GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

ISSN 1612-6793
Nr. ZFI-BM-2017-22

Bachelor's Thesis

submitted in partial fulfilment of the
requirements for the course "Applied Computer Science"

Model order reduction for linear PDE problems with the reduced basis method

Vivian Raulin

Institute of Computer Science

Bachelor's and Master's Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

August 24th, 2017

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
🌍 `www.informatik.uni-goettingen.de`

First Supervisor:	Jun.-Prof. Dr. Christoph Lehrenfeld
Second Supervisor:	Dr. Jochen Schulz

I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, August 24th, 2017

Abstract

Linear partial differential equations (PDEs) play an essential role in mathematics and many practical applications. Solving PDEs and especially parameter-dependent PDEs efficiently is important in engineering day-to-day work and is still an active field of research.

The problem of solving parameter-dependent PDEs efficiently is demonstrated by the example of the thermal block, a 2-dimensional surface that is heated in its interior and can have different thermal conductivities. The problem is defined as finding the temperature distribution for a specific heat conduction coefficient.

The important method that is discussed in this thesis is called reduced basis method and uses a fixed set of parameters for the parameter-dependent PDE. From this set there is a basis derived that is usable to solve many often changing parameters (many-query problem) with a significantly reduced complexity. We differentiate two classes of problems for solving parameter-dependent PDEs: Solving the problem just for one single parameter (one-query) versus solving it for many parameters.

We show how the reduced basis method can be used for efficiently calculating many-query problems. We take the simple example of a thermal block to demonstrate and visualize the method.

Besides presenting the mathematical foundations of the named methods, one further objective is to implement them. The algorithms are written in Python and built on the basis of the finite element libraries Netgen [1] and NGSolve [2].

Contents

Chapter 1

Introduction

Most physical processes are highly complicated and interactive systems and their behaviors are difficult to compute. Common examples are the propagation of sound, wind streams, force effects or fluid mechanics where many particles interact in a complex manner. The simulation of such systems is an important task in the applied sciences to understand them and derive knowledge to gain technological progress. Although they can often be described by partial differential equations, the outcome of such processes are hard to approximate.

One approach to solve these problems are finite element methods. These are commonly used for numerical approximations of real world scenarios that involve complex interactions in closed systems.

The solution of these numerical approximations can be computationally expensive. When working with parameter-dependent PDEs where the parameters are fixed (a one-query problem), higher computational time might be acceptable. For many-query tasks on the other hand, where we want to solve the given PDE for many different parameters, low computation time is pursued. This can be achieved by exploiting similarities of solutions for different parameters. Based on pre-computation where the problem is solved for a fixed set of parameters, the complexity reduction is achieved. We call the first step with high computational costs where the pre-computation is done *offline phase* and the following computation step of low cost *online phase*. The division into offline and online phase is a methodical approach used for many engineering tasks and systems.

One such system is given by the thermal block example where the objective is to model the heat distribution on a 2D or 3D surface under certain heat conductivities when heat is introduced on a specific location. The thermal block is subdivided into four equally sized blocks which have different thermal conductivities. When the heating is in the interior of the thermal block, we receive heat distributions from the basis functions in Figure 1.1 that are multiplied by coefficients. The framework that we use for these images and in general for building 2D and 3D models and mesh generation is Netgen [1] and the associated finite element solver library NGSolve [2].

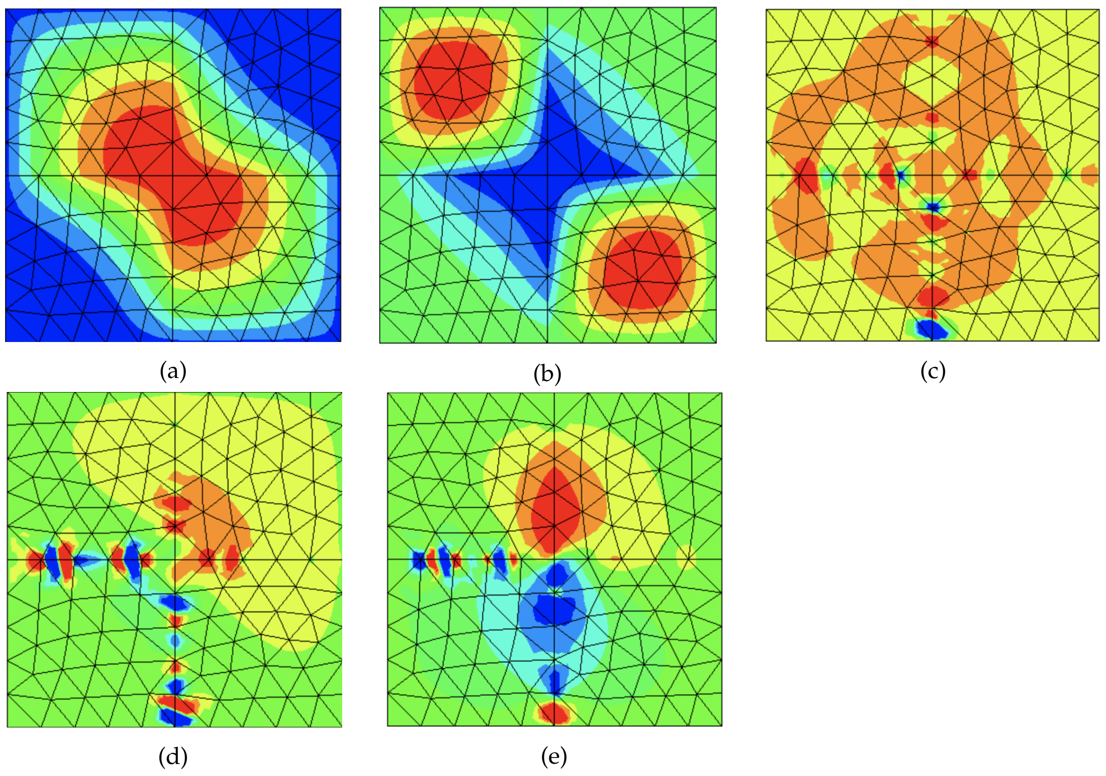

Figure 1.1: Orthonormalized basis functions for calculating the approximated heat distributions of the thermal block. While (a) to (c) are decent, we can see small disturbances in (d) and (e). However, those disturbances do not influence the results in a way that would be inacceptable.

A big part of this thesis consisted of building a detailed Python implementation for the reduced basis method by using above libraries. The code was used for detailed experiments and the consecutive evaluations of physical phenomena concerning the thermal block. Although the implementation was important, we emphasize on the mathematical foundations and results produced by the code instead of focusing on the implementation details.

In Chapter 2 a quick overview of the mathematical problem formulation is given. We introduce partial differential equations and present the physical formulation of the heat equation, especially of the Poisson equation. Furthermore, we have a look at the finite element method which is followed by the example of the thermal block. The reduced basis method is explained step-by-step in Chapter 3, looking both on the theoretical background and later on the implementation. In the end, we present the results for several evaluations in Chapter 3.3. It follows a conclusion in Chapter 4 with an outlook on future work.

Chapter 2

Mathematical Foundations

At first, we introduce partial differential equations and ordinary differential equations. After that, we present the finite element method by looking at the Poisson equation as an example which is the fundamental differential equation in this thesis.

2.1 Differential Equations

To introduce PDEs we need to recall the basics of functions and derivatives. The derivative of a function gives information about the rate of change. Functions can have multiple derivatives up to some order n. A function that consists of one single variable has *total derivatives*. In case the function consists of more than one variable it has *partial derivatives* where each derivative depends on one individual variable. In the following, we assume that all derivatives that we use are well-defined.

A differential equation is a mathematical equation which consists of both, a target function f and some of its derivatives. Differential equations thereby describe the relation between a function and its derivates and can be used to build a mathematical model of many real world processes.

Differential equations can be classified into ordinary or partial, as explained in the definitions 2.1.1 and 2.1.2. While ordinary differential equations depend on only one single variable, partial differential equations depend on multiple variables and therefore partial derivatives.

Further distinctions differentiate between linear and nonlinear PDEs. We will only deal with linear PDEs in this work.

Ordinary differential equations have the most simple form and can be defined in the following form:

Definition 2.1.1
Let F, f be functions depending on $y^{(n)}$ for $0 < n \leq \infty$. An *ordinary differential equation* is given by

$$F(x, y(x), y^{(1)}(x), \dots, y^{(n)}(x)) = 0 \qquad , \ x \in [0, T) \tag{2.1}$$

or

$$y^{(n)}(x) = f(x, y(x), y^{(1)}(x), \dots, y^{(n-1)}(x)) \qquad , \ x \in [0, T) \tag{2.2}$$

where $y : [0, T) \Rightarrow \mathbb{R}$ is the sought-after function.

A partial differential equation depends on several variables and partial derivatives:

Definition 2.1.2
Given the partial derivative $\frac{\partial f}{\partial x}$ with respect to x, a linear partial differential equation is:

$$\begin{aligned} &\text{Find} \quad f : \Omega \Rightarrow \mathbb{R} \\ &\text{so that} \quad F\left(x, y, f(x,y), \frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}, \dots, \frac{\partial^2 f(x,y)}{\partial x \, \partial y}\right) = 0 \qquad , \ x \in \Omega \end{aligned} \tag{2.3}$$

where Ω is the domain on which f is defined.

Additionally, PDEs can be classified in groups that have similar characteristics depending on the mathematical or physical behavior.

2.1.1 Heat Equation and Poisson Equation

To introduce the PDE of major interest in this thesis we will have a closer look at the heat equation.

The inhomogeneous heat equation or simply *heat equation* on a domain $\Omega \in \mathbb{R}^d$ is given as

$$\frac{\partial u(x,t)}{\partial t} - a \cdot \Delta u(x,t) = f(x,t) \tag{2.4}$$

where $a \in \mathbb{R}_{>0}$ is the material specific thermal conductivity, $x \in \Omega$ is the position, and $t \in [0, \infty)$ is the time. The heat equation describes the connection between the temporal and spatial change of the temperature $u(x,t)$ on a defined domain under influence of additional heating. In case that there is no change in time, i.e. the configuration is in an equilibrium state, there is $\frac{\partial u}{\partial t} = 0$ and we receive the simpler *Poisson equation*, a stationary problem that does not longer depend on t:

$$-a \cdot \Delta u(x) = f, \ x \in \Omega. \tag{2.5}$$

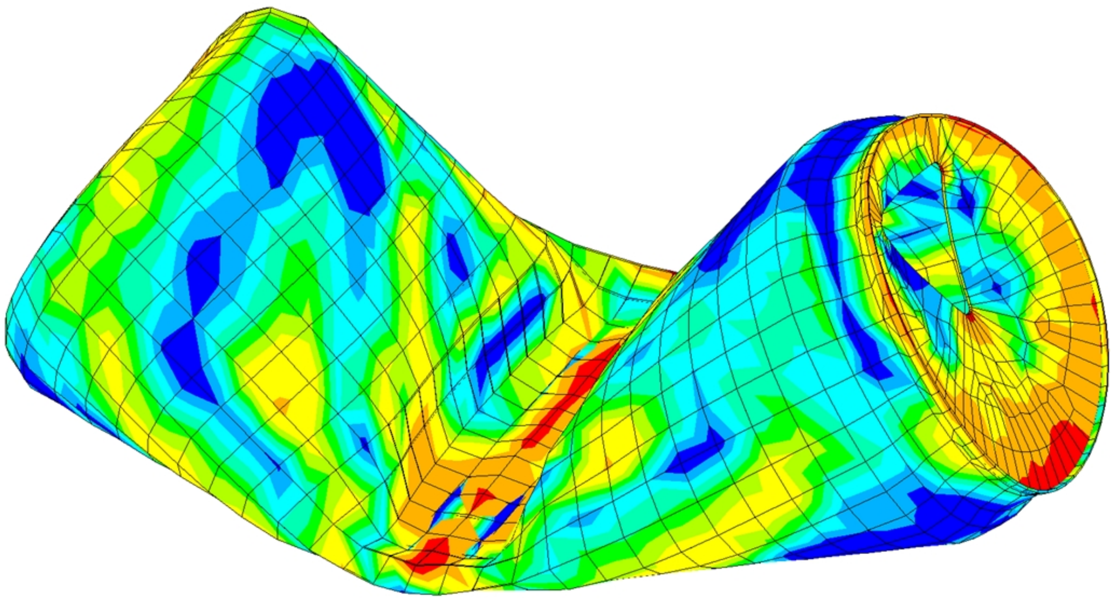

Figure 2.1: Grid approximation of a creased can [3]. The colors on the can show the load exerted on each cell by power influence. A similar approximation grid based on triangles is used for the thermal block to model the heat distribution.

Solving the Poisson Problem To be able to solve the Poisson equation on a bounded domain we need a boundary condition. We state the Dirichlet boundary condition for a given function $g : \Omega \Rightarrow \mathbb{R}$ as

$$u(x) = g(x),\ x \in \partial\Omega$$

to be able to solve the problem. We obtain

$$-a \cdot \Delta u(x) = f,\ \text{ in } \Omega \tag{2.6}$$

$$u = g,\ \text{ on } \partial\Omega. \tag{2.7}$$

To solve a differential equation there exist several approaches. Most used are approximative procedures in which PDEs are discretized due to the complexity of a problem that in most cases disallows closed form solutions. A good example of such a complex problem is shown in Figure 2.1.

2.2 Finite Element Method

The finite element method (FEM) belongs to the approximating PDE solvers which discretize the domain into disjunct parts of simple shape, e.g. triangles or rectangles, to simplify calculations.

Suitable to the given problem, different geometric forms are chosen that build a grid. Again, we can see for example that squares are chosen in Figure 2.1. Depending on the outer shape, the squares are deformed to get in form with the given body as can be seen on the right hand side of the can where some highly deformed squares are.

The Math Behind FEM The starting point of the FEM is the PDE that should be solved. A PDE has an alternative formulation, the *weak formulation* which is a variational form of the PDE. To introduce this weak form we require function spaces which describe a set (vector space) of functions which have well-defined and bounded first derivatives.
The concept of function spaces is important for the mathematical analysis of FEM, but not necessary in the context of this thesis. One typical function space is

$$V = H_0^1(\Omega) = \left\{ u : \Omega \to \mathbb{R}, \int_\Omega u^2 \,\mathrm{d}x < \infty, \int_\Omega ||\nabla u||_2^2 \,\mathrm{d}x < \infty, u|_{\partial\Omega} = 0 \right\}. \tag{2.8}$$

For details on these so-called *Sobolev spaces* we refer to a text book on PDEs, e.g. [4]. What remains for this thesis is that we look at V as a closed set (vector space) of functions which are "smooth" enough to define the differential operators involved in the PDE in a weak form. Typically, V is infinite dimensional. To compute approximations with V we make finite dimensional approximations of V, namely V_h.

The PDE can also be written in the *strong formulation* that consists of the equation itself and a boundary condition. Two boundary conditions are available:

- Neumann: The boundary of the solution is a value of the derivative, e.g. $\frac{\partial u}{\partial h} = g(x)$ on $\partial\Omega$.
- Dirichlet: The boundary of the solution is a function value of a given function, e.g. $u(x) = h(x)$ on Ω.

In the context of the FEM the *weak formulation* means that an integral is solved over Ω in connection with a test function $v \in V$. We transform the original PDE into a weak formulation which is more appropriate for the FEM. This step is described in detail in Section 2.2.1.

Solving the weak formulation is equivalent to finding a trial function $u \in V$ so that a bilinear form $B(u, v) \in \text{Fun}(V \times V, \mathbb{R})$ equals $F(v) \in \text{Fun}(V, \mathbb{R})$ for all $v \in V$. In the FEM, we search for a finite dimensional approximation:

$$\begin{aligned} \text{Find} \quad & u \in V \\ \text{so that} \quad & B(u,v) = f(v),\ \forall v \in V. \end{aligned} \tag{2.9}$$

In order to do so we consider a finite dimensional subspace $V_h \subset V$ and reformulate the weak form for this space (the h in the notation indicates the dependency of the solution on V_h which depends on the mesh with mesh size h):

$$\begin{aligned} \text{Find} \quad & u_h \in V_h \\ \text{so that} \quad & B(u_h,v_h) = f(v_h),\ \forall v_h \in V_h. \end{aligned} \tag{2.10}$$

All u_h and v_h are linear combinations of the basis $\{\phi_i\}_{i=1}^n$ with span$(\{\phi_i\}) = V_h$ performing $u_h(x,y) = \sum_{i=1}^n \mathbf{u}_i \cdot \phi_i(x,y)$.

From this finite dimensional formulation (2.10) we can derive a representation as a linear system for the coefficient vector $\mathbf{u} = (\mathbf{u}_i)_{i=1,\ldots,n}$. This last step is also illustrated in the practical Poisson example in Section 2.2.1.

2.2.1 The Poisson Problem

The basic problem to be solved in this thesis is the Poisson problem. We consider $f = 1$ and $g = 0$ and get the strong form

$$-a\Delta u = 1 \quad \text{in } \Omega, \tag{2.11}$$

$$u = 0 \quad \text{on } \partial\Omega. \tag{2.12}$$

To arrive at a weak formulation, Equation (2.11) is multiplied by a test function $v \in V$ and integrated over Ω what results in:

$$\int_\Omega -a \cdot \Delta u \cdot v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x\,,\ \forall v \in V. \tag{2.13}$$

After further partial integration this gives us:

$$\int_\Omega a \cdot \nabla u \cdot \nabla v \,\mathrm{d}x\,\mathrm{d}y = \int_\Omega f v \,\mathrm{d}x\,\mathrm{d}y,\ \forall v \in V. \tag{2.14}$$

We use the bilinear form $B(\cdot,\cdot)$ from (2.9) as well as the discrete and finite dimensional solution $u_h \in V_h$ from (2.10) in the following.

For the discrete u_h we use the basis $\{\phi_i\}_{i=1}^n$ of V_h:

$$u_h(x,y) = \sum_{i=1}^{n} \mathbf{u_i} \cdot \phi_i(x,y). \tag{2.15}$$

Using these results, the bilinear form in (2.10) is replaced by

$$B(u_h, v_h) \coloneqq \int_\Omega a \cdot \nabla v_h \cdot \nabla u_h \, \mathrm{d}x \, \mathrm{d}y \tag{2.16}$$

$$= \int_\Omega a \cdot \nabla \left(\sum_{i=1}^{n} \mathbf{v_i} \cdot \phi_i(x,y) \right) \cdot \nabla \left(\sum_{j=1}^{n} \mathbf{u_j} \cdot \phi_j(x,y) \right) \, \mathrm{d}x \, \mathrm{d}y \tag{2.17}$$

$$= \sum_{i=1}^{n} \mathbf{v}_i \sum_{j=1}^{n} \mathbf{u}_j \int_\Omega a \cdot \nabla\phi_i(x,y) \cdot \nabla\phi_j(x,y) \, \mathrm{d}x \, \mathrm{d}y \tag{2.18}$$

$$= \sum_{i,j=1}^{n} \mathbf{v}_i \cdot \mathbf{B}'_{ij} \cdot \mathbf{u}_j \tag{2.19}$$

with

$$\mathbf{B}'_{ij}(a)(x,y) = \int_\Omega a \cdot \nabla\phi_j \cdot \nabla\phi_i \, \mathrm{d}x \, \mathrm{d}y. \tag{2.20}$$

Analogously, we conclude that for

$$f(v_d) \coloneqq \sum_{i=1}^{n} \mathbf{f}_i \cdot \mathbf{v}_i \tag{2.21}$$

the following applies:

$$\mathbf{f}'_i = \int_\Omega f \cdot \phi_i \, \mathrm{d}x \, \mathrm{d}y \tag{2.22}$$

That leads to the equation for $u \in \mathbb{R}^n$:

$$\mathbf{v}^\top \cdot \mathbf{B}' \cdot \mathbf{u} = \mathbf{v}^\top \cdot f, \ \forall \mathbf{v} \in \mathbb{R}^n \tag{2.23}$$

which for $\mathbf{v} = \mathbf{e}_j$ (unit vector) implies

$$\sum_{i=0}^{n} \mathbf{B}'_{ji} \cdot \mathbf{u}_i = \mathbf{f}'_j, \ j = 1, 2, \ldots, n \tag{2.24}$$

which can be transformed to matrix notation

$$\mathbf{B}' \cdot \mathbf{u} = \mathbf{f}'. \tag{2.25}$$

The result is $\mathbf{u} = (\mathbf{B}')^{-1} \cdot \mathbf{f}'$. The computational complexity of solving this equation for $\mathbf{u}$ can be very high. The costs depend on the dimension n of the finite element space V_h. Typically n is very large, e.g. $n \in 1000, \ldots, 10^8$. In this thesis it is bound to $n = 1069$.

Chapter 3

Reduced Basis Method

The basic idea of the reduced basis method (RB method) is to improve the process of solving parameter-dependent PDEs such as the Poisson equation that depends on the heat conductivity. Typically, the finite element space is of very high dimension (between 10^3 and 10^8) what leads to enormous computational costs when solving the problem the naive way. The RB method solves in its offline phase the already mentioned computational expensive solutions for a few parameters. In the second phase, called online phase, the RB method solves many low-dimensional problems in low computational time. Therefore, the RB method is one approach to solve many-query problems as it can be used to generate quickly solutions to different parameterizations of a given problem. In comparison, the RB method is mostly unsuitable for one-query problems. The reduction of the dimension of high-dimensional problems is done by simplification, in particular by exploiting similarities of solutions from different parameters. The result, the reduced basis, should be found in computation time that is bound to $\mathcal{O}(N^3)$. Here is to differentiate that n is the dimension of the high-level dimension solution and N is the dimension of the low complexity solution.

In this chapter we will have a look at the theoretical background which is then explained step-by-step by the example of the thermal block that was shortly introduced in Chapter 1. Further information on the thermal block can be found in Section 3.2. For the computational background there are several pseudo codes added at the end of each step which give short overviews about the programming behind FEM and RBM in the language Python. In the end we examine some evaluations concerning the thermal block with help of the result of the new final basis, that consists of N representative high-dimensional basis vectors, and by the help of low-dimensional matrices.

3.1 Theoretical Foundations

The RB method is divided into offline and online phase: In the offline phase a nonparametric dimension reduction of the high dimensional problem is computed. The idea brakes down to

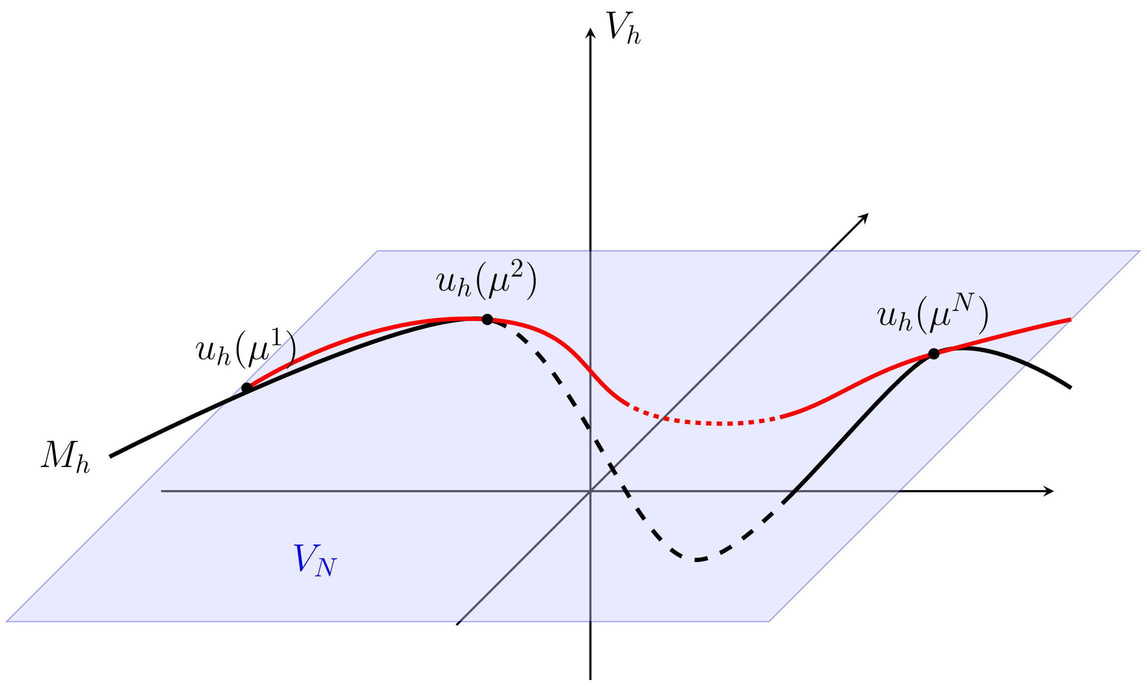

Figure 3.1: A reduced basis approximation of a function in V_h. The blue hyperplane V_N represents the reduced basis space. The black function M_h is the high-dimensional function we want to approximate (as done by the red function).

choosing parameter values α_i so that the span of all solutions to α_i provides a good approximation to all solutions in a (continuous) range of parameters. To be more precise we follow the notation from last chapter: The discrete finite element space V_h is replaced with V_N, a function space with very low dimension $N = \dim(V_h^{\text{red}}),\ N \ll n$.

In the online phase the reduced basis is used for fastly computing many-query solutions in our reduced problem space V_N.

The size N of the basis determines how good the approximation of the results are: The larger the basis, the lower is the error with the cost of increased computational complexity. If the basis is smaller, solutions can be calculated more quickly but will inevitably come with a greater error.

An introductory example Figure 3.1 characterizes the main idea of the RB method. We explain the sketch.

The low dimensional space V_N (blue plane) is a subspace (or hyperplane) of the finite high dimensional space V_h. The function M_h (black line) lies in V_h. We search an approximation of M_h that is restricted to V_N. After having done the RB approximation we receive a set of N different elements $\{u_h(\mu^i)\}_{i=1,\dots,N}$ (seen as red points in the sketch) and an approximate function that consists of $\{u_1, \dots, u_N\}$ in the subspace V_N and where $\{\mu_i\}_{i=1,\dots,N}$ are all possible parameters.

The difference between M_h and its approximation is almost zero in the chosen points $\{u_h(\mu^i)\}_{i=1,\dots,N}$. With increasing size N (the underlying basis), the difference between the approximation and M_h, here the area between the red and black functions, will decrease.

3.1.1 Orthonormalization

Now we have a set $T = \{t_i\}_{i=1,\dots,N}$ of vectors which spans the finite element space V_h. These t_i may have an almost linear dependency that we want to eliminate. If a matrix is symmetric and t_i build an orthonormal basis (ONB), then the condition number of this matrix is bounded. That guarantees algebraic stability to a certain extent.

In each step k (where $1 \leq k \leq n$) of the process we subtract linear combinations of the former $k-1$ vectors from the current vector t_k. For each step a new vector is projected onto the orthogonal space of the already created vectors. Each vector t_i should be orthogonal to all the other t_i's. Such an algorithm is given by the Gram-Schmidt orthonormalization which is explained in detail in Section 3.2.1. The output of the orthonormalization is an orthonormal matrix W of dimension n with the column vectors: $w_1, \dots, w_k$. The orthogonal basis $W = \{w_i\}_{i=1,\dots,N}$ produced of the set $T = \{t_i\}_{i=1,\dots,N}$ spans the same subspace of $\mathbb{R}^n$ as T.

3.1.2 Basis of a Reduced Space

Although having computed an orthonormal basis with the Gram-Schmidt algorithm from the α-dependent solution $\mathbf{u}$ for

$$\mathbf{B}' \cdot \mathbf{u}_\alpha = \mathbf{f}$$

from 2.25, $\mathbf{u}_\alpha$ ist still high-dimensional. Therefore we search for a dimension reduction, a projection between two finite element spaces, a high- and a low-dimensional space. We already have the high-dimensional space V_h and call the low-dimensional space V_N.

Equation (2.15)

$$u_n(a_k)(x,t) = \sum_{i=1}^{n} \mathbf{u}_i \cdot \phi_i \tag{3.1}$$

can be rewritten in another basis representation such that

$$u'_n(a_k)(x,t) = \sum_{j=1}^{n} \mathbf{u}'_j \cdot \phi_j. \tag{3.2}$$

Based on a parameter set S = $\{\alpha_i\}_{\alpha_i \in \{\alpha_i, i=1,\dots,N\}}$ we define $V_N = \text{span}\{u'(\alpha_i)\}_{\alpha_i \in \{\alpha_i, i=1,\dots,N\}}$, i.e. we set $\phi_i^N := u'(\alpha_i)$ as the basis of V_N. Any function $u_N \in V_N$ can be written as

$$u_N(x,y) = \sum_{i=1}^{N} \mathbf{u}_i^N \cdot {\phi_i}^N(x,y), \;\; \text{for } u^N \in \mathbb{R}^n. \tag{3.3}$$

Hence,

$$u_N = \sum_{i=1}^{N} \sum_{j=1}^{n} \mathbf{u}_i^N \cdot (\mathbf{u}_j^i \cdot \phi_j^n). \tag{3.4}$$

Reshaping gives us

$$u_N = \sum_{j=1}^{n} \sum_{i=1}^{N} \mathbf{u}_i^N \cdot \mathbf{u}_j^i \cdot \phi_j^n \tag{3.5}$$

$$= \sum_{j=1}^{n} \mathbf{c}_j^n \cdot \phi_j^n \tag{3.6}$$

for

$$\mathbf{c^n} = \sum_{i=1}^{N} \mathbf{u}_i^N \cdot \mathbf{u}_j^i \in \mathbb{R}^n \tag{3.7}$$

with

$$\mathbf{c}^n = \mathbf{Q} \cdot \mathbf{u}^N, \;\; \text{for } \mathbf{Q} \in \mathbb{R}^{n \times N}. \tag{3.8}$$

$\mathbf{Q}$ is the matrix that represents the change of basis between V_h and V_N, i.e. the N columns of $\mathbf{Q}$ correspond to the basis functions of V_N. We note that $\mathbf{Q}$ is originally obtained from the solution vectors $\mathbf{u}_i$ but may be changed by orthonormalization as in Section 3.1.1.

In the following we define the *reduced matrices* which will be needed later. For this purpose we redefine 2.10 so that:

$$\begin{aligned} \text{Find} \quad & u \in V_N \\ \text{so that} \quad & B(u^N, v^N) = f(v^N), \; \forall v^N \in V_N. \end{aligned} \tag{3.9}$$

Furthermore, we use (3.5) and insert it into (3.9) what finally results in

$$B(u^N, v^N) = \sum_{k,l=1}^{N} u_k^N(a_j) \cdot u_l^i \cdot B(\phi_k, \phi_l) \tag{3.10}$$

which, with Equation (3.4) and Equation (3.5), equals

$$\mathbf{Q}^\top \cdot \mathbf{B}' \cdot \mathbf{Q}. \tag{3.11}$$

Analogously,

$$f(\phi_i) = \mathbf{Q}^\top \cdot \mathbf{f}'. \tag{3.12}$$

So we conclude:

$$\mathbf{B}'_{\text{red}}(\alpha) = \mathbf{Q}^\top \cdot \mathbf{B}'(\alpha) \cdot \mathbf{Q} \tag{3.13}$$

$$\mathbf{B}'_{\text{red}}(1) = \mathbf{Q}^\top \cdot \mathbf{B}'(1) \cdot \mathbf{Q} \tag{3.14}$$

$$\mathbf{f}'_{\text{red}} = \mathbf{Q}^\top \cdot \mathbf{f}' \tag{3.15}$$

for $\mathbf{Q} \in \mathbb{R}^{n\times N}$.

3.1.3 The Greedy Algorithm and Offline Error Bounds

To refer back to the goal of the RB method we search now for an algorithm that gives us parameters α_i to build the reduced basis, i.e. helps to find a suitable set $P = \{\alpha_i\}_{i=1,\dots,N}$ as shown in Section 3.2.1. We still search for a low-dimensional approximation V_N for the naively solved solutions in V_h. At the beginning, the approximation depends on the solutions from an initial set P_0 which can still be insufficient for most parameters α.

We want to find an $\alpha \in P$ (the set of possible parameters) that maximizes the error between the approximated and the full-dimensional solutions.

A greedy algorithm chooses in each iteration of its execution that element which guarantees the maximal profit. In the case at hand the greedy algorithm picks the maximum error between the approximation of U_N and the full-dimensional problem. The error is usually assessed by a function evaluation, e.g. by an error estimation which we discuss next.

Having received a low-dimensional representation of the basis in $\mathbb{R}^N$, the next goal is to find a measure which chooses the next parameter α to enlarge the basis B. The error norm states how good the approximation works for another subset P_1 of P, the so-called "training set" and simulates the error behavior assuming to have included the elements of P_1.

To discuss the error estimation we start with the residual where $\mathbf{B}'(\alpha)$ is based on (2.20)

$$r(\alpha) = ||\mathbf{B}'(\alpha) \cdot \mathbf{u}_\text{n} - \mathbf{B}'(\alpha) \cdot \mathbf{u}_\text{h}|| \tag{3.16}$$

that describes the difference between an approximated, numerical solution u_n and the exact high dimensional solution u_h. We do a forecast for the error's value which depends both on the exact solution u_h and the approximated u_n. Since we are working on a function space we use

$$\int_\Omega u \cdot v \,\mathrm{d}x \,\mathrm{d}y \tag{3.17}$$

as an inner product which translates to

$$\langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{W}} = \mathbf{u}^\top \cdot \mathbf{W} \cdot \mathbf{v} \tag{3.18}$$

where $\mathbf{W}$ is the *mass matrix*. The output after calculating this error for each parameter in the list of possible parameters is a list of error values whose maximum can be chosen by the greedy algorithm. Having the input N as the predetermined size of the future reduced basis, we do N iterations of the greedy algorithm when starting from an initially empty set of parameters.

Reduced Basis A-Posteriori Error Bound As the error bound in Section 3.2.1 depends on the high-dimensional vectors $\mathbf{u}$ and $\mathbf{v}$ and the high-dimensional matrix $\mathbf{W}$ which are derived from the V_h, it results in high computational cost. That motivates an alternative error bound that not just predicts the error but can absolutely be computed with already done measurements in lower complexity $\mathcal{O}(N^3)$.

The *reduced basis a-posteriori error estimator* estimates the error after a computation. Here, it also computes a global error bound but neither takes into account the high dimensional solution or mesh-depending values. It is also driven from the a-priori error formula but transformed into an atomic form that consisting of the elementary matrices. For more details we can have a look at Section 3.2 in paragraph *a-posteriori theory*.

3.2 The Thermal Block

The thermal block represents a class of bodies that are heated and where the temperature flows from one side to another side. Furthermore, the temperature distribution is an embodiment of the solution to the Poisson equation. The simulation of the distributions is done by the Netgen framework (see Figure 3.3).

After having introduced the theoretical background of the RB method, a practical example is explained step-by-step.

Physics and Mathematics Let $\Omega \subset \mathbb{R}^n$ with $\Omega = [-1, 1] \times [-1, 1]$. Define equally sized subdomains $\Omega_i, i = 1, \dots, 4$ with coefficients $a(x) = 1$ in $\Omega_1 \cup \Omega_3$ and $a(x) = \alpha$ in $\Omega_2 \cup \Omega_4$ (as shown in Figure 3.3). The thermal conductivity α is chosen as $\alpha \in P = [0.1; 10]$.

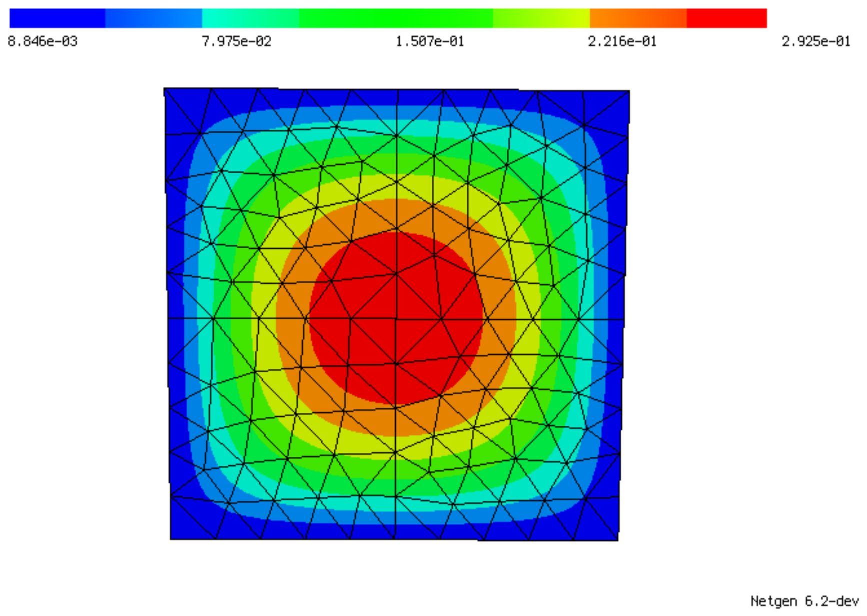

Figure 3.2: Exact heat distribution on Ω calculated for $\alpha = 1$ with the naive method.

The field variable is the temperature which satisfies the Poisson equation ($-a \cdot \Delta u = f$). The thermal block is constantly heated in the interior. The heat flows in the direction of lower temperatures. Depending on the thermal conductivities the four single blocks heat not equally strong. The system reaches a stable state after some time, where the heat distribution does not change anymore. This is called the stationary state: The temperature distribution and the heat flow are constant in time. Given this stationary state, mathematical computations are easier because the system does no longer depend on time.

Necessary for solving the PDE is primarily the knowledge of the boundary condition. For the boundary condition there are two possibilities: The boundary's temperature can be unaffected by the surrounding temperature u_{outside} and is then called *isolated* (Neumann-boundary condition). The boundary's temperature can as well converge to u_{outside}. Then there is an temperature exchange between the block and u_{outside} (Dirichlet-boundary condition).

For the following computations we introduce notations for matrices that were already defined in Chapter 2, Equation 2.20:

$$\mathbf{A} := A_{ij}(\alpha) = \alpha \cdot \int_{\Omega_1 \cup \Omega_3} \nabla \phi_j \cdot \nabla \phi_i \tag{3.19}$$

$$\mathbf{B} := B_{ij}(1) = \int_{\Omega_2 \cup \Omega_4} \nabla \phi_j \cdot \nabla \phi_i. \tag{3.20}$$

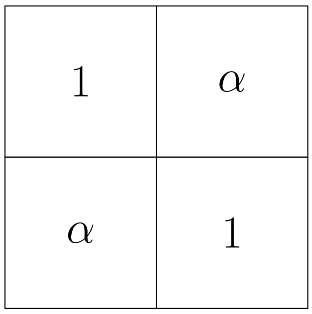

Figure 3.3: Subdivision of Ω into 4 parts. In the top left and bottom right the thermal conductivity is kept constant: $a(\Omega_2) = a(\Omega_4) = 1$. In the two remaining parts the thermal conductivity is varied: $a(\Omega_1) = a(\Omega_3) = \alpha$.

3.2.1 Practical Realization

The Poisson problem for the thermal block has already been solved in Chapter 2 (via the finite element method). In this chapter there will be a detailed explanation concerning the RB method to show a way to efficiently calculate solutions.

Build Finite Element Matrices At first, we build up a finite element space, here called V_h. The dimension of V_h is $n = 1069$. A mesh is computed which is based on the thermal block's basic forms. The software's blackbox outputs high-dimensional sparse matrices $\mathbf{A}_1$ (depending on α), and $\mathbf{B} \in \mathbb{R}^{1069\times1069}$, based on the weak problem from Equation (2.14). Analogously, we define the vector $\mathbf{f}$ which is a quotient of the heat source and the heat capacity (Equation (2.22)). For an initial list of widespread α_i's we compute the high-dimensional base solutions

$$\mathbf{u}_{\alpha_\mathbf{i}} = (\mathbf{A} \cdot \alpha_i + \mathbf{B})^{-1} \cdot \mathbf{f}. \tag{3.21}$$

These solutions $\mathbf{u}_{\alpha_i}$ for $\alpha_i \in \{0.1, 0.2, \dots, 10\}$ build the first basis. We remember that the linearly independent vectors of a basis span a subspace in V_h. That means the new matrix (from here on called $\mathbf{\Sigma}$) spans a space for all solutions which are possible for a certain load of coefficients.

Gram-Schmidt process The Gram-Schmidt process (GSP) is a method for orthogonalization or orthonormalization of a set of vectors and is here used to transform the given basis $\mathbf{\Sigma}$. Orthogonal vectors (rows or columns) $\mathbf{v}_i$ and $\mathbf{v}_j$ fulfill: $\langle \mathbf{v}_i, \mathbf{v}_j \rangle_\mathbf{W} = 0$. If the vectors are additionally orthonormal, the vectors $\mathbf{v}_i$ fulfill: $||\mathbf{v}_i||$ = 1. The GSP is useful to eliminate linear dependencies in matrices. Let $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^n$ be linear independent vectors which are the former solutions $\mathbf{u}_{\alpha_i}$ from Equation (3.21). The resulting vectors $\mathbf{w_1}, \dots, \mathbf{w_n} \in \mathbb{R}^n$ are columnwise stored in the matrix $\mathbf{\Sigma}$. Now the basis matrix is out of linear dependencies and further computations are easier.

Algorithm 1 Generate Matrices/Solve Problem

```
procedure GENERATEMESH
    order ← 3
    boundaryCondition ← $Dirichlet$
    $V \leftarrow FiniteElementSpace(order, boundaryCond)$
    $mesh \leftarrow [-1, 1] \times [-1, 1]$
procedure GETFEMMATRIX
    $u \leftarrow trialfunction$
    $v \leftarrow testfunction$
    $\mathbf{FEMMatrix} \leftarrow BilinearForm(a \cdot \nabla u \cdot \nabla v)$
procedure GETRHSVECTOR
    $v \leftarrow testfunction$
    $\mathbf{RHSVector} \leftarrow LinearForm(v)$
procedure INITIALIZATION
    initialList ← [0.1, 1, 10]
    listOfParameters ← [0.1, . . . , 10]
    $\mathbf{A}$ ← GetFEMMatrix([1,0,1,0])
    $\mathbf{f}$ ← GetRHSVector(1)
procedure SOLVEPROBLEM
    for each element in initialList do:
        $\mathbf{\Sigma}.\mathbf{column}(\mathbf{i}) \leftarrow \mathbf{A}^{-1} \cdot \mathbf{f}$
```

The Reduced Problem Still having to deal with huge matrices the next step is a projection from the high-dimensional space to the low dimensional space. The original problem of solving PDEs can be partially simplified and is represented by the *reduced problem*. The reduced problem depends on the three matrices $\mathbf{A}_{\text{red}}, \mathbf{B}_{\text{red}} \in \mathbb{R}^{N \times N}$ and $\mathbf{f}_{\text{red}} \in \mathbb{R}^{N}$ which are the result of the dimension reduction. Thereby, the crucial parameter of the problem is the given parameter N. If N is chosen small, the solution can be calculated fast with the reduced problem. However, the solution will be inaccurate. The larger we choose N, the better the solution will get by the cost of increased of computational complexity. We get new matrices $\mathbf{A_{red}}$ and $\mathbf{B_{red}}$ which have a dimension $N \ll n$.

Algorithm 2 Gram-Schmidt

```
procedure ORTHONORMALIZATION($\mathbf{\Sigma}$)
    $\mathbf{A}_{norm}$ ← GetMatrix([1, 1 ,1 ,1])
    $k$ ← shape($\mathbf{\Sigma}$)
    for i in [1, . . . , size-1] do:
        $\mathbf{w}_i \leftarrow \mathbf{\Sigma}$.column(i)
        $\mathbf{v}_i \leftarrow \mathbf{w}_i - \sum_{j=1}^{i} \frac{\langle \mathbf{v}_j, \mathbf{w}_i \rangle}{\langle \mathbf{v}_j, \mathbf{v}_j \rangle} \cdot \mathbf{v}_j.$
        $\mathbf{v}_i \leftarrow \frac{\mathbf{v}_i}{||\langle \mathbf{A}_{norm} \cdot \mathbf{v}_i, \mathbf{v}_i \rangle||}$
    $\mathbf{W}$.column(i) ← $\mathbf{v}_i$
```

The functionality of the code lines 2 to 4 in Algorithm 3 is explained in Section 3.1.2 because this step is important since it realizes the dimension reduction. Having received the reduced matrices

Algorithm 3 Reduced Problem

```
1: procedure COMPUTE REDUCED MATRICES
2:     $\mathbf{A}_{\text{red}} \leftarrow \mathbf{\Sigma}^\top \cdot \mathbf{A} \cdot \mathbf{\Sigma}$
3:     $\mathbf{B}_{\text{red}} \leftarrow \mathbf{\Sigma}^\top \cdot \mathbf{B} \cdot \mathbf{\Sigma}$
4:     $\mathbf{f}_{\text{red}} \leftarrow \mathbf{\Sigma}^\top \cdot \mathbf{f}$
5:     $\mathbf{M} \leftarrow \mathbf{A} + \mathbf{B}$
6:     $i \leftarrow 0$
7:     for coefficient in verificationList do:
8:         $\mathbf{solution}_{\text{red}} \leftarrow \mathbf{M}^{-1} \cdot \mathbf{f}_{\text{red}}$
9:         $\mathbf{coefficients}_i \leftarrow \mathbf{solutions}_{\text{red}}$
10:        i ← i + 1
```

and vectors in lines 2 to 4 we can redefine the naive solution from Equation (3.21) as:

$$\mathbf{u}_{a_{\text{red}}} = \mathbf{A}_{a_{\text{red}}}^{-1} \cdot \mathbf{f}_{\text{red}} \tag{3.22}$$

Error Estimation For the very first iterations the difference between the original and the approximated solution will be too large for most parameters in the interval of possible parameters. The algorithm needs a method to choose a parameter in order to add it to the initial basis to reduce this difference. Therefore, we want to find those parameters for which the approximation is worst.

To compare are the two vectors

$$\mathbf{u}'_a = \mathbf{A}_a^{-1} \cdot \mathbf{f} \tag{3.23}$$

and

$$\mathbf{u}_a = \mathbf{\Sigma} \cdot (\mathbf{A}_{a_{\text{red}}}^{-1} \cdot \mathbf{f}_{\text{red}}) \tag{3.24}$$

for a specific $\alpha \in \mathbb{R}^n$. In Equation (3.23) $\mathbf{u}'_a$ is the solution of the high-dimensional problem. The solutions for $\mathbf{u}'_a$ depending on α span the image room of possible solutions. Also, $\mathbf{u}_a$ results from solving the reduced problem and multiplying the result to the orthonormal basis.
The error norm consists of

$$e_a = \sqrt{(\mathbf{u}_a - \mathbf{u}'_a)^\top \cdot \mathbf{A}_{\text{norm}} \cdot (\mathbf{u}_a - \mathbf{u}'_a)} \tag{3.25}$$

where $\mathbf{A}_{\text{norm}}$ is a *weighting matrix* that weights the error over the area of Ω. The algorithm chooses the maximal e_a and appends its coefficient a to the list of the coefficients belonging to the reduced basis.

The operation in code line 7, Algorithm 4 is really expensive because we have a matrix of dimension 1069×1069. Usually, the inverting of a matrix is done in $\mathcal{O}(n^3)$ where n is the matrix dimension. As

Algorithm 4 Error Estimation

```
1: procedure COMPUTE ERROR($\Sigma$, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{f}$)
2:     index $\leftarrow 0$
3:     i $\leftarrow 0$
4:     for coefficient in verificationList do:
5:         $\mathbf{A_2} \leftarrow a \cdot \mathbf{A} + \mathbf{B}$
6:         $\mathbf{u} \leftarrow (\Sigma, \mathbf{c}_{index})$
7:         $\mathbf{u_2} \leftarrow \mathbf{A}_2^{-1} \cdot f$
8:         $\mathbf{A_e} \leftarrow (\mathbf{A}_{\text{norm}}, \mathbf{u} - \mathbf{u_2})$
9:         $error \leftarrow \sqrt{(\mathbf{A_e}, \mathbf{u} - \mathbf{u_2})}$
10: procedure GREEDY ALGORITHM
11:     $errorList \leftarrow$ List of Errors
12:     $maxError \leftarrow max(errorList)$
13:     $initialList \leftarrow initialList.append(maxError)$
14:     goto Algorithm 1.
```

the reduced basis method has the purpose to guarantee lower computational costs, it is impractical to have such high costs for each parameter in the parameter list. That motivates a computationally cheaper error bound that we discuss in the *a-posteriori theory*.

Evaluation of Error Boundaries in Four Iterations Figure 3.4 shows the errors in four iterations for every possible parameter α from the parameter list.

With every new added parameter the error decreases around a tenth. In Figure 3.4 we see that the error for each added parameter is zero. The values between those points, where the error approximately vanishes, shrink with every iteration. The maxima are approximately in the middle between every two parameters from our current basis. In conclusion, with increasing basis size the error strongly decreases as was the goal. By choosing a bigger final basis it is possible to get even better results, i.e. lower errors between approximation and real result, but for the cost of computational time.

A-Posteriori Theory The a-posteriori estimator is an alternative formulation for the global error bound that can be quickly calculated after the offline phase. The computational complexity is $\mathcal{O}(N^2)$ compared to $\mathcal{O}(n^3)$ from the former error formulation. We have a look at the results and see how big our error between the naive and the approximated solution is at the current iteration. The a-posteriori error estimator depends on already computed terms after having done some measurements. In the beginning, its formula equals the a-priori error estimator but is then transformed. It follows that

$$e_a = \sqrt{||\mathbf{A}_a \cdot \mathbf{u}_a - \mathbf{f}||^2_{\mathbf{A}^{-1}}} \tag{3.26}$$

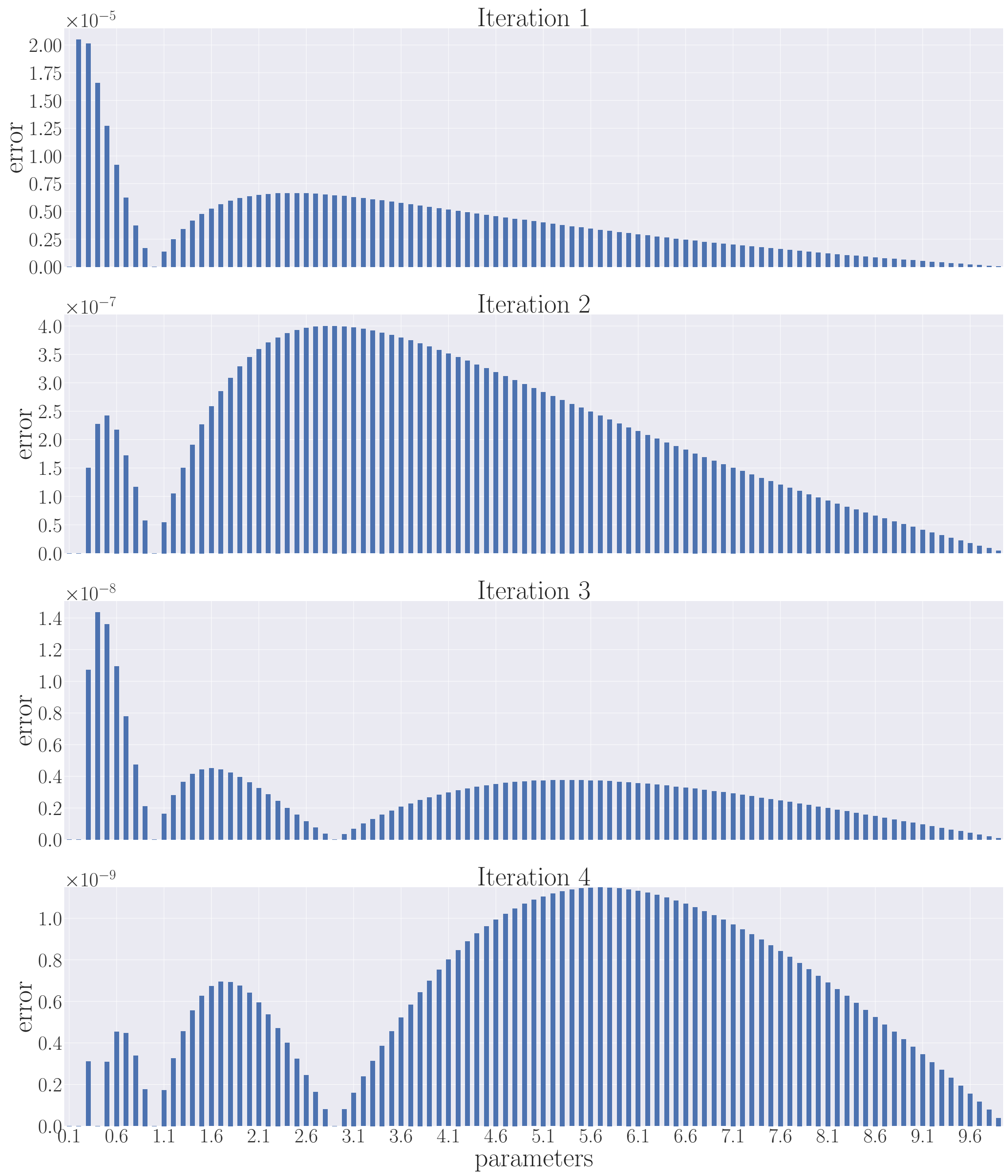

Figure 3.4: Errors between approximated and exact solutions for different iterations during offline phase. The initial basis is $[0.1, 1, 10]$, hence the errors at the respective parameters are nearly zero in iteration 1. The greedy algorithms chooses $\alpha = 0.2$ as next additional parameter for building our basis since it had the highest error in iteration 1. This procedure continues for the subsequent iterations. We observe that the error decreases already by 4 orders of magnitude through the shown iterations.

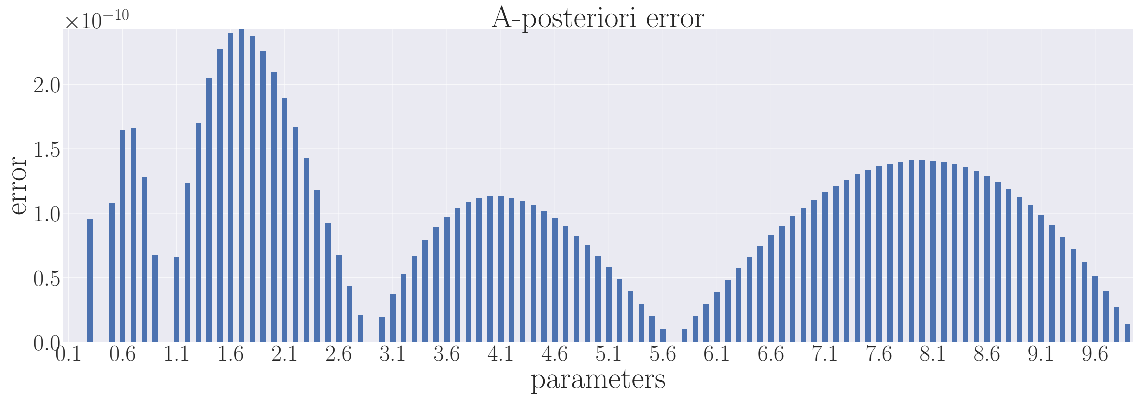

Figure 3.5: Error between approximated and exact solution with the a-posteriori error estimation. The final basis for $N = 7$ consists of the parameters $[0.1, 0.2, 0.4, 1.0, 2.9, 5.7, 10]$.

can be written in a form which mainly consists of matrices and vectors which have been computed during the offline phase and partly consists of some vectors which can computed with low-level costs. The error formula

$$e_\alpha = \sqrt{a + \mathbf{c}_{\text{red}}^\top \cdot \mathbf{b_1} + \alpha \cdot \mathbf{c}_{\text{red}}^\top \cdot \mathbf{b_2} + \mathbf{c}_{\text{red}}^\top \cdot \mathbf{D_1} \cdot \mathbf{c}_{\text{red}} + \alpha \cdot \mathbf{c}_{\text{red}}^\top \cdot \mathbf{D_2} \cdot \mathbf{c}_{\text{red}} + \alpha^2 \cdot \mathbf{c}_{\text{red}}^\top \cdot \mathbf{D_3} \cdot \mathbf{c}_{\text{red}}} \tag{3.27}$$

consists of

$$\begin{aligned}
\mathbf{a} &= \mathbf{f}^\top \cdot \mathbf{A}^{-1} \cdot \mathbf{f} \\
\mathbf{b_1} &= -2 \cdot \mathbf{f}^\top \cdot \mathbf{A}^{-1} \cdot \mathbf{B} \cdot \mathbf{\Sigma} \\
\mathbf{b_2} &= -2 \cdot \mathbf{f}^\top \cdot \mathbf{A}^{\mathbf{-1}} \cdot \mathbf{A_1} \cdot \mathbf{\Sigma} \\
\mathbf{D_1} &= \mathbf{B} \cdot \mathbf{\Sigma} \cdot \mathbf{B}^\top \cdot \mathbf{A}^{-1} \cdot \mathbf{B} \cdot \mathbf{\Sigma} \\
\mathbf{D_2} &= \mathbf{B} \cdot \mathbf{\Sigma} \cdot \mathbf{A}^{-1} \cdot \mathbf{A_1} + \mathbf{A_1}^\top \cdot \mathbf{A}^{-1} \cdot \mathbf{B} \cdot \mathbf{\Sigma} \\
\mathbf{D_3} &= \mathbf{A_1} \cdot \mathbf{\Sigma} \cdot \mathbf{A_1}^\top \cdot \mathbf{A}^{-1} \cdot \mathbf{A_1} \cdot \mathbf{\Sigma}.
\end{aligned}$$

The final basis $\mathbf{\Sigma}$ is of size N and $\mathbf{c}_{\text{red}}$ is the solution of the reduced problem and therefore easy to compute. All other matrices from the offline phase can be stored what results in a running time of $\mathcal{O}(n^2)$.

The errors for all parameters in iteration 4 are depicted in Figure 3.5 and were computed using the a-posteriori error estimator.

3.3 Evaluation in Online Phase

Having build up the reduced basis, we can evaluate functions concerning the thermal block like the heat flow or the temperature. Examples include the thermal resistance or heat fluctuation between domains of higher and lower temperature.

3.3.1 Evaluation of Functionals

At the present step, all values are available which are needed for evaluation. In the following, we do several online calculations. An important ingredient is the mass matrix which can be summed up element wise and returns the entire temperature.

Definition 3.3.1
The *mass matrix* is a bilinear form $\mathbf{M} \in \mathbb{R}^{n \times n}$ so that $\mathbf{u}^\top \cdot \mathbf{M} \cdot \mathbf{v} = \int_\Omega u \cdot v$.

Definition 3.3.2
Let V be a K-vector space with $K \in \mathbb{R}, \mathbb{C}$.
A functional T is a function $T\colon\ f \in V \to T[f] \in K$.

A functional over V_h is a linear mapping from V_h to $\mathbb{R}$. A functional of interest can be an integral belonging to a function for computing a mass or an area. Also, it can be a norm's function or an inner product of a function.

Example 1: Average Temperature

At first, we will look at the average temperature on Ω depending on the chosen input a. For this purpose we reduce the mass matrix $\mathbf{M}$ to $\mathbf{M_{red}}$.

$$u_{\text{average}} = \frac{\int_\Omega u \,\mathrm{d}x}{\int_\Omega 1 \,\mathrm{d}x} = \frac{1}{C} \cdot \int_\Omega u \,\mathrm{d}x = \frac{1}{C} \cdot \mathbf{u}^\top \cdot \mathbf{M} \cdot \mathbf{v} \tag{3.28}$$

Herein, C is the size of the area, $\mathbf{v}$ is considered to be a vector with just ones and $g = \frac{1}{C} \cdot \mathbf{M}_{\text{red}} \cdot \mathbf{v}$ can be computed in the offline phase whereas the reduced problem $\mathbf{u} = (\alpha \cdot \mathbf{A}_{\text{red}} + \mathbf{B}_{\text{red}})^{-1} \cdot \mathbf{f}_{\text{red}}$ needs to be solved in the online phase. The average temperature for each parameter is visualized in Figure 3.6.

Example 2: Heat Flow on Boundary

A second evaluation of interest is the heat flow on the upper boundary of the thermal block. The heat flow $\dot{Q}$ is the quotient of the transferred heat depending on the past time. In case of a

Algorithm 5 Evaluation of Functionals

```
1: procedure GETAVERAGETEMPERATURE(C,finalList)
2:     $\mathbf{M} \leftarrow GetMassMatrix$
3:     $\mathbf{M_{red}} \leftarrow ReduceMatrix(\mathbf{M})$
4:     $\mathbf{v} \leftarrow onesVector$
5:     $\mathbf{g} \leftarrow C \cdot \mathbf{M} \cdot \mathbf{v}$
6:     for $\alpha \in finalList$ do
7:         $\mathbf{u}_\alpha \leftarrow SolveReducedProblem([\alpha, 1, \alpha, 1])$
8:         $averageTemperature_\alpha \leftarrow \mathbf{u}_\alpha \cdot \mathbf{g}$
9:     return $averageTemperature$
10: procedure GETMASSMATRIX
11:     $u \leftarrow trialfunction$
12:     $v \leftarrow testfunction$
13:     $\mathbf{M} \leftarrow BilinearForm(u \cdot v)$
14:     return $\mathbf{M}$
```

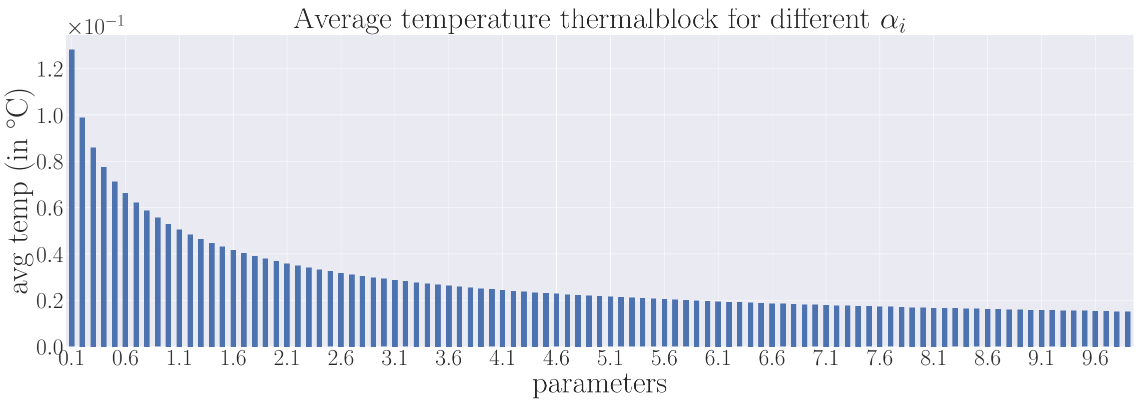

Figure 3.6: Average temperature on Ω for α in $[0.1, 0.2, \dots, 10]$. As expected, the temperature is higher for smaller values α because the thermal conductivity is lower. The average temperature decreases exponentially with increasing thermal conductivities.

stationary case, we look at the heat flow over a certain area. The heat flow is also described by Fourier's first law in the multidimensional extension:

$$\dot{q} = -\lambda \cdot \nabla T. \tag{3.29}$$

The *temperature gradient* ∇T shows in which direction and how strong the temperature raises for all points in the vector field T.

The thermal block presents a 2D surface, so we assume the temperature gradient to be

$$\nabla T = \left(\frac{\delta T}{\delta x}, \frac{\delta T}{\delta y} \right). \tag{3.30}$$

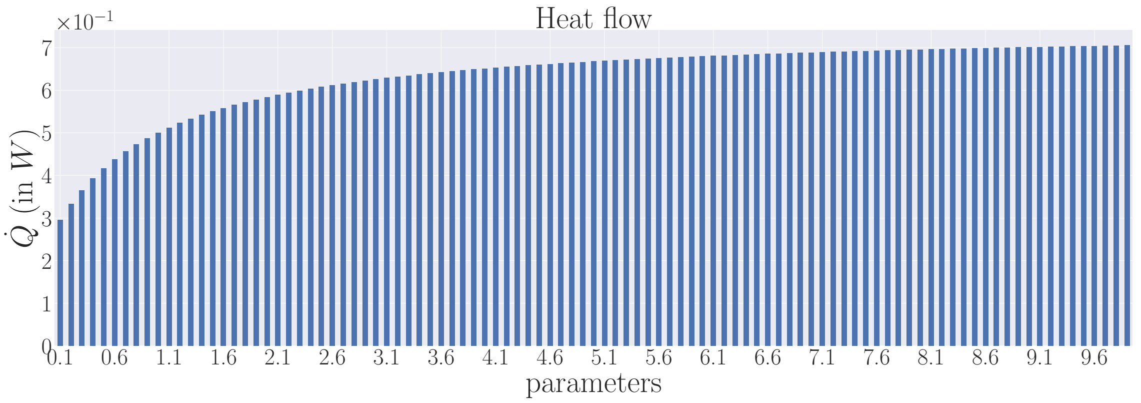

Figure 3.7: Heat flow $\dot{Q}$ for upper boundary Γ_{top} of the thermal block. The heat flow increases for increasing thermal conductivity $\alpha \in [0.1, 0.2, \ldots, 10]$.

We look at the heat flow over a straight line, so the formula for $\dot{Q}$ changes to

$$\dot{Q} = \int_{\text{upperBoundary}} \dot{q} \cdot \mathbf{n} \, \mathrm{d}s \tag{3.31}$$

with $\mathbf{n}$ being the normal vector that is orthogonal to the upper boundary.

Further calculations give:

$$\dot{q} \cdot \mathbf{n} = \begin{pmatrix} -\lambda \cdot \frac{\partial T}{\partial x} \\ -\lambda \cdot \frac{\partial T}{\partial y} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = -\lambda \cdot \frac{\partial T}{\partial y}. \tag{3.32}$$

Replacing $\dot{Q}$ from (3.31) with the linear form $\mathbf{f}(\mathbf{v})$ we receive:

$$\mathbf{f}(\mathbf{v}) = \int_{\text{upperBoundary}} -\lambda \cdot \frac{\partial v}{\partial y}. \tag{3.33}$$

The result of the heat flow will be

$$\dot{Q} = \mathbf{f}_{\text{red}} \cdot \mathbf{u}_{\alpha_{red}} \tag{3.34}$$

as pictured in Figure 3.7.

3.3.2 Comparison of Corresponding Computing Times

To refer to the goal of the reduced basis method we would like to show that the implementation of the solutions in the online phase is way more efficient than the naive computation. To display the

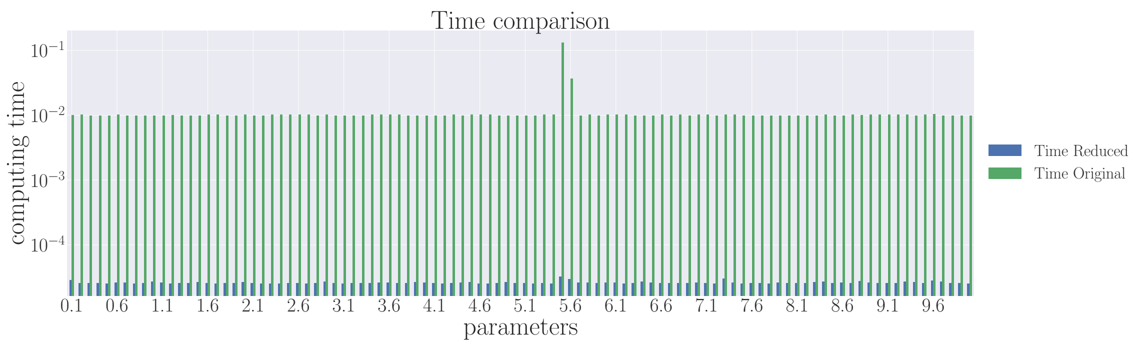

Figure 3.8: Logaritmic scaled comparison between computing times for original naive approach and reduced basis method. The improvement is around 3 orders of magnitude and the parameters are taken from $[0.1, 0.2, \ldots, 10]$.

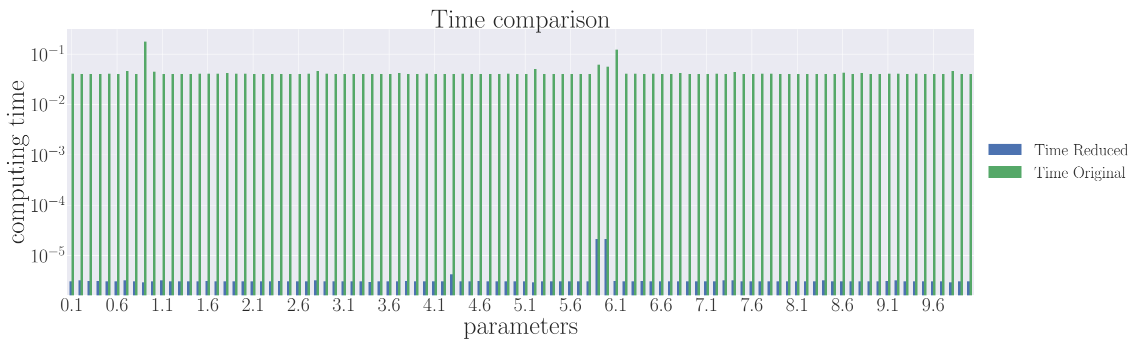

Figure 3.9: Logaritmic scaled comparison between computing times for original naive approach and reduced basis method. The main difference to 3.8 is that the order of the finite element space is changed to $h = 4$. The improvement is around 4 orders of magnitude and the parameters are taken from $[0.1, 0.2, \ldots, 10]$.

improvement in computational time we compare the measured execution times of the original and the reduced solution.

The large improvement in computation time, as shown in Figure 3.8 and Figure 3.9, acknowledges what we have presented so far. For the many-query problem of the thermal block with changing thermal conductivities, the parameter-dependent PDE is efficiently solved with the help of the reduced basis method. The more complex or high-dimensional the finite element space V_h is chosen, the more time is saved by the reduced basis method. The code is executed in running time $\mathcal{O}(N^3)$ but could be done even more efficiently by using methods for the evaluation of quadratic terms or more efficient methods for inverting matrices as the Cholesky decomposition.

Chapter 4

Summary

We presented the theoretical foundation of parameter-dependent PDEs and the finite element method for simulating complex physical processes. In the course thereof, ordinary and partial differential equations were introduced, the Poisson equation was presented and subsequently the Poisson problem was derived from the heat equation.

The thermal block with additional heating in the interior served as a real world example to comprehend and visualize application of PDEs. The problem of the thermal block can be summarized as finding the heat distribution on a two dimensional surface with different thermal conductivities when it is heated.

With the reduced basis method a technique for efficiently solving the problem at hand was introduced. We looked at the major steps of the reduced basis method: (1) the orthonormalization of the exact high-dimensional solutions to build a representative basis of the problem space; (2) the building of the reduced matrices depending on the orthonormal basis; (3) and the search for an appropriate parameter that is appended to the extended basis matrix.

Additionally, evaluations were done regarding the error between the exact and approximated solutions. The residuals decrease with increased basis size N. With already 4 iterations the decrease of the error amounts to 4 orders of magnitude. Furthermore, we modeled the average temperature for a discrete set of thermal conductivities and computed the heat flow over certain domains on the thermal block. We noticed that the heat flow increases with increased thermal conductivity.

The main focus was not only to represent the above methods formally, but also to implement and test every single step practically. We chose Python with Netgen and NGSolve for this task and evaluated the time performance of the reduced basis method. The difference of computation time between the high-dimensional full problem and the approximated solution shows an improvement of around 3 to 4 orders of magnitude.

Small issues existed with the the high-dimensional solutions from the FEM which later served as an orthonormal basis for the reduced basis method. The solutions were subject to disturbances with the result that some of the basis vectors had small inaccuracies. Also, more efficient implementations of parts of the used algorithms might be possible to achieve greater gains in running time.

Possible steps to further explore the reduced basis method and the thermal block could involve looking at the non-stationary heat equation or varying the boundary conditions of the thermal block.

However, the work as presented here succeeds at giving a self-contained view on the theoretical and practical problems as well as solutions that arise when dealing with numerical finite element methods and specifically the reduced basis method.

Bibliography

[1] J. Schöberl, “NETGEN - An Advancing Front 2D/3D Mesh Generator Based on Abstract Rules,” *Comput Vis Sci*, vol. 1, no. 1, pp. 41–52, 1997.

[2] H. Eggert, J. Gopalakrishnan, and C. Lehrenfeld, “NGSolve.” [Online]. Available: ngsolve.org

[3] J. E. Akin, *Finite Element Analysis Concepts*, Rice University, 2010.

[4] L. Evans, “Partial Differential Equations,” p. 672, 1996. [Online]. Available: http://books.google.com/books?hl=en{&}lr={&}id=okkLW-Nw5oQC{&}oi=fnd{&}pg=PR13{&}dq=Partial+differential+equations{&}ots=11wIYjAVSb{&}sig=j2h1z0i8lUtZU{_}5gHqTALO3hoe8{%}5Cnhttp://books.google.com/books?hl=en{&}lr={&}id=okkLW-Nw5oQC{&}oi=fnd{&}pg=PR13{&}dq=Partial+differential+equatio