Tor Eivind Palm

Reduced order modelling for linear elastic problems with geometric variation

Master's thesis in Bygg- og miljøteknikk
Supervisor: Kjell Magne Mathisen, Trond Kvamsdal
January 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Structural Engineering

NTNU
Norwegian University of
Science and Technology

Tor Eivind Palm

Reduced order modelling for linear elastic problems with geometric variation

Master's thesis in Bygg- og miljøteknikk
Supervisor: Kjell Magne Mathisen, Trond Kvamsdal
January 2019

Norwegian University of Science and Technology
Faculty of Engineering
Department of Structural Engineering

NTNU
Norwegian University of
Science and Technology

Abstract

The purpose of this thesis is to explore the use of reduced order models for solving parametric partial differential equations. This will be done through constructing and analyzing reduced models within the field of structural mechanics with finite element models as starting point. A special emphasis will be put on the construction and behaviour of reduced models with respect to geometric parameters. The fundamental aspects of reducible problems is given in the introductory chapter, describing motivation for reduction and outline of the thesis.

In order to achieve the thesis goal, a literature review was done. Firstly the theoretical foundation of the Finite element method is presented in Chapter 2, outlining the key concepts of this approach as well as more practical derivations for the case of linear elasticity in structural mechanics. The theory of reduced order models is then presented in Chapter 3, explaining the background for reducibility, key assumptions and the approach of constructing a reduced model through *reduced basis* projection methods.

Based on this theoretical background a numerical study on two example problems was implemented which aimed to explore the over all performance of the reduced models. Explicit expression for the linear elastic case was derived and implemented, and the necessary python code can be found in Appendix B.

For both examples reduced order models where created with overall satisfactory performance, however two key finding from the numerical study emerged. The first being numerical noise in the reduced model that proved to be associated with geometric parametrization. As geometric variation affected the equilibrium of the supported boundary, the reduced model had problems capturing this behaviour. The effect of this turned out to be negligible, but might prove to be of importance if this approach is applied to more complex systems.

The second finding revolves around the assumption of affinity with respect to parameters in order to ensure reducibility, and how this influence the construction of the full order model. As a result of a complicated connection between finite element model and geometric parameters this meant altering the assembly of the finite element system, diverging greatly from the general approach usually found in existing finite element software. This impacts the applications of reduced order models as customized finite element software becomes a necessity.

Sammendrag

Formålet med denne oppgaven er å undersøke bruken av reduserte modeller for å løse parametriske partielle differensialligninger. Dette vil bli gjort ved å konstruere og analysere reduserte modeller innen konstruksjonsmekanikk med elementmetode-modeller som utgangspunkt. Spesiell vekt legges på hvordan de reduserte modellene skal konstrueres og dere oppførsel med hensyn til geometriske parametre. De grunnleggende aspektene for reduserbare problemer er gitt i introduksjonskapittelet, og beskriver motivasjon for reduksjon og en skissering av oppgaven.

For å oppnå målet med oppgaven ble det gjennomført et litteratursøk. Først presenteres det teoretiske grunnlaget for elementmetoden i Kapittel 2, som beskriver nøkkelbegrepene samt gir en mer praktisk innføring i lineær elastisitet innen konstruksjonsmekanikk. Teori om reduserte modeller presenteres i Kapittel 3, som forklarer bakgrunnen for reduksjon, nøkkelforutsetninger og fremgangsmåten for å bygge en redusert modell gjennom projeksjonsmetoder.

Basert på denne teoretiske bakgrunnen ble det gjennomført en numerisk studie på to eksempelproblemer som har som mål å undersøke oppførselen av de reduserte modellene. Eksplisitt uttrykk for det lineær elastiske tilfellet ble utledet og implementert, og den nødvendige Python-koden finnes i tillegg B.

For begge eksemplene ble reduserte modeller konstruert med tilfredsstillende ytelse, men to nøkkelresultater dukket opp. Den første er numerisk støy i den reduserte modellen som viste seg å være forbundet med nettopp geometrisk parametrisering. Geometrisk variasjon påvirket likevekt ved grensebetingelsene, og den reduserte modellen hadde problemer med å fange opp denne oppførselen. Effekten av dette viste seg å være ubetydelig i dette tilfellet, men det kan vise seg å være viktig hvis denne tilnærmingen blir brukt på mer komplekse systemer.

Det andre funnet dreier seg om hvordan antagelsen om affinitet med hensyn til parametere som muliggjør reduksjon, og hvordan dette påvirker konstruksjonen av elementmodellen. På grunn av det kompliserte forholdet mellom elementmodellen og geometriske parametre resulterte dette i å forandre hvordan elementsystemet ble konstruert, noe som divergerte sterkt fra den generelle fremgangsmåten som vanligvis finnes i eksisterende elementmetode-programvare. Dette påvirker bruken av reduserte modeller ettersom spesialtilpasset elementmetode-programvare blir en nødvendighet.

Acknowledgements

I would like to thank my supervisors Kjell Magne Mathisen and Trond Kvamsdal for helping my decide on a semester project and master thesis that proved to be very interesting to study, as well as answering my questions in what has really been an educational year. I would also like to extend a special thanks to co-supervisor Eivind Fonn for taking his time to help me with the more practical parts of the computer implementation which would have been immensely more time consuming without his help.

Sigurd Strømsem also deserves some recognition for help with everything computer related, from Python problems to LaTeXformatting, and at last the boys in PG8A for two and a half great years.

Contents

List of Figures

Abbreviations

- **PDE** - Partial differential equation
- **FEM** -Finite element method
- **FEA** - Finite element analysis
- **dof** - Degree of freedom
- **BC** - Boundary condition
- **ROM** - Reduced order model
- **RSM** - Response surface methodology
- **POD** - Proper orthogonal decomposition
- **SVD** - Singular value decomposition
- **RB** - Reduced basis

1 Introduction

1.1 Numerical methods in civil engineering

In several engineering disciplines partial differential equations (PDEs) are used to describe field problems. As practical problems often are quite complex, analytic solutions are usually not available. This has given rise to the need for numerical approximation. One of the most common approaches is the finite element method (FEM). The reason for the wide spread use of FEM is its many advantages over most other numerical methods [1], and this has lead to FEM being a well known and proven practice. Although there exist other approaches, like the finite differences method, this thesis will only consider the FEM which will be introduced in Chapter 2. The FE-approach is a discretization method which transforms differential equations into first order matrix equations on the form

$$\begin{aligned} &\boldsymbol{A}_{N_h}\boldsymbol{d}_{N_h} = \boldsymbol{f}_{N_h} \\ &\boldsymbol{A}_{N_h} \in \mathbb{R}^{N_h \cdot N_h} \quad and \quad \boldsymbol{d}_{N_h}, \boldsymbol{f}_{N_h} \in \mathbb{R}^{N_h} \end{aligned} \tag{1.1}$$

where $\boldsymbol{A}_{N_h}$ and $\boldsymbol{f}_{N_h}$ defines the system and $\boldsymbol{d}_{N_h}$ are nodal values of the unknown field quantity which the matrix equation is solved for. Also N_h denotes the order of the system, and in turn the computational complexity, and the subfix h refers to how *fine* the discretization is. In general it can be said that accuracy is increased by increasing numeric complexity N_h, making computational cost a key feature for FEM as for any other numerical method. As computing power has increased the last several decades, roughly accordingly to Moore's law [2], the rise of numerical methods has been possible making them viable for a wide range of problems. By being able to handle models of higher numerical order due to better computing power and knowledge of the methods, increasingly complex problems can now be analyzed with better accuracy.

Although technological advancements have been done in the field of computer science, the computational cost of numerical methods remains a key factor analysts have to address. For many situations this revolves around constructing a numerical model which is *good enough*, meaning that a sufficient accuracy is reached for a manageable numerical order N_h. But for some problems, considering model order reduction can be highly beneficial.

1.2 Motivation for reduction

The overall motivation for reduced order models (ROMs) is simply to reduce the numerical order N_h in (1.1) in order to decrease computational cost. Specifically this is done by

creating a new model based on the original, but with a significantly lower order. It can be seen as constructing a new numerical model

$$\begin{aligned} &\boldsymbol{A}_N \boldsymbol{d}_N = \boldsymbol{f}_N \\ &\boldsymbol{A}_N \in \mathbb{R}^{N \cdot N} \quad and \quad \boldsymbol{d}_N, \boldsymbol{f}_N \in \mathbb{R}^N \end{aligned} \tag{1.2}$$

Solving (1.2) should be relatively easy compared to (1.1) assuming $N \ll N_h$. A key feature of the construction of the reduced model is that it is based on manipulating the original model, making this a N_h-dependent process. As a result, constructing reduced order models is computationally costly in itself, as we can assume that N_h becomes rather large when model order reduction is considered. However the goal of the reduction is all about the behaviour of the reduced model. If one is able to create a well performing reduced model this can be highly beneficial. This can either make up for the time consuming construction step, or be a pure benefit if the problem can be split into two steps where only high-speed solving of the final system is of interest. The original or full-order model can become prohibitively slow when it is applied to repetitive solving of PDEs and expected to do so quickly and efficiently.

1.3 Parametric problems

The theory of reducible models and the construction of these will be given later in Chapter 3, but initially it is key to introduce the concept of parametric problems. Although model order reduction may be of interest for problems without any parametric variation, being able to capture parametric dependencies reveals a new dimension of problems that can be solved. For example the repetitive solving of PDEs for optimizing with respect to some quantity by variation of a set of parameters. Introducing the input-parameter vector $\boldsymbol{\mu} = [\mu_1, ..., \mu_p]$, where each parameter is stored, the parametric counterparts of (1.1) and (1.2) becomes

$$\begin{aligned} \boldsymbol{A}_{N_h}(\boldsymbol{\mu}) \boldsymbol{d}_{N_h}(\boldsymbol{\mu}) &= \boldsymbol{f}_{N_h}(\boldsymbol{\mu}) \\ \boldsymbol{A}_N(\boldsymbol{\mu}) \boldsymbol{d}_N(\boldsymbol{\mu}) &= \boldsymbol{f}_N(\boldsymbol{\mu}) \end{aligned} \tag{1.3}$$

Parametric problems aims to calculate some behaviour or field quantity dependent on a set of input parameters. As each input parameter is allowed to vary over a given interval the solution methods used must be able to capture the parameter dependencies on the solution, as found in the original or physical problem. For this thesis the field of structural mechanics and linear elasticity will be used as framework as numerical studies will consider the deformation on 2D beams. For this kind of problems there are many candidates for input parameters such as material properties, loading and geometry. A special emphasis will be put on geometrical parameter variation and how this influence the construction and performance of reduced models. Geometric parameters is found as variables that define the geometry of the system, an easy example being the height and length of a rectangular beam. Enabling geometric variation leads to a more general solver and a very powerful tool as changes in geometry usually results in the complete or partial reconstruction of a

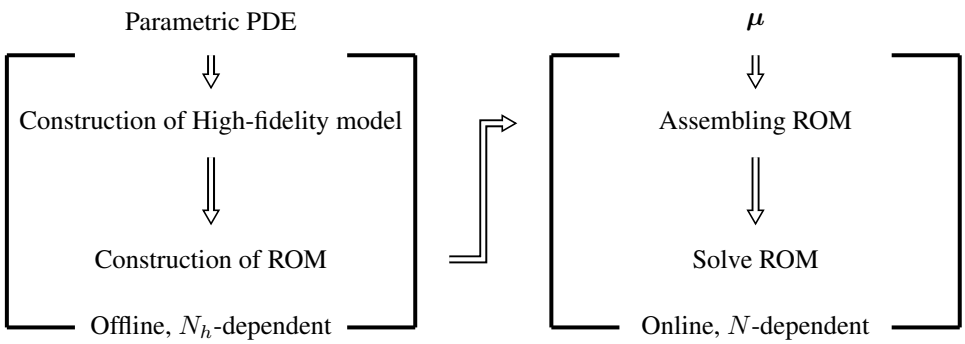


Figure 1.1: ROM workflow

model. As will be described further in Chapter 2 and Chapter 3, being able to produce a reduced model with respect to geometric variation will decide how the finite element system i constructed, and this will be a major step to overcome.

In Figure 1.1 the computational workflow for redcued order models is outlined. As can be seen the process is split into two steps, online and offline. This will be discussed in further detail in Chapter 3, but it is key to understanding the benefits of a ROM. For now it can be said that the online step is crucial to the sucess of a reduced model, and it is recognized as defining the input-parameter vector $\boldsymbol{\mu}$ and solving the fully N dependent problem.

1.4 Scope of thesis

There exist several fields where reduced order models can be applied, and several approaches to construct these models. For this thesis however, Galerkin reduced basis method will be applied to reduce full order finite element models. Some comparisons with other methods are given, but the main emphasis lies on the construction of reduced order models with respect to geometric variation.

The overall goal of this thesis therefore becomes creating a reduced order model for a linear elastic 2D beam with material and geometrical parameters, and analyzing its behaviour. This is done through computer implementation and numerical studies in Chapter 4, where emphasis is put on overall performance according to the theory of reduced order models as well as robustness, applicability of this approach and the construction of the reduced models.

The necessary theoretical foundation for the computer implementation was obtained through a literature review of the fields of Finite element method and Reduced order modelling, represented in Chapters 2 and 3 respectively. This was partly done through a preliminary project [3] where numerical studies was done for simple 2D beams.

2 | The Finite Element Method

In the following chapter the fundamental theory of finite element analysis (FEA) is given, as well as a more practical derivation for liner elastic problems which will be implemented for numerical studies in Chapter 4.

2.1 Theoretical foundation of the Finite Element Method

2.1.1 Formulation of PDEs

The strong formulation of PDEs can be written on the following form [4]

Find u such that

$$\begin{aligned} \mathcal{L}(u) &= l \quad in\ \Omega \\ \mathcal{K}(u) &= h \quad on\ \Gamma_N \\ u &= g \quad on\ \Gamma_D \end{aligned} \tag{2.1}$$

In this formulation, Ω is the domain, Γ is the boundary, and $\mathcal{L}$ is a differential operator. The Dirichlet boundary conditions is found as prescribing values of the function u on the boundary Γ_D, and the Neumann boundary conditions is found as prescribing values of the derivatives, $\mathcal{K}(u)$, on the boundary Γ_N, $\mathcal{K}$ also being a differential operator.

For the derivation of the finite element method, another formulation is needed, namely the weak formulation. The weak formulation is obtained by multiplying the strong form (2.1) with a test function $v \in V$ and integrating over the domain.

$$\int_\Omega (\mathcal{L}(u))v d\Omega = \int_\Omega l v d\Omega \tag{2.2}$$

Trough integrating by parts it can be shown that the following expression can be obtained

Find $u \in V$ such that

$$a(u, v) = f(v) \quad \forall v \in V \tag{2.3}$$

This is known as the weak formulation and it is the starting point for constructing the FE-formulation. $a(.,.)$ is a bilinear form and $f(.)$ is linear form, both derived from integration by parts of (2.2) and imposing boundary conditions. V is recognized as the space of admissible functions u and v. In section 2.2 the weak form is derived for linear elasticity. For now homogeneous Dirichlet boundaries is assumed, $g = 0$, but the general case is

treated in section 2.1.6. The space V is described in more detail in the next section, but for now it can be said that functions $v \in V$ needs to meet some requirements of *smoothness*.

2.1.2 Finite element spaces and discretization

The FEM solution is based on solving an altered form of the weak formulation, mainly by approximating the functions u and v. This is seen as finding the best approximation of u by searching in a subset of V, the FE-solution becomes $u_h \in V_h \subset V$. The finite element space is a subspace of the solution space, which is called a conforming finite element space [5]. The FEM solution should be the best approximation in V_h with respect to some norm.

The space V describes the admissible functions u and v for the problem to be well posed, and some knowledge of V is needed if a subset V_h is to be found. The space is determined with respect to $a(.,.)$ and the Dirichlet boundary conditions. For the problem to have a unique solution, $a(v,v)$ and $f(v)$ must remain finite for all v, which in turn describes the space V. By using $a(.,.)$ as the *inner product* a space can be defined

$$V = \{v|\ a(v,v) < \infty\} \tag{2.4}$$

The bilinear form $a(.,.)$ can be impractical to use and the space can also be constructed by making use of other well known spaces. This is done by choosing a Hilbert space corresponding to the derivative order of the terms in $a(.,.)$, and taking the subset of this space where the Dirichlet boundary conditions are satisfied. For 2D linear elastics, which will be implemented in this thesis, $a(.,.)$ contains first order derivatives in two axis (x_1, x_2). This leads to the following space

$$\begin{aligned} &H^1(\Omega) \equiv \{v| \int_\Omega v^2 d\Omega < \infty, \int_\Omega v_{x_1}^2 d\Omega < \infty, \int_\Omega v_{x_2}^2 d\Omega < \infty\} \\ &V = \{v \in H^1(\Omega) \mid v_{|\Gamma_D} = 0\} \end{aligned} \tag{2.5}$$

It can also be shown that $f(.) \in V'$ needs to be satisfied for the problem to be well posed, however this is not as straight forward and is not done here. In the case of linear elastics this can be interpreted as which loads are allowed to be considered.

From [4], $V_h \subset V$ is found by introducing a triangulation $\mathcal{T}_h$ of the domain Ω. $\mathcal{T}_h$ is a union of non-overlapping elements T_h^k that cover the domain. This is known as the mesh of the domain. The meshing of a domain can be done by elements of different polygonal shapes, for example segments in $\mathbb{R}$, quadrilaterals in $\mathbb{R}^2$ and hexahedron in $\mathbb{R}^3$. The subfix h refers to the general size of the elements in the mesh.

The closed domain can then be written as

$$\bar{\Omega} = \bigcup_{k=1}^{K} \bar{T}_h^k, \quad k = 1, ..., K : Elements \tag{2.6}$$

Figure 2.1 shows an example of a uniform mesh of quadrilaterals for a 2D geometry. The meshing of the domain is directly related to the performance of the FE solution, and becomes challenging as complex geometries are considered. For this thesis however rather simple geometries are studied, and all meshing will be done by uniform meshes of quadrilaterals as seen in Figure 2.1.

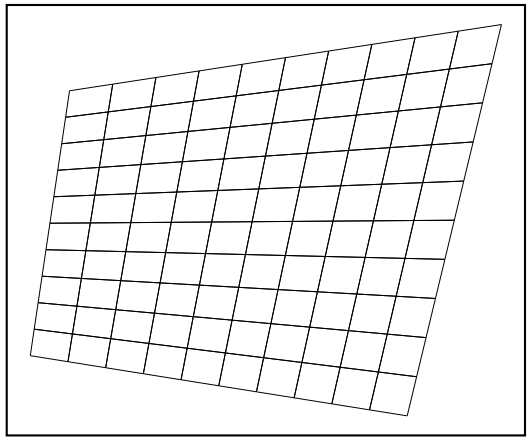


Figure 2.1: Example of uniform mesh

The most common way to define a finite element space is to consider globally continuous functions that are polynomials of degree r on the single elements of the triangulation $\mathcal{T}_h$.

$$V_h^r = \{v_h \in V |\ v_{|T_h^k} \in \mathbb{P}_r(T_h^k) \quad k = 1, ..., K\} \tag{2.7}$$

By introducing interpolation functions $\varphi_i \in V_h^r$ and demanding that φ_i is zero at all other nodes except x_i and $\sum_{i=1}^{n} \varphi_i = 1$, any function $v \in V_h^r$ can now be written as

$$\begin{aligned} v(\boldsymbol{x}) &= \sum_{i=1}^{n} \varphi_i(\boldsymbol{x}) v_i \\ v(\boldsymbol{x}_i) &= v_i \\ \boldsymbol{v} &= \begin{bmatrix} v_1 & ... & v_n \end{bmatrix}^T \end{aligned} \tag{2.8}$$

The number n refers to the number of nodes. The field is now approximated by nodal values v_i and basis functions φ_i. If a function v is of polynomial degree r or less, $v \in \mathbb{P}_r(\Omega)$, the function can be represented exactly by (2.8). If (2.8) is used to represent a function v of polynomial degree $p > r$, it will produce an approximation. The quality of the approximation increases as the difference in polynomial degree $\Delta_p = p - r$ decreases and it is also increased by refining the mesh. In practice this means that the FE approach approximates the weak formulation with suitable basis functions, and accuracy is increased by refining the mesh or the polynomial order of the basis functions.

By introducing $\boldsymbol{N}_0 = \begin{bmatrix}\varphi_1 & ... & \varphi_n\end{bmatrix}$ any field variable can be written on the form

$$v(\boldsymbol{x}) = \boldsymbol{N}_0 \boldsymbol{v} \tag{2.9}$$

This is used to approximate the functions u and v, and discretizes the problem as it is now dependent on the nodal values of u_h and v.

2.1.3 Mapping

To enable irregular an complicated geometry, mapping is introduced as found in [6]. The concept is based on mapping the physical geometry to a reference geometry with the help of a set of interpolation functions. There are different approaches for choosing these interpolation functions, and two will be discussed here.

Isoparametric mapping is a well known approach for mapping in FE-analysis. In this case the same set of interpolation functions, $\boldsymbol{N}_0$, used to approximate the field is chosen to map the geometry. This method is beneficial as it makes use of already created functions.

Another approach, which will be used later in this thesis, is sub-parametric mapping. In general $\boldsymbol{N}_0$ contains n basis functions, which might be a lot more terms than what is needed to describe the actual geometry. For this thesis the geometries of interest will only consist of quadrilaterals, and the interpolation functions needed can easily be derived by hand, as is done later in section 2.3.3.

The isoparametric approach is the more general solution, while the sub-parametric approach is usually only suitable for simple geometries. Isoparametric refers to the fact that the geometry and field variable are sampled at all the same nodes, while for sub-parametric mapping the geometry is sampled at fewer nodes than the field variables.

The original problem is mapped from the (x, y)-space to a *natural* coordinate system (ξ, η) with dimensionless axis. This is shown in Figure 2.2. The interpolation functions are derived in terms of natural coordinates, $\varphi_i(\xi, \eta)$ instead of the physical axis of the original problem. The problem remains the same, but the transformation between the two coordinate systems has to be taken into account.

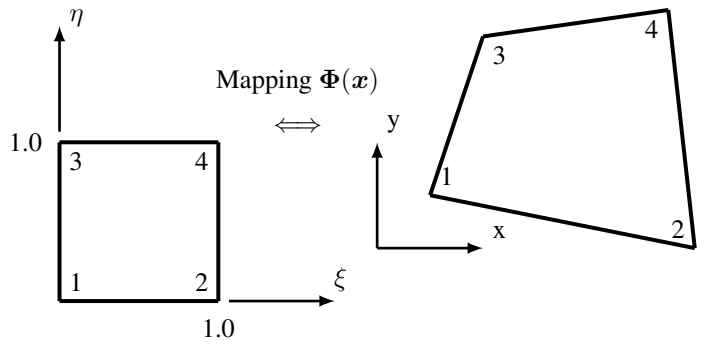


Figure 2.2: Illustration of mapping

The geometry as any other field variable can be approximated by interpolation functions. For a 2D geometry this leads to the following expression

$$\boldsymbol{\Phi}(\boldsymbol{x}) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \boldsymbol{N}_g & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{N}_g \end{bmatrix} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \end{bmatrix} \tag{2.10}$$

Where $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^n$ holds the nodal (x, y) coordinates. The functions $\boldsymbol{N}_g$ and nodal coordinates are dependent on the mapping approach, either making use of $\boldsymbol{N}_0$ or creating new ones.

The PDE is dependent on derivation and integration with respect to the Cartesian coordinates (x, y). To obtain a relationship between derivation in the two spaces the Jacobian matrix is introduced. This is done by establishing the *natural* derivatives expressed by Cartesian derivatives, utilizing the chain rule

$$\begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi}\frac{\partial}{\partial x} + \frac{\partial y}{\partial \xi}\frac{\partial}{\partial y} \\ \frac{\partial x}{\partial \eta}\frac{\partial}{\partial x} + \frac{\partial y}{\partial \eta}\frac{\partial}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \boldsymbol{J} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \tag{2.11}$$

The Jacobian matrix, $\boldsymbol{J}$, is denoted as

$$\nabla \boldsymbol{\Phi} = \boldsymbol{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial \boldsymbol{N}_g}{\partial \xi} \\ \frac{\partial \boldsymbol{N}_g}{\partial \eta} \end{bmatrix} \begin{bmatrix} \boldsymbol{X} & \boldsymbol{Y} \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{2.12}$$

The inverse relationship yields the derivatives with respect to (x, y).

$$\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \boldsymbol{J}^{-1} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} \tag{2.13}$$

For integration the relation $d\Omega = J \partial \xi \partial \eta$ is applied, where J known as the Jacobian, is the determinant of the Jacobian matrix.

For the mapping to be applicable to a problem it needs to be unique, meaning that there exists a *one-to-one* relationship between each point in the physical and the natural coordinate system. It can be shown that the condition for unique mapping is [7]

$$J = det(\boldsymbol{J}) > 0 \tag{2.14}$$

The mapping of geometry does not come without disadvantages as it introduces computational complexity, and can lead to lower convergence rates and loss of accuracy. The quality of the solution is therefore dependent on the level of distortion between physical and reference geometry, meaning that the more regular the problem geometry is, the better the performance.

The concept of mapping will prove to be important to enable reduction of the model, and in section 2.3.3 the implementation is taken further for the linear elastic case.

2.1.4 Projection and orthogonality

The FE-problem aims to find the best approximation $u_h \in V_h \subset V$ of the original function u with respect to some norm. Implementing this in the weak formulation (2.3) yields

Find $u_h \in V_h$ such that

$$a(u_h, v) = f(v) \quad \forall v \in V_h \tag{2.15}$$

This leads to the following orthogonality

$$\begin{aligned} a(u, v) &= f(v) \quad \forall v \in V \\ a(u, v) &= f(v) \quad \forall v \in V_h \\ a(u_h, v)) &= f(v) \quad \forall v \in V_h \\ a(u - u_h, v) &= 0 \quad \forall v \in V_h \end{aligned} \tag{2.16}$$

Meaning that u_h is minimizing the error $\epsilon = u - u_h$ in the energy-norm (a-norm), defined as $||v||_a^2 = a(v, v)$. From Figure 2.3 this can be seen as u_h being the orthogonal projection of u onto V_h, and there is no better approximation of u in V_h with respect to $a(.,.)$. This property is called Galerkin orthogonality [8].

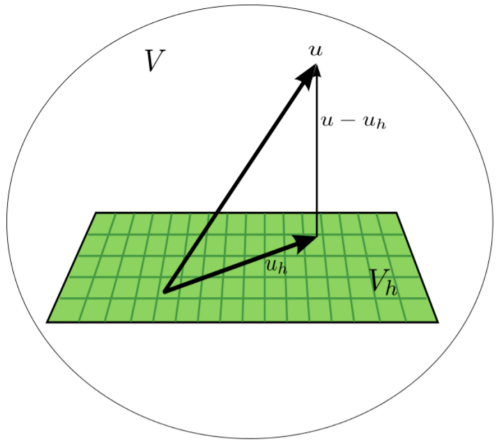


Figure 2.3: Galerkin orthogonality, Figure from [8]

2.1.5 Assembly

The FE solution can be written on the form

$$\begin{aligned} u_h(\boldsymbol{x}) &= \sum_{j=1}^{n} \varphi_j(\boldsymbol{x}) u_h^j \\ u_h(\boldsymbol{x}_j) &= u_h^j \\ \boldsymbol{u}_h &= \begin{bmatrix} u_h^1 & \dots & u_h^n \end{bmatrix}^T \end{aligned} \tag{2.17}$$

Inserting (2.8) and (2.17) into the weak formulation (2.15) yields the following discretized system

$$\begin{aligned} \boldsymbol{A}_h \boldsymbol{u}_h &= \boldsymbol{f}_h \\ A_{ij} &= a(\varphi_i, \varphi_j) \\ f_i &= f(\varphi_i) \end{aligned} \tag{2.18}$$

Proof:

Find $u_h \in X_h$ such that

$$\begin{aligned} a(u_h, v) &= f(v) \quad \forall v \in X_h \\ a(\sum_{j=1}^{n} \varphi_j u_h^j, \sum_{i=1}^{n} \varphi_i v_i) &= f(\sum_{i=1}^{n} \varphi_i v_i) \\ \sum_{i=1}^{n}\sum_{j=1}^{n} v_i a(\varphi_i, \varphi_j) u_h^j &= \sum_{i=1}^{n} v_i f(\varphi_i) \\ \boldsymbol{v}^T \boldsymbol{A}_h \boldsymbol{u}_h &= \boldsymbol{v}^T \boldsymbol{f}_h \end{aligned} \tag{2.19}$$

2.1.6 Non-homogeneous Dirichlet boundary conditions

The derivation this far has been based on the assumption of homogeneous Dirichlet boundary conditions. In the more general case it is of interest to be able to solve non-homogeneous problems as well, $u = g \neq 0$ on Γ_D. However this is not possible to solve with the weak formulation as it stands as the sum of two admissible functions no longer coincide with the boundary conditions [9].

This problem can be overcome by introducing a lifting function r_g which enforces the Dirichlet boundary conditions

$$\begin{aligned} r_g &\in V_E \\ r_{g|\Gamma_D} &= g \end{aligned} \tag{2.20}$$

The space V_E refers to a subset of the Hilbert space introduced in section 2.1.2 where the inhomogenous Dirichlet boundary conditions are satisfied. The solution then becomes $u + r_g$, where u is the homogeneous solution. After a suitable lifting function has been chosen, a slightly altered problem can be solved for u as

$$a(u + r_g) = a(u, v) + a(r_g, v) \tag{2.21}$$

the weak formulation becomes

Find u in V such that

$$a(u, v) = f(v) - a(r_g, v), \quad \forall v \in V \tag{2.22}$$

The forms $a(.,.)$ and $f(.)$ remain unchanged, and V is still the FE-space with homogeneous Dirichlet boundary conditions.

2.2 Linear elastic problems

In structural mechanics many problems can be viewed through the theory of linear elasticity, and FE-analysis is the most common approach to calculate more complex systems where other methods become impractical.

2.2.1 Setting up the problem

Linear elastic problems is described in terms of the stress tensor $\boldsymbol{\sigma} : \mathbb{R}^d \rightarrow \mathbb{R}^d$, the strain tensor $\boldsymbol{e} : \mathbb{R}^d \rightarrow \mathbb{R}^{d \cdot d}$, the body force $\boldsymbol{f} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ and the displacement field $\boldsymbol{u} : \mathbb{R}^d \rightarrow \mathbb{R}^d$. Here d is the dimensionality of the problem, and $d = 2$ for a 2D problem which will be investigated in the following. In Figure 2.4 a linear elastic body is seen with a domain Ω, a body force $\boldsymbol{f}$ and a traction load $\boldsymbol{g}_n$. The body is fixed along the boundary Γ_D, $\boldsymbol{g}_n$ acts along the boundary Γ_N and the dashed line shows the deformed state of the system.

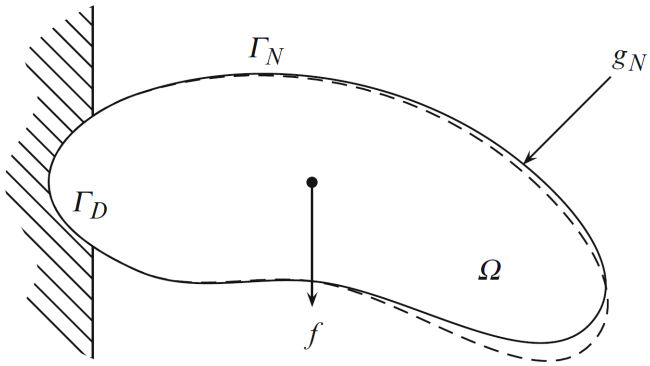


Figure 2.4: Illustration of a linear elastic model, Figure from [10]

The governing equations are as follows [11]

Equilibrium of forces

$$-\nabla \cdot \boldsymbol{\sigma} = \boldsymbol{f} \quad in\ \Omega \tag{2.23}$$

Strain-displacement relation

$$\boldsymbol{\varepsilon}(\boldsymbol{u}) = \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) \tag{2.24}$$

Constitutive law

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon}(\boldsymbol{u}) + \lambda(div(\boldsymbol{u}))\boldsymbol{I} \tag{2.25}$$

Together with Neumann and Dirichlet boundary conditions this yields

$$\begin{aligned} -div(\mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T) + \lambda(div(\mathbf{u}))\mathbf{I}) &= \mathbf{0} \quad in\ \Omega \\ \mathbf{u} &= \mathbf{0} \quad on\ \Gamma_D \\ \boldsymbol{\sigma}\boldsymbol{n} &= \boldsymbol{g}_n \quad on\ \Gamma_N \end{aligned} \tag{2.26}$$

2.2.2 Deriving the weak formulation

The weak formulation of linear elasticity is derived, as found in [10], starting by multiplying with a test function v and integration by parts

$$\begin{aligned} -\nabla \cdot \boldsymbol{\sigma} &= \boldsymbol{f} \\ -\int_\Omega \nabla \cdot \boldsymbol{\sigma}\boldsymbol{v} d\Omega &= \int_\Omega \boldsymbol{f}\boldsymbol{v} d\Omega \\ -\int_\Omega \nabla(\boldsymbol{\sigma}\boldsymbol{v}) d\Omega + \int_\Omega \boldsymbol{\sigma} : \nabla \boldsymbol{v} d\Omega &= \int_\Omega \boldsymbol{f}\boldsymbol{v} d\Omega \end{aligned}$$

Where $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^2$. Using Green's theorem together with $\boldsymbol{\sigma}\boldsymbol{n} = \boldsymbol{g}_n$ on Γ_N, and $\boldsymbol{v} = \mathbf{0}$ on Γ_D yields

$$\int_\Omega \boldsymbol{\sigma} : \nabla \boldsymbol{v} d\Omega = \int_\Omega \boldsymbol{f}\boldsymbol{v} d\Omega + \int_{\Gamma_N} \boldsymbol{g}_n \boldsymbol{v} d\Gamma$$

then decomposing $\nabla \boldsymbol{v}$ in its symmetric and anti-symmetric part we get

$$\int_\Omega \boldsymbol{\sigma} : \nabla \boldsymbol{v} d\Omega = \int_\Omega \boldsymbol{\sigma} : (\frac{1}{2}(\nabla \boldsymbol{v} + \nabla \boldsymbol{v}^T) + \frac{1}{2}(\nabla \boldsymbol{v} - \nabla \boldsymbol{v}^T)) d\Omega$$

The product of the stress tensor and the anti-symmetric part of v is zero, and the symmetric part of $\boldsymbol{v}$ is recognized as $\boldsymbol{\varepsilon}(\boldsymbol{v})$ as seen in (2.24). Inserting the constitutive law (2.25) together with $\boldsymbol{I} : \boldsymbol{\varepsilon}(\boldsymbol{v}) = div(\boldsymbol{v})$ yields

$$\begin{aligned} \int_\Omega \boldsymbol{\sigma} : \boldsymbol{\varepsilon}(\boldsymbol{v}) d\Omega &= \int_\Omega (2\mu \boldsymbol{\varepsilon}(\boldsymbol{u}) + \lambda(div(\boldsymbol{u}))\boldsymbol{I}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) d\Omega \\ &= \int_\Omega 2\mu \boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) d\Omega + \int_\Omega \lambda div(\boldsymbol{u}) div(\boldsymbol{v}) d\Omega \end{aligned}$$

The weak statement then becomes

Find $\boldsymbol{u} \in V$ such that

$$a(\boldsymbol{u}, \boldsymbol{v}) = f(\boldsymbol{v}) \quad \forall \boldsymbol{v} \in V$$

where

$$a(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega 2\mu \boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) d\Omega + \int_\Omega \lambda div(\boldsymbol{u}) div(\boldsymbol{v}) d\Omega \tag{2.27}$$

and

$$f(\boldsymbol{v}) = \int_\Omega \boldsymbol{f}\boldsymbol{v} d\Omega + \int_{\Gamma_N} \boldsymbol{g}_n \boldsymbol{v} d\Gamma \tag{2.28}$$

2.2.3 Implementing the discrete system

The next step is the discretization of the problem by introducing a triangulation of the domain and finding the basis functions,$\varphi_i = \varphi_i(\xi, \eta)$, as described in section 2.1.2. This yields the following expression for field variables

$$\begin{aligned} \boldsymbol{v} &= \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \varphi_1 & \varphi_2 & ... & \varphi_n & 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 & \varphi_1 & \varphi_2 & ... & \varphi_n \end{bmatrix} \boldsymbol{d} \\ \boldsymbol{v} &= \begin{bmatrix} \boldsymbol{N}_0 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{N}_0 \end{bmatrix} \boldsymbol{d} = \boldsymbol{N}\boldsymbol{d} \end{aligned} \tag{2.29}$$

Before introducing the discretization in the weak form the following notation is introduced for the derivatives

$$\frac{\partial \boldsymbol{N}_0}{\partial i} = \boldsymbol{N}_{0,i} \in \mathbb{R}^n \tag{2.30}$$

matrix outer product

$$\boldsymbol{N}_{0,i} \otimes \boldsymbol{N}_{0,j} = \boldsymbol{N}_{0,i}\boldsymbol{N}_{0,j} \in \mathbb{R}^{n \cdot n} \tag{2.31}$$

and expression through submatrices

$$\begin{aligned} \boldsymbol{M} &= \begin{bmatrix} a_1\boldsymbol{A} & a_2\boldsymbol{A} \\ a_3\boldsymbol{A} & a_4\boldsymbol{A} \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \boldsymbol{A} \\ \boldsymbol{M} &\in \mathbb{R}^{2n \cdot 2n}, \quad \boldsymbol{A} \in \mathbb{R}^{n \cdot n}, \quad a_i \in \mathbb{R} \end{aligned} \tag{2.32}$$

The computer implementation of calculating the system matrix and vector is simply implementing (2.18) by inserting (2.29) into (2.27) and (2.28) yielding the following terms.

Part of system matrix dependent on λ

$$\begin{aligned} \boldsymbol{A}_\lambda &= \int_\Omega div(\boldsymbol{N}) \otimes div(\boldsymbol{N}) d\Omega \\ div(\boldsymbol{N}) &= \begin{bmatrix} \boldsymbol{N}_{0,x} & \boldsymbol{N}_{0,y} \end{bmatrix} \\ div(\boldsymbol{N}) \otimes div(\boldsymbol{N}) &= \begin{bmatrix} \boldsymbol{N}_{0,x}\boldsymbol{N}_{0,x} & \boldsymbol{N}_{0,x}\boldsymbol{N}_{0,y} \\ \boldsymbol{N}_{0,y}\boldsymbol{N}_{0,x} & \boldsymbol{N}_{0,y}\boldsymbol{N}_{0,y} \end{bmatrix} \end{aligned} \tag{2.33}$$

Part of system matrix dependent on μ

$$\begin{aligned} \boldsymbol{A}_\mu &= \int_\Omega 2\boldsymbol{\varepsilon}(\boldsymbol{N}) : \boldsymbol{\varepsilon}(\boldsymbol{N}) d\Omega \\ 2\boldsymbol{\varepsilon}(\boldsymbol{N}) : \boldsymbol{\varepsilon}(\boldsymbol{N}) &= \begin{bmatrix} 2\boldsymbol{N}_{0,x}\boldsymbol{N}_{0,x} + \boldsymbol{N}_{0,y}\boldsymbol{N}_{0,y} & \boldsymbol{N}_{0,y}\boldsymbol{N}_{0,x} \\ \boldsymbol{N}_{0,x}\boldsymbol{N}_{0,y} & \boldsymbol{N}_{0,x}\boldsymbol{N}_{0,x} + 2\boldsymbol{N}_{0,y}\boldsymbol{N}_{0,y} \end{bmatrix} \end{aligned} \tag{2.34}$$

This leads to

$$(\lambda\boldsymbol{A}_\lambda + \mu\boldsymbol{A}_\mu)\boldsymbol{d}_h = \boldsymbol{A}_h\boldsymbol{d}_h = \boldsymbol{f} \tag{2.35}$$

Where the left hand side is known as the stiffness matrix

$$\begin{aligned}
\boldsymbol{A}_h &= \sum_{i=1}^{4} \boldsymbol{A}_i = \sum_{i=1}^{4} \int_\Omega \boldsymbol{I}_i d\Omega \\
\boldsymbol{I}_1 &= \begin{bmatrix} \lambda + 2\mu & 0 \\ 0 & \mu \end{bmatrix} \boldsymbol{N}_{0,x} \boldsymbol{N}_{0,x} \\
\boldsymbol{I}_2 &= \begin{bmatrix} 0 & \lambda \\ \mu & 0 \end{bmatrix} \boldsymbol{N}_{0,x} \boldsymbol{N}_{0,y} \\
\boldsymbol{I}_3 &= \begin{bmatrix} 0 & \mu \\ \lambda & 0 \end{bmatrix} \boldsymbol{N}_{0,y} \boldsymbol{N}_{0,c} \\
\boldsymbol{I}_4 &= \begin{bmatrix} \mu & 0 \\ 0 & \lambda + 2\mu \end{bmatrix} \boldsymbol{N}_{0,y} \boldsymbol{N}_{0,y}
\end{aligned} \tag{2.36}$$

and the right hand side is known as the load vector

$$\boldsymbol{f} = \int_\Omega \boldsymbol{N}^T \boldsymbol{f} d\Omega + \int_{\Gamma_N} \boldsymbol{N}^T \boldsymbol{g}_n d\Gamma \tag{2.37}$$

Only traction forces will be applied in the numerical studies, and the body forces $\boldsymbol{f}$ is neglected in the following. The vector $\boldsymbol{d}_h$ holds the nodal values of the FE-solution while the entire field is found as

$$\boldsymbol{u}_h = \boldsymbol{N} \boldsymbol{d}_h \tag{2.38}$$

2.3 Parametric PDEs

2.3.1 Parametric problems

All PDEs are dependent on a set of parameters that define the specific problem. For parametric problems however some of these parameters are allowed to vary, as the dependencies between parameter variation and response is of interest. These parameters are stored in the parameter-input vector $\boldsymbol{\mu} = [\mu_1, ..., \mu_p] \in \mathcal{P} \subset \mathbb{R}^p$. Examples of parameters can be boundary conditions, geometrical properties, the Reynolds number for fluid mechanics or the material constants in structural mechanics. This changes the weak formulation (2.3) to

Given $\boldsymbol{\mu} \in \mathcal{P}$, find $u(\boldsymbol{\mu}) \in V$ such that

$$a(u(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = f(v; \boldsymbol{\mu}) \quad \forall v \in V \tag{2.39}$$

Both u, as well as $a(.,.)$ and $f(.)$ are dependent on $\boldsymbol{\mu}$, and the FEM-expression (2.18) becomes

Given $\boldsymbol{\mu} \in \mathcal{P}$, find $u_h(\boldsymbol{\mu}) \in V_h$ such that

$$\begin{aligned}
a(u_h(\boldsymbol{\mu}), v_h; \boldsymbol{\mu}) &= f(v_h; \boldsymbol{\mu}) \quad \forall v_h \in V_h \\
\boldsymbol{A}_h(\boldsymbol{\mu}) \boldsymbol{u}_h(\boldsymbol{\mu}) &= \boldsymbol{f}_h(\boldsymbol{\mu})
\end{aligned} \tag{2.40}$$

The FE-solution, u_h, will in the latter be referred to as the high-fidelity solution. This comes from the fact that for a given $\boldsymbol{\mu}$ the FE-solution can be obtain with a desired accuracy. The desired accuracy is achieved by refining the mesh or the polynomial order of the basis functions, and thereby increasing the computational cost. This makes solving the FE-problem accurate at the expense of computing time which can be problematic if the problem is to be solved numerous times for different inputs $\boldsymbol{\mu}$.

For a problem to be reducible it must be described affinely dependent on the parameters of interest. This will be described in section 3.2.2, but in the next two sections this affinity will be achieved for different parameters in linear elasticity.

2.3.2 Material and load parametrization

For the material parameters, affine representation is easily achieved, especially if the problem is described in terms of Lamè coefficients, $\boldsymbol{\mu}_M = [\lambda, \mu]$. As only the stiffness matrix is dependent on the material parameters it can be seen from (2.35) that the stiffness matrix can be written on the following form

$$\boldsymbol{A}_h = \lambda \boldsymbol{A}_\lambda + \mu \boldsymbol{A}_\mu \tag{2.41}$$

For the traction forces on the different boundaries it is obvious that the load vectors will be scaled by the load intensities on each boundary, $\boldsymbol{\mu}_n = [\mu_{n1}, ..., \mu_{nk}]$ yielding

$$\boldsymbol{f} = \sum_{i=1}^{k} \mu_{ni} \int_{\Gamma_{N_i}} \boldsymbol{g}_{ni} \boldsymbol{N}^T d\Gamma \tag{2.42}$$

2.3.3 Geometric parametrization

For geometrical parametrization the nodal coordinates used in the mapping of the physical geometry found in section 2.1.3, $\boldsymbol{\mu}_G = [\boldsymbol{X}, \boldsymbol{Y}]$, will be used as parameters. The examples investigated in this thesis will be of quadrilateral shape, and are therefore described uniquely by the (x, y)-coordinates of the four corner nodes and the corresponding Lagrange polynomials. These functions are used instead of the basis functions of the mesh as this reduces complexity and allows for hand calculation of some key expressions. By implementing the mapping introduced in Figure 2.2 it can be showed that this leads to the following expression

$$\begin{aligned} \boldsymbol{\Phi}(\boldsymbol{x}) &= (1-\xi)(1-\eta)\boldsymbol{C}_1 + \xi(1-\eta)\boldsymbol{C}_2 + (1-\xi)\eta\boldsymbol{C}_3 + \xi\eta\boldsymbol{C}_4 \\ \boldsymbol{C}_i &= \begin{bmatrix} x_i \\ y_i \end{bmatrix} \end{aligned} \tag{2.43}$$

For simplicity $\boldsymbol{C}_1 = \boldsymbol{0}$ is assumed which yields the following expression for the Jacobian matrix

$$\begin{aligned}\nabla\boldsymbol{\Phi} = \boldsymbol{J} &= [\boldsymbol{C}_2, \boldsymbol{C}_3]^T + \begin{bmatrix}\eta \\ \xi\end{bmatrix}[\boldsymbol{C}_4 - \boldsymbol{C}_2 - \boldsymbol{C}_3]^T \\ &= \boldsymbol{C}(\boldsymbol{\mu}) + \tilde{\boldsymbol{x}}\boldsymbol{S}(\boldsymbol{\mu})^T \\ &= \begin{bmatrix}x_2 & y_2 \\ x_3 & y_3\end{bmatrix} + \begin{bmatrix}\eta S_x & \eta S_y \\ \xi S_x & \xi S_y\end{bmatrix} \\ S_x = x_4 - x_2 - x_2, &\quad S_y = y_4 - y_2 - y_3\end{aligned} \tag{2.44}$$

The matrix function $\boldsymbol{t}(\boldsymbol{\mu})$ is introduced

$$\boldsymbol{t}(\boldsymbol{\mu}) = \boldsymbol{S}^T(\boldsymbol{\mu})^T\boldsymbol{C}(\boldsymbol{\mu})^{-1} \tag{2.45}$$

Where $\boldsymbol{C}(\boldsymbol{\mu}) \in \mathbb{R}^{2\cdot 2}$ and $\boldsymbol{S}(\boldsymbol{\mu}), \boldsymbol{t}(\boldsymbol{\mu}) \in \mathbb{R}^2$. From this expressions for the Jacobi determinant and the inverse Jacobi matrix can be found

$$\begin{aligned}J = det(\boldsymbol{J}) &= (1 + \boldsymbol{t}(\boldsymbol{\mu})\tilde{\boldsymbol{x}})det\boldsymbol{C}(\boldsymbol{\mu}) = (1 + t_1\eta + t_2\xi)det\boldsymbol{C}(\boldsymbol{\mu}) \\ \boldsymbol{J}^{-1} &= \frac{1}{J}(\begin{bmatrix}y_3 & -y_2 \\ -x_3 & x_2\end{bmatrix} + \begin{bmatrix}\xi S_y & -\eta S_y \\ -\xi S_x & \eta S_x\end{bmatrix}) \\ R &= \frac{1}{1 + \boldsymbol{t}(\boldsymbol{\mu})\tilde{\boldsymbol{x}}} = \frac{1}{1 + t_1\eta + t_2\xi}\end{aligned} \tag{2.46}$$

For $\boldsymbol{J}^{-1}$ to be a linear combination scaled by the coordinates in $\boldsymbol{\mu}_G$, enabling affine representation the term R has to be approximated by a series. This can be done by recognizing R as a geometric series [12], and by utilizing the binomial theorem [13]. This is derived in the following given $|t_1\eta + t_2\xi| < 1$

$$\begin{aligned}R = \frac{1}{1 + t_1\eta + t_2\xi} &= \sum_{i=0}^{\infty}(-1)^i(t_1\eta + t_2\xi)^i \\ &= \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}(-1)^i\binom{i}{j}(t_1\eta)^j(t_2\xi)^{i-j} \\ R_n = \sum_{i=0}^{n}\sum_{j=0}^{i}(-1)^i\binom{i}{j}(\xi t_2)^i(\frac{\eta}{\xi})^j(\frac{t_1}{t_2})^j\end{aligned} \tag{2.47}$$

For computer implementation, a finite value of n must be chosen to obtain a sufficient approximation $R_n \approx R$. As increasing n introduces computational complexity a reasonable value must be chosen. It is key to recognize that the series expansion of R alters the problem as an approximate stiffness matrix is calculated. This will be investigated in the numerical studies for the different examples, but again it is noted that as the distortion between physical and reference geometry increases, the accuracy of the FE solution decreases.

With the expressions for the Jacobian matrix as well as the inverse relations, general quadrilateral shapes can be mapped to the same reference geometry with the same element mesh. This makes it possible to utilize the ROM approach which will be introduced

in Chapter 3. The affine representation of the problem with respect to the corner nodes is not as straight forward as for material and load parameters and is not done here. It should be noted that the general case is quit cumbersome to implement, but if symmetric conditions can be introduced the expressions become less complex. This is done for two different examples i chapter 4.

3 | Reduced order modelling

The use of numerical approximation like FEA to solve PDEs transform the problem into first order derivation matrix equations. This leads to the order of solving such problems being equal to the size of the matrix, not the order of the derivatives. Therefore, model order reduction relates to reducing the size of the matrix and can be useful to improve efficiency of analysis.

The concepts of model order reduction is mathematically founded in eigen-value problems [14], and in the following chapter the theoretical foundation for model order reduction and its algebraic structure is presented.

3.1 Introduction to model order reduction

3.1.1 Motivation

The goal of constructing a reduced order model is to transform the original N_h-dependent problem to a reduced N-dependent problem, and as $N \ll N_h$ this should lead to a large reduction in computational complexity. Here N_h and N refers to the number of degrees of freedom for the high-fidelity and reduced model respectively.

The idea is that although the construction of the reduced model might be somewhat cumbersome and time consuming, the reduced model can afterwards be solved with fairly good accuracy and much faster than the original problem. This is referred to as offline and online computation as was seen in Figure 1.1. The offline step consist of constructing the FE-model and from this obtaining a reduced model with the help of eigen-values and vectors. The online phase simply consist of assembling and solving the N-dependent problem given an input vector $\boldsymbol{\mu}$. The reduction in computational complexity for the online step gives ROMs a large upside once they are created, as a small reduction in accuracy is traded for a large reduction in computational cost. The applicability of model order reduction is therefore dependent on how a problem can be split in offline and online steps and which factors are of emphasis, two examples are given.

Optimization problems can greatly benefit from ROMs with respect to computational efficiency as a high-fidelity model usually has to be solved numerous times [15], and can be replaced by a ROM. An example of this is the design process of structural systems where numerous load cases is simulated in each iteration of the optimization [16].

Another example where ROMs are applicable are problems where real-time solving of a problem is of interest, which means that solving the high-fidelity problem is too slow. This

makes model order reduction highly relevant for the emergence of Digital Twins which is viewed as an important technology for the future [17].

The types of problems where ROMs are of interest are often characterized by being variational problems in the sense that some parameter variation is studied, for example the parametric PDEs described in section 2.3. For ROMs to be applicable they have to be robust with respect to the parameter changes that are studied. This means that the process of model order reduction must preserve the parametric dependencies as found in the original problem.

3.1.2 Reduced order model vs. Response surface methodology

Response surface methodology (RSM) is an area of statistics for model fitting of a response value to a group of input variables [18]. For a practical problem this means sampling the parameter space, and building an approximation of the response based on the samples. This is a well known method for reducing computational complexity. How the variables are sampled are crucial for the accuracy of this method as there is always the risk of important parameter dependencies being lost.

Although the term *reduced order modelling* is used for a variety of approaches in the literature, in this thesis it is recalled as the reduction of a higher order system through projection based methods.

The main advantage of ROMs is the coupling with the high-fidelity model, and that it in some sense "captures the physics" of a problem. Response surface methodology on the other hand is more of a black box approach and no other relations than input-parameter to response is captured. This leads to some differences which can be exemplified for FEA in structural mechanics.

The ROM will be created with the deformations of the system as principal unknowns. However, as the deformations is solved for by the reduced model, other quantities can be derived and the physical behavior of the system can be analyzed, for example analyzing the stress distribution in the domain. The response surface method does not retain these physical traits and the only way to include relations to other quantities is to include them as part of the response, thereby increasing the complexity of the approximation.

3.2 Galerkin reduced basis method

A well known approach for constructing reduced order models is by the use of reduced basis methods (RB-methods). The following derivation of reduced order models is based on [11].

3.2.1 Theoretical foundation of RB-methods

The Galerkin reduced basis method is in a nutshell a Galerkin projection of the high-fidelity solution onto a N-dimensional space V_N for any $\boldsymbol{\mu} \in \mathcal{P}$. The reduced basis $V_N \subset$

V_h is generated from a set of high-fidelity solutions $[u_h(\boldsymbol{\mu}^1), ..., u_h(\boldsymbol{\mu}^{n_s})]$, called *snapshots* which corresponds to a set of $n_s \geq N$ selected parameter vectors $[\boldsymbol{\mu}^1, ..., \boldsymbol{\mu}^{n_s}] \subset \mathcal{P}$. A set of N functions $[\zeta_1, ...\zeta_N]$ is generated to be the reduced basis. This is done by orthonormalization of the snapshots with respect to a suitable scalar product $(.,.)_X$.

We have that

$$V_N = span\{\zeta_1, ..., \zeta_N\} = span\{u(\boldsymbol{\mu}^1), ..., u(\boldsymbol{\mu}^{n_s})\} \tag{3.1}$$

This is an important trait for the reduced basis as only dominant directions in the span of snapshots are identified, and non contributing vectors are neglected. The central assumption being that the solution manifold of the high-fidelity model can be spanned by a low-dimensional basis [19]. If this is not the case this may lead to large sized models for fine discretizations of the parameter space [20].

The RB solution, $u_N(\boldsymbol{\mu}) \in V_N$, is then expressed as a linear combination of the reduced basis functions ζ_m and RB-coefficients $u_N^{(m)}(\boldsymbol{\mu})$

$$u_N(\boldsymbol{\mu}) = \sum_{m=1}^{N} \zeta_m u_N^{(m)}(\boldsymbol{\mu}) \tag{3.2}$$

This expression is analogue to (2.8) of the finite element approach.

This form of reduction although being a projection method is also a response approximation as the reduced basis is generated from snapshots of the response. Since this method depends on responses, just like RSM accuracy is highly influenced by the choice of snapshots and there is always a risk of neglecting important parameter dependencies. This leads to parameter sampling becoming a field in its own. This is given a brief overview in section 3.5, but is not studied in detail for this thesis.

3.2.2 Algebraic form of the ROM machinery

Given a reduced basis V_N, combining (3.2) with weak formulation (2.15) transforms the high-fidelity problem. The transformation consists of projecting solutions and test functions $u_h, \in V_h$ onto the subspace V_N. This leads to

Find $u_N(\boldsymbol{\mu}) \in V_N$ such that

$$a(u_N(\boldsymbol{\mu}), v_N; \boldsymbol{\mu}) = f(v_N; \boldsymbol{\mu}), \quad \forall v_N \in V_N \tag{3.3}$$

As was done when deriving the FEM, test functions v_N is chosen to be expressed by the same interpolation functions as u_N. By setting $v_N^{(n)} = \zeta_n$, $1 \leq n \leq N$, a set of N linear algebraic equations emerges

$$\sum_{m=1}^{N} a(\zeta_m, \zeta_n; \boldsymbol{\mu}) u_N^{(m)}(\boldsymbol{\mu}) = f(\zeta_n; \boldsymbol{\mu}), \quad 1 \leq n \leq N \tag{3.4}$$

Which leads to the following discrete system

$$\begin{aligned}
&\boldsymbol{A}_N(\boldsymbol{\mu})\boldsymbol{u}_N(\boldsymbol{\mu}) = \boldsymbol{f}_N(\boldsymbol{\mu}) \\
&(\boldsymbol{A}_N(\boldsymbol{\mu}))_{nm} = a(\zeta_m, \zeta_n; \boldsymbol{\mu}) \\
&(\boldsymbol{f}_N(\boldsymbol{\mu}))_n = f(\zeta_n; \boldsymbol{\mu}) \\
&\boldsymbol{u}_N(\boldsymbol{\mu}) = \{u_N^1(\boldsymbol{\mu}), .., u_N^N(\boldsymbol{\mu})\}^T
\end{aligned} \tag{3.5}$$

Where $\boldsymbol{A}_N \in \mathbb{R}^{N \cdot N}$ and $\boldsymbol{f}_N \in \mathbb{R}^N$. Since the basis functions ζ_m belongs to V_h of the high-fidelity system, representation by the original interpolation functions φ^i and coefficients $\zeta_m^{(i)} = \zeta_m(\boldsymbol{x}_i)$ is possible

$$\zeta_m(\boldsymbol{x}) = \sum_{i=1}^{N_h} \zeta_m^{(i)} \varphi^i(\boldsymbol{x}) \quad 1 \leq m \leq N \tag{3.6}$$

By inserting (3.6) into the expression for $\boldsymbol{A}_N$ and $\boldsymbol{f}_N$ as found in (3.5), the following is obtained

$$\begin{aligned}
\boldsymbol{A}_N = a(\zeta_m, \zeta_n) &= \sum_{i=1}^{N_h} \sum_{j=1}^{N_h} \zeta_m^{(j)} a(\varphi^j, \varphi^i) \zeta_n^{(i)} \\
\boldsymbol{f}_N = f(\zeta_n) &= \sum_{i=1}^{N_h} f(\varphi^i) \zeta_n^{(i)}
\end{aligned} \tag{3.7}$$

Introducing the transformation matrix $\boldsymbol{V} \in \mathbb{R}^{N_h \cdot N}$ as

$$(\boldsymbol{V})_{im} = \zeta_m^{(i)}, \quad 1 \leq m \leq N,\ 1 \leq i \leq N_h \tag{3.8}$$

Transforming the system to

$$\boldsymbol{V}^T \boldsymbol{A}_h(\boldsymbol{\mu}) \boldsymbol{V} \boldsymbol{u}_N(\boldsymbol{\mu}) = \boldsymbol{V}^T \boldsymbol{f}_h(\boldsymbol{\mu}) \tag{3.9}$$

The term reduced basis refers to the set RB functions ζ_m, but as these are now represented with interpolation functions φ^i and coefficients $\zeta_m^{(i)}$ the term is shifted to refer to the transformation matrix $\boldsymbol{V}$. Also, by using the original interpolation functions φ^i to describe the basis functions ζ_m all that is needed to reduce the model is the discrete transformation matrix $\boldsymbol{V}$, as $\boldsymbol{A}_h(\boldsymbol{\mu})$ and $\boldsymbol{f}_h(\boldsymbol{\mu})$ are known.

This looks promising, but there is one more obstacle that needs to be dealt with. As described in section 3.1.1 the ROM should be robust with respect to parameter variation. If parameter changes is introduced to (3.9) the N_h-dependent matrix $\boldsymbol{A}_h(\boldsymbol{\mu}) \in \mathbb{R}^{N_h \cdot N_h}$ and vector $\boldsymbol{f}_h(\boldsymbol{\mu}) \in \mathbb{R}^{N_h}$ has to be recalculated for each input-vector $\boldsymbol{\mu}$. Construction of such a model is not viable as it has the same computational complexity as the high-fidelity

model. This is overcome by demanding both $a(.,.)$ and $f(.)$ to be affine (or separable) with respect to $\boldsymbol{\mu}$. This can be written as

$$\begin{aligned} a(w, v; \boldsymbol{\mu}) &= \sum_{q=1}^{Q_a} \theta_a^q(\boldsymbol{\mu}) a_q(w, v) \quad \forall v, w \in V \\ f(v; \boldsymbol{\mu}) &= \sum_{q=1}^{Q_f} \theta_f^q(\boldsymbol{\mu}) f_q(v) \qquad \forall v \in V \end{aligned} \tag{3.10}$$

Which in turn leads to the discrete system

$$\begin{aligned} \boldsymbol{A}_h(\boldsymbol{\mu}) &= \sum_{q=1}^{Q_a} \theta_a^q(\boldsymbol{\mu}) \boldsymbol{A}_h^q \\ \boldsymbol{f}_h(\boldsymbol{\mu}) &= \sum_{q=1}^{Q_f} \theta_f^q(\boldsymbol{\mu}) \boldsymbol{f}_h^q \end{aligned} \tag{3.11}$$

Leading to the reduced system

$$\begin{aligned} &\boldsymbol{A}_N(\boldsymbol{\mu}) \boldsymbol{u}_N(\boldsymbol{\mu}) = \boldsymbol{f}_N(\boldsymbol{\mu}) \\ &\boldsymbol{A}_N(\boldsymbol{\mu}) = \sum_{q=1}^{Q_a} \theta_a^q(\boldsymbol{\mu}) \boldsymbol{A}_N^q = \sum_{q=1}^{Q_a} \theta_a^q(\boldsymbol{\mu}) \boldsymbol{V}^T \boldsymbol{A}_h^q \boldsymbol{V} \\ &\boldsymbol{f}_N(\boldsymbol{\mu}) = \sum_{q=1}^{Q_f} \theta_f^q(\boldsymbol{\mu}) \boldsymbol{f}_N^q = \sum_{q=1}^{Q_f} \theta_f^q(\boldsymbol{\mu}) \boldsymbol{V}^T \boldsymbol{f}_h^q \end{aligned} \tag{3.12}$$

This is why affine representation of the stiffness matrix and load vector with respect to different parameters was explored in section 2.3. By doing this the parameter independent matrices $\boldsymbol{A}_N^q$ and vectors $\boldsymbol{f}_N^q$ can be calculated once and stored for assembly of the ROM which is now a fully N-dependent process. The RB-solution can be calculated for any $\boldsymbol{\mu} \in \mathcal{P}$ and transformed back to the high-fidelity domain by

$$\boldsymbol{u}_N^h(\boldsymbol{\mu}) = \boldsymbol{V} \boldsymbol{u}_N(\boldsymbol{\mu}) \tag{3.13}$$

The fact that there exists an affinity as described in (3.10) is a central assumption for developing a reduced order model. It can be seen from (3.11) that the assembly of the high-fidelity system can increase vastly in complexity by increasing Q_a and Q_f. This is because $\boldsymbol{A}_h^q$ and $\boldsymbol{f}_h^q$ are constructed from numerical integration of matrices and vectors of size N_h. Remembering the splitting of the problem in offline and online steps this should not be a problem for the ROM. From (3.12) it is seen that the complexity of Q_a and Q_f are carried over to the assembly of the ROM, but this is in the form of adding matrices and vectors of size N which is computationally negligible up to very high values of Q_a and Q_f assuming a *small* N. Although this is usually not a problem it is key to be aware of that this sets some criterion on how the high-fidelity system is created.

Figure 3.1, inspired by [11], shows the algebraic workflow for the ROM machinery. The process of choosing N has not yet been discussed, but will be introduced in the following and discussed in further detail in Section 3.4.

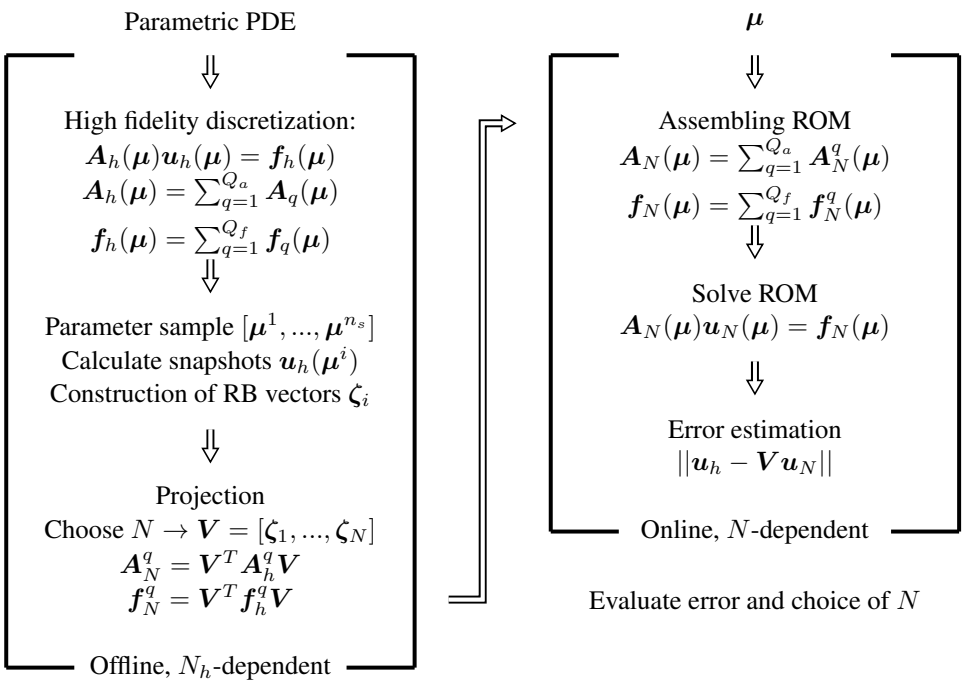


Figure 3.1: ROM machinery

3.3 Proper Orthogonal Decomposition

There are multiple ways to create the reduced basis from which the ROM shall be constructed. For this thesis, proper orthogonal decomposition (POD) was chosen. POD is a numerical technique of compressing and approximating a high-dimensional data set by an orthonormal basis. For the FE-case this means that the original variables $\boldsymbol{u}_h$ (dofs), are transformed into a new set of uncorrelated variables (POD modes), where the first few modes is expected to retain most of the *energy* in the system. Before applying POD to parametric PDEs the concept of singular value decomposition (SVD) is needed.

3.3.1 Singular Value Decomposition

The singular value decomposition is defined for all real matrices, and for a matrix $\boldsymbol{A} \in \mathbb{R}^{m \cdot n}$ reads

$$\begin{aligned}
&\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{Z}^T \\
&\boldsymbol{U} = [\boldsymbol{\zeta_1}|...|\boldsymbol{\zeta_m}] \in \mathbb{R}^{m\cdot m} \quad \boldsymbol{Z} = [\boldsymbol{\Psi_1}|...|\boldsymbol{\Psi_n}] \in \mathbb{R}^{n\cdot n} \\
&\boldsymbol{\Sigma} = diag(\sigma_1, ..., \sigma_r) \in \mathbb{R}^{m\cdot n} \\
&\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0, \quad r \leq min(m, n)
\end{aligned} \tag{3.14}$$

The numbers σ_i are called singular values listed in decreasing order. $\boldsymbol{\zeta}_i$ are the left singular vectors and $\boldsymbol{\Psi}_i$ are the right singular vectors, and both $\boldsymbol{U}$ and $\boldsymbol{Z}$ always form orthogonal sets [21]. The rank of $\boldsymbol{A}$ equals the number of nonzero singular values r, and in the presence of numerical noise the *numerical rank* is taken as the number of singular values larger than some suitable fraction of the first (largest) singular value [22]. From (3.14) the following can be obtained

$$\boldsymbol{A}\boldsymbol{\Psi}_i = \sigma_i\boldsymbol{\zeta}_i \quad \boldsymbol{A}^T\boldsymbol{\zeta}_j = \sigma_j\boldsymbol{\Psi}_j, \quad i, j = 1, ..., r \tag{3.15}$$

The goal of SVD is low rank approximation. For a matrix $\boldsymbol{A} \in \mathbb{R}^{m\cdot n}$ with rank equal to r, it can be written as the sum of r rank-1 matrices based on (3.15)

$$\boldsymbol{A} = \sum_{i=1}^{r} \sigma_i\boldsymbol{\zeta}_i\boldsymbol{\Psi}_i^T \tag{3.16}$$

This becomes very useful as the sum the first $k \leq r$ matrices captures as much of the energy as possible. In this context *energy* refers to either the Frobenius norm or the 2-norm, defined as [23]

$$\begin{aligned}
||\boldsymbol{A}||_F &= \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\sum_{i=1}^{r}\sigma_i^2} \\
||\boldsymbol{A}||_2 &= \sigma_{max}
\end{aligned} \tag{3.17}$$

From (3.16) and the norms (3.17) the following minimization statements of the rank-k approximation of $\boldsymbol{A}$ can be derived.

$$\begin{aligned}
\boldsymbol{A}_k &= \sum_{i=1}^{k} \sigma_i\boldsymbol{\zeta}_i\boldsymbol{\Psi}_i^T \\
||\boldsymbol{A} - \boldsymbol{A}_k||_F &= \min_{\boldsymbol{B}\in\mathbb{R}^{m\cdot n}} ||\boldsymbol{A} - \boldsymbol{B}||_F = \sqrt{\sum_{i=k+1}^{r}\sigma_i^2} \\
||\boldsymbol{A} - \boldsymbol{A}_k||_2 &= \min_{\boldsymbol{B}\in\mathbb{R}^{m\cdot n}} ||\boldsymbol{A} - \boldsymbol{B}||_2 = \sigma_{k+1} \\
rank(\boldsymbol{B}) &\leq k
\end{aligned} \tag{3.18}$$

Meaning there is no better rank-k approximation with respect to the energy of the system.

3.3.2 POD for parameterized problems

The starting point for the POD approach is a parameter sample set $[\boldsymbol{\mu}^1, ..., \boldsymbol{\mu}^{n_s}] \subset \mathcal{P}$ for which high-fidelity solutions $u_h(\boldsymbol{\mu}^i)$ are calculated. The high-fidelity solutions are defined by the degrees of freedom $\boldsymbol{u}_h^{(i)} \in \mathbb{R}^{N_h}$ which in turn define the snapshot matrix $\boldsymbol{S} \in \mathbb{R}^{N_h \cdot n_s}$ as

$$\boldsymbol{S} = [\boldsymbol{u}_h^{(1)} | ... | \boldsymbol{u}_h^{(n_s)}] \tag{3.19}$$

Utilizing SVD (3.14) on $\boldsymbol{S}$ yields

$$\begin{aligned} &\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{Z}^T \\ &\boldsymbol{U} = [\boldsymbol{\zeta}_1 | ... | \boldsymbol{\zeta}_{N_h}] \in \mathbb{R}^{N_h \cdot N_h} \quad \boldsymbol{Z} = [\boldsymbol{\Psi}_1 | ... | \boldsymbol{\Psi}_{n_s}] \in \mathbb{R}^{n_s \cdot n_s} \\ &\boldsymbol{\Sigma} = diag(\sigma_1, ..., \sigma_r) \in \mathbb{R}^{N_h \cdot n_s} \\ &\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0, \quad r \leq min(N_h, n_s) \end{aligned} \tag{3.20}$$

As before r denotes the rank of $\boldsymbol{S}$ and the following relations can be introduced

$$\boldsymbol{S}\boldsymbol{\Psi}_i = \sigma_i \boldsymbol{\zeta}_i \quad and \quad \boldsymbol{S}^T\boldsymbol{\zeta}_i = \sigma_i \boldsymbol{\Psi}_i, \qquad i = 1, ..., r \tag{3.21}$$

$$\boldsymbol{S}^T\boldsymbol{S}\boldsymbol{\Psi}_i = \sigma_i^2 \boldsymbol{\Psi}_i \quad and \quad \boldsymbol{S}\boldsymbol{S}^T\boldsymbol{\zeta}_i = \sigma_i^2 \boldsymbol{\zeta}_i, \quad i = 1, ..., r \tag{3.22}$$

The singular values squared σ_i^2 and singular vectors $\boldsymbol{\Psi}_i$ and $\boldsymbol{\zeta}_i$ are recognized as the eigenvalues and eigenvectors of matrices $\boldsymbol{S}^T\boldsymbol{S} \in \mathbb{R}^{n_s \cdot n_s}$ and $\boldsymbol{S}\boldsymbol{S}^T \in \mathbb{R}^{N_h \cdot N_h}$. The goal now is to obtain the best N-dimensional basis $\boldsymbol{V}$ where the projection error is calculated in the Frobenius norm. The set of all N-dimensional orthonormal bases is introduced as

$$\mathcal{V}_N = \{\boldsymbol{W} \in \mathbb{R}^{N_h \cdot N} : \boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}_N\}$$

and the projection of any vector $\boldsymbol{v} \in \mathbb{R}^{N_h}$ onto span($\boldsymbol{W}$) as

$$\mathbb{P}_W \boldsymbol{v} = \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{v}$$

From SVD the best rank-N approximation of $\boldsymbol{S}$ becomes

$$\boldsymbol{S}_N = \sum_{i=1}^{N} \sigma_i \boldsymbol{\zeta}_i \boldsymbol{\Psi}_i^T \tag{3.23}$$

Inserting (3.21) into to (3.23), and introducing $\boldsymbol{V} = [\boldsymbol{\zeta}_1, ..., \boldsymbol{\zeta}_N]$ yields

$$\boldsymbol{S}_N = \sum_{i=1}^{N} \sigma_i \boldsymbol{\zeta}_i \frac{1}{\sigma_i}(\boldsymbol{S}^T\boldsymbol{\zeta}_i)^T = \sum_{i=1}^{N} \boldsymbol{\zeta}_i(\boldsymbol{\zeta}_i^T\boldsymbol{S}) = \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{S} \tag{3.24}$$

For a matrix $\boldsymbol{A} \in \mathbb{R}^{m \cdot n}$ the Frobenius norm can be rewritten with respect to each column vector $\boldsymbol{a}_i$, yielding $||\boldsymbol{A}||_F^2 = \sum_{i=1}^{n} ||\boldsymbol{a}_i||_2^2$. The sum of squared errors between each snapshot and its projection by a basis $\boldsymbol{W}$ reads

$$\sum_{i=1}^{n_s} ||\boldsymbol{u}_h^{(i)} - \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{u}_h^{(i)}||_2^2 = ||\boldsymbol{S} - \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{S}||_F^2 \tag{3.25}$$

Recalling (3.18) the following minimization statement for the POD basis is obtained

$$||\boldsymbol{S} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{S}||_F^2 = \min_{W \in \mathcal{V}_N} ||\boldsymbol{S} - \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{S}||_F^2 = \sum_{i=N+1}^{r} \sigma_i^2 \tag{3.26}$$

The POD basis is orthonormal by construction, but it is also the minimization of the squares of errors between each snapshot vector $\boldsymbol{u}_h^{(i)}$ and its projection onto the subspace spanned by any N-dimensional orthonormal basis $\boldsymbol{W} \in \mathbb{R}^{N_h \cdot n_s}$.

A POD basis for any $N \leq min(n_s, N_h)$ can now be constructed by the first N left singular vectors. The singular values and vectors is calculated from either $\boldsymbol{S}^T\boldsymbol{S}$ or $\boldsymbol{S}\boldsymbol{S}^T$ as seen from (3.22), and the smallest one should be chosen for the eigenvalue problem to avoid unnecessary computational complexity. This results in the following algorithm

If $n_s \leq N_h$

- Form matrix $\boldsymbol{C} = \boldsymbol{S}^T\boldsymbol{S} \in \mathbb{R}^{n_s \cdot n_s}$
- Solve eigenvalue problems $\boldsymbol{C}\boldsymbol{\Psi}_i = \sigma_i^2\boldsymbol{\Psi}_i, \quad i = 1, ..., r$
- Set $\boldsymbol{\zeta}_i = \frac{1}{\sigma_i}\boldsymbol{S}\boldsymbol{\Psi}_i$
- $\boldsymbol{V} = [\boldsymbol{\zeta}_1, ..., \boldsymbol{\zeta}_N]$

Else if $N_h < n_s$

- Form matrix $\boldsymbol{C} = \boldsymbol{S}\boldsymbol{S}^T \in \mathbb{R}^{N_h \cdot N_h}$
- Solve eigenvalue problem $\boldsymbol{C}\boldsymbol{\zeta}_i = \sigma_i^2\boldsymbol{\zeta}_i, \quad i = 1, ..., r$
- $\boldsymbol{V} = [\boldsymbol{\zeta}_1, ..., \boldsymbol{\zeta}_N]$

As the need for a ROM usually arises when there is a high number of dofs, $n_s \leq N_s$ is more likely to be the case. How the number N is chosen is described further in section 3.4, but a sufficient number of modes should be present to capture the majority of the systems energy.

With this approach a reduced basis can be constructed for any snapshot matrix $\boldsymbol{S}$ minimizing the projection error in the Frobenius norm, however a more general approach is desirable for constructing a reduced basis minimizing different norms.

3.3.3 POD with respect to energy inner product

The snapshot functions $u_h(\boldsymbol{\mu}^i)$ belong to the space $V_h \subset V$, and it becomes natural to seek a POD basis minimizing the norm defined by the inner product of V which is usually referred to as the *energy* norm and *energy* inner product . This is done by introducing a matrix

$$\begin{aligned}(\boldsymbol{X}_h) &= (\varphi^i, \varphi^j)_V, \quad \boldsymbol{X}_h \in \mathbb{R}^{N_h \cdot N_h} \\ ||v_h||_V^2 &= \boldsymbol{v}^T \boldsymbol{X}_h \boldsymbol{v}, \quad \forall v_h \in V_h\end{aligned} \tag{3.27}$$

For a FE problem this is the same norm as defined in section 2.1.2, and the inner product is the same as $a(.,.)$. In general other inner products may be used instead as long as it is bounded by the original one. For linear elastic problems where the a-form is dependent only on the first derivatives the H1-seminorm can be chosen to describe $\boldsymbol{X}_h$

$$(\boldsymbol{X}_h)_{ij} = \int_A \nabla \phi^i \nabla \phi^j d\Omega \tag{3.28}$$

The problem becomes finding a N-dimensional basis $\boldsymbol{V} \in \mathcal{V}_N^{\boldsymbol{X}_h}$ where

$$\mathcal{V}_N^{\boldsymbol{X}_h} = \{\boldsymbol{W} \in \mathbb{R}^{N_h \cdot N} : \boldsymbol{W}^T \boldsymbol{X}_h \boldsymbol{W} = \boldsymbol{I}_N\} \tag{3.29}$$

which should minimize the error between each snapshot and its $\boldsymbol{X}_h$-orthogonal projection onto the subspace spanned by $\boldsymbol{W}$. The $\boldsymbol{X}_h$-orthogonal projection of any $\boldsymbol{v} \in \mathbb{R}^{N_h}$ onto span($\boldsymbol{W}$) is written as

$$\mathbb{P}_W^{X_h} \boldsymbol{v} = \boldsymbol{W}\boldsymbol{W}^T \boldsymbol{X}_h \boldsymbol{v} \tag{3.30}$$

yielding the minimization statement

$$\sum_{i=1}^{n_s} ||\boldsymbol{u}_h^{(i)} - \mathbb{P}_V^{\boldsymbol{X}_h} \boldsymbol{u}_h^{(i)}||_{\boldsymbol{X}_h}^2 = \min_{\boldsymbol{W} \in \mathcal{V}_N^{\boldsymbol{X}_h}} \sum_{i=1}^{n_s} ||\boldsymbol{u}_h^{(i)} - \mathbb{P}_W^{\boldsymbol{X}_h} \boldsymbol{u}_h^{(i)}||_{\boldsymbol{X}_h}^2 \tag{3.31}$$

The norm in the minimization statement (3.31) is transformed in the following

$$\begin{aligned}&\sum_{i=1}^{n_s} ||\boldsymbol{u}_h^{(i)} - \boldsymbol{W}\boldsymbol{W}^T \boldsymbol{X}_h \boldsymbol{u}_h^{(i)}||_{\boldsymbol{X}_h}^2 \\ =&\sum_{i=1}^{n_s} ||\boldsymbol{X}_h^{\frac{1}{2}} \boldsymbol{u}_h^{(i)} - \boldsymbol{X}_h^{\frac{1}{2}} \boldsymbol{W}\boldsymbol{W}^T \boldsymbol{X}_h \boldsymbol{u}_h^{(i)}||_2^2 = ||\boldsymbol{X}_h^{\frac{1}{2}} \boldsymbol{S} - \boldsymbol{X}_h^{\frac{1}{2}} \boldsymbol{W}\boldsymbol{W}^T \boldsymbol{X}_h \boldsymbol{S}||_F^2\end{aligned} \tag{3.32}$$

The transformation to the Frobenius norm is introduced to make use of the low rank approximation obtained by SVD (3.18). The approach for finding a POD basis is similar to what was done in the previous section, but SVD is done one an altered snapshots matrix, $\tilde{\boldsymbol{S}} = \boldsymbol{X}^{\frac{1}{2}} \boldsymbol{S}$

$$\begin{aligned}
&\tilde{\boldsymbol{S}} = \tilde{\boldsymbol{U}}\boldsymbol{\Sigma}\tilde{\boldsymbol{Z}}^T \\
&\tilde{\boldsymbol{U}} = [\tilde{\boldsymbol{\zeta}}_1|...|\tilde{\boldsymbol{\zeta}}_{N_h}] \in \mathbb{R}^{N_h \cdot N_h} \quad \tilde{\boldsymbol{Z}} = [\tilde{\boldsymbol{\Psi}}_1|...|\tilde{\boldsymbol{\Psi}}_{n_s}] \in \mathbb{R}^{n_s \cdot n_s} \\
&\boldsymbol{\Sigma} = diag(\sigma_1, ..., \sigma_r) \in \mathbb{R}^{N_h \cdot n_s} \\
&\sigma_1 \geq, \sigma_2 \geq ... \geq \sigma_r \geq 0, \quad r \leq min(N_h, n_s)
\end{aligned} \tag{3.33}$$

Some useful relations are introduced

$$\begin{aligned}
&\tilde{\boldsymbol{S}}^T\tilde{\boldsymbol{S}}\tilde{\boldsymbol{\Psi}}_i = \sigma_i^2\tilde{\boldsymbol{\Psi}}_i, \quad \tilde{\boldsymbol{S}}^T\tilde{\boldsymbol{S}} = \boldsymbol{S}^T\boldsymbol{X}_h\boldsymbol{S} \\
&\tilde{\boldsymbol{S}}\tilde{\boldsymbol{S}}^T\tilde{\boldsymbol{\zeta}}_i = \sigma_i^2\tilde{\boldsymbol{\zeta}}_i, \qquad \tilde{\boldsymbol{S}}\tilde{\boldsymbol{S}}^T = \boldsymbol{X}_h^{\frac{1}{2}}\boldsymbol{S}\boldsymbol{S}^T\boldsymbol{X}_h^{\frac{1}{2}}
\end{aligned} \tag{3.34}$$

Again, the singular values squared σ_i^2 and the singular vectors $\tilde{\boldsymbol{\zeta}}_i$ and $\tilde{\boldsymbol{\Psi}}_i$ can be found from solving eigenvalue problems. The POD basis is now found as $\boldsymbol{V} = [\boldsymbol{X}_h^{-\frac{1}{2}}\tilde{\boldsymbol{\zeta}}_1|...|\boldsymbol{X}_h^{-\frac{1}{2}}\tilde{\boldsymbol{\zeta}}_N]$, with $N \leq r$. Substituting in $\tilde{\boldsymbol{S}} = \boldsymbol{X}_{\boldsymbol{h}}^{\frac{1}{2}}\boldsymbol{S}$, $\tilde{\boldsymbol{W}} = \boldsymbol{X}_{\boldsymbol{h}}^{\frac{1}{2}}\boldsymbol{W}$, $\tilde{\boldsymbol{V}} = \boldsymbol{X}_{\boldsymbol{h}}^{\frac{1}{2}}\boldsymbol{V}$ and (3.32) into (3.31), and again recalling (3.18) yields the following minimization statement

$$||\tilde{\boldsymbol{S}} - \tilde{\boldsymbol{V}}\tilde{\boldsymbol{V}}^T\tilde{\boldsymbol{S}}||_F^2 = \min_{\boldsymbol{W} \in \mathcal{V}_N^{X_h}} ||\tilde{\boldsymbol{S}} - \tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^T\tilde{\boldsymbol{S}}||_F^2 = \sum_{i=N+1}^{r} \sigma_i^2 \tag{3.35}$$

A N-dimensional POD basis for any $N \leq min(n_s, N_h)$ can now be constructed by the first N column vectors $\boldsymbol{\zeta}_i$. As in the previous section, matrix dimensions should be taken into consideration resulting in the following algorithm

If $n_s \leq N_h$

- Form matrix $\tilde{\boldsymbol{C}} = \boldsymbol{S}^T\boldsymbol{X}_h\boldsymbol{S}$
- Solve eigenvalue problem $\tilde{\boldsymbol{C}}\tilde{\boldsymbol{\Psi}}_i = \sigma_i^2\tilde{\boldsymbol{\Psi}}_i, \quad i = 1, ..., r$
- Set $\boldsymbol{\zeta}_i = \boldsymbol{X}_h^{-\frac{1}{2}}\tilde{\boldsymbol{\zeta}}_i = \frac{1}{\sigma_i}\boldsymbol{S}\tilde{\boldsymbol{\Psi}}_i$
- $\boldsymbol{V} = [\boldsymbol{\zeta}_1, ..., \boldsymbol{\zeta}_N]$

Else if $N_h \leq n_s$

- Form matrix $\tilde{\boldsymbol{C}} = \boldsymbol{X}_h^{\frac{1}{2}}\boldsymbol{S}\boldsymbol{S}^T\boldsymbol{X}_h^{\frac{1}{2}}$
- Solve eigenvalue problem $\tilde{\boldsymbol{C}}\tilde{\boldsymbol{\zeta}}_i = \sigma_i^2\tilde{\boldsymbol{\zeta}}_i, \quad i = 1, ..., r$
- Set $\boldsymbol{\zeta}_i = \boldsymbol{X}_h^{-\frac{1}{2}}\tilde{\boldsymbol{\zeta}}_i$
- $\boldsymbol{V} = [\boldsymbol{\zeta}_1, ..., \boldsymbol{\zeta}_N]$

As noted earlier $n_s \leq N_h$ is most likely the case, and this also has the beneficial property of not having to form matrices $\boldsymbol{X}_h^{\frac{1}{2}}$ and $\boldsymbol{X}_h^{-\frac{1}{2}}$. With this approach a reduced basis can be constructed for any snapshot matrix $\boldsymbol{S}$ minimizing the norm described by $\boldsymbol{X}_h$.

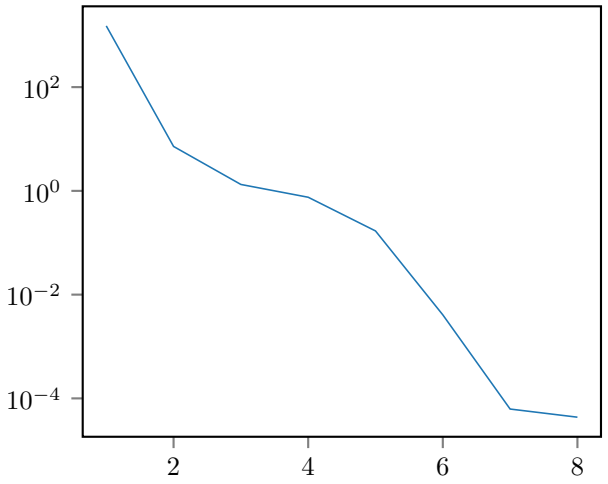


Figure 3.2: Spectrum of singular values

3.4 Retaining modes and error analysis

The last step in construction of the reduced basis and in turn the reduced model is determining the number of POD modes which should be retained, and evaluating the overall performance of the ROM. Among other things this means choosing the number N, the number of retained POD modes. This is done after solving the eigenvalue problem, when all singular values σ_i and column vectors $\boldsymbol{\zeta}_i$ have been calculated. Just as the construction of a FE model requires expertise by the analyst in order to construct a robust and usable high-fidelity model, the same can be said for the construction of ROMs. Setting the high-fidelity model and parameter sampling aside, the construction of the reduced model up to this point has been completely algorithmic. However the choice of which modes to retain requires key insight by the analyst to obtain a valid ROM.

3.4.1 Singular value spectrum analysis

The first step in this process should be to analyze the spectre of singular values σ_i. This is done because it gives a quick and easy estimate of which POD modes should be retained in the reduced basis. The singular values are listed in decreasing order, and from the minimization statements, (3.26) and (3.35), it is clear that the projection error reduces with the sum of squared singular values not included in the basis. This means that POD modes associated with *small* singular values can usually be neglected as they do not capture much of the energy in the system. To evaluate the threshold for neglecting values, a good approach is to plot the spectrum of singular values. The singular values on the logarithmic y-axis is plotted against their index number i on the x-axis in Figure 3.2.

As mentioned earlier a central assumption behind the POD approach is that most of the energy in the system would be retained by the first few POD modes. This should be reflected

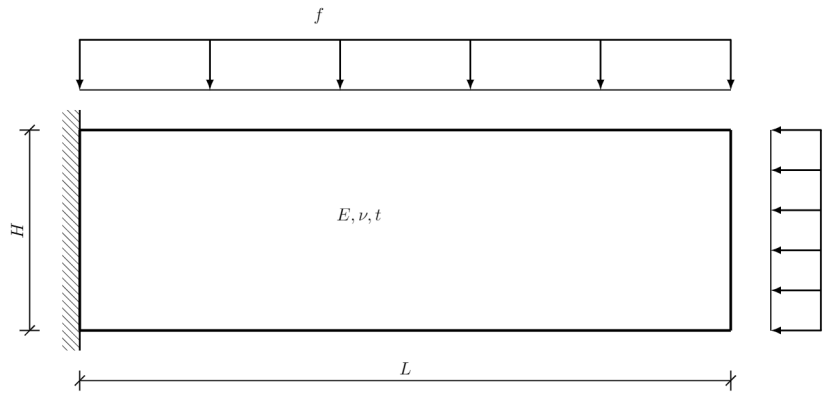


Figure 3.3: 2D beam with distributed and axial load

in the spectre of singular values where the values are expected to drop of after the first few indexes. From this the analyst hopes to find a threshold value for which all singular values below this threshold and their corresponding POD modes are neglected.The expected behaviour is seen in Figure 3.2 as the trailing singular values are negligible compared to the first few. This is a good estimate of which modes to retain, and the overall performance of the ROM, but this is not sufficient on its own to determine which modes to retain.

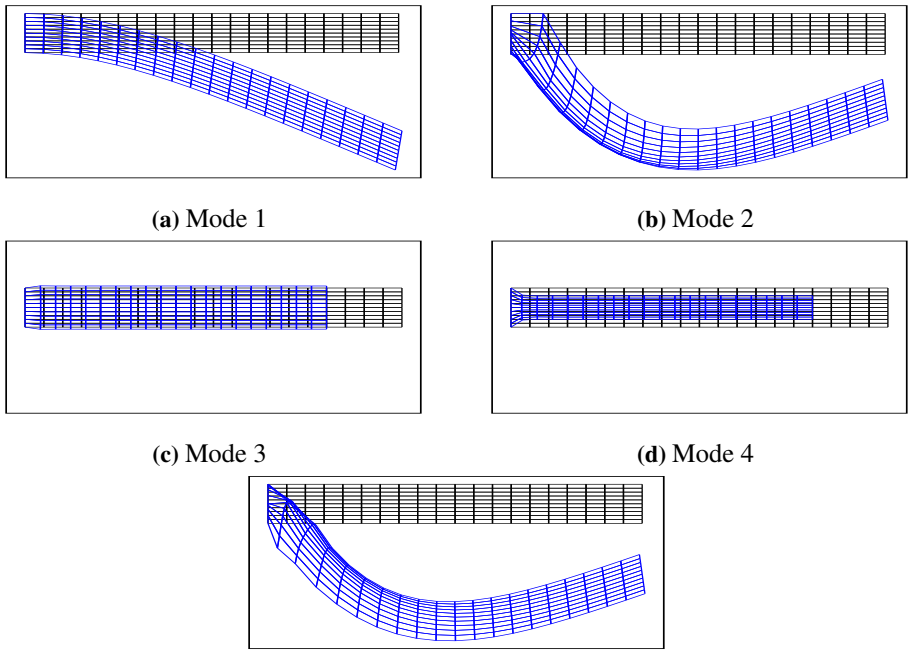
3.4.2 POD Mode analysis

Each column vector ζ_i in the reduced basis corresponds to a POD mode, and describes this mode by the high-fidelity degrees of freedom. This means that each column vector in the reduced basis can be compared to the responses of the high-fidelity model. This is further explained in terms of a linear elasticity.

For a linear elastic problems the dofs of the system are the nodal displacements, which in turn means that the POD modes make up different displacement patterns. For a 2D beam as seen in Figure 3.3, illustrating a beam with distributed and axial load, the response consist of bending and axial deformation. Therefor the POD modes should make up both bending axial deformation patterns. Plotting the POD modes is no different from plotting the solution from a high-fidelity calculation and should usually be a simple process, and an example is given in Figure 3.4. Insight in the expected response can be used to control the validity of the POD modes. Listed below are two attributes that should be checked.

- Can the POD modes describe the high-fidelity response sufficiently?
- Does there exist POD modes containing large amounts of numerical noise?

The first point refers to the fact that if a ROM should be able to represent a type of behaviour it must be present in the reduced basis. For the beam in Figure 3.3 it is obvious that the reduced basis must contain both bending and axial deformation modes. Typical for linear elastics is that the energy related to bending is much higher than for axial deformation and if this is not treated axial modes can easily be neglected by the POD approach.



(a) Mode 1 **(b)** Mode 2

(c) Mode 3 **(d)** Mode 4

(e) Mode 5

Figure 3.4: Illustration of mode shapes for a 2D beam

As seen in Figure 3.4 modes 1, 2 and 5 are bending modes while modes 3 and 4 are axial modes. An important fact to note is that the response of a ROM is the combination of all retained POD modes and the response from a single POD mode might look non-physical even though it captures important behavior of the model. This can be seen by mode 4 as it displays symmetric axial and transverse contraction which by itself looks faulty with the given load case, but still proves to increase the accuracy of the model.

Numerical noise is always something to look out for dealing with numerical approximation methods. Assuming that the high-fidelity response is free of numerical noise seeing numerical noise in the POD modes is a warning sign. The numerical noise can stem from different places in the ROM construction, but it should be investigated as it leads to poorer performance, and usually modes affected by numerical problems will be omitted. Figure 3.5 illustrates a POD mode displaying numerical noise, and such behaviour is undesirable to retain in the reduced basis. It is however not uncommon that faulty modes containing large amounts of noise can occur for the smallest singular values due to round-off error.

The process of analyzing the POD modes becomes increasingly difficult by increasing complexity in the high-fidelity response. Many modes and different types of behaviour can be difficult comprehend. However, seeing the *physical* behaviour of each POD mode can be of great value for evaluating the reduced basis, and should always be done to some extent.

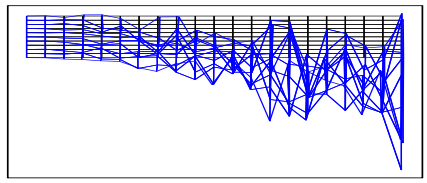


Figure 3.5: Mode with numerical noise

3.4.3 Error analysis

The last and most important evaluation of a ROM is of course analyzing the error between RB-solutions and high-fidelity solutions. There are different ways to go about this, but for this thesis the following approach for error analysis was chosen.

As both the high-fidelity model and ROM are discrete systems on the form $\boldsymbol{Ad} = \boldsymbol{F}$, where displacements $\boldsymbol{d}$ are the unknowns, the energy of each system can be calculated by

$$E = \frac{1}{2}\boldsymbol{d}^T \boldsymbol{A}\boldsymbol{d} \tag{3.36}$$

and the relative error of the ROM with respect to the high-fidelity model calculated in the energy norm can be found as

$$\epsilon = \sqrt{\frac{|E_h - E_N|}{E_h}} \tag{3.37}$$

Remembering the parameter sample set $[\boldsymbol{\mu}^1, ..., \boldsymbol{\mu}^{n_s}]$ and snapshot matrix $\boldsymbol{S} = [\boldsymbol{u}_h^{(1)}, ..., \boldsymbol{u}_h^{(n_s)}]$, the energy of the high-fidelity snapshots can easily be found as

$$E_h(\boldsymbol{\mu}^i) = \frac{1}{2}(\boldsymbol{u}_h^{(i)})^T \boldsymbol{A}_h \boldsymbol{u}_h^{(i)} \tag{3.38}$$

The ROM is then solved for each parameter vector $\boldsymbol{\mu}^i$ and the energy is found as

$$E_N(\boldsymbol{\mu}^i) = \frac{1}{2}(\boldsymbol{u}_N^{(i)})^T \boldsymbol{A}_N \boldsymbol{u}_N^{(i)} \tag{3.39}$$

The relative error can now be calculated for each parameter sample $\epsilon(\boldsymbol{\mu}^i)$ to give an overview of the performance. A more interesting quantity however is the averaged aggregated error

$$\epsilon_{ag} = \frac{1}{n_s}\sum_{i=1}^{n_s} \epsilon(\boldsymbol{\mu}^i) \tag{3.40}$$

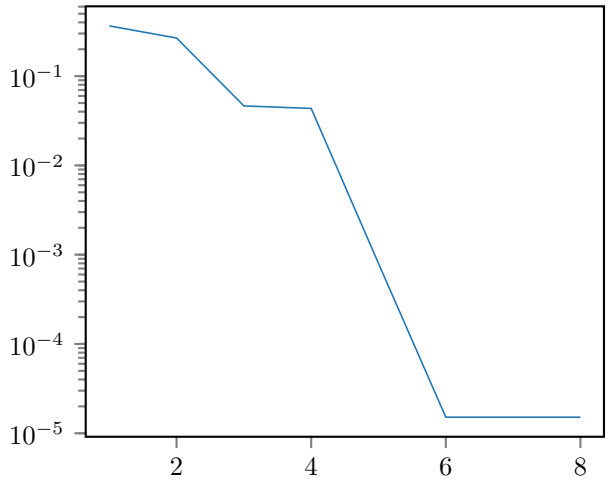


Figure 3.6: Averaged error plotted against number of retained modes

To decide which modes to retain, successive ROMs are constructed by increasing the number of retained modes i. Plotting the averaged aggregated error against the number of retained modes i, as seen in Figure 3.6 makes for a great visualization of the contribution from each mode. From this the performance of the ROM becomes quite obvious, and the final decision on which modes to include can be made. In line with the assumption of POD, including the first few modes should capture most of the energy in the system, and just as for the spectrum of singular values the error is expected to drop under a tolerable value after including the first few modes.

As the ROM was constructed by minimizing the projection error of the snapshots the measure of accuracy obtained above cannot be ensured for all parameter vectors $\boldsymbol{\mu} \in \mathcal{P}$. Generally the error for parameter vectors not corresponding with the sample set will be higher than those of the sample set. Therefore the error calculated by this method should be taken as a lower bound, and viewed as *best performance*. In spite of this, the aforementioned approach of error analysis is usually sufficient for evaluating the ROM. A more general approach would be to make a new sample of parameters including parameter vectors not included in the original sample, and evaluating the averaged aggregated error. A key benefit however of the first approach is that it makes use of the already calculated snapshots, meaning that high-fidelity solutions need not be calculated. This is not the case for the latter approach.

The main idea of ROMs is that a small reduction in accuracy is traded for a large reduction in computational complexity. It is however worth mentioning that the accuracy of the high-fidelity model should be controlled before starting the construction of a reduced model. As the reduced model introduces more error, starting of with a model that is already not sufficiently accurate will in all likelihood lead to a bad approximation of the original problem. A proposed rule of thumb is that the high-fidelity model should have a relative error $\epsilon_h \leq 1\%$ in order to *enable* a relative error of the reduced model $\epsilon_N \leq 10\%$.

Enable is a key word as even though the high-fidelity model might be sufficiently accurate this does not ensure satisfactory performance from the reduced model.

3.5 Parameter sampling

As discussed earlier in this chapter the accuracy of a ROM is highly dependent on the snapshots. The snapshots themselves are dependent on the set of parameter samples, $\{\boldsymbol{\mu}^1, ..., \boldsymbol{\mu}^{n_s}\} \subset \mathcal{P}$, and these parameter samples are selected by the analyst. Assuming p different parameters $\boldsymbol{\mu}^i \in \mathbb{R}^p$ the goal of the parameter sample becomes spanning a p-dimensional space as accurate and efficiently as possible with respect to the high-fidelity solution. Although the computational cost of constructing the ROM is assumed to be of little interest it is beneficial to seek snapshots that will increase the accuracy, and it would be highly ineffective to calculate more high-fidelity solutions than necessary. There are several different methods for parameter sampling, and only a brief overview of three methods is given here. In Figure 3.7 the three methods are illustrated in a 2-dimensional parameter space for visibility.

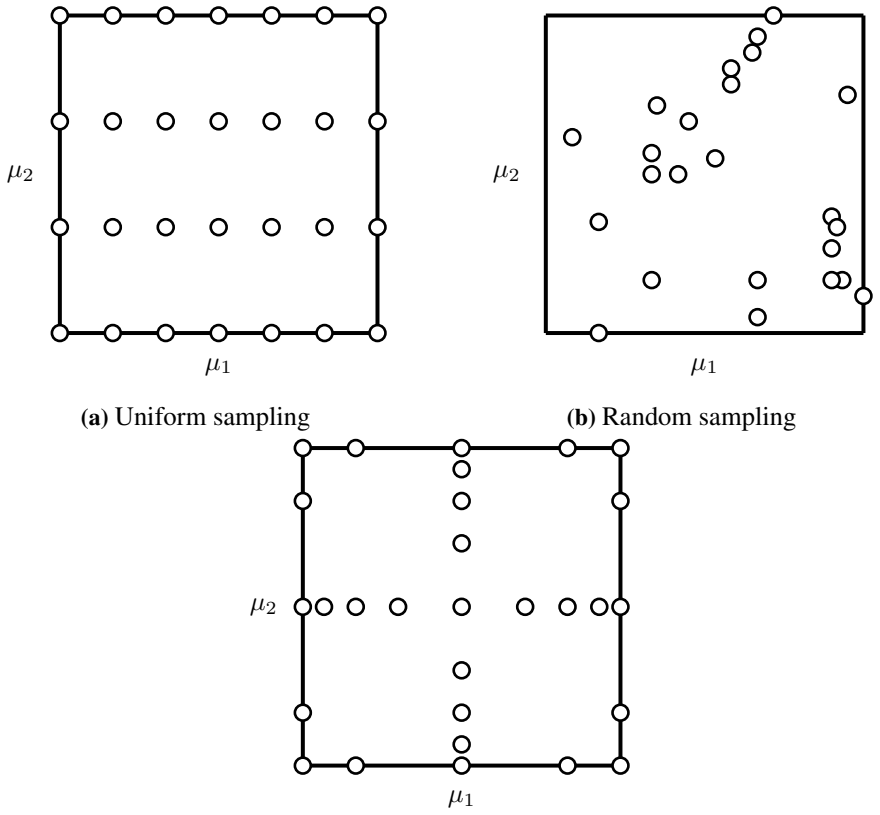
The easiest method is uniform sampling in each dimension of the parameter space. This methods works for moderate parameter spaces, $p \leq 5$, as the number of samples grows exponentially making this method highly inefficient for higher order parameter spaces. This method is the easiest to implement and works for simpler problems. The number of samples in each dimension should be chosen to capture parameter dependencies sufficiently. This method was chosen for the numerical studies of this thesis.

Another method to be used is random sampling of the parameter domain. This can either be done by *fully* random sampling, or by some *near* random method like Latin hypercube sampling. Usually the latter method is preferred as a fully random sample can produce very misleading results, and a poor span of the entire parameter space.

The last method introduced here is the use of sparse grid quadrature, which is a set of advanced sampling methods enabling high-dimensional parameter spaces to be spanned efficiently with relatively few sampling points. These methods seek to overcome the curse of dimensionality and the approach is theoretically based on combining tensors products of one-dimensional quadrature rules [24]. The use of more advanced sampling methods becomes necessary for more complex problems and the choice of sampling method is a critical part of the construction of reduced models.

Since the response is directly dependent on the parameter sample there is always a chance that some important parameter dependencies can be neglected. However this risk can be mitigated by knowing the characteristics of the problem of interest and choosing a suitable sampling method.

For low dimensional parameter spaces and problems where the parameter dependencies is thought to be of lower polynomial order, uniform sampling can safely be used. The number of the sampling points in each direction is a critical decision where a higher density of sampling points should be used in directions and intervals where parameter sensitivity is thought to be higher.



(a) Uniform sampling

(b) Random sampling

(c) Sparse sampling

Figure 3.7: Illustration of different sampling methods in a 2-dimensional parameter space

4 | Numerical studies

In this chapter the theory of FEM and ROM is applied for two linear elastic example problems. The main focus is on geometric variation, but material variation is also present.

The examples was implemented using the Python library Nutils [25], and a simple FE solver was constructed for linear elastic 2D problems assuming plane strain for each of the examples. The solver maps quadrilateral shapes to a reference square where a uniform mesh of rectangles are used, and the field is approximated by bilinear interpolation functions. This approach is equal to that of using Q4 elements. Although Q4-elements display a disadvantageous behaviour in bending due to spurious shear strain this mesh was chosen to simplify computer implementation as accuracy of the high-fidelity solutions is not a critical result for this thesis. The FE-solver is therefore not as accurate as one would want for solving an actual problem, but it has the necessary properties for applying the theory of reduced order models. The complete python code for both examples can be found in Appendix B.

For the full order model to be reducible when geometric parameters are studied, the different geometries has to be mapped to the same mesh in the natural coordinate system. The approach for each of the examples becomes choosing a mesh, with $n_{e,\xi}, n_{e,\eta}$ number of elements in ξ and η direction respectively, and then defining the mapping between the physical and natural coordinates. An instance of a mesh in the natural coordinate system is seen in Figure 4.1

4.1 Example 1 - Rectangle

4.1.1 Problem description and modelling

For the first example a rather simple geometric shape was chosen, as seen in Figure 4.2, and it is recognized as the linear elastic response of a rectangular cantilever beam with a transverse normal load. For this example the parameters of interest where chosen to be the Young's modulus, and the height and length of the beam. Meanwhile the load intensity and Poisson ratio was chosen to be kept constant, $f = 0.8\frac{N}{mm}$ and $\nu = 0.3$. This lead to the following parameter space

$$\begin{aligned} &\boldsymbol{\mu} \in \mathbb{R}^3 \\ &\mu_1 = E = [10, 90] && GPa \\ &\mu_2 = x_4 = L = [2000, 4000] && mm \\ &\mu_3 = y_4 = H = [150, 250] && mm \end{aligned} \tag{4.1}$$

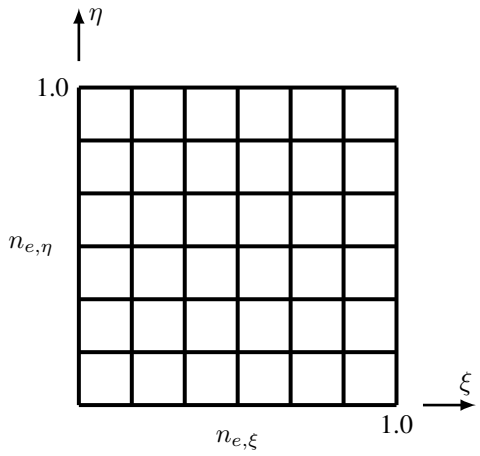


Figure 4.1: Mesh in natural coordinate system

Uniform sampling in each direction in the parameter space was used, with 5 samples for each parameter resulting in $n_s = 5^3 = 125$ parameter samples which the high-fidelity system must be solved for. As the shape of the beam remains a rectangle through the geometric variation there is no distortion to speak of between the physical and reference geometry. For the high-fidelity solution this means that there is no loss of accuracy through irregular geometry.

Mapping as described in section 2.1.3 and 2.3.3 is introduced, and the geometry from Figure 4.2 leads to the following relations between coordinates

$$\begin{aligned} x_1 = x_3 = y_1 = y_2 = 0 \\ x_4 = x_2, \quad y_4 = y_3 \end{aligned} \tag{4.2}$$

This means that the geometry is uniquely described by two parameters $\boldsymbol{\mu}_G = [x_4, y_4]$, inserting this in (2.44) and (2.46) the mapping is obtained as

$$\begin{aligned} \boldsymbol{C}(\boldsymbol{\mu}) = \begin{bmatrix} x_4 & 0 \\ 0 & y_4 \end{bmatrix}, \quad \boldsymbol{S}(\boldsymbol{\mu}) = \begin{bmatrix} x_4 - x_2 \\ y_4 - y_3 \end{bmatrix} = \mathbf{0} \\ \boldsymbol{J} = \boldsymbol{C}(\boldsymbol{\mu}), \qquad J = det\boldsymbol{C}(\boldsymbol{\mu}) = x_4 y_4 = L \cdot H \\ \boldsymbol{J}^{-1} = \boldsymbol{C}(\boldsymbol{\mu})^{-1} = \frac{1}{det\boldsymbol{C}(\boldsymbol{\mu})} \begin{bmatrix} y_4 & 0 \\ 0 & x_4 \end{bmatrix} = \begin{bmatrix} x_4^{-1} & 0 \\ 0 & y_4^{-1} \end{bmatrix} \end{aligned} \tag{4.3}$$

Introducing this to (2.13) the following expression for derivatives with respect to Cartesian coordinates is derived

$$\begin{aligned} \frac{\partial}{\partial x} = \frac{1}{L}\frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial y} = \frac{1}{H}\frac{\partial}{\partial \eta} \end{aligned} \tag{4.4}$$

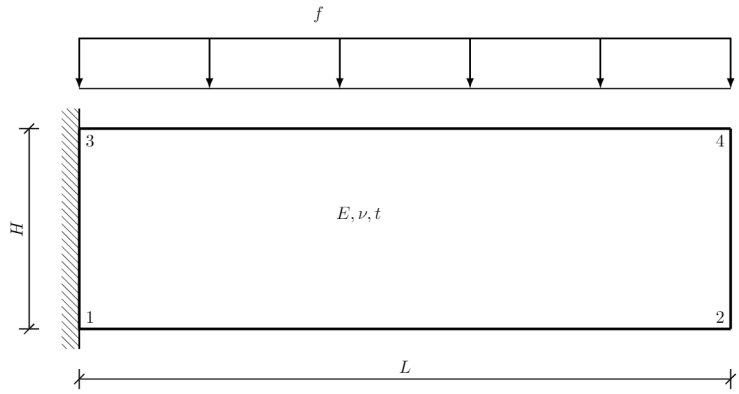


Figure 4.2: Example 1 - Physical model

The same relations is used for transforming the integrals

$$\begin{aligned} d\Omega &= \partial x \partial y = J \partial \xi \partial \eta \\ d\Gamma_i: \quad &\partial x = L \partial \xi, \quad \partial y = H \partial \eta \end{aligned} \tag{4.5}$$

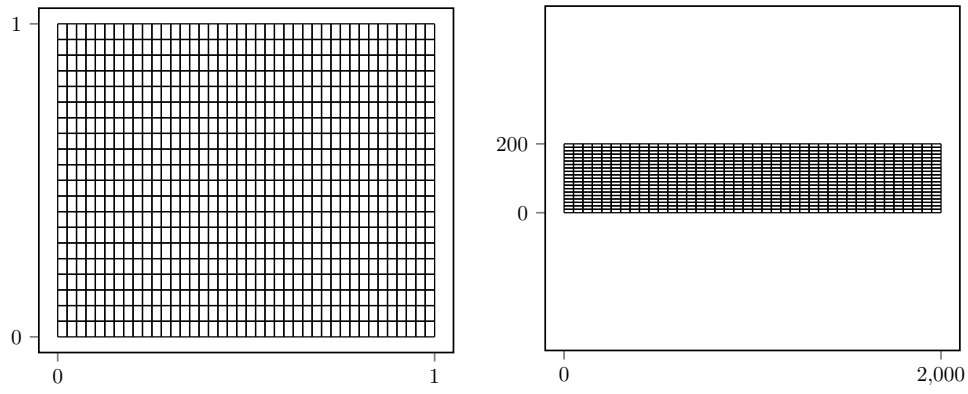
As seen from (4.4) and (4.5) the mapping is simply based on scaling the axis with the parameters $\boldsymbol{\mu}_G$. Combining this with the discrete equations for the stiffness matrix (2.36), the expression for $\boldsymbol{A}_h$ can be written as a linear combination

$$\begin{aligned} \boldsymbol{A}_h &= \sum_{i=1}^{4} \theta_i(\boldsymbol{\mu}) \boldsymbol{A}_i = \sum_{i=1}^{4} \theta_i(\boldsymbol{\mu}) \int_\Omega \boldsymbol{I}_i d\partial\xi\partial\eta \\ \theta_1(\boldsymbol{\mu}) &= \frac{H}{L} \begin{bmatrix} \lambda + 2\mu & 0 \\ 0 & \mu \end{bmatrix} & \boldsymbol{I}_1 &= \boldsymbol{N}_{0,\xi} \boldsymbol{N}_{0,\xi} \\ \theta_2(\boldsymbol{\mu}) &= \begin{bmatrix} 0 & \lambda \\ \mu & 0 \end{bmatrix} & \boldsymbol{I}_2 &= \boldsymbol{N}_{0,\xi} \boldsymbol{N}_{0,\eta} \\ \theta_3(\boldsymbol{\mu}) &= \begin{bmatrix} 0 & \mu \\ \lambda & 0 \end{bmatrix} & \boldsymbol{I}_3 &= \boldsymbol{N}_{0,\eta} \boldsymbol{N}_{0,\xi} \\ \theta_4(\boldsymbol{\mu}) &= \frac{L}{H} \begin{bmatrix} \mu & 0 \\ 0 & \lambda + 2\mu \end{bmatrix} & \boldsymbol{I}_4 &= \boldsymbol{N}_{0,\eta} \boldsymbol{N}_{0,\eta} \end{aligned} \tag{4.6}$$

As the four integrals $\boldsymbol{I}_i$ can be calculated without knowing $\boldsymbol{\mu}$, assembly of the stiffness matrix is simply a linear combination of constant matrices $\boldsymbol{I}_i$ multiplied with the scaling functions $\theta_i(\boldsymbol{\mu})$, $\forall \boldsymbol{\mu} \in \mathcal{P}$. This has to be implemented in the FE solver, and although not being particularly challenging this diverges from the general construction of the stiffness matrix.

The load vector can be found from implementing the mapping (2.37)

$$\boldsymbol{f} = \int_0^1 f \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{N}_0^T \end{bmatrix} L d\xi, \quad \eta = 1 \tag{4.7}$$



(a) Reference geometry with mesh **(b)** Instance of physical geometry with mesh

Figure 4.3: Example 1 - Reference and physical geometry with mesh

Where f is the load intensity.

As the parameters space and sample (4.1), stiffness matrix (4.6) and load vector (4.7) are defined, a ROM can be constructed following the approach of Chapter 3. A mesh with $(40, 20)$-elements in ξ and η direction was chosen. The problem before reduction consisted of $N_h = 1722$ dofs and $n_s = 125$ parameter samples, and in Figure 4.3 the reference geometry and an instance of the physical geometry is plotted.

4.1.2 Results

A reduced basis was calculated by the POD approach for energy inner product as described in section 3.3.3. This resulted in a reduced basis with 64 modes which would create a ROM with 64 dofs. Even though this might seem as vast reduction from 1 722 dofs the reduced basis should be analyzed to only consist of the most contributing modes, and the performance must also be controlled.

The spectrum of the first eight singular values is plotted in Figure 4.4. From this it can be seen that the singular values quickly decreases in magnitude. This is in line with the first modes capturing most of the energy. It is however not enough to conclude which modes should be retained.

Next the POD modes is analyzed. The first eight modes can be found in Appendix A, and in Figure 4.5 the first four modes are plotted. Since the geometry varies the modes are plotted in the natural coordinate system against the reference geometry. As can be seen all modes display different bending dependent displacement patterns, but the fourth mode displays some unexpected behaviour, which is also true for all trailing modes. The plotted modes are vastly scaled for visibility, but the behaviour seen in fourth mode is a warning sign and this is related to numerical issues due to the stiffness of the boundary conditions.

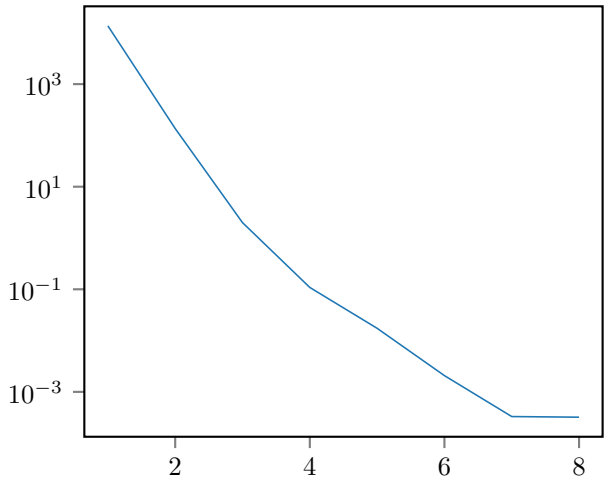


Figure 4.4: Example 1 - Singular value spectrum

From this, one would want to only retain the first three modes.

From studying similar problems without geometric variation it should be noted that the dominating modes corresponds with mode 1 and 3, while mode 2 is not present. This implies that the geometric variation introduces the need for this mode, but does not mean that mode 1 and 3 are independent of the geometric variation.

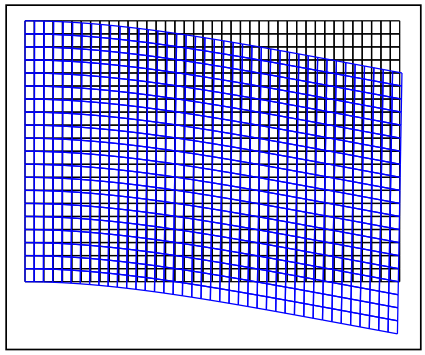
To check the accuracy of the reduced model the averaged aggregated error in the energy norm is calculated for RB's with different numbers of retained modes. This is seen in Figure 4.6 and there is now sufficient information to decide which modes should be retained in the basis.

From analyzing the spectrum of singular values, the plotted POD modes and the error plot, it becomes natural to create a reduced basis by including the first three modes. This should lead to a sufficiently accurate reduced model, as the error is bounded $\epsilon = \mathcal{O}(10^{-3})$. It is worth repeating that this is the error relative to the high-fidelity model, and that accuracy of the ROM relative to the original problem in general will be worse. For visualization a high-fidelity solution is plotted over a RB solution in Figure 4.7, and there is no visible difference between the two.

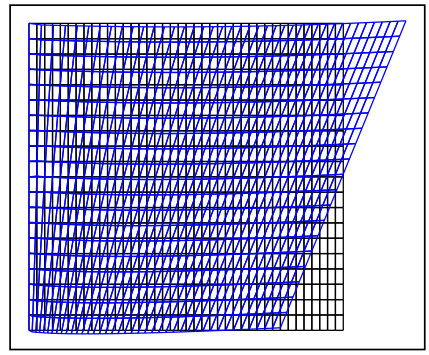
By choosing a RB consisting of the first three POD modes a ROM can be constructed with only 3 dofs. This leads to a reduction from 1 722 to 3, or 574:1.

4.1.3 Discussion

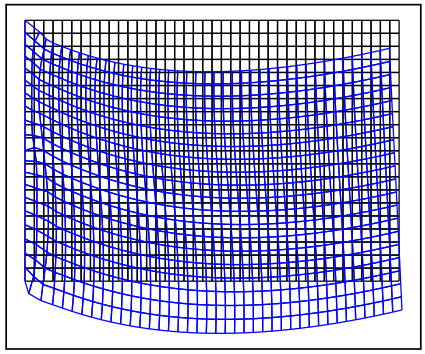
From this first example a ROM was created from the snapshots of a high-fidelity model. This was done by retaining the first three POD modes. The accuracy of this reduced model proved to be quite good as error bounded $\epsilon = \mathcal{O}(10^{-3})$ usually is sufficient for most practical problems.



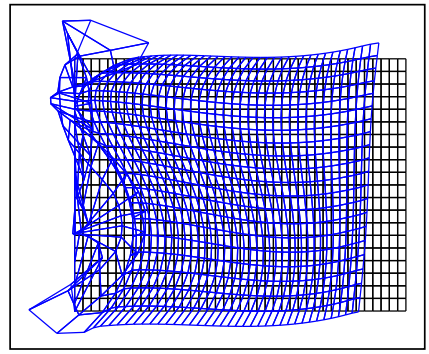
(a) Example 1 - Mode 1



(b) Example 1 - Mode 2



(c) Example 1 - Mode 3



(d) Example 1 - Mode 4

Figure 4.5: Example 1 - First four POD modes

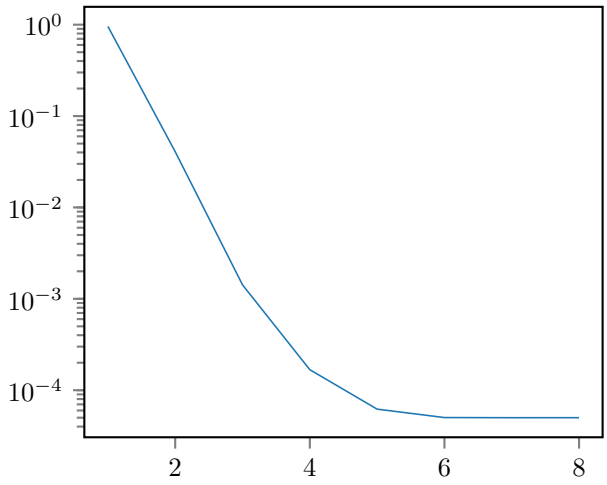


Figure 4.6: Example 1 - Averaged aggregated error

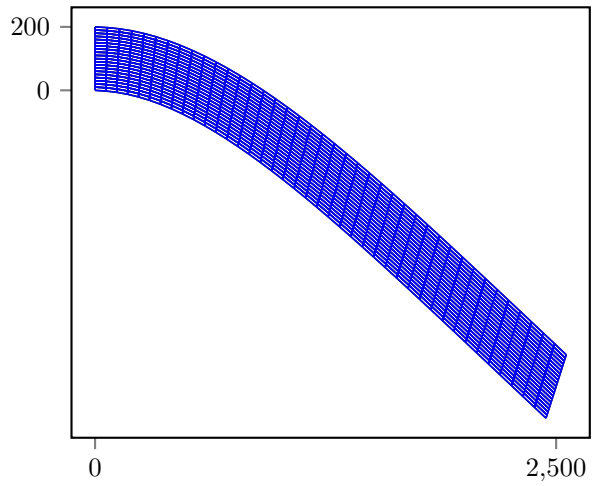
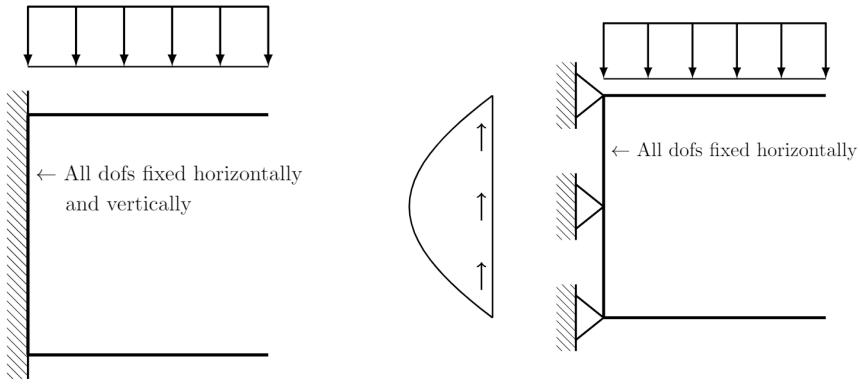


Figure 4.7: Example 1 - High-fidelity vs. RB solution



(a) Completely stiff end

(b) Less rigid implementation

Figure 4.8: Two alternatives of implementing boundary conditions

Overall the ROM behaved just as expected as the singular values dropped of in magnitude, the POD modes displayed bending patterns, and only including the first few modes captured most of the systems energy.

Also important to note is the behaviour of the latter modes, which all displayed unexpected behaviour close to the fixed boundary as can be seen from Mode 4. The significance of these modes turned out to be negligible, it is however of interest why this behaviour occurs. From analyzing different parameter spaces it became evident that this issue did not stem from load and material variation, but emerged when geometric variation was considered. As changing the height and length of the beam affects the equilibrium at the fixed end, the faulty modes is recognized as the reduced basis not being able to correctly capture this effect. Two implementations of the fixed end was explored to overcome this problem, the first being restraining all dofs at the left boundary horizontally and vertically. The second was done by applying a parabolic vertical load, equal to the loading of the beam, together with horizontal clamping and only fixing some of the dofs in the vertical direction. Both implementations can be seen in Figure 4.8. Although the second approach gave better results overall, the first being overly stiff, the undesired behaviour in Mode 4 and trailing was still present. Compared to similar problems it should be noted that a total of 64 POD modes is higher than expected. This means that not only has this problem lead to erroneous modes, but a high number of them. This becomes an interesting result from geometrical variation in ROMs even though the modes where not of significance in this example. It is quite possible that there exists ways to implement the fixed boundary which do not lead to the erroneous modes, however this implies that the geometric variation introduces restrictions on how BC's should be implemented, and this is not seen when other parameters are considered.

The reduction factor of 574:1 can be viewed as not that impressing, but this also stems from the fact that the high-fidelity model was constructed with a fairly small number of dofs. For larger models the reduction will usually be more significant. The reduced model

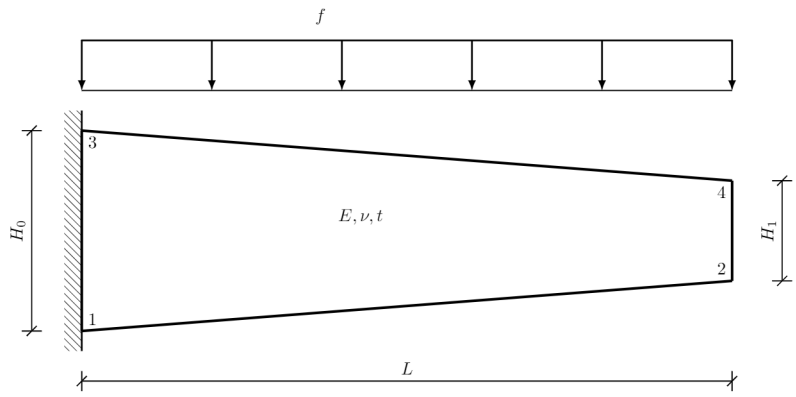


Figure 4.9: Example 2 - Physical model

itself ended up consisting only of three dofs which is very small and gives for extremely fast solving of the reduced problem. When the number of dofs is in this range the practical difference in computing time of retaining or neglecting a few extra modes is usually insignificant.

The computer implementation of the affine expression of the problem proved to be quite easy as all parameter dependencies were linear. The assembly of the ROM was dependent on four matrices making the assembly computationally effortless. A key takeaway however is that although this being quite easy it diverges from the general implementation in regular FE-software. The affinity demands the construction be split in four matrices which makes the use existing FE-solvers difficult as they only construct and save the complete matrix. For an example as easy as this, with just three parameters each with a linear dependency on the stiffness matrix, it would be possible to obtain the parameter independent matrices by interpolating between stiffness matrices with different input parameters. However this is not possible when parametric dependencies becomes many and of higher order.

4.2 Example 2 - Trapezoid

4.2.1 Problem description and modelling

For this example a cantilever beam shaped as a trapezoid was chosen as shown in Figure 4.9. In this example the Young's modulus and the height at the end of the cantilever are the parameters, while the load intensity and Poisson ratio are kept constant, $f = 0.8\frac{N}{mm}$ and $\nu = 0.3$, as well as the other geometric properties $L = 3000mm$ and $H_0 = 350mm$. This lead to the following parameter space

$$
\begin{aligned}
&\boldsymbol{\mu} = \mathbb{R}^2 \\
&\mu_1 = E = [35, 65] \qquad GPa \\
&\mu_2 = H_1 = [150, 350] \quad mm \\
&\theta_0(\boldsymbol{\mu}) = \mu_2 - H_0
\end{aligned}
\tag{4.8}
$$

Uniform sampling in each direction in the parameter space was used, with 4 samples in each direction resulting in $n_s = 4^2 = 16$ parameter samples which the high-fidelity system must be solved for. When $H_1 = 350mm$ the beam is shaped as a rectangle, but as $H_1 \to 150mm$ the shape of the beam transforms into an increasingly steep trapezoid and the physical geometry is distorted from the reference square. This leads to a loss of accuracy for the high-fidelity solutions. The variable θ_0 is introduced to ease notation later on.

Introducing the mapping from section 2.1.3 and 2.3.3 the geometry in Figure 4.9 leads to the following relations between coordinates

$$
\begin{aligned}
&x_1 = x_3 = y_1 = 0 \\
&x_4 = x_2 = L \\
&y_2 = \frac{1}{2}(H_0 - H_1) = \frac{-\theta_0}{2} \\
&y_3 = H_0 \\
&y_4 = \frac{1}{2}(H_0 + H_1) = \frac{\theta_0}{2} + H_0
\end{aligned}
\tag{4.9}
$$

For this example there is only one geometric parameter, $\boldsymbol{\mu}_G = H_1$, as opposed to the two from Example 1, but in the following it becomes obvious that the mapping in this problem is more complex despite having fewer geometric parameters. By introducing this into (2.44), (2.45) and (2.46) we obtain the matrices $\boldsymbol{C}(\boldsymbol{\mu})$, $\boldsymbol{S}(\boldsymbol{\mu})$ and $\boldsymbol{t}(\boldsymbol{\mu})$ as

$$
\begin{aligned}
\boldsymbol{C}(\boldsymbol{\mu}) &= \begin{bmatrix} x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} = \begin{bmatrix} L & \frac{-\theta_0}{2} \\ 0 & H_0 \end{bmatrix} \\
\boldsymbol{S}(\boldsymbol{\mu}) &= \begin{bmatrix} x_4 - x_2 - x_3 \\ y_4 - y_2 - y_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \theta_0 \end{bmatrix} \\
\boldsymbol{C}(\boldsymbol{\mu})^{-1} &= \frac{1}{LH_0} \begin{bmatrix} H_0 & \frac{\theta_0}{2} \\ 0 & L \end{bmatrix} \\
\boldsymbol{t}(\boldsymbol{\mu}) &= \frac{\theta_0}{H_0} \begin{bmatrix} 0 & 1 \end{bmatrix}
\end{aligned}
\tag{4.10}
$$

This allows us to write the Jacobian matrix and its determinant as

$$
\begin{aligned}
\boldsymbol{J} &= \begin{bmatrix} L & \theta_0(\eta - \frac{1}{2}) \\ 0 & H_0 + \theta_0 \xi \end{bmatrix} \\
J &= L(H_0 + \xi\theta_0)
\end{aligned}
\tag{4.11}
$$

And the inverse relations can be found as

$$\begin{aligned}\boldsymbol{J}^{-1} &= \frac{1}{J}\begin{bmatrix} H_0+\theta_0\xi & \theta_0(\frac{1}{2}-\eta) \\ 0 & L \end{bmatrix} \\ &= \frac{1}{L}\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \frac{R}{LH_0}\begin{bmatrix} 0 & \theta_0(\frac{1}{2}-\eta) \\ 0 & L \end{bmatrix} \\ R &= \frac{1}{1+\theta_0\frac{\xi}{H_0}} = \frac{1}{J}LH_0\end{aligned} \tag{4.12}$$

With the inverse relations at hand together with (2.13), the derivatives with respect to Cartesian coordinates is derived

$$\begin{aligned}\frac{\partial}{\partial x} &= \frac{1}{L}\frac{\partial}{\partial \xi} + \frac{R\theta_0(\frac{1}{2}-\eta)}{LH_0}\frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial y} &= \frac{R}{H_0}\frac{\partial}{\partial \eta}\end{aligned} \tag{4.13}$$

The expressions above in (4.13) are obviously more complex than those for Example 1 seen in (4.4). This complexity increases during the construction of the stiffness matrix therefore some notation should be introduced before moving on.

$$\begin{aligned}&\frac{\partial \boldsymbol{N}_0}{\partial x} = \frac{1}{L}\frac{\partial \boldsymbol{N}_0}{\partial \xi} + \frac{R\theta_0(\frac{1}{2}-\eta)}{LH_0}\frac{\partial \boldsymbol{N}_0}{\partial \eta} = \boldsymbol{B}_1 + \theta_0\boldsymbol{B}_2 R \\ &\frac{\partial \boldsymbol{N}_0}{\partial y} = \frac{R}{H_0}\frac{\partial \boldsymbol{N}_0}{\partial \eta} = \boldsymbol{B}_3 R \\ &\boldsymbol{B}_i \in \mathbb{R}^n\end{aligned} \tag{4.14}$$

As seen from (2.36) the stiffness matrix is dependent on four submatrices which can now be expressed as

$$\begin{aligned}&\int_\Omega \boldsymbol{N}_{0,x}\boldsymbol{N}_{0,x}Jd\Omega = \\ &\quad LH_0\int_\Omega(\boldsymbol{B}_1\boldsymbol{B}_1 + \theta_0(\frac{\xi}{H_0}\boldsymbol{B}_1\boldsymbol{B}_1 + \boldsymbol{B}_2\boldsymbol{B}_1 + \boldsymbol{B}_2\boldsymbol{B}_1) + \theta_0^2 R\boldsymbol{B}_2\boldsymbol{B}_2)d\Omega \\ &\int_\Omega \boldsymbol{N}_{0,x}\boldsymbol{N}_{0,y}Jd\Omega = LH_0\int_\Omega(\boldsymbol{B}_1\boldsymbol{B}_3 + \theta_0 R\boldsymbol{B}_2\boldsymbol{B})d\Omega \\ &\int_\Omega \boldsymbol{N}_{0,y}\boldsymbol{N}_{0,x}Jd\Omega = LH_0\int_\Omega(\boldsymbol{B}_3\boldsymbol{B}_1 + \theta_0 R\boldsymbol{B}_3\boldsymbol{B}_2)d\Omega \\ &\int_\Omega \boldsymbol{N}_{0,y}\boldsymbol{N}_{0,y}Jd\Omega = LH_0\int_\Omega R\boldsymbol{B}_3\boldsymbol{B}_3 d\Omega\end{aligned} \tag{4.15}$$

The load vector can be found by introducing the mapping to (2.37), but as the expression does not contain any derivation, the load vector is instead obtained by considering a projection load yielding

$$\boldsymbol{f} = \int_0^1 f \begin{bmatrix} \mathbf{0} \\ \boldsymbol{N_0^T} \end{bmatrix} \cdot \sqrt{L^2 + \frac{\theta_0^2}{4}} d\xi, \quad \eta = 1 \tag{4.16}$$

By utilizing the behavior of geometric series [12], R in (4.12) can be written as

$$\begin{aligned} R &= \frac{1}{1 + \frac{\theta_0}{H_0}\xi} = \sum_{i=0}^{\infty} (-1)^i (\frac{\theta_0}{H_0}\xi)^i \\ for \quad &|\frac{\theta_0}{H_0}\xi| < 1 \end{aligned} \tag{4.17}$$

R is represented as a geometric series to enable an affine representation of the submatrices above with respect to the geometric parameter, but as some symmetrical conditions have been enforced it is reduced from what was seen in (2.47). For computer implementation a finite number n is chosen so that $R_n \approx R$. This ads major complexity to the construction of the stiffness matrix which is illustrated with the last submatrix in (4.15)

$$\int_\Omega \boldsymbol{N}_{0,y} \boldsymbol{N}_{0,y} J d\Omega = \sum_{i=0}^{n} (-1)^i (\frac{\theta_0}{H_0})^i L H_0 \int_\Omega \xi^i \boldsymbol{B}\boldsymbol{B} d\Omega \tag{4.18}$$

Meaning that for this submatrix alone, n scaling functions $\theta_i = (-1)^i (\frac{\theta_0}{H_0})^i L H_0$ and n matrices $\boldsymbol{A}_i = \int_\Omega \xi^i \boldsymbol{B}\boldsymbol{B} d\Omega$ has to be calculated and stored. By doing this for all four submatrices the assembly becomes quite cumbersome. The final assembly can be slightly simplified by adding together all matrices with equal scaling functions as the assembly should consist only of a set of unique scaling functions, however this assembly has transformed quite a bit from the original expression (2.36). As for Example 1 this has to be implemented in the FE solver.

Remembering (3.11) it is clear that the offline step becomes much slower, and also from (3.12) the computational cost of assembling the ROM must be checked. This will only be problematic if a very large number of terms in R is needed to achieve sufficient accuracy. The relative error of R_n is plotted against n and for different heights H_1 in Figure 4.10. The error is calculated against the exact R for $\xi = 1$ as this maximizes the error.

Introducing the series approximation of R_n leads to a change in the stiffness matrix meaning the high-fidelity problem has been altered, but from Figure 4.10 it can be seen that a sufficient accuracy can be obtained for manageable values of n. By analyzing the error plot it can be seen that the approximation looses accuracy as the geometry is being increasingly distorted from the reference geometry, $H_1 \to 150mm$. This will influence the quality of the ROM relative to the original problem and even though a large number of sampling heights are used the accuracy of increasingly distorted geometry is highly dependent on R_n. To ensure an approximation error of R_n bounded $\epsilon_R = \mathcal{O}(10^{-4})$, $n = 20$ was chosen. This lead to the assembly of the stiffness matrix being a summation of 120 parameter independent matrices.

Again, as the parameter space (4.8), stiffness matrix (4.15) and load vector (4.16) are defined, a ROM can now be constructed. The second alternative for implementation of

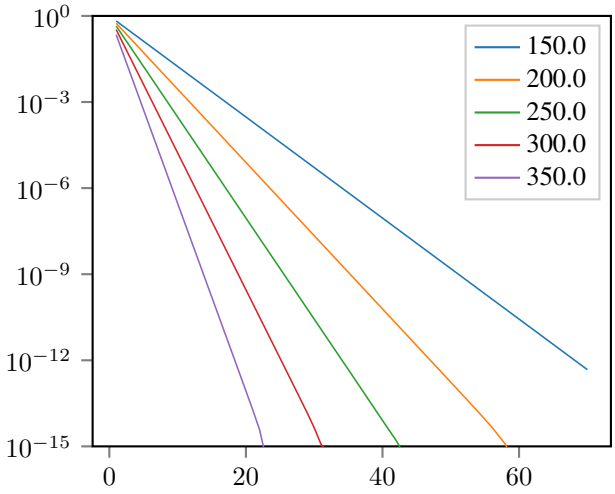


Figure 4.10: Example 2 - Relative error of R_n

the fixed end, as seen in Figure 4.8, was implemented also for this example. A mesh with $(40, 20)$-elements in ξ and η direction was chosen. The problem before reduction consisted of $N_h = 1722$ dofs and $n_s = 16$ parameter samples, and in Figure 4.11 the reference geometry and an instance of the physical geometry is plotted.

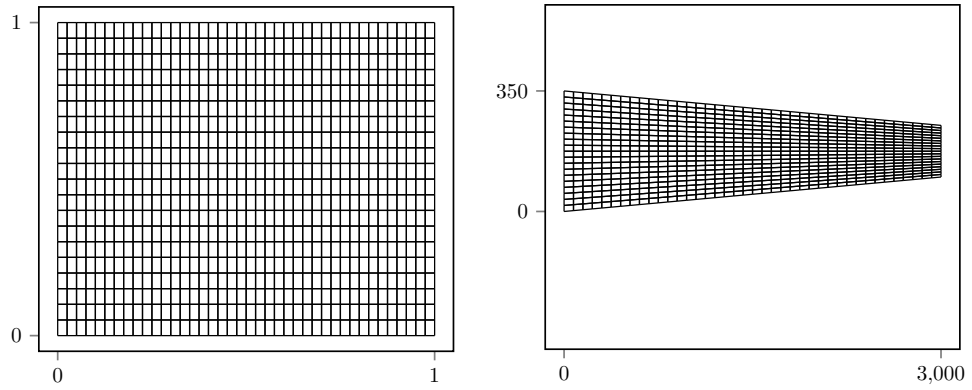
4.2.2 Results

A reduced basis was calculated by the POD approach with respect to the energy inner product. This resulted in a reduced basis with 10 modes, which would give ROM with 10 dofs, but again the reduced basis should be analyzed so that only the most critical modes are retained.

The spectrum of the first 8 singular values is plotted in Figure 4.12. The singular values quickly drop to negligible values which indicates that a well performing ROM can be constructed by the first few modes, which modes specifically will be decided in the following.

As for the first example the POD modes are plotted in the natural coordinate system. All the first 8 modes can be found in Appendix A, and in Figure 4.13 the first four modes can be seen. All modes display bending patterns as expected, and it can be seen that the modes occur in an increasing hierarchy of bending patterns. There is no numerical issues or unexpected behaviour in any of the modes, meaning they can all be applied in the reduced basis.

To control the accuracy of the model the averaged aggregated error in the energy norm is calculated for RB's with different number of retained modes, and this is plotted Figure 4.14. There is now sufficient information to decide which modes should be retained in the reduced basis.



(a) Reference geometry with mesh **(b)** Instance of physical geometry with mesh

Figure 4.11: Example 2 - Reference and physical geometry with mesh

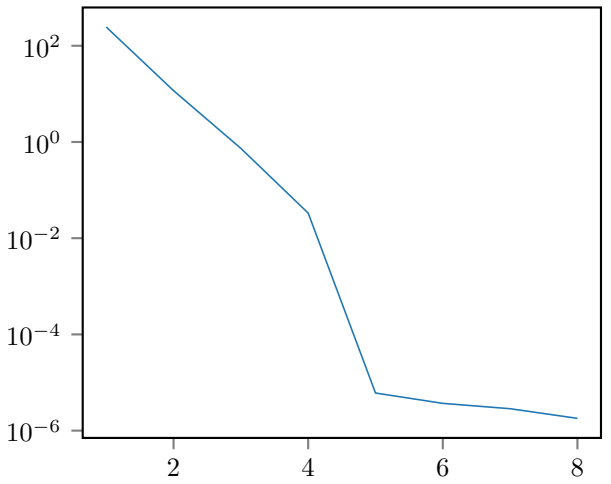
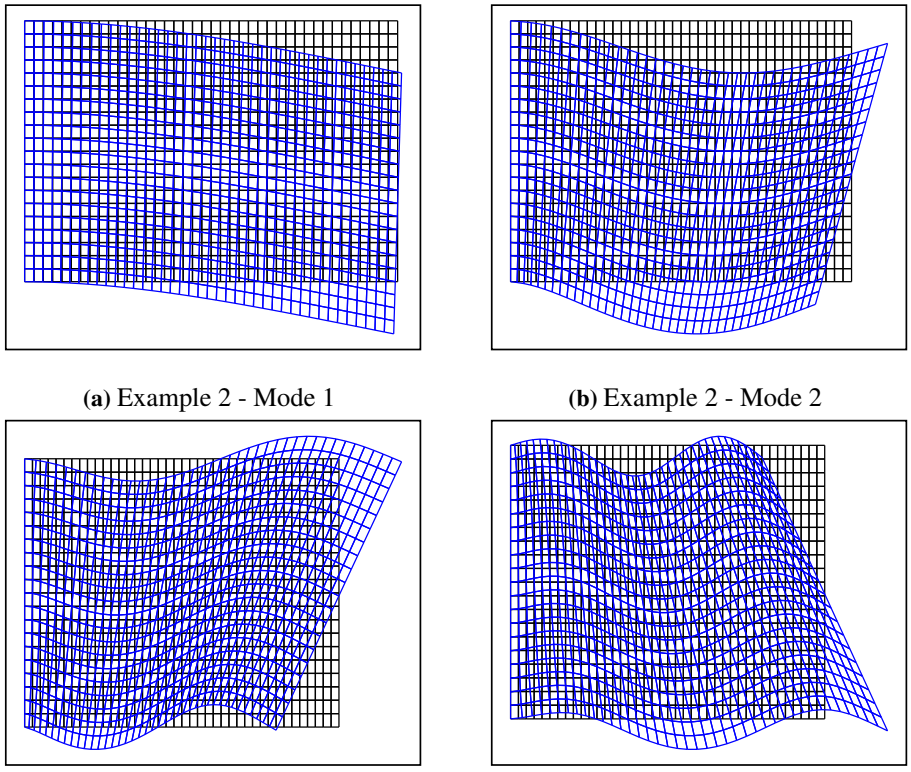


Figure 4.12: Example 2 - Singular value spectrum



(a) Example 2 - Mode 1

(b) Example 2 - Mode 2

(c) Example 2 - Mode 3

(d) Example 2 - Mode 4

Figure 4.13: Example 2 - First four POD modes

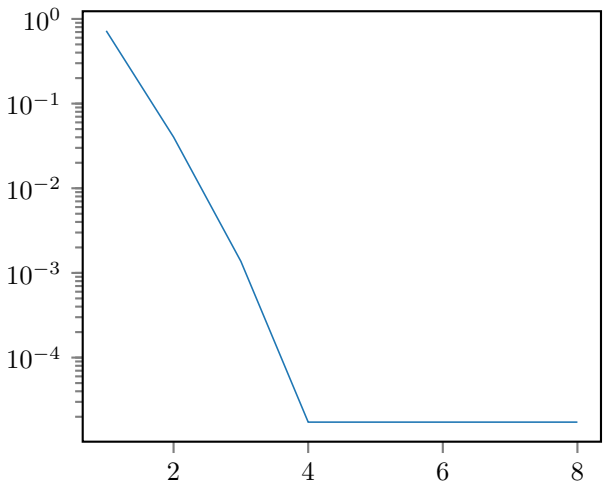


Figure 4.14: Example 2 - Averaged aggregated error

From analyzing the spectrum of singular values, the plotted POD modes, and the error plot it becomes natural to create a reduced basis by including the first four modes. One could argue that an acceptable accuracy is achieved already for retaining just the two first modes, and that including the two trailing modes effectively doubles the matrix size. However, there is no practical difference between solving matrix equations of such small sizes as they are extremely fast. Therefore the added accuracy of retaining all four modes is easily chosen as there is no added benefit of retaining more modes. All the trailing modes are higher order bending patterns which are not activated for the given loading and boundary conditions, and can therefore be neglected. The reduced basis should result in a sufficiently accurate model , as the error is bounded $\epsilon = \mathcal{O}(10^{-4})$. For visualization a high-fidelity solution is plotted over a RB solution in Figure 4.15, and there is no visible difference between the two.

By choosing a RB consisting of these modes a ROM can be constructed with 4 dofs. This leads to a reduction from 1 722 to 4, or 431:1.

4.2.3 Discussion

A ROM has been constructed for this example by retaining the first four POD modes. The accuracy of this model proved to be quite satisfactory as an error bounded $\varepsilon = \mathcal{O}(10^{-4})$ is sufficient for most practical problems.

The construction of the ROM went just as expected as the singular values dropped of in magnitude, the POD modes displayed bending patterns, and including the first few modes captured most of the systems energy.

As opposed to the first example the POD modes did not display any unwanted behaviour, the difference being the boundary remained the same for all instances of the geometry.

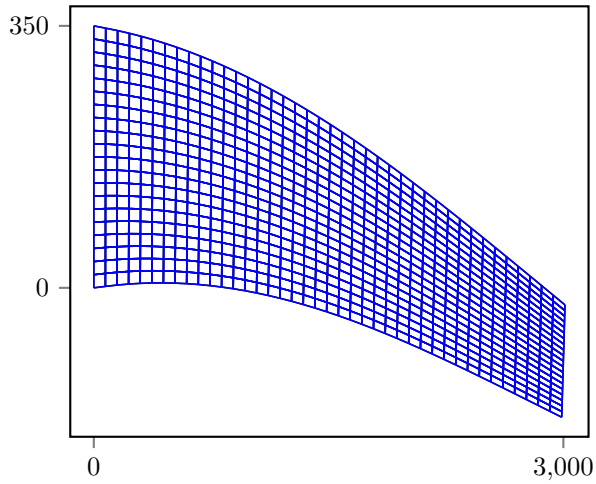


Figure 4.15: Example 2 - High-fidelity vs. RB solution

This means that every mode could be included in the basis if need be.

Just as for Example 1 the reduction factor, 430:1, can be viewed as not that impressing, but as the original number of dofs where low this is not surprising. Even though the reduction factor was unimpressive a ROM was constructed with a low number of modes which is exactly what the POD approach aims to do.

For this example problem it became more evident that the affine expression of the system can be problematic. The high-fidelity model and the ROM matrices was created by a linear combination of 120 matrices each. This made for much more cumbersome computer implementation than for the general case and Example 1. The computer code can be found in Appendix B. For this thesis the necessary expressions was obtained through hand calculation although it should be possible to obtain the same results through symbolic manipulation of the original expression, but this is not as straight forward. It should be noted that this representation of the system matrices would not easily be achieved with the use of general FE-solvers as base point as the final result is quite complex. This again underlines the fact that affine representation for complex parameter dependencies renders the need for custom assembly of the high-fidelity system matrices and vectors.

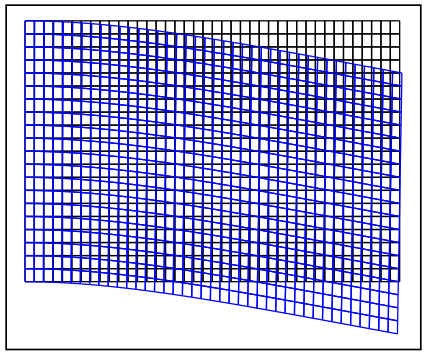
5 | Conclusion

After conducting numerical studies on two examples of linear elastic problems with geometric variation, two well performing reduced order models was constructed. Both models behaved well in line with the assumptions and expectations of reduced order models through Galerkin projection, and dislayed how computational cost can be greatly reduced by constructing reduced order models. The models where robust with respect to both material and geometric parameters, but there are however two main findings, each emerging in separate examples.

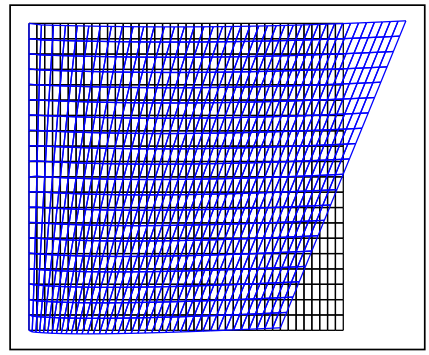
Numerical issues was present in the first example. Although a robust reduced model was constructed, it became evident that the effect of geometric variation on the fixed boundary was not captured without problem by the reduced basis. This presented itself in the form of a high number of erroneous modes in the reduced basis, and can be seen in the fourth mode in Figure 4.5. However this problem was not present in the dominant modes, meaning a valid reduced model could still be obtained. The significance of this would be of interest to analyze in further work, as it influence the applicability of ROMs on more complex and general problems than that of this thesis. The effect of this behaviour should be studied both in terms of different implementations of boundary conditions in order to avoid this problem, as well as for complicated systems containing several structural elements and boundaries. Being robust with respect to parametric variation is one of the key attributes should reduced order models become a viable approach for parametric problems.

Affine representation of the weak statement with respect to the parameters of interest, which is a key assumption for construction of reduced order models, turned out to be a major challenge in the second example. This determines how the system matrices and vectors of the full order model, Finite element model in this thesis, is assembled and in turn also how the reduced model is assembled. The finite element model is intricately dependent on geometric parameters resulting in a complex expression in order to achieve the aforementioned affinity. This had little effect on the assembly of the reduced model, but made a major difference for the full order model. As the construction diverged highly from that of the original problem, it made for a highly customized Finite element solver for this example. This underlines how the use of existing Finite element software can be challenging depending on the parameters of interest. For reduced order models to be viable with respect to geometric parameters a key challenge in further work becomes obtaining a general way of deriving the weak statement affinely dependent on all geometric parameters of the problem, without a prohibitive computational cost. Software which is able to express the system matrices and vectors of a problem affinely dependent on the parameters would significantly lower the threshold for applying reduced order models.

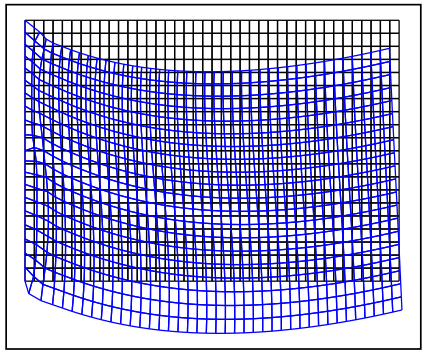
A Appendix 1



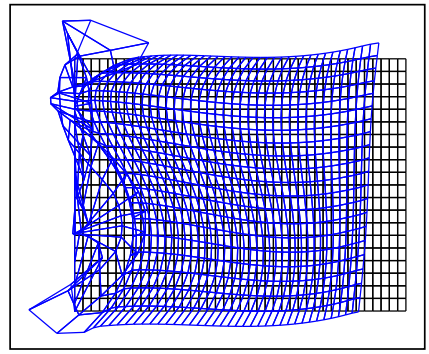
(a) Example 1 - Mode 1



(b) Example 1 - Mode 2

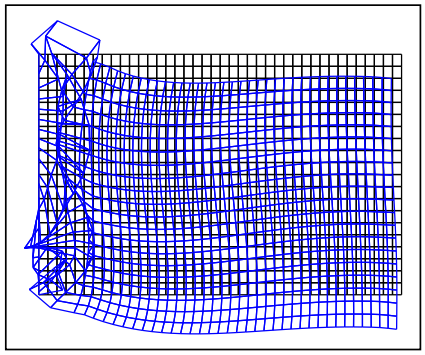


(c) Example 1 - Mode 3

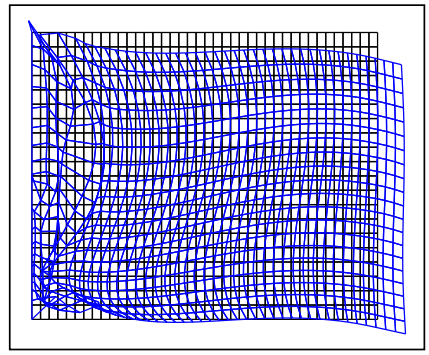


(d) Example 1 - Mode 4

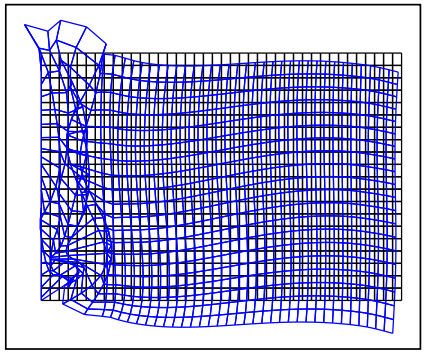
Figure A.1: Example 1 - First eight POD modes



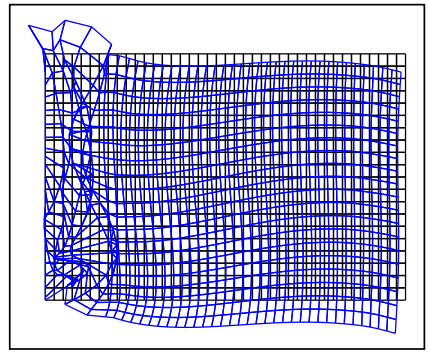
(e) Example 1 - Mode 5



(f) Example 1 - Mode 6

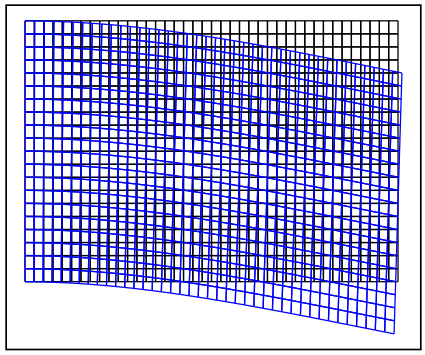


(g) Example 1 - Mode 7

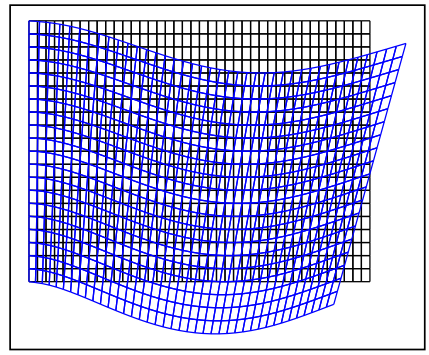


(h) Example 1 - Mode 8

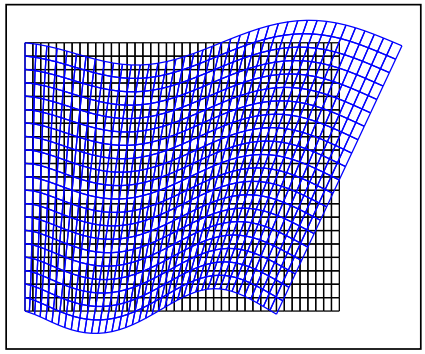
Figure A.1: Example 1 - First eight POD modes



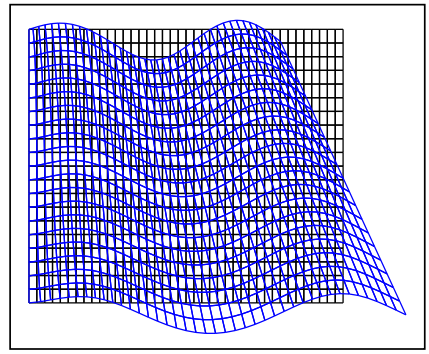
(a) Example 2 - Mode 1



(b) Example 2 - Mode 2

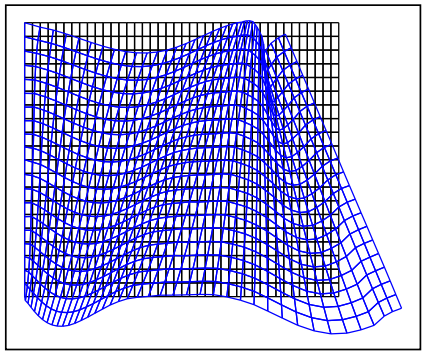


(c) Example 2 - Mode 3

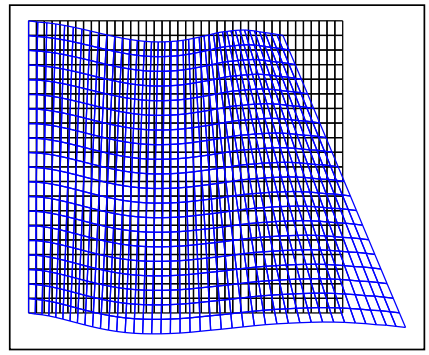


(d) Example 2 - Mode 4

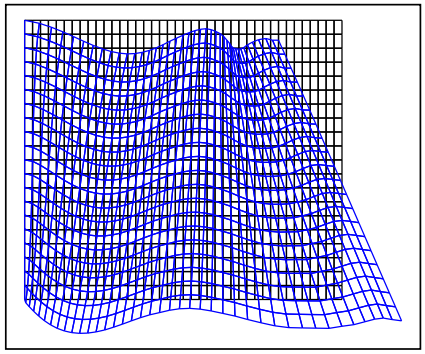
Figure A.2: Example 2 - First eight POD modes



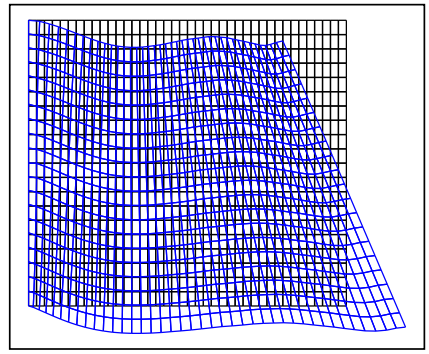
(e) Example 2 - Mode 5



(f) Example 2 - Mode 6



(g) Example 2 - Mode 7



(h) Example 2 - Mode 8

Figure A.2: Example 2 - First eight POD modes

B | Appendix 2

In this chapter the Python code for the numerical studies as presented. The code is organized in the following hierarchy.

- **Analyze** - Scripts running each of the examples
- **Assembly** - Necessary functions for FEM calculation called by the scritpts
- **ROM** - Necessary functions for ROM construction called by the scripts
- **Postprocessing** - Functions for plotting the results of high-fidelity and RB solutions

The code requires the instalation of the Nutils library [25].

List of Python code


```
#Importing libraries
import numpy as np
from matplotlib import pyplot as plt
from matplotlib2tikz import save as tikz_save


#Importing functions
from rectangle_assembly import elastic_assembly_rectangle, Xvec
from elastic_assembly import seminorm
from plot import  dplot, plot_POD_Modes, gplot
from ROM_assembly import generate_RB, retain_RBs, generate_ROM,
    generate_snapshots_rectangle, retain_Modes

#geometry, number of elements must be multiples
nel_length = 40              # number of elements in length dir.
nel_height = 20               # number of elements in height dir.


#number of sampling points in each direction
n = 5

#geometric parameters
l = np.linspace(2000,4000,n)
l0 = 0.5*(l[-1]+l[0])
h = np.linspace(150,250,n)
h0 = 0.5*(h[-1]+h[0])

X0 = np.array([[0,0,l0,l0],[0,h0,0,h0]]) #([x1,x3,x2,x4],[y1,y3,y2,y4])
    mean geometry

# Material parameters
E = np.linspace(10,90,n)*1e3       # Young's modulus [N/mm^2]
```

```
nu = 0.3                                # Poisson's ratio

#Load
L1 = -0.8                                # Normal load top boundary [N/mm^2]

#parameter indeendet assembly
domain, geom, basis, LHS_lam_i, LHS_mu_i, RHS_i, constrain =
    elastic_assembly_rectangle(nel_length,nel_height)
Model =  [LHS_lam_i, LHS_mu_i, RHS_i, constrain]

#H1-seminorn for the high-fidelity system
X = seminorm(basis, geom, domain)

#Genereate snapshots
snapshots, EnergyFEM = generate_snapshots_rectangle(LHS_lam_i,LHS_mu_i,
    RHS_i, constrain, E, l, h, nu, L1)

#Generating the reduced basis wrt. the energy inner product and retaining
    essential RB-modes
w, RB = generate_RB(snapshots, X.core)
io = w.size       #original number of POD modes
w, RB = retain_RBs(w, RB)

#Generating the reduced model by multiplying the highfidelity model with
    the reduced basis
LHS_lam_iR, LHS_mu_iR, RHS_iR, constrainR = generate_ROM(Model, RB)
ROM = [LHS_lam_iR, LHS_mu_iR, RHS_iR, constrainR]

#Calculating snapshots and corresponding energy for the parameter sample
snapshotsRB, EnergyRB = generate_snapshots_rectangle(LHS_lam_iR, LHS_mu_iR
    , RHS_iR, constrainR, E, l, h, nu, L1)

#--------------------post-processing----------------------
bezier = domain.sample('bezier', 2)     #sampling of domain
pts = bezier.eval(geom)                  #reference geometry

#Caclulating mean error in energy-norm for different RBs
error = np.zeros(RB.shape[1])
for i in range (RB.shape[1]):
    LHS_lam_temp, LHS_mu_temp, RHS_temp, constrain_temp = retain_Modes(ROM
    , i+1)
    _, Energy_temp = generate_snapshots_rectangle(LHS_lam_temp,LHS_mu_temp
    , RHS_temp, constrain_temp, E, l, h, nu, L1)
    error[i] = sum((abs(EnergyFEM - Energy_temp)/EnergyFEM)**0.5)/
    EnergyFEM.size

#numer of retained modes
x = np.linspace(1, error.shape[0],num = error.shape[0])

#plotting aggregated relative error for different RBs
```

```
plt.figure()
plt.semilogy(x,error)
#tikz_save("Ex1_agg_error.tex")

#Plotting the spectrum of singular values (sigma) of the retained RB-modes
plt.figure()
plt.semilogy(x,w)
#tikz_save("Ex1_spectrum.tex")

#plotting POD-modes on reference geometry
plot_POD_Modes(RB, basis, bezier,pts)

#Plotting high-fidelity(1)- vs. RB(2)-solution for comparisson for
    parameter sample i
i = 7
a = int(i%n)
b = int((i-a)/n)

li = l[b]
hi = h[a]
Xi = np.array([[0,0,li,li],[0,hi,0,hi]]) #([x1,x4,x2,x3],[y1,y4,y2,y3])
    for samle i
Xi = Xvec(nel_length,nel_height,Xi)
geomi = basis.dot(Xi)

sol1 = snapshots[:,i]
disp1 = basis.dot(sol1)

sol2 = snapshotsRB[:,i]
sol2h = RB@sol2
disp2 = basis.dot(sol2h)

ptsi, dpts1, dpts2 = bezier.eval([geomi, disp1, disp2])
dpts1 += ptsi
dpts2 += ptsi

dplot(bezier, dpts1, dpts2)
#tikz_save("Ex1_hf_vs_rb.tex")

#Plotting instance of physical geometry and reference geometry
gplot(bezier, ptsi)
#tikz_save("Ex1-instance.tex")

gplot(bezier, pts)
#tikz_save("Ex1-reference.tex")



#plotting POD-modes on mean geometry
#X0 = Xvec(nel_length,nel_height,X0)
```

```
#geom0 = basis.dot(X0)
#pts0 = bezier.eval(geom0)
#plot_POD_Modes(RB, basis, bezier, pts0)
```

B.1: Example 1 - script

```
#Importing libraries
import numpy as np
from matplotlib import pyplot as plt
from matplotlib2tikz import save as tikz_save

#importing functions
from elastic_assembly import  seminorm
from trapeze_assembly import elastic_assembly_trapeze
from rectangle_assembly import Xvec
from plot import dplot, plot_POD_Modes, gplot
from ROM_assembly import generate_RB, retain_RBs, generate_ROM,
    genereate_snapshots_trapeze, retain_Modes

#geometry, number of elements must be multiples and nel_height should be a
    even number
nel_length =  40                    # number of elements in length dir.
nel_height = 20                       # number of elements in height dir.

#number of sampling points in each direction
n = 4

#number of terms i geometric series R
k = 20

#geometric parameters
L = 3000    #mm
H0 = 350     #mm
H1 = np.linspace(150,350,n)

H1m = 0.5*(H1[0]+H1[-1])
X0 = np.array([[0,0,L,L],[0,H0,0.5*(H0-H1m),0.5*(H0+H1m)]]) #([x1,x4,x2,x3
    ],[y1,y4,y2,y3]) Mean geomtry

# Material parameters
E = np.linspace(35,65, n)*1e3       #Young's modulus [N/mm^2]
nu = 0.3                            # Poisson's ratio

#Load
L1 = -0.8                            # Normal load top boundary [N/mm^2]

#parameter independet assembly
LHS_lam, coff_lam, LHS_mu, coff_mu, RHS, coff_RHS, constrain, domain, geom
    , basis, mu2 = elastic_assembly_trapeze(nel_length,nel_height, L, H0,
    k)
Model = [LHS_lam, LHS_mu, RHS, constrain]
```

```
41
42 #H1-seminorn for the high-fidelity system
43 X_norm = seminorm(basis, geom, domain)
44
45 #Iterating through the parameter selection creating snapshots and
       corresponding energy
46 snapshots, FEM_Energy = genereate_snapshots_trapeze(LHS_lam, coff_lam,
       LHS_mu, coff_mu, RHS, coff_RHS, constrain, mu2, E, nu, L, H0, H1, L1)
47
48 #Generating the reduced basis wrt. the energy inner product and retaining
       essential RB-modes
49 w, RB = generate_RB(snapshots, X_norm.core)
50 io = w.size                                    #original number of POD
       modes
51 w, RB = retain_RBs(w, RB)
52
53 #Generating the reduced model by multiplying the highfidelity model with
       the reduced basis
54 LHS_lamR, LHS_muR, RHS_R, constrainR = generate_ROM(Model, RB)
55 ROM = [LHS_lamR, LHS_muR, RHS_R, constrainR]
56
57 #Calculating RB snapshots and corresponding energy for the parameter
       sample
58 snapshotsRB, RB_Energy = genereate_snapshots_trapeze(LHS_lamR, coff_lam,
       LHS_muR, coff_mu, RHS_R, coff_RHS, constrainR, mu2, E, nu, L, H0, H1,
       L1)
59
60 #-----------------------post processing-----------------------------
61 bezier = domain.sample('bezier', 2)    #sampling of domain
62 pts = bezier.eval(geom)                 #reference geomtry
63
64
65 #Caclulating mean error in energy-norm for different RBs
66 error = np.zeros(RB.shape[1])
67 for i in range (RB.shape[1]):
68     LHS_lam_temp, LHS_mu_temp, RHS_temp, constrain_temp = retain_Modes(ROM
       , i+1)
69     _, Energy_temp = genereate_snapshots_trapeze(LHS_lam_temp, coff_lam,
       LHS_mu_temp, coff_mu, RHS_temp, coff_RHS, constrain_temp, mu2, E, nu,
       L, H0, H1, L1)
70     error[i] = sum((abs(FEM_Energy - Energy_temp)/FEM_Energy)**0.5)/
       FEM_Energy.size
71
72 #numer of retained modes
73 x = np.linspace(1, error.shape[0],num = error.shape[0])
74
75 #plotting relative error for for different RBs
76 plt.figure()
77 plt.semilogy(x,error)
78 #tikz_save("Ex2_agg_error.tex")
```

```
79
80 #Plotting the spectrum of singular values (sigma) of the retained RB-modes
81 plt.figure()
82 plt.semilogy(x,w)
83 #tikz_save("Ex2_spectrum.tex")
84
85 #plotting POD-modes on reerence geometry
86 plot_POD_Modes(RB, basis, bezier,pts)
87
88 #Plotting high-fidelity(1)- vs. RB(2)-solution for comparisson for
       parameter sample i
89 i = 0
90 a = int(i%n)
91
92 hi = H1[a]
93 Xi = np.array([[0,0,L,L],[0,H0, 0.5*(H0 - hi),0.5*(H0 + hi)]]) #([x1,x3,x2
       ,x4],[y1,y3,y2,y4]) for sample i
94 Xi = Xvec(nel_length,nel_height,Xi)
95 geomi = basis.dot(Xi)
96
97 sol1 = snapshots[:,i]
98 disp1 = basis.dot(sol1)
99
100 sol2 = snapshotsRB[:,i]
101 sol2h = RB@sol2
102 disp2 = basis.dot(sol2h)
103
104 ptsi, dpts1, dpts2 = bezier.eval([geomi, disp1, disp2])
105 dpts1 = dpts1*3              #scaling for visibility
106 dpts2 = dpts2*3              #scaling for visibility
107 dpts1 += ptsi
108 dpts2 += ptsi
109
110 dplot(bezier, dpts1, dpts2)
111 #tikz_save("Ex2_hf_vs_rb.tex")
112
113
114 #Plot instance of physical and reference geomtry
115 gplot(bezier, ptsi)
116 #tikz_save("Ex2-instance.tex")
117
118 gplot(bezier, pts)
119 #tikz_save("Ex2-reference.tex")
120
121 #plotting POD-modes on mean geometry
122 #X0 = Xvec(nel_length,nel_height,X0)
123 #geom0 = basis.dot(X0)
124 #pts0 = bezier.eval(geom0)
125 #plot_POD_Modes(RB, basis, bezier, pts0)
```

B.2: Example 2 - script

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib2tikz import save as tikz_save

#importing functions
from plot import  dplot,plot_POD_Modes, gplot
from elastic_assembly import elastic_assembly, seminorm
from ROM_assembly import generate_RB, generate_ROM, retain_RBs,
    generate_snapshots, retain_Modes

#geometry
nel_length = 40                     # number of elements in length dir.
nel_height = 20                     # number of elements in height dir.

L = 3000
H0 = 350
H1 = 150
Xgeom = np.array([[0,L,0,L],[0,0.5*(H0-H1),H0,0.5*(H0+H1)]]) #([x1,x2,x3,
    x4],[y1,y2,y3,y4])

#Generating the high-fidelity model
domain, geom, basis, lhs_lam, lhs_mu, rhs_b, rhs_t, rhs_r, rhs_l,
    constrain = elastic_assembly(Xgeom ,nel_length,nel_height)
rhs_t = rhs_t
rhs_l = rhs_l
Model =  [lhs_lam, lhs_mu, rhs_b, rhs_t, rhs_r, rhs_l, constrain]

#parameters
E = np.array([1,5,9])*1e4                  # Young's modulus [N/mm^2]
nu = np.array([0.25, 0.335, 0.42])          # Poisson's ratio
l_b = np.array([-0.4,0,0.4])                 #[MPa]
l_r = np.array([-.4,0,.4])                   #[MPa]


#H1-seminorn for the high-fidelity system
X = seminorm(basis, geom, domain)

#Calculating the solutions for all parameter samples, and the
    corresponding strain energy
snapshots, EnergyFEM = generate_snapshots(E, nu, l_b, l_r, lhs_lam, lhs_mu
    , rhs_t, rhs_b, rhs_r, rhs_l, constrain)

#Generating the reduced basis wrt. the energy inner product and retaining
    essential RB-modes
w, RB = generate_RB(snapshots, X.core)
io = w.size       #original number of POD modes
w, RB = retain_RBs(w, RB)

#Generating the reduced model by multiplying the highfidelity model with
    the reduced basis
```

```
lhs_lamR, lhs_muR, rhs_bR, rhs_tR, rhs_rR, rhs_lR, constrainR =
    generate_ROM(Model, RB)
ROM = [lhs_lamR, lhs_muR, rhs_bR, rhs_tR, rhs_rR, rhs_lR, constrainR]


snapshotsRB, EnergyRB = generate_snapshots(E, nu, l_b, l_r, lhs_lamR,
    lhs_muR, rhs_tR, rhs_bR, rhs_rR, rhs_lR, constrainR)
#-------------------Post-processing----------------------------------
bezier = domain.sample('bezier', 2)
pts = bezier.eval(geom)

#Testing accuracy mean accuracy in energy-norm for retaining POD-modes
error = np.zeros(RB.shape[1])
for i in range (RB.shape[1]):
    lhs_lam_temp, lhs_mu_temp, rhs_b_temp, rhs_t_temp, rhs_r_temp,
    rhs_l_temp, constrain_temp = retain_Modes(ROM,i+1)
    _, Energy_temp = generate_snapshots(E, nu, l_b, l_r, lhs_lam_temp,
    lhs_mu_temp, rhs_t_temp, rhs_b_temp, rhs_r_temp, rhs_l_temp,
    constrain_temp)
    error[i] =sum( (abs(EnergyFEM - Energy_temp)/EnergyFEM)**0.5)/
    EnergyFEM.size

x = np.linspace(1, error.shape[0],num = error.shape[0])

plt.figure()
plt.semilogy(x,error)
#tikz_save("avg_error.tex")

#Plotting the spectrum of singular values (sigma) of the retained RB-modes
plt.figure()
plt.semilogy(x,w)
#tikz_save("spectrum.tex")

#plotting POD-modes
plot_POD_Modes(RB, basis, bezier,pts)

#Plotting high-fidelity(1)- vs. RB(2)-solution for comparisson for
    parameter sample i
i = 2
sol1 = snapshots[:,i]
disp1 = basis.dot(sol1)

sol2 = snapshotsRB[:,i]
sol2h = RB@sol2
disp2 = basis.dot(sol2h)


pts, dpts1, dpts2 = bezier.eval([geom, disp1, disp2])
dpts1 += pts
dpts2 += pts
```

```
87
88 dplot(bezier, dpts1, dpts2)
89
90 #Plot geometry
91 gplot(bezier, pts)
```

B.3: Test script

```
1  #Importing libraries
2  import numpy as np
3  import sympy as sp
4  from matplotlib import pyplot as plt
5  from matplotlib2tikz import save as tikz_save
6
7  #Geometry
8  xi = 1
9  H0 = 450
10 mu = sp.symbols("mu")
11 theta = mu-H0
12
13 #Height vector
14 m = 5
15 H1 = np.linspace(150,350, m)
16
17 labels = []
18 for k in range(H1.size):
19     labels.append(str(H1[k]))
20
21 #max number of terms
22 n = 70
23
24 x= np.linspace(1,n,n)
25
26 #reference R
27 Rs_ref = 1/(1+theta/H0)
28
29 R = -np.ones([n])*(theta*xi/H0)
30
31 for i in range(n):
32     R[i] = R[i]**i
33
34 error = np.zeros([n,m])
35
36 for i in range(n):
37     for j in range(m):
38         temp = sum(R[:i+1])
39         temp = temp.subs(mu, H1[j])
40         Rs_ref_temp = Rs_ref.subs(mu, H1[j])
41         error[i,j] = abs((Rs_ref_temp-temp)/Rs_ref_temp)
42
43 #Plotting results
```

```
plt.figure()
plt.semilogy(x,error)
plt.gca().legend(labels)
plt.ylim(10**(-15),1)

#tikz_save('error_R.tex')
```

B.4: Evaluation of geometric series representation of R

```
from nutils import mesh, matrix, function as fn
import numpy as np

def elastic_assembly_rectangle(nel_length,nel_height):
    # Create domain and reference geometry
    xpts = np.linspace(0, 1, nel_length + 1)
    ypts = np.linspace(0, 1, nel_height + 1)
    domain, geom = mesh.rectilinear([xpts, ypts])
    zeta, eta = geom

    # Create a basis
    basis = domain.basis('spline', degree=1)
    basis2 = fn.vectorize([basis,basis])
    n = basis.length

    #calculating submatrices
    grad = basis.grad(geom)
    m1 = domain.integrate(fn.outer(grad[:,0],grad[:,0]) * fn.J(geom),
    ischeme='gauss5').core
    m2 = domain.integrate(fn.outer(grad[:,0],grad[:,1]) * fn.J(geom),
    ischeme='gauss5').core
    m3 = domain.integrate(fn.outer(grad[:,1],grad[:,0]) * fn.J(geom),
    ischeme='gauss5').core
    m4 = domain.integrate(fn.outer(grad[:,1],grad[:,1]) * fn.J(geom),
    ischeme='gauss5').core

    #Lambda dependent assembly
    K_lam_1 = matrix.NumpyMatrix(multiply_I_mat(n,1,0,0, 0,m1))
    K_lam_2 = matrix.NumpyMatrix( multiply_I_mat(n,0,1,0, 0, m2) +
    multiply_I_mat(n,0,0,1, 0,m3))
    K_lam_3 = matrix.NumpyMatrix(multiply_I_mat(n,0,0,0,1,m4))

    #mu dependent assembly
    K_mu_1 = matrix.NumpyMatrix(multiply_I_mat(n,2,0,0,1,m1))
    K_mu_2 = matrix.NumpyMatrix( multiply_I_mat(n,0,0,1, 0, m2) +
    multiply_I_mat(n,0,1,0, 0,m3))
    K_mu_3 = matrix.NumpyMatrix(multiply_I_mat(n,1,0,0,2,m4))

    LHS_lam_i = [K_lam_1, K_lam_2, K_lam_3]

    LHS_mu_i = [K_mu_1, K_mu_2, K_mu_3]

```

```
    # RHS vector
    # Construct a vector-valued function and integrate it for all boundaries
    boundary_b = domain.boundary['bottom']
    boundary_t = domain.boundary['top']
    boundary_r = domain.boundary['right']
    boundary_l = domain.boundary['left']
    rhs_n = fn.matmat(basis2, geom.normal())
    rhs_x = fn.matmat(basis2,[1,0])
    rhs_y = fn.matmat(basis2,[0,1])
    y_par = 6*eta*(1-eta)

    rhs_b = boundary_b.integrate( rhs_y * fn.J(geom), ischeme='gauss5')
    rhs_t = boundary_t.integrate( rhs_y * fn.J(geom), ischeme='gauss5')
    rhs_r = boundary_r.integrate( rhs_x * fn.J(geom), ischeme='gauss5')
    rhs_l = -2*boundary_l.integrate( rhs_y * y_par * fn.J(geom), ischeme='gauss5')

    RHS_i= [rhs_t, rhs_b, rhs_r, rhs_l]

    # Dirichlet boundary conditions
    # Fix the displacement on the left side
    zero = fn.zeros((2,))
    boundary = domain.boundary['left']
    constrain = boundary.project(zero, onto=basis2, geometry=geom, ischeme='gauss5')
    nan = constrain[-1]
    constrain[n+1:n+int(nel_height/2)] = nan
    constrain[n+int(nel_height/2)+1:n+nel_height] = nan

    #return model, lhs, rhs and constraints
    return domain, geom, basis2, LHS_lam_i, LHS_mu_i, RHS_i, constrain

def theta_calc(l,h):
    theta_1 = h/l
    theta_2 = 1
    theta_3 = l/h
    theta = [theta_1,theta_2,theta_3]
    J = l*h
    return theta, J

def multiply_I_mat(n,a1,a2,a3, a4, m):
    I_0 = np.eye(n)
    I = np.zeros([2*n,2*n])
    I[:n,:n] = a1*I_0@m
    I[:n,n:] = a2*I_0@m
    I[n:,:n] = a3*I_0@m
    I[n:,n:] = a4*I_0@m
    return I

```

```
84 def Xvec(nel_length,nel_height,X):#returning a vector of geometrical nodal
       values for the given grid
85     #reference element
86     x0 = np.linspace(0,1,2)
87     y0 = np.linspace(0,1,2)
88     domain, geom = mesh.rectilinear([x0, y0])
89 
90     #basis function over reference domain
91     basis = domain.basis('spline', degree=1)
92     basis = fn.vectorize([basis,basis])
93 
94     #sampling using a bezier rule corresponding to the mesh of the global
       domain
95     bezier = domain.sample('bezier', (nel_length+1,nel_height+1))
96 
97     #creating the field(n,g) from the corner nodes
98     X = basis.dot(X.flatten())
99 
100    #evaluting with respect to the bezier sample
101    x = bezier.eval(X)
102    Xvec = x.T.flatten()
103    return Xvec
```

B.5: Example 1 - FEM functions

```
1  from nutils import mesh, function as fn
2  import numpy as np
3  import sympy as sp
4  
5  
6  def elastic_assembly_trapeze(nel_length,nel_height, L, H0, k ):
7      # Create domain and reference geometry
8      xpts = np.linspace(0, 1, nel_length + 1)
9      ypts = np.linspace(0, 1, nel_height + 1)
10     domain, geom = mesh.rectilinear([xpts, ypts])
11     zeta, eta = geom
12     mu2 = sp.Symbol('mu2')
13     theta = mu2-H0
14     r0 = theta/H0
15 
16     # Create a basis
17     basis = domain.basis('spline', degree=1)
18     basis2 = fn.vectorize([basis,basis])
19     n = basis.length
20 
21     #derivatives of basis
22     grad = basis.grad(geom)
23 
24     B1 = grad[:,0]/L
25     B2 = grad[:,1]*(0.5-eta)/(L*H0)
26     B3 = grad[:,1]/H0
```

```
27
28      #submatrices and their corresponding coeficcients
29      N0_xx = integrate_matrix( [fn.outer(B1,B1), (zeta/H0)*fn.outer(B1,B1)
        + fn.outer(B2,B1) + fn.outer(B1,B2)],L,H0, geom, domain)
30      coffxx = [1, theta]
31      N0_xxR = [fn.outer(B2,B2)]
32      coffxxR = [ theta**2]
33      N0_xxR, coffxxR = assemle_integrate_R_terms(N0_xxR, coffxxR, geom,
        domain, L, H0,  zeta, r0, k)
34      N0_xx.extend(N0_xxR)
35      coffxx.extend(coffxxR)
36      #coffxx = [1, theta*R, theta^2*R]
37
38
39      N0_xy = integrate_matrix( [fn.outer(B1,B3)],L,H0, geom, domain)
40      coffxy = [1]
41      N0_xyR = [fn.outer(B2,B3)]
42      coffxyR = [theta]
43      N0_xyR, coffxyR = assemle_integrate_R_terms(N0_xyR, coffxyR, geom,
        domain, L, H0,  zeta, r0, k)
44      N0_xy.extend(N0_xyR)
45      coffxy.extend(coffxyR)
46      #coffxy = [1, theta*R]
47
48      N0_yx = [N0_xy[0].T]
49      coffyx = [1]
50      N0_yxR = [fn.outer(B3,B2)]
51      coffyxR = [theta]
52      N0_yxR, coffyxR = assemle_integrate_R_terms(N0_yxR, coffyxR, geom,
        domain, L, H0,  zeta, r0, k)
53      N0_yx.extend(N0_yxR)
54      coffyx.extend(coffyxR)
55      #coffyx = [1, theta*R]
56
57      N0_yy = [fn.outer(B3,B3)]
58      coffyy = [1]
59      N0_yy, coffyy = assemle_integrate_R_terms(N0_yy, coffyy, geom, domain,
         L, H0,  zeta, r0, k)
60      #coffyy = [R]
61
62      #assembly of submatrices construction of the stiffness matrices
63      #terms deppendent on lambda
64      axx = [1,0,0,0]
65      axy = [0,1,0,0]
66      ayx = [0,0,1,0]
67      ayy = [0,0,0,1]
68
69      LHS_lam, coff_lam = assemble_terms(N0_xx, coffxx, axx, N0_xy, coffxy,
        axy, N0_yx, coffyx, ayx, N0_yy, coffyy, ayy, n)
70      LHS_lam, coff_lam = organize(LHS_lam, coff_lam)
```

```
71
72
73      #terms deppendent on mu
74      axx = [2,0,0,1]
75      axy = [0,0,1,0]
76      ayx = [0,1,0,0]
77      ayy = [1,0,0,2]
78      LHS_mu, coff_mu = assemble_terms(N0_xx, coffxx, axx, N0_xy, coffxy,
        axy, N0_yx, coffyx, ayx, N0_yy, coffyy, ayy, n)
79      LHS_mu, coff_mu = organize(LHS_mu, coff_mu)
80
81      # RHS vector
82      # Construct a vector-valued function and integrate it for all
        boundaries
83      boundary_b = domain.boundary['bottom']
84      boundary_t = domain.boundary['top']
85      boundary_r = domain.boundary['right']
86      boundary_l = domain.boundary['left']
87      rhsn = fn.matmat(basis2, geom.normal())
88      rhsy = fn.matmat(basis2,[0,1])
89      rhsx = fn.matmat(basis2,[1,0])
90      y_par = 6*eta*(1-eta)
91
92      rhs_b = L*boundary_b.integrate( rhsy * fn.J(geom), ischeme='gauss5')
93      rhs_t = L*boundary_t.integrate( rhsy * fn.J(geom), ischeme='gauss5')
94      rhs_r = boundary_r.integrate( rhsx * fn.J(geom), ischeme='gauss5')
95      rhs_l = -boundary_l.integrate( rhsy * y_par  *fn.J(geom), ischeme='
        gauss5')
96
97      RHS= [ rhs_b, rhs_t, rhs_r, rhs_l ]
98      coff_RHS = [(L**2 + (theta**2)/4)**0.5/L ,(L**2 + (theta**2)/4)**0.5/L
        *0,mu2*0,L]
99
100     # Dirichlet boundary conditions
101     # Fix the displacement on the left side
102     zero = fn.zeros((2,))
103     boundary = domain.boundary['left']
104     constrain = boundary.project(zero, onto=basis2, geometry=geom, ischeme
        ='gauss5')
105     nan = constrain[-1]
106     constrain[n+1:n+int(nel_height/2)] = nan
107     constrain[n+int(nel_height/2)+1:n+nel_height] = nan
108
109
110     return LHS_lam, coff_lam, LHS_mu, coff_mu, RHS, coff_RHS, constrain,
        domain, geom, basis2, mu2
111
112
113 def integrate_matrix(M,L,H, geom, domain):
114     MI = []
```

```
115     for element in M:
116         MI.append(L*H*domain.integrate(element*fn.J(geom),ischeme = '
        gauss10'))
117     return MI
118
119
120 def dot_Imat(n,a, m):
121     I_0 = np.eye(n)
122     I = np.zeros([2*n,2*n])
123     I[:n,:n] = a[0]*I_0@m.core
124     I[:n,n:] = a[1]*I_0@m.core
125     I[n:,:n] = a[2]*I_0@m.core
126     I[n:,n:] = a[3]*I_0@m.core
127     return I
128
129 def assemble_terms(N0_xx, coffxx, axx, N0_xy, coffxy, axy, N0_yx, coffyx,
        ayx, N0_yy, coffyy, ayy, n):
130     LHS = []
131     coff = []
132     for i in range(len(N0_xx)):
133         LHS.append(dot_Imat(n,axx,N0_xx[i]))
134         coff.append(coffxx[i])
135     for j in range (len(N0_xy)):
136         LHS.append(dot_Imat(n,axy,N0_xy[j]) + dot_Imat(n,ayx,N0_yx[j]))
137         coff.append(coffxy[j])
138     for k in range(len(N0_yy)):
139         LHS.append(dot_Imat(n,ayy,N0_yy[k]))
140         coff.append(coffyy[k])
141
142     return LHS, coff
143
144 def assemle_integrate_R_terms(N0_R, coffR, geom, domain, L, H0,  zeta, r0,
        k):
145     M = []
146     coff = []
147     for j in range(len(N0_R)):
148         temp = N0_R[j]
149         coff_temp = coffR[j]
150         for i in range(k):
151             M.append( temp*((-zeta)**i))
152             coff.append(coff_temp*(r0**i))
153     MI = integrate_matrix(M,L,H0, geom, domain)
154     return MI, coff
155
156 #Equal coeficcients are detected and the corresponding matrices is added
        together
157 def organize(M, coff):
158     Mo = []
159     n = len(coff)
160     c = coff
```

```
    i = 0
    while i< n:
        index = [j for j, e in enumerate(c) if e == c[i]]
        Mo.append(M[index[0]])
        del index[0]
        index.reverse()
        if len(index)>0:
            for k in index:
                del c[k]
                Mo[i] += M[k]
                del M[k]
            n = len(c)
        i +=1
    return Mo, coff


def evaluate_list(ls, var, value):
    Ls = []
    for element in ls:
        if type(element) != int and type(element) != float :
            Ls.append(element.subs(var, value))
        else:
            Ls.append(element)
    Ls = np.asarray(Ls,dtype = np.float)
    return Ls

def multiply_coff(coff, M):
    K = np.zeros(M[0].shape)
    for i in range(len(coff)):
        K = K+  M[i]*coff[i]
    return K

def geometry(zeta, eta, Xgeom):
    N0Q4 = [zeta*(1-eta), (1-zeta)*eta, zeta*eta]
    NQ4 = fn.vectorize([N0Q4,N0Q4])
    geom = NQ4.dot(Xgeom)
    return geom
```

B.6: Example 2 - FEM functions

```
from nutils import mesh, function as fn
import numpy as np

#returns LHS matrices which can be scaled by lame-coeficients, and
    loadvectors for normal loads on all three surfaces
def elastic_assembly(X, nel_length, nel_height):
    # Create a domain and a geometry
    xpts = np.linspace(0, 1, nel_length + 1)
    ypts = np.linspace(0, 1, nel_height + 1)
    domain, geom0 = mesh.rectilinear([xpts, ypts])
    zeta, eta = geom0
```

```
11
12      # Create a basis
13      # Like the geometry, a basis is also a function
14      basis = domain.basis('spline', degree=1)
15      n = basis.shape[0]
16
17      # Vectorize the basis (we need two components)
18      basis = fn.vectorize([basis, basis])
19
20      #Creating nodal values of the geometry (x,y)
21      Xgeom = X[:,1:].flatten()
22      geom = geometry(zeta, eta, Xgeom)
23
24      # LHS matrix
25      # Construct a matrix-valued function and integrate it
26      #seperated in two matrices dependent on each lamè parameter
27      lhs_lam = fn.outer(basis.div(geom))
28      lhs_lam = domain.integrate(lhs_lam * fn.J(geom), ischeme='gauss5')
29
30      lhs_mu = fn.outer(basis.symgrad(geom)).sum([-1, -2])
31      lhs_mu = domain.integrate(2*lhs_mu * fn.J(geom), ischeme='gauss5')
32
33      # RHS vector
34      # Construct a vector-valued function and integrate it for all
        boundaries
35      boundary_b = domain.boundary['bottom']
36      boundary_t = domain.boundary['top']
37      boundary_r = domain.boundary['right']
38      boundary_l = domain.boundary['left']
39      rhs_n = fn.matmat(basis, geom.normal())
40      rhs_x = fn.matmat(basis,[1,0])
41      rhs_y =  fn.matmat(basis,[0,1])
42      y_par = 6*eta*(1-eta)
43      rhs_b = boundary_b.integrate( rhs_y * fn.J(geom), ischeme='gauss5')
44      rhs_t = boundary_t.integrate( rhs_y * fn.J(geom), ischeme='gauss5')
45      rhs_r = boundary_r.integrate( rhs_x * fn.J(geom), ischeme='gauss5')
46      rhs_l = -2*boundary_l.integrate( rhs_y * y_par *fn.J(geom), ischeme='
        gauss5')*Xgeom[0]/Xgeom[-2]
47
48      # Dirichlet boundary conditions
49      # Fix the displacement on the left side
50      zero = fn.zeros((2,))
51      boundary = domain.boundary['left']
52      constrain = boundary.project(zero, onto=basis, geometry=geom, ischeme=
        'gauss5')
53      nan = constrain[-1]
54      constrain[n+1:n+int(nel_height/2)] = nan
55      constrain[n+int(nel_height/2)+1:n+nel_height] = nan
56
57      #return model, lhs, rhs and constraints
```

```
58      return domain, geom, basis, lhs_lam, lhs_mu, rhs_b, rhs_t, rhs_r,
        rhs_l, constrain
59
60  #returns lames first parameter
61  def lame_lambd(E,nu):
62      return (E*nu)/((1 + nu)*(1 - 2*nu))
63
64  #returns lames second parameter
65  def lame_mu(E,nu):
66      return E / (2*(1 + nu))
67
68  def geometry(zeta, eta, Xgeom):
69      N0Q4 = [zeta*(1-eta), (1-zeta)*eta, zeta*eta]
70      NQ4 = fn.vectorize([N0Q4,N0Q4])
71      geom = NQ4.dot(Xgeom)
72      return geom
73
74  #Calculates H1-seminorm/energynorm
75  def seminorm(basis, geom, domain):
76      X = fn.outer(basis.grad(geom)).sum([-1, -2])
77      X = domain.integrate(X * fn.J(geom), ischeme='gauss5')
78      return X
79
80  def energynorm(basis,geom,domain,my,lam):
81      X = lam*fn.outer(basis.div(geom)) + 2*my*fn.outer(basis.symgrad(geom))
        .sum([-1, -2])
82      #X = fn.outer(basis.div(geom)) + 2*fn.outer(basis.symgrad(geom)).sum
        ([-1, -2])
83      X = domain.integrate(X * fn.J(geom), ischeme='gauss5')
84      return X
85
86  def energynorm_strain(basis,geom,domain):
87      X = fn.outer(basis.symgrad(geom)).sum([-1, -2])
88      X = domain.integrate(X * fn.J(geom), ischeme='gauss5')
89      return X
90
91  def Xvec(nel_length,nel_height,X):#returning a vector of geometrical nodal
        values for the given grid
92      #reference element
93      x0 = np.linspace(0,1,2)
94      y0 = np.linspace(0,1,2)
95      domain, geom = mesh.rectilinear([x0, y0])
96
97      #basis function over reference domain
98      basis = domain.basis('spline', degree=1)
99      basis = fn.vectorize([basis,basis])
100
101     #sampling using a bezier rule corresponding to the mesh of the global
        domain
102     bezier = domain.sample('bezier', (nel_length+1,nel_height+1))
```

```
103
104     #creating the field(n,g) from the corner nodes
105     X = basis.dot(X.flatten())
106
107     #evaluting with respect to the bezier sample
108     x = bezier.eval(X)
109     Xvec = x.T.flatten()
110     return Xvec
```

B.7: Genereal FEM functions

```
1  from scipy.linalg import fractional_matrix_power as frac_power
2  import numpy as np
3  from nutils import matrix
4
5  from elastic_assembly import lame_lambd, lame_mu
6  from rectangle_assembly import theta_calc
7  from trapeze_assembly import evaluate_list, multiply_coff
8
9  #Solving the eigenvalue problem and returning the reduced basis on
       descending order
10 def generate_RB(snapshots, X):
11     ns = snapshots.shape[1]   #number of samples
12     Nh = snapshots.shape[0]   #number of dofs
13
14     if ns<=Nh:
15         A = snapshots.T@X@snapshots  #Matrix wich we are solving the
       eigenvalue problem for
16         w,v = np.linalg.eigh(A)
17         w,v = ordered(w,v)
18         w = w**0.5                   #singular values for the RB
19         RB = (snapshots@v)*(1/w)     #multiplying to construct the reduced
       basis
20     else:
21         x5 = frac_power(X,0.5).real
22         x_5 = np.linalg.inv(x5)
23         A = x5@snapshots@(snapshots.T)@x5  #Matrix wich we are solving the
        eigenvalue problem for
24         w,v = np.linalg.eigh(A)
25         w,v = ordered(w,v)
26         w = w**0.5                   #singular values for the RB
27         RB = x_5@v                   #multiplying to construct the reduced
       basis
28     return w, RB
29
30 #ordering the RB and the eigenvalues in decreasing order, neglecting all
       negative eigenvalues
31 def ordered(w,v):
32     index = np.where(w>0)[0]     #removing negative eigenvalues
33     w = w[index]
34     v = v[:,index]
```

```
    y = np.zeros(v.shape)        #ordered RB's
    x = -np.sort(-w).T           #ordered eigenvalues
    index = (-w).argsort()       #sorting indexes for RB's
    y[:,index]  = v[:,:]
    return x, y

#retaining RB modes
def retain_RBs(w, RB):
    n =8 #number of retained modes, limit at will for simplicity
    RB = RB[:,:n]
    w = w[:n]
    RB = normalize(RB)
    return w, RB

def normalize(RB):
    lengths = 1/sum(RB)
    RB = RB*lengths
    return RB

#generating the ROM by multiplying high-fidelity model with reduced basis V
def generate_ROM(List, RB):
    Rlist = []
    for element in List :
        if type(element) == matrix.NumpyMatrix:
            Rlist.append(matrix.NumpyMatrix(RB.T@element.core@RB))
        elif type(element) == list:
            list_temp = []
            for i in np.arange(0,len(element)):
                if type(element[i]) == matrix.NumpyMatrix:
                    list_temp.append(matrix.NumpyMatrix(RB.T@element[i].core@RB))
                elif type(element[i]) == np.ndarray:
                    if element[i].shape[0] == element[i].size:
                        list_temp.append(RB.T@element[i])
                    else:
                        list_temp.append(RB.T@element[i]@RB)
                else:
                    list_temp.append(matrix.NumpyMatrix(RB.T@element[i]))
            Rlist.append(list_temp)
        else:
            Rlist.append((RB.T@element))
    return Rlist

#generating snapshots for the parameter selection, ass well as the energy for each sample
def generate_snapshots(E, nu, l_b, l_r, lhs_lam, lhs_mu, rhs_t, rhs_b, rhs_r, rhs_l, constrain):
   snapshots = np.zeros((rhs_b.size, E.size * nu.size * l_b.size * l_r.size - l_b.size*l_r.size))
```

```
    Energy = np.zeros(snapshots.shape[1])
    i = 0
    for E_temp in E:
        for nu_temp in nu:
            lam = lame_lambd(E_temp,nu_temp)
            mu = lame_mu(E_temp,nu_temp)
            LHS = lam*lhs_lam + mu*lhs_mu
            for l_b_temp in l_b:
                for l_r_temp in l_r:
                    if l_b_temp == 0 and l_r_temp == 0:
                        i = i
                    else:
                        RHS = l_b_temp*(rhs_b + rhs_t + rhs_l) + l_r_temp*
     rhs_r
                        sol_temp = LHS.solve(RHS, constrain=constrain)
                        snapshots[:,i] = sol_temp
                        Energy[i] = 0.5*sol_temp.T@LHS.core@sol_temp
                        i +=1
    return snapshots, Energy

#generate snapshots and energy for the rectangle case
def generate_snapshots_rectangle(LHS_lam_i,LHS_mu_i, RHS, constrain, E, l,
      h, nu, L1):
    snapshots = np.zeros((RHS[0].size, E.size*l.size*h.size ))
    Energy = np.zeros(snapshots.shape[1])
    i = 0
    for E_temp in E:
        lam = lame_lambd(E_temp,nu)
        mu = lame_mu(E_temp,nu)
        for l_temp in l:
            RHS_temp = (RHS[0] + RHS[1] + RHS[-1])*l_temp*L1
            for h_temp in h:
                theta, J = theta_calc(l_temp,h_temp)
                LHS = mu*(theta[0]*LHS_mu_i[0] + theta[1]*LHS_mu_i[1] +
     theta[2]*LHS_mu_i[2]) + lam*(theta[0]*LHS_lam_i[0] + theta[1]*
     LHS_lam_i[1] + theta[2]*LHS_lam_i[2])
                sol_temp = LHS.solve(RHS_temp, constrain=constrain)
                snapshots[:,i] = sol_temp
                Energy[i] = 0.5*sol_temp.T@LHS.core@sol_temp
                i +=1
    return snapshots, Energy

#generate snashots and eneergy for the trapezoid case
def genereate_snapshots_trapeze(LHS_lam, coff_lam, LHS_mu, coff_mu, RHS_i,
      coff_RHS, constrain, mu2, E, nu, L, H0, H1, L1):
    snapshots = np.zeros((RHS_i[0].size, E.size*H1.size ))
    Energy = np.zeros(snapshots.shape[1])
    i = 0
    for E_temp in E:
        lam = lame_lambd(E_temp,nu)
```

```
        mu = lame_mu(E_temp,nu)
        for H_temp in H1:
            #calcutating coefficients
            coff_lam1 = evaluate_list(coff_lam, mu2, H_temp)
            coff_mu1 = evaluate_list(coff_mu, mu2, H_temp)
            coff_RHS1 = evaluate_list(coff_RHS, mu2, H_temp)

            #multiplying coefficients with corresponding matrices
            RHS_temp = multiply_coff(coff_RHS1, RHS_i)*L1
            lhs_lam1 = multiply_coff(coff_lam1,LHS_lam)
            lhs_mu1 = multiply_coff(coff_mu1,LHS_mu)
            LHS = lhs_lam1*lam + lhs_mu1*mu
            LHS = matrix.NumpyMatrix(LHS)

            sol_temp = LHS.solve(RHS_temp, constrain=constrain)
            snapshots[:,i] = sol_temp
            Energy[i] = 0.5*sol_temp.T@LHS.core@sol_temp
            i +=1
    return snapshots, Energy


def retain_Modes(ROM, i):
    ROMi = []
    for element in ROM:
        if type(element) == matrix.NumpyMatrix:
            ROMi.append(matrix.NumpyMatrix(element.core[:i, :i]))

        elif type(element) == list:
            list_temp = []
            for j in np.arange(0,len(element)):
                if type(element[j]) == matrix.NumpyMatrix:
                    list_temp.append(matrix.NumpyMatrix(element[j].core[:i
    , :i]))
                elif type(element[j]) == np.ndarray:
                    if element[j].shape[0] == element[j].size:
                        list_temp.append(element[j][:i])
                    else:
                        list_temp.append(element[j][:i,:i])
                else:
                    list_temp.append(matrix.NumpyMatrix(element[j][:i]))
            ROMi.append(list_temp)
        else:
            ROMi.append(element[:i])
    return ROMi
```

B.8: General ROM functions

```
from matplotlib import pyplot as plt, collections
from matplotlib2tikz import save as tikz_save

def plot(bezier, pts, func, filename):
```

```
    # Plot the function
    plt.tripcolor(pts[:,0], pts[:,1], bezier.tri, func, shading='gouraud',
     rasterized=True)

    # Add element lines
    plt.gca().add_collection(collections.LineCollection(pts[bezier.hull],
    linewidth=0.1, color='black'))

    plt.colorbar(orientation='horizontal')
    plt.gca().set_aspect('equal')
    plt.gca().autoscale(enable=True, axis='both', tight=True)
    plt.axis('off')
    plt.savefig(filename, dpi=200, bbox_inches='tight')
    plt.clf()


def fplot(bezier, pts, func):#color plot of function over geometry
    # Plot the function
    plt.tripcolor(pts[:,0], pts[:,1], bezier.tri, func, shading='gouraud',
     rasterized=True)

    # Add element lines
    plt.gca().add_collection(collections.LineCollection(pts[bezier.hull],
    linewidth=0.1, color='black'))

    plt.colorbar(orientation='horizontal')
    plt.gca().set_aspect('equal')
    plt.gca().autoscale(enable=True, axis='both', tight=True)
    plt.axis('off')


def dplot(bezier, pts,dpts):#plots two geometries over eachother, for
    example deformed over original geometry
    plt.figure()
    plt.gca().set_aspect('equal')
    triang = bezier.tri
    plt.triplot(pts[:,0], pts[:,1],triang, 'ko-', lw=0,markersize=0)
    plt.triplot(dpts[:,0], dpts[:,1],triang, 'bo-', lw=0, markersize = 0)
    plt.gca().add_collection(collections.LineCollection(pts[bezier.hull],
    linewidth=0.5, color='black'))
    plt.gca().add_collection(collections.LineCollection(dpts[bezier.hull],
     linewidth=0.5, color='blue'))

def gplot(bezier, pts):#plots a geometry
    plt.figure()
    plt.gca().set_aspect('equal')
    triang = bezier.tri
    plt.triplot(pts[:,0], pts[:,1],triang, 'ko-', lw=0,markersize=0)
    plt.gca().add_collection(collections.LineCollection(pts[bezier.hull],
    linewidth=0.5, color='black'))
```

```
47
48
49
50 def plot_POD_Modes(RB, basis, bezier,pts): #plots each POD mode in the
       reduced basis over a geometry
51     Ex = 'Ex2_POD_mode'
52     tex = '.tex'
53     for i in range(RB.shape[1]):
54         u = RB[:,i]
55         disp = basis.dot(u)
56         dpts = bezier.eval(disp)
57         c = abs(pts).max()/abs(dpts).max()*0.2
58         dpts = -dpts*c
59         dpts += pts
60         dplot(bezier,pts,dpts)
61         string = Ex + str(i+1) + tex
62         #tikz_save(string)
```

B.9: Postprocessing

Bibliography

[1] Michael E. Plesha Robert D. Cook David S. Malkus and Robert J. Witt. *Concepts and applications of the finite element analysis*. 2001.

[2] David Brock. *Understanding Moore's Law: Four Decades of Innovation*. 2006.

[3] Tor Eivind Palm. *Reduced order modelling, with a 2D beam example*. Unpublished semester project. 2018.

[4] A. Patera. "Finite Element Methods for Elliptic Problems". Lecture notes. 2003.

[5] Zhilin Li, Zhonghua Qiao, and Tao Tang. *Numerical Solution of Differential Equations: Introduction to Finite Difference and Finite Element Methods*. Cambridge University Press, 2017.

[6] Kolbein Bell. *An engineering approach to Finite Element Analysis of linear structural mechanics problems*. 2013.

[7] Thomas J. R Hughes. *The Finite Element Method : Linear Static and Dynamic Finite Element Analysis*. Newburyport, 2012.

[8] Anders Logg and Kent-Andre Mardal. "*Lectures on the Finite Element Method*". Lecture notes.

[9] Francesco Züger. *Solution of Non-Homogenous Dirichlet Problems with FEM*. `https://www.math.uzh.ch/li/index.php?file&key1=25297`. 2013.

[10] M. G. Larson and F. Bengzon. *The Finite Element Method: Theory, Implementation and Applications*. 2013.

[11] A. Quarteroni. *Reduced Basis Methods for Partial Differential Equations*. 2016.

[12] Godfrey Harold Hardy. *A Course of Pure Mathematics*. 2012.

[13] M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions with formulas, graphs and mathematical tables*. 2002.

[14] Eun Jung Yoo. "Parametric Model Order reduction for Structural Analysis and Control". `https://mediatum.ub.tum.de/doc/997277/997277.pdf`. PhD thesis. TUM, 2010.

[15] G. Weickum and Advisor Kurte Maute. *Design Optimization Using Reduced Order Models*. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.560.5608&rank=1`.

[16] J. Speet. *Parametric reduced order modeling for structural models by manifold interpolation techniques*. `https://repository.tudelft.nl/islandora/object/uuid%3Ad1b77685-5b95-4991-bd41-7ce4cdd01605`. 2017.

[17] Matthias Herz Dirk Hartmann and Utz Wever. “Model Order Reduction, a key technology for Digital Twins”. In: *IEEE* (2017). https://www.researchgate.net/publication/324470068_Model_Order_Reduction_a_Key_Technology_for_Digital_Twins.

[18] André I. Khuri. “Response Surface Methodology”. In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1229–1231. URL: https://doi.org/10.1007/978-3-642-04898-2_492.

[19] Benjamin Stamm Jan S. Hesthaven Gianluigi Rozza. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. 2016.

[20] M. A. Bazaz, Mashuq-un-Nabi, and S. Janardhanan. “A review of parametric model order reduction techniques”. In: *2012 IEEE International Conference on Signal Processing, Computing and Control*. Mar. 2012, pp. 1–6.

[21] John Hocroft Avrim Blum and Ravindran Kannan. *Foundations of Data Science*. unpublished. 2018.

[22] “An introduction to the proper orthogonal decomposition”. In: *Current science*. 78.7 (2000), pp. 808–817.

[23] James W. Demmel. *Applied Numerical Linear Algebra*. 1997.

[24] Markus Holtz. *Sparse grid quadrature in high dimensions with applications in finance and insurance*. eng. Heidelberg, 2011.

[25] Gertjan van Zwieten et al. *Nutils*. Aug. 2018. URL: https://doi.org/10.5281/zenodo.1405137.

Ivar Eivind Palm

Reduced order modelling for linear elastic problems with geometric variation

NTNU
Norwegian University of
Science and Technology