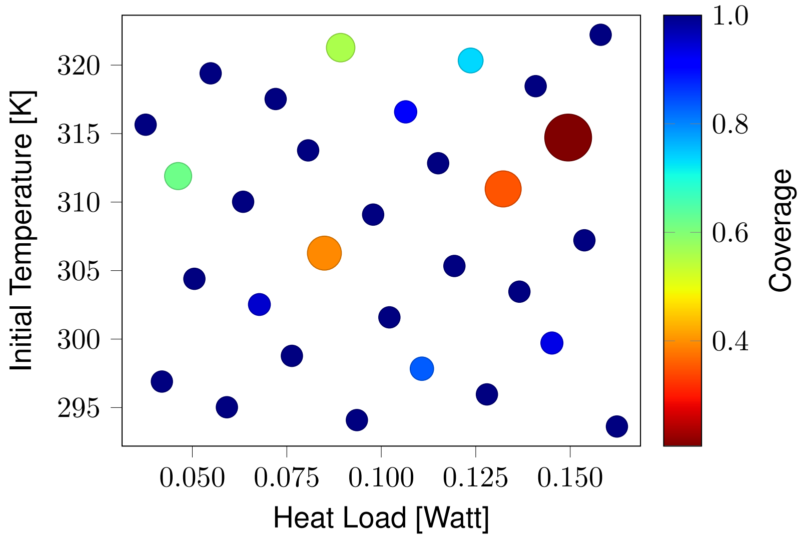


Figure 3.27.: Coverage of the FOM over the parameter space for a 99.994% interval for the NLML, where the coverage is indicated by its colour and the size of the circles regarding the sample in the parameter space. The size is inverse proportional to the coverage. All samples from the training set are covered by 100%, whereas the coverage inside the domain goes down to 20.6% which seems to correlate with the ROM infinity error we have seen in Figure 3.25b

The two difference to the PyMOR example in Section 3.3 are that we do not use an intrusive reduction method anymore, and that the maximum error of the ROM was within the training set for the SROM. As the current problem is a SROM for an OI-based ROM, it can not account for sampling errors, and the training set does not include the maximum error of the ROM, which leads to a lower coverage over the parameter space. The worst case in terms of coverage and ROM error can be seen in Figure 3.28 . The interval is very close to the FOM but not large enough to actually cover it.

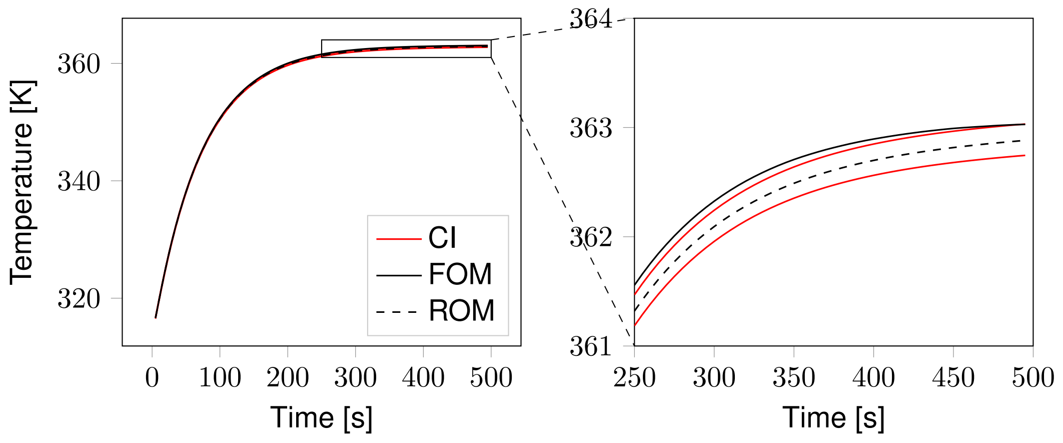


Figure 3.28.: 99.994% confidence interval from the NLML optimised model for the sample with the highest ROM error. The plotted index of the state space is 26313. In the zoomed window it can be seen that the FOM is only slightly out of the confidence interval.

Even if the method was not able to reach the desired coverage over the whole parameter space, it was able to cover 100% of the training data. A reason for its poor generalisation over the parameter space is that that the ROM was able to rather accurately describe the training set, but for other parameter samples in the parameter space it yielded a higher maximum error. By comparing Figure 3.25b with Figure 3.27 we can see a clear inverse correlation of the ROM error and the coverage. Additionally, we were restricted to the samples of the parameter space we had from the given data. It would be interesting to see how the method performs on a training set gained from a greedy sampling algorithm as proposed by Soize [36].

3.4.2. Gap Radiation

This last presented problem shows that the investigated UQ approach can work for OI with lifting, and is therefore potentially applicable to a wide range of PDEs. Firstly, the model is described and how to perform the lifting operation to make OI applicable to the problem. Afterwards, we present the results for the SROM optimised over the NLML via the OSA, which show that the SROM is capable of describing the errors of the ROM over the whole parameter space. As last experiment we tried to extrapolate the SROM in time, which led again to invalid confidence intervals.

The model of this problem is shown in Figure 3.29 and has been taken from [42]. It consist of two stacked steel plates, with a gap inbetween, and a heat load on top of the upper plate. The gap between the plates prevents direct heat transfer from the upper to the lower plate under the assumption that the scenario is in vacuum. The plates have a thermal capacity of $434 \frac{\mathrm{J}}{\mathrm{kg \cdot K}}$, and a thermal conductivity of $14 \frac{\mathrm{W}}{\mathrm{m \cdot K}}$. Similar to the heat sink scenario, the parameter space is two dimensional, the initial temperature $[20°\mathrm{C}, 50°\mathrm{C}]$ and the heat load $[20\mathrm{W}, 80\mathrm{W}]$ [42].

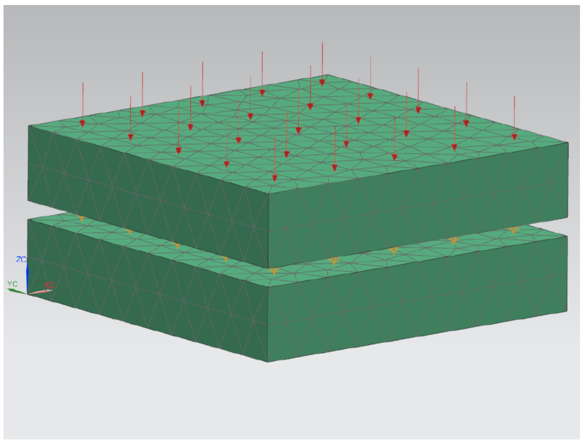


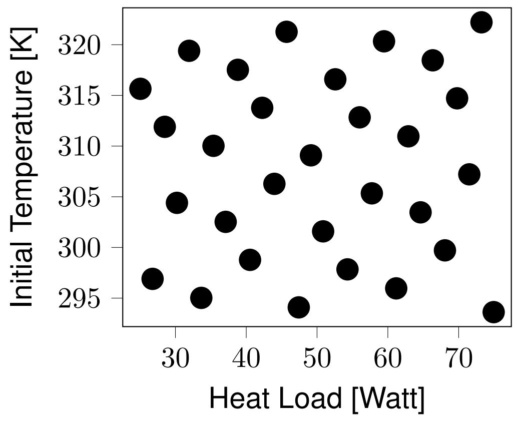
Figure 3.29.: The Gap radiation model consists of two stacked steel plates, with a gap inbetween, and a heat load on top of the upper plate [42].

Full Order Model

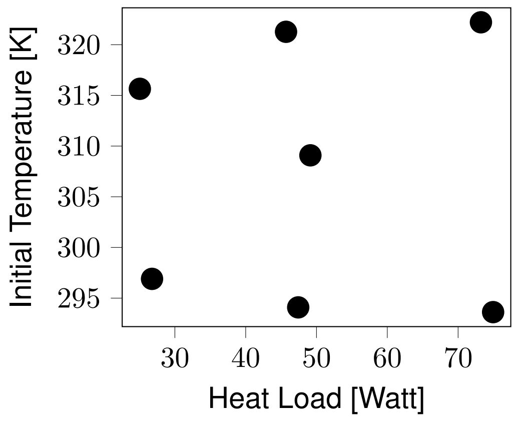
We assume again that the material properties are temperature independent. This results in the following equation for the FOM

$$[C_p]\dot{\mathbf{y}}(t) = [K]\mathbf{y}(t) + [R]\mathbf{y}^4(t) + [B]\boldsymbol{u} \tag{3.58}$$

where $[C_p]$ is the thermal capacity matrix, $[K]$ as thermal conductivity matrix, $[B]$ is the matrix that distributes the heat load over the state space, $\mathbf{y}(t)$ is the temperature at a certain time t, and $[R]$ is the radiation matrix. Here we also had 30 simulations with different parameter combinations, shown in Figure 3.30a . The simulations have a state space dimension of $N = 3686$ and a time span of 3600 seconds with 100 snapshots. One simulation of FOM took 24 seconds. The state values are again in degree Celsius, and had to be converted to Kelvin (+273.15).



(a) All samples



(b) Used for POD

Figure 3.30.: Sampled initial temperature and heat load for gap radiation simulation. The samples are indicated with black dots. For the gap radiation problem we used seven samples.

Reduced Order Model

For the ROM we used OI to infer a linear-, a quadratic- and an input-operator, which is not the same structure as Equation [3.58] . However, the Equation [3.58] can be transformed via lifting to a form with a quadratic leading term, which is shown in the following. The inference was done base on a set of snapshots, which were first transformed via the lifting operation and reduced by a generated ROB. The ROM has then the form of

$$\mathbf{y}^{(n)}(t) = [V]\mathbf{q}(t), \tag{3.59}$$

$$\dot{\mathbf{q}}(t) = [L]\mathbf{q}(t) + [H](\mathbf{q}(t) \otimes \mathbf{q}(t)) + [B]\boldsymbol{u}, \tag{3.60}$$

assuming that $[C_p]$ is invertible. To be able to use this ROM we had to lift the state space into a form where $\dot{\mathbf{y}}$ has a quadratic leading term. Hence, we extended the state space by the state values to the power of two, and the state values to the power of three. Since the state values are all around 300 K, those additional state values were around 10^5 larger than the original values. For the quadratic operator we had to calculate $(\mathbf{y} \otimes \mathbf{y})$, which results in an even larger difference. This led to an over-prioritisation of the larger values in the linear LS optimisation of the operator inference. Therefore, we scaled the new state vector down by the maximum value of the lifted state vectors from the training dataset. Hence, all values were between zero and one. The newly introduced variables are

$$\mathbf{y}_1 = \frac{\mathbf{y}}{c}, \quad \mathbf{y}_2 = \frac{\mathbf{y}^2}{c}, \quad \text{and} \quad \mathbf{y}_3 = \frac{\mathbf{y}^3}{c}, \tag{3.61}$$

with the constant scaling factor c. The dynamics of the system described through the newly introduced variables is

$$\dot{\mathbf{y}}_1 = \frac{1}{c}\dot{\mathbf{y}}, \quad \dot{\mathbf{y}}_2 = \frac{2}{c}\mathbf{y}\dot{\mathbf{y}}, \quad \text{and} \quad \dot{\mathbf{y}}_3 = \frac{3}{c}\mathbf{y}^2\dot{\mathbf{y}}.$$

By using Equation [3.58] and Equation [3.61] we get

$$[C_p]\dot{\mathbf{y}}_1 = [K]\mathbf{y}_1 + c[R]\mathbf{y}_2^2 + \frac{1}{c}[B]\boldsymbol{u}, \tag{3.62}$$
$$[C_p]\dot{\mathbf{y}}_2 = 2\left([K]\mathbf{y}_2 + c[R]\mathbf{y}_2\mathbf{y}_3 + \mathbf{y}_1[B]\boldsymbol{u}\right), \tag{3.63}$$
$$[C_p]\dot{\mathbf{y}}_3 = 3\left([K]\mathbf{y}_3 + c[R]\mathbf{y}_3^2 + \mathbf{y}_2[B]\boldsymbol{u}\right). \tag{3.64}$$

For creating the required form, we dropped all constant factors, as they are going to be inferred anyway, and neglected the $\mathbf{y}_1$ and $\mathbf{y}_2$ in front of $[B]$ in Equation [3.63] and Equation [3.64] which means we set them equal to one. This resulted in in the required form with a quadratic leading term and a state dimension of $N = 11058$, which is three times the original one.

This transformation was done on the whole dataset, which we could use then to build a ROM. First, we built a POD based on a selection of the available simulations. We chose seven samples, one from each corner of the parameters space and one from top, middle and bottom of the average heat load, as shown in Figure 3.30b . In comparison to Section 3.4.1, we have chosen more data points in the training sample due to an error of 20 K on the initial training set. We kept sampling until, we got a reasonable error on the training data. We dropped again the initial state of every simulation for the same reasons as for the heat sink example. Afterwards, we built the POD based on the lifted and scaled training set. As we lifted the state space, we did not want to minimise the integral over time of the whole state space, as it was done in Section 2.1.1, but only of the first third of it which represents the actual state values. Therefore, we generate the POD as if it were a static problem, as it was noticed that the maximum error of the ROM doubles in case we used the POD for time dependent problems. The resulting eigenvalues of the POD are shown in Figure 3.31 . The built ROM had an evaluation time of 8.6 ± 0.6 ms which is a speedup of the factor 2790.

We decided to use the first 12 eigenvectors to build the ROM, which cover 99.994% of the total energy. With the POD we can infer the operators. The resulting model achieved an infinity error of 5.23 K on the training data, which is also the infinity error of the whole dataset.

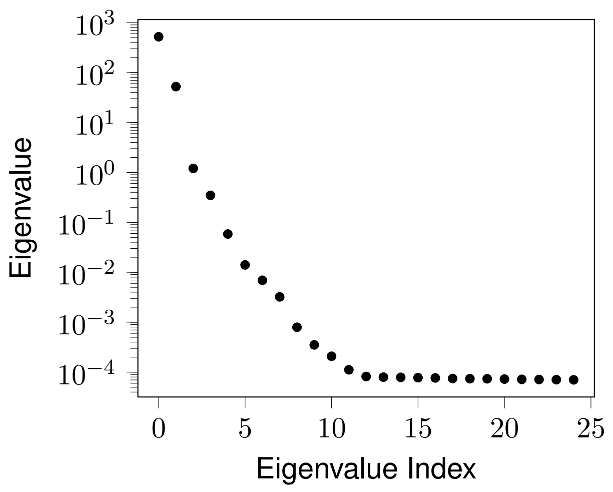
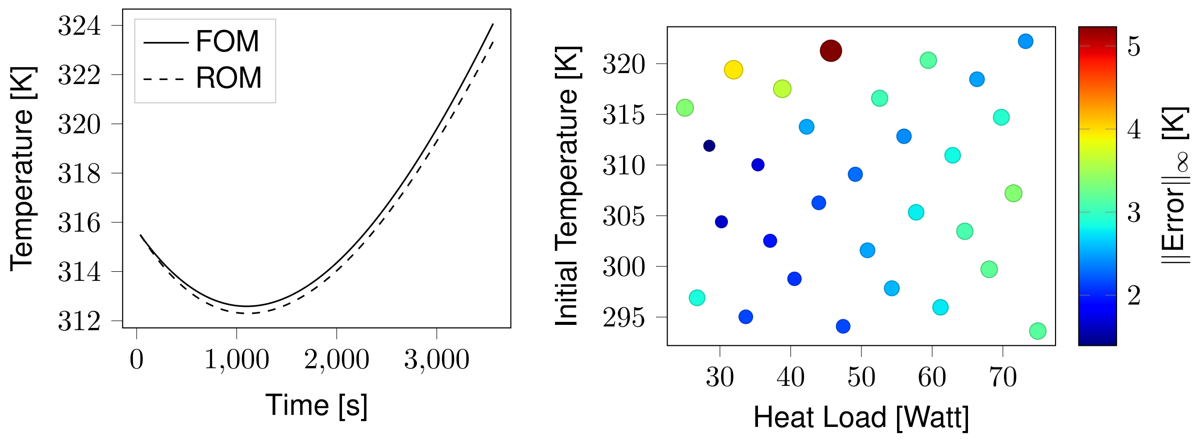


Figure 3.31.: Eigenvalues from the POD of the selected training data for the gap radiation problem in descending order on a log scale. It can be seen that most of the energy is covered by the first 12 EWs.



(a) The 99th state vector value of the first training simulation over time.

(b) Infinity error over all samples.

Figure 3.32.: The value of the 99th DOF of the first training simulation from the gap radiation problem on the left and the infinity error over all samples from the parameter space on the right where the infinity error is indicated by the colour and is proportional to the size of the circles. The ROM is visible different from the FOM and has a maximum error of 5.23 Kelvin.

Stochastic Reduce Order Model

The SROM is then of the form

$$\boldsymbol{Y}^{(n)}(t) = [V]\boldsymbol{Q}(t), \tag{3.65}$$

$$\dot{\boldsymbol{Q}}(t) = [\boldsymbol{L}]\boldsymbol{Q}(t) + [\boldsymbol{H}](\boldsymbol{Q}(t) \otimes \boldsymbol{Q}(t)) + [\boldsymbol{B}]\boldsymbol{u}. \tag{3.66}$$

To optimise the SROM we just had to make one small adjustment to the adjoint problem due to the usage of lifting. As the SROB is used on the already lifted data we did not have to take lifting into account in the beginning. However, when we compute the cost, we have to transform the lifted state back to the original state space, which has to be considered in the gradient computation, shown in Appendix A.1.5.

This time, we directly optimised the SROM for the NLML cost function, as the nonlinear LS cost function did not lead to confidence intervals in the region of 99.7%. The initial values have been set to

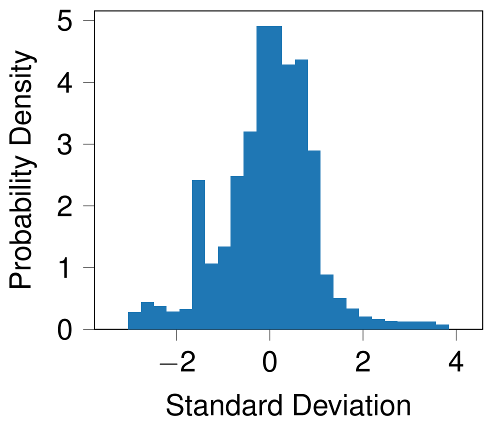
$$\boldsymbol{\alpha}_{04} = \{s_{04} = 10^{-7}, \quad \beta_{04} = 0.2, \quad [\sigma_{04}] = [I_n]\}, \tag{3.67}$$

where s is chosen to be that small, as the POD, which is going to be perturbed, acts on the scaled state vector. Therefore, small fluctuations have a huge impact. The boundaries were set to

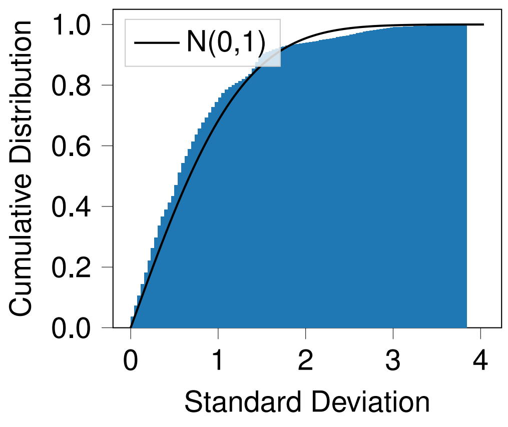
$$\begin{aligned}&\{10^{-9} \leq s \leq 10^{-5},\\ &\ 0.01 \leq \beta \leq 0.3,\\ &\ 0.01 \leq [\sigma]_{11}, \dots, [\sigma]_{nn} \leq 20, [\sigma]_{kk'} \in \mathbb{R}, k < k'\}.\end{aligned} \tag{3.68}$$

The optimisation with the OSA took 223 min for six optimisation steps. The resulting distribution of the FOM around the expectation value of the SROM is shown in Figure 3.33a the optimisation algorithm managed to place all data points of the training set within a range of four standard deviations. The corresponding cumulative distribution (Figure 3.33b) indicates valid confidence intervals from zero to almost two standard deviations. For high confidence the interval is only valid from four standard deviations on.

The interval of a range of three standard deviations covers exactly 99.7% of the training data, but only in average. The minimum coverage over the training set is 93.6% for a range of three standard deviations. That is why we increased the range to four standard deviations. The 99.994% interval of the optimised SROM covers 100% of all samples over the parameter space, which validates the confidence intervals. Furthermore, we have to mention that this time the maximum error of the ROM was among the samples in the training set. The maximum error of the ROM with the 99.994% confidence interval from the SROM is shown in Figure 3.34 . In the end of the time frame, the FOM is already very close to the confidence interval. The evaluation of the SROM at one point in the parameter space took 13.38 ± 0.21 seconds which is a speedup of a factor of 1.8 to the FOM.



(a) Probability Density



(b) Cumulative Distribution

Figure 3.33.: The probability distribution and the cumulative distribution of the gap radiation FOM training data around the expectation value of the SROM standardised by the SROMs standard deviation. The CDF of the normal distribution is plotted for indicating valid regions of the confidence interval. The plot shows that all data points could be placed in a range of four standard deviations and that the coverage at three standard deviations is exactly 99.7% over the whole training set.

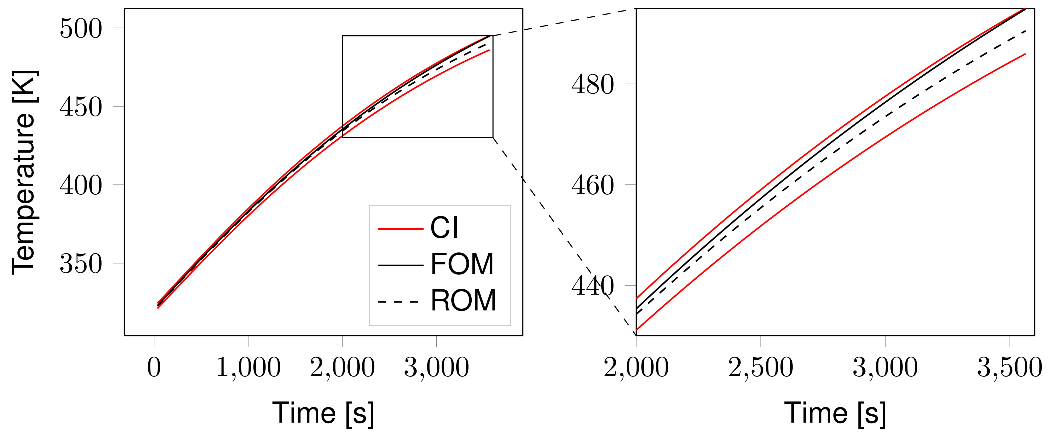
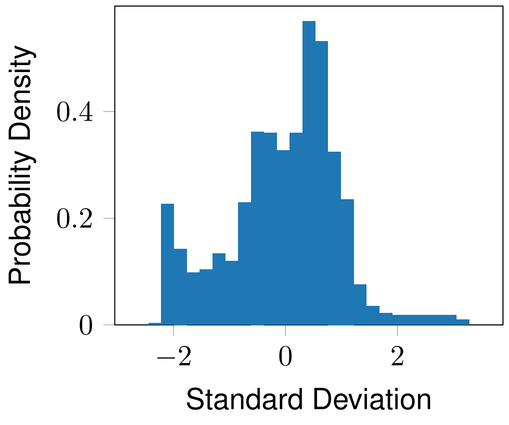


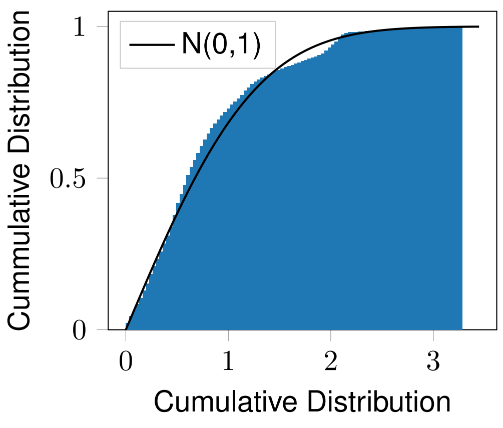
Figure 3.34.: 99.994% confidence interval for the maximum error of the ROM regarding the FOM over the parameter space. This sample of the training set would only be covered by 93.6% in case of an interval of three standard deviations.

Time Extrapolation

As in this case the model manged to generalise over the parameter space, we wanted to check the capability of the SROM to handle extrapolation in time. For the optimisation, we chose the same samples from the parameter space. However, we used only the first 70 from 100 snapshots. The first initial constant state is still neglected. Optimising the SROM with the same initial values and boundaries as earlier in this section led to the distribution of the FOM training around the expectation values, shown in Figure 3.35a . All of the training data lay within a close range of approximately 3.5 standard deviations. The corresponding valid confidence intervals are indicated by Figure 3.35b . We get valid confidence intervals in the range of one standard deviation up to approximately 1.5 standard deviations and from 2.2 standard deviations upwards.



(a) Probability Density



(b) Cumulative Distribution

Figure 3.35.: The probability distribution function and the cumulative distribution function of the gap radiation up to timestep 70 around the expectation value of the SROM normed by its standard deviation. The CDF of the normal distribution is plotted as black solid line for indicating valid regions of the confidence interval. All data points could be placed within a range of slightly above three standard deviations, but the three standard deviation range covers again 99.7% of the FOM data points.

The coverage of the optimised SROM over the whole parameter space is shown in Figure 3.36 . It is striking that the coverage of the samples on the upper edge of the parameter space have significantly less coverage than the rest of the samples which happened because those samples extrapolation goes to temperatures which were not covered by the training data. Over the whole parameter space 21 out of 30 samples were covered with 100%.

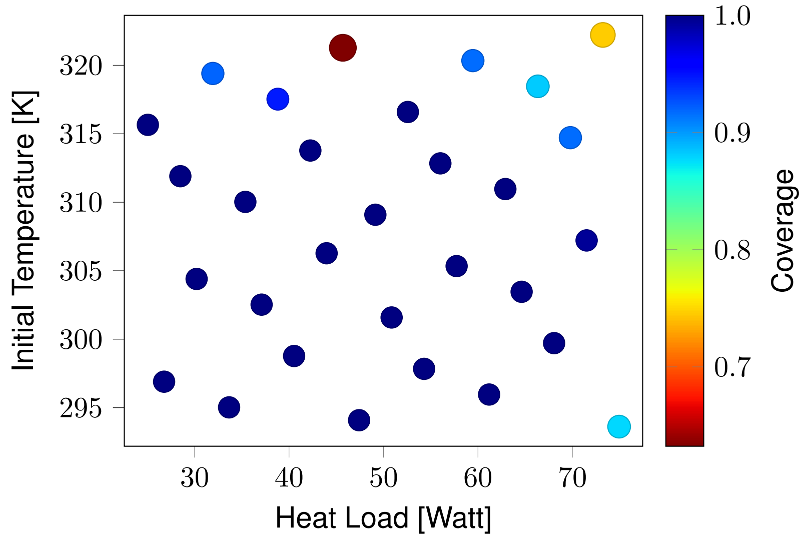


Figure 3.36.: Coverage of the FOM over the parameter space for a 99.7% interval for the NLML optimised model.

Figure 3.37 shows the same sample as for the SROM trained on the whole time frame, with the maximum error of the ROM. The zoomed part of the figure shows the data points which were not covered by the training data. We can see that the FOM leaves the confidence interval soon after leaving the area covered by the training data set.

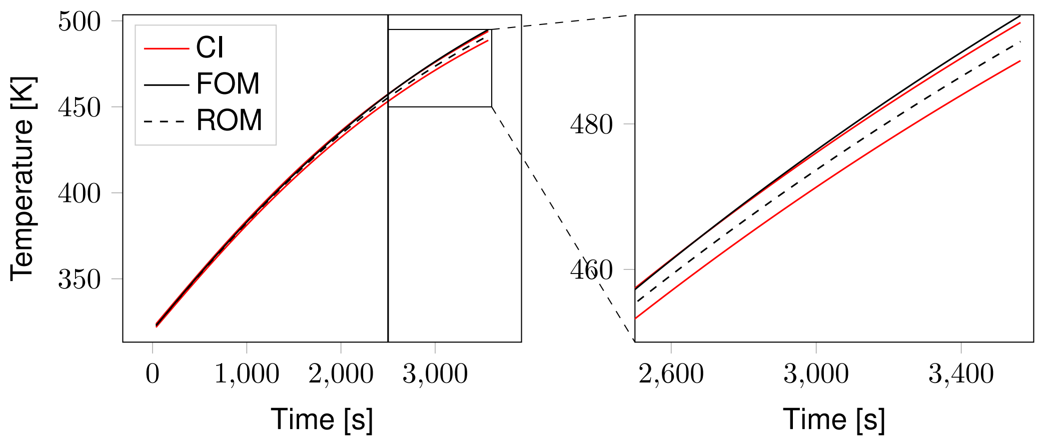


Figure 3.37.: 99.7% confidence interval of the SROM for the gap radiation problem for the maximum error of the ROM compared to the FOM over the parameter space.

In this section we have seen that the nonparametric probabilistic approach from Soize and Farhat is in principle capable of describing the uncertainty of an OI-based ROM over a parameter space. However, for this example we could not show that it works for an extrapolation in time.

4. Conclusion

In the final section of this thesis, we want to summaries our results and add a brief discussion. Eventually, we give an outlook on possible further research directions that could follow based on this thesis.

4.1. Summary and Discussion

The overall goal was to validate the nonparametric probabilistic approach for UQ from Soize and Farhat [36] for μ-parametric OI-based ROMs and to achieve confidence intervals in the region of 99.7% and above. Furthermore, we wanted to improve the training and evaluation procedure of SROMs to be able to run this method on a laptop as used in this thesis.

Therefore, we developed a tool for building SROMs in a highly modular way that can be easily extended and understood as described in Section 3.1. The modular structure is quite useful for trying different cost functions, models and optimisation algorithms, as they can be added by implementing the defined interfaces and then used in a plug and play fashion. It supports not only OI-based ROMs but also projection-based ROMs. The optimisation of the SROMs was improved by introducing an analytical gradient via the adjoint method which still requires the implementation of several functions regarding the model in case of a projection-based ROM, but is fully automated for OI-based ROMs without lifting. Even if the code is still highly sequential, we shortened the optimisation time up to a factor of 80 in comparison to the former algorithm with a FD gradient approximation by introducing the analytical gradient and the OSA, for the generic static problem in Section 3.2. Without this optimisation we would not have been able to optimise the SROM for the other problems covered in this thesis due to time constrains. However, even if we could achieve a significant speedup for the training procedure, the resulting SROMs still take approximately 1,000 times longer than the ROM of a problem as the SROM is built based on 1,000 samples of the SROB. Hence, to be worth using the SROM, the speedup of the ROM has to be over 1,000 for the current implementation. This is not given for the static problems in Section 3.2 and 3.3. This is alright as those problems were only for exploration purposes and not the real problems of interest. For the transient problems which are computationally more intense, the SROM led to a speedup factor of 13 for the heat sink problem and of 1.8 for the gap radiation problem. This difference origins from the higher speedup of the ROM for the heat sink problem. Considering that the program is mostly sequential, it still possible to get further speedup via parallelisation. However, taking into account the time it takes to set up the SROM and to train it, it depends on the number of evaluation points that we are interested in, whether it is worth setting up a SROM.

To achieve the objective of gaining confidence intervals which are valid in the range of 99.7% and above, we introduced the NLML cost function in Section 3.2, as the nonlinear LS cost function has shown issues with covering areas of low error from the ROM to the FOM close to the domain boundaries. The SROM optimised over the NLML has shown an improved coverage of the FOM data points throughout the whole thesis. But it has also shown a highly underconfident interval for the static pyMOR problem in Section 3.3 which was possibly caused by numerical errors close to the Dirichlet boundaries which the SROM tries cover with its confidence interval. This issues might be resolvable by removing the boundary matrix from the SROB, this leads to non zero values on the boundary which should allow the SROM to cover all values within its confidence interval. The SROM is going to be underconfident on the boundary, but might regain more confidence in the inner part of the domain. This would lead to an overall better solution, as we assume that the interval on the boundaries grows only as much as it has to, to cover all the values of the FOM. Additionally, we used the pyMOR problem to showcase the impact of not using the grid information and on how to use the developed tool with a model implemented in pyMOR. It was clearly visible that not including the grid information leads to an underconfident interval for the SROM which was caused by a skewed distribution of the SROM. The interval yielded covers more data points of the FOM than the SROM without grid information. This shows that the UQ method works even without the grid information, but it can lead to underconfident intervals.

However, this was not the case for the two OI-based SROMs, where the grid was not available. As already mentioned achieved both SROMs a speedup in comparison to the FOM, but only the SROM for gap radiation was able to yield a coverage over the parameter space of 99.7%, where we saw that we have to look at the coverage of each individual sample from the training set to determine the valid range of the confidence interval as it might only be valid in average. The heat sink SROM perfectly covered the training set but could not generalise to the parameter space. It seems to be related to the maximum error covered by the training set. All samples with an error above the one included in the training set have shown an inverse correlation of the coverage to the error of the ROM. We have shown that the probabilistic approach for UQ proposed from Soize and Farhat is in principal capable of learning the errors of a OI-based ROM and to generalise over a parameter space. However, it still needs further investigation regarding different sampling strategies which was in this case not possible as it requires access to the FOM. The extrapolation in time of the gap radiation data had only little success, but it might be worth trying for an example where the grid data is available as this could lead to a better generalisation.

4.2. Outlook

The modular structure of the code enables the evaluation of the SROM in parallel, as it consists out of 1,000 ROMs. This reduces not only the training time even further, but also enables a faster evaluation time in production. Furthermore, it would make sense to write a generic projection-based ROM-factory, so the adjoint problem does not have to be implemented again for every new model by the user. As the tool is quite compute intense, moving from Python to a language like C++ that is more suitable for writing programmes for distributed systems would lead to a huge performance boost on larger multi-core systems which could be used for the optimisation. An other option is to write a C++ back-end for a SROM Python package to keep the user friendly Python interface, but have the power of C++ for the evaluation of the SROM.

Another, way of getting a faster evaluation of the SROM is to take the approach of optimising the NLML one step further. This means after optimising the SROM over the NLML we use the expectation values and the standard deviation values of the optimised SROM to fit a GP which functions on his own. The expectation values function as mean and the standard deviation as noise of the GP. The only thing that needs to be learned is the kernel to actually be able to interpolate in the parameter space and evaluate the GP for different parameter sets. The evaluation of a GP is way faster than the evaluation of the SROM itself and the uncertainties are going to be calculated implicitly.

Furthermore, it is also worth investigate the robustness of the UQ approach covered in this thesis. It would be interesting to look at different sampling algorithms over the parameter space to build the POD, as we have seen, in Section 3.4.1 that sampling plays an important role. It would also be helpful to come up with an indicator which describes how good the confidence interval is, as for now we just looked at the coverage of the FOM but need in addition to plot the interval to check whether it is not unreasonably underconfident. As last outlook, it would help to find a general setting for the IPopt procedure that converges to a reasonable solution for all of the problems.

A. Appendix

A.1. Detailed Solution of Adjoint Problem

A.1.1. SROB Adjoint Problem for Static Example

In this section we show how to solve the adjoint problem regarding a general cost function and the generic static problem covered in Section 3.2. The Lagrangian is formulated as

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{\alpha}, \underbrace{[\mathbf{W}]_i, [\mathbf{H}_s]_i, [\mathbf{Z}]_i, [\mathbf{D}]_i, [\mathbf{A}]_i, [\mathbf{U}]_i, \mathbf{q}_i, \mathbf{O}_i}_{\mathcal{U}}, \\
&\underbrace{[\mathbf{ZW}]_i, [\mathbf{ZH}]_i, [\mathbf{ZZ}]_i, [\mathbf{ZD}]_i, [\mathbf{ZA}]_i, [\mathbf{ZU}]_i, \mathbf{Zq}_i, \mathbf{ZO}_i}_{\mathcal{Z}}) = \\
&J(\mathbf{O}) \\
&-[\mathbf{ZU}]_i \cdot \Big([\mathbf{U}]_i - [\mathbf{G}(\beta)]_i[\sigma]\Big) \\
&-[\mathbf{ZA}]_i \cdot \Big([\mathbf{A}]_i - [\mathbf{U}]_i + [B]\left\{[B]^T[\mathbf{U}]_i\right\}\Big) \\
&-[\mathbf{ZD}]_i \cdot \Big([\mathbf{D}]_i - \left([V]^T[\mathbf{A}]_i + [\mathbf{A}]_i^T[V]\right)/2\Big) \\
&-[\mathbf{ZZ}]_i \cdot \Big([\mathbf{Z}]_i - [\mathbf{A}]_i + [V][\mathbf{D}]_i\Big) \\
&-[\mathbf{ZH}]_i \cdot \Big(\left([I_n] + s^2[\mathbf{Z}]_i^T[\mathbf{Z}]_i\right)[\mathbf{H}_s]_i[\mathbf{H}_s]_i - [I_n]\Big) \\
&-[\mathbf{ZW}]_i \cdot \Big([\mathbf{W}]_i - ([V] + s[\mathbf{Z}]_i)[\mathbf{H}_s]_i\Big) \\
&-\mathbf{Zq}_i^\top \Big([\mathbf{W}]_i^\top[K][\mathbf{W}]_i\mathbf{q}_i - [\mathbf{W}]_i^\top \boldsymbol{f}\Big) \\
&-\mathbf{ZO}_i^\top \Big(\mathbf{O}_i - [\mathbf{W}]_i\mathbf{q}_i\Big),
\end{aligned} \tag{A.1}$$

where ″·″ is the inner Frobenius product and we introduce again the Langrangian multipliers $\mathcal{Z}$ and the helper variables $\mathcal{U}$ for the equality constrains. Now we are interested in calculating the values for $[\mathbf{ZU}]_i$, $[\mathbf{ZW}]_i$ and $[\mathbf{ZH}]_i$, as they are are needed for computing the gradients regarding $\boldsymbol{\alpha}$.

At first we have to compute the derivative regarding all variables of $\mathcal{U}$ and set it to zero

$$\begin{aligned}
[\mathbf{ZW}]_i \cdot [\delta\mathbf{W}]_i + \mathbf{Zq}_i^\top \Big([\mathbf{W}]_i^\top [K][\delta\mathbf{W}]_i\mathbf{q}_i + [\delta\mathbf{W}]_i^\top [K][\mathbf{W}]_i\mathbf{q}_i - [\delta\mathbf{W}]_i^\top \mathbf{f}\Big) & \\
+\mathbf{ZO}_i^\top \Big(-[\delta\mathbf{W}]_i\mathbf{q}_i\Big) &= 0 \quad \forall[\delta\mathbf{W}]_i, \\
[\mathbf{ZH}]_i \cdot \Big(\left([I_n] + s^2[\mathbf{Z}]_i^T[\mathbf{Z}]_i\right)\left([\delta\mathbf{H}_s]_i[\mathbf{H}_s]_i + [\mathbf{H}_s]_i[\delta\mathbf{H}_s]_i\right)\Big) & \\
+[\mathbf{ZW}]_i \cdot \Big(-\left([V] + s[\mathbf{Z}]_i\right)[\delta\mathbf{H}_s]_i\Big) &= 0 \quad \forall[\delta\mathbf{H}_s]_i, \\
[\mathbf{ZZ}]_i \cdot [\delta\mathbf{Z}]_i + [\mathbf{ZH}]_i \cdot \Big(s^2\left([\delta\mathbf{Z}]_i^T[\mathbf{Z}]_i + [\mathbf{Z}]_i^T[\delta\mathbf{Z}]_i\right)[\mathbf{H}_s]_i[\mathbf{H}_s]_i\Big) & \\
+[\mathbf{ZW}]_i \cdot \Big(-s[\delta\mathbf{Z}]_i[\mathbf{H}_s]_i\Big) &= 0 \quad \forall[\delta\mathbf{Z}]_i, \\
[\mathbf{ZD}]_i \cdot [\delta\mathbf{D}]_i + [\mathbf{ZZ}]_i \cdot \Big([V][\delta\mathbf{D}]_i\Big) &= 0 \quad \forall[\delta\mathbf{D}]_i, \\
[\mathbf{ZA}]_i \cdot [\delta\mathbf{A}]_i + [\mathbf{ZD}]_i \cdot \Big(-\left([V]^T[\delta\mathbf{A}]_i + [\delta\mathbf{A}]_i^T[V]\right)/2\Big) + [\mathbf{ZZ}]_i \cdot \Big(-[\delta\mathbf{A}]_i\Big) &= 0 \quad \forall[\delta\mathbf{A}]_i, \\
[\mathbf{ZU}]_i \cdot [\delta\mathbf{U}]_i + [\mathbf{ZA}]_i \cdot \Big(-[\delta\mathbf{U}]_i + [B]\left\{[B]^T[\delta\mathbf{U}]_i\right\}\Big) &= 0 \quad \forall[\delta\mathbf{U}]_i, \\
\mathbf{Zq}_i^\top \Big([\mathbf{W}]_i^\top [K][\mathbf{W}]_i\delta\mathbf{q}_i\Big) + \mathbf{ZO}_i^\top \Big(-[\mathbf{W}]_i\delta\mathbf{q}_i\Big) &= 0 \quad \forall\delta\mathbf{q}_i, \\
-\mathbf{ZO}_i^\top \delta\mathbf{O}_i + \left(\nabla_{\mathbf{O}_i} J\right)^\top \delta\mathbf{O}_i &= 0 \quad \forall\delta\mathbf{O}_i.
\end{aligned}$$

This formulation has to be reformulated in a way that all the variables with a δ are out of the brackets. This leads to the following system

$$\begin{aligned}
[\delta\mathbf{W}]_i \cdot \Big([\mathbf{ZW}]_i + [K]^\top[\mathbf{W}]_i\mathbf{Zqq}_i^\top + [K][\mathbf{W}]_i\mathbf{q}_i\mathbf{Zq}_i^\top - \mathbf{fZq}_i^\top - \mathbf{ZO}_i\mathbf{q}_i^\top\Big) &= 0 \quad \forall[\delta\mathbf{W}]_i, \\
[\delta\mathbf{H}_s]_i \cdot \Big([\mathbf{H}_s]_i^\top \Big([I_n] + s^2[\mathbf{Z}]_i^\top[\mathbf{Z}]_i\Big)[\mathbf{ZH}]_i + \Big([I_n] + s^2[\mathbf{Z}]_i^\top[\mathbf{Z}]_i\Big)[\mathbf{ZH}]_i[\mathbf{H}_s]_i^\top & \\
-\Big([V]^\top + s[\mathbf{Z}]_i^\top\Big)[\mathbf{ZW}]_i\Big) &= 0 \quad \forall[\delta\mathbf{H}_s]_i, \\
[\delta\mathbf{Z}]_i \cdot \Big([\mathbf{ZZ}]_i + s^2[\mathbf{Z}]_i\Big([\mathbf{H}_s]_i[\mathbf{H}_s]_i[\mathbf{ZH}]_i^\top + [\mathbf{ZH}]_i[\mathbf{H}_s]_i^\top[\mathbf{H}_s]_i^\top\Big) - s[\mathbf{ZW}]_i[\mathbf{H}_s]_i^\top\Big) &= 0 \quad \forall[\delta\mathbf{Z}]_i, \\
[\delta\mathbf{D}]_i \cdot \Big([\mathbf{ZD}]_i + [V]^\top[\mathbf{ZZ}]_i\Big) &= 0 \quad \forall[\delta\mathbf{D}]_i, \\
[\delta\mathbf{A}]_i \cdot \left([\mathbf{ZA}]_i - \frac{1}{2}[V]\Big([\mathbf{ZD}]_i + [\mathbf{ZD}]_i^\top\Big) - [\mathbf{ZZ}]_i\right) &= 0 \quad \forall[\delta\mathbf{A}]_i, \\
[\delta\mathbf{U}]_i \cdot \Big([\mathbf{ZU}]_i - [\mathbf{ZA}]_i + [B]\left\{[B]^T[\mathbf{ZA}]_i\right\}\Big) &= 0 \quad \forall[\delta\mathbf{U}]_i, \\
\Big(\mathbf{Zq}_i^\top[\mathbf{W}]_i^\top[K][\mathbf{W}]_i - \mathbf{ZO}_i^\top[\mathbf{W}]_i\Big)\delta\mathbf{q}_i &= 0 \quad \forall\delta\mathbf{q}_i, \\
\Big(-\mathbf{ZO}_i^\top + \left(\nabla_{\mathbf{O}_i} J\right)^\top\Big)\delta\mathbf{O}_i &= 0 \quad \forall\delta\mathbf{O}_i,
\end{aligned}$$

where the the terms in the brackets have to be zero. After this step we already can solve the system for $[\mathbf{ZU}]_i$ and all the other variables of interest are calculated along

$$\nabla_{\mathbf{O}_i} J = \mathbf{ZO}_i, \tag{A.2}$$

$$[\mathbf{W}]_i^\top [K]^\top [\mathbf{W}]_i \mathbf{Zq}_i = [\mathbf{W}]_i^\top \mathbf{ZO}_i, \tag{A.3}$$

$$-[K]^\top [\mathbf{W}]_i \mathbf{Zqq}_i^\top - [K][\mathbf{W}]_i \mathbf{q}_i \mathbf{Zq}_i^\top + \mathbf{fZq}_i^\top + \mathbf{ZO}_i \mathbf{q}_i^\top = [\mathbf{ZW}]_i, \tag{A.4}$$

$$\begin{aligned} &[\mathbf{H}_s]_i^\top \left([I_n] + s^2 [\mathbf{Z}]_i^\top [\mathbf{Z}]_i\right) [\mathbf{ZH}]_i \\ &+ \left([I_n] + s^2 [\mathbf{Z}]_i^\top [\mathbf{Z}]_i\right) [\mathbf{ZH}]_i [\mathbf{H}_s]_i^\top = \left([V]^\top + s[\mathbf{Z}]_i^\top\right) [\mathbf{ZW}]_i, \end{aligned} \tag{A.5}$$

$$-s^2 [\mathbf{Z}]_i \left([\mathbf{H}_s]_i [\mathbf{H}_s]_i [\mathbf{ZH}]_i^\top + [\mathbf{ZH}]_i [\mathbf{H}_s]_i^\top [\mathbf{H}_s]_i^\top\right) + s[\mathbf{ZW}]_i [\mathbf{H}_s]_i^\top = [\mathbf{ZZ}]_i, \tag{A.6}$$

$$-[V]^\top [\mathbf{ZZ}]_i = [\mathbf{ZD}]_i, \tag{A.7}$$

$$\frac{1}{2}[V]\left([\mathbf{ZD}]_i + [\mathbf{ZD}]_i^\top\right) + [\mathbf{ZZ}]_i = [\mathbf{ZA}]_i, \tag{A.8}$$

$$[\mathbf{ZA}]_i - [B]\left\{[B]^T [\mathbf{ZA}]_i\right\} = [\mathbf{ZU}]_i, \tag{A.9}$$

where calculating $[\mathbf{ZH}]_i$ is the only none straight forward computation. Here we have to use that $[\mathbf{H}_s] = [\mathbf{H}_s]^\top$

$$[\mathbf{H}_s]_i [\mathbf{ZH}]_i + [\mathbf{ZH}]_i [\mathbf{H}_s]_i^\top = [\mathbf{H}_s]_i^\top [\mathbf{H}_s]_i^\top \left([V]^\top + s[\mathbf{Z}]_i^\top\right) [\mathbf{ZW}]_i, \tag{A.10}$$

which can be solve with the continuous Lyapunov function from Scipy.

A.1.2. Cost Function Derivatives

In this section we define the derivatives of the used cost functions in this thesis, as they are needed to compute the adjoint problem.

Soize Nonlinear LS Method

The cost function defined in Equation [2.35] is given as

$$J(\mathbf{O}) = w_J J_{\text{mean}}(\mathbf{O}) + (1 - w_J)\, J_{\text{std}}(\mathbf{O}). \tag{A.11}$$

The components $J_{\text{mean}}(\boldsymbol{\alpha})$ and $J_{\text{std}}(\boldsymbol{\alpha})$ can be written as

$$J_{\text{mean}}(\mathbf{O}) = \frac{1}{\|\mathbf{o}^{\text{ref}}\|^2} \left(\mathbf{o}^{\text{ref}} - \frac{1}{\nu_s}\sum_i^{\nu_s} \mathbf{O}_i\right)^\top \left(\mathbf{o}^{\text{ref}} - \frac{1}{\nu_s}\sum_i^{\nu_s} \mathbf{O}_i\right), \tag{A.12}$$

$$\begin{aligned} J_{\text{std}}(\mathbf{O}) = \frac{1}{\left\|\mathbf{v}^{(\text{ref},n)}\right\|^2} &\left(\mathbf{v}^{(\text{ref},n)} - \Big(\frac{1}{\nu_s}\sum_i^{\nu_s} \mathbf{O}_i^2 - (\frac{1}{\nu_s}\sum_i^{\nu_s} \mathbf{O}_i)^2\Big)^{\frac{1}{2}}\right)^\top \\ &\left(\mathbf{v}^{(\text{ref},n)} - \Big(\frac{1}{\nu_s}\sum_i^{\nu_s} \mathbf{O}_i^2 - (\frac{1}{\nu_s}\sum_i^{\nu_s} \mathbf{O}_i)^2\Big)^{\frac{1}{2}}\right), \end{aligned} \tag{A.13}$$

which allows us to take the derivative regarding $\mathbf{O}_i$

$$\partial_{\mathbf{O}_i} J_{\text{mean}}(\mathbf{O})(\delta\mathbf{O}_i) = \Big(\nabla_{\mathbf{O}_i} J_{\text{mean}}\Big)^\top \delta\mathbf{O}_i := \frac{-2}{\nu_s \left\|\mathbf{o}^{\text{ref}}\right\|^2}\Big(\mathbf{o}^{\text{ref}} - \frac{1}{\nu_s}\sum_i^{\nu_s} \mathbf{O}_i\Big)^\top \Big(\delta\mathbf{O}_i\Big), \quad \text{(A.14)}$$

$$\begin{aligned}\partial_{\mathbf{O}_i} J_{\text{std}}(\mathbf{O})(\delta\mathbf{O}_i) =& \Big(\nabla_{\mathbf{O}_i} J_{\text{std}}\Big)^\top \delta\mathbf{O}_i := \\ & \frac{-2}{\left\|\mathbf{v}^{(\text{ref},n)}\right\|^2}\Bigg(\Bigg(\mathbf{v}^{(\text{ref},n)} - \Big(\frac{1}{\nu_s}\sum_i^{\nu_s}\mathbf{O}_i^2 - \frac{1}{\nu_s^2}\big(\sum_i^{\nu_s}\mathbf{O}_i\big)^2\Big)^{\frac{1}{2}}\Bigg) \\ & \circ \Big(\frac{1}{\nu_s}\sum_i^{\nu_s}\mathbf{O}_i^2 - \frac{1}{\nu_s^2}\big(\sum_i^{\nu_s}\mathbf{O}_i\big)^2\Big)^{-\frac{1}{2}} \circ \Big(\frac{1}{\nu_s}\mathbf{O}_i - \frac{1}{\nu_s^2}\sum_i^{\nu_s}\mathbf{O}_i\Big)\Bigg)^\top \delta\mathbf{O}_i.\end{aligned} \quad \text{(A.15)}$$

The total derivative is then given as

$$\nabla_{\mathbf{O}_i} J(\mathbf{O})\delta\mathbf{O}_i = \Bigg(w_J\Big(\nabla_{\mathbf{O}_i} J_{\text{mean}}\Big)^\top + (1-w_J)\,\Big(\nabla_{\mathbf{O}_i} J_{\text{std}}\Big)^\top\Bigg)\delta\mathbf{O}_i. \quad \text{(A.16)}$$

Negative Marginal Log Likelihood

This Negative Marginal Log Likelihood is given as

$$\mathcal{NLML}(\mathbf{O}) = \frac{N}{2}\log(2\pi) + \frac{1}{2}\sum_l^N \frac{\mathbf{y}_l^2}{\sigma_l^2} + \log(\sigma_l^2), \quad \text{(A.17)}$$

where $\mathbf{y} = \mathbf{o}^{\text{ref}} - \sum_{i=1}^{\nu_s}\mathbf{O}_i$. The derivative regarding $\mathbf{O}_i$ is

$$\begin{aligned}&\partial_{\mathbf{O}_{ik}}\mathcal{NLML}(\mathbf{O})(\delta\mathbf{O}_{ik}) = \\ &\frac{1}{\nu_s}\sum_k^N \frac{1}{\sigma_k^2}\Big(-(\mathbf{o}_k^{\text{ref}} - \frac{1}{\nu_s}\sum_j^{\nu_s}\mathbf{O}_{jk})(1 + \frac{(\mathbf{o}_k^{\text{ref}} - \frac{1}{\nu_s}\sum_j^{\nu_s}\mathbf{O}_{jk})(\mathbf{O}_{ik} - \frac{1}{\nu_s}\sum_j^{\nu_s}\mathbf{O}_{jk})}{\sigma_k^2}) \\ &\quad + \mathbf{O}_{ik} - \frac{1}{\nu_s}\sum_j^{\nu_s}\mathbf{O}_{jk}\Big)(\delta\mathbf{O}_{ik}).\end{aligned} \quad \text{(A.18)}$$

However, the derivative was calculated in the following form

$$\begin{aligned}&\partial_{\mathbf{O}_{ik}}\mathcal{NLML}(\mathbf{O})(\delta\mathbf{O}_{ik}) = \\ &\frac{1}{\nu_s}\sum_k^N \Big(-\frac{\mathbf{o}_k^{\text{ref}} - \frac{1}{\nu_s}\sum_j^{\nu_s}\mathbf{O}_{jk}}{\sigma_k^2}(1 + \frac{(\mathbf{o}_k^{\text{ref}} - \frac{1}{\nu_s}\sum_j^{\nu_s}\mathbf{O}_{jk})}{\sigma_k^2}(\mathbf{O}_{ik} - \frac{1}{\nu_s}\sum_j^{\nu_s}\mathbf{O}_{jk})) \\ &\quad + \frac{1}{\sigma_k^2}(\mathbf{O}_{ik} - \frac{1}{\nu_s}\sum_j^{\nu_s}\mathbf{O}_{jk})\Big)(\delta\mathbf{O}_{ik}).\end{aligned} \quad \text{(A.19)}$$

A.1.3. Heun Time Stepping

The Heun time stepping scheme is used in this thesis for OI problems. It is a second order scheme, which is sufficient as the operators are inferred based on a second order gradient computation. The time stepping is done via

$$\mathbf{q}_0 = [\widehat{\mathbf{Q}}]_0, \tag{A.20}$$

$$\dot{\mathbf{q}}_{j-1} = f_{model}(\mathbf{q}_{j-1}, t), \tag{A.21}$$

$$\widehat{\mathbf{q}}_j = \mathbf{q}_{j-1} + dt \cdot \dot{\mathbf{q}}_{j-1}, \tag{A.22}$$

$$\dot{\widehat{\mathbf{q}}}_j = f_{model}(\widehat{\mathbf{q}}_j, t + dt), \tag{A.23}$$

$$\mathbf{q}_j = \mathbf{q}_{j-1} + \frac{dt}{2}\left(\dot{\mathbf{q}}_{j-1} + \dot{\widehat{\mathbf{q}}}_j\right), \tag{A.24}$$

where $[\widehat{\mathbf{Q}}]_{0i}$ is the initial point, and $f_{model}(\mathbf{q}_{k,i}, t)$ is the evaluation of the model for state values $\mathbf{q}_{k,i}$ and time t. The time step is done based on the average of the model evaluation at the current position and the position after an Euler step.

Its contribution to the adjoint problem is

$$\begin{aligned}
-\mathbf{Zq}_0^\top(\mathbf{q}_0 - [\widehat{\mathbf{Q}}]_0) \\
-\mathbf{Z\dot{q}}_j^\top(\dot{\mathbf{q}}_j - f_{model}(\mathbf{q}_j)) \\
-\mathbf{Z\widehat{q}}_j^\top(\widehat{\mathbf{q}}_j - (\mathbf{q}_{j-1} + dt \cdot \dot{\mathbf{q}}_{j-1})) \\
-\mathbf{Z\dot{\widehat{q}}}_j^\top(\dot{\widehat{\mathbf{q}}}_j - f_{model}(\widehat{\mathbf{q}}_j)) \\
-\mathbf{Zq}_j^\top(\mathbf{q}_j - (\mathbf{q}_{j-1} + \frac{dt}{2}\left(\dot{\mathbf{q}}_{j-1} + \dot{\widehat{\mathbf{q}}}_j\right))),
\end{aligned} \tag{A.25}$$

with the Lagrangian factors $\mathbf{Zq}$, $\mathbf{Z\dot{q}}$, $\mathbf{Z\widehat{q}}$ and $\mathbf{Z\dot{\widehat{q}}}$. The partial derivatives are

$$\begin{aligned}
\mathbf{Z\dot{q}}_j^\top\delta\dot{\mathbf{q}} + \mathbf{Z\widehat{q}}_{j+1}^\top(-dt\delta\dot{\mathbf{q}}_j) + \mathbf{Zq}_{j+1}^\top(-\frac{dt}{2}\delta\dot{\mathbf{q}}_j) &= 0 \quad \forall\delta\dot{\mathbf{q}}, \\
\mathbf{Z\widehat{q}}_j^\top\delta\widehat{\mathbf{q}}_j + \mathbf{Z\dot{\widehat{q}}}_j^\top(-f_{model}(\delta\widehat{\mathbf{q}}_j)) &= 0 \quad \forall\delta\widehat{\mathbf{q}}, \\
\mathbf{Z\dot{\widehat{q}}}_j^\top\delta\dot{\widehat{\mathbf{q}}}_j + \mathbf{Zq}_j^\top(-\frac{dt}{2}\delta\dot{\widehat{\mathbf{q}}}_j) &= 0 \quad \forall\delta\dot{\widehat{\mathbf{q}}}, \\
\mathbf{Zq}_0^\top\delta\mathbf{q}_0 + \mathbf{Z\dot{q}}_0^\top(-f_{model}(\delta\mathbf{q}_0)) + \mathbf{Z\widehat{q}}_1^\top(-\delta\mathbf{q}_0) & \\
+\mathbf{Zq}_1^\top(-\delta\mathbf{q}_0) + \mathbf{ZO}_0^\top\Big(-[\mathbf{W}]\delta\mathbf{q}_0\Big) &= 0 \quad \forall\delta\mathbf{q}_0, \\
\mathbf{Zq}_j^\top\delta\mathbf{q}_j + \mathbf{Z\dot{q}}_j^\top(-f_{model}(\delta\mathbf{q}_j)) + \mathbf{Z\widehat{q}}_{j+1}^\top(-\delta\mathbf{q}_j) & \\
+\mathbf{Zq}_{j+1}^\top(-\delta\mathbf{q}_j) + \mathbf{ZO}_j^\top\Big(-[\mathbf{W}]\delta\mathbf{q}_j\Big) &= 0 \quad \forall\delta\mathbf{q}_j.
\end{aligned}$$

Refactoring leads to

$$\begin{aligned}
(\mathbf{Z}\dot{\mathbf{q}}_j^\top - dt\mathbf{Z}\widehat{\mathbf{q}}_{j+1}^\top - \frac{dt}{2}\mathbf{Z}\mathbf{q}_{j+1}^\top)\delta\dot{\mathbf{q}}_j = 0 \quad \forall\delta\dot{\mathbf{q}}, \\
(\mathbf{Z}\widehat{\mathbf{q}}_j^\top - f_{mAdj}(\mathbf{Z}\dot{\widehat{\mathbf{q}}}_j, \widehat{\mathbf{q}}_j))\delta\widehat{\mathbf{q}}_j = 0 \quad \forall\delta\widehat{\mathbf{q}}, \\
(\mathbf{Z}\dot{\widehat{\mathbf{q}}}_j^\top - \frac{dt}{2}\mathbf{Z}\mathbf{q}_{ji}^\top)\delta\dot{\widehat{\mathbf{q}}}_{ji} = 0 \quad \forall\delta\dot{\widehat{\mathbf{q}}}_i, \\
(\mathbf{Z}\mathbf{q}_{ji}^\top - f_{mAdj}(\mathbf{Z}\dot{\mathbf{q}}_{ji}, \mathbf{q}_{ji}) - \mathbf{Z}\widehat{\mathbf{q}}_{j+1,i}^\top - \mathbf{Z}\mathbf{q}_{j+1,i}^\top - \mathbf{Z}\mathbf{O}_{ji}^\top[\mathbf{W}]_i)\delta\mathbf{q}_{ji} = 0 \quad \forall\delta\mathbf{q}_{ji},
\end{aligned}$$

with f_{mAdj} as the solution of the adjoint problem regarding the model used for the time derivative computation. This function is basically the derivative regarding $\mathbf{q}$, but with the incorporation of the Lagrangian multiplier $\mathbf{Z}\mathbf{q}$. The end result is then

$$[\mathbf{W}]_i^\top \mathbf{Z}\mathbf{O}_{ki} = \mathbf{Z}\mathbf{q}_{ki}, \tag{A.26}$$

$$\frac{dt}{2}\mathbf{Z}\mathbf{q}_{ji} = \mathbf{Z}\dot{\widehat{\mathbf{q}}}_{ji}, \quad j = 1, \dots k, \tag{A.27}$$

$$f_{mAdj}(\mathbf{Z}\dot{\widehat{\mathbf{q}}}_{ji}, \widehat{\mathbf{q}}_{ji})^\top = \mathbf{Z}\widehat{\mathbf{q}}_{ji}, \quad j = 1, \dots k, \tag{A.28}$$

$$dt\mathbf{Z}\widehat{\mathbf{q}}_{j+1,i} + \frac{dt}{2}\mathbf{Z}\mathbf{q}_{j+1,i} = \mathbf{Z}\dot{\mathbf{q}}_{ji}, \quad j = 0, \dots k-1, \tag{A.29}$$

$$f_{mAdj}(\mathbf{Z}\dot{\mathbf{q}}_{ji}, \mathbf{q}_{ji})^\top + \mathbf{Z}\widehat{\mathbf{q}}_{j+1,i} + \mathbf{Z}\mathbf{q}_{j+1,i} + [\mathbf{W}]_i^\top\mathbf{Z}\mathbf{O}_{ji} = \mathbf{Z}\mathbf{q}_{ji}, \quad j = 0, \dots k-1. \tag{A.30}$$

A.1.4. Operator Inference

The contribution of the OI model is slightly more difficult, as the SROB is used to infer the operators and it is used to reduce the initial state, which gets afterwards integrated with the inferred model. Therefore, the model contributes twice into the adjoint problem, with its own Lagrangian multipliers and with its model derivative regarding $\mathbf{q}_i$.

We remember, that we infer the operators via

$$\min_{[OP]} \left\| [D][OP]^\top - [R]^\top \right\|_F^2, \tag{A.31}$$

where $\|\cdot\|_F$ is the Frobenius norm and

$$\begin{aligned}
[OP] &= \begin{bmatrix} \mathbf{c} & [A] & [H] & [B] \end{bmatrix} \in \mathbb{M}_{n,d(n,m)}, \quad \text{(unknown operators)}, \\
[D] &= \begin{bmatrix} \mathbf{1}_k & [Q]^\top & ([Q]^\top \otimes [Q]^\top) & [U]^\top \end{bmatrix} \in \mathbb{M}_{k,d(n,m)}, \quad \text{(known data)}, \\
[R] &= \begin{bmatrix} \dot{\mathbf{q}}_0 & \dot{\mathbf{q}}_1 & \cdots & \dot{\mathbf{q}}_{k-1} \end{bmatrix} \in \mathbb{M}_{n,k}, \quad \text{(time derivatives)}, \\
[U] &= \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \cdots & \mathbf{u}_{k-1} \end{bmatrix} \in \mathbb{M}_{m,k}, \quad \text{(inputs)},
\end{aligned}$$

where $d(n,m) = 1 + n + \frac{1}{2}n(n+1) + m$ and the time derivatives are approximated via a central difference scheme. The solution of the minimisation problem has to be rewritten into an explicit form to add it to the adjoint problem. The transposed solution to the linear least squares minimisation is given as

$$[\mathbf{OP}]_i = [\widehat{\mathbf{R}}]_i[\widehat{\mathbf{D}}]_i([\widehat{\mathbf{D}}]_i^\top[\widehat{\mathbf{D}}]_i)^{-1}, \tag{A.32}$$

which can now be added to the adjoint problem. The whole contribution to the Lagrangian is given as

$$\begin{aligned}
-[\mathbf{Z}\widehat{\mathbf{D}}]_i \cdot ([\widehat{\mathbf{D}}]_i - ([I_{k+1}]_{0,k}^{(k+1+d(n,m))\times k+1}(\mathbb{1}_{k+1}[I_1]_0^{1\times d(n,m)} \\
+[\widehat{\mathbf{Q}}]_i^\top[I_n]_{1,n}^{n\times d(n,m)} + ([\widehat{\mathbf{Q}}]_i \otimes [\widehat{\mathbf{Q}}]_i)^\top[I_{n(n+1)/2}]_{1+n,d(n,m)-m-1}^{n(n+1)/2\times d(n,m)} \\
+[\widehat{\mathbf{U}}]_t^\top[I_m]_{d(n,m)-m,d(n,m)-1}^{m\times d(n,m)}) + [I_{d(n,m)}]_{k+1,k+1+d(n,m)-1}^{(k+1+d(n,m))\times d(n,m)}[\Gamma])) \\
-[\mathbf{ZR}]_{0i}^\top([\mathbf{R}]_{0i} - \frac{[\widehat{\mathbf{Q}}]_{1,i} - [\widehat{\mathbf{Q}}]_{0,i}}{dt}) \\
-[\mathbf{ZR}]_{ji}^\top([\mathbf{R}]_{ji} - \frac{[\widehat{\mathbf{Q}}]_{j+1,i} - [\widehat{\mathbf{Q}}]_{j-1,i}}{2dt}) \\
-[\mathbf{ZR}]_{ki}^\top([\mathbf{R}]_{k,i} - \frac{[\widehat{\mathbf{Q}}]_{k,i} - [\widehat{\mathbf{Q}}]_{k-1,i}}{dt}) \\
-[\mathbf{Z}\widehat{\mathbf{R}}]_i \cdot ([\widehat{\mathbf{R}}]_i - [\mathbf{R}]_i[I_n]_{0,n-1}^{(k+1)\times(k+1+d(n,m))}) \\
-[\mathbf{ZOP}]_i \cdot ([\mathbf{OP}]_i([\widehat{\mathbf{D}}]_i^\top[\widehat{\mathbf{D}}]_i) - [\widehat{\mathbf{R}}]_i[\widehat{\mathbf{D}}]_i) \\
-\mathbf{Z}\widehat{\mathbf{C}}_i^k(\widehat{\mathbf{C}}_i - [\mathbf{OP}]_i[I_1]_0^{d(n,m),1}) \\
-[\mathbf{Z}\widehat{\mathbf{A}}]_i \cdot ([\widehat{\mathbf{A}}]_i - [\mathbf{OP}]_i[I_n]_{1,n}^{d(n,m)\times n}) \\
-[\mathbf{Z}\widehat{\mathbf{H}}]_i \cdot ([\widehat{\mathbf{H}}]_i - [\mathbf{OP}]_i[I_{n(n+1)/2}]_{1+n,d(n,m)-m-1}^{d(n,m)\times n(n+1)/2}) \\
-[\mathbf{Z}\widehat{\mathbf{B}}]_i \cdot ([\widehat{\mathbf{B}}]_i - ([\mathbf{OP}]_i[I_m]_{d(n,m)-m,d(n,m)-1}^{d(n,m)\times m})),
\end{aligned} \tag{A.33}$$

which represents the assembly of all the matrices needed for the minimisation problem and afterwards, the selection of the operator out of the inferred matrix. The indexing of the identity matrices might be a little confusing, but they are just there for assembling matrices or selecting certain parts from a matrix. The matrix can be seen as a matrix of zeros of the size given in the exponent, with an identity matrix of the size given inside the brackets inside of this matrix at the position given in the lower right.

Solving this problem is quite tedious and error prone. The interested reader can try to solve the problem on his own or send me an email for a detailed calculation. The end result with Heun time stepping is given as

$$\sum_{j=0}^{k-1} \mathbf{Z}\dot{\mathbf{q}}_{ji}[\widehat{\mathbf{U}}]_{ji}^{\top} + \sum_{j=1}^{k} \mathbf{Z}\dot{\widehat{\mathbf{q}}}_{ji}[\widehat{\mathbf{U}}]_{ji}^{\top} = [\mathbf{Z}\widehat{\mathbf{B}}]_i, \tag{A.34}$$

$$\sum_{j=0}^{k-1} \mathbf{Z}\dot{\mathbf{q}}_{ji}(\mathbf{q}_{ji} \otimes \mathbf{q}_{ji})^{\top} + \sum_{j=1}^{k} \mathbf{Z}\dot{\widehat{\mathbf{q}}}_{ji}(\widehat{\mathbf{q}}_{ji} \otimes \widehat{\mathbf{q}}_{ji})^{\top} = [\mathbf{Z}\widehat{\mathbf{H}}]_i, \tag{A.35}$$

$$\sum_{j=0}^{k-1} \mathbf{Z}\dot{\mathbf{q}}_{ji}\mathbf{q}_{ji}^{\top} + \sum_{j=1}^{k} \mathbf{Z}\dot{\widehat{\mathbf{q}}}_{ji}\widehat{\mathbf{q}}_{ji}^{\top} = [\mathbf{Z}\widehat{\mathbf{A}}]_i, \tag{A.36}$$

$$\sum_{j=0}^{k-1} \mathbf{Z}\dot{\mathbf{q}}_{ji} + \sum_{j=1}^{k} \mathbf{Z}\dot{\widehat{\mathbf{q}}}_{ji} = \mathbf{Z}\widehat{\mathbf{C}}_i, \tag{A.37}$$

$$\begin{aligned}[I_1]_0^{d(n,m)\times 1}\mathbf{Z}\widehat{\mathbf{C}}_i^{\top} + [I_n]_{1,n}^{d(n,m)\times n}[\mathbf{Z}\widehat{\mathbf{A}}]_i^{\top} + [I_{n(n+1)/2}]_{1+n,d(n,m)-m-1}^{d(n,m)\times n(n+1)/2}[\mathbf{Z}\widehat{\mathbf{H}}]_i^{\top} \\ +[I_m]_{d(n,m)-m,d(n,m)-1}^{d(n,m)\times m}[\mathbf{Z}\widehat{\mathbf{B}}]^{\top} = ([\widehat{\mathbf{D}}]_i^{\top}[\widehat{\mathbf{D}}]_i)[\mathbf{ZOP}]_i^{\top},\end{aligned} \tag{A.38}$$

$$[\mathbf{ZOP}]_i[\widehat{\mathbf{D}}]_i^{\top} = [\mathbf{Z}\widehat{\mathbf{R}}]_i, \tag{A.39}$$

$$[\mathbf{Z}\widehat{\mathbf{R}}]_i[I_n]_{0,n-1}^{(k+1)\times(k+1+d(n,m))\,\top} = [\mathbf{ZR}]_i, \tag{A.40}$$

$$-[\widehat{\mathbf{D}}]_i([\mathbf{ZOP}]_i^{\top}[\mathbf{OP}]_i + [\mathbf{OP}]_i^{\top}[\mathbf{ZOP}]_i) + [\widehat{\mathbf{R}}]_i^{\top}[\mathbf{ZOP}]_i = [\mathbf{Z}\widehat{\mathbf{D}}]_i, \tag{A.41}$$

$$\begin{aligned}[I_n]_{1,n}^{n\times d(n,m)}[\mathbf{Z}\widehat{\mathbf{D}}]_i^{\top} - f_{\otimes Adj}([\mathbf{Z}\widehat{\mathbf{D}}]_i^{\top}, [I_{n(n+1)/2}]_{1+n,d(n,m)-m-1}^{n(n+1)/2\times d(n,m)\,\top}) \\ +\frac{1}{dt}[\mathbf{ZR}]_i(\hat{\mathbf{e}}_0(\hat{\mathbf{e}}_1 - \hat{\mathbf{e}}_0)^{\top} + \frac{1}{2}\sum_{j=1}^{k-1}\hat{\mathbf{e}}_j(\hat{\mathbf{e}}_{j+1} - \hat{\mathbf{e}}_{j-1})^{\top} + \hat{\mathbf{e}}_k(\hat{\mathbf{e}}_k - \hat{\mathbf{e}}_{k-1})^{\top}) \\ +\mathbf{Z}\mathbf{q}_{0i}\hat{\mathbf{e}}_0^{\top} = [\mathbf{Z}\widehat{\mathbf{Q}}]_i,\end{aligned} \tag{A.42}$$

where we can see that the Lagrangian multipliers of the explicit step and the Heun step (every time the model is evaluated) contribute to the Lagrangian multipliers of the operators of the OI model. Furthermore, we have to state that $\hat{\mathbf{e}}_j$ are unit vectors with a one at position j and $f_{\otimes Adj}$ is the solution for the adjoint problem of the $\otimes$ operator, which is just the Lagrangian multiplier times the derivative. The operator can be written as

$$\mathbf{q} \otimes \mathbf{q} = \sum_j [I_{j(j+1)/2+1}]_{j,j(j+1)/2}^{n(n+1)/2\times(j+1)}(\hat{\mathbf{e}}_j^{\top}\mathbf{q})[I_{j+1}]_{0,j}^{(j+1)\times n}\mathbf{q}, \tag{A.43}$$

where the corresponding derivative is

$$(\mathbf{q}\otimes\mathbf{q})'\delta\mathbf{q}_k = \left(\sum_j [I_{j(j+1)/2+1}]_{j,j(j+1)/2}^{n(n+1)/2\times(j+1)}[I_{j+1}]_{0,j}^{(j+1)\times n}(\delta_{jk}\mathbf{q} + \mathbf{q}_j\hat{\mathbf{e}}_j)\right)\delta\mathbf{q}_k. \tag{A.44}$$

A.1.5. Lifting for Gap Radiation

This section describes how to include lifting into the adjoint problem. The lifting procedure described in Section 3.4.2 can be written as

$$[\mathbf{L}] = \frac{1}{s_L}\left([I_n]^{3n\times n}_{0,n-1}[\widehat{\mathbf{Q}}] + [I_n]^{3n\times n}_{n,2n-1}[\widehat{\mathbf{Q}}]^2 + [I_n]^{3n\times n}_{2n,3n-1}[\widehat{\mathbf{Q}}]^3\right), \tag{A.45}$$

where s_L is the lifting scaling factor and $[\widehat{\mathbf{Q}}]$ is the snapshot matrix from a system of interest. However, as already stated in Section 3.4.2, the lifting procedure does not have to be considered in the adjoint method, but the back transformation

$$\mathbf{O} = s_L[I_N]^{N\times 3N}_{0,N-1}[\mathbf{W}]\mathbf{q}, \tag{A.46}$$

which is the final step to get the result from the ROM. After redoing the calculations of Appendix A.1.1 we get an additional contribution to the Lagrange variable of the reduced order state and an additional contribution to the Lagrange variable of each sample of the ROB

$$s_L[\mathbf{W}]_i^\top [I_N]^{N\times 3N}_{0,N-1}{}^\top \mathbf{ZO}_{ji} = \mathbf{Zq}_{ji}, \tag{A.47}$$

$$[\mathbf{L}][\mathbf{Z}\widehat{\mathbf{Q}}]_i^\top + s_L\sum_j^k [I_N]^{N\times 3N}_{0,N-1}{}^\top \mathbf{ZO}_{ji}\mathbf{q}_{ji}^\top = [\mathbf{ZW}]_i. \tag{A.48}$$

For this specific example the contribution is just scaling $\mathbf{ZO}$ with s_L and expanding each vector of $\mathbf{ZO}$ by $2N$ zeros.

A.2. Basic SROB Optimisation Algorithm

The basic algorithm introduced from Soize and Farhat for the non-convex optimisation problem with constrain, defined by Equation [2.42] , is divided into four stages and elaborated below [36]. The initial values $\boldsymbol{\alpha}^{(0)}$ are given by $(s_0, \beta_0, [I_n])$.

- Stage 1. The first stage is for getting close to a solution with only a few parameters, as the computation time of the gradient linearly increases with the number of parameters (FD-gradient). Only the parameters with the most impact get optimised on this stage, s, β, and the diagonal elements of $[\sigma]$. Hence, the number of variables is $2+n$. The for this stage optimal parameters $\boldsymbol{\alpha}^{(1)}$ is determined via the interior-point algorithm with the constraints $\epsilon_0 \leq s \leq 1$, $\beta_d \leq \beta \leq \beta_u$, and $\epsilon_0 \leq [\sigma]_{11}, ..., [\sigma]_{nn} \leq \sigma_u$.
- Stage 2. The second stage is for optimising the rest of the parameters, which means the off diagonal elements of $[\Sigma]$. The number of parameters is $\frac{(n-1)}{2}n$. This is done via solving an unconstrained interior-point optimisation problem with $\boldsymbol{\alpha}^{(1)}$ as initial values, means only zeros on the off diagonal elements, and the optimised parameters $\boldsymbol{\alpha}^{(2)}$.

- Stage 3. This stage solves for the optimal value of β_{opt}. Originally, it was supposed to be found by trial method. However, we solved this problem with a constrained interior-point method with$\beta_d \leq \beta \leq \beta_u$.
- Stage 4. The last stage solve for the optimal value of the other $1+\frac{n}{2}(n+1)$ parameters, s and $[\sigma]$. The optimisation is done via the interior-point optimisation with the same bounds as above. The initial values are taken from the previous optimisation steps.

A.3. Description of Generic Linear Static System

In this appendix, we describe the linear static computational model defined by Equation [3.1] and Equation [3.5] for $N = 1000$ and $N_{CD} = 2$. The description is take from the original paper [36]. Let $x_1, ..., x_N$ be the points in $[0, 1]$ such that $x_j = (j-1)/(N-1)$. Let $\lambda_1, ..., \lambda_N$ be the positive real numbers such that $\lambda_k = 4\pi^2 k^2$, and let $[\lambda]$ be the diagonal matrix in $\mathbb{M}_N^+$ such that $[\lambda]_{kk'}$. Let $[e]$ be the square matrix in $\mathbb{M}_{\mathbb{N}}$ such that $[e]_{jk} = sin(k\pi x_j)$. Let $[\Phi] = [\phi^1....\phi^N]$ be the orthogonal matrix in $\mathbb{M}_N$ obtained by the QR decomposition of matrix $[e]$ (we thus haven $[e] = [\Phi][R]$ with $[\Phi][\Phi]^\top = [\Phi]^\top[\Phi] = [I_N]$ and therefore, $<\phi^k, \phi^{k'}> = \delta_{kk'}$). In addition, for $k = 1, ..., N$, we have $\phi_1^k = [\Phi]_{1k} = 0$ and $\phi_N^k = [\Phi]_{Nk} = 0$. The computational HDM is then generated as follows:

- Matrix $[K]$ in $\mathbb{M}_N^+$ is written as $[K] = [\Phi][\lambda][\Phi]^\top$. Consequently Equation [3.1] has a unique solution such that $y1 = y2 = 0$.
- Vector $\mathbf{f}$ in $\mathbb{R}^N$ is written as $\mathbf{f} = \frac{1}{0.27702}(0.1\phi^2 + 0.4\phi^4 + 0.6\phi^8 + 2.5(\phi^{29} + \phi^{30} + \phi^{31}))$ and consequently, $\mathbf{f}_1 = \mathbf{f}_N = 0$.
- Let $[b] = [0_{N,N_{CD}}]$ be the zero matrix in $\mathbb{M}_{N,N_{CD}}$ except $[b]_{11} = [b]_{N,N_{CD}} = 1$. Matrix $[B]$ in $\mathbb{M}_{N,N_{CD}}$ such that $[B]^\top[B] = [I_{N_{CD}}]$ is written as the orthonormal basis for the range of $[b]$.

A.4. Description of Extracting All Needed Matrices from a PyMOR Model for the SROB

In the following we will briefly describe how the coordinates of the grid nodes of the PyMOR discretisation, and the boundary matrix can be extracted from the "data"-object that comes with the FOM. The code is shown in Listing A.1. In line 1 we instantiate a class describing the analytical problem. Afterwards, we use a predefined discretiser to get a discretisation using the finite element method, which gives us the FOM and an additional "data"-object holding information about the grid and the boundaries. After extracting the grid dimension and the dimension of the state space of the FOM, we extract directly the node coordinates of the grid. This can be done by the "centers" function, which takes the codimension as input. The codimension of a point is always the dimension of the grid.

Extracting the boundary matrix is not as straight forward, as the indices of the boundary nodes are stored in sets according to each dimension of the grid. Therefore, in line 11 to 15 we loop over all dimension and add the indices to a unique set. The number of indices in the set is then the number of Dirichlet boundary conditions. From line 17 to 21 we loop over all boundary conditions and set a 1 at the node index. To get the orthonormal boundary matrix we then use the ”orth”-function from scipy.linalg.

```
1  problem = thermal_block_problem((3, 3))
2  fom, data = discretize_stationary_cg(problem, diameter=1 / 100)
3  nd = data['grid'].domain.ndim
4  N = fom.solution_space.dim
5
6  # get node coordinates (codimension=2)
7  nodeCoordinates = data['grid'].centers(nd)
8
9  # extract boundary matrix
10 # init boundary_set with the boundary nodes in the first grid dimension
11 boundary_set = {*data['boundary_info'].dirichlet_boundaries(1)}
12 for dim in range(2, nd + 1):
13     boundary_set = boundary_set.union(
14         {*data['boundary_info'].dirichlet_boundaries(dim)}
15     )
16 Nc = len(boundary_set)
17 B = numpy.zeros((N, Nc))
18 boundaryIndx = 0
19 for rowindx in boundary_set:
20     B[rowindx, boundaryIndx] = 1
21     boundaryIndx += 1
22 BoundaryMatrix = scipy.linalg.orth(B)
```

Source Code A.1.: Extraction of SROB input Matrices

Bibliography

[1] M Necati Ã-zisik, M Necati Özışık, and M Necati Özısık. *Heat conduction*. John Wiley & Sons, 1993.

[2] John David Anderson and J Wendt. *Computational fluid dynamics*, volume 206. Springer, 1995.

[3] Suwardi Annas, Muh Isbar Pratama, Muh Rifandi, Wahidah Sanusi, and Syafruddin Side. Stability analysis and numerical simulation of seir model for pandemic covid-19 spread in indonesia. *Chaos, Solitons & Fractals*, 139:110072, 2020.

[4] Henry P Bakewell Jr and John L Lumley. Viscous sublayer and adjacent wall region in turbulent pipe flow. *The Physics of Fluids*, 10(9):1880–1889, 1967.

[5] James R Barber. *Elasticity*. Springer, 2002.

[6] Gal Berkooz, Philip Holmes, and John L Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual review of fluid mechanics*, 25(1):539–575, 1993.

[7] Richard H Byrd, Guanghui Liu, and Jorge Nocedal. On the local behavior of an interior point method for nonlinear programming. *Numerical analysis*, 1997:37–56, 1997.

[8] Wolfgang Demtröder. *Experimentalphysik*, volume 3. Springer, 1998.

[9] Wolfgang Demtröder. *Experimentalphysik 3: Atome, Moleküle und Festkörper*. Springer-Verlag, 2016.

[10] Charbel Farhat, Radek Tezaur, Todd Chapman, Philip Avery, and Christian Soize. Feasible probabilistic learning method for model-form uncertainty quantification in vibration analysis. *AIAA Journal*, 57(11):4978–4991, 2019.

[11] T Garel and H Orland. Mean-field model for protein folding. *EPL (Europhysics Letters)*, 6(4):307, 1988.

[12] Glen D Granzow. A tutorial on adjoint methods and their use for data assimilation in glaciology. *Journal of Glaciology*, 60(221):440–446, 2014.

[13] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[14] Intel. Intel Core i7-7500U Prozessor. https://ark.intel.com/content/www/de/de/ark/products/95451/intel-core-i7-7500u-processor-4m-cache-up-to-3-50-ghz.html, 2021. [Online; accessed 25-June-2021].

[15] Intel. Intel Xeon Prozessor E5-2640. https://ark.intel.com/content/www/de/de/ark/products/64591/intel-xeon-processor-e5-2640-15m-cache-2-50-ghz-7-20-gt-s-intel-qpi.html, 2021. [Online; accessed 25-June-2021].

[16] Ipopt documentation. https://coin-or.github.io/Ipopt/. Accessed: 2021-05-25.

[17] Nishant Kumar and Thomas D Burton. Use of random excitation to develop pod based reduced order models for nonlinear structural dynamics. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 48027, pages 1627–1633, 2007.

[18] Karl Kunisch and Stefan Volkwein. Galerkin proper orthogonal decomposition methods for parabolic problems. *Numerische mathematik*, 90(1):117–148, 2001.

[19] Zhen-dong Luo, Rui-wen Wang, and Jiang Zhu. Finite difference scheme based on proper orthogonal decomposition for the nonstationary navier-stokes equations. *Science in China Series A: Mathematics*, 50(8):1186–1196, 2007.

[20] Zhendong Luo, Jing Chen, Jiang Zhu, Ruiwen Wang, and IM Navon. An optimizing reduced order fds for the tropical pacific ocean reduced gravity model. *International Journal for Numerical Methods in Fluids*, 55(2):143–161, 2007.

[21] Xinyou Ma and William L Hase. Perspective: chemical dynamics simulations of non-statistical reaction dynamics. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2092):20160204, 2017.

[22] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[23] Andrew J Majda and Di Qi. Strategies for reduced-order models for predicting the statistical responses and uncertainty quantification in complex turbulent dynamical systems. *SIAM Review*, 60(3):491–549, 2018.

[24] Shane A McQuarrie, Cheng Huang, and Karen E Willcox. Data-driven reduced-order models via regularised operator inference for a single-injector combustion process. *Journal of the Royal Society of New Zealand*, pages 1–18, 2021.

[25] René Milk, Stephan Rave, and Felix Schindler. pymor–generic algorithms and interfaces for model order reduction. *SIAM Journal on Scientific Computing*, 38(5):S194–S216, 2016.

[26] Roger Ohayon and Christian Soize. *Advanced computational vibroacoustics: reduced-order models and uncertainty quantification*. Cambridge University Press, 2014.

[27] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[28] Benjamin Peherstorfer and Karen Willcox. Data-driven operator inference for nonintrusive projection-based model reduction. *Computer Methods in Applied Mechanics and Engineering*, 306:196–215, 2016.

[29] William H Press, H William, Saul A Teukolsky, William T Vetterling, A Saul, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[30] Elizabeth Qian, Boris Kramer, Benjamin Peherstorfer, and Karen Willcox. Lift & learn: Physics-informed machine learning for large-scale nonlinear dynamical systems. *Physica D: Nonlinear Phenomena*, 406:132401, 2020.

[31] Gianluigi Rozza. Reduced basis methods for stokes equations in domains with non-affine parameter dependence. *Computing and Visualization in Science*, 12(1):23–35, 2009.

[32] B. Sengupta, K.J. Friston, and W.D. Penny. Efficient gradient computation for dynamical models. *NeuroImage*, 98:521–527, 2014.

[33] Mohammad Haris Shamsi, Usman Ali, Eleni Mangina, and James O'Donnell. A framework for uncertainty quantification in building heat demand simulations using reduced-order grey-box energy models. *Applied Energy*, 275:115141, 2020.

[34] Felix Sievers. Thesis' Code. `https://code.siemens.com/felix.sievers.ext/test-project`, 2021.

[35] Lawrence Sirovich. Turbulence and the dynamics of coherent structure. part i, ii, iii. *Quat. Appl. Math.*, 3:583, 1987.

[36] C. Soize and C. Farhat. A nonparametric probabilistic approach for quantifying uncertainties in low-dimensional and high-dimensional nonlinear models. *International Journal for Numerical Methods in Engineering*, 109, 2017.

[37] Christian Soize. *Uncertainty quantification*. Springer, 2017.

[38] Christian Soize and Charbel Farhat. Probabilistic learning for modeling and quantifying model-form uncertainties in nonlinear computational mechanics. *International Journal for Numerical Methods in Engineering*, 117(7):819–843, 2019.

[39] Stefan Volkwein. Model reduction using proper orthogonal decomposition. *Lecture Notes, Institute of Mathematics and Scientific Computing, University of Graz. see http://www. uni-graz. at/imawww/volkwein/POD. pdf*, 1025, 2011.

[40] Costas D Vournas, Emmanuel G Potamianakis, Cédric Moors, and Thierry Van Cutsem. An educational simulation tool for power system control and stability. *IEEE Transactions on Power Systems*, 19(1):48–55, 2004.

[41] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[42] Qinyu Zhuang, Juan Manuel Lorenzi, Hans-Joachim Bungartz, and Dirk Hartmann. Model order reduction based on runge-kutta neural network. *arXiv preprint arXiv:2103.13805*, 2021.