Auto-generated structured meshes for evolving domains

Dissertation

zur Erlangung des Grades eines Doktors der Naturwissenschaften

Bergische Universität Wuppertal

Fakultät für Mathematik und Naturwissenschaften

vorgelegt von

Camilla Hahn

Wuppertal, 2021

The PhD thesis can be quoted as follows:

urn:nbn:de:hbz:468-20210914-112409-1
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20210914-112409-1

DOI: 10.25926/gmyp-7s53
[https://doi.org/10.25926/gmyp-7s53]

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

Within the scope of meshing for finite element methods, two major problems arise, where the solution of one often prevents the solution of the other. The first problem is the accuracy of the mesh. Assuming polyhedral meshes, these meshes can always only approximate non-polyhedral domains up to a certain error. Nevertheless, modern meshing approaches manage to approximate even complicated domains very well with unstructured meshes [34]. The second problem is the computational cost. Especially in three dimensions, the problem size very quickly exceeds the capacities of standard solvers. Highly structured grids have proven to deliver speedup on modern computer architectures and especially GPU's [16, 43]. Following either a structured or an unstructured meshing approach always entails the loss of the advantages of the other. Therefore, research in recent years focuses on the development of new structured meshing approaches that succeed in capturing ever more complicated domains, such as the cut cells method [5]. In the context of shape optimization, where usually hundreds or thousands of the optimization steps with subsequent but ever changing domains have to be solved, an additional question arises. The domains do change in each optimization step but in general these changes are not very large. Instead of creating entirely new meshes in each step of optimization, many approaches of meshing in shape optimization focus on morphing the mesh from one domain to the next [50, 51].

In this work, we present a highly structured meshing approach that is based on the concept of composite finite elements presented in [24, 25, 26]. It possesses the advantages of structured meshes and is capable of resolving boundaries accurately. Embedded in a gradient based, PDE constrained shape optimization procedure, this highly structured approach leads to cost reduction in the re-meshing and the assembly of the system matrix of the governing PDE. We also present an approach to apply Krylov subspace recycling to speed up the computations further.

GIVEN Project

This work was conducted as part of the GIVEN project (Shape Optimization for Gas Turbines in Volatile Energy Networks). The project is concerned with the development of a novel process for multi-objective shape optimization of gas turbine blades. The rise of renewable energies has led to an increase of volatility in the energy supply in energy networks in recent years. Not only due to the resulting increase in the frequency of starts and shut downs of the gas turbines in power plants, it has become inevitable to determine the design decisions of gas turbines not only by efficiency but the durability of the components as well. The GIVEN project combines a state of the art adjoint fluid dynamics solver (TRACE) [4] for objectives such as efficiency with a newly developed mechanic solver for objectives such as reliability or material consumption, which is coupled with an ODE solver to simulate the influence of a cooling channel.
Therefore, in this work we describe how to apply the previously described meshing approach as the basis of a finite element solid mechanics solver for gradient based shape optimization that is constrained by mechanic equations such as the Poisson equation and the linear elasticity equation.

Outline

To obtain the described aim of this work, in Chapter 2 we provide the reader with the foundations of the theory of partial differential equations, followed by an introduction to finite element methods and fundamental convergence results thereof. In Chapter 3 we describe the basic concepts of shape optimization and sensitivity analysis. We introduce the two objective functionals that are supposed to be optimized and discretize these functionals and their adjoints with finite elements. In chapter 4, we give an overview of different meshing techniques and describe our structured meshing approach in detail. We address the challenges of two-dimensional and three-dimensional meshing. This is followed by Chapter 5, where we introduce the method of Krylov subspace recycling and a novel approach to utilize this method on evolving domains. In Chapter 6, we provide more details of the implementation of the finite element solver on the structured meshes and give numerical results for two and three-dimensional meshing and the solution of partial differential equation on those meshes as well as examples of Krylov subspace recycling. We conclude this work in Chapter 7 and give a brief summary as well as an outlook to future interesting fields of research.

Chapter 2

Finite element method

The Finite element method (FEM) is a method to numerically solve partial differential equations (PDEs). In this chapter, we give an introduction to the analysis of partial differential equations and the corresponding theory of solution of the equations considered in this work. Subsequently, the finite element method is introduced as the method of choice to numerically solve these problems.

2.1 Elliptic boundary value problems

In this work, we consider partial differential equations (PDEs) of second order. The general form of these equations in n dimensions is

$$-\sum_{i,k=1}^{n} a_{ik}(x)\partial_i\partial_k u + \sum_{i=1}^{n} b_i(x)\partial_i u + c(x)u = f(x). \tag{2.1}$$

With Schwarz's theorem [17], the assigned matrix

$$A(x) := (a_{ik}(x)) \tag{2.2}$$

is symmetric. Second order PDEs are categorized in three categories, elliptic, hyperbolic and parabolic PDEs.

Definition 2.1.1. A second order linear partial differential equation in n

variables is called

- elliptic in x if $A(x)$ is positive definite,
- hyperbolic in x if $A(x)$ has one negative and $n-1$ positive eigenvalues,
- parabolic in x if $A(x)$ is positive semi-definite but not definite and $(A(x), b(x))$ has rank n,
- elliptic, hyperbolic or parabolic if the respective condition holds for all points x in the domain.

As the problems this work is concerned with are elliptic, we focus on the theory of these equations, more precisely on the theory of elliptic boundary value problems. Without further conditions, PDEs in general do not possess a unique solution. For an elliptic problem, correct boundary conditions have to be provided to guarantee the existence of such a unique solution.

2.1.1 Elliptic partial differential equations with boundary conditions

For elliptic problems there are three major kinds of boundary conditions. They are introduced in this section.
Let $\Omega \in \mathbb{R}^3$ be a domain with Lipschitz boundary $\partial\Omega$, that is there exists an $N \in \mathbb{N}$ and open sets $U_1, U_2, ..., U_N$ such that

1. $\partial\Omega \subset \bigcup_{i=1}^{N} U_i$
2. For all $i = 1, ..., N$, $\partial\Omega \cap U_i$ is the graph of a Lipschitz continuous function.

This domain Ω wil be the domain on which the PDE is defined.

Dirichlet boundary conditions

With Ω defined as above, an *elliptic (or coercive) boundary value problem with Dirichlet boundary condition* is given by

$$\begin{aligned} -\sum_{i,k} \partial_i(a_{ik}(x)\partial_k u) + a_0(x)u = f \quad & \textit{in } \Omega \\ u = g \quad & \textit{on } \partial\Omega, \end{aligned} \tag{2.3}$$

where

$$a_0(x) \geq 0 \quad \text{for } x \in \Omega \tag{2.4}$$

and $A(x)$ is positive definite. f and g are arbitrary functions on Ω and $\partial\Omega$, respectively. This can be written more shortly as

$$\begin{aligned} Lu = f \quad & \textit{in } \Omega \\ u = g \quad & \textit{on } \partial\Omega, \end{aligned} \tag{2.5}$$

we call L a second order elliptic partial differential operator.
As one can transform every Dirichlet boundary value problem to a homogeneous problem, we only consider problems with $g = 0$ in the following section.

Neumann boundary conditions

Another important boundary condition is the *Neumann boundary condition.* Contrary to the Dirichlet boundary condition, it gives the values of the solution's derivative on $\partial\Omega$. So referring to the notation in (2.5), an elliptic boundary value problem with Neumann boundary conditions is given by

$$\begin{aligned} Lu = f \quad & \text{in } \Omega, \\ \sum_{i,k} n_i a_{ik} \partial_k u = h \quad & \text{on } \partial\Omega, \end{aligned} \tag{2.6}$$

where $n := n(x)$ is the outward pointing normal defined almost everywhere on $\partial\Omega$. Hence for Neumann boundary conditions, the derivative of u in direction of n is considered. Therefore we can reformulate the problem more shortly

$$\begin{aligned} Lu &= f \quad \text{in } \Omega, \\ \tfrac{\partial u}{\partial n} &= h \quad \text{on } \partial\Omega. \end{aligned} \tag{2.7}$$

Neumann boundary conditions are also called *natural boundary conditions* in contrast to the *essential boundary conditions*, the Dirichlet boundary conditions, which are essential to obtain a uniquely solvable problem.

Robin boundary conditions

The last type of considered boundary conditions are *Robin boundary conditions.* They are a weighted combination of Dirichlet and Neumann boundary conditions, hence they have the following form

$$\begin{aligned} Lu &= f \quad \text{in } \Omega, \\ \alpha\tfrac{\partial u}{\partial n} + \beta u &= h \quad \text{on } \partial\Omega. \end{aligned} \tag{2.8}$$

Of course, there also exist problems with multiple boundary conditions, so called *mixed boundary conditions.* In contrast to Robin boundary conditions, which are a combination of Dirichlet and Neumann boundary conditions on the same part of the boundary, the boundary of the concerned domain is divided into several parts with one boundary condition each. For example, we can write a problem with mixed Dirichlet-Neumann boundary conditions in the following form

$$\begin{aligned} Lu &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \partial\Omega_D, \\ \tfrac{\partial u}{\partial n} &= h \quad \text{on } \partial\Omega_N, \end{aligned} \tag{2.9}$$

with $\partial\Omega_N = \partial\Omega \backslash \partial\Omega_D$ and where $\partial\Omega_D$ is the part of the boundary where Dirichlet conditions hold and $\partial\Omega_N$ the part of the Neumann boundary conditions. f, g and h will be described more precisely in the following sections.

2.1.2 Weak solutions of elliptic PDEs

A classical solution u to a Dirichlet or Neumann boundary problem lies in $C^2(\Omega) \cap C^0(\bar{\Omega})$ or $C^2(\Omega) \cap C^1(\bar{\Omega})$, respectively. However, these solutions often do not exist. Therefore, in the following we introduce the theory of weak solutions, that guarantees solutions in a different (weaker) sense and also lead to a formulation of the problem that is well tractable numerically.

The variational formulation and existence of solution

In this section we work out the existence and uniqueness of a weak solution for the boundary value problems introduced above. This is in most parts based on [10]. For this purpose we give some essential definitions and then show the existence and uniqueness for Dirichlet-boundary problems. Following this section we show this for Neumann-boundary problems as well.

Let Ω be an open subset of $\mathbb{R}^n$ with Lipschitz boundary. Sobolev spaces are built on the function space $L_2(\Omega) := \{f : \Omega \to \mathbb{R} | \int_\Omega |f(x)|^2 dx < \infty\}$. $L_2(\Omega)$ becomes a Hilbert space with the scalar product

$$(u, v)_0 := \int_\Omega u(x)v(x)dx \tag{2.10}$$

and the corresponding norm

$$||u||_0 = \sqrt{(u, u)_0}. \tag{2.11}$$

For a systematic definition of Sobolev spaces, we introduce *weak derivatives.*

Definition 2.1.2. $u \in L_2(\Omega)$ possesses the **weak derivative** $\partial^\alpha u$ in $L_2(\Omega)$ provided that $v \in L_2(\Omega)$ and

$$(\phi, v)_0 = (-1)^{|\alpha|}(\partial^\alpha \phi, u)_0 \textit{ for all } \phi \in C_0^\infty(\Omega) \tag{2.12}$$

with $C_0^\infty(\Omega) = \{\phi \in C^\infty(\Omega) | \operatorname{supp}(\phi) \textit{ is a compact subset of } \Omega\}$.

In the case that u is differentiable, the classical derivative equals the weak derivative.

Definition 2.1.3 (Sobolev space). Given an integer $m \geq 0$, let $H^m(\Omega)$ be the set of all functions $u \in L_2(\Omega)$ which possess weak derivatives $\partial^\alpha u$ for all $|\alpha| \leq m$. Then we call $H^m(\Omega)$ a Sobolev space.
We define a scalar product on $H^m(\Omega)$ by

$$(u, v)_m := \sum_{|\alpha| \leq m} (\partial^\alpha u, \partial^\alpha v)_0 \tag{2.13}$$

with the associated norm

$$||u||_m := \sqrt{(u, u)_m} = \sqrt{\sum_{|\alpha| \leq m} ||\partial^\alpha u||_0^2}. \tag{2.14}$$

$H^m(\Omega)$ is complete with respect to the norm $|| \cdot ||_m$ [2] and is thus a Hilbert space.
The completion of $C_0^\infty(\Omega)$ regarding the Sobolev norm $|| \cdot ||_m$ is denoted by $H_0^m(\Omega)$. We need the next result to formulate the variational problem.

Theorem 2.1.4 (Characterization theorem [10]). *Let V be a linear space, and suppose that*

$$a : V \times V \to \mathbb{R} \tag{2.15}$$

is a symmetric positive bilinear form, i.e., $a(v, v) > 0$ for all $v \in V, v \neq 0$. In addition, let

$$\ell : V \to \mathbb{R} \tag{2.16}$$

be a linear functional. Then the quantity

$$J(v) := \frac{1}{2} a(v, v) - \langle \ell, v \rangle \tag{2.17}$$

attains its minimum over V at u if and only if

$$a(u,v) = \langle \ell, v\rangle \text{ for all } v \in V. \tag{2.18}$$

Moreover, (2.18) has at most one solution.

Proof. For $u, v \in V$ and $t \in \mathbb{R}$ we calculate

$$\begin{aligned} J(u+tv) &= \frac{1}{2}a(u+tv, u+tv) - \langle \ell, u+tv\rangle \\ &= \frac{1}{2}\big(a(u,u) + ta(u,v) + ta(v,u + t^2a(v,v)\big) - \langle \ell, u\rangle - t\langle \ell, v\rangle \\ &= J(u) + t(a(u,v) - \langle \ell, v\rangle) + \frac{1}{2}t^2a(v,v). \end{aligned} \tag{2.19}$$

Now let u satisfy condition (2.18), then it follows with $t = 1$:

$$\begin{aligned} J(u+v) &= J(u) + \frac{1}{2}a(v,v) \\ &> J(u), \end{aligned} \tag{2.20}$$

if $v \neq 0$ and as a is positive definite. It follows that J obtains its unique minimum at u.
On the other hand, if J has a unique minimum at u, then for every $v \in V$, the derivative of the function $t \mapsto J(u+tv)$ must vanish for $t = 0$. With the calculation above, the derivative is $a(u,v) - \langle \ell, v\rangle$, which leads to condition (2.18). □

The Characterization Theorem can now be used to prove the following proposition. The proof can be found in [10, p. 35].

Proposition 2.1.5 ([10]). *Every classical solution of the boundary value problem*

$$\begin{aligned} -\sum_{i,k} \partial_i(a_{ik}(x)\partial_k u) + a_0(x)u = f \quad &\text{in } \Omega \\ u = 0 \quad &\text{on } \partial\Omega \end{aligned} \tag{2.21}$$

is a solution of the variational problem

$$J(v) := \int_\Omega \left[\frac{1}{2}\sum_{i,k} a_{ik}\partial_i v \partial_k v + \frac{1}{2}a_0 v^2 - fv\right] dx \to \min! \tag{2.22}$$

among all functions in $C^2(\Omega) \cap C^0(\bar{\Omega})$ *with zero boundary conditions.*

By setting

$$a(u,v) := \int_\Omega \left[\sum_{i,k} a_{ik}\partial_i u \partial_k v + a_0 uv\right] dx \tag{2.23}$$

and

$$\langle \ell, v\rangle := \int_\Omega fv dx, \tag{2.24}$$

we can deduce from proposition 2.1.5 that if there exists a solution for our boundary value problem, it is the solution of equation (2.18). What remains is the question if there exists such a solution. This is not evident for classical solutions. But by solving the variational problem on a suitable Hilbert space, existence and even uniqueness of the (weak) solution can be achieved. For better comprehension, we define weak solutions and then show their existence and uniqueness.

Definition 2.1.6. A function $u \in H_0^1(\Omega)$ is called weak solution of the second order elliptic boundary value problem

$$\begin{aligned} -\sum_{i,k}\partial_i(a_{ik}\partial_k u) + a_0 u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{2.25}$$

with homogeneous Dirichlet boundary conditions, provided that

$$a(u,v) = (f,v)_0 \quad \text{for all } v \in H_0^1(\Omega), \tag{2.26}$$

where a is the associated bilinear form defined as

$$a(u,v) := \int_\Omega \left[\sum_{i,k} a_{ik} \partial_i u \partial_j v + a_0 uv \right] dx. \tag{2.27}$$

Definition 2.1.7. Let H be a Hilbert space. A bilinear form $a : H \times H \to \mathbb{R}$ is called continuous provided there exists $C > 0$ s.t.

$$|a(u,v)| \leq C||u||||v|| \textit{ for all } u, v \in H. \tag{2.28}$$

A symmetric bilinear form a is called H-elliptic or short elliptic or coercive, provided for some $\alpha > 0$,

$$a(v,v) \geq \alpha ||v||^2 \textit{ for all } v \in H. \tag{2.29}$$

This induces the following norm

$$||v||_a := \sqrt{a(v,v)}. \tag{2.30}$$

The norm (2.30) is called energy norm.
A second order linear elliptic operator is called *uniform elliptic* if there exists an $\alpha > 0$ such that

$$\xi' A(x) \xi \geq \alpha \|\xi\|^2, \quad \text{for } \xi \in \mathbb{R}^d,\ x \in \Omega. \tag{2.31}$$

Denote the space of continuous linear functionals on a normed linear space V by V'.

Theorem 2.1.8 (Lax-Milgram theorem (for convex sets)[10])**.** *Let V be a closed convex set in a Hilbert space H, and let $a : H \times H \to \mathbb{R}$ be an elliptic bilinear form. Then, for every $\ell \in H'$, the variational problem*

$$J(v) := \frac{1}{2} a(v,v) - \langle \ell, v \rangle \longrightarrow \min! \tag{2.32}$$

has a unique solution in V.

Proof. J is bounded from below because of the following:

$$\begin{aligned} J(v) &\geq \frac{1}{2}\alpha||v||^2 - ||\ell|| \cdot ||v|| \\ &= \frac{1}{2\alpha}(\alpha||v|| - ||\ell||)^2 - \frac{||\ell||^2}{2\alpha} \geq -\frac{||\ell||^2}{2\alpha}. \end{aligned} \tag{2.33}$$

Let $c_1 := \inf\{J(v) : v \in V\}$ and let (v_n) be a minimizing sequence. We now show that this is sequence is a Cauchy sequence. Therefore we consider

$$\begin{aligned} \alpha||v_n - v_m||^2 &\leq a(v_n - v_m, v_n - v_m) \\ &= 2a(v_n, v_n) + 2a(v_m, v_m) - a(v_n + v_m, v_n + v_m) \\ &= 4J(v_n) + 4J(v_m) - 8J\left(\frac{v_n + v_m}{2}\right) \\ &\leq 4J(v_n) + 4J(v_m) - 8c_1. \end{aligned} \tag{2.34}$$

The last estimate is true because V is a convex set and with this $\left(\frac{v_n+v_m}{2}\right)$ is also in V. With $J(v_n), J(v_m) \to c_1$ it follows that $||v_n - v_m|| \to 0$ for $n, m \to \infty$. It follows that (v_n) is a Cauchy sequence in H and $u = \lim_{n\to\infty} v_n$ exists .
Furthermore, as V is a closed set, it holds that $u \in V$. As J is continuous, it follows that $J(u) = \lim_{n\to\infty} J(v_n) = \inf_{v\in V} J(v)$.
It remains to show that u is the unique solution. Suppose there exist two solutions u_1 and u_2. Then the sequence $u_1, u_2, u_1, u_2, ...$ is a minimizing sequence. But as we saw above, every minimizing sequence is a Cauchy sequence so it must hold that $u_1 = u_2$.

□

With this theorem of Lax-Milgram we find the following

Theorem 2.1.9 (Existence theorem [10]). *Let L be a second order uniformly elliptic partial differential operator. Then the Dirichlet problem (2.25) always has a weak solution in $H_0^1(\Omega)$. It is the minimum of the variational problem*

$$\frac{1}{2}a(v, v) - (f, v)_0 \longrightarrow \min! \tag{2.35}$$

over $H_0^1(\Omega)$.

The result of this section is the established existence of a unique weak solution of the elliptic Dirichlet boundary value problem in Sobolev spaces. It is the solution of (2.26) where f is the right hand side of the boundary value problem and a is the bilinear form associated to the left hand side. This result can also be obtained for the problems with Neumann boundary conditions. We examine this below.

Solutions of Neumann boundary value problems

We want to guarantee the existence of a unique weak solution not only for a problem with Dirichlet boundary conditions, but also for those with Neumann boundary conditions. To get there, we first need the theorem following after introducing the *cone condition.* Consider the following notation

$$C(\zeta, \theta, l) := \{x \in \mathbb{R}^3 : |x| < l, x \cdot \zeta > |x| \cos(\theta)\} \tag{2.36}$$

for the cone with height l, direction ζ, and opening angle θ. Thereby we can define the cone property or cone condition

Definition 2.1.10. Let $\hat{\Omega}$ be a bounded open set in $\mathbb{R}^3$. For $\theta \in]0, \frac{\pi}{2}[$, $l, r > 0$, $2r \le l$ we denote by $\Pi(\theta, l, r)$ the set of all subsets Ω of $\hat{\Omega}$ satisfying the cone condition, i.e. for any $x \in \partial\Omega$ there exists a cone $C_x = C_x(\zeta_x, \theta, l)$, where ζ_x denotes a unit vector in $\mathbb{R}^3$ such that

$$y + C_x \subset \Omega, \quad y \in B(x, r) \cap \Omega, \tag{2.37}$$

where $B(x, r)$ is the open ball with radius r centered at x.

Theorem 2.1.11 (Trace theorem [10])**.** *Let Ω be bounded, and suppose Ω has a Lipschitz boundary. In addition, suppose Ω satisfies the cone condition. Then there exists a bounded linear mapping*

$$\gamma : H^1(\Omega) \to L_2(\partial\Omega), \quad ||\gamma(v)||_{0,\partial\Omega} \le c||v||_{1,\Omega}, \tag{2.38}$$

such that $\gamma v = v_{|\partial\Omega}$ for all $v \in C^1(\bar{\Omega})$.

The proof of the trace theorem can be found in [10, p. 45].

Theorem 2.1.12. *[10] Let $\partial\Omega_D \subset \partial\Omega$ be the part of the boundary where zero-boundary conditions hold. Suppose the domain Ω satisfies the assumptions of the trace theorem and suppose that $\partial\Omega_D$ has positive two-dimensional measure. Then the variational problem $J(v) := \frac{1}{2}a(v,v) - (f,v)_{0,\Omega} - (g,v)_{0,\partial\Omega} \to \min!$ has a unique solution in $H^1(\Omega)$. The solution of the variational problem lies in $C^2(\Omega) \cap C^1(\bar{\Omega})$ if and only if there exists a classical solution of the boundary value problem*

$$\begin{aligned} Lu &= f \quad \text{in } \Omega, \\ \sum_{i,k} n_i a_{ik} \partial_k u &= g \quad \text{on } \partial\Omega, \end{aligned} \tag{2.39}$$

in which case the two solutions are identical.

The proof of the theorem makes use of the well known Green's formula

Lemma 2.1.13 (Green's formula [55]). *For $u, v \in C^1(\bar{\Omega})$ the following equation holds:*

$$\int_\Omega v\partial_i u dx = -\int_\Omega u\partial_i v dx + \int_{\partial\Omega} vun_i ds. \tag{2.40}$$

The proof of this lemma can be found in several books, for example in [55].

Now we can provide the proof for theorem 2.1.12.

Proof. As a is a H^1-elliptic bilinear form, the existence of an unique solution $u \in H^1(\Omega)$ follows from the theorem of Lax-Milgram. In particular,u is characterized by

$$a(u,v) = (f,v)_{0,\Omega} + (g,v)_{0,\partial\Omega} \quad \text{for all } \in H^1(\Omega). \tag{2.41}$$

Now let (2.41) hold for $u \in C^2(\Omega) \cap C^1(\bar{\Omega})$. For $v \in H_0^1(\Omega)$ it is $\gamma v = 0$ and we get with (2.41)

$$a(u,v) = (f,v)_0, \quad \text{for all } v \in H_0^1(\Omega). \tag{2.42}$$

One can see that u is also a solution of the Dirichlet problem and suppose the boundary conditions are given. That is

$$Lu = f \quad \text{in } \Omega. \tag{2.43}$$

For $v \in H^1(\Omega)$, Green's formula yields

$$\int_\Omega v\partial_i(a_{ik}\partial_k u)dx = -\int_\Omega \partial_i v a_{ik}\partial_k u dx + \int_{\partial\Omega} v a_{ik}\partial_k u n_i ds. \tag{2.44}$$

Hence,

$$a(u,v) - (f,v)_0 - (g,v)_{0,\partial\Omega} = \int_\Omega v[Lu - f]dx + \int_{\partial\Omega}[\sum_{i,k} n_i a_{ik}\partial_k u - g]ds. \tag{2.45}$$

From (2.41) and (2.43) it results that the second integral on the right side of (2.45) vanishes. Suppose the function $v_0 = n_i a_{ik}\partial_k u - g$ does not vanish. Then it is $\int_{\partial\Omega} v_0^2 ds > 0$. As $C^1(\bar{\Omega})$ is dense in $C^0(\bar{\Omega})$ it exists a $v \in C^1(\bar{\Omega})$ such that $\int_{\partial\Omega} v_0 v ds > 0$. This is a contradiction, and the boundary condition must me satisfied.
On the other hand, form (2.45) we see immediately that every classical solution of (2.42) satisfies (2.43). □

This result and the results of the previous section will be used in the next section to establish the finite element method. Before that, we introduce the two partial differential equation that we focus on in this work and that we will revisit in later sections.

Poisson equation with Robin boundary conditions

The first PDE we want to examine is the Poisson equation with Robin boundary conditions. Robin boundary conditions are closely linked to the Poisson equation and the heat equation. In this work, we only consider Robin boundary conditions for the Poisson equation, therefore we restrict ourselves to show existence and uniqueness of the solution for this particular problem.

Theorem 2.1.14. *[2] Let Ω be a bounded domain in $\mathbb{R}^d$ and let $\partial\Omega$ ly in C^1. Furthermore let $b : \partial\Omega \to [0, \infty)$ be measurable, bounded, such that the measure of $\{z : b(z) \neq 0\}$ is greater than zero. Then, for every $f \in L_2(\Omega)$ there exists a unique solution $u \in H^2(\Omega) \cap H^1(\bar{\Omega})$*

$$-\Delta u = f \tag{2.46}$$

$$\frac{\partial u}{\partial \nu} + b\gamma u = 0, \tag{2.47}$$

with $\gamma : H^1(\bar{\Omega}) \to L_2(\partial\Omega)$ denoting the trace operator.

Note that, for $b = 0$, (2.47) is a Neumann boundary condition. We give a concept of the proof, for details see [2, Ch. 7.5]. We define the variational formulation with linear form $a : H^1(\bar{\Omega}) \times H^1(\bar{\Omega}) \to \mathbb{R}$

$$\int\limits_{\Omega} \nabla u(x) \nabla v(x) dx + \int\limits_{\partial\Omega} b(z)\gamma u(z)\gamma v(z) d\sigma = \int\limits_{\Omega} f(x)v(x)dx. \tag{2.48}$$

One can show that $a(\cdot, \cdot)$ is elliptic. With the Lax-Milgram theorem it follows that there exists a unique solution $u \in H^1(\bar{\Omega})$ such that

$$a(u, v) = \int\limits_{\Omega} fv dx, \quad \text{for all } v \in H^1(\bar{\Omega}). \tag{2.49}$$

With the theorem about local maximal regularity for solutions of the Poisson equation [2] it follows that $u \in H^2(\Omega)$. Finally, it is shown that every solution of (2.49) satisfies the conditions of (2.47) and vice versa.

Linear elasticity theory

The other elliptic PDE we consider in larger detail is the linear elasticity equation. The materials we are dealing with in this work have, to a certain point, a linear elastic behavior under load. Ceramic material, for example, has a linear elastic behavior as long as the stress remains below the ultimate tensile strength. Therefore, we closely follow [10] in this section to derive the linear elasticity equation and show existence and uniqueness of the solutions.

Recall the Frobenius inner product

$$A : B = (A, B)_F := \sum_{i,j=1} \bar{a}_{ij} b_{ij}. \tag{2.50}$$

Within the linear elasticity theory the variational problem arises to minimize the energy

$$\Pi := \int_\Omega [\frac{1}{2}\varepsilon : \sigma - f \cdot u] dx - \int_{\partial\Omega} g \cdot u \, dx, \tag{2.51}$$

where $f : \Omega \to \mathbb{R}^3$ is the body force and $g : \Omega \times S^2 \to \mathbb{R}^3$ is the surface force and S^2 is the unit sphere in $\mathbb{R}^3$. Furthermore it is $\varepsilon : \sigma := \sum_{ik} \varepsilon_{ik}\sigma_{ik}$. The variables ε, σ and u are coupled by the kinematic equations

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{2.52}$$

or short $\varepsilon = \varepsilon(u) := \nabla u$ and the linear constitutive equations

$$\varepsilon = \frac{1+\nu}{\mathtt{E}}\sigma - \frac{\nu}{\mathtt{E}}\text{tr } (\sigma) I. \tag{2.53}$$

This gives

$$\sigma = \frac{\mathtt{E}}{1+\nu}(\varepsilon + \frac{\nu}{1-2\nu}\text{tr } (\varepsilon) I). \tag{2.54}$$

Substituting with Lamé 's constants $\lambda = \frac{\mathtt{E}\nu}{(1+\nu)(1-2\nu)}$ and $\mu = \frac{\mathtt{E}}{2(1+\nu)}$ we find the common and in the course of this work often used formulation for σ

$$\sigma = \lambda \text{tr } (\varepsilon(u)) I + 2\mu \, \varepsilon(u) \tag{2.55}$$

and hence the energy density

$$\frac{1}{2}\sigma : \varepsilon = \frac{1}{2}(\lambda \text{tr } (\varepsilon) I + 2\mu \, \varepsilon) : \varepsilon = \frac{\lambda}{2}(\text{tr } (\varepsilon))^2 + \mu \, \varepsilon : \varepsilon. \tag{2.56}$$

To simplify the problem we can use the mixed method of Hellinger and Reissner. With (2.56) we can rewrite Π

$$as\Pi = \int_\Omega [\mu\,\varepsilon(v) : \varepsilon(v) + \frac{\lambda}{2}(\mathrm{div}\ v)^2 - f \cdot v]\,dx + \int_{\partial\Omega_1} g \cdot v\,dx \longrightarrow \min!, \tag{2.57}$$

where $\partial\Omega$ is divided into $\partial\Omega_0$ and $\partial\Omega_1$, and $\partial\Omega_0$ is the part where zero boundary conditions hold.

We hence get a classical differential equation

$$\begin{array}{rcll} \mathrm{div}\,\sigma &=& -f & \text{in } \Omega \\ u &=& 0 & \text{on } \partial\Omega_0, \\ \sigma \cdot n &=& g & \text{on } \partial\Omega_1, \end{array} \tag{2.58}$$

where σ is defined as in (2.54). Now we want to apply the results of section 2.1.2 to this differential equation. Therefore we need the following inequality.

Theorem 2.1.15 (Korn's inequality [10]). *Let $\Omega \subset \mathbb{R}^3$ be an open bounded set with piecewise smooth boundary. In addition, suppose $\partial\Omega_0 \subset \partial\Omega$ has positive two-dimensional measure. Then there exists a positive number $C = C(\Omega, \partial\Omega_0)$ such that*

$$\int_\Omega \varepsilon(v) : \varepsilon(v) dx \geq c||v||_1^2 \textit{ for all } v \in H^1_{\partial\Omega}(\Omega). \tag{2.59}$$

Here $H^1_{\partial\Omega}(\Omega)$ is the closure of $\{v \in C^\infty(\Omega)^3,\ v(x) = 0 \textit{ for } x \in \partial\Omega_0\}$ with respect to the $||\cdot||_1$-norm.

With this inequality we have that (2.57) is elliptic (see definition 2.1.7), hence we can apply the Lax-Milgram-Theorem. The existence theorem follows immediately from this.

Theorem 2.1.16 (Existence theorem [10]). *Let $\Omega \subset \mathbb{R}^3$ be a domain with piecewise smooth boundary and suppose $\partial\Omega_0$ has a positive two-dimensional measure. Then the variational problem of linear elasticity theory has a unique solution.*

Now we can apply the results of the sections above, to state the variational formulation for the linear elasticity PDE which is given by

$$B(u,v) = \int\limits_{\Omega} f \cdot v dx + \int\limits_{\partial\Omega_1} g \cdot v dA,\ \forall v \in H^1_{\partial\Omega}(\Omega), \tag{2.60}$$

where the bilinear form on the left side is defined in the following way

$$B(u,v) = \int\limits_{\Omega} \sigma(u) : \varepsilon(v) dx = \lambda \int\limits_{\Omega} \nabla \cdot u \nabla \cdot v + 2\mu \int\limits_{\Omega} \varepsilon(u) : \varepsilon(v) dx. \tag{2.61}$$

2.2 Introduction to finite elements

This chapter is in most parts based on [10] and is supposed to provide an introduction to the finite element method. The base of the finite elements method is to not solve the variational problem (2.32) on $H^m(\Omega)$ but on a discretized, finite-dimensional subspace, called S_h. To illustrate that the initial space is a Sobolev-space over $\mathbb{R}^d$ one can also write $H^m_h(\Omega, \mathbb{R}^d)$. The idea is to partition Ω, for example in the three-dimensional case, by, not necessarily regular, tetrahedra, cubes or rectangular parallelepipeds, etc. The resulting grid possesses a finite number of grid points. Over these grid points we can define basis functions and with the help of these basis functions we are able to define an approximation of the weak solution over $H^m_h(\Omega, \mathbb{R}^d)$. At first there are certain constraints we need to demand from the resulting grid to be able to make statements about the quality of the solutions resulting from this method.

Definition 2.2.1.
- A partition $\mathcal{T} = \{T_1, T_2, ..., T_M\}$ of Ω into tetrahedra or rectangular parallelepipeds is called admissible provided the following properties hold:

  1. $T_i \subset \Omega$ is open for $1 = 1, ..., N$;
  2. $T_i \cap T_j = \emptyset, i \neq j, i, j = 1, ..., N$;
  3. $\bigcup_{i=1}^{N} \bar{T}_i = \bar{\Omega}$.

- We will write $\mathcal{T}_h$ instead of $\mathcal{T}$ when every element has diameter at most $2h$.
- A family of partitions $\{\mathcal{T}_h\}$ is called shape regular provided that there exists a number $\kappa > 0$ such that every $T \in \mathcal{T}_h$ contains a circle of radius ρ_T with

  $$\rho_T \geq h_T/\kappa,$$

  where h_T is half the diameter of T.
- A family of partitions $\{T_h\}$ is called uniform provided that there exists a number $\kappa > 0$ such that every $T \in \mathcal{T}_h$ contains a circle with radius $\rho_T \geq h/\kappa$.

Now we can give a formal definition of a finite element:

Definition 2.2.2. A finite element is a triple (T, Π, Σ) with the following properties:

1. T is a polyhedron in $\mathbb{R}^d$.
2. Π is a subspace of $C(T)$ with finite dimension s.
3. Σ is a set of s linearly independent functionals over Π. These functionals are also called *interpolation conditions.* Every $p \in \Pi$ can be uniquely identified by the s functionals in Σ.

Functions $\theta \in \Pi(T)$ are called *local shape functions* if they form a basis of $\Pi(T)$.

Note that with the local shape functions we can rewrite every $u \in \Pi(T)$ as $u = \sum\limits_{i=1}^{s} u_i\theta_i$ with $u_i \in \mathbb{R}$.
In this work, we focus on rather simple types of P_t-finite elements that are defined below. For these elements, additionally s interpolation points (or nodes) $X_1, ..., X_s \in T$ are needed, including at least the vertices of T. They give a so called *nodal basis* of $\Pi(T)$.

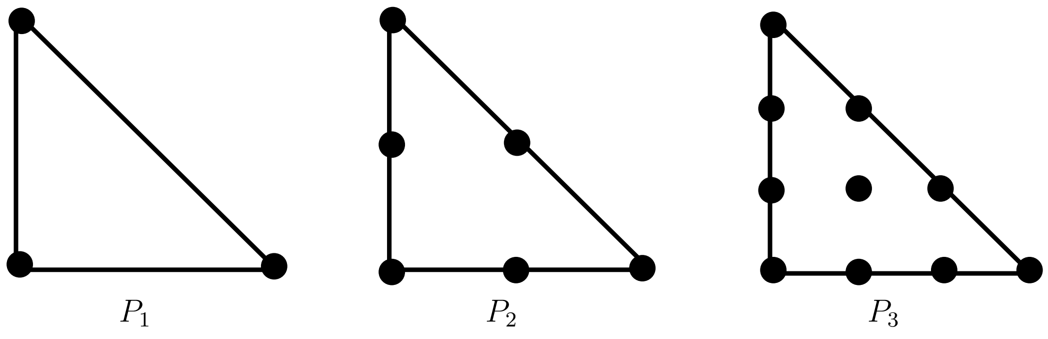


Figure 2.1: Linear, quadratic and cubic P_t elements.

The most common finite elements are the Lagrange elements. These elements are determined by their following interpolation conditions φ at the s interpolation points X_j

$$\varphi_j(\theta_i) = \theta_i(X_j) = \delta_{ij}, \quad \text{for } i, j \in \{1, ..., s\}.$$

The most basic of these elements are polynomial triangle elements in the two dimensional space and tetrahedral elements in the three dimensional space, respectively, denoted by P_t. Let $\mathcal{P}_t$ denote the set of polynomials of degree t and let $\mathcal{T}$ be a triangulation of Ω. Then P_t is defined by

$$P_t := \{v \in L_2(\Omega) : v|_T \in \mathcal{P}_t \quad \forall T \in \mathcal{T}\} \cap H^1(\Omega). \tag{2.62}$$

P_1, P_2 and P_3 are visualized in Figure 2.1.

In the next subsection, a common method to solve finite element problems is introduced briefly.

Galerkin method

The standard approach to transform the analytical variational problem into an algebraic form is the Galerkin method.

Consider the variational problem

$$J(v) := \frac{1}{2}a(v,v) - \langle \ell, v\rangle \longrightarrow \min_{S_h}!$$

in the subspace S_h. From the results of 2.1.2 we know that u_h is a solution of the problem provided

$$a(u_h, v) = \langle \ell, v\rangle, \ \textit{for all } v \in S_h. \tag{2.63}$$

With the above base, (2.63) is equivalent to

$$a(u_h, \theta_i) = \langle \ell, \theta_i\rangle, \ i = 1, 2, ..., N. \tag{2.64}$$

For $u_h \in S_h$, we assume that

$$u_h = \sum_{k}^{N} z_k \theta_k, \tag{2.65}$$

and this leads to the system of equations

$$\sum_{k=1}^{N} a(\theta_k, \theta_i) z_k = \langle \ell, \theta_i\rangle, \ i = 1, 2, ..., N \tag{2.66}$$

which we can write in matrix-vector form as

$$\mathbf{Az} = \mathbf{b}$$

and $\mathbf{A}$ is positive definite, if a is a H^m-elliptic bilinear form. Hence, this approach still leads to a unique solution.
Through this approach the problem is reduced to a system of linear equations. We consider the linear equations and suitable methods to solve them later on.

2.2.1 Error bounds

The Galerkin method establishes a finite element approach to numerically solve elliptic PDE's. However, it does not provide us with any knowledge about the accuracy of the discrete solution. Therefore, this section gives some of the fundamental statements concerning the quality of approximation of and convergence towards the analytical solution. For the conforming finite elements shown in the previous section, all convergence results base on the following Lemma.

Lemma 2.2.3 (Céa's Lemma[10]). *Let the bilinear form a be H^m-elliptic. Additionally let u and u_h be the solution of the variational problem in H^m and in $S_h \subset H^m$, respectively. Then it is*

$$||u - u_h||_m \leq \frac{C}{\alpha} \inf_{v_h \in S_h} ||u - v_h||_m. \tag{2.67}$$

Proof. According to the definition of u and u_h it holds that

$$\begin{aligned} a(u, v) &= \langle \ell, v \rangle \quad \text{for } v \in H^m \\ a(u_h, v) &= \langle \ell, v \rangle \quad \text{for } v \in S_h. \end{aligned} \tag{2.68}$$

As $S_h \subset H^m$ it follows with subtraction that

$$a(u - u_h, v) = 0 \quad \text{for } v \in S_h. \tag{2.69}$$

Let $v_h \in S_h$. With $v = v_h - u_h \in S_h$ it follows immediately from the equation above that $a(u - u_h, v_h - u_h) = 0$ and

$$\begin{aligned} \alpha||u - u_h||_m^2 &\leq a(u - u_h, u - u_h) \\ &= a(u - u_h, u - v_h) + a(u - u_h, v_h - u_h) \\ &\leq C||u - u_h||_m||u - v_h||_m. \end{aligned} \tag{2.70}$$

By dividing by $||u - u_h||_m$ we find the Lemma's assertion. □

Céa's Lemma states that the accuracy of the solution u_h depends primarily on a good choice of the functional spaces, so u is well approximated.

This can be controlled by the fineness of the partition and by the order of the polynomials in the shape functions.
Most finite element methods are based on C^0 elements, as higher order elements are difficult and expensive to construct. Those elements lie only in $H^1(\Omega)$, thus higher order norms are not defined on this space. Therefore, in the following we define mesh depended norms.

Definition 2.2.4. For a discretization $\mathcal{T}_h$ on Ω and $m \geq 1$ define

$$\|v\|_{m,h} = \sqrt{\sum_{T \in \mathcal{T}} \|v\|_{m,T}^2}. \tag{2.71}$$

Obviously, $\|v\|_{m,h} = |v\|_{m,\Omega}$ for $v \in H^m(\Omega)$.

A crucial advantage in speed of the finite element method is the concept of affine families and the reference element.

Definition 2.2.5. A family of finite element spaces S_h for partitions $\mathcal{T}_h$ of $\Omega \subset \mathbb{R}^d$ is called an affine family provided there exists a finite element $(\hat{T}, \hat{\Pi}, \hat{\Sigma})$ called the *reference element* with the following properties:

- For every $T \in \mathcal{T}_h$, there exists an affine mapping $B_T : \hat{T} \longrightarrow T$ such that
  - $\hat{\Pi} = \Pi \circ B_T$,
  - $\hat{\theta}_j = \theta_j \circ B_T$ and
  - $\hat{\varphi}_j(p \circ B_T) = \varphi_j(p)$.

Through the reference element it is possible to calculate every required solution only once on the reference element and then transform the solution to solution on each element T. The following transformation formula assures that we do not loose accuracy by using the reference element.

Lemma 2.2.6 (Transformation formula [10])**.** *Let Ω and $\hat{\Omega}$ be affine equivalent, that is there exists a bijective affine mapping*

$$\begin{aligned} &F : \hat{\Omega} \to \Omega, \\ &F\hat{x} = x_0 + B\hat{x} \end{aligned} \tag{2.72}$$

with a non-singular matrix B. Then, for $v \in H^m(\Omega)$ it is $\hat{v} \circ F \in H^m(\hat{\Omega})$ and there exists a constant $c = c(\hat{\Omega}, m)$ such that

$$|\hat{v}|_{m,\hat{\Omega}} \leq \|B\|^m \cdot |\det B|^{-1/2} |v|_{m,\Omega}. \tag{2.73}$$

With this and the generalization of definition 2.2.5, the following approximation theorem can be stated for the C^0 P_t-elements defined in (2.62).

Theorem 2.2.7 (Approximation theorem [10]). *Let $t \geq 2$ and let $\mathcal{T}_h$ a quasi-uniform triangulation of Ω. Then, for the interpolation with piecewise polynomials of degree $d-1$ the following inequality holds with a constant $c = c(\Omega, \kappa, t)$*

$$\|u - I_h u\|_{m,h} \leq c \cdot h^{t-m} |u|_{t,\Omega} \quad \textit{for } u \in H^t(\Omega), \quad 0 \leq m \leq t. \tag{2.74}$$

As mentioned earlier, if a method converges at all depends on the quality of the mesh. The presented results state the assumptions the discretization method has to fulfill in order to deliver a solution that converges towards the actual solution. These requirements are fulfilled if certain angle conditions hold. These are conditions that have to be met by the angles of the finite element cells. In [11] a comprehensive overview of these conditions can be found. In practice, the condition that is mainly used is the maximum angle condition. If the condition for two-dimensional triangulations holds, that

$$\exists \gamma_0 < \pi \quad \forall \mathcal{T}_h \in \{\mathcal{T}_h\} \quad \forall T \in \mathcal{T}_h : \gamma_T \leq \gamma_0, \tag{2.75}$$

where γ_T is the maximal angle of a triangle T, then u_h converges to u as $h \longrightarrow 0$.

Non-conforming finite elements

In section 2.2 several assumption were made that are often violated in numerical computations. In the following, we obtain generalizations of Céa's

Lemma. In a first step we replace the original problem

$$a(u,v) = \langle \ell, v\rangle, \quad v \in V \tag{2.76}$$

by a sequence of finite dimensional problems

$$a_h(u_h,v) = \langle \ell_h, v\rangle, \quad v \in V_h, \tag{2.77}$$

and $u_h \in S_h$. With this generalization, it is not necessary anymore that we can evaluate the bilinear form analytically, but approximate the integrals with quadrature, for example. Strang's first Lemma is a generalization of Céa's Lemma for this case.

Lemma 2.2.8 (Strang's first lemma [10]). *Assume that $(a_h(\cdot,\cdot))_h$ is a uniformly S_h-elliptic family of bilinear forms, that is there exists a positive constant α independent of h such that*

$$a_h(v,v) \geq \alpha ||v||^2_{m,\Omega}, \quad v \in S_h \tag{2.78}$$

Then there exists a positive constant c independent of h such that

$$\begin{aligned} ||u-u_h|| \leq c(\inf_{v_h\in S_h}\{||u-v_h||_m + \sup_{w_h\in S_h} \frac{a(v_h,w_h)-a_h(v_h,w_h)}{||w_h||}\} \\ + \sup_{w_h\in S_h}\{\frac{\langle \ell, w_h\rangle - \langle \ell_h, w_h\rangle}{||w_h||}\}). \end{aligned} \tag{2.79}$$

The next generalization we need is the case that S_h is not a subspace of V. This occurs for example, if it is not possible to exactly resolve the boundary by the chosen finite elements.

Additionally to the assumption that $a_h(\cdot,\cdot)$ is S_h-elliptic, we also need to assume that it is uniformly bounded in the following sense

$$|a_h(u,v)| \leq C\|u\|_h \cdot \|v\|_h, \quad u \in V + S_h,\, v \in S_h. \tag{2.80}$$

Lemma 2.2.9 (Strang's second lemma [10]). *Assume that $(a_h(\cdot,\cdot))_h$ is a uniformly S_h-elliptic and uniformly bounded family of bilinear forms, then* *there exists a positive constant c independent of h such that*

$$||u - u_h||_h \leq c(\inf_{v_h \in S_h} ||u - v_h||_h + \sup_{w_h \in S_h} \frac{|a_h(u, w_h) - \langle \ell_h, w_h \rangle|}{||w_h||_h}). \tag{2.81}$$

The first part of the upper bound is called **approximation error** and the second part is called **consistency error**. In other words, the first part of the upper bound is caused by the fact that we do not solve the problem in the original space V but in an approximate finite dimensional space S_h, the second part is caused by the approximation of the original problem by quadrature, for example.
The results of this section lead to the following theorem.

Theorem 2.2.10 ([10]). *Let Ω be a domain with C^2-boundary. For a finite element approximation with linear triangle elements and a quasi-uniform triangulation, it holds that*

$$\begin{aligned} ||u - u_h||_{1,\Omega} &\leq c\,h||u||_{2,\Omega} \\ &\leq c\,h||f||_{0,\Omega}. \end{aligned} \tag{2.82}$$

The estimate still holds if a is replaced by

$$a_h(u, v) := \int_{\Omega_h} \sum_{k,\ell} a_{k\ell} \partial_k u \partial_\ell v \, dx. \tag{2.83}$$

2.2.2 Discretization of the linear elasticity equation via finite elements

As an example, in the following we discretize the Linear Elasticity equation via Lagrange finite elements in a formal way. This section has been published in a similar form in [8]. Recall the linear elasticity equation

$$L(u, v) = \int\limits_{\Omega} f \cdot v \, dx + \int\limits_{\partial\Omega_N} g \cdot v \, dA, \, \forall v \in H^1_{D,h}(\Omega, \mathbb{R}^3),$$

where $H^1_{D,h}(\Omega,\mathbb{R}^3)$ is the subspace of $H^1_h(\Omega,\mathbb{R}^3)$ with functions $u \in H^1_h(\Omega,\mathbb{R}^3)$ vanishing on the boundary $\partial\bar{\Omega}_D$.
The integrals in the equation are approximated via Gauss quadrature. In a first step, Ω is partitioned by a finite mesh $\mathcal{T}$ represented by the N grid points $X = \{X_1, ..., X_N\}$. This mesh gives as well N_{el} (Lagrange) finite elements $\{T, \Pi(T), \Sigma(T)\}$ with n_{sh} local shape functions $\theta_{T,m} \in \Pi(T)$ which are defined by the nodes $X^T_1, ..., X^T_{n_{sh}} \in T$.
As in Definition 2.2.5 we assume that there exists a reference element $\{\hat{T}, \hat{\Pi}, \hat{\Sigma}\}$ and a bijective transformation for each element $T \in \mathcal{T}_h$ $B_T : \hat{T} \to T$ such that $\hat{\Pi} = \Pi \circ B_T$, $\hat{\theta} j = \theta \circ B_T$, $j \in \{1, ..., n_{sh}\}$ and

$$B_T = B_T(\hat{\xi}, X) = \sum_{j=1}^{n_{sh}} \hat{\theta}_j(\hat{\xi}) X^T_j,\ \hat{\xi} \in \hat{T}. \tag{2.84}$$

To numerically calculate the integral, for each $T \in \mathcal{T}$ we chose q^T_l quadrature points $\hat{\xi}^T_l$ on the reference element $\hat{T}$ with weights $\hat{\omega}^T_l$. Reminding the form of the bilinear form in section 2.1.2, we then have

$$\begin{aligned}
L(u,v) &= \lambda \sum_{T\in\mathcal{T}_h} \int_T \nabla\cdot u\, \nabla\cdot v dx + 2\mu \sum_{T\in\mathcal{T}_h} \int_T \varepsilon(u) : \varepsilon(v)\, dx \\
&= \lambda \sum_{T\in\mathcal{T}_h} \int_T \nabla\cdot u(B_T(\hat{\xi}))\nabla\cdot v(B_T(\hat{\xi}))\text{det }(\hat{\nabla} B_T(\hat{\xi}))\, d\hat{\xi} \\
&+ 2\mu \sum_{T\in\mathcal{T}_h} \int_T \varepsilon(u(B_T(\hat{\xi}))) : \varepsilon(v(B_T(\hat{\xi})))\text{det }(\hat{\nabla} B_T(\hat{\xi}))\, d\hat{\xi} \\
&\approx \lambda \sum_{T\in\mathcal{T}_h} \sum_{l=1}^{q^K_l} \hat{\omega}^T_l \text{det }(\hat{\nabla} B_T(\hat{\xi}_l))\, \nabla\cdot u(B_T(\hat{\xi}_l))\, \nabla\cdot v(B_T(\hat{\xi}_l)) \\
&+ 2\mu \sum_{T\in\mathcal{T}_h} \sum_{l=1}^{q^K_l} \hat{\omega}^T_l \text{det }(\hat{\nabla} B_T(\hat{\xi}_l))\varepsilon(u(B_T(\hat{\xi}_l))) : \varepsilon(v(B_T(\hat{\xi}_l))).
\end{aligned} \tag{2.85}$$

For each element $T \in \mathcal{T}$ and $\xi \in T$ we can write $u(\xi)$ in terms of the local shape functions on the reference element: $u(\xi) = \sum_{m=1}^{n_{sh}} u_m \hat{\theta}_m \circ B_T^{-1}(\xi)$ and hence

$$\nabla u(\xi) = \sum_{m=1}^{n_{sh}} u_m \otimes (\hat{\nabla} B_T(\hat{\xi})^T)^{-1} \hat{\nabla}\theta_m(\hat{\xi}). \tag{2.86}$$

With this, we also instantly get

$$\nabla \cdot u(x) = \sum_{m=1}^{n_{sh}} \operatorname{tr} \left(u_m \otimes (\hat{\nabla} B_T(\hat{\xi})^T)^{-1} \hat{\nabla}\theta_m(\hat{\xi}) \right). \tag{2.87}$$

Similar to the discretization of the bilinear form, the volume force can be discretized in the following way

$$\int_\Omega f \cdot v \, dx = \sum_{T\in\mathcal{T}_h} \sum_{l_1}^{q_l^T} \hat{\omega}_l^T \det \left(\hat{\nabla} B_T(\hat{\xi}_l) \right) f(B_T(\hat{\xi}_l)) \cdot v(B_T(\hat{\xi}_l)).$$

The discretization of the surface force is a bit different from the discretization above, as it is not a volume but a surface integral. Here, not all elements T are considered but there faces F that lie in $\partial\Omega$. For each of these faces $F \in \mathcal{N}_h$, where $\mathcal{N}_h$ is the collection of these specific faces, the respective element is identified by $T = T(F) \in \mathcal{T}_h$. One can assume that there also exists a reference face $\hat{F}$ on $\hat{T}$ such that $B_{T(F)} : \hat{F} \to F$. Additionally, for the quadrature, surface quadrature points $\hat{\xi}_l^F$ and weights $\hat{\omega}_l^F$ have to be chosen, as the face possesses one dimension less than the elements. And finally, for the transformation the square root of the Gram determinant $\sqrt{\det g_F(\hat{\xi}_l^F)}$ is required instead of the determinant of the derivative of B_T. It is

$$g_F(\hat{\xi}) = \hat{\nabla}^F (B_T\big|_{\hat{F}})(\hat{\xi}) \left(\hat{\nabla}^F (B_T\big|_{\hat{F}}) \right)^T (\hat{\xi}),$$

and thus

$$\int_{\partial\Omega} g \cdot v \, A = \sum_{F\in\mathcal{N}_h} \sum_{l=1}^{q_l^F} \hat{\omega}_l^F \sqrt{\det g_F(\hat{\xi}_l^F)} \, g(B_{T(F)}(\hat{\xi}_l^F)) \cdot v(B_{T(F)}(\hat{\xi}_l^F)).$$

The discretized equation can be rewritten in a shorter form, in terms of the global degrees of freedom $U = (u_j)_{j\in\{1,\ldots,N\}}$, $u_j \in \mathbb{R}^3$ and the node coordinates X, where it is understood that $u_j = 0$ if $X_j \in \partial\Omega_D$. Then we have

$$\begin{aligned} L(X)U &= F(X), \\ L(X)_{(j,r),(k,s)} &= L(e_r\theta_j, e_s\theta_k), \\ F_{(j,r)} &= \int_\Omega f \cdot e_r\theta_j dx + \int_{\partial\Omega_N} g \cdot e_r\theta_j dA; \end{aligned} \tag{2.88}$$

with $e_r, r = 1, 2, 3$ the standard basis on $\mathbb{R}^3$.

Chapter 3

Objective functionals

In this chapter, we introduce the objectives and general setting that set the stage for gradient based shape optimization. The work leading to this thesis is concerned with optimizing the reliability of mechanic components. Therefore, we start this chapter by briefly introducing the concept of shape optimization [27], followed by the description of the two objectives that we are considering in this work.
Parts of Section 3.2.1, Section 3.3 and Section 3.4 have been published in a similar form in [8].

3.1 Shape optimization

For a given bounded domain $\Omega \subset \mathbb{R}^d$ with Lipschitz boundary $\partial\Omega$, in shape optimization the aim is to optimize the domain without any changes in its inherent topology in order to optimize a given objective. During the optimization process, the domain will evolve from one configuration Ω_t to the next. This can be described in a formal way by introducing an artificial time dimension $t \in \mathbb{R}_+$.

Definition 3.1.1. Let Ω be defined as above and let $\{F_t\}$, $t \in [0, t_0]$ be a family of mappings with

$$\begin{aligned} &F_t : \Omega \longrightarrow \mathbb{R}^d, \\ &F_t = \mathrm{id} + t\mathcal{V}, \quad t > 0, \end{aligned} \tag{3.1}$$

where $\mathcal{V} \in (H^{1,\infty}(\Omega))^2$ is a velocity field. Then, a new configuration of Ω at time t is given by

$$\Omega_t = F_t(\Omega) \tag{3.2}$$

The objectives that are being optimized are given as integral functions over Ω or $\partial\Omega$. Consider such a functional

$$J_t = \int_{\Omega_t} f(t, x_t) dx. \tag{3.3}$$

Then, with (3.1) we can define the derivative of this functional as

$$\dot{J} := \dot{J}(\Omega; \mathcal{V}) = \frac{d}{dt} J_t|_{t=0+}, \tag{3.4}$$

which is equal to

$$\dot{J} = \int_\Omega \dot{f} dx + \int_\Omega f \mathrm{div}\mathcal{V} dx, \tag{3.5}$$

with the pointwise material derivative

$$\dot{f} = \frac{\partial f}{\partial t} + \nabla_x u \cdot \mathcal{V} \quad \text{in } \Omega, \tag{3.6}$$

as proven for example in [27]. In gradient based shape optimization, which is rather called sensitivity analysis, these material derivatives are used as search directions in the process to optimize the given domain. In the following sections, we introduce two objective functionals that are differentiable in the sense that we just introduced. These functionals measure the probability of failure of components made of ceramic material under tensile load and components that fail due to low cycle fatigue (LCF). As the idea of both functionals is similar, we introduce only one in more detail.

3.2 Survival probabilities for mechanical components

When designing a mechanical component, many different objectives have to be considered. Next to efficiency or material consumption, reliability is a very important factor to be considered for obvious reasons. When considering the reliability or robustness of a mechanical component a common criterion is the van Mises yield criterion [56]. This criterion unfortunately has the disadvantage of not being differentiable. To overcome this issue, a probabilistic model was proposed in [48] for LCF and in [9] for ceramic material. The functionals of the probabilistic model have been proven to be differentiable. Both models are categorized as PDE-constrained shape optimization problems. They are introduced in the next sections.

3.2.1 Ceramic material

In this section, a functional is introduced giving the probability of failure for a component made from ceramic material under tensile load. Tensile load is a force applied to a body pulling in a certain direction. In the following, first the governing partial differential equation is described and subsequently we deduce the objective functional which will be derived following [9].

The state equation

Ceramic is a linear elastic material whose displacement under load is described by the linear elasticity equation. We consider a compact body $\Omega \subset \mathbb{R}^3$ which is filled with the ceramic material. Furthermore, we assume that the boundary $\partial\Omega$ of Ω can be divided in three different parts

$$\partial\Omega = \overline{\partial\Omega}_D \cup \overline{\partial\Omega}_{N_{fixed}} \cup \overline{\partial\Omega}_{N_{free}}. \tag{3.7}$$

Here, $\partial\Omega_D$ is the part of the boundary where Dirichlet-boundary conditions hold. $\partial\Omega_{N_{fixed}}$ is the part where surface forces may act. These two parts of the boundary are fixed. We restrict these parts of the Neumann-boundary from

being altered in the optimization process, since we want to find an optimal shape under a given load. If we allow for this part of the boundary to be optimized as well, it would tend to simply minimize the area where the surface forces are applied. And finally $\partial\Omega_{N_{free}}$ is the free part of the boundary which can be modified in order to optimally comply with the design objective 'reliability', with implicit zero-Neumann-boundary conditions. Forces that act on Ω are represented by functions $f \in L^2(\Omega, \mathbb{R}^3)$ and $g \in L^2(\partial\Omega_{N_{fixed}}, \mathbb{R}^3)$. The volume force f that acts on the whole domain can be gravity, for example, the surface force g is the tensile load.
Furthermore we assume that there is a bounded set $\hat{\Omega} \subseteq \mathbb{R}^3$, such that $\Omega \subseteq \hat{\Omega}$ for all admissible choices of the free portion of the boundary. The behavior of ceramic material under load can be approximated as linear elastic. Hence, its displacement is described by the linear elasticity equation, which is already introduced in more detail in Section 2.1.2. With this, the governing PDE for the present problem is given by

$$\begin{aligned} -\mathrm{div}(\sigma(u(\Omega))) &= f \quad \text{on } \Omega \\ u(\Omega) &= 0 \quad \text{on } \partial\Omega_D \\ \sigma(u(\Omega))\hat{n} &= g \quad \text{on } \partial\Omega_{N_{fixed}} \\ \sigma(u(\Omega))\hat{n} &= 0 \quad \text{on } \partial\Omega_{N_{free}}. \end{aligned} \tag{3.8}$$

Derivation of the objective functional

Our aim is to optimize the reliability of a component made of ceramic material. The main source of failure for those components is exposure to tensile loading. The idea is to derive a functional that measures the probability of failure of a ceramic component under such tensile loading. In order to obtain this functional, it is necessary to comprehend the mechanical behavior of ceramic material under load. Assume the domain Ω is filled with the ceramic material and it is exposed to a tensile load σ_n. The component described by Ω fails if a fracture occurs under the tensile load. Therefore the question is what are the main drivers for those fracture and how to measure the probability that these drivers occur.
Ceramic material is produced in a process called Sintering. It is a process

that transforms material powder through a thermal treatment into a solid structure [19]. From this process, small flaws arise in the material, which, in the first place, have no influence on the quality of the material. Under load however, these flaws may grow cracks and thus lead to failure. Hence, the probability of failure heavily depends on the probability that these cracks occur. In a first step, we model these flaws and cracks. They can be described by the following properties:

1. Their location $x \in \Omega$
2. Their orientation μ
3. Their radius $a \in \mathbb{R}_+$
4. Their shape.

Following the common literature, we assume that the cracks are "penny shaped". As there is no indication that the location, orientation, or the radius is anyhow determined by the Sintering process or any other influences, we assume that these features occur arbitrarily. With this assumption we can find the configuration of a crack

$$(x, a, \mu) \in (\bar{\Omega} \times (0, \infty) \times S^2) := \mathcal{C}, \tag{3.9}$$

with S^2 the unit sphere in $\mathbb{R}^3$ and x and μ are uniformly distributed on $\bar{\Omega}$ and S^2, respectively. The distribution of a will be discussed later on. We call $\mathcal{C}$ the crack configuration space.
To go on, it is necessary to consider classical engineering analysis of spontaneous failure of mechanical components from brittle material under given mechanical loads. The following section refers in wide parts to [23].
In linear fracture mechanics, the three dimensional stress field close to a crack in a two-dimensional plane close to the tip of the crack is of the form

$$\sigma = \frac{1}{\sqrt{2\pi r}}\{K_I \tilde{\sigma}^I(\phi) + K_{II}\tilde{\sigma}^{II}(\phi) + K_{III}\tilde{\sigma}^{III}(\phi)\} + \text{regular terms}, \tag{3.10}$$

where r is the distance to the crack front and ϕ the angle of the shortest connection point considered to the crack front with the crack plane.
For our purposes the stress intensity factor K_I is key, as it describes the influence of mode I load to the stress field.

With the tensile load σ_n it is

$$K_I := \frac{2}{\pi}\sigma_n\sqrt{\pi a}. \tag{3.11}$$

A crack becomes critical, i.e. a fracture occurs, if K_I exceeds a critical value K_{I_c}. Obviously we can neglect compressive load, that is negative values for σ_n. With this and following [9], we set

$$\sigma_n := (n \cdot \sigma(\nabla u)n)^+ = \max\{n \cdot \sigma(\nabla u)n, 0\}. \tag{3.12}$$

As K_I determined we define the following:

Definition 3.2.1. Let $\mathcal{C}$ be the crack configuration space, then we call

$$A_c := A_c(\Omega, \nabla u) = \{(x, a, n) \in \mathcal{C} : K_I(a, \sigma_n(x)) > K_{I_c}\}$$

the set of critical crack configurations.

Construction of the probability measure

In Definition 3.2.1 we introduce the set A_c. A_c contains all crack configurations that would lead to failure of Ω given the displacement u, which itself depends on the load σ_n. As there is no other indication, we assume that the number of cracks in a component determines the probability of failure of said component. Hence, we want to minimize the probability of A_c not being empty. To model this, the following section requires a foundation in probability theory. We provide the most important definitions here and refer the reader for further comprehension to [32].

Definition 3.2.2 (Topological space)**.** Let $\Omega \neq \emptyset$ be an arbitrary set. A collection of sets $\tau \subset 2^{\Omega}$ is called topology, if the following conditions hold.

1. $\emptyset, \Omega \in \tau$.

2. If $A, B \in \tau$, then $A \cap B \in \tau$.

3. For an arbitrary family $\mathcal{F} \in \tau$, it follows that $\bigcup\limits_{A \in \mathcal{F}} A \in \tau$.

A pair (Ω, τ) is called topological space, sets $A \in \tau$ are called open, sets $A \in \Omega$ with $A^c \in \tau$ are called closed.

Definition 3.2.3 (Borel algebra)**.** Let $\Omega \neq \emptyset$ be a set. A collection of subsets $\mathcal{A} \subseteq 2^\Omega$ is called σ-Algebra, if the following conditions hold.

1. $\Omega \in \mathcal{A}$.

2. If $A \in \mathcal{A} \Rightarrow A^c \in \mathcal{A}$.

3. For $A_1, A_2, \dots \in \mathcal{A}$ it follows that $\bigcup\limits_{n=1}^{\infty} A_n \in \mathcal{A}$.

Now let (Ω, τ) be a topological space. Then the σ-Algebra $\sigma(\tau)$ generated by τ,

$$\sigma(\tau) := \bigcap_{\substack{\mathcal{A} \subset 2^\Omega, \\ \mathcal{A} \supset \tau}} \mathcal{A}, \tag{3.13}$$

is called Borel σ-Algebra and is denoted by $\mathcal{B}(\Omega) := \mathcal{B}(\Omega, \tau) := \sigma(\tau)$. Elements $A \in \mathcal{B}(\Omega)$ are called Borel sets.

Definition 3.2.4 (Measure)**.** Let $\mathcal{A}$ be a σ-Algebra. A function $\mu : \mathcal{A} \to [0, \infty]$ is called a measure if

- $\mu(\emptyset) = 0$.
- For $A_1, A_2, \dots \in \mathcal{A}$ and $A \subset \bigcup\limits_{n=1}^{\infty} A_n$ it follows that $\mu(A) \leq \sum\limits_{n=1}^{\infty} \mu(A_n)$.

A normed measure is called probability measure.

Definition 3.2.5 (Hausdorff space)**.** A topological space X is called a Hausdorff space, if $\forall\, x, y \in X\; x \neq y$: $\exists$ neighborhoods U_x, V_y with $U_x \cap V_y = \emptyset$.

With this, we can give a first Lemma concerning A_c.

Lemma 3.2.6 ([9]). *A_c is in the Borel σ-Algebra on $\mathcal{C}$, i.e. is measurable on $\mathcal{C}$.*

Let us come back to the aim of this section. We want to minimize the probability that the set of critical configurations A_c is not empty. Now that we established that A_c is measurable, we can try to construct a measure that counts the configurations in A_c, that is the critical cracks depending on the tensile load σ_n.
Let $A \subseteq \bar{\Omega}$, $\Lambda \subseteq S^2$, and let $(A \times (a,b] \times \Lambda) \subseteq \mathcal{C}$. We want to specify the measure

$$N((A \times (a,b] \times \Lambda)) = \text{ number of cracks.}$$

The probability of failure would than equal the probability of that measure N of A_c to be larger than zero. To find the distribution of N we have to state the properties that N should have, given the natural properties of the cracks.

1. Cracks are uniformly distributed over Ω with an average number z of cracks per unit volume.

2. Two cracks can always be distinguished.

3. Orientations are uniformly distributed over S^2 and are independent of the crack's location.

4. The distribution of the crack's radius is independent from location and orientation.

5. The number of flaws/cracks of non intersecting volumes $A_1, ..., A_n \subseteq \Omega$ is statistically independent of each other.

Given these properties it follows with [63] and [30][Corollary 7.4.] that N is a Poisson point process (PPP).
To understand this assertion we introduce point processes and especially the Poisson point process. For this purpose, we introduce the Radon measure.

Definition 3.2.7 (Radon measure)**.** Let ν be a measure on the σ-algebra of Borel sets of a Hausdorff topological space X. If

1. ν is an inner regular, that is for any Borel set B,
   $\nu(B) = \sup\{\nu(K)|K \subseteq B \text{ is compact}\}$ and
2. ν is locally finite, that is for every $x \in X$ there is an open neighborhood U s.t. $\nu(U) < \infty$,

then ν is called a Radon measure.

$R(\mathcal{C})$ are the Radon measures on $\mathcal{C}$. Then $R_C(\mathcal{C})$ are the counting measures on $R(\mathcal{C})$ that is $\gamma \in R_C(\mathcal{C})$ if $\gamma(B) \in \mathbb{N}_0$ for all measurable $B \subset \mathcal{C}$. The definition of a point process is to be found for example in [30].

Now we can identify the specifications we stated for N with the properties in the definition above. We want to have uniformly distributed cracks. From this it follows that N is nonatomic. From the second specification it follows that N is simple. And from the fifth specification it is obvious that N has independent increments. With these three properties, as mentioned above, Corollary 7.4. from [30] indicates that N is a Poisson point process.

Definition 3.2.8 ((heterogeneous) Poisson Point Process)**.** Given a Radon measure ν. A Poisson Point Process N is a Point Process such that

1. For every measurable B it holds that
$$P(N(B) = n) = e^{-\nu(B)}\frac{\nu(B)^n}{n!} \sim Po(\nu(B)),$$
2. $N(B_1), ..., N(B_k)$ are independent for pairwise disjoint $B_1, ..., B_k$.

$\nu(B)$ is the intensity measure of the PPP.

The component Ω fails if a fracture occurs, that is $N(A_c) \geq 1$. With this, we can give the survival probability of the component Ω, given the displacement field $u \in H^1(\Omega, \mathbb{R}^3)$ as

$$p_s(\Omega|\nabla u) = P(N(A_c(\Omega\nabla u)) = 0) = \exp\{-\nu(A_c(\Omega, \nabla u))\}, \tag{3.14}$$

which is obviously Weibull distributed. We aim to maximize the survival probability of our component. With the above survival probability the problem to minimize the intensity measure arises! Therefore we need to find a more explicit representation of the intensity measure $\nu(A_c(\Omega, \nabla u))$. The domain of ν is the crack configuration space $\mathcal{C} := (\bar{\Omega} \times (0, \infty) \times S^2)$. As the location is uniformly distributed on Ω and the orientation is isotropic, it follows that

$$\nu = dx \otimes \nu_a \otimes \frac{dn}{4\pi}, \tag{3.15}$$

with dx the Lebesgue measure on $\mathbb{R}^3$, dn the surface measure on S^2 and a certain positive Radon measure ν_a on $(\mathbb{R}_+, \mathcal{B}(\mathbb{R}_+))$. As the surface area of S^2 is equal to 4π, we norm the measure by this value.
As previously mentioned, there is a critical value K_{I_c} that must not be exceeded to avoid failure. Therefore, A_c only contains configurations with a radius a such that $K_I(a) > K_{I_c}$. This is true for all $a_c > \frac{\pi}{4}\left(\frac{K_{I_c}}{\sigma_n}\right)^2$. Due to these considerations, the intensity measure is given by

$$\nu(A_c(\Omega, \nabla u)) = \frac{1}{4\pi} \int\limits_{\Omega} \int\limits_{S^2} \int\limits_{a_c}^{\infty} d\nu_a(a) dn dx. \tag{3.16}$$

Following the common literature, we assume that $d\nu(a) = c \cdot a^{-\tilde{m}} da$, with a certain constant $c > 0$ and $\tilde{m} > 1$. Then we can easily calculate the inner integral

$$\int\limits_{a_c}^{\infty} d\nu_a(a) = \tilde{c} \left(\frac{\pi}{4} \left(\frac{K_{I_c}}{\sigma_n} \right)^2 \right)^{-\tilde{m}+1}$$

With this and setting $m := 2(\tilde{m} - 1)$, with the assumption that $\tilde{m} \geq \frac{3}{2}$ holds and in assembling all constant values in the positive constant $\left(\frac{1}{\sigma_0^m}\right)$ we find our objective functional to be

$$J(\Omega, \nabla u) := \nu(A_c(\Omega, \nabla u)) = \frac{1}{4\pi} \int\limits_{\Omega} \int\limits_{S^2} \left(\frac{\sigma_n}{\sigma_0} \right)^m dndx. \tag{3.17}$$

The optimization problem

In the sections above we have derived the state equation and the objective functional. The convexity of the objective functional is proved in [9]. Assembling the state equation and the objective functional yield the PDE-constrained optimization problem:

Definition 3.2.9. The problem of optimal reliability for a ceramic component $\Omega \in \mathbb{R}$ under a given volume load $f \in L^2(\hat{\Omega}), \mathbb{R}^3$ and surface load $g \in L^2(\partial\Omega_{N_{fixed}}, \mathbb{R}^3)$ is defined as the shape optimization problem:
Find $\Omega^* \subset \hat{\Omega}$ such that $J(\Omega^*, u(\Omega^*)) \leq J(\Omega, u(\Omega))$ $\forall \Omega \subset \hat{\Omega}$ and such that the minimization problem is solved

$$\begin{array}{lrcl} & \min J(\Omega, u(\Omega)) & & \\ \text{s.t.} & & & \\ & -\text{div}(\sigma(u(\Omega))) & = & f \quad \text{on } \Omega \\ & u(\Omega) & = & 0 \quad \text{on } \partial\Omega_D \\ & \sigma(u(\Omega))\hat{n} & = & g \quad \text{on } \partial\Omega_{N_{fixed}} \\ & \sigma(u(\Omega))\hat{n} & = & 0 \quad \text{on } \partial\Omega_{N_{free}} \end{array} \tag{3.18}$$

3.2.2 Low cycle fatigue

In material science, fatigue describes the failure of material when exposed to cyclic and varying stress. Low cycle fatigue is one of several categories of fatigue. It describes failure that occurs in cases of cyclic stress. The cyclic stress exceeds the yield stress and thus leads to a lower number of cycles until failure occurs, compared to high cycle fatigue where a large number of cycles of smaller stress amplitude eventually leads to failure. In the context of the GIVEN project, we consider gas turbine blades, made of polycristalline metal. The failure mechanism here is fundamentally different from the one

introduced in the section before, as the initial crack occurs at the boundary of the component and propagates from there, into the component until it becomes to large leading to a fracture and the failure of the component.
The functional measuring the probability of failure for these components made from polycristalline metal with a Low Cycle Fatigue (LCF) failure mechanism is constructed via the same principles as the functional for ceramic material in the section above. It is a Weibull type functional depending on the solution of a governing PDE modeling the displacement of the component under load. The displacement of these materials also shows a linear elastic behavior, hence the governing PDE for this problem is the linear elasticity equation.
According to [48], the objective functional for this problem is given by

$$J(\Omega, u) := \int_{\partial\Omega} \frac{1}{N_{\mathrm{det}}(\sigma(x))^{\bar{m}}} dA. \tag{3.19}$$

Here, N_{det} is the solution of the Coffin-Manson-Basquin equation (CMB)

$$\varepsilon(x) = \frac{\sigma_f^{'}}{E}(2N_{\mathrm{det}}(x))^b + \varepsilon_f^{'}(2N_{\mathrm{det}}(x))^c, \tag{3.20}$$

with $x \in \partial\Omega$, $\varepsilon(x)$ the strain field and $\sigma_f^{'}$, $\varepsilon_f^{'}$, b and c are parameters proprietary to the considered material. N_{det} can be interpreted as the estimate of the deterministic number of cycles until a fracture occurs. Obviously, an inverse operation is needed to obtain N_{det}. This leads to a computation of the LCF-functional that is not as straightforward as the computation of the ceramic functional and thus is further addressed in Section refsec:impl-lcf.

3.3 Discretization of the objective functionals

To compute the objective functionals introduced in Section 3.2 we discretize them via the finite element method, same as we have already discretized the linear elasticity equation in Section 2.2.2. We do this exemplarily for the two dimensional functional to measure the probability of failure for a ceramic component in the following section. Subsequently, we consider how to obtain the discrete derivatives of this functional via a discrete adjoint approach.

3.3.1 Discretization of the two-dimensional functional for ceramic material

Recall the inner integral of the objective functional (3.17) in two dimensions

$$I(f) = \int\limits_{S^1} \left((n \cdot \sigma(Du)n)^+\right)^m dn.$$

Since solving this integral analytically yields the Appell hypergeometric function it seems to be more appropriate to interpolate this integral. Therefore we transform the integrand via polar coordinates to

$$I(f) = \int_0^{2\pi} \left(\left(\cos^2(\varphi)\sigma_{11} + 2\cos(\varphi)\sin(\varphi)\sigma_{12} + \sin^2(\varphi)\sigma_{22}\right)^+\right)^m \varphi. \quad (3.21)$$

As it is a periodic function over its whole period, the trapezoidal rule is the method of choice as it shows exponential convergence in this case [58]. Recall the trapezoidal rule with n interpolation points, the integrally limits a, b and $h = \frac{b-a}{n}$

$$T^{(n)}(f) = h\left(\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{i=1}^{n-1} f(a+ih)\right). \quad (3.22)$$

The function is integrated over its period, in this case $f(a) = f(b)$. Thus for the present integral it yields

$$\begin{aligned} T^{(n)}(f) = \frac{2\pi}{n}\Bigg((\sigma_{11}^+)^m + \sum_{i=1}^{n-1}\Bigg(\Bigg(\cos^2\left(\frac{i2\pi}{n}\right)\sigma_{11} \\ + \ 2\cos\left(\frac{i2\pi}{n}\right)\sin\left(\frac{i2\pi}{n}\right)\sigma_{12} + \sin^2\left(\frac{i2\pi}{n}\right)\sigma_{22}\Bigg)^+\Bigg)^m\Bigg). \end{aligned} \quad (3.23)$$

By setting $T^{(n)}(f) = p$, we can know discretize the objective functional as in Section 2.2.2. It is

$$
\begin{aligned}
J(\Omega, u) &= \sum_{T \in \mathcal{T}_h} \int_T p\left(\sigma\left(x\right), \varphi\right) dx \\
&= \sum_{T \in \mathcal{T}_h} \int_{\hat{K}} p\left(\sigma\left(B_T\left(\hat{x}\right)\right), \varphi\right) \det\ \left(\hat{\nabla} B_T\left(\hat{x}\right)\right) d\hat{x} \\
&\approx \sum_{T \in \mathcal{T}_h} \sum_{l=1}^{q_l^{\hat{T}}} \hat{\omega}_l^T p\left(\sigma\left(B_T\left(\hat{\xi}_l^T\right)\right), \varphi\right) \det\ \left(\hat{\nabla} B_T\left(\hat{\xi}_l^T\right)\right) \\
&\approx \sum_{T \in \mathcal{T}_h} \sum_{l=1}^{q_l^{\hat{T}}} \hat{\omega}_l^T T^{(n)}\left(f\right)\left(\sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)\right) \det\ \left(\hat{\nabla} B_T\left(\hat{\xi}_l^T\right)\right) \\
&= \sum_{T \in \mathcal{T}_h} \sum_{l=1}^{q_l^{\hat{T}}} \hat{\omega}_l^T \frac{2\pi}{n} \left(\left(\sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{11}^{+}\right)^m\right. \\
&+ \sum_{i=1}^{n-1} \left(\left(\cos^2\left(\frac{i2\pi}{n}\right) \sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{11}\right.\right. \\
&+ 2\cos\left(\frac{i2\pi}{n}\right) \sin\left(\frac{i2\pi}{n}\right) \sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{12} \\
&+ \left.\left.\left.\sin^2\left(\frac{i2\pi}{n}\right) \sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{22}\right)^{+}\right)^m\right) \cdot \det\ \left(\hat{\nabla} B_T\left(\hat{\xi}_l^T\right)\right).
\end{aligned}
\tag{3.24}
$$

3.3.2 Adjoint equation

In the prior section we derived the PDE - constrained optimization problem (3.18) and discretized it via the finite element method. For the gradient based optimization we need to compute the material derivative of $J(X,U)$ with respect to X. This is

$$
\frac{dJ(X,U(X))}{dX} = \frac{\partial J(X,U(X))}{\partial X} + \frac{\partial J(X,U(X))}{\partial U} \frac{\partial U(X)}{\partial X}, \tag{3.25}
$$

which obviously has to be equal to 0 to be minimal.
The computation of $\frac{\partial U(X)}{\partial X}$ is very expensive. Therefore, we avoid computing this term by applying the discrete adjoint approach.

Consider the derivative of the state equation

$$\begin{aligned} \frac{\partial L(X)}{\partial X} U(X) + L(X) \frac{\partial U(X)}{\partial X} &= \frac{\partial F(X)}{\partial X} \\ \Leftrightarrow \quad \frac{\partial U(X)}{\partial X} &= L(X)^{-1} \left[\frac{\partial F(X)}{\partial X} - \frac{\partial L(X)}{\partial X} U(X) \right]. \end{aligned} \tag{3.26}$$

Now we can substitute $\frac{\partial U(X)}{\partial X}$ in 3.25 by 3.26 and obtain

$$\begin{aligned} \frac{dJ(X, U(X))}{dX} &= \frac{\partial J(X, U(X))}{\partial X} \\ &\quad + \frac{\partial J(X, U(X))}{\partial U} L(X)^{-1} \left[\frac{\partial F(X)}{\partial X} - \frac{\partial L(X)}{\partial X} U(X) \right]. \end{aligned} \tag{3.27}$$

Now we define

$$\Lambda := \frac{\partial J(X, U(X))}{\partial U} L(X)^{-1} \tag{3.28}$$

which leads (as L is symmetric) to the **adjoint equation**:

$$L^T(X)\Lambda = \frac{\partial J(X, U(X))}{\partial U} \tag{3.29}$$

This gives us the adjoint state method: The solution $\bar{U}$ of the following system of equations

$$\begin{aligned} L^T(X)\Lambda &= \frac{\partial J(X, U(X))}{\partial U} \\ L(X)U(X) &= F(X) \\ \frac{\partial J(X, U(X))}{\partial X} + \Lambda \left[\frac{\partial F(X)}{\partial X} - \frac{\partial L(X)}{\partial X} U(X) \right] &= 0 \end{aligned} \tag{3.30}$$

is the solution of the problem (3.18). It is straightforward that this method also applies to the functional describing the failure of low cycle fatigue.

3.4 Derivative of the objective functional

In this rather technical section we describe the discretization of the adjoint state method (3.30) for the two-dimensional objective functional for ceramic material by first discretizing the derivatives of J with respect to U and with respect to X. The other derivatives appearing in that system, as well as the derivatives of the LCF-functional are already discretized in [21]. Different to that paper we do not calculate the global derivatives but the local ones that is with respect to the local degrees of freedom on each element. Also as we have already done in the previous sections, we discretize the derivatives element-wise. Hence, we calculate for every T the derivative of J^{loc} with respect to U^{loc} and X^{loc} which are eventually assembled to the global derivatives. We consider each element T separately and calculate the derivative locally. For each $T \in \mathcal{T}_h$ set $\omega_l = \hat{\omega}_l^T \cdot \det \hat{\nabla} B_T(\hat{\xi}_l^T)$ for $l = 1, ..., q_l^{\hat{T}}$. Then we have for each T

$$\begin{aligned} J^{loc}(X, U(X)) = \sum_{l=1}^{q_l^{\hat{T}}} \omega_l \frac{2\pi}{n} \Bigg(\left(\sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{11}^{+} \right)^m & \\ + \sum_{i=1}^{n-1} \Bigg(\Bigg(\cos^2\left(\frac{i2\pi}{n}\right) \sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{11} & \\ + 2\cos\left(\frac{i2\pi}{n}\right)\sin\left(\frac{i2\pi}{n}\right)\sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{12} & \\ + \sin^2\left(\frac{i2\pi}{n}\right)\sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{22}\Bigg)^{+}\Bigg)^m\Bigg). & \end{aligned} \tag{3.31}$$

3.4.1 Derivative with respect to U^{loc}

For the derivative we need to take into account the positive part. For better legibility, set $\sigma_{11} := \sigma\left(B_T\left(\hat{\xi}_l\right)\right)_{11}$ and

$$\begin{aligned} T_i^{(n)} := &\cos^2\left(\frac{i2\pi}{n}\right)\sigma\left(B_T\left(\hat{\xi}_l\right)\right)_{11} \\ &+ 2\cos\left(\frac{i2\pi}{n}\right)\sin\left(\frac{i2\pi}{n}\right)\sigma\left(B_T\left(\hat{\xi}_l\right)\right)_{12} \\ &+ \sin^2\left(\frac{i2\pi}{n}\right)\sigma\left(B_T\left(\hat{\xi}_l\right)\right)_{22}. \end{aligned} \tag{3.32}$$

With this, we can right down for each $T \in \mathcal{T}_h$ the local derivative with respect to local U, with $j = 1, ..., n_{sh}$ being the index of the respective shape function and for $k = 1, 2$ being the index of the dimension.

$$\begin{aligned} \frac{\partial J^{loc}(X, U(X))}{\partial U^{loc}_{j,k}} =& \frac{\partial}{U^{loc}_{j,k}} \sum_{l=1}^{q_l^{\hat{T}}} \omega_l \frac{2\pi}{n}\left(\left(\sigma_{11}^{+}\right)^m + \sum_{i=1}^{n-1}\left(\left(T_i^{(n)}(\hat{\xi}_l)\right)^+\right)^m\right) \\ =& \sum_{l=1}^{q_l^{\hat{T}}} \omega_l \frac{2\pi}{n}\left(\frac{\partial}{U^{loc}_{j,k}}\left(\sigma_{11}^{+}\right)^m\right. \\ &\left.+ \sum_{i=1}^{n-1}\frac{\partial}{U^{loc}_{j,k}}\left(\left(T_i^{(n)}(\hat{\xi}_l)\right)^+\right)^m\right). \end{aligned} \tag{3.33}$$

We start with the derivatives of σ. Remember

$$\sigma_{il} = \mu(\nabla_{il}u + \nabla_{li}u) + \lambda\delta_{il}\sum_{k=1}^{2}\nabla_{kk}u. \tag{3.34}$$

Hence, the derivative of $\sigma(u)$ is

$$\frac{\partial \sigma_{i\ell}}{\partial U_{rs}} = \mu\left(\frac{\partial \nabla_{i\ell}u}{\partial U_{rs}} + \frac{\partial \nabla_{\ell i}u}{\partial U_{rs}}\right) + \lambda\delta_{i\ell}\sum_{k=1}^{2}\frac{\partial \nabla_{kk}u}{\partial U_{rs}}. \tag{3.35}$$

Therefore, we need the derivatives of ∇u with respect to U. On the local element, they are given by

$$\begin{aligned}\frac{\partial \hat{\nabla} u(\hat{\xi})_{i\ell}}{\partial U_{rs}} &= \sum_{j=1}^{n_{sh}} \sum_{k=1}^{2} \frac{\partial U_{jk}}{\partial U_{rs}^{loc}} \left(\left(\hat{\nabla} B_T(\hat{\xi}) \right)^{-1} \right)_{k\ell} \hat{\nabla}_k \hat{\theta}_j(\hat{\xi}) \\ &= \sum_{j=1}^{n_{sh}} \sum_{k=1}^{2} \delta_{jr}\delta_{is} \left(\left(\hat{\nabla} B_T(\hat{\xi}) \right)^{-1} \right)_{k\ell} \hat{\nabla}_k \hat{\theta}_j(\hat{\xi}) \\ &= \delta_{is} \sum_{k=1}^{2} \left(\left(\hat{\nabla} B_T(\hat{\xi}) \right)^{-1} \right)_{k\ell} \hat{\nabla}_k \hat{\theta}_j(\hat{\xi}).\end{aligned} \tag{3.36}$$

Next, we calculate the derivative of $\left(\sigma_{11}^{+}\right)^{m}$. With simple chain rule, the derivative yields

$$\frac{\partial}{U_{j,k}^{loc}} \left(\sigma_{11}^{+}\right)^{m} = \mathbb{1}_{\{\sigma_{11}>0\}}(\hat{\xi}_l) \frac{\partial \sigma \left(B_T \left(\hat{\xi}_l^T \right) \right)_{11}}{U_{j,k}^{loc}} m \left(\sigma \left(B_T \left(\hat{\xi}_l^T \right) \right)_{11} \right)^{m-1}. \tag{3.37}$$

In the same way, the remaining derivative is calculated

$$\frac{\partial}{U_{j,k}^{loc}} \left(\left(T_i^{(n)}(\hat{\xi}_l) \right)^{+} \right)^{m} = \mathbb{1}_{\{T_i^{(n)}>0\}}(\hat{\xi}) \frac{\partial T_i^{(n)}(\hat{\xi}_l)}{U_{j,k}^{loc}} m \left(T_i^{(n)}(\hat{\xi}_l) \right)^{m-1}, \tag{3.38}$$

where, again, the derivative of $T_i^{(n)}$ is easily calculated as

$$\begin{aligned}\frac{\partial T_i^{(n)}(\hat{\xi}_l)}{U_{j,k}^{loc}} &= \cos^2\left(\frac{i2\pi}{n}\right) \frac{\partial \sigma \left(B_T \left(\hat{\xi}_l^T \right) \right)_{11}}{\partial U_{jk}^{loc}} \\ &+ 2\cos\left(\frac{i2\pi}{n}\right) \sin\left(\frac{i2\pi}{n}\right) \frac{\partial \sigma \left(B_T \left(\hat{\xi}_l^T \right) \right)_{12}}{U_{jk}^{loc}} \\ &+ \sin^2\left(\frac{i2\pi}{n}\right) \frac{\partial \sigma \left(B_T \left(\hat{\xi}_l^T \right) \right)_{22}}{U_{jk}^{loc}},\end{aligned} \tag{3.39}$$

which completes the derivative with respect to local U.

3.4.2 Derivative with respect to local X^{loc}

The derivative with respect to local X has the form

$$\begin{aligned}\frac{\partial J^{loc}(X,U(X))}{\partial X^T_{jk}} &= \frac{\partial}{\partial X^T_{jk}} \sum_{l=1}^{q_l^{\hat{T}}} \omega_l \frac{2\pi}{n} \left(\left(\sigma^+_{11}\right)^m + \sum_{i=1}^{n-1} \left(\left(T_i^{(n)}(\hat{\xi}_l) \right)^+ \right)^m \right) \\ &= \sum_{l=1}^{q_l^{\hat{T}}} \left[\frac{\partial \omega_l}{\partial X^T_{jk}} \cdot \frac{2\pi}{n} \left(\left(\sigma^+_{11}\right)^m + \sum_{i=1}^{n-1} \left(\left(T_i^{(n)}(\hat{\xi}_l) \right)^+ \right)^m \right) \right. \\ &\quad \left. + \, \omega_l \frac{2\pi}{n} \left(\frac{\partial}{\partial X^T_{jk}} \left(\sigma^+_{11}\right)^m + \sum_{i=1}^{n-1} \frac{\partial}{\partial X^T_{jk}} \left(\left(T_i^{(n)}(\hat{\xi}_l) \right)^+ \right)^m \right) \right].\end{aligned} \tag{3.40}$$

First, we want to calculate

$$\frac{\partial \omega_l}{\partial X^{loc}_{jk}} = \frac{\partial}{\partial X^{loc}_{jk}} \left(\hat{\omega}_l \det(\hat{\nabla} B_T(\hat{\xi}_l)) \right) = \hat{\omega}_l \frac{\partial}{\partial X^{loc}_{jk}} \left(\det(\hat{\nabla} B_T(\hat{\xi}_l)) \right).$$

With the formula for the derivative of determinants

$$\frac{\partial \det(A)}{\partial x} = \det(A) \mathrm{tr} \left(A^{-1} \frac{\partial A}{\partial x} \right) \tag{3.41}$$

we have

$$\frac{\partial}{\partial X^{loc}_{jk}} \left(\det\left(\hat{\nabla} B_T(\hat{\xi}_l) \right) \right) = \det\left(\hat{\nabla} B_T(\hat{\xi}_l) \right) \mathrm{tr} \left(\left(\hat{\nabla} B_T(\hat{\xi}_l) \right)^{-1} \frac{\partial \hat{\nabla} B_T(\hat{\xi}_l)}{\partial X^{loc}_{jk}} \right).$$

Thus, we need to closer excermine $\hat{\nabla} B_T(\hat{\xi})$. It is given by

$$\hat{\nabla} B_T(\hat{\xi}_l) = \begin{pmatrix} \frac{\partial X_1^{loc}}{\partial \hat{X}_1} & \frac{\partial X_1^{loc}}{\partial \hat{X}_2} \\ \frac{\partial X_2^{loc}}{\partial \hat{X}_1} & \frac{\partial X_2^{loc}}{\partial \hat{X}_2} \end{pmatrix} = \sum_{r=1}^{n_{sh}} \begin{pmatrix} X^{loc}_{r1} \frac{\partial \hat{\theta}_r(\hat{\xi}_l)}{\partial \hat{X}_1} & X^{loc}_{r1} \frac{\partial \hat{\theta}_r(\hat{\xi}_l)}{\partial \hat{X}_2} \\ X^{loc}_{r2} \frac{\partial \hat{\theta}_r(\hat{\xi}_l)}{\partial \hat{X}_1} & X^{loc}_{r2} \frac{\partial \hat{\theta}_r(\hat{\xi}_l)}{\partial \hat{X}_2} \end{pmatrix}. \tag{3.42}$$

With this, its derivative is

$$\frac{\partial \hat{\nabla} B_T(\hat{\xi}_l)_{i\ell}}{\partial X^{loc}_{jk}} = \sum_{r=1}^{n_{sh}} \frac{\partial X^{loc}_{ri}}{\partial X^{loc}_{jk}} \frac{\partial \hat{\theta}_r(\hat{\xi}_l)}{\partial \hat{X}_\ell} = \sum_{r=1}^{n_{sh}} \delta_{rj} \delta_{ik} \frac{\partial \hat{\theta}_r(\hat{\xi}_l)}{\partial \hat{X}_\ell} = \delta_{ik} \frac{\partial \hat{\theta}_j(\hat{\xi}_l)}{\partial \hat{X}_\ell}, \tag{3.43}$$

and thus the first part of the derivative is completed. Now we calculate the derivatives of σ with respect to X_{jk}^{loc}

$$\frac{\partial \sigma_{i\ell}}{\partial X_{rs}} = \mu \left(\frac{\partial \nabla_{i\ell} u}{\partial X_{rs}} + \frac{\partial \nabla_{\ell i} u}{\partial X_{rs}} \right) + \lambda \delta_{i\ell} \sum_{k=1}^{2} \frac{\partial \nabla_{kk} u}{\partial X_{rs}}, \tag{3.44}$$

with

$$\begin{aligned} \frac{\partial \nabla u}{\partial X_{jk}^{loc}} &= \frac{\partial}{\partial X_{jk}^{loc}} \left(\sum_{r=1}^{n_{sh}} u_r \otimes \left(\left(\hat{\nabla} B_T^T(\hat{\xi}) \right)^{-1} \hat{\nabla} \hat{\theta}_r(\hat{\xi}) \right) \right) \\ &= \sum_{r=1}^{n_{sh}} u_r \otimes \left(\frac{\partial}{\partial X_{jk}^{loc}} \left(\hat{\nabla} B_T^T(\hat{\xi}) \right)^{-1} \right) \hat{\nabla} \hat{\theta}_r(\hat{\xi}). \end{aligned}$$

For the derivative of the inverse we can apply the formula $\frac{\partial A^{-1}}{\partial x} = -A^{-1} \frac{\partial A}{\partial x} A^{-1}$. With the consideration above we can calculate the remaining derivatives similar to the derivatives with respect to u. The derivative of $\left(\sigma_{11}^{+}\right)^m$ similar to the derivative with respect to U is given by

$$\begin{aligned} \frac{\partial}{X_{j,k}^{loc}} \left(\sigma \left(B_T \left(\hat{\xi}_l^T \right) \right)_{11}^{+} \right)^m =& \mathbb{1}_{\{\sigma_{11} > 0\}} (\hat{\xi}_l^T) \frac{\partial \sigma \left(B_T \left(\hat{\xi}_l^T \right) \right)_{11}}{X_{j,k}^{loc}} \\ & \cdot m \left(\sigma \left(B_T \left(\hat{\xi}_l^T \right) \right)_{11} \right)^{m-1}, \end{aligned} \tag{3.45}$$

and the derivative of $\left(\left(T_i^{(n)}(\hat{\xi}_l) \right)^{+} \right)^m$

$$\frac{\partial}{X_{j,k}^{loc}} \left(\left(T_i^{(n)}(\hat{\xi}_l) \right)^{+} \right)^m = \mathbb{1}_{\{T_i^{(n)} > 0\}} (\hat{\xi}) \frac{\partial T_i^{(n)}(\hat{\xi}_l)}{X_{j,k}^{loc}} m \left(T_i^{(n)}(\hat{\xi}_l) \right)^{m-1}, \tag{3.46}$$

with

$$\begin{aligned}
\frac{\partial T_i^{(n)}(\hat{\xi}_l)}{X_{j,k}^{loc}} &= \cos^2\left(\frac{i2\pi}{n}\right)\frac{\partial\sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{11}}{\partial X_{jk}^{loc}} \\
&+ 2\cos\left(\frac{i2\pi}{n}\right)\sin\left(\frac{i2\pi}{n}\right)\frac{\partial\sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{12}}{X_{jk}^{loc}} \\
&+ \sin^2\left(\frac{i2\pi}{n}\right)\frac{\partial\sigma\left(B_T\left(\hat{\xi}_l^T\right)\right)_{22}}{X_{jk}^{loc}}.
\end{aligned} \tag{3.47}$$

Chapter 4

Structured meshing for evolving geometries

As we learned in Chapter 2, to numerically solve partial differential equations on a domain Ω, a discretization of the given domain is necessary. Although there exist so called meshfree methods [13], most methods to solve PDEs base on discretization through meshes, which is what we focus on in this work. There are many different approaches and methods to discretize a given domain, and there is not the one best method. Depending on the problem, one method may suit the cause better than the other. This may depend on different factors. In some applications, it is important to resolve the boundary with high accuracy, in other applications it is necessary to achieve real time results in the calculations. If the domain changes over the cause of the calculations, one might choose another approach as if different calculations are performed on the ever same domain. In this chapter, we look at some examples of different meshing techniques and highlight their advantages and disadvantages. Subsequently, we consider more closely approaches used in shape optimization. Finally, we introduce a structured meshing approach to combine the best qualities of the different techniques.

4.1 Introduction to mesh generation

The simulation of physical behavior on domains is a largely spread problem in mathematics and computer science. The characteristics of the problem are modeled mathematically, often resulting in a partial differential equation or a system thereof. The best option would be to solve this problem analytically. In practice, however, this is possible in only very rare cases. For the very most partial differential equations, we only know that there exists a unique solution, and in some cases, like the Navier-Stokes equation, we do not know even that for certain.

Therefore, these problems are usually solved numerically. In Chapter 2, we introduced the widely used approach of the Finite Element method. In this approach, as in many related approaches, the problem is solved by discretizing the considered domain $\Omega \subset \mathbb{R}^d$, $d \in \{2, 3\}$ by a partition or triangulation as described in definition 2.2.1 $\mathcal{T}(\Omega)$, that we will also refer to as mesh.

To find a suitable mesh generation approach, we have to answer several questions. The most fundamental one is the distinction between the following two cases. Either, we want to simulate physical behavior that might change over time, in other words is time dependent, but the domain stays unchanged; or the domain evolves over time, as it is the case in shape optimization. In the former, we might accept more computational cost to gain higher accuracy, as the mesh has only to be generated once. In the latter, as mesh generation is one of the key bottlenecks, the cost of the mesh generation is an important factor in the decision for or a against a certain technique. Since the objective of this work is shape optimization, in this work we focus on the latter.

In the setting of shape optimization, the next decision is to be made between mesh morphing and re-meshing. These two approaches are described in the next section.

4.1.1 Mesh morphing and re-meshing

In the following we assume an initial domain $\Omega_0 \subset \mathbb{R}^d$ and that we have determined a deformation of this domain in form of a velocity field as described in section 3.1, stemming from an optimization process. Recall the perturbation of identity $F_t : \Omega \to \mathbb{R}^d$,

$$F_t = \mathrm{id} + t\mathcal{V}, \quad t > 0,$$

where $\mathcal{V} \in (H^{1,\infty}(\Omega))^d$ is the velocity field. This leads to a domain $\Omega_1 := F_1(\Omega_0)$ and then Ω_2 and so on. To be able to obtain any convergence result, we have to chose one approach to discretize every Ω_t that arises during the optimization process. We need to find a discretization $\mathcal{T}_0$ and a discretization $\mathcal{T}_1$. There are two different general ideas on the discretizations of these new domains. One is called re-meshing the other is referred to as mesh morphing or mesh warping. Re-meshing means that in every optimizations step, a (completely) new mesh is generated, with no knowledge of the previous domain and triangulation used in the meshing process. This has the great advantage that the mesh can be manufactured for the specific domain, in general leading to stable meshes and thus stable and precise numerical calculations. But meshing is time-consuming and as a usual shape optimization process consists of many hundreds or even thousands of optimization steps, re-meshing in every step will most likely lead to a slow process. This is why recent research focuses on mesh morphing techniques instead. In the following, we look at some basic concepts of mesh depended mesh morphing techniques.

Assume Ω_0 is discretized using a triangulation $\mathcal{T}_0$. Furthermore, assume from the optimization process, an initial descent direction is given, for example the material derivative u of the objective functional. The idea is to find a 'good' displacement field to shift the triangulation $\mathcal{T}_0 := \mathcal{T}(\Omega_0)$ by moving its nodes, giving the new triangulation $\mathcal{T}_1$ and hence implicitly the new domain Ω_1. In this way, only the locations of the nodes have to be updated, connectivity and all other features of the mesh are kept. Depending on how the displacement field is generated, this can reduce the computational cost enormously. But what does 'good' mean. First of all, as we are optimizing, the new domain Ω_1 should sufficiently decrease the value of the objective functional. And equally important, the resulting triangulation must remain stable, meaning allowing for numerically stable calculations. [54] provides a comparison of some mesh morphing methods that approach the issue from the angle of mesh generation. It recommends the FEMWARP method [51], which takes the solutions of Poisson equations that are informed by a displacement field on the boundary to move the vertices of the mesh.

In recent years, in the specific setting of gradient based shape optimization, research focuses on solving the problem on metric spaces with well chosen scalar products. If chosen wisely, these scalar products act as implicit smoothing, such as the scalar product that induces the Steklov-Poincaré metric [50]. In this approach, instead of calculating the gradient in the standard Euclidean scalar product, it is calculated in an alternative one. In practice, it is computed by using the discretized shape derivative as a source term for a linear elasticity equation. The resulting displacement of this elasticity equation is the shape gradient in the Steklov-Poincaré sense. We will consider this approach more closely in Section 6 as a smoothing operator in our numerical examples.

Although these approaches show increasingly good results, they have two major disadvantages. One is, that even when applying the most recent methods, in many cases re-meshing becomes unavoidable after some steps of optimization. The other is, that all of these methods are based on unstructured meshes. Especially in a high performance computing (HPC) setting, structured grids are preferable for numerous reasons, such as lower memory consumption of structured grids and to benefit from caching [15]. Therefore, in the next section, we introduce a structured meshing approach, that comes with lower computational cost than common re-meshing strategies and still leads to stable meshes.

4.2 Structured meshing

Common structured grid methods are generated by a so-called mapping approach. These methods, like Arbitrary Lagrangian Eulerian (ALE) [29], use a, generally uniform, reference grid, similar to the concept of a reference element in the finite element method. The computations are performed on this computational domain and then mapped to the actual domain. These

methods are highly robust and efficient, but it is clear that they are only applicable to very simple domains. An interesting structured meshing approach that allows for more complicated domains is the method of Cut Cells [5]. The basis is a Cartesian mesh, which is not adapted to the geometry of the concerned domain. Instead, it determines the cells of the mesh that intersect with the boundary of the domain. The part of each of these cells lying outside the geometry are cut and the computations are only performed on the remaining part of the cell. These meshes are generated fast, and are also suited to discretize domains of highly complex geometry [5]. The problem here is that the cut cells can be arbitrarily small and distorted, which makes convergence theory and measuring the robustness very difficult. Related approaches are immersed boundary methods [37, 42]. These methods do not resolve the boundary at all and are thus not suited for (gradient based) shape optimization problems, since most objective functionals highly depend on the shape of the surface of the domain.
The structured meshing approach that we introduce in this section is inspired by the idea of Composite Finite Elements introduced by Hackbusch and Sauter in [24, 25, 26]. We believe that this method is a great way to discretize shape optimization problems, since it results in highly structured meshes that are still capable of resolving domains with rather complicated boundaries. We will also see that the method allows to carry over knowledge from one mesh to the next, reducing the computational cost in the mesh generation process. Here, the method is described in two dimensions. It can easily be extended to three dimensions, which we will discuss in Section 4.2.5.

4.2.1 Two-dimensional meshing

Recall the definition of the shape optimization problem (3.18) from section 3.1,

$$\begin{aligned} \min_{\Omega} \qquad & J(\Omega, u) \\ \text{s.t.} \qquad & Lu = b, \end{aligned} \tag{4.1}$$

with Ω is a bounded domain with piece-wise Lipschitz boundary $\partial\Omega$ con-

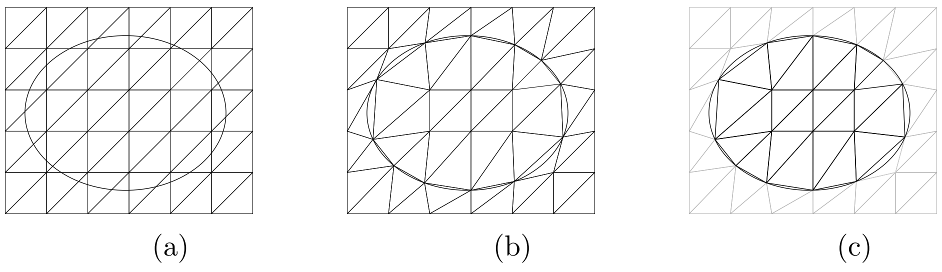


Figure 4.1: Visualization of the three steps of the meshing, $\widetilde{\mathcal{T}}$, $\mathcal{T}^\infty$ and $\mathcal{T}$. From [7].

tained in a bounded set $\hat{\Omega} \subset \mathbb{R}^2$. The functional is of the form $J(\Omega, u) := \int_\Omega f(u(x))dx$.
Furthermore, assume that an initial admissible shape Ω_0 is given. We define a rectangular area $\widetilde{\Omega}$ that contains $\hat{\Omega}$ and thus Ω_0. In other words, all domains $\Omega_i, \quad i = 1, 2, \dots,$ that stem from the optimization process will also lie in $\widetilde{\Omega}$. $\widetilde{\Omega}$ is discretized by a regular grid $\widetilde{\mathcal{T}} := \mathcal{T}_h(\widetilde{\Omega})$ with mesh size h. The number of elements are denoted by $\tilde{N}^{el}$ and the number of nodes by $\tilde{N}^{no}$. As in Figure 4.1a the boundary $\delta\Omega_0$ of the component to be optimized is superimposed onto the grid, represented for example by a set of splines. The mesh size h is assumed to be fine enough such that the following two assumptions hold.

Definition 4.2.1 (Assumptions)**.** If we assume that kinks in $\partial\Omega$ are specifically declared and hence treated separately, then

1. If $\partial\Omega$ intersects an element T, it has exactly two intersections with the boundary of T; in other words, the cell never contains two separate parts of the boundary, that is two parts that are not connected in the area of the cell.

2. The boundary is locally sufficiently smooth, that is it is locally convex (or concave) on the area of one cell, or locally monotone.

The stated assumptions are visualized in Figure 4.2 and can always be satisfied by refining the grid. The grid is adapted to the boundary as follows. For every edge in $\widetilde{\mathcal{T}}$, we check if it intersects with the boundary of Ω. If

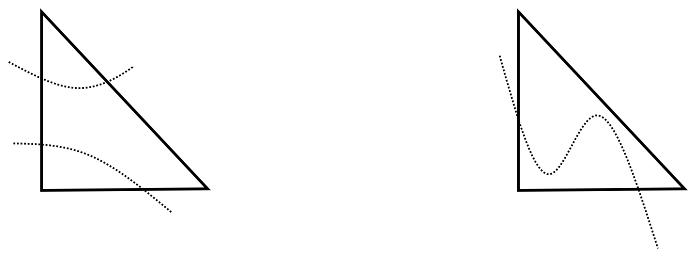

(a) Contradicts assumption 1 (b) Contradicts assumption 2

Figure 4.2: Visualization of the cell boundary interaction that is not allowed.

this is true, we move the vertex that lies closest to the intersection onto the intersection. This simple procedure is given in Algorithm 1 and the result is visualized in Figure 4.1b.

Algorithm 1 Adapt

```
for $k = 1 : 2 : \tilde{N}^{el}$ do
    $n_1, n_2, n_3 = \mathsf{connectivity}(k)$
    for $i, j \in \{1, 2, 3\}$ do
        if $\overline{x_{n_i} x_{n_j}}$ intersects with boundary then
            move closest point to intersection of edge and boundary
        end if
    end for
end for
```

The adapted grid is denoted by $\mathcal{T}^\infty$ or $\mathcal{T}_i^\infty$, where the index i indicates the current optimization step. With this, the grid is divided in two parts; cells, that lie inside Ω_i and cells that lie outside of it. The cells that lie inside of the domain, we call them active cells, form the active mesh $\mathcal{T}_i$. $\mathcal{T}_i$ is the actual computational domain. The computations are only performed on the cells, i.e. elements lying inside the boundary. To refer to this mesh and to distinguish it from $\mathcal{T}_i^\infty$ and $\widetilde{\mathcal{T}}$, we will use the term active mesh or active nodes, etc. $\mathcal{T}_i$ is a subset of $\mathcal{T}_i^\infty$ which itself is a perturbation of $\widetilde{\mathcal{T}}$. This means, all three meshes share the same connectivity, only to access $\mathcal{T}_i$ we need to store the status of each cell as active or inactive. Let us consider Algorithm 1 again. This strategy is highly local. It is not possible to control

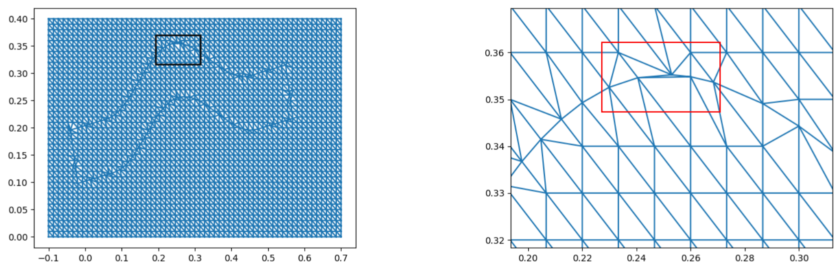


Figure 4.3: Example for a strongly degenerated element.

the regularity of the grid, that is it does not prevent cells from degenerating and therefore the maximum angle condition (2.75) is violated. An example of such a situation is given in Figure 4.3. It is clear that in a worst case scenario, we cannot guarantee for the existence of a $\gamma_0 < \pi$, hence the convergence is not given. Therefore, some sort of pruning is needed to avoid these kind of cells. To do this, a better understanding of the proposed mesh generation approach is needed. So first, we describe the approach of the proposed mesh generation step-by-step and then describe how we guarantee to fulfill the maximum angle condition.
First $\widetilde{\mathcal{T}}$ is initialized. It is determined by $\widetilde{\Omega}$ and $n_x \in \mathbb{R}$ and $n_y \in \mathbb{R}$, being the discretization parameters in the first and second coordinate, respectively. From this, the coordinates of the vertices i.e. the nodes of the grid $\widetilde{\mathcal{T}}$ are generated and a connectivity determining the cells. Both, coordinates of $\widetilde{\mathcal{T}}$ and the connectivity have only to be generated once and will not change during the complete optimization process. Therefore, they can be implemented as functions of the respective indices. The method coordinates-struct is given in Algorithm 2. The numbering is in positive direction of the x-axis first, then in positive direction of the y-axis.

Algorithm 2 method coordinates-struct

```
    Input $\widetilde{\Omega}, n_x, n_y$, index $c$
    Output coordinates $(x_c, y_c)$
1: $h_x = (x_{max}(\widetilde{\Omega}) - x_{min}(\widetilde{\Omega}))/(n_x - 1)$
2: $h_y = (y_{max}(\widetilde{\Omega}) - y_{min}(\widetilde{\Omega}))/(n_y - 1)$
3: $m, r = (c - 1)$ modulo $n_x$
4: $x_c = x_{min}(\widetilde{\Omega}) + h_x \cdot r$
5: $y_c = y_{min}(\widetilde{\Omega}) + h_y \cdot m$
```

The method connectivity is given in algorithm 3. We need to distinguish the cases for even and odd element numbers since it is crucial for the implementation that all elements are oriented in the same direction, to avoid sign errors in the computation.

Algorithm 3 method connectivity

```
     Input $\widetilde{\Omega}, n_x, n_y$, index $k$
     Output connectivity of element $k : (n_{k,1}, n_{k,2}, n_{k,3})$
 1: $m, r = (k - 1)$ modulo $(2 * (n_x - 1))$
 2: if $k$ modulo 2 then
 3:     $n_{k,1} = n_x \cdot m + \lceil r/2 \rceil$
 4:     $n_{k,2} = n_{k,1} + 1$
 5:     $n_{k,3} = n_{k,1} + n_x$
 6: else
 7:     $n_{k,1} = n_x \cdot (m + 1) + r/2 + 1$
 8:     $n_{k,2} = n_{k,1} - 1$
 9:     $n_{k,3} = n_{k,1} - n_x$
10: end if
```

To make the following algorithms more legible, we write mesh instead of $\widetilde{\mathcal{T}}$, etc. or leave it out entirely, if it is clear that we are referring to a method of the mesh. We think of the mesh as an object of a class with attributes and methods, such as the connectivity or the coordinates.

After the initialization of the mesh $\widetilde{\mathcal{T}}$, we generate $\mathcal{T}_i^\infty$ in two steps. At first, we determine for all nodes if they lie inside or outside of Ω, nodes that lie outside are assigned the value 0, nodes that lie inside are assigned the value -1, see Algorithm 4.

Algorithm 4 Initialization status nodes

```
    Input mesh, $\Omega_0$
    Output
 1: initialize mesh.node-status $= \mathbf{0} \in \mathbb{R}^{\tilde{N}^{no}}$
 2: for $k = 1 : 2 : \tilde{N}^{el}$ do
 3:     $j_{loc} =$ mesh.connectivity$(k)$
 4:     $x_{loc} =$ mesh.coordinates$(j_{loc})$
 5:     if $\overline{x_{loc,1}x_{loc,2}}$ intersects $\partial\Omega_0$ then
 6:         if mesh.node-status$[j_{loc}[1] == 0$ then
 7:             mesh.node-status$[j_{loc}[2]] == -1$
 8:         else
 9:             mesh.node-status$[j_{loc}[2]] == 0$
10:         end if
11:     end if
12: end for
```

After this, the actual adaption routine is applied, see Algorithm 5. As already indicated above, the algorithm runs over all elements and checks if their edges intersect the boundary of Ω. If that is the case, it moves the closest node to this intersection. The node that has been moved is assigned the status 1, indicating a boundary node. The coordinates of the node of the adapted grid are stored in a lookup-table, that is initialized with the coordinates of the structured grid $\widetilde{\mathcal{T}}$.

In the last step, the active mesh $\mathcal{T}$ is determined, according to Algorithm 6. For better understanding, it is given as a for-loop but can also be implemented in vector arithmetic. If all nodes of an element show an active status, the element is assigned an active status as well. It rests to determine if an element is an inner element or a boundary element. The latter is the case if at least one node status is equal to 1.

We want to come back to the issue addressed earlier in this chapter. Due to the highly local adapting approach, there might arise distorted elements. These elements can naturally only appear at the boundary. If we restrict ourselves to only move nodes once and with the assumptions stated in Definition 4.2.1, it is easy to see that we actually can control the maximum angle of an

Algorithm 5 Adapt mesh

```
    Input mesh, $\Omega_0$
    Output mesh $\mathcal{T}^\infty$
 1: for $k = 1 : 2 : \tilde{N}^{el}$ do
 2:     $j_{loc}$ = mesh.connectivity($k$)
 3:     $x_{loc}$ = mesh.coordinates($j_{loc}$)
 4:     for every pair $x_{loc,i}, x_{loc,j}$ do
 5:         $x_{inter}$ = intersection($\overline{x_{loc,1}x_{loc,2}}, \partial\Omega_0$)
 6:         if $x_{inter}$ then
 7:             if $\text{dist}(x_{inter}, x_{loc,i}) \leq \text{dist}(x_{inter}, x_{loc,j})$ then
 8:                 mesh.coordinates($j_{loc}[i]$) = $x_{inter}$
 9:                 mesh.node-status[$j_{loc}[i]$] = 1
10:             else
11:                 mesh.coordinates($j_{loc}[j]$)) = $x_{inter}$
12:                 mesh.node-status[$j_{loc}[j]$] = 1
13:             end if
14:         end if
15:     end for
16: end for
```

element if at most two nodes are moved. We can simply write down every possible case and determine the worst case scenario among them. Without loss of generality, let us assume that $h = h_x = h_y$. Since the nodes are moved along the edges of the grid, each node can be moved in six different directions and at most $h/2$ along the vertical or horizontal lines and $\sqrt{2}h/2$ along the diagonals, away from its original location in $\widetilde{\mathcal{T}}$. If only one node is moved, the largest angle appearing is $\alpha = \arccos(-1/\sqrt(5)) \approx 116°$. With the adapt algorithm defined as it is, we only move the nodes along the axes of the structured grid, that is in 6 possible directions. In the case that two nodes are moved we have 3 different combinations of nodes that are moved at the same time, each in one of the possible 6 directions. The only case that is not possible is the case when both would move towards each other. Therefore, $3 \cdot 6^2 - 3$ scenarios are possible. For all of these 105 scenarios, one can look at the worst case and see that for all of them, one can give bounds for each angle, which only depend on the ratio of h_x and h_y. For example, the maximum angle of the worst case scenario element displayed in Figure

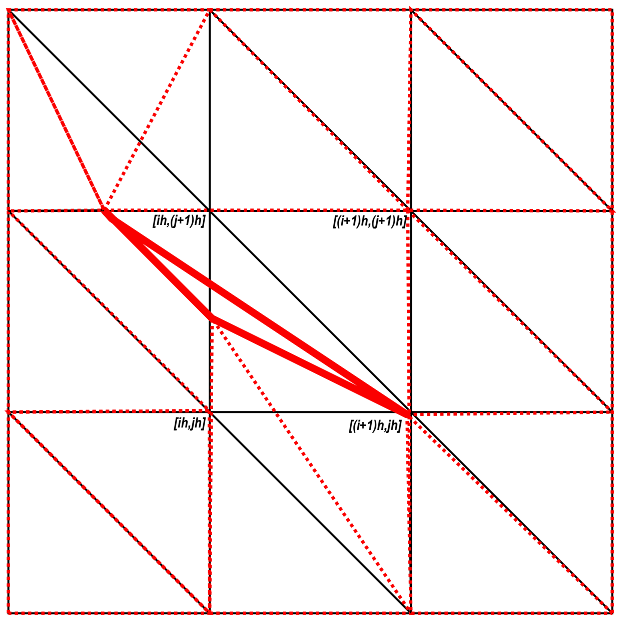


Figure 4.4: Example of worst case element contortion when two nodes are moved.

4.4 is

$$\alpha = \arccos\left(-\frac{3}{\sqrt{10}}\right) \approx 161^\circ.$$

It remains to consider the relatively rare case that all nodes have been adapted to the boundary. In these cases, the maximum angle condition might be violated, as we are not able to determine γ_0 . Therefore, we implement a sort of pruning in the following way.

If all nodes of the considered element have the status 1, that is all of them are boundary nodes, the maximum angle of this element is calculated. If it exceeds a given tolerance γ_{tol}, the status of the element is set to 0, indicating an inactive element. The node at the maximum angle is moved to its initial location in $\widetilde{\mathcal{T}}$. If it lies inside Ω, it is assigned the status -1, that is inner node, if it lies outside it is assigned the status 0, see Algorithm 6. In other words, if the maximum angle of a boundary element exceeds a given tolerance, we reject the element and reset it to its original form in $\widetilde{\mathcal{T}}$. Thus, the part of the boundary that was approximated by the given element, is still approximated by the edge opposing the maximum angle. Through the adapting procedure, it is guaranteed that the length of this edge is bounded by $2\sqrt{2}h$. With this and Theorem 2.2.10, we can state the following theorem.

Theorem 4.2.2. *The triangulation $\mathcal{T}_h$ as described above is admissible and uniform, hence the assumptions of Strang's lemma hold.*
Furthermore, the maximum angles of all elements $T \in \mathcal{T}$ (and also in $\mathcal{T}^\infty$) can be bounded by a constant κ, depending uniformly on γ_{tol} and therefore the maximum angle condition is fulfilled.

Algorithm 6 Determine active mesh $\mathcal{T}$

```
    Input mesh $\mathcal{T}^\infty$
    Output mesh $\mathcal{T}$
 1: initialize mesh.element-status $= \mathbf{0} \in \mathbb{R}^{\tilde{N}^{el}}$
 2: for $k = 1 : \tilde{N}^{el}$ do
 3:     $j_{loc}$ = mesh.connectivity($k$)
 4:     if all mesh.node-status[$j_{loc}$] $\neq 0$ then
 5:         if all mesh.node-status[$j_{loc}$] $== -1$ then
 6:             mesh.element-status[$k$] $== -1$
 7:         else
 8:             mesh.element-status[$k$] $== 1$
 9:             if all mesh.node-status[$j_{loc}$] $== 1$ then
10:                 Calculate angles of element: $angles_k$
11:                 $\gamma_{max}$ = max($angles_k$)
12:                 if $\gamma_{max} > \gamma_{tol}$ then
13:                     mesh.element-status[$k$] $== 0$
14:                     $i_a$ = argmax($angles_k$)
15:                     mesh.coordinates[$i_a$] = mesh.coordinates-struct($i_a$)
16:                     Determine all neighboring nodes of $i_a$: $neighbors_a$
17:                     if all mesh.node-status($neighbors_a$) $\neq 0$ then
18:                         mesh.node-status[$i_a$] $== -1$
19:                     else
20:                         mesh.node-status[$i_a$] $== 0$
21:                     end if
22:                 end if
23:             end if
24:         end if
25:     end if
26: end for
```

4.2.2 How to treat kinks

In Definition 4.2.1 we stated that kinks are treated separately. Since kinks have a crucial influence on the stress on a domain and thus on the reliability of a component, the tips of the kinks have to be resolved precisely. Therefore, before adapting $\widetilde{\mathcal{T}}$ to $\widetilde{\mathcal{T}}^{\infty}$ for each of these tips we could simply move the respective closest grid point onto the location of the tip, assign it the node status 1 indicating boundary node and store the indices of these nodes in a list I_{kinks}. This however implies that the assumptions that we stated do develop the pruning routine we introduced in the previous section would not hold anymore since these nodes are not moved along the edges of the mesh. Additionally, if in Algorithm 5 the node that has to be removed from the active mesh happens to be a node in I_{kinks}, that would mean that the tip of that kink would not be resolved by the mesh anymore. Therefore, we need a more evolved approach to treat these cases, see Algorithm 7. As these kinks in a domain usually appear only in a very low number this is acceptable. With this procedure we make sure that the choice of the node that is moved to the tip of the kink will not lead to a degenerated element in the adapting procedure which follows.

4.2.3 Shape optimization on the structured mesh

Previously, we introduced the meshing approach and described how to generate the initial mesh $\mathcal{T}$. Now assume that from an optimization step, we generated a displacement field $u_i : \Omega_i \longrightarrow \Omega^*$ which is an admissible descent in the sense that $\Omega_{i+1} := \Omega_i + u(\Omega_i) \subset \Omega^*$. Now the mesh generation differs from generating the initial mesh in several ways. First of all, we can skip the initialization, since $\widetilde{\mathcal{T}}$ stays the same for any optimization step and therefore the methods coordinates-struct and connectivity stay the same as well. We can further assume that $\max_{x\in\Omega_i} \|u_i(x)\| \leq h$. On the one hand this assumption is satisfied in most steps of the optimization process anyway, due to constraints on the regularity of the boundary. On the other hand we can embed this assumption in the optimization process by enforcing it in an upper boundary for the step length in the optimization process, for example. If we

Algorithm 7 Pretreatment of kinks

```
    Input mesh $\widetilde{\mathcal{T}}$, array $X_{kinks}$ with location of kink tips
 1: initialize list $I_{kinks}$
 2: for $\hat{x}$ in $X_{kinks}$ do
 3:     Find element $T_k$ that contains $\hat{x}$
 4:     $j_{loc}$ = mesh.connectivity($k$)
 5:     $i$ = argmin(dist($\hat{x}$, mesh.coordinates-struct($j_{loc}$)))
 6:     $i_c = j_{loc}[i]$
 7:     mesh.coordinates[$j_{loc}[i]$] = $\hat{x}$
 8:     $x^s_{i_c-1}$ = mesh.coordinates-struct($i_c - 1$)
 9:     $x^s_{i_c-N_x}$ = mesh.coordinates-struct($i_c - N_x$)
10:     $x^s_{i_c+1}$ = mesh.coordinates-struct($i_c + 1$)
11:     $x^s_{i_c+N_x}$ = mesh.coordinates-struct($i_c + N_x$)
12:     if either edge $\overline{x^s_{i_c-1}x^s_{i_c-N_x}}$ or edge $\overline{x^s_{i_c+1}x^s_{i_c+N_x}}$ intersect $\partial\Omega$ then
13:         if dist($x_{inter}, \hat{x}$) < constant $c(h)$ then
14:             $I_{pot} = [i_c - 1, i_c - N_x, i_c + 1, i_c + N_x]$
15:             $i_c = I_{pot}$[argmin(dist($\hat{x}, X_{I_{pot}}$))]
16:         end if
17:     end if
18:     mesh.coordinates[$i_c$] = $\hat{x}$
19:     mesh.node-status[$i_c$] = 1
20:     append($I_{kinks}, i_c$)
21: end for
```

bound the deformation of the domain in such a way, it is clear that $\partial\Omega_{i+1}$ can only lie in the cells of $\widetilde{\mathcal{T}}$ that are boundary elements in $\mathcal{T}_i$ or their neighbor elements. Therefore, in the adapt-algorithm, we only have to run through those elements. This reduces the cost of the for-loop in this algorithm from quadratic in h to only linear, compare Algorithm 8. The restriction to the step length gives another advantage. Thereby we made sure that the status of a node can never change from inner node to outer node or vice versa. It can only change from inner node to boundary node and so on. Hence, the algorithm to determine the status of the nodes that have not been adapted in Algorithm 8 reduces to a short procedure described in Algorithm 9. The boundary nodes are already determined in Algorithm 8. The very most of the inner nodes can be found by setting the difference of the inner nodes of $\mathcal{T}_i$ and the boundary nodes of $\mathcal{T}^{\infty}_{i+1}$. After this, it only remains to determine

Algorithm 8 Update mesh

```
    Input mesh $\mathcal{T}^{\infty}$, old-mesh $\mathcal{T}_i$
    Output mesh $\mathcal{T}^{\infty}_{i+1}$
 1: initialize mesh.node-status $= \mathbf{0} \in \mathbb{R}^{\tilde{N}^{no}}$
 2: $K_{active} =$ find(old-mesh.element-status $== 1$)
 3: $check = 0$
 4: for $k = 1$ in $K_{active}$ do
 5:     $j_{loc} =$ mesh.connectivity($k$)
 6:     $x_{loc} =$ mesh.coordinates($j_{loc}$)
 7:     for every pair $x_{loc,i}, x_{loc,j}$ do
 8:         $x_{inter} =$ intersection($\overline{x_{loc,1}x_{loc,2}}, \partial\Omega_0$)
 9:         if $x_{inter}$ then
10:             check = 1
11:             if dist($x_{inter}, x_{loc,i}$) $\leq$ dist($x_{inter}, x_{loc,j}$) then
12:                 mesh.coordinates($j_{loc}[i]$) $= x_{inter}$
13:                 mesh.node-status[$j_{loc}[i]$] $= 1$
14:             else
15:                 mesh.coordinates($j_{loc}[j]$)) $= x_{inter}$
16:                 mesh.node-status[$j_{loc}[j]$] $= 1$
17:             end if
18:         end if
19:     end for
20:     if $!check$ then
21:         check = 0
22:         for $m$ in (neighbor elements of $k$) do
23:             $j_{loc} =$ mesh.connectivity($m$)
24:             $x_{loc} =$ mesh.coordinates($j_{loc}$)
25:             for every pair $x_{loc,i}, x_{loc,j}$ do
26:                 $x_{inter} =$ intersection($\overline{x_{loc,1}x_{loc,2}}, \partial\Omega_0$)
27:                 if $x_{inter}$ then
28:                     check = 1
29:                     if dist($x_{inter}, x_{loc,i}$) $\leq$ dist($x_{inter}, x_{loc,j}$) then
30:                         mesh.coordinates($j_{loc}[i]$) $= x_{inter}$
31:                         mesh.node-status[$j_{loc}[i]$] $= 1$
32:                     else
33:                         mesh.coordinates($j_{loc}[j]$)) $= x_{inter}$
34:                         mesh.node-status[$j_{loc}[j]$] $= 1$
35:                     end if
36:                 end if
37:             end for
38:         end for
39:     end if
40: end for
```

the status of the nodes that have been boundary nodes in optimization step (i) but are no longer boundary nodes in the current optimization step $(i+1)$.

Algorithm 9 Update status nodes

```
 1: $j_{bound}$ = find(mesh.node-status = 1)
 2: $j_{old-inner}$ = find(old-mesh.node-statu s = −1)
 3: $j_{old-bound}$ = find(old-mesh.node-statu s = 1)
 4: $j_{inner} = j_{old-inner} \setminus j_{bound}$
 5: mesh.node-status[$j_{inner}$] = −1
 6: $j_{potential} = j_{old-bound} \setminus j_{bound}$
 7: for $j$ =  in $j_{potential}$ do
 8:     Determine all neighboring nodes of $j$: $neighbors_j$
 9:     $x_{loc}$ = mesh.coordinates($j_{loc}$)
10:     if any mesh.node-status($neighbors_j$) == −1 then
11:         mesh.node-status[$j$] == −1
12:     else if all mesh.node-status($neighbors_j$)! = 0 then
13:         mesh.node-status[$j$] == 1
14:     end if
15: end for
```

Finally, the active mesh has to be determined in the same ways as in the initial step via Algorithm 6.
The proposed procedure brings other advantages in the assembly of the linear system to be solved, which we discuss in chapter 6.

4.2.4 Higher order finite elements

Although we prefer to work with linear finite elements whenever possible, in some cases higher order finite elements are necessary to maintain the regularity of the problem. The presented meshing approach gives a rather elegant way to obtain these higher elements. Recall the finite elements P_t from section 2.2. The basis nodes for the linear P_1-element lie on the corners of the triangle. The basis nodes of the quadratic element P_2 lie on the same positions, additionally to one node on the middle of each edge. For the cubic P_3-element we find two equally distant basis nodes on each edge and one in the center of the cell, additionally to the three nodes on the vertices of the cell.

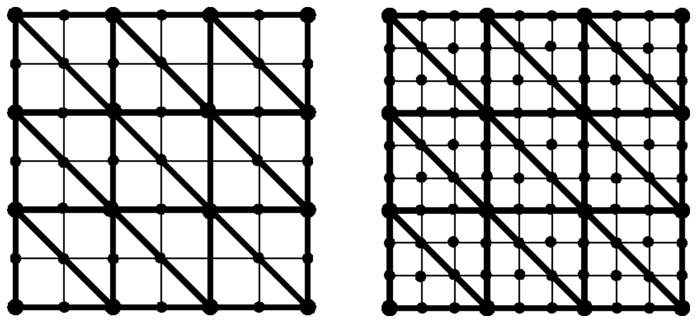


Figure 4.5: Higher order finite element meshes.

That means for $\widetilde{\mathcal{T}}_h$, the basis nodes for higher order finite elements lie on the vertices of a grid $\widetilde{\mathcal{T}}_{h/2}$ or $\widetilde{\mathcal{T}}_{h/3}$, etc. We can make use of this circumstance when creating a higher order finite element mesh. For simplicity, let us look at the case of P_2 finite elements. We proceed as follows. First, we create the usual mesh $\mathcal{T}_h$. Second, we create a mesh $\widetilde{\mathcal{T}}_{h/2}$. Through the structure of the grid, it is easy to map the nodes of the original grid $\widetilde{\mathcal{T}}_h$ to those that coincide with them on $\widetilde{\mathcal{T}}_{h/2}$, with a mapping $R : \mathbb{N} \longrightarrow \mathbb{N}$. With this, we can easily construct a connectivity for the elements defined on $\widetilde{\mathcal{T}}_h$ but the nodes defined on $\widetilde{\mathcal{T}}_{h/2}$. The coordinates of those nodes that exist in both meshes are set to the ones of $\mathcal{T}_h$. The boundary nodes that only exists on the finer grid are moved to the respective center of the edge between the two adapted edges via vector arithmetic. To summarize, the mesh is adapted as usual but the connectivity for the mesh now points to nodes and node indices that lie on a once refined mesh.

4.2.5 Adapting in three dimensions

The three dimensional meshing is conducted similar to the two-dimensional meshing. Again, the adapting process is split into three parts, additionally to an initialization phase. The admissible cuboid $\widetilde{\Omega}$ is discretized by a regular $N_x \times N_y \times N_z$ tetrahedral finite element grid. The tetrahedrons are arranged

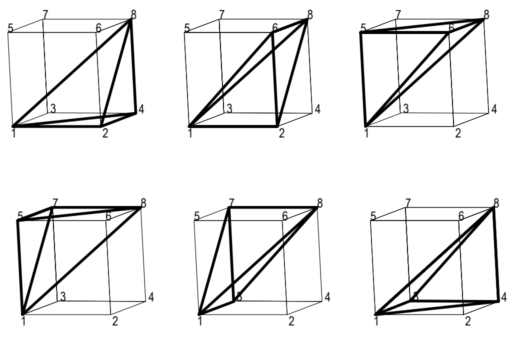


Figure 4.6: Triangulation of one cuboid in the 3D structured mesh.

according to the triangulation Type B in [36]. It has the advantage that the connectivity is easier to determine as in other triangulations, since every cuboid can be triangulated in exactly the same way, whereas most other triangulations alternate the orientation of the triangulation from one cuboid to the next. Additionally, if $\widetilde{\Omega}$ is uniformly partitioned, every simplex is of the same volume.

The methods coordinates-struct and connectivity are build in the same way as in two dimensions, but with six different cases in the connectivity.

The representation of Ω

Now we assume to have the boundary of Ω given as a list of clustered point clouds, equipped with a triangulation as visualized in Figure 4.7. Each cluster represents a part of the boundary of Ω. As the points form a surface, the triangulation is achieved by projecting the points onto a two-dimensional plane, see Figure 4.8. Therefore, for each cluster we store the information over which plane it is defined. Naturally that means that the points are pairwise distinguishable by their entries on that plane. Additionally, we assume that points at the edges of each cloud lie in the adjacent cloud as well, such that the boundary is closed. These points form the edges of the

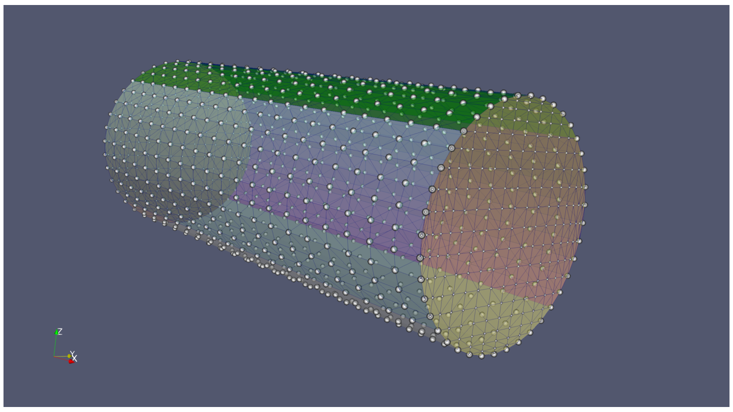
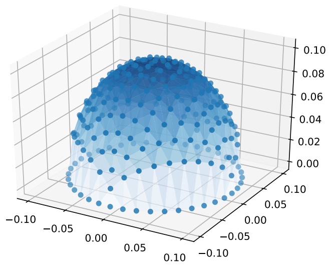


Figure 4.7: Example of clustered point cloud representation of the surface of a cylinder.

component. On each cell of the boundary's triangulation, we assume a linear interpolation, such that the boundary is given as piecewise linear.
We distinguish three different classes of points: the first class contains points that lie in the end of the edges. From the assumptions above it follows that they lie on more than one edge and therefore belong to more than two surfaces. Therefore, they may be the tip of a kink and have to be resolved exactly; secondly the class of edge points: these points are points defining an edge but are not corner nodes and therefore lie on exactly two surfaces. We can assume that for each edge, these points form a one dimensional curve, which has to be resolved by the mesh; and the last class of surface points that lie on exactly one surface.

The adapting process

In the first part of the adaption process we determine if a node of the regular grid $\widetilde{T}$ lies outside or inside the given component Ω_0. Each node is assigned a status 'outside' (0) or 'inside' (-1). This is done by a for-loop over all line segments in z-direction. For each of these line segments $\overline{x_i x_{i+1}}$, we check if we have an intersection with the boundary of the component. If so, the node x_{i+1} is assigned the opposed value of x_i, if not it is assigned the same status. We assume that node x_0 always lies outside of Ω.
Again, in the second step we adapt the regular grid to the boundary. We first


(a) Point cloud in $\mathbb{R}^3$

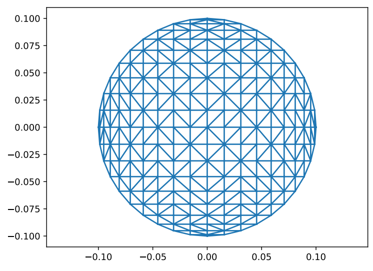

(b) Projection onto $\mathbb{R}^2$

Figure 4.8: Visualization of the projection of the three dimensional surface onto the two-dimensional plane.

adapt the grid to the corner nodes, such that corners are perfectly retained in the mesh. We simply drag the closest grid node in an euclidean sense to the location of the respective corner node. The respective index and the indices of the corresponding surfaces of a corner node are stored in a list of key value pairs, with the index of the corner node being the key and the indices of the corresponding surfaces being the values stored as a set. In a second step, we adapt the mesh to the edges. For that purpose, we assume the edge points to be a grid point of a linear interpolated curve. For each considered element of the mesh, we check for all faces of the respective element if they intersect with this curve. If that is the case, we move the closest node of the face to the intersection and mark this node as edge node. The edge nodes as well as the corner nodes are stored in a separate list. After adapting the mesh to the corner nodes and edges, the rest of the mesh is adapted by Algorithm 10. Instead of a for-loop over all elements, we can determine the set of those elements that contain both inner nodes and outer nodes. This set can be determined by vector arithmetic, though in Algorithm 10 we write it as a for-loop. In the end of the process, each node is assigned a status, 'inside', 'outside' or 'boundary' with the number indicating which part of the boundary it belongs to.

Algorithm 10 Adapt 3D mesh

```
1:  $elements_{check} = []$
2:  **for** $k = 1 : \tilde{N}^{el}$ **do**
3:      $I_{loc} = \mathsf{connectivity}(k)$
4:      **if** any $\mathsf{node\text{-}status}(I_{loc}) == -1$ and any $\mathsf{node\text{-}status}(I_{loc}) == 0$ **then**
5:          $\mathsf{append}(elements_{check}, k)$
6:      **end if**
7:  **end for**
8:  **for** $k$ in $elements_{check}$ **do**
9:      $I_{loc} = \mathsf{connectivity}(k)$
10:     $x_{loc} = \mathsf{coordinates}(I_{loc})$
11:     **for** $i = 1, \cdots, 3$ **do**
12:         **for** $j = i + 1, \cdots, 4$ **do**
13:             **for** $S$ in $surfaces$ **do**
14:                 $x_{inter} = \mathsf{intersection}(S, \overline{x_{loc,i}, x_{loc,j}})$
15:                 **if** $x_{inter}$ **then**
16:                     **if** $\mathrm{dist}(x_{inter}, x_{loc,i}) \leq \mathrm{dist}(x_{inter}, x_{loc,j})$ **then**
17:                         $\mathsf{coordinates}(j_{loc}[i]) = x_{inter}$
18:                         $\mathsf{node\text{-}status}[j_{loc}[i]] = 1$
19:                     **else**
20:                         $\mathsf{coordinates}(j_{loc}[j])) = x_{inter}$
21:                         $\mathsf{node\text{-}status}[j_{loc}[j]] = 1$
22:                     **end if**
23:                 **end if**
24:             **end for**
25:         **end for**
26:     **end for**
27: **end for**
```

Chapter 5

Krylov subspace recycling

In algorithms solving shape optimization problems, similar to topology optimization, often hundreds or thousands of optimization steps are performed to reach a converged optimal solution. When the problem is PDE constrained, at least the same number of linear system of equations representing the constraint are solved. These systems are sparse and especially in three dimensions very large. For decades, iterative solvers have been developed and proven to be the method of choice to solve these large sparse systems, the family of Krylov subspace methods in particular.
In many applications, such as simulations of physical behavior over time (fluid dynamics, etc.) or shape and topology optimization, long sequences of linear systems appear which are subject to only small changes over time. In recent years, the method of Krylov subspace recycling gained popularity [62], which aims to carry over information from one system to the next to reduce the number of iterations in the respective iterative solve.
In this section, we will introduce the general idea of Krylov subspace methods, followed by a discussion of Krylov subspace recycling. Subsequently we will propose an approach to apply a recycling method in the setting of shape optimization, with numerical examples for the meshing technique proposed in Chapter 4.
Section 5.2 and Section 5.3 have been published in a similar form in [7].

5.1 Krylov subspace methods

Krylov subspace methods are widely used iterative solvers for large sparse linear systems. To introduce these methods, this chapter follows [20, 35, 57]. Assume a given linear system $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{n\times n}, \mathbf{b} \in \mathbb{R}^n$ and assume n to be very large. The idea is to project a problem of dimension n into a lower-dimensional space, a Krylov subspace, to find an approximate solution here and then iteratively expand the dimension of the subspace.
Let $\mathbf{x}_0 \in \mathbb{R}^n$ be an initial guess with the residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$. The corresponding mth-Krylov subspace is the space defined by

$$\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0) := \text{span}\{\mathbf{r}_0, \mathbf{Ar}_0, \mathbf{A}^2\mathbf{r}_0, \cdots, \mathbf{A}^{m-1}\mathbf{r}_0\} \tag{5.1}$$

A Krylov subspace method finds the solution of the given linear system by iteratively expanding the subspace in which the current solution lies. In iteration m it finds a solution

$$\mathbf{x}_m \in \mathbf{x}_0 + \mathcal{K}_m, \quad \text{with} \tag{5.2}$$

$$\mathbf{r}_m \perp \mathcal{L}_m, \tag{5.3}$$

where $\mathcal{L}_m$ some subspace, for example $\mathcal{L}_m$=$\mathcal{K}_m$. Krylov subspace methods are advantageous for numerous reasons, for example for the fact that only matrix-vector operations are needed and thus the system only needs to be stored as an operator therefore. Additionally, most Krylov subspace methods find, in exact arithmetic, the true solution in a finite number of steps, although a very good approximation is often already derived after much fewer steps.

5.1.1 MINRES

There exist many different iterative methods to solve large sparse systems of equations. Which method to chose depends on the properties of the system matrix. The most common methods can be divided in two approaches

$$\mathcal{L}_m = \mathcal{K}_m, \quad \text{(Galerkin method)}, \tag{5.4}$$

$$\mathcal{L}_m = \mathbf{A}\mathcal{K}_m, \quad \text{(Minimal residual method)}. \tag{5.5}$$

One of the oldest and most renowned methods is the conjugate gradients method (CG) introduced by Hestenes and Stiefel in 1952 [28] for symmetric positive definite linear systems of equations, which can be categorized as a Galerkin method. For the more general case that the system matrix $\mathbf{A}$ is regular, GMRES (generalized minimal residuals) [45] is the standard tool to solve the linear system of equations. As the name suggests, it is a minimal residual method. Here, at iteration m, the approximate $x_m \in \mathcal{K}_m$ is the vector that minimizes the norm of the residual $||\mathbf{r}_m|| = \|\mathbf{b} - \mathbf{A}\mathbf{x}_m\|$ over $\mathcal{K}_m$. MINRES [40] is a variation of GMRES for the special case of problems with hermitian system matrix $\mathbf{A}$.
In the following, we will derive MINRES for regular symmetric matrices step by step. As it is sufficient for the purpose of this work, we only consider real valued problems. Again we want to solve the equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ with $\mathbf{A} \in \mathbb{R}^{n \times n}$, with $\mathbf{A}$ is symmetric, sparse and $n \in \mathbb{N}$ is very large. With an initial guess $\mathbf{x}_0$ and the residual $\mathbf{r}_0$, a minimal residual method finds in every iteration k the solution of the minimization problem

$$x_k := \text{argmin}_{x \in V_k} \|r(x)\|,$$

with $V_k := x_0 + \mathcal{K}_k(\mathbf{A}, r_0)$ and

$$r_k = b - \mathbf{A}x_k \in r_0 + \mathbf{A}\mathcal{K}_k(\mathbf{A}, r_0), \quad r_k \perp \mathbf{A}\mathcal{K}_k(\mathbf{A}, r_0).$$

In words, in iteration k, the weighted image of the residual $\mathbf{s}_k = \mathbf{A}\mathbf{r}_k$ is added to $\mathbf{r}_k$ in a way that the norm $\|\mathbf{r}_{k+1}\|_2$, $\mathbf{r}_{k+1} := \mathbf{r}_k - \varepsilon \cdot \mathbf{s}_k$ is minimized. This is achieved by securing that the added vector $\mathbf{r}_{k+1}$ is orthogonal to all the images $\mathbf{s}_0, \cdots, \mathbf{s}_k$. In other words, in each iteration, we find an orthogonal basis of the current supspace by expanding the orthogonal basis of the subspace of the previous iteration. To deduce an algorithm to do so, consider the following.

A matrix $\mathbf{H}$ is called upper Hessenberg, if for all $i > j + 1 : h_{ij} = 0$, i.e. all entries below the first subdiagonal equal zero. A Hessenberg reduction of $\mathbf{A}$ is a orthogonal similarity transformation

$$\mathbf{AQ} = \mathbf{QH}. \tag{5.6}$$

As we assume $\mathbf{A}$ to be very large, we only want to consider the first k columns of $\mathbf{AQ} = \mathbf{QH}$. Let $\mathbf{Q}_k$ denote the first k columns of $\mathbf{Q}$ and let $\tilde{\mathbf{H}}_k$ be the $k + 1 \times k$-upper-left section of $\mathbf{H}$. It is

$$\mathbf{AQ}_k = \mathbf{Q}_{k+1}\tilde{\mathbf{H}}_k. \tag{5.7}$$

With $\mathbf{q}_i$ denoting the ith-column vector of $\mathbf{Q}$ and h_{ij} denoting the entries of $\mathbf{H}$, the kth column of (5.7) can be written as

$$\mathbf{Aq}_k = h_{1,k}\mathbf{q}_1 + \cdots + h_{kk}\mathbf{q}_k + h_{k+1k}\mathbf{q}_{k+1}. \tag{5.8}$$

That means the kth vector can be derived by a recursive orthogonalization scheme. The Arnoldi iteration in Algorithm 11 is such a scheme which is based on the Gram-Schmidt iteration. With this, the minimization problem in iteration k can be rewritten as

$$\|\mathbf{b} - \mathbf{Ax}_k\|_2 = \|\mathbf{b} - \mathbf{AQ}_i\mathbf{y}\|_2 = \|\|\mathbf{r}_0\|_2\mathbf{Q}_{k+1}\mathbf{e}_1 - \mathbf{Q}_{k+1}\tilde{\mathbf{H}}_k\mathbf{y}\|_2. \tag{5.9}$$

With $\mathbf{Q}_{k+1}$ being an orthonormal transformation with respect to $\mathcal{K}_{k+1}$, this reduces to

$$\|\mathbf{b} - \mathbf{Ax}_k\|_2 = \|\|\mathbf{r}_0\|_2\mathbf{e}_1 - \tilde{\mathbf{H}}_k\mathbf{y}\|_2, \tag{5.10}$$

which leads to solving a small least squares problem. From (5.8) it is obvious that the columns of $\mathbf{Q}$ form a orthogonal basis of the Krylov subspaces that are utilized in the Krylov subspace methods. Hence, the Arnoldi iteration in Algorithm 11 can be used to successively produce the orthogonal basis vectors we are looking for.

Algorithm 11 Arnoldi iteration

```
1: $\mathbf{b}$ arbitrary, $\mathbf{q}_1 = \mathbf{b}||\mathbf{b}||$
2: for $k = 1, 2, 3, \cdots$ do
3:     $\mathbf{v} = \mathbf{A}\mathbf{q}_n$
4:     for $j = 1, \cdots, k$ do
5:         $h_{jk} = \mathbf{q}_j \cdot \mathbf{v}$
6:         $\mathbf{v} = \mathbf{v} - h_{jk}\mathbf{q}_j$
7:     end for
8:     $h_{k+1,k} = ||\mathbf{v}||$
9:     $\mathbf{q}_{k+1} = \mathbf{v}/h_{k+1,k}$
10: end for
```

Until this point, we have not made use of the fact that we assume $\mathbf{A}$ to be symmetric. When we look at the inner for-loop in algorithm 11, it is obvious that for increasing k, the computational cost becomes higher and higher. This is because the new basis vector has to be orthogonalized against all the existing basis vectors. For symmetric matrices, the Hessenberg matrix in (5.6) reduces to a tridiagonal matrix. That means, the new basis vector only needs to be orthogonalized against the previous two basis vectors, and thus the inner for-loop in the Arnoldi Iteraion reduces to a so called *three-term recurrence*. The corresponding algorithm is the Lanczos Iteration in Algorithm 12.

Algorithm 12 Lanczos iteration

```
1: $\beta_0 = 0, \quad \mathbf{q}_0 = 0, \quad \mathbf{b} = \text{arbitrary}, \quad \mathbf{q}_1 = \mathbf{b}/||b||$
2: for $n = 1, 2, 3, \ldots$ do
3:     $\mathbf{v} = \mathbf{A}\mathbf{q}_n$
4:     $\alpha_n = \mathbf{q}_n^T\mathbf{v}$
5:     $\mathbf{v} = \mathbf{v} - \beta_{n-1}\mathbf{q}_{n-1} - \alpha_n\mathbf{q}_n$
6:     $\beta_n = ||\mathbf{v}||$
7:     $\mathbf{q}_{n+1} = \mathbf{v}/\beta_n$
8: end for
```

Let us denote the tridiagonal matrix by $\mathbf{T}_{k+1,k}$, emphasizing with this notation that only this element in the subdiagonal is unequal to zero due to the previous iterations. We see right away that we obtain a QR-factorization of $\mathbf{T}_{k+1,k}$ with one simple Givens rotation:

$$\mathbf{T}_{k+1,k} = \tilde{\mathbf{Q}}_{k+1,k}\mathbf{R}_{k,k}. \tag{5.11}$$

This, together with (5.10) and the Lanczos iteration lead to MINRES.

Algorithm 14 The unpreconditioned MINRES algorithm, as found in [59]

```
1: $\mathbf{x}_0 =$ arbitrary
2: $\mathbf{v}_1 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
3: $\beta_1 = \|\mathbf{v}_1\|_2; \eta = \beta_1;$
4: $\gamma_1 = \gamma_0 = 1; \sigma_1 = \sigma_0 = 0;$
5: $\mathbf{v}_0 = 0; \mathbf{w}_0 = \mathbf{w}_{-1} = 0;$
6: for $i = 1, 2, 3, \ldots$ do
7:     Lanczos recurrence:
8:     $\mathbf{v}_i = \frac{1}{\beta_i}\mathbf{v}_i; \mathbf{v} = \mathbf{A}\mathbf{v}_i$
9:     $\alpha_i = \mathbf{v}_i^T\mathbf{v};$
10:    $\mathbf{v}_{i+1} = \mathbf{v} - \alpha_i\mathbf{v}_i - \beta_i\mathbf{v}_{i-1}$
11:    $\beta_{i+1} = \|\mathbf{v}_{i+1}\|_2$
12:    QR-factorization with Givens-Rotations:
13:    $\delta = \gamma_i\alpha_i - \gamma_{i-1}\sigma_i\beta_i;$
14:    $\rho_1 = \sqrt{\delta^2 + \beta_{i+1}^2};$
15:    $\rho_2 = \sigma_i\alpha_i + \gamma_{i-1}\gamma_i\beta_i;$
16:    $\rho_3 = \sigma_{i-1}\beta_i$
17:    $\gamma_{i+1} = \delta/\rho_1; \sigma_{i+1} = \beta_{i+1}$
18:    Update:
19:    $\mathbf{w}_i = (\mathbf{v}_i - \rho_3\mathbf{w}_{i-2} - \rho_2\mathbf{w}_{i-1})/\rho_1$
20:    $x_i = x_{i-1} + \gamma_{i+1}\eta\mathbf{w}_i$
21:    $\eta = -\sigma_{i+1}\eta$
22:    if convergence criterion is satisfied then
23:        break
24:    end if
25: end for
```

5.2 Krylov subspace recycling

To motivate Krylov subspace recycling, we will look into the convergence behavior of Krylov subspace methods, following [33] and [59]. A vector $x \in$

$\mathcal{K}_m(A, b)$ can be written as

$$x = c_0\mathbf{b} + c_1\mathbf{A}\mathbf{b} + c_2\mathbf{A}^2\mathbf{b} + \cdots + c_{m-1}\mathbf{A}^{m-1}\mathbf{b}. \tag{5.12}$$

This is nothing but a polynomial in $\mathbf{A}$ times $\mathbf{b}$. It is well known [22] that in every iteration m of the MINRES, the norm of the residual satisfies

$$\|\mathbf{r}_m\|_2 = \min_{p \in \Pi_m^{(0)}} |p(\mathbf{A})\mathbf{r}_0|. \tag{5.13}$$

With this, one can easily see that the relative residual norm in each step can be bounded by

$$\frac{||\mathbf{r}_m||}{|||\mathbf{r}_0||} \leq \min_{p_m \in \Pi_m^{(0)}} \max_{\lambda \in \lambda(\mathbf{A})} |p_m(\lambda)|, \tag{5.14}$$

where $\Pi_m^{(0)}$ is the set of polynomials p_m of degree m such that $p_m(0) = 1$ and $\lambda(\mathbf{A})$ is the set of eigenvalues of $\mathbf{A}$. Obviously, the convergence rate depends on the spectrum of the matrix. Many approaches to augment Krylov subspace methods base on the idea to somehow remove an appropriate subset of the eigenvalues, denoted by M such that the bound

$$\min_{p_m \in \Pi_m^{(0)}} \max_{\lambda \in (\lambda(\mathbf{A}) \setminus M)} |p_m(\lambda)| \tag{5.15}$$

can be significantly smaller. The idea of Krylov subspace recycling [31] is the following. Assume there is a series of linear systems

$$\mathbf{A}^{(i)}\mathbf{x} = \mathbf{b}^{(i)}, \tag{5.16}$$

with $\mathbf{A}^{(i)} \in \mathbb{R}^{n \times n}$, $\mathbf{b}^{(i)} \in \mathbb{R}^n$ that change only little over time. Then, one could assume that the spectrum of the matrices might change only little as well. Krylov subspace recycling aims to carry over knowledge about the spectrum of the system matrix in iteration (i) to iteration $(i + 1)$ by recycling a judiciously selected subspace of $\mathbf{range}(\mathbf{A}^{(i)})$.

To avoid confusion and as the series of linear systems in this work clearly stems from optimization processes, in the following we will refer to one iteration of this optimization process as optimization step, the term iteration will from now on always refer to an iteration of the MINRES or (rMINRES). Due to the normalizing condition $p_m(0) = 1$, it is easy to see from (5.13) and (5.15) that it is favorable to remove small eigenvalues. An approach to do this is via deflation. Here, a suitable invariant subspace of the system matrix $\mathbf{range}(\mathbf{A}^{(i)})$ from one optimization step is recycled for the next optimization step in the following way. Consider $\mathbf{W}^{(i)} \in \mathbb{R}^{n\times k}$ with k very small and $\mathbf{range}(\mathbf{W}^{(i)})$ being a (nearly) invariant subspace of $\mathbf{range}(\mathbf{A}^{(i)})$. As we assume only small changes from the linear system of equations in optimization step i to the one in optimization step $i+1$, we can assume that $\mathbf{range}(\mathbf{W}^{(i)})$ is an approximate invariant subspace of $\mathbf{range}(\mathbf{A}^{(i+1)})$. Before the actual MINRES is started, we first optimize over $\mathbf{range}(\mathbf{W}^{(i)})$. In [41] and [52] it was shown that with this augmentation, the method nearly converges as if the residual vector has almost no components lying in $\mathbf{range}(\mathbf{W}^{(i)})$. Hence, if $\mathbf{range}(\mathbf{W}^{(i)})$ corresponds to the k absolute smallest eigenvalues, this leads to an improved convergence.
The introduction above refers to a rather specific recycling subspace approach. A survey of general Krylov subspace recycling methods can be found in [53]. Here, following [7], we will describe the recycling MINRES (rMINRES) algorithm from [38, 62], which is a recycling version of the MINRES algorithm introduced in the section before. As we are looking at the rMINRES at one optimization step, for the sake of clarity we drop the index (i) from here on. Consider at some optimization step the linear system $\mathbf{Ax} = \mathbf{b}$ with $\mathbf{A} \in \mathbb{R}^{n\times n}$, and let the space to be recycled be defined as $\mathbf{range}(\mathbf{W}_k)$ with $\mathbf{W}_k \in \mathbb{R}^{n\times k}$ computed in the previous optimization step. We compute $\widetilde{\mathbf{C}}_k = \mathbf{AW}_k$ and its thin QR decomposition $\widetilde{\mathbf{C}}_k = \mathbf{C}_k\mathbf{R}_k$. Furthermore, assume an initial guess $\widetilde{\mathbf{x}}_0$ and corresponding residual $\widetilde{\mathbf{r}}_0 = \mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}}_0$. We first update the initial guess, adding the optimal correction (in minimum residual norm) from the space $\mathbf{range}(\mathbf{W}_k)$. We set $\mathbf{x}_0 = \widetilde{\mathbf{x}}_0 + \mathbf{W}_k(\mathbf{R}_k^{-1}\mathbf{C}_k^T\widetilde{\mathbf{r}}_0)$, which gives the updated residual $\mathbf{r}_0 = \widetilde{\mathbf{r}}_0 - \mathbf{C}_k\mathbf{C}_k^T\widetilde{\mathbf{r}}_0 = (\mathbf{I} - \mathbf{C}_k\mathbf{C}_k^T)\widetilde{\mathbf{r}}_0$.

We implement recycling Lanczos as follows. Let $\mathbf{v}_1 = \mathbf{r}_0/\|\mathbf{r}_0\|_2$. We use the following augmented three-term recurrence

$$\begin{aligned}\mathbf{v}_2 t_{2,1} &= \mathbf{A}\mathbf{v}_1 - \mathbf{C}_k\mathbf{p}_1 - \mathbf{v}_1 t_{1,1}, &(5.17)\\ \mathbf{v}_{j+1} t_{j+1,j} &= \mathbf{A}\mathbf{v}_j - \mathbf{C}_k\mathbf{p}_j - \mathbf{v}_j t_{j,j} - \mathbf{v}_{j-1} t_{j-1,j}, &(5.18)\end{aligned}$$

where $t_{j,j} = \mathbf{v}_j^T\mathbf{A}\mathbf{v}_j$, $\mathbf{p}_j = \mathbf{C}_k^T\mathbf{A}\mathbf{v}_j$, $t_{j+1,j} = \|\mathbf{A}\mathbf{v}_j - \mathbf{C}_k\mathbf{p}_j - \mathbf{v}_j t_{j,j} - \mathbf{v}_{j-1}t_{j-1,j}\|_2$, and $t_{j-1,j} = t_{j,j-1}$ was defined in the previous iteration, leading to the augmented Lanczos relation (with $\mathbf{T}$ for tridiagonal)

$$\mathbf{A}\mathbf{V}_j = \mathbf{C}_k\mathbf{P}_j + \mathbf{V}_{j+1}\underline{\mathbf{T}}_j = \mathbf{C}_k\mathbf{P}_j + \mathbf{V}_j\mathbf{T}_j + \mathbf{v}_{j+1}\mathbf{e}_j^T t_{j+1,j}, \quad (5.19)$$

where $\mathbf{P}_j = [\mathbf{p}_1 \;\; \mathbf{p}_2 \;\; \ldots \;\; \mathbf{p}_j]$. Note that the operator used in the Lanczos recurrence, $(\mathbf{I} - \mathbf{C}_k\mathbf{C}_k^T)\mathbf{A}$, is self-adjoint over the space $\textbf{range}(\mathbf{C}_k)^\perp$, which contains the space spanned by the Lanczos vectors, i.e., $(\mathbf{I} - \mathbf{C}_k\mathbf{C}_k^T)\mathbf{A}$ acts like a symmetric matrix on the vectors in the Krylov subspace [53]. The approximate solution in step j of the recycling MINRES algorithm is given by $\mathbf{x}_j = \mathbf{x}_0 + \mathbf{W}_k\mathbf{z} + \mathbf{V}_j\mathbf{y}$, where $\mathbf{y}$ and $\mathbf{z}$ are determined by the minimum residual condition. We have

$$\begin{aligned}\mathbf{r}_j = \mathbf{b} - \mathbf{A}\mathbf{x}_j &= \mathbf{r}_0 - \mathbf{C}_k(\mathbf{R}_k\mathbf{z}) - \mathbf{C}_k\mathbf{P}_j\mathbf{y} - \mathbf{V}_{j+1}\underline{\mathbf{T}}_j\mathbf{y} &(5.20)\\ &= \mathbf{V}_{j+1}\left(\mathbf{e}_1\|\mathbf{r}_0\|_2 - \underline{\mathbf{T}}_j\mathbf{y}\right) - \mathbf{C}_k\left(\mathbf{P}_j\mathbf{y} + \mathbf{R}_k\mathbf{z}\right) &(5.21)\end{aligned}$$

Due to the orthogonality $\mathbf{C}_k^T\mathbf{V}_{j+1} = \mathbf{O}$, the minimization can be done in two separate steps. The first step, solving for $\mathbf{y}$, minimizes the residual over the space $\textbf{range}(\mathbf{V}_{j+1})$, which is independent of the minimization over the space $\textbf{range}(\mathbf{C}_k)$. The minimization over $\textbf{range}(\mathbf{C}_k)$ simply requires that $\mathbf{P}_j\mathbf{y} + \mathbf{R}_k\mathbf{z} = \mathbf{0}$, and hence that $\mathbf{z} = -\mathbf{R}_k^{-1}\mathbf{P}_j\mathbf{y}$. Note that the Lanczos recurrence is the same as for standard MINRES, and we use a similar change of basis to develop a short term recurrence [40]. Using the thin QR decomposition of the tridiagonal matrix $\underline{\mathbf{T}}_j = \underline{\mathbf{Q}}_j^{(j+1)\times j}\mathbf{S}_j^{j\times j}$, which is computed one column at a time, we recursively define $\widetilde{\mathbf{V}}_j$ through $\widetilde{\mathbf{V}}_j\mathbf{S}_j = \mathbf{V}_j$ (which leads to an additional three term recurrence, as for MINRES), set $\widetilde{\mathbf{y}} = \underline{\mathbf{Q}}_j^T\mathbf{e}_1\|\mathbf{r}_0\|_2$, and $\mathbf{z} = -\mathbf{R}_k^{-1}\mathbf{P}_j\mathbf{S}_j^{-1}\widetilde{\mathbf{y}}$. We use the $\widetilde{\mathbf{v}}_j$ vectors to update the solution $\mathbf{x}_j$, so that the $O(n)$ vectors $\mathbf{v}_j$ and $\widetilde{\mathbf{v}}_j$ can be discarded. For efficiency, the solution update $\mathbf{W}_k\mathbf{z}$ can be postponed until after convergence. If the number of iterations is large, $\mathbf{z}$ can be updated recursively using an additional recurrence for the $O(k)$ vectors $\widetilde{\mathbf{P}}_j\mathbf{S}_j = \mathbf{P}_j$. For details, see [62, 38].

Algorithm 15 outlines the Recycling MINRES algorithm that includes the recycle space into the search space. For details on updating the recycle space, see [62, 38].

Algorithm 15 Recycling MINRES

```
1: $\widetilde{\mathbf{r}}_0 = \mathbf{p} - \mathbf{A}\widetilde{\mathbf{x}}_0$
2: $\mathbf{x}_0 = \widetilde{\mathbf{x}}_0$;   $\widehat{\mathbf{p}} = \mathbf{R}_k^{-1}(\mathbf{C}_k^T\widetilde{\mathbf{r}}_0)$;   $\mathbf{r}_0 = \widetilde{\mathbf{r}}_0 - \mathbf{C}_k\mathbf{C}_k^T\widetilde{\mathbf{r}}_0$
3: $\mathbf{v}_1 = \mathbf{r}_0/\|\mathbf{r}_0\|_2$;   $\widetilde{\mathbf{y}} = \mathbf{e}_1\|\mathbf{r}_0\|_2$
4: for $j = 1, 2, \dots$ do
5:     $\widehat{\mathbf{v}} = \mathbf{A}\mathbf{v}_j$
6:     $\widehat{\mathbf{v}} = \widehat{\mathbf{v}} - \mathbf{C}_k(\mathbf{C}_k^T\widehat{\mathbf{v}})$;   $\mathbf{p}_j = \mathbf{R}_k^{-1}(\mathbf{C}_k^T\widehat{\mathbf{v}})$
7:             $\triangleright$ use modified Gram-Schmidt orthogonalization for updating $\widehat{\mathbf{v}}$
8:     $t_{j-1,j} = t_{j,j-1}$;   $\widehat{\mathbf{v}} = \widehat{\mathbf{v}} - \mathbf{v}_{j-1}t_{j-1,j}$
9:     $t_{j,j} = \mathbf{v}_j^T\widehat{\mathbf{v}}$;   $\widehat{\mathbf{v}} = \widehat{\mathbf{v}} - \mathbf{v}_j t_{j,j}$
10:    $t_{j+1,j} = \|\widehat{\mathbf{v}}\|_2$;   $\mathbf{v}_{j+1} = \widehat{\mathbf{v}}/t_{j+1,j}$
11:    $\mathbf{S}_{:,j} = \mathbf{G}_{j-1}\mathbf{G}_{j-2}\underline{\mathbf{T}}_{:,j}$
12:    $\triangleright$ apply the previous two Given's rotations to the new column of $\underline{\mathbf{T}}_j$
13:    Compute Given's rotation $\mathbf{G}_j$ such that $\mathbf{S}_{:,j} = \mathbf{G}_j\mathbf{S}_{:,j}$ has $s_{j+1,j} = 0$.
       $\triangleright$ see MINRES [22, p. 41–44]
14:    $\widetilde{\mathbf{y}} = \mathbf{G}_j\widetilde{\mathbf{y}}$
15:    $\widetilde{\mathbf{v}}_j = s_{j,j}^{-1}(\mathbf{v}_j - \widetilde{\mathbf{v}}_{j-1}s_{j-1,j} - \widetilde{\mathbf{v}}_{j-2}s_{j-2,j})$
16:    $\widetilde{\mathbf{p}}_j = s_{j,j}^{-1}(\mathbf{p}_j - \widetilde{\mathbf{p}}_{j-1}s_{j-1,j} - \widetilde{\mathbf{p}}_{j-2}s_{j-2,j})$
17:    $\mathbf{x}_j = \mathbf{x}_{j-1} + \widetilde{\mathbf{v}}_j\widetilde{y}_j$        $\triangleright$ $\widetilde{y}_j$ is the $j$th entry of vector $\widetilde{\mathbf{y}}$
18:    $\widehat{\mathbf{p}} = \widehat{\mathbf{p}} - \widetilde{\mathbf{p}}_j\widetilde{y}_j$
19:
20:    if $|\widetilde{y}_{j+1}| \le \varepsilon\|\mathbf{r}_0\|_2$ then break
21: end for
22: $\mathbf{x} = \mathbf{x}_j + \mathbf{W}_k\widehat{\mathbf{p}}$
```

While algorithm 15 allows any subspace to be recycled, we focus here on approximate invariant subspaces, in particular, those corresponding to small eigenvalues. First, this leads to substantially improved rates of convergence as the condition number is significantly improved. Second, the small eigenvalues correspond to smooth modes that can be transferred effectively from one iteration of the shape optimization to the next.

5.3 Recycling MINRES for evolving geometries

As already mentioned in the introduction to this chapter, shape optimization is on of the problems where long series of linear systems of equations arise. Shape optimization problems are often governed by partial differential equations, as we have seen in Chapter 3. In Chapter 2, we have seen that through discretization, these partial differential equations lead to large sparse systems of equations. The solution of these PDEs depend on the domain Ω they are defined on and as Ω changes over the course of the optimization procedure, so does the linear system of equations and its solution. Therefore, if the changes in the geometry are only small from one optimization step to the next, we could assume that the changes in the system matrix are only small as well, and Krylov subspace recycling would be applicable. But the changes in the geometry lead to changes in the underlying mesh and thus may prevent a straightforward application of Krylov subspace recycling. Several issues have to be considered. Nodes may have changed their position, which would have an impact on the entries of the corresponding system matrix. New nodes might have been added or others removed and a re-meshing could mean that the system matrix in one optimization step is completely different compared to the next one. Hence, depending on the meshing technique, a mapping of the matrix representing the subspace in one optimization step to the next might be necessary.

Mapping between successive meshes

Let Ω_i and Ω_{i+1} be two domains representing two immediately consecutive shapes stemming from an iterative shape optimization procedure, see Figure 5.1. These domains are discretized by finite element meshes T_i and T_{i+1}, respectively. In general, the meshes feature different connectivities and different numbers of nodes, N_i and N_{i+1}, respectively. Additionally, from opti-

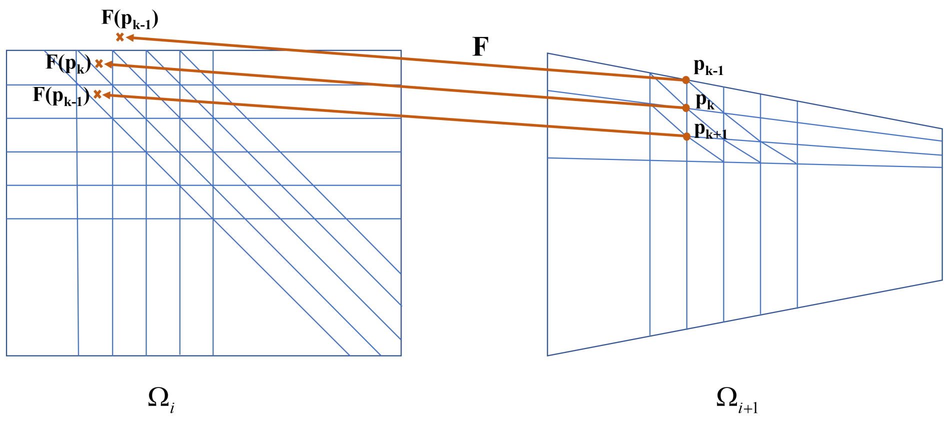


Figure 5.1: Concept of the mapping of the grid points to the domain of the previous optimization step. From [7].

mization step i, an approximate invariant subspace is given by $\mathbf{range}(\mathbf{W}_k^{(i)})$ with $\mathbf{W}_k^{(i)} \in \mathbb{R}^{N_i \times k}$, that is supposed to be recycled in optimization step $i+1$. The system in this step however is of dimension N_{i+1}. Therefore, we are in need of a function that maps the $N_i \times k$ - dimensional matrix representing the approximate invariant subspace of the system in optimization step i to a $N_{i+1} \times k$ - dimensional matrix, that hopefully still represents a good approximate invariant subspace of the linear system in optimization step $i+1$.

The mapping we propose exploits the fact that the algebraic systems we consider are closely linked to the continuous finite element spaces $\mathcal{V}_h(\Omega_i, \mathcal{T}_i)$ and $\mathcal{V}_h(\Omega_{i+1}, \mathcal{T}_{i+1})$, determined by $\mathcal{T}_i$ and $\mathcal{T}_{i+1}$, respectively. Recall the Galerkin method from Chapter 2. The continuous equation

$$a(u, v) = \langle f, v \rangle, \quad \text{for all } v \in H_0^1(\Omega) \tag{5.22}$$

is defined on the finite dimensional subspace $V_h(\Omega)$ with nodal basis functions $\Phi_1, \ldots, \Phi_N$ of the subspace such that (5.22) is equivalent to

$$a(u_h, \Phi_j) = \langle f, \Phi_j \rangle, \quad j = 1, \ldots, N, \tag{5.23}$$

with $u_h \in V_h(\Omega)$. With $u_h = \sum_{j=1}^{N} u_j \Phi_j$ formulated in terms of the basis functions, we obtain the system of equations

$$\sum_{k=1}^{N} a(\Phi_k, \Phi_j)u_j = \langle f, \Phi_j \rangle, \quad j = 1, \dots, N, \tag{5.24}$$

or in matrix notation $\mathbf{Ax} = \mathbf{b}$. Now, consider the approximate invariant subspace $\mathbf{range}(\mathbf{W}_k^{(i)})$ with $\mathbf{W}_k^{(i)} \in \mathbb{R}^{N_i \times k}$. Instead of considering the coefficients of the matrix in the algebraic sense, we can see the columns of the matrix as vectors containing the coefficients of continuous functions defined on the finite element space $\mathcal{V}_h(\Omega_i, \mathcal{T}_i)$. These functions are defined as

$$w_m^{(i)}(x) := \sum_{j=1}^{N_i} (\mathbf{W}_k^{(i)})_{(j,m)} \Phi_j^{(i)}(x), \quad m = 1, \dots, k, \quad x \in \Omega_i, \tag{5.25}$$

$\Phi_j^{(i)}(x)$ being the nodal basis functions of the finite element space $\mathcal{V}_h(\Omega_i, \mathcal{T}_i)$. If we are able to find a transformation $F(x^{(i+1)}) : \Omega_{i+1} \to \Omega_i$, that maps each node p_l in $\mathcal{T}_{i+1}$, $l = 1, \dots, N_{i+1}$ to a corresponding point x_l in Ω_i, we can build the matrix representing the mapped approximate invariant subspace, $\widetilde{\mathbf{W}}_k^{(i)} \in \mathbb{R}^{N_{i+1} \times k}$, in the following way:

$$(\widetilde{\mathbf{W}}_k^{(i)})_{(l,m)} := w_m^{(i)}(F(p_l)). \tag{5.26}$$

In summary, this method consists of two steps: First, we define a map from the nodes of the new mesh to points in the old domain (see Fig. 5.1); and second, we interpolate the values of the matrix $\mathbf{W}_k^{(i)}$ via the functions defined in (5.25) in the finite element space and evaluate these functions at the points in Ω_i corresponding to the nodes in of $\mathcal{T}_{i+1}$.

Although the second step is straightforward, the first step can be challenging, depending on the meshing technique that is used. Therefore, we briefly discuss general concepts to realize such a mapping for some of the meshing techniques discussed in Chapter 4. If the update of the shape in the optimization procedure is performed via mesh morphing, each node in $\mathcal{T}_{i+1}$ can be mapped uniquely to the corresponding node in $\mathcal{T}_i$, hence the approximate invariant subspace can be recycled without being transformed. Techniques that work with a reference frame come with an inherent mapping from the reference frame to the domain. Through this reference frame, we can map each node in $\mathcal{T}_{i+1}$ to a node in $\mathcal{T}_i$ and thus recycle the approximate invariant subspace as in the mesh morphing case. For meshing techniques that do not have such an inherent mapping the task becomes more difficult. If we assume, for example, a general re-meshing scheme without a prescribed number of nodes or restrictions on the connectivity, then the optimization procedure does not provide any information about the relation between the two domains Ω_i and Ω_{i+1}; see Figure 5.1. Additionally, the two meshes generally may differ in the number of nodes and thus basis functions. Therefore, a more sophisticated mapping of the approximate invariant subspace of the linear system matrix derived from mesh $\mathcal{T}_i$ to an approximate invariant subspace for the system matrix derived from mesh $\mathcal{T}_{i+1}$ becomes inevitable. If no other information is available, the simplest map from Ω_{i+1} to Ω_i is $F(x) := x$. This choice, however, does not guarantee that $F[x] \in \Omega_i$. Therefore, we suggest to choose $F(x)$ as the minimizer, in a suitable norm, of the distance between the given point x and points in Ω_i, i.e., $F(x) := \arg\min_{\tilde{x}_i \in \Omega_i} ||\tilde{x}_i - x||$, which implies the identity for $x \in \Omega_i$. Another approach would be to extrapolate the $w_m^{(i)}(\cdot)$ in a simple way. We discuss this idea in more detail in section 6.4. A special case of re-meshing is mesh refinement (and coarsening), especially adaptive mesh refinement (AMR) [6]. Although in this case the two systems will certainly differ in dimension, the new nodes will definitely lie inside Ω_i. Additionally, for the new nodes the global evaluation in (5.25) reduces to a local evaluation on the respective refined element; a mapping of approximate invariant subspaces between AMR meshes is described in [61]. The discussed approach results in two obvious challenges: The first is already addressed at the end of section 5.2. To reduce the condition number of a system, it sometimes can be desirable to remove the absolute largest eigenvalues rather than the smallest ones or both the smallest and the largest. Yet, the large eigenvalues often correspond to high frequency modes which are clearly very difficult to approximate by the presented approach. We therefore restrict ourselves to remove small eigenvalues, which generally is a good choice. The second challenge is that small changes in the geometry do not necessarily result in small changes in the eigenvectors of the system matrix. In such cases, the approximate invariant subspace of the system matrix for the new geometry and mesh, computed by the mapping described above, is not sufficiently accurate that it leads to a fast convergence rate. This issue is addressed in [7].

5.3.1 Mapping on structured meshes

In Section 4.2.1 we introduced a structured meshing approach. It turns out that the specific nature of this approach facilitates the construction of a suitable mapping F. This is why, as a proof of concept, we introduce a quite simple mapping adjusted for this specific meshing technique, which still leads to a considerable speed up in many of our test cases.
Consider the linear systems $\mathbf{A}(\rho^{(i)})\mathbf{x}^{(i)} = \mathbf{b}^{(i)}$ and $\mathbf{A}(\rho^{(i+1)})\mathbf{x}^{(i+1)} = \mathbf{b}^{(i+1)}$ and $\mathbf{W}^{(i)} \in \mathbb{R}^{N_a^{(i)} \times k}$, with $\mathbf{range}(\mathbf{W}^{(i)})$ approximating the invariant subspace corresponding to the k smallest eigenvalues of $\mathbf{A}(\rho^{(i)})$. As in this meshing approach the initial connectivity of the mesh is kept for all optimization steps, we can uniquely identify each node in iteration $(i+1)$ with a node in iteration (i). We distinguish between three cases to determine the matrix $\widetilde{\mathbf{W}}^{(i)}$ representing the mapped approximate invariant subspace : (1) Matrix entries corresponding to inner nodes that stay inner nodes are kept; (2) matrix entries corresponding to nodes that change from inner node to boundary node or vice versa are recalculated according to (5.25); and (3) matrix entries corresponding to nodes that change from inactive to active are calculated in the following way. Assuming only small changes in the geometry, these nodes must be boundary nodes or close to boundary nodes. It is therefore likely that the values of former active nodes in their neighborhood provide a better approximation than a value resulting from interpolating former active and inactive nodes. We therefore propose for these points the following extrapolation: Consider that the status of node $\tilde{x}_\ell^{(i+1)}$ changes from inactive to active. From the set of nodes that share a finite element, we consider only the subset of nodes that were active in iteration (i). For each active node, $x_\ell^{(i+1)}$ we have such a set $S_a^{(i,\ell)}$. We set the entries of $\widetilde{\mathbf{W}}^{(i)}$ corresponding to

these nodes to the weighted mean of the values at $S_a^{(i,\ell)}$, given by

$$(\widetilde{\mathbf{W}}^{(i)})_{(\ell,j)} = \frac{1}{|S_a^{(i,\ell)}| - 1} \sum_{s \in S_a^{(i,\ell)}} \frac{\sum_m ||s_m - x_\ell^{(i+1)}||_2 - ||s - x_\ell^{(i+1)}||_2}{\sum_m ||s_m - x_\ell^{(i+1)}||_2} (\mathbf{W}^{\infty,(i)})_{(s,j)}, \tag{5.27}$$

with weights corresponding to the distance between the neighbors' location on the old grid and $x_\ell^{(i+1)}$. The numerical results will be presented in Section 6.4.

Chapter 6

Implementation and numerical results

In this chapter, we combine the results of the previous chapters. In the first section, we provide details concerning the implementation of the finite element method on the structured meshes introduced in Chapter 4. This is followed by the description of the implementation of the objective functionals in two dimensions introduced in Chapter 3. Here, we will also look at some results from the corresponding shape optimization procedure. After this, we will see some results for the three dimensional case and finally, we will discuss results of the implementation of the Krylov subspace method introduced in Chapter 5.

6.1 Details of the implementation

The implementation of the solver on the structured grid follows the principles of implementation of the finite element method. We will describe the approach in the next section for the assembly of the linear system of equations for the linear elasticity equation in detail and transfer the procedure to the implementation of the objective functionals and their derivatives in the sections that follow.

6.1.1 Assembling the linear system of equations

The linear elasticity equation is discretized via the finite element method as we have already seen in Section 2.2.2. To build the system matrix, the natural approach would be to calculate the entries node by node. That would result in two for-loops in the assembly and thus would lead to N^2 calculations, when N is the number of active nodes in $\mathcal{T}$. Lets recall the discretization of the bilinear form (2.85).

$$\begin{aligned} L(u,v) \approx \lambda \sum_{T\in\mathcal{T}_h} \sum_{l=1}^{q_l^T} \hat{\omega}_l^T \det\,(\hat{\nabla} B_T(\hat{\xi}_l))\, \nabla \cdot u(B_T(\hat{\xi}_l))\, \nabla \cdot v(B_T(\hat{\xi}_l)) \\ + 2\mu \sum_{T\in\mathcal{T}_h} \sum_{l=1}^{q_l^T} \hat{\omega}_l^T \det\,(\hat{\nabla} B_T(\hat{\xi}_l)) \varepsilon(u(B_T(\hat{\xi}_l))) : \varepsilon(v(B_T(\hat{\xi}_l))). \end{aligned} \tag{6.1}$$

The discertization suggest a different approach which is element oriented [12]. Instead of calculating every entry for every pair of nodes, as the discretization suggests for every element $T \in \mathcal{T}_h$ we additively calculate the local $s \times s$-system matrix, where s is the local degree of freedom. This leads to $N^{el} \cdot s^2$ calculations needed. Hence, this approach is linear in the number of elements and quadratic only in the local degrees of freedom. Additionally to that, it is important to consider that the system matrix is sparse. In each row, there will only be entries in those columns corresponding to nodes that are put on the same elements as the considered node. Hence, instead of assembling the full matrix, it is more effective to assemble an array of stencils, that is vectors containing only the non-zero coefficients per row, which can then efficiently be transformed into a sparse matrix or can be used directly as a matrix operator in a sparse solver. For a problem lying in $\mathbb{R}^d$ and with P_1 elements, that is the degrees of freedom lie on the nodes of the vertices of the mesh, the general assembly scheme for the Poisson matrix is described in Algorithm 16. The algorithm to build the system matrix for the linear elasticity equation works similar but is less comprehensible as it represents a bilinear form.

Algorithm 16 General assembly scheme

```
initialize array $\mathbf{L}_s \in \mathbb{R}^{\tilde{N}^{no} \times 7}$ with 0
for $T_k \in \mathcal{T}$ do
    $I_{loc}$ =connectivity$(k)$
    $\ell = L(T_k)$
    for $j = 1, 2, 3$ do
        Determine neighbor indices $j_1, j_2, j_3 \in 1, \cdots, 7$
        $\mathbf{L}_s[I_{loc}[j]][(j_1, j_2, j_3)]$ += $\ell[j][]$
    end for
end for
```

We initialize a tensor $\mathbf{L}$ on the nodes of $\widetilde{\mathcal{T}}(\Omega)$ with zero entries and then let the for-loop run only over the finite elements of $\mathcal{T}(\Omega)$. The connectivity, which is defined for $\mathcal{T}^\infty(\Omega)$ will link the cell of the element in $\mathcal{T}$ with the indices of the nodes of $\mathcal{T}^\infty(\Omega)$. This gives an array of stencils that is defined on $\mathcal{T}^\infty(\Omega)$, with zeros in the entries corresponding to nodes that lie outside of $\mathcal{T}$. Therefore, when building the operator from the stencils at the same time we need to project the data onto $\mathcal{T}$. Exemplary, we give a scheme to build a sparse matrix from the array of stencils in Algorithm 17. For simplicity we assume zero-Dirichlet-boundary conditions.

Algorithm 17 Project global matrix onto $\mathcal{T}$

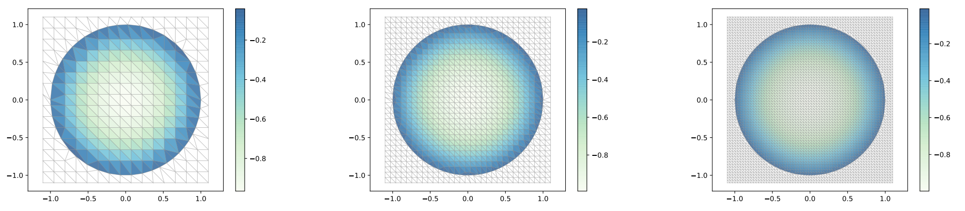
```
initialize list $rows$
initialize list $columns$
initialize list $data$
for $i$ in $active - nodes$ do
    append$(rows,$ repeat$(i, 7))$
    append$(columns,$ neighbors$(i))$
    append$(data, \mathbf{L}_s[i])$
end for
$I_d$ = Dirichlet-boundary nodes
$I_0$ = find(node-status) == 0
$I_r = I_d \cup I_0$
Build sparse matrix $\mathbf{L}_{sp}(rows, columns, data)$
remove rows and columns $I_r$: $\mathbf{L}_r = \mathbf{L}_{sp} \backslash \mathbf{L}[I_r][I_r]$
```

As the cost of assembling the system matrix is quadratic in s, higher order finite elements are in general avoided. Another advantage of this element-wise approach is the reference element. Recall for example the discretization of the gradient of the displacement $u(\xi) = \sum_{m=1}^{s} u_m \hat{\theta}_m \circ B_T^{-1}(\xi)$. Like every other function, it can be computed via the shape functions on the reference element as can be their derivatives and then be transformed to the values of the actual local element. These shape functions only have to be computed once. Therefore, the only thing that has to be computed for every element is the local transformation from reference element to local element. This highlights an additional great advantage of structured meshes. The local values of the system matrix on element T_k do not depend on its global location, but only on the shape of the considered element, that is the length of the edges and the angles. Since all the inner elements are of the same size and shape, for those elements we need to calculate these values only once and assign the result to all inner elements. This changes the assembly of the system matrix to the one in Algorithm 18.

Algorithm 18 Assembly scheme for structured meshes

```
initialize array $\mathbf{L}_s \in \mathbb{R}^{\tilde{N}^{no} \times 7}$ with 0
$\ell_{inner} = L(T_{inner})$, where $T_{inner}$ is a random inner element
for $T_k \in \mathcal{T}$ do
    if element-status$(k)! = 0$ then
        if element-status$(k) == -1$ then
            $\ell = \ell_{inner}$
        else
            $I_{loc}$ =connectivity$(k)$
            $\ell = L(T_k)$
        end if
        for $j = 1, 2, 3$ do
            Determine neighbor indices $j_1, j_2, j_3 \in 1, \cdots, 7$
            $\mathbf{L}_s[I_{loc}[j]][(j_1, j_2, j_3)]$ += $\ell[j][]$
        end for
    end if
end for
```

The right hand side of the equation is computed in a similar way. We initialize a vector $\mathbf{b} \in \mathbb{R}^{\tilde{N}^{no}}$ and we run a for-loop over all $T \in \mathcal{T}^{\infty}$. Then we



(a) $N_x = N_y = 16$ (b) $N_x = N_y = 32$ (c) $N_x = N_y = 64$

Figure 6.1: Convergence of the solution of the Poisson equation on the unit circle.

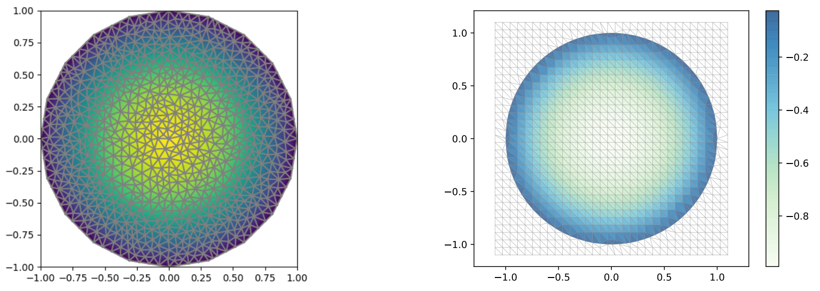
project the vector $\mathbf{b}$ from $\mathcal{T}^\infty$ onto $\mathcal{T}$, see Algorithm 19.

Algorithm 19 Assembly of the right hand side

```
initialize $\mathbf{b} := \tilde{N}^{no}$
for $T \in \mathcal{T}$ do
    if element-status$(k)! = 0$ then
        $I_{loc} =$connectivity$(T)$
        $\mathbf{b}[I_{loc}] \mathrel{+}= b(x_{I_{loc}})$
    end if
end for
$I_d =$ Dirichlet-boundary nodes
$I_0 =$ find(node-status $== 0$)
$I_r = I_d \cup I_0$
remove rows $I_r$: $\mathbf{b}_r = \mathbf{b} \backslash \mathbf{b}[I_r]$
```

6.1.2 Simple test case

As a first test, we compute the solution of the Poisson equation on the unit circle. The code is implemented as a Python3 project [60] with interpreter version 3.6.2. For this purpose, we define the Dirichlet-boundary conditions for the equation with the true solution $u(x, y) = x^2 + y^2 - 1$, so that we are able to compare the computed solution with the true solution. We compute the solutions on grids with $N = 16$, $N = 32$, $N = 64$ as visualized in Figure 6.1, as well as for $N = 128$, which we do not visualize here because the grid is too fine to be resolved properly. For each mesh size we calculate the L_2-norm



(a) FEniCS triangulation (b) Structured triangulation

Figure 6.2: Comparison of the solution of the Poisson equation on the presented meshing approach to a standard FEniCS triangulation.

of the error, as displayed in Table 6.1.

N	16	32	64	128
L_2-norm	1.7718×10^{-3}	2.5176×10^{-4}	3.4303×10^{-5}	9.1292×10^{-6}

Table 6.1: L_2-norm of the error of the Poisson equation for different mesh sizes

Subsequently, we compare our solution for $N_x = N_y = 32$ with the solution of the given problem solved with FEniCS [1] for Python. The unit circle is discretized via a standard Delaunay triangulation [14]. As it is not possible to prescribe the exact number of discretization points, we specify the number of nodes that resolve the boundary and aim for a similar amount of nodes as in our discretization. Both solutions are visualized in Figure 6.2. We compare the error of the solutions in the L_2-norm and in the L_∞-norm, see Table 6.2. We can see that the error that arises from the structured meshing approach is comparable in both norms to the one that arises when solving on the FEniCS-mesh.

6.1.3 Updating the mesh

In this section, we introduce a model problem that we will refer to in other sections as well. We consider a bent rod, see Figure 6.3, that is supposed to be made out of ceramic material. It has diameter $0.1m$ and length $0.6m$

	FEniCS	**CFE**
boundary nodes	102	102
vertices	688	711 ($N = 32$)
L2	$4.203563201 \times 10^{-4}$	$2.517583282 \times 10^{-4}$
Max	$1.633036056 \times 10^{-3}$	$9.930186560 \times 10^{-3}$

Table 6.2: Comparison of error norms for the solution of the Poisson equation.

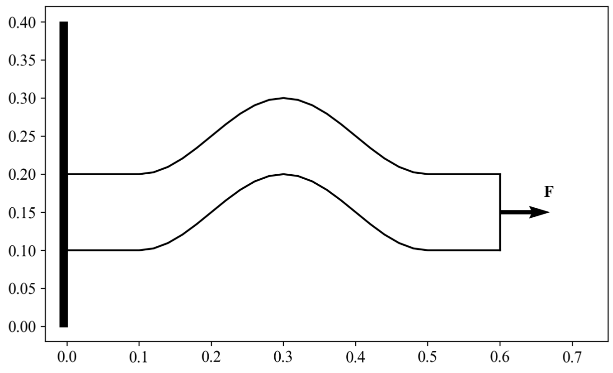


Figure 6.3: The standard testing geometry. From [7].

and is bent in the middle. It is clamped at the left part of the boundary, i.e. it is the part of the boundary $\partial\Omega_D$ where zero-boundary conditions hold. On the right side of the boundary we apply tensile load, here that is surface force in the positive direction of the x-axis. This is the part of the Neumann-boundary that is not to be changed in the optimization $\partial\Omega_{N_{fixed}}$. The rest of the boundary has implicit zero-Neumann-boundary condition and is the part of the boundary that can be deformed in the optimization $\partial\Omega_{N_{free}}$. We use this geometry for the obvious reason that we know what the optimal shape is supposed to look like. The most robust geometry in this setting is a straight rod. That makes it easier to control and validate the optimization. To design the model problem more interesting we rotate the domain by $\alpha = 0.2$. We set $\widetilde{\Omega}$ to be the rectangle $[-0.08, 0.62] \times [0.05, 0.4]$ and discretize with $N_x = 61$ and $N_y = 31$, see Figure 6.4. As a first example, we generate a sequence of shapes, starting with the one visualized in Figure 6.4 and ending with a

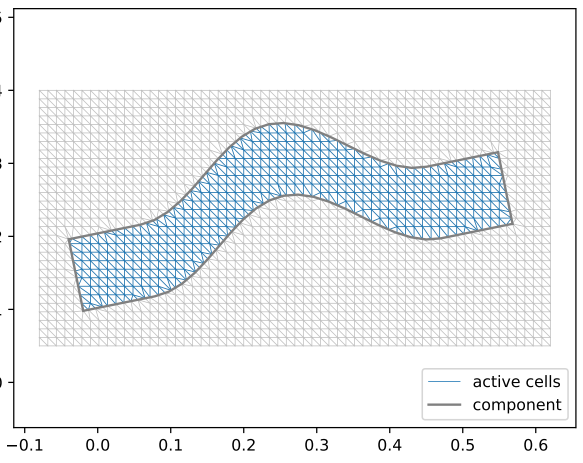


Figure 6.4: Mesh for the standard testing geometry.

straight rod, while keeping the left and right part of the boundary fixed. We restrict the changes in the boundary in each step such that the step length does not exceed the given h. We update the mesh according to Algorithm 8. The intermediate shapes and the final shape with their corresponding meshes are visualized in Figure 6.5.

Updating the system matrix

As already hinted in Section 4.2.3, one of the major advantages of the presented structured mesh generation method is the assembly of the system matrix when we update the mesh. As only nodes in a restricted neighborhood of the boundary change their location and the connectivity is kept, it is possible to preserve the system matrix from the previous iteration and only update those entries effected by the change of location. This can be done, for example, in the following way. Assume that in Algorithm 18, we have stored the local parts of the system matrix ℓ in an $\widetilde{N}^{el} \times s \times s$-array $\mathbf{L}_a$. Further assume that in Algorithm 8, we have stored the indices of all elements that we have visited in the algorithm, which are highlighted in Figure 6.5, in I_v^{el}. By design, this includes at least all elements that have been inner boundary

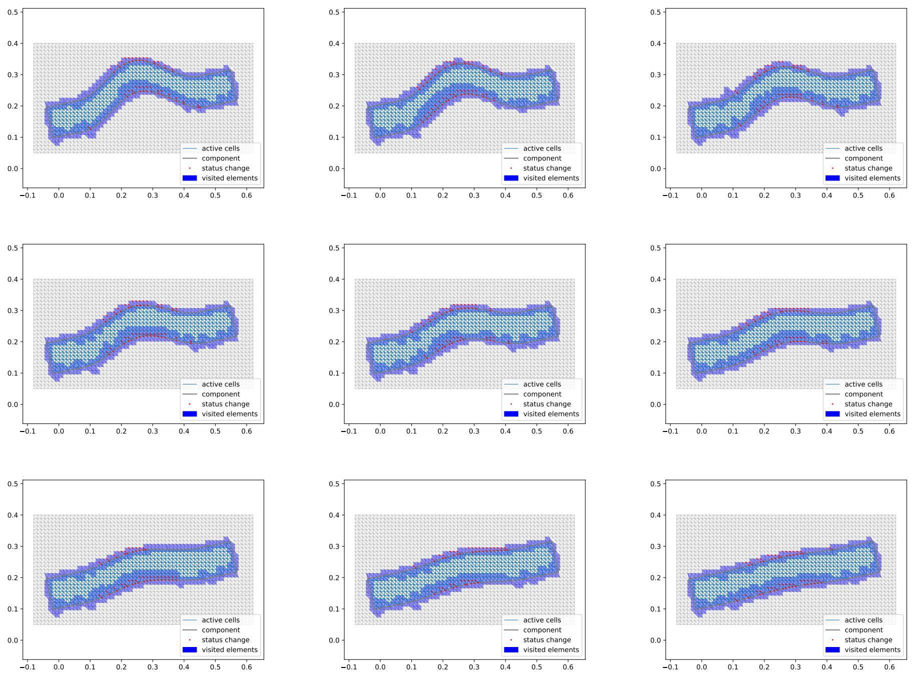


Figure 6.5: Updating the shape in a controlled manor, with step size bounded by h. Elements that are visited in the updating algorithm are highlighted, nodes that change status are marked.

elements in the previous optimization step and all elements that are boundary elements in the present optimization step. Under the assumption that we made in Section 4.2.3, that is that the step length in the optimization is bounded by h, it follows that I_v^{el} contains all elements where nodes might have changed their location. With this, we can update the system matrix via a for-loop over all elements in I_v^{el}, calculate the local system matrix anew, add its values to the respective entries in the global system matrix and subtract them by the respective entries stored in $\mathbf{L}_a$, see Algorithm 20. This leads to a comparable cost in the computation of the local entries on the initial grid, as we already only have to explicitly compute the local stiffness matrices for boundary elements, but it saves the cost of the full assembly.

Algorithm 20 Update of the system matrix

```
project $\mathbf{L}_r$ back to $\mathbf{L}_{sp}$
for $k$ in $I_v^{el}$ do
    $\ell := -\mathbf{L}_a$
    $I_{loc}$ =connectivity($k$)
    if element-status[$k$]! = 0 then
        $\ell$ += $L(T_k)$
    end if
    $\mathbf{L}_{sp}[I_{loc}][I_{loc}] = \ell$
end for
$I_d$ = Dirichlet-boundary nodes
$I_0$ = find(node-status) == 0
$I_r = I_d \cup I_0$
remove rows and columns $I_r$: $\mathbf{L}_r = \mathbf{L}_{sp} \backslash \mathbf{L}[I_r][I_r]$
```

6.2 The objective functionals

The implementation of the objective functionals and their derivatives follows the same principals that we have already applied in Section 6.1.1 when assembling the linear system. As they are similar for both functionals, we will not describe the implementation of the objective functional and its adjoint for the two of them in full detail. Instead we describe how to compute the LCF-functional followed by the implementations of the adjoint of the objective functional for ceramic material.

6.2.1 The functional for low cycle fatigue

As we already stated in Section 3.2.2, in this section we will go into more detail about how to derive the probability of failure for LCF, as it is not as straight forward as is the one for ceramic material under tensile load. The implementation is entirely based on the work done in [46]. In this PhD thesis, a flow chart is provided, which contributes to the understanding of the implementation, thus we reproduce it here with minor changes, see Figure 6.6.

Reconsider (3.19) from Section 3.2.2. The probability of failure solely depends on the deterministic life cycle prediction N_{det} which itself depends on the material parameters and the solution of the linear elasticity equation u on Ω. Therefore, in the following section we will describe how to obtain the value of N_{det}, assuming that the functional is discretized via finite elements analoguous to the discretization done in Section 3.4. N_{det} is the solution of the Coffin-Manson-Basquin (CMB) equation, which we can rewrite as a function of $x := N_{det}$

$$CMB : \mathbb{R}^+ \longrightarrow \mathbb{R}, \quad CMB(x) = \frac{\sigma_f^{'}}{E}(2x)^b + \varepsilon_f^{'}(2x)^c. \tag{6.2}$$

Hence, if we know the value of CMB, which is the strain ε, we can obtain N_{det} by inverting the given function. Since ε here is the elastic-plastic strain, we use the Ramberg-Osgood (RO) equation

$$\varepsilon^{el-pl} = \frac{\sigma^{el-pl}}{E} + \frac{\sigma^{el-pl}}{K}^{1/n}, \tag{6.3}$$

and the method of Neuber shake down [39] to derive ε^{el-pl} from the purely elastic stress $\sigma(u)$ stemming from the linear elasticity equation. We write ε^{el-pl} to emphasize the difference to the elastic strain that we encountered earlier. The coefficients E, K and n are the material parameters Young's modulus, strain hardening coefficient and strain hardening exponent, respectively. The Ramberg-Osgood equation, that we can rewrite as a function, here of $x := \sigma^{el-pl}$, as well

$$RO : \mathbb{R}^+ \longrightarrow \mathbb{R}, \quad RO(x) = \frac{x}{E} + \frac{x}{K}^{1/n}, \tag{6.4}$$

gives the dependency of the elastic-plastic strain and the elastic-plastic stress, while the Neuber shake-down

$$\frac{\sigma^{vM^2}}{E} = \frac{\sigma^{el-pl^2}}{E} + \sigma^{el-pl}\frac{\sigma^{el-pl}}{K}^{1/n} \tag{6.5}$$

provides a way to derive the elastic-plastic stress from the elastic van Mises stress only. Rewriting this again as a function of the elastic-plastic stress, we obtain N_{det} from the chain of functions

$$N_{det} = CMD^{-1} \circ RO \circ SD^{-1}(\frac{\sigma^{el^2}}{E}). \tag{6.6}$$

Since the van Mises stress σ^{vM} is fully determined by the material parameters and the solution u of the linear elasticity equation, we now have the means to determine N_{det} as visualized in Figure 6.6. We obtain the inverse functions in the following way. We compute the values of the respective functions for all values that can possibly be obtained, that is we create a vector of values from zero to some value depending on Young's modulus,

$$\boldsymbol{\sigma}^{el-pl} = \mathsf{linspace}(0, \sqrt{E/10}, 1000)^2.$$

The squaring is a technicality to resolve the function better for values closer to the origin. Then we compute the respective values of the functional, here the Neuber shake down and store them in a second vector,

$$\boldsymbol{\sigma}^{vM} = \mathsf{SD}(\boldsymbol{\sigma}^{el-pl}) := \boldsymbol{\sigma}^{el-pl^2}/E + \boldsymbol{\sigma}^{el-pl}\boldsymbol{\sigma}^{el-pl}/K^{1/n}.$$

And finally, create a spline function with the values of $\boldsymbol{\sigma}^{vM}$ as sampling points and the values of $\boldsymbol{\sigma}^{el-pl}$ as values. Thereby, we have created the inverse function of the Neuber shake down,

$$\mathsf{SD}^{-1} = \mathsf{CubicSpline}(\boldsymbol{\sigma}^{vM}, \boldsymbol{\sigma}^{el-pl}).$$

For the inverse of the CMB function, we can proceed in the same way. The rest of the LCF functional is computed in the same way as the functional for ceramic material, i.e. the integral is approximated via Gauss-quadrature and the whole functional is discretized via finite elements.

6.2.2 Ceramic material

The computation of the derivatives follows the same principles as the assembly of the system matrix earlier in this chapter. We already calculated the

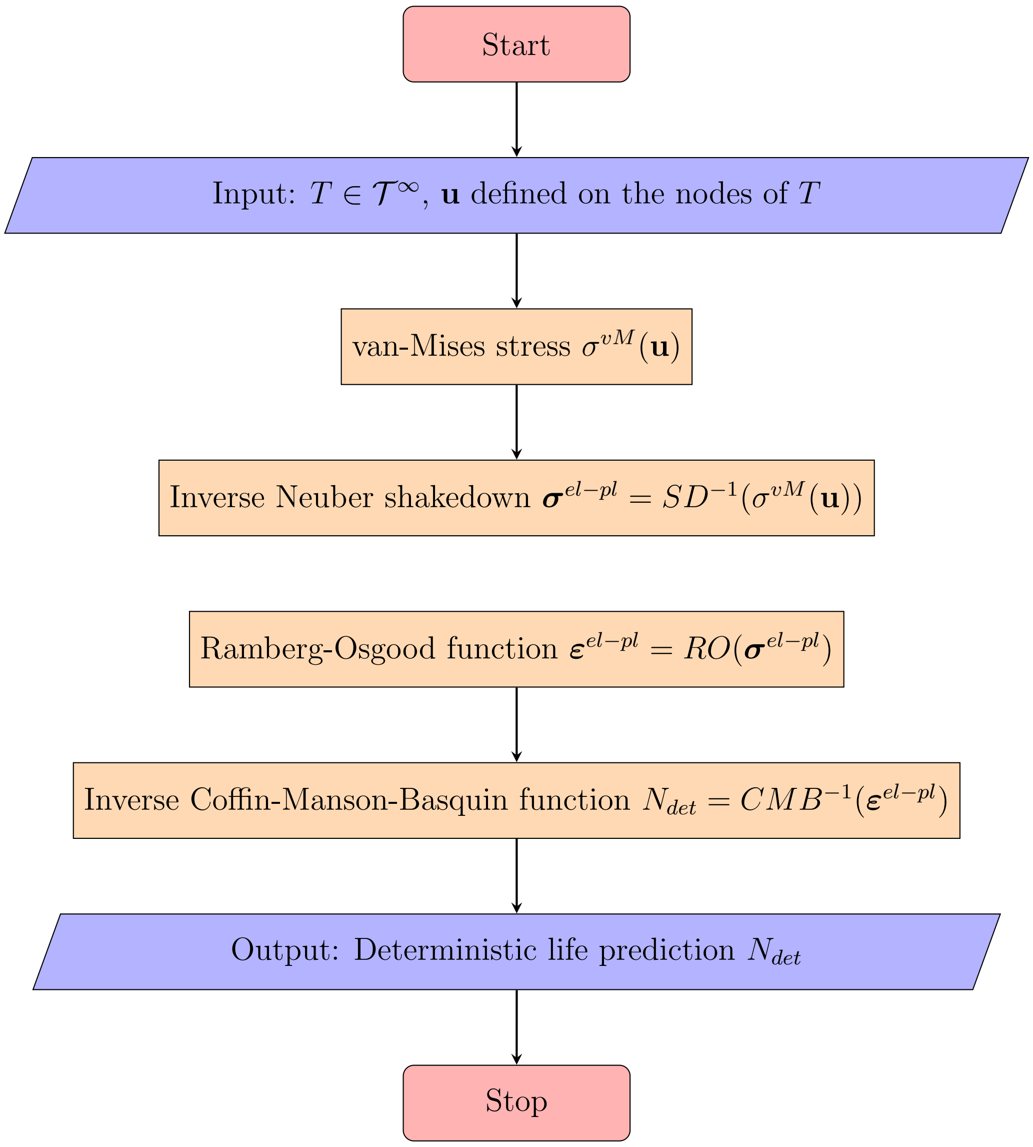

Start
Input: $T \in \mathcal{T}^{\infty}$, $\mathbf{u}$ defined on the nodes of T
van-Mises stress $\sigma^{vM}(\mathbf{u})$
Inverse Neuber shakedown $\boldsymbol{\sigma}^{el-pl} = SD^{-1}(\sigma^{vM}(\mathbf{u}))$
Ramberg-Osgood function $\boldsymbol{\varepsilon}^{el-pl} = RO(\boldsymbol{\sigma}^{el-pl})$
Inverse Coffin-Manson-Basquin function $N_{det} = CMB^{-1}(\boldsymbol{\varepsilon}^{el-pl})$
Output: Deterministic life prediction N_{det}
Stop

Figure 6.6: Computation of the probability of failure for LCF.

local finite element discretization of the derivatives of the discrete adjoint in Section 3.3.1. Therefore, in this section we will only describe the global assembly of the derivatives and refer the reader for details to said Section 3.3.1. The process is visualized in the flow chart in Figure 6.7. First we assemble the linear elasticity equation as described in Section 6.1.1 and solve the system, at least in two dimensions, with a direct sparse solver. The system is solved on the active mesh $\mathcal{T}^{\infty}$, hence the solution u is defined on the active mesh $\mathcal{T}$ as well. For the further assembly, this solution is projected back on the whole grid $\mathcal{T}^{\infty}$. With this we can compute the partial derivatives $\partial J/\partial U$, whose assembly we provide exemplary for all assemblies in Algorithm 21.

Algorithm 21 Assembly of $\partial J/\partial U$

```
  **Input** mesh $\mathcal{T}^{\infty}$, array $U \in \mathbb{R}^{\tilde{N}^{no} \times d}$ solution of linear elasticity equation
Initialize array$J_{\partial u} \in \mathbb{R}^{\tilde{N}^{no} \times d}$ with zeros
Compute $\hat{\mathbf{q}}_p, \hat{\mathbf{q}}_w$ quadrature points and weights on $\hat{T}_{ref}$
Compute $\nabla\hat{\theta}(\hat{\mathbf{q}}_p)$ values of the derivatives of the shape functions
for $k = 1, \cdots, \tilde{N}^{el}$ do
    if mesh.element-status$[k]! = 0$ then
        $I_{loc}$ =mesh.connectivity$(k)$
        $x_{loc}$ = mesh.coordinates$[I_{loc}]$
        $u_{loc} = U[I_{loc}]$
        $J_{\partial u}[I_{loc}]$ += $J^{loc}_{\partial u}(u_{loc}, x_{loc}, \hat{\mathbf{q}}_p, \hat{\mathbf{q}}_w, \nabla\hat{\theta})$
    end if
end for
$I_d$ = Dirichlet-boundary nodes
$I_0$ = find(mesh.node-status) == 0
$I_r = I_d \cup I_0$
remove rows $I_r$: $J^a_{\partial u} = J_{\partial u} \backslash J_{\partial u}[I_r]$
```

As the partial derivative is the right hand side for the adjoint equation, we once more have to project the result of the assembly onto the mesh $\mathcal{T}$. The system matrix of the adjoint equation is the system matrix $\mathbf{L}$ of the governing linear elasticity equation and is solved in the same way. Now, the partial derivatives $\frac{\partial J}{\partial X}$, $\frac{\partial F}{\partial X}$ and $\frac{\partial B}{\partial X}U$ are assembled like the derivative $\partial J/\partial U$, which then form the material derivative dJ/dX.

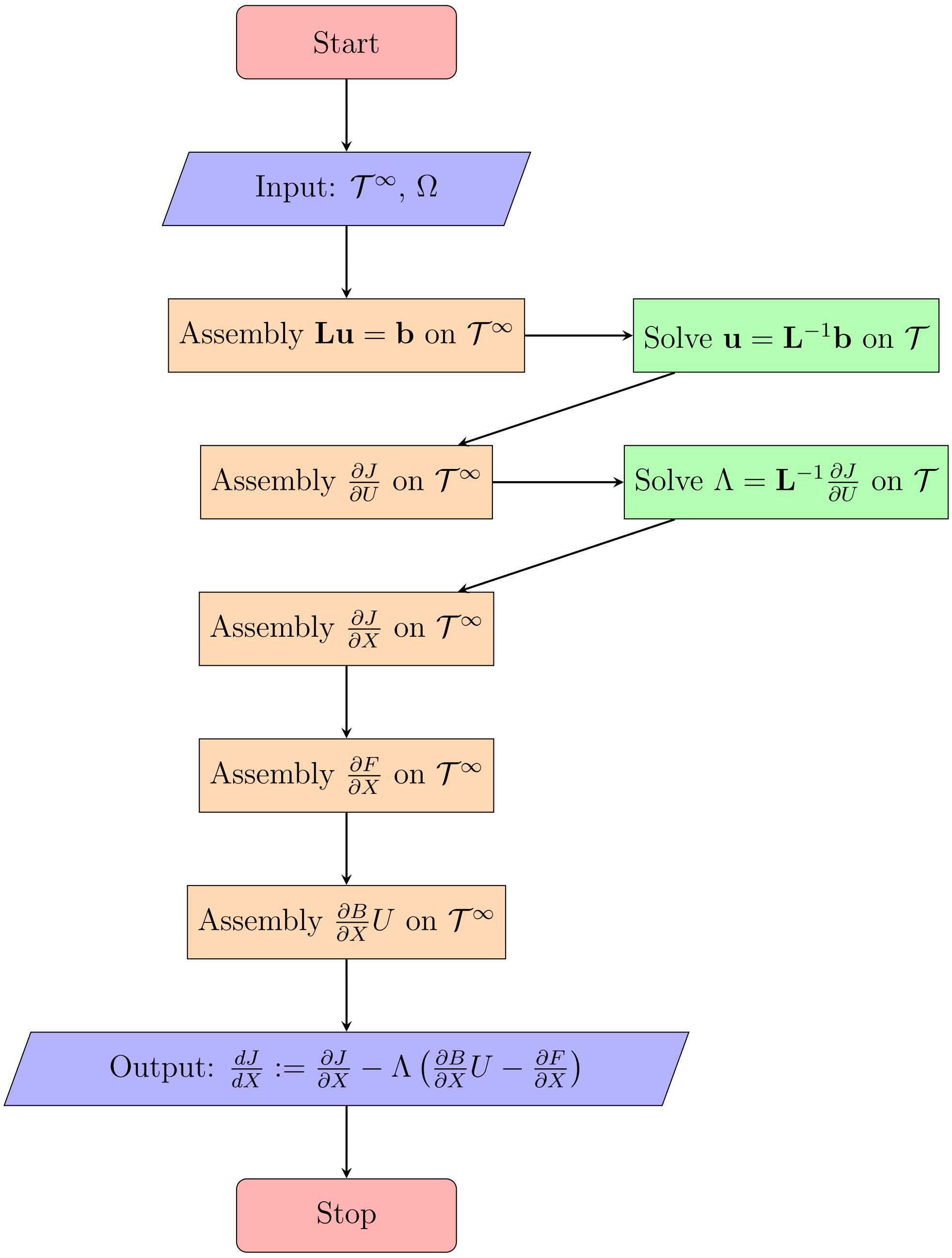
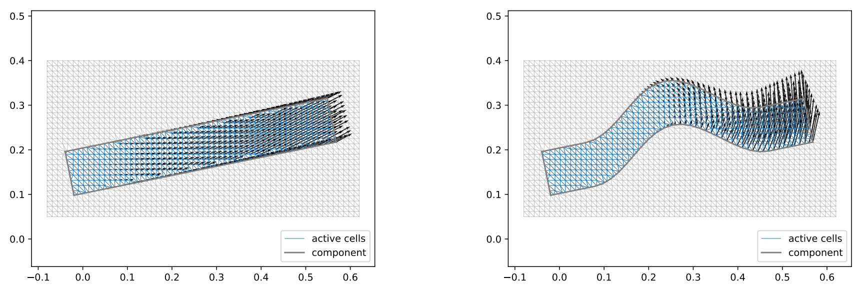

Start
Input: $\mathcal{T}^\infty$, Ω
Assembly $\mathbf{Lu} = \mathbf{b}$ on $\mathcal{T}^\infty$
Solve $\mathbf{u} = \mathbf{L}^{-1}\mathbf{b}$ on $\mathcal{T}$
Assembly $\frac{\partial J}{\partial U}$ on $\mathcal{T}^\infty$
Solve $\Lambda = \mathbf{L}^{-1}\frac{\partial J}{\partial U}$ on $\mathcal{T}$
Assembly $\frac{\partial J}{\partial X}$ on $\mathcal{T}^\infty$
Assembly $\frac{\partial F}{\partial X}$ on $\mathcal{T}^\infty$
Assembly $\frac{\partial B}{\partial X}U$ on $\mathcal{T}^\infty$
Output: $\frac{dJ}{dX} := \frac{\partial J}{\partial X} - \Lambda\left(\frac{\partial B}{\partial X}U - \frac{\partial F}{\partial X}\right)$
Stop

Figure 6.7: Assembly of the material derivative.



(a) Straight rod (b) Bent rod

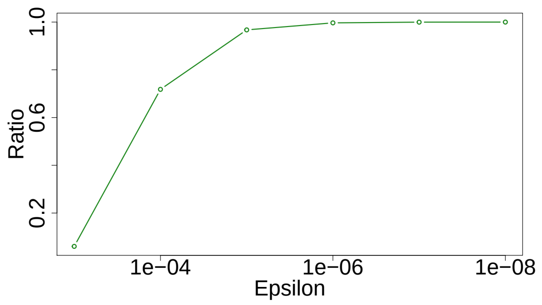
Figure 6.8: Solutions of the linear elasticity equation.

6.2.3 Numerical results

In this section, we provide some examples of the computations of the shape derivatives of the two functionals that we are considering in this work.

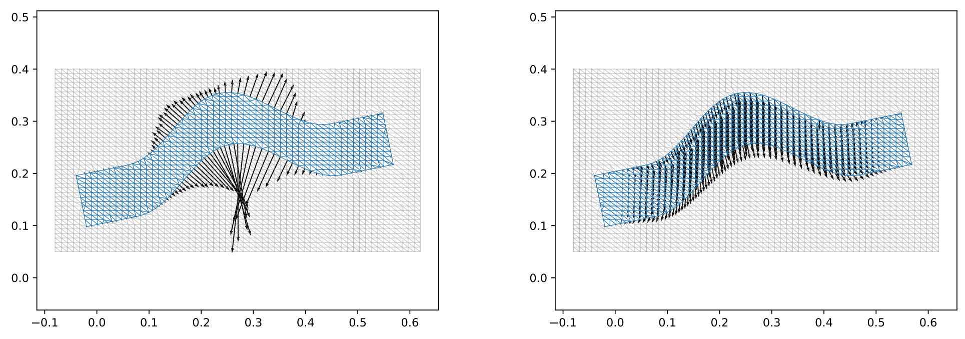
Ceramic Material

We consider the model problem of the bent rod introduced in Section 6.1.3. As described earlier, it is clamped on the left boundary and the tensile load is applied on the right part of the boundary. We assume that the rod is made from a ceramic material, in this case Aluminium oxide (Al_2O_3) and set the material parameters $E = 300$ GPa and $\mu = 0.21$ accordingly. We chose this material for the numerical tests since it is a technical ceramic with low Weibull module m, which is favorable for the smoothness of the derivatives of the objective functional. We consider the PDE-constrained optimization problem that we described in equation (3.18).
As a first test, we solve the linear elasticity equation on the bent rod and on the straight rod that we have seen in Section 6.1.3 as well and visualize it in Figure 6.8. The resulting displacement is smooth and behaves as expected from a material science perspective. We tested the implementation of the linear elasticity equation and the adjoints of the ceramic functional in an earlier implementation and on a different mesh [8] with finite differences. We analyzed the convergence of the ratio of the norms of the finite element discrete



Epsilon	Ratio
10^{-3}	0.0603255
10^{-4}	0.7180954
10^{-5}	0.9672460
10^{-6}	0.9966801
10^{-7}	0.9996691
10^{-8}	0.9999882

Figure 6.9: Convergence of $||\frac{dJ(X,U)}{dX}||_2/||(\frac{J(X+\varepsilon V,U(X+\varepsilon V))-J(X,U)}{\varepsilon V})||_2$ for the two-dimensional objective functional. From [8].



(a) Standard gradient (b) Smoothed gradient

Figure 6.10: Gradient in standard scalar product and smoothed gradient; $N_x = 64, N_y = 32$. From [3].

adjoint and the finite differences derivative, as displayed in Figure 6.9. We compute the derivative according to the discretizations provided in Section 3.3 on the bent rod. The parameters are set to those of Aluminium oxide and the Weibull parameter $m = 2$. To discretize $\widetilde{\Omega}$, we set with $N_x = 61$ and $N_y = 41$. The derivative is displayed in Figure 6.10a. To be utilized in gradient based optimization, the derivative needs smoothing, which we conduct using a Dirichlet-to-Neumann map [47], that is closely related to the method of Steklov-Poincaré gradients, introduced in Chapter 4. The smoothing is performed by solving a linear elasticity equation on $\mathcal{T}(\Omega)$ and using the part of the solution on the boundary nodes as search direction. As system matrix,

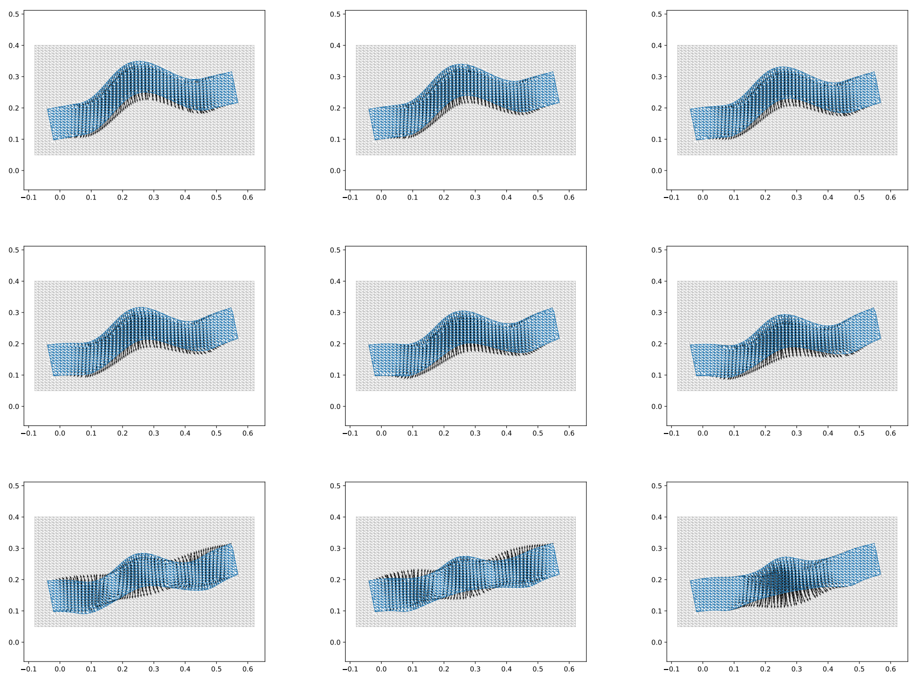
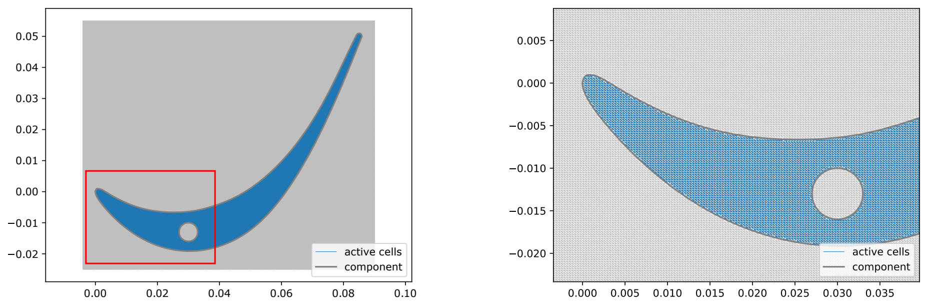


Figure 6.11: Optimizing the shape in a controlled manor, with step size bounded by h. Displayed are iterations no. 1, 3, 5, 7, 9, 11, 13, 15 and 16.

we reuse the system matrix of the governing linear elasticity equation and as right hand side the previously computed shape derivative is used. The system is modified by setting zero-Dirichlet-boundary conditions on those parts of the boundary that are not to be changed, that is the previously defined parts $\partial\Omega_D$ and $\partial\Omega_{N_{fixed}}$. The smoothed gradient (on the whole domain Ω) is visualized in Figure 6.10b.

To highlight what the present implementation is able to do, we perform several steps of a basic gradient based optimization with the smoothed gradient, as visualized in Figure 6.11. The step length is controlled only in the way that it is bounded such that each step is moved by at most $\min(h_x, h_y)$. All the meshes and gradients are computed fully automatically. For a more advanced application of the method in multi-objective optimization, we refer the reader to [49].

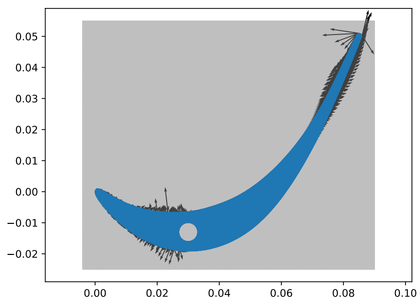


(a) Turbine blade (b) Zoom in on the mesh

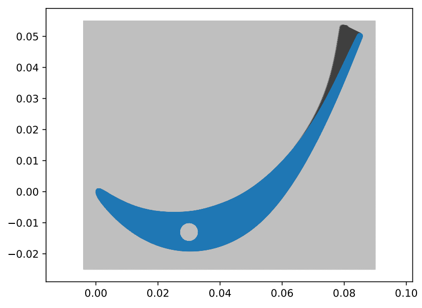
Figure 6.12: Discretized turbine blade and zoom in on part of the mesh.

Low cycle fatigue

For the low cycle fatigue functional, we want to consider a second model problem which is the two-dimensional slice of a turbine blade, visualized in Figure 6.12. The turbine blade possesses a hole in the center, that is supposed to be a cooling channel. We can see that the mesh can capture not only the outer shape of the domain, but can also handle different topologies. We set $\widetilde{\Omega}$ to be the rectangle of $[-0.004, 0.09] \times [-0.025, 0.055]$ and discretize with $n_x = n_y = 299$. As this is a very fine mesh, in Figure 6.12b we provide an enlarged view of a part of the mesh. Neumann-boundary conditions hold on the outer part of the boundary, the part that defines the shape of the blade, with constant load in direction of the outer normal. Additionally, we define zero-Dirichlet-boundary conditions on the inner part of the boundary that is supposed to describe the cooling channel. We calculate the LCF shape derivative, visualized in Figure 6.13a. As the derivative is not smooth, we smooth the gradient once again with the Dirichlet-to-Neumann map, same as in the previous example, see Figure 6.13b.



(a) Standard gradient

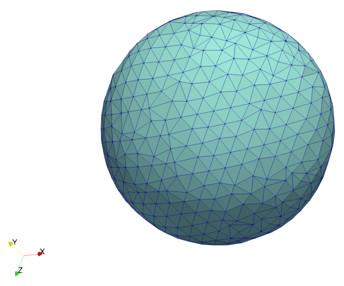


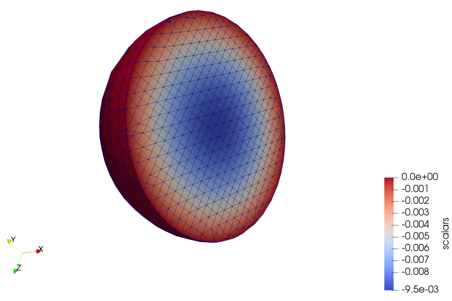
(b) Smoothed Gradient

Figure 6.13: Discrete adjoint for LCF on blade.

6.3 Three Dimensional Examples

As a first model problem, we consider a ball centering at the origin and with radius $r = 0.1$. We define $\widetilde{\Omega}$ as the cuboid $[-0.11, 0.11]^3$ and discretize it with $n = 16$ as visualized in Figure 6.14a. We solve the Poisson equation


(a) Mesh representation



(b) Solution of Poisson equation

Figure 6.14: Visualization of the solution of the Poisson equation on a ball in 3D with $n = 16$.

with the true solution $u(x, y, z) = x^2 + y^2 + z^2 - r^2$ and compare it to the true solution, which gives an error in the L_2-norm of $\hat{e}_{rel} = 1.005 \cdot 10^{-08}$, see Figure 6.14b. As the second test case we consider the shape of a cylinder that we can bent in the mid-section, similar to the rod in the two dimensional case. It is 0.5m long and has radius 0.1m. We set $\widetilde{\Omega}$ to be the cuboid $[-0.1, 0.6] \times [-0.2, 0.2] \times [-0.2, 0.2]$. We discretize by $n_x = n_y = n_z = 31$.

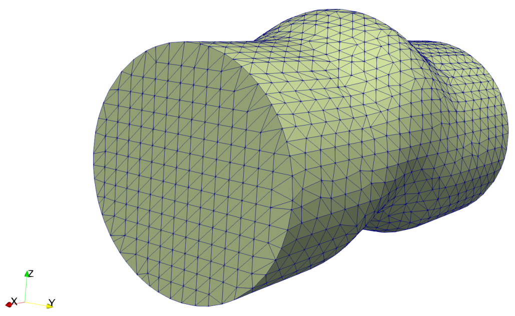


Figure 6.15: Mesh for bent rod.

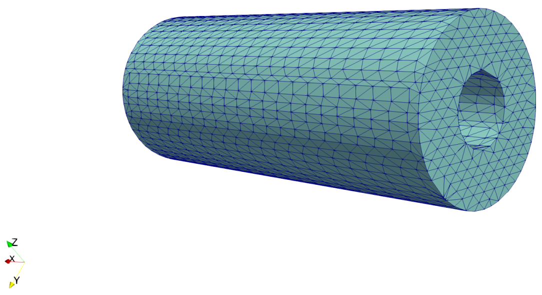


Figure 6.16: Mesh for rod with cooling channel.

The result is visualized in Figure 6.15. We can see that we are able to capture the smooth parts and the edges of the boundary. In Figure 6.16, we see a straight cylinder with a cooling channel. The solver for the linear elasticity equation is also implemented in three dimensions. For the bent cylinder, we compute the solution of the linear elasticity equation. We apply the force on the right end of the cylinder, as visualized in Figure 6.17. The solution of the linear elasticity equation is visualized in Figure 6.18.

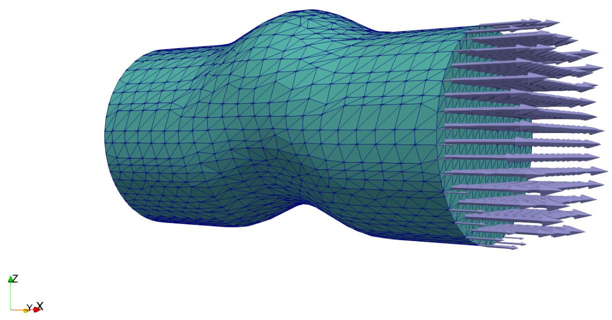


Figure 6.17: Right hand side of linear elasticity equation.

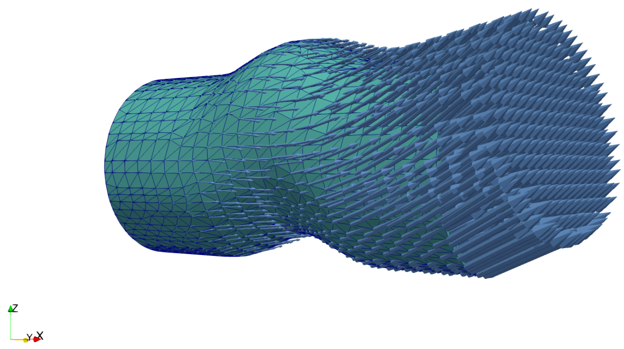


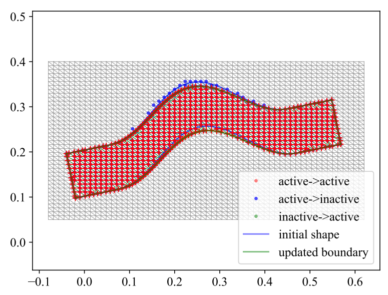
Figure 6.18: Solution of the linear elasticity equation in three dimensions.

6.4 Krylov subspace recycling for evolving geometries

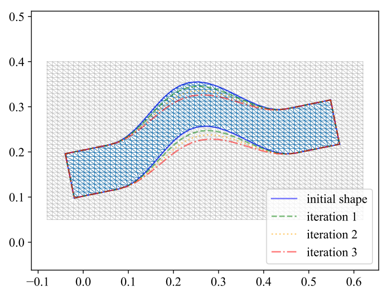
In this section, we give two examples of Krylov subsapce recycling on the presented meshes, to demonstrate the efficacy of the method described in Section 5. The first model problem that we consider is the turbine blade introduced in Section 6.2.3, one which we solve the Poisson equation. For the second one, we solve the linear elasticity equation on the bent rod introduced in the same section. As well as some parts of Chapter 5, the results of this section have been published in [7].

6.4.1 Test of mapping an approximate invariant subspace

As our first example, we consider a model problem solving the linear elasticity equations on the bent rod that we already encountered in Section 6.1.3, with a 181×121 nodes grid, resulting in 5507 active nodes. To get a first impression of the quality of the approximation of the invariant subspace via the mapping, we perform deformation steps of the shape in a controlled way, see Figure 6.19b, and calculate the principal angles of the resulting mapped approximate invariant subspaces and the true invariant subspace corresponding to the smallest eigenvalues of the new system matrix, which has been computed for the purpose of comparison only. We perform three steps of deformation of the original shape, with $N_{rec} = 15$, and we calculate the principal angles between the mapped approximate invariant subspace and the invariant subspace corresponding to the 20 smallest eigenvalues according to [20, p. 604]. We can see that most of the angles are rather small, i.e., our approach approximates these spaces quite well. Note that we only consider small geometric deformations.



(a) The status changes for nodes near the boundary for a small change in shape. From [7].



(b) Sequence of shapes for a model problem. From [7].

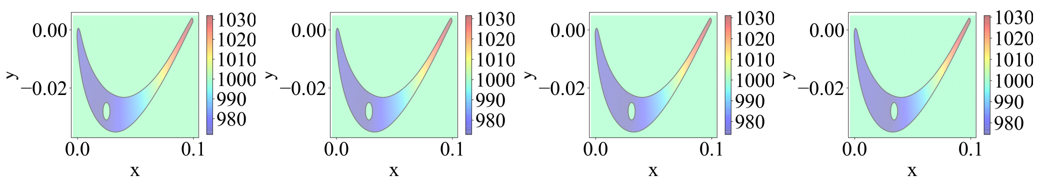
Figure 6.19: Visualization of the changes in the geometry for the model problem.

no.	1	⋯	8	9	10	11	12	13	14	15
i 1	0.992	⋯	0.945	0.939	0.935	0.935	0.913	0.872	0.806	0.150
i 2	0.983	⋯	0.904	0.843	0.785	0.722	0.551	0.331	0.153	0.102
i 3	0.987	⋯	0.933	0.908	0.899	0.844	0.781	0.700	0.476	0.217

Table 6.3: Cosines of principal angles ($\cos(\theta_i)$) between the approximate invariant subspace and the true invariant subspace corresponding to the 20 smallest eigenvalues. From [7].

6.4.2 Poisson equation on a turbine blade

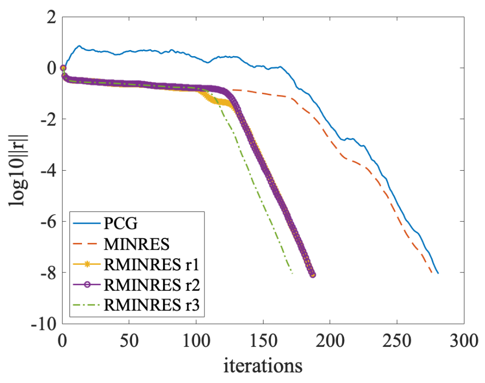
As a first example, we solve the Poisson equation on the turbine blade from Section 6.2.3. The hole that is representing the cooling channel is artificially moved to produce a change in the geometry. $\widetilde{\Omega}$ is discretized by $N_x = 361$ and $N_y = 181$. This leads to close to $12,000$ active nodes on $\mathcal{T}$. On the boundary of the blade, Robin-boundary conditions hold with constant heat coefficients, and the temperature on the cooling channel boundary is chosen to be two times lower than the one on the outer boundary. We perform an initial solve using MINRES and then three rMINRES solves for three consecutive changes of the domain; the geometries are visualized in Figure 6.20. A simple incomplete cholesky factorization $IC(0)$ [44, p.312] is used as preconditioner. Figure 6.21 demonstrates that a speed-up of more than 30% is obtained. In this example, the number of nodes does not change, due to



(a) Initial domain (b) Domain 1 (c) Domain 2 (d) Domain 3

Figure 6.20: Test problem 1: Poisson equation on a turbine blade. Three consecutive positions of the hole. From [7].

the fact that the hole is moved in a very controlled way and without changes in size. Nevertheless, the rows and columns of the matrix do not represent the same nodes in the region of the hole.

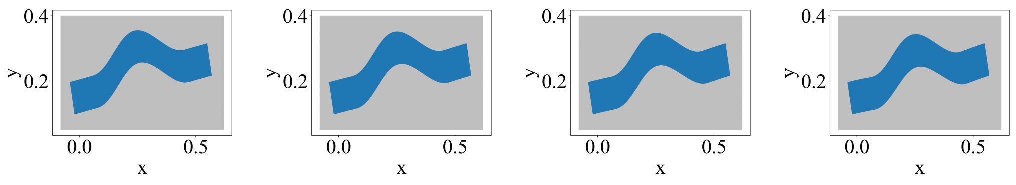


Data for $k = 15$			
Opt. step	N	# its	active (inactive)
0	11,893	275	-
1	11,893	186	210 (210)
2	11,893	186	210 (210)
3	11,893	171	210 (210)

Figure 6.21: Convergence results for solving Poisson's equation for 3 consecutive optimization steps. On the left, the residual norm convergence. Right N the number of unknowns, number of rMINRES iterations, and number of nodes changing from active to inactive or vice versa. From [7].

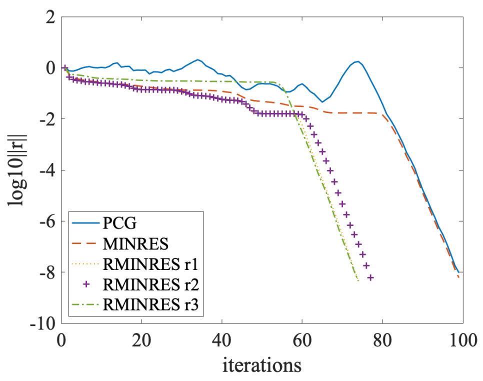
6.4.3 Gradient based shape optimization with linear elasticity as governing PDE

In the second example, we consider the bent rod from Section 6.2.3 with the same boundary conditions and the same load applied. For this example, the deformation in the domain originates from steps of a gradient based optimization with the gradients computed according to Section 6.2.2 and smoothed with a Dirichlet-to-Neumann map, as we have already done in the



(a) Initial domain (b) Domain 1 (c) Domain 2 (d) Domain 3

Figure 6.22: Test problem 2: subsequent shapes in the optimization of a bent rod. From [7].



Data for $k = 20$			
Opt. step	N	# its	active (inactive)
0	$30,108$	98	-
1	$30,062$	73	352 (375)
2	$30,030$	76	362 (387)
3	$29,982$	73	353 (376)

Figure 6.23: Convergence results for solving the linear elasticity equation for 3 consecutive optimization steps. On the left, the residual norm convergence. Right N the number of unknowns, number of rMINRES iterations, and number of nodes changing from active to inactive or vice versa. From [7].

previous section. $\widetilde{\Omega}$ is discretized by $N_x = 301$ and $N_y = 201$ which gives approximately $15,000$ active nodes. As the linear elasticity equation bases on a bilinear form, this leads to about $30,000$ unknowns. The resulting linear system is ill-conditioned, therefore we allow for a drop tolerance of $tol = 0.001$ for the IC-preconditioning. Additionally, the matrix is reordered via reverse Cuthill-McKee reordering [18]. As in the previous example, three rMINRES solves are performed after the initial MINRES solve on three consecutive configurations in the optimization process. The convergence is visualized in Figure 6.23. Here we observe a speed up of the convergence of around 25%.

Chapter 7

Conclusion

In the present work, we presented an approach to automatically generate structured finite element meshes for shape optimization applications. We demonstrated for the two-dimensional case, that with few conditions posed on the shape of the domains and the step length in the optimization, the presented algorithm leads to stable meshes, that fulfill the conditions of the standard convergence results in finite element analysis. We gave examples by implementing solvers for the Poisson equation and the linear elasticity equation in two and three dimensions and solved these equations on different auto-generated meshes. Furthermore, we saw that this structured meshing approach reduces the computational cost in the assembly of the linear system. Additionally to that we described objective functionals for shape optimization that measure the probability of failure of components made of ceramic material and materials that are subject to low cycle fatigue. We computed their discrete adjoint derivatives and applied them in gradient based shape optimization.
In the scope of Krylov subspace recycling, we proposed an approach to make Krylov subspace recycling accessible for re-meshing approaches in shape optimization by introducing transformation, mapping basis vectors of subspaces from one mesh representation to the next. We analyzed the accuracy of the mapping by considering the principle angles between the subspaces and performed several examples on two dimensional meshes, observing that the

convergence improves through the recycling substantially.

In future research, a rigorous analysis of the three dimensional meshing is necessary, to show that the approach fulfills the conditions for finite element convergence theorems. Furthermore, the meshing should be paired with an appropriate parallel solver to really benefit from this structured approach. It would be also interesting to carefully add adaptive mesh refinement in a way that does not counter the possible speed up gained through the presented approach.

Bibliography

[1] M. S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[2] W. Arendt and K. Urban. *Partielle Differenzialgleichungen. Eine Einführung in analytische und numerische Methoden.* Spektrum Akademischer Verlag, Heidelberg, 2010.

[3] J. Backhaus, M. Bolten, O. T. Doganay, M. Ehrhardt, B. Engel, C. Frey, H. Gottschalk, M. Günther, C. Hahn, J. Jaschke, P. Jaksch, K. Klamroth, A. Liefke, D. Luft, L. Mäde, V. Marciniak, M. Reese, J. Schultes, V. Schulz, S. Schmitz, J. Steiner, and M. Stiglmayr. Given - shape optimization for gas turbines in volatile energy networks. In S. Goettlich, M. Herty, and A. Milde, editors, *Mathematical MSO for Power Engineering and Management, Mathematics in Industry*, Cham, 2021. Springer.

[4] K. Becker, K. Heitkamp, and E. Kügeler. Recent progress in a hybrid-grid cfd solver for turbomachinery flows. In J. C. F. Pereira, A. Sequeira, and J. M. C. Pereira, editors, *V European Conference on Computational Fluid Dynamics ECCOMAS CFD 2010*, 2010.

[5] M. Berger. Chapter 1 - cut cells: Meshes and solvers. In R. Abgrall and C.-W. Shu, editors, *Handbook of Numerical Methods for Hyperbolic Problems*, volume 18 of *Handbook of Numerical Analysis*, pages 1–22. Elsevier, 2017.

[6] M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53:84–512, 1984.

[7] M. Bolten, E. de Sturler, and C. Hahn. Krylov subspace recycling for evolving structures, 2020.

[8] M. Bolten, H. Gottschalk, C. Hahn, and M. Saadi. Numerical shape optimization to decrease failure probability of ceramic structures. *Comput. Visual Sci.*, 2019.

[9] M. Bolten, H. Gottschalk, and S. Schmitz. Minimal failure probability for ceramic design via shape control. *J. Optim. Theory Appl.*, 166(3):983–1001, 2015.

[10] D. Braess. *Finite elements. Theory, Fast Solvers, and Applications in Solid Mechanics.* Cambridge University Press, Cambridge, 1997.

[11] J. Brandts, A. Hannukainen, S. Korotov, and M. Kříźek. On angle conditions in the finite element nethod. *SeMA Journal*, 56(1):81–95, 2011.

[12] S. Brenner and R. Scott. *The Mathematical Theory of Finite Element Methods.* Springer Verlag, New York, 2008.

[13] J.-S. Chen, M. Hillman, and S.-W. Chi. Meshfree methods: Progress made after 20 years. *Engineering Mechanics*, 143:04017001, 2017.

[14] B. Delaunay. Sur la sphère vide. a la mémoire de georges voronoï. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, 6:793–800, 1934.

[15] P. J. Denning. The locality principle. *Commun. ACM*, 48(7):19–24, July 2005.

[16] C. C. Douglas, J. Hu, M. Kowarschik, U. Rüde, and C. Weiss. Cache optimization for structured and unstructured grid multigrid. *Electron. Trans. Numer. Anal.*, 10:21–40, 2000.

[17] O. Forster. *Analysis 2*. Springer Spektrum, Wiesbaden, 11 edition, 2017.

[18] A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall Professional Technical Reference, 1981.

[19] R. M. German and Z. Z. Fang. *1 - Thermodynamics of sintering*, pages 3–32. Woodhead Publishing, 2010.

[20] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore and London, 3 edition, 1996.

[21] H. Gottschalk and M. Saadi. Shape gradients for the failure probability of a mechanic component under cyclic loading: a discrete adjoint approach. *Comput. Mech.*, 64(4):895–915, 2019.

[22] A. Greenbaum. *Iterative methods for solving linear systems*, volume 17 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[23] D. Gross and T. Seelig. *Bruchmechanik. Mit einer Einführung in die Mikromechanik*. Springer-Verlag, Berlin Heidelberg, 2007.

[24] W. Hackbusch and S. Sauter. Adaptive composite finite elements for the solution of pdes containing nonuniformely distributed micro-scales. *Matem. Mod.*, 8:31–43, 1996.

[25] W. Hackbusch and S. Sauter. Composite finite elements for problems containing small geometric details. *Comput. Visual. Sci.*, 1:15–25, 1997.

[26] W. Hackbusch and S. Sauter. Composite finite elements for the approximation of pdes on domains with complicated micro-structures. *Num. Math.*, 75:447–472, 1997.

[27] J. Haslinger and R. A. E. Mäkinen. *Introduction to shape optimization: theory, approximation, and computation*. SIAM, Philadelphia, 2003.

[28] M. R. Hestenes and E. Stiefel. Method of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand*, 49:409–436, 1952.

[29] C. Hirt, A. Amsden, and J. Cook. An arbitrary lagrangian-eulerian computing method for all flow speeds. *J. Comput. Phys.*, 14:227–253, 1974.

[30] O. Kallenberg. *Random measures.* Akademie Verlag, Berlin, 1982.

[31] M. Kilmer and E. de Sturler. Recycling subspace information for diffuse optical tomography. *SIAM J. Sci. Comput.*, 27(6):2140–2166, 2006.

[32] A. Klenke. *Probability Theory.* Springer, London, 2 edition, 2014.

[33] J. Liesen and P. Tichý. Convergence analysis of krylov subspace methods. *GAMM-Mitteilungen*, 27(2):153–173, 2004.

[34] S. H. Lo. Finite element mesh generation and adaptive meshing. *Progr. Struct. Eng. Mater.*, 4(4):381–399, 2002.

[35] A. Meister. *Numerik linearer Gleichungssysteme.* Vieweg, Braunschweig/Wiesbaden, 1999.

[36] R. Misener and C. A. Floudas. Piecewise-linear approximations of multidimensional functions. *J. Optim. Theory Appl.*, 145:120–147, 2010.

[37] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annual Review of Fluid Mechanics*, 37(1):239–261, 2005.

[38] L. Motta Mello, E. de Sturler, G. Paulino, and E. C. Nelli Silva. Recycling Krylov subspaces for efficient large-scale electrical impedance tomography. *Comput. Methods Appl. Mech. Engrg.*, 199:3101–3110, 2010.

[39] H. Neuber. Theory of Stress Concentration for Shear-Strained Prismatical Bodies With Arbitrary Nonlinear Stress-Strain Law. *J. Appl. Mech.*, 28(4):544–550, 12 1961.

[40] C. C. Paige and M. A. Saunders. Solutions of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12(4):617–629, 1975.

[41] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti. Recycling krylov subspaces for sequences of linear systems. *SIAM J. Sci. Comput.*, 28:1651–1674, 2004.

[42] C. Peskin. Recycling krylov subspaces for sequences of linear systems. *J. Comp. Phys.*, 10:252–271, 1972.

[43] A. Reisner, L. N. Olson, and J. D. Moulton. Scaling structured multigrid to 500K+ cores through coarse-grid redistribution. *SIAM J. Sci. Comput.*, 40(4):C581–C604, 2018.

[44] Y. Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics(SIAM), USA, 2nd edition, 2003.

[45] Y. Saad and M. H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[46] M. Saadi. *Shape Sensitivities for the Failure Probability of Mechanical Components.* PhD thesis, Bergische Universität Wuppertal, Wuppertal, 2020.

[47] S. Schmidt and V. Schulz. Impulse response approximations of discrete shape Hessians with application in CFD. *SIAM J. Contr. Optim.*, 48(4):2562–2580, 2009.

[48] S. Schmitz, T. Seibel, T. Beck, G. Rollmann, R. Krause, and H. Gottschalk. A probabilistic model for lcf. *Comput. Mater. Sci.*, 79:584–590, 2013.

[49] J. Schultes, M. Stiglmayr, K. Klamroth, and C. Hahn. Hypervolume scalarization for shape optimization to improve reliability and cost of ceramic components. *Optim. Eng.*, 2021.

[50] V. Schulz, M. Siebenborn, and K. Welker. Efficient PDE constrained shape optimization based on steklov-poincaré-type metrics. *SIAM J. Optim.*, 26:2800–2819, 2016.

[51] S. M. Shontz and S. A. Vavasis. Analysis of and workarounds for element reversal for a finite element-based algorithm for warping triangular and tetrahedral meshes. *BIT Numerical Mathematics*, 50(4):863–884, 2010.

[52] V. Simoncini and D. Szyld. On the occurrence of superlinear convergence of exact and inexact krylov subspace methods. *Siam Review - SIAM REV*, 47:247–272, 01 2005.

[53] K. M. Soodhalter, E. de Sturler, and M. E. Kilmer. A survey of subspace recycling iterative methods. Technical Report https://arxiv.org/abs/2001.10347, arXiv, 2020.

[54] M. L. Staten, S. J. Owen, S. M. Shontz, A. G. Salinger, and T. S. Coffey. A comparison of mesh morphing methods for 3d shape optimization. In W. R. Quadros, editor, *Proceedings of the 20th International Meshing Roundtable*, pages 293–311, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[55] W. A. Strauss. *Partielle Differentialgleichungen*. Vieweg+Teubner, Wiesbaden, 1995.

[56] G. Taylor and H. Quinney. The plastic distortion of metals. *Phil. Trans. R. Soc., London*, 230:323–362, 1931.

[57] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[58] L. N. Trefethen and J. A. C. Weideman. The exponentially convergent trapezoidal rule. *SIAM Review*, 56(3):385–458, 2014.

[59] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2003.

[60] G. van Roosum. Python tutorial. Technical Report CS-R9526, CWI, 05 1995.

[61] S. Wang. *Krylov Subspace Methods for Topology Optimization on Adaptive Meshes*. PhD thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, 2007. Advisor: Eric de Sturler.

[62] S. Wang, E. de Sturler, and G. H. Paulino. Large scale topology optimization using preconditioned Krylov subspace methods with recycling. *Int. J. Numer. Meth. Engng.*, 69:2441–2468, 2007.

[63] S. Watanabe. On discontinuous additive functionals and lévy measures of a markov process. *Japan J. Math.*, 34, 1964.