Diplomarbeit

Adaptive Netzverfeinerung in der Formoptimierung mit der Methode der Diskreten Adjungierten

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Mathematik

eingereicht: Andreas Günnel
geb. am 12.03.1986 in Zwickau

Studiengang: Technomathematik

Betreuer: Prof. Dr. Arnd Meyer
Dr. René Schneider

Chemnitz, den 22. Januar 2010

Diploma Thesis

Adaptive Mesh Design in Shape Optimization with the Discrete Adjoint Method

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Andreas Guennel

Chemnitz, January 22, 2010

Notation

$(\cdot)_{,s}$	Partial differential with respect to the s-th component on the world element
$[\cdot]_r$	r-th component of a vector
u	Analytical solution of the deformation of the Lamé problem
x_1, x_2	Coordinates in a two-dimensional space
λ, μ	Lamé constants
σ	Stress tensor
ε	Strain tensor
Γ_D	Partial boundary with the Dirichlet condition
Γ_N	Partial boundary with the Neumann condition
Ω	Domain where the PDE is solved
u, u_h	Discretized Solution
N	Number of nodes of a mesh
$\mathbb{T}$	Triangulation of Ω
φ_i	Basis function of node i
φ_i^r	Test functions of node i with φ_i in the r-th component
$a(\cdot,\cdot)$	Bilinear form
$b(\cdot)$	Linear right hand side form
K	Stiffness matrix
$\underline{f}$	Right hand side or load vector
$\underline{u}$	Solution vector
J	Jacobian of the element mapping
I	Performance function
p	Vector holding the shape parameters
$u_{r,s}$	Partial differential of the r-th component of u with respect to x_s on the world element
$\overline{r}$	is equal 2 for $r = 1$ and equal 1 for $r = 2$
DAM	Discrete Adjoint Method

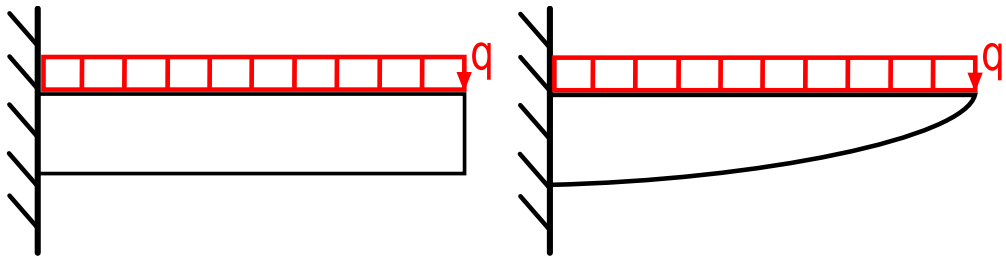
dof.....	Degrees of freedom
FEINS.....	Finite Elements for Incompressible Navier-Stokes
FEM.....	Finite Element Method
PCG-Method	Preconditioned Conjugate Gradient Method

Preface

Shape optimization describes the determination of the geometric shape of a domain with a partial differential equation (PDE) with the purpose that a specific given performance function is minimized, its values depending on the solution of the PDE. The Discrete Adjoint Method can be used to evaluate the gradient of a performance function with respect to an arbitrary number of shape parameters by solving an adjoint equation of the discretized PDE. This gradient is used in the numerical optimization algorithm to search for the optimal solution. As both function value and gradient are computed for the discretized PDE, they both fundamentally depend on the discretization. In using the coarse meshes, discontinuities in the discretized objective function can be expected if the changes in the shape parameters cause discontinuous changes in the mesh (e.g. change in the number of nodes, switching of connectivity). Due to the convergence of the discretization these discontinuities vanish with increasing fineness of the mesh. In the course of shape optimization, function value and gradient require evaluation for a large number of iterations of the solution, therefore minimizing the costs of a single computation is desirable (e.g. using moderately or adaptively refined meshes). Overall, the task of the diploma thesis is to investigate if adaptively refined meshes with established methods offer sufficient accuracy of the objective value and gradient, and if the optimization strategy requires readjustment to the adaptive mesh design. For the investigation, applicable model problems from the science of the strength of materials will be chosen and studied.

1. Introduction

The aim of shape optimization is to answer the question which shape of a partial differential equation (PDE) domain causes the PDE solution to have the most desirable properties concerning a given criterion. In this context the meaning of optimality is manifold, whereby various practical models can be the object of investigation. In this work, we consider the Lamé problem, that is, for example, a beam under the influence of exterior forces. Its shape shall be determined such that a given goal function is minimized. Figure 1.1 shows an example of two beams with almost the same stiffness but requiring different amounts of material.



(a) Initial beam (b) Optimized beam

Figure 1.1.: Shape optimization of a standard beam

Since most of the problems are not solvable with analytic elasticity theory we use numerical approximation methods like the finite element method (FEM). The standard FEM does not allow parameters describing the domain, yet the optimization is still possible. We use the discrete adjoint method (DAM) which is used to determine the gradient of the goal function efficiently. This gradient can be used by a numerical optimization algorithm to solve the optimization problem. Within the scope of this work, we consider only two-dimensional linear elasticity problems. The task is to extend the FE-solver *FEINS*[1] from R. Schneider with the adaptive mesh design and the DAM and embed it in an optimization algorithm.

[1] Finite Elements for Incompressible Navier Stokes. Although starting as a solver for Navier Stokes problems, it was extended to Lamé problems as well.

2. Lamé problem

2.1. Analytic formulation of the Lamé problem

Linear elasticity

Object of our investigation is a beam that is loaded with an external force. Volume forces like gravity are neglected in this work. We are interested in the displacement u of the beam due to the external force. Assuming that the material has isotropic behavior and that the deformations are very small compared to the geometry, the linear elasticity theory can be applied, [3, chapter 6.3], thus the symmetrical strain tensor is approximated by

$$\varepsilon_{ij}(\mathrm{u}) := \frac{1}{2}(\mathrm{u}_{i,j} + \mathrm{u}_{j,i}) \quad \text{with} \quad \mathrm{u}_{i,j} := \frac{\partial \mathrm{u}_i}{\partial \mathrm{x}_j},$$

and the symmetrical stress tensor is

$$\sigma(\mathrm{u}) := C\varepsilon(\mathrm{u}), \tag{2.1}$$

with the symmetrical stiffness tensor C which is the tensor expression of Hooke's law, [3, chapter 6.1]. Further, using the linear material law from Hooke, the stiffness tensor C can be written as

$$\begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{13} \\ \sigma_{23} \end{bmatrix} = C \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \varepsilon_{12} \\ \varepsilon_{13} \\ \varepsilon_{23} \end{bmatrix} = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & & & \\ \lambda & \lambda+2\mu & \lambda & & & \\ \lambda & \lambda & \lambda+2\mu & & & \\ & & & 2\mu & & \\ & & & & 2\mu & \\ & & & & & 2\mu \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \varepsilon_{12} \\ \varepsilon_{13} \\ \varepsilon_{23} \end{bmatrix}, \tag{2.2}$$

and its inverse

$$C^{-1} = \frac{1}{2\mu(3\mu+2\lambda)} \begin{bmatrix} 2(\lambda+\mu) & -\lambda & -\lambda & & & & \\ -\lambda & 2(\lambda+\mu) & -\lambda & & & & \\ -\lambda & -\lambda & 2(\lambda+\mu) & & & & \\ & & & 3\lambda+2\mu & & & \\ & & & & 3\lambda+2\mu & & \\ & & & & & 3\lambda+2\mu & \end{bmatrix}, \quad (2.3)$$

with the Lamé constants λ and μ that characterize the material. A common alternative characterization of the material is by Young's modulus E and Poisson's ration ν which are related to λ and μ according to

$$\begin{array}{rclcrcl} \lambda & = & \frac{E\nu}{(1+\nu)(1-2\nu)}, & & \mu & = & \frac{E}{2(1+\nu)}, \\ E & = & \frac{\mu(3\lambda+2\mu)}{\lambda+\mu}, & & \nu & = & \frac{\lambda}{2(\lambda+\mu)}. \end{array}$$

The physics background constrains these constants to $\lambda > 0$, $\mu > 0$ and $E > 0$, $0 < \nu < \frac{1}{2}$. Equation (2.1) can be rewritten as the Lamé partial differential equation

$$-2\mu \text{ div } \varepsilon(\mathrm{u}) - \lambda \text{ grad div } \mathrm{u} = 0 \qquad \text{in } \Omega \qquad (2.4)$$

with boundary conditions

$$\mathrm{u} = 0 \qquad \text{on } \Gamma_D \text{ (Dirichlet)} \qquad (2.5)$$

$$\sigma(\mathrm{u}) \cdot n = g \qquad \text{on } \Gamma_N \text{ (Neumann)}, \qquad (2.6)$$

where Ω is the domain describing the geometrical shape of the mechanical structure, u is the displacement due to the forces, $\varepsilon(\mathrm{u})$ is the resulting strain and $\sigma(u)$ the stress. The external force per area g acts on the part Γ_1 of the boundary, called the Neumann boundary. The beam is clamped on the boundary part Γ_D, called the Dirichlet boundary. The remaining edge $\partial\Omega \setminus (\Gamma_D \cup \Gamma_N)$ is considered stress-free, meaning a Neumann condition with $g = 0$. This partial differential equation in its analytical formulation is a second order elliptic PDE.

Reduction to two dimensions

Additionally, the beam should be reducible to a two-dimensional shape in the x_1-x_2-plane, meaning that the shape of the beam and the forces do not depend on the depth x_3. There are two main cases that appear in practice: Plane strain, where the strain vanishes in the

x_3-direction, and plane stress, where the stress vanishes in x_3-direction, [3, chapter 6.5].
The plane strain applies for cases where displacements in x_3-direction are not possible, for example very wide beams. Therefore strains in x_3-direction vanish,

$$\varepsilon_{=} \begin{bmatrix} \varepsilon_{11} & \varepsilon_{12} & 0 \\ \varepsilon_{12} & \varepsilon_{22} & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

This yields $\sigma_{i3} = \sigma_{3i} = 0$ for $i = 1, 2$ and with $\varepsilon_{33} = 0$ and the material law (2.3), we gain the equation

$$\sigma_{33} = \frac{\lambda}{2(\lambda + \mu)}(\sigma_{11} + \sigma_{22}).$$

By eliminating σ_{33}, (2.1) is reduced to

$$\begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} = \begin{bmatrix} \lambda + 2\mu & \lambda & 0 \\ \lambda & \lambda + 2\mu & 0 \\ 0 & 0 & 2\mu \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{12} \end{bmatrix} \tag{2.7}$$

meaning that the x_3-component in the Lamé equation (2.4) is simply ignored.
The plane stress applies for cases where displacements in x_3-direction are possible, for example thin plates, but stresses in x_3-direction vanish,

$$\sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & 0 \\ \sigma_{12} & \sigma_{22} & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

This yields $\varepsilon_{i3} = \varepsilon_{3i} = 0$ for $i = 1, 2$ and with $\sigma_{33} = 0$ and (2.2), we gain the equation

$$\varepsilon_{33} = -\frac{\lambda}{\lambda + 2\mu}(\varepsilon_{11} + \varepsilon_{22}).$$

By eliminating ε_{33} in (2.1) and with $\tilde{\lambda} = \frac{4\mu(\lambda+\mu)}{\lambda+2\mu}$, we gain

$$\begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} = \begin{bmatrix} \tilde{\lambda} + 2\mu & \tilde{\lambda} & 0 \\ \tilde{\lambda} & \tilde{\lambda} + 2\mu & 0 \\ 0 & 0 & 2\mu \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{12} \end{bmatrix}. \tag{2.8}$$

Comparing this result with (2.7), it is reasonable to consider only plane strain because the case of plane stress is covered by simply adjusting the Lamé constant λ.

2.2. Finite Element Method

There are a number of special domain geometries where the Lamé problem and its PDE can be solved analytically but these are not sufficient for the shape optimization in general. A common and wide-spread numerical approach to solve PDEs is the Finite Element Method (FEM). The main idea is to introduce a discretization of the infinite dimensional function space to approximate the solution in a suitable finite dimensional function space, i.e. only with a finite number of degrees of freedom. For that purpose the domain Ω is triangulated into a finite number of elements, called the mesh. Rewriting the PDE into a weak formulation and introducing basis functions on the elements, a linear system of equation can be obtained as an equivalent problem. Due to the nature of the basis functions, this linear system is sparse and can be efficiently solved with modern iterative solvers even when the number of unknowns is very large and classical direct solvers cannot handle it. In this chapter, the FEM is briefly introduced to clarify notation and its application.

Weak formulation

Applying the FEM requiers the weak formulation of the PDE (2.4). We therefor introdce the function space

$$H^1_{\Gamma_D}(\Omega) = \left\{ v \in H^1(\Omega) : v|_{\Gamma_D} = 0 \right\}.$$

Multiplying with an arbitrary test function $v \in H^1_{\Gamma_D}(\Omega)^2$ and integrating by parts yields the variational formulation, [3, chapter 6.3]:
Find $\mathrm{u} \in H^1_{\Gamma_D}(\Omega)^2$ such that

$$\begin{aligned} a(\mathrm{u}, v) &= b(v) \qquad \forall v \in H^1_{\Gamma_D}(\Omega)^2 && (2.9)\\ a(\mathrm{u}, v) &:= \int_\Omega \varepsilon(v) : C\varepsilon(\mathrm{u})\ dx && (2.10)\\ &:= \int_\Omega \lambda(\nabla \cdot \mathrm{u})(\nabla \cdot v) + 2\mu \sum_{i,j=1}^{2} \varepsilon_{ij}(\mathrm{u})\varepsilon_{ij}(v)\ dx && (2.11)\\ b(v) &:= \int_{\Gamma_N} g \cdot v\ ds. && (2.12) \end{aligned}$$

We remark that a is a bilinear form, b is linear and that u and v have components in x_1 and x_2 direction. The operator ':' is defined as $\varepsilon : \sigma := \sum_{ij} [\varepsilon]_{ij}[\sigma]_{ij} = tr(\varepsilon\sigma^T)$. The Neumann boundary condition in (2.4) is included in the linear form (2.9). To implement the Dirichlet boundary condition we use a projector to fix the values of the affected nodes [8].

Meshing and element type

A mesh generator creates a triangulation $\mathbb{T}$ of the domain Ω, yielding triangular elements $T_i \in \mathbb{T}$. We now choose basis functions on these elements, usually polynomials as their arithmetics are relatively easy. For example, a linear function over T_i has three degrees of freedom that can be uniquely defined by its function values of the three vertices of the triangle. In this work, quadratic basis functions are preferred because they offer a higher convergence rate than linear ones. Besides the three corner nodes the three mid nodes of the edges are used too, so there are six base nodes in total, compare Figure 2.1.

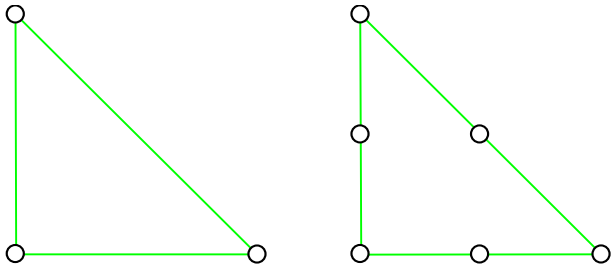


Figure 2.1.: Linear and quadratic elements and their nodes that define the ansatz functions

The additional mid nodes do not need to be in the center of the edges, which leads to the idea of curvilinear triangles that approximate curvilinear shapes much better than straight triangles. Instead of the function space $H^1_{\Gamma_D}(\Omega)$ we consider the discretized function space

$$S_h = \left\{ v \in H^1_{\Gamma_D}(\Omega)^2 : v\,|_{T_i} \in P_2(T_i)^2 \ \forall T_i \in \mathbb{T} \right\}$$

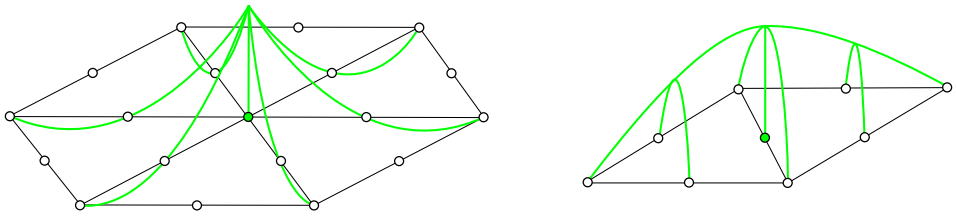
of piecewise quadratic functions. The ansatz functions are chosen such that

$$\varphi_i(s_j) = \delta_{ij} := \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases}, \tag{2.13}$$

where s_j are the nodes of the mesh. This condition uniquely defines the basis functions for each node (Lagrange interpolation). Examples of basis functions for corner and mid nodes can be seen in Figure 2.2 while the ansatz functions are zero on the remaining elements.
The solution u and the test function v can be approximated by

$$\mathrm{u}(x) \approx u_h(x) = \sum_{i=1}^{N} \sum_{r=1}^{2} u_i^r \varphi_i^r(x) \in S_h, \qquad v(x) \approx v_h(x) = \sum_{i=1}^{N} \sum_{r=1}^{2} v_i^r \varphi_i^r(x) \in S_h, \tag{2.14}$$

i.e. restricting (2.9) to the finite dimensional space S_h. N is the number of nodes, $u_i^r, v_i^r \in \mathbb{R}$ are coefficients, and $\varphi_i^r \in S_h$ denotes the basis function, i.e. a vector valued function, corresponding to the node i for the r-th component where the other component is zero. As one can see easily,



(a) in a corner node (b) in a mid node

Figure 2.2.: The two types of ansatz functions

due to the definition (2.13) of the basis functions, u_i^r is the function value of u_h at the node x_i in the r-th dimension. We insert u $\approx u_h$ and $v \approx v_h$ in the weak formulation (2.9) and due to the linearity of $a(u, \cdot)$ and $b(\cdot)$, testing with all functions $v_h \in S_h$ is equivalent to testing with all basis functions only. Thus we get the discretized formulation

$$a(u_h, \varphi_i^r) = b(\varphi_i^r) \qquad \forall i = 1, \ldots, N; r = 1, 2. \tag{2.15}$$

In addition, we introduce the quantities $K \in \mathbb{R}^{2N \times 2N}$ and $\underline{u}, \underline{f} \in \mathbb{R}^{2N}$ defined by

$$\begin{aligned} [K]_{(i,r),(j,s)} &= a(\varphi_j^s, \varphi_i^r) \qquad \forall i, j = 1, \ldots, N; r, s = 1, 2 \quad , \\ [\underline{f}]_{(i,r)} &= b(\varphi_i^r), \\ [\underline{u}]_{(i,r)} &= u_i^r. \end{aligned} \tag{2.16}$$

To understand $\underline{u}$ as a one-column vector, it is necessary to sort the entries. In this work the x_1-entries are sorted first, i.e.

$$\underline{u} = (u_1^1, u_2^1, \ldots, u_N^1, u_1^2, \ldots, u_N^2).$$

With this numbering, the index (i, r) represents a number $i + (r-1)N$ and with the notation (2.16), the discretized Lamé equation (2.15) can be rewritten as the system of linear equations

$$K\underline{u} = \underline{f}. \tag{2.17}$$

The matrix K is called stiffness matrix and $\underline{f}$ load vector. It remains to solve the linear system of equations (2.17) and gain the solution vector $\underline{u} \in \mathbb{R}^{2N}$. Once we have $\underline{u}$ the approximate solution u_h is known and can be evaluated in each point.

Mapping between master and world element

To evaluate the entries of K effectiently, they are assembled element wise with the use of cubature formulas and mapping functions M between a master $\widehat{T}$ element and a world element T_i. In the following all symbols with ' $\widehat{}$ ' refer to the master element, a triangle defined by the vertices $(0,0)$, $(1,0)$ and $(0,1)$. The basis functions $\widehat{\varphi}_i$ on the master element are used for the mapping M,

$$x = M(\widehat{x}) = \sum_i x_i \widehat{\varphi}_i(\widehat{x}),$$

where x_i are the positions of the world element's nodes. This mapping leads to curvilinear triangles (compare Figure 2.3). The basis functions are simply mapped like $\varphi(x) = \widehat{\varphi}(M^{-1}(x)) = \widehat{\varphi}(\widehat{x})$, in particular $\varphi(x_k) = \widehat{\varphi}(\widehat{x}_k)$ for the integration points of the cubature formulas. Due to this mapping, the integration points and their weights for the cubature formulas need to be defined only on the master element. However, one can see that the gradient $\nabla\varphi(x_k)$ and hessian $H(x_k)$ require a seperate transformation.

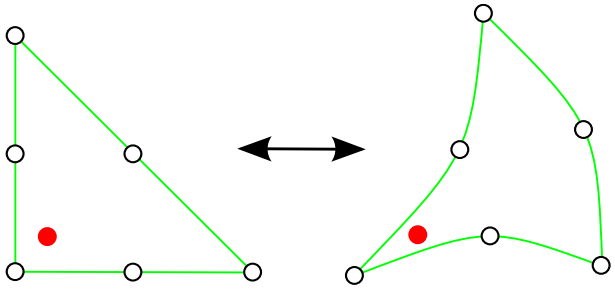


Figure 2.3.: Mapping M of a point on the master element to the world element T_i

The Jacobian $J \in \mathbb{R}^{2\times 2}$ in a point x_k is required for the mapping of the gradient or Hessian and also for the cubature formulas. Recalling some properties of isoparametric finite elements, [9], we have

$$J(\widehat{x}_k) = \left[\frac{\partial \mathrm{x}}{\partial \widehat{\mathrm{x}}}\right] = \sum_i x_i (\widehat{\nabla} \left[\widehat{\varphi}_i(\widehat{x}^k)\right)^T]^T.$$

The world gradient $\nabla\varphi$ can be calculated by

$$\nabla\varphi(x(\widehat{x})) = J(\widehat{x})^{-T}\widehat{\nabla}\widehat{\varphi}(\widehat{x}).$$

Differentiating the world gradient again, we obtain the world Hessian

$$H(x(\widehat{x})) = \left[\frac{\partial^2 \widehat{\mathrm{x}}_k}{\partial \mathrm{x}_i \partial \mathrm{x}_j}\frac{\partial \widehat{\varphi}}{\partial \widehat{\mathrm{x}}_k}\right]_{(i,j)} + J^{-T}(\widehat{x})\widehat{H}_{\widehat{\varphi}}(\widehat{x})J^{-1}(\widehat{x}),$$

where we use the Einstein summation convention for the first quantity which contains the second order differentials of the mapping M^{-1}. This quantity vanishes if the triangle is not curvilinear but straight. Additionally, the Hessian H is only needed in the error estimator. We neglect this quantity because the gained accuracy would not justify the additional computational costs. Therefore the Hessian is evaluated by

$$H_{\varphi}(x(\widehat{x})) \approx J(\widehat{x})^{-T}\widehat{H}_{\widehat{\varphi}}(\widehat{x})J(\widehat{x})^{-1}.$$

Once we are able to compute the world gradient and Hessian, integrals over a curvilinear triangle T_i and integrals over curvilinear edges Γ_i can be evaluated by cubature formulas

$$\begin{aligned}
\int_{T_i} f(x)\ d\Omega &\approx \sum_k \omega_k \widehat{f}(\widehat{x}_k)\left|\det(J(\widehat{x}_k))\right|, \\
\int_{\Gamma_i} f(x)\ ds &\approx \sum_k \omega_k \widehat{f}(\widehat{x}_k)\left\|t(\widehat{x}_k)\right\|_2,
\end{aligned}$$

where the tangent t is computed by

$$\left\|t(\widehat{x}_k)\right\|_2 := \left\|\sum_i x_i \widehat{\nabla}\widehat{\varphi}_i(\widehat{x}_k)\right\|_2. \tag{2.18}$$

Normally, standard triangles can be exactly evaluated with a sufficiently high order cubature formula. Since the inverse mapping M^{-1} does not have to be quadratic, cubature formulas of degree two with six integrations points ([5], [6] and [4]) are an approximation. Again, the less curvilinear the triangle is and the more it relates to a standard triangle, the less is the error.

2.3. Solution method

There are several ways to solve linear systems of equations, ranging from direct solvers like Gaussian elimination or various decompositions to iterative solvers like Gauss-Seidel or conjugate gradient method. Usually the FEM disrectization of a Lamé problem requires a large number of nodes, thus the number of degree of freedom of the linear system can be too large to solve with direct solvers, mainly due to memory requirements, computational cost and even for numerical accuracy. In contrast to that, iterative solvers may take advantage of the sparsity of the system

to keep memory usage and computational cost at a level still manageable. As the convergence rate of such solvers often depends on the condition of the system, various methods have been developed to compensate large condition numbers. In this section, the preconditioned conjugate gradient method (PCG) and the multigrid preconditioner are briefly presented to show their application in this work.

Preconditioned Conjugate Gradient method

Let the system of linear equations (2.17) be given and let u^* be its exact solution. The stiffness matrix is symmetric and positive definite, as well as sparse, that is only $O(N)$ entries are non-zero, where N is the node number of the triangulation. Unfortunately it has a high condition number for large systems,

$$\kappa(K) := \|K\|_2 \left\|K^{-1}\right\|_2 = O(h^{-2}) = O(N) \quad \text{(in 2D)}.$$

The step size h rapidly diminishes in size when mesh refinement steps are applied. Classic direct solvers, like Gaussian elimination or various decompositions, become impractical for very large node numbers because the computational cost is of order $O(N^2)$ or even higher. This is due to fill-in during the factorization of the matrix K that destroys sparsity and thus increases memory usage enormously. However, there are also direct solver that exploit sparsity to a certain extend and can solve systems of moderate size.

Iterative solvers like the preconditioned conjugate gradient method rank among the most efficient solvers for PDE problems and avoid these problems by exploiting the sparsity of K and compensating the high condition number with a preconditioner. Therefore, memory usage can be kept at $O(N)$ and with good preconditioners like multigrid or BPX, the computational times too. The multigrid method as a preconditioner is discussed in the next section.

Let u_k be the approximate solution in the k-th step, then we have

$$\|\underline{u} - u_k\|_K \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k \|\underline{u} - u_0\|_K ,$$

meaning that the convergence rate significantly depends on the condition number κ, [3, chapter 4.3]. To accelerate the convergence, a preconditioner C^{-1} is used such that $\kappa(C^{-1}K) \ll \kappa(K)$ in which way the iteration number is reduced to obtain a sufficiently accurate approximate solution.

Multigrid method

Multigrid methods are fast and efficient algorithms to solve systems of linear equations that arise from PDEs. In this work a simple version is used as a preconditioner, that is only a V-cycle

is performed as a preconditioner step. We briefly introduce the basic idea of it and how its components smoothing, restriction and prolongation work.
The idea bases on the observation that classic iterative solvers, like the Gauss-Seidel method, smooth the errors of an approximate solution. In the process high frequency errors are more strongly damped than low frequency errors, [3, chapter 5.1]. If one considers the latter type of error on a coarser mesh, its relative frequency increases, thus the damping will be stronger if the smoothing is applied on the coarser mesh. By doing this the high and low frequency errors can be damped and the multigrid method works efficiently on a wide spectrum of error components. This motivates the following approach:
A few smoothing steps are applied for the residual on the current mesh, i.e. current level of refinement. Then the remaining residual is restricted to the next coarser level and the smoothing steps are applied again. This procedure is repeated until the coarsest level is reached. The node number is usually quite small on this level, so a direct solver can be used for the system of equations. The gained correction is interpolated to the next finer levels where smoothing steps are applied each time again. Thus, the multigrid method consists of three main ingredients:

- Smoothing: here a Gauss-Seidel sweep, reduces the high frequency errors.
- Restriction: transfers the residual error to a coarser level.
- Prolongation: interpolates the correction to a finer level.

Smoothing

We briefly introduce the Gauss-Seidel method an how it is implemented for the multigrid preconditioner. Let the system of linear equations

$$K\ x\ =\ f \tag{2.19}$$

be given. The matrix K is decomposed into a lower triangular matrix L and a strictly upper triangular matrix U,

$$K = L + U, \qquad \text{where} \qquad L_{ij} = \begin{cases} k_{ij} & \text{if } i \geq j \\ 0 & \text{else} \end{cases}, \qquad U_{ij} = \begin{cases} k_{ij} & \text{if } i < j \\ 0 & \text{else} \end{cases}.$$

The system of linear equations (2.19) may be rewritten as

$$Lx = f - Ux.$$

Let $x^{(0)}$ be an initial guess. Then $x^{(k+1)}$ can be iteratively evaluated by

$$x^{(k+1)} = L^{-1}(f - Ux^{(k)}),$$

what is done by the algorithm 1.

Algorithm 1 Smoothing $x^{(k+1)} = L^{-1}(f - Ux^{(k)})$

$x := x^{(k)} = [x_1, \ldots, x_{2N}]^T$ is given
for $i = 1$ to 2N **do**
  $x_i := (f_i - \sum_{j \neq i} k_{ij} x_j)/k_{ii}$
end for
return $x^{(k+1)} := x$

It is required that the preconditioner C^{-1} is symmetric and positive definite. To achieve that, the Gauss-Seidel iteration operator and its transpose is applied. The action of the transpose is achieved by reversing the order of i in the smoothing algorithm 1: "for $i = 2$N to 1".

Prolongation and Restriction

The choice of the transfer operators of the residual r and correction δ between the different mesh levels influence the convergence rate of the multigrid method. Let $r^{(l)}$ and $\delta^{(l)}$ be on the l-th level and P_{l-1}^{l} the prolongation matrix and R_l^{l-1} the restriction matrix that perform the prolongation and restriction between the $l-1$-th and l-th level,

$$\delta^{(l)} = P_{l-1}^{l}\delta^{(l-1)}, \qquad r^{(l-1)} = R_l^{l-1} r^{(l)}.$$

Conforming finite elements the prolongation P_{l-1}^{l} and the restriction R_l^{l-1} arise canonically, [3, chapter 5.1]. The matrix P_{l-1}^{l} is not really formed and multiplied with $\delta^{(l-1)}$ but rather its action is locally performed for each node based on hierarchy information stored together with the mesh. For simplicity of presentation we restrict the discussion to the transition from level $(l-1)$ to level l. Let T denote an arbitrary element on the level $(l-1)$ with child elements on level l. Since the child elements have nodes that may not belong to the father element T, we need to interpolate $\delta|_T$ in these child nodes. Let c_j be a child node that belongs to one of the child elements. The function $\delta|_T$ can be evaluated in every point on T with the help of the values in the father nodes f_i,

$$\delta(c_j) = \sum_i \delta(f_i)\varphi_i(c_j).$$

Since the entries of $x^{(l-1)}$ are the function values $x(f_i)$ by definition (2.13) and for the mapping there holds $\varphi_{(c_j)} = \widehat{\varphi}_{(\widehat{c}_j)}$, we get

$$x(c_j) = \sum_i [x^{(l-1)}]^i \ \widehat{\varphi}_i(\widehat{c}_j), \tag{2.20}$$

where $[x^{(l-1)}]^i \in \mathbb{R}^2$ are the components in x_1- and x_2-dimension in the father node i. It can be easily seen that this rule also includes the copying of the father nodes that belong to both levels. This allows us to prolong $\delta^{(l-1)}$ to $\delta^{(l)}$ by using only itself and the values of the basis functions in the child nodes on the master element.

This method is used in a similar way for the restriction matrix which is chosen to be the transpose of the prolongation matrix, [3, chapter 5.1]. Let $r^{(l)}$ be a vector representation of the residual functional that is restricted to the $(l-1)$-th level by multiplying the restriction matrix R_l^{l-1},

$$r^{(l-1)} = R_l^{l-1} r^{(l)}.$$

For each child node on the l-th level, the residual in the node is distributed over the father nodes on the $l-1$-th level. Again, the basis functions are used to determine the weights of this distribution. Let f_i be a father node on the l-th level and $r|_{T_1 \dots T_n}$ be given on the child elements with the nodes c_j on the l-th level. The residual in the father node on the $(l-1)$-th level is

$$r(f_i) = \sum_j r(c_j)\varphi_i(c_j).$$

With the same arguments as in (2.20), this simplifies to

$$r(f_i) = \sum_j [r^{(l)}]^i \ \widehat{\varphi}_i(\widehat{c}_j). \tag{2.21}$$

For the father nodes that are on both levels, this means a copying of the function value in this node plus the weighted contributions from the child nodes. Again, the residual can be restricted by using the values of the basis functions in the child nodes on the master element.

3. Shape Optimization

Once we can solve the Lamé problem we can face the problem of shape optimization. The task is to find a domain such that a given goal function is optimized. In this chapter we discuss an approach to solve this numerically, based on the approach from [8].

Parametrization of the shape

By shape, we understand the geometrical shape of the boundary $\partial\Omega$. The shape of $\partial\Omega$ can be interpreted locally as isomorph to a function, which has to be of a certain degree of smoothness in order to guarantee well posed PDE problems. Thus the shape is an object from infinite dimensional space (e.g. the function space $C^1[a, b]$). Numerical optimization requires a description of the shape with a finite set of parameters. Obviously, we cannot describe all possible shapes with a finite set but it is possible to approximate it. Bézier splines are a good approach because they offer a great range of useful shapes and it is one of the parameterizations that are used in manufacturing as well. A Bézier spline is a parametric curve described by cubic segments

$$x(t) = \sum_{i=0}^{3} x_i b_i^3(t) \quad , \quad t \in [0, 1],$$

where $x_i \in \mathbb{R}^2$ are the control points and

$$b_i^n(t) := \binom{n}{i} t^i (1-t)^{n-i}$$

denotes the Bernstein polynomials. Figure 3.1 shows an example of a Bézier spline and illustrates the parametrization. The points x_0 and x_3 are start and ending point of the segment and $\overline{x_0 x_1}$ and $\overline{x_2 x_3}$ define the tangents in these points.

One can increase the number of Bézier splines if a greater variety of shapes is required which increase the parameters for the numerical optimization too. We denote the vector holding the shape parameters with $p \in \mathbb{R}^n$ and the set of all admissible vectors of shape parameters with $P \subset \mathbb{R}^n$.

As we are planning to use adaptive mesh design, special attention has to be devoted to determine the value of the curve parameter t corresponding to nodes on the shape segment. Since the parameter t is not the arc length of the spline, an inhomogeneous distribution can cause problems

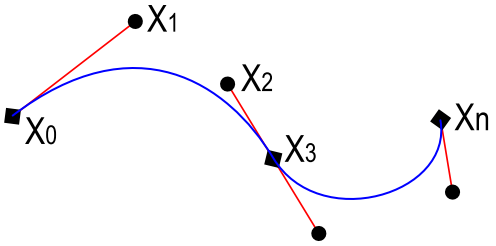


Figure 3.1.: Segment of a Bézier spline with its control points, start and end point

in the restriction and prolongation of the multigrid method, resulting even in divergence of the method. Since the restriction and prolongation are completely described on the master element to simplify the calculation, it is required that the child nodes are fixed on the master element. As it is illustrated in Figure 3.2, the mid-node of an edge does not need to be close to the point $x(t = 0.5)$ of a spline segment. This means that the restriction and prolongation are computed in the wrong point. To avoid that, we set the parameter t for each each point such that the point on the spline is close to the reference point (e.g. a child node) of the edge. This also gives an almost uniform distribution on the spline.

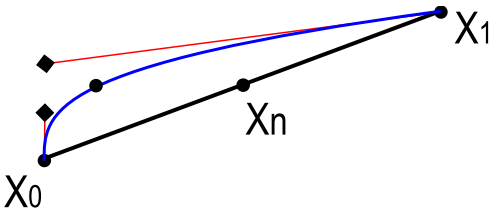


Figure 3.2.: The mid node $x(t = 0.5)$ of a Bézier splines may not be near the mid node of the edge.

A simple way to achieve this is a bisection method. We begin with the start x_0 and end x_1 and their t-values t_0 and t_1. There is a reference point x_m on the edge $\overline{x_0 x_1}$ for which we want to find a point on the spline and its t-value. To obtain an inexpensive algorithm, we consider the proportion between the two distances,

$$\omega^* := \frac{|x_m - x_0|_2^2}{|x_m - x_1|_2^2} \quad \text{and} \quad \omega(t) := \frac{|x(t) - x_0|_2^2}{|x(t) - x_1|_2^2}. \tag{3.1}$$

The ratio ω is the ratio between the actual euclidean distances. Our goal is to find a point $\tilde{x}_m = x(\tilde{t})$ on the spline that keeps this ratio ω^* approximately which can be achieved by Algorithm 2.

We remark that this approach with the Bézier splines as parametric shapes does not allow topological changes. However, desired holes can be added to the initial mesh and their shape may change during the optimization.

Algorithm 2 Bisection for $x(t)$ on the Interval $[t_0, t_1]$

Input: $tol \in (0,1), t_0, t_1, x_0, x_1, x_m$ are given
$t := (t_0 + t_1)/2$
$h := (t_1 - t_0)/2$
Evaluate $\omega^*, \omega(t)$ by (3.1)
while $\omega(t) < \omega^* tol$ or $\omega(t) > \omega^*/tol$ **do**
  $h := h/2$;
  if $\omega(t) < \omega^* tol$ **then**
    $\{x(t)$ is too close to $x_1\}$
    $t := t + h$ and recompute $\omega(t)$
  end if
  if $\omega(t) > \omega^*/tol$ **then**
    $\{x(t)$ is too close to $x_2\}$
    $t := t - h$ and recompute $\omega(t)$
  end if
end while
$\tilde{x}_m := x(t)$

Examples for the performance function

In this work we consider to find a compromise between the material volume or area and the deformation of the structure. The area can simply be described by the integral $\int_\Omega dx$. There are several formulations for the deformation, for example integrals of the displacement u over subdomains $\Omega' \subset \Omega$, parts of the boundary $\Gamma' \subset \Gamma$ or certain points x_i:

$$\int_{\Omega'} |\mathrm{u}(x)|^2 \, dx, \quad \int_{\Gamma'} |\mathrm{u}(x)|^2 \, dx, \quad \sum_i |\mathrm{u}(x_i)|^2 .$$

We need to remark that Ω' should be independent of the shape parameters because otherwise the optimization tends to shrink Ω' rather than reduce $|u(x)|^2$, the same holds true for parts of the boundary Γ'. The third quantity, the evaluation in certain given points, offers a good flexibility and is a generalization of the cubature formulas that are used to compute the other integrals. The given points should be part of the mesh for easier computation. Another important performance function is the potential energy given by

$$\int_\Omega \frac{1}{2} \varepsilon(\mathrm{u}) : C\varepsilon(\mathrm{u}) \, dx + \int_{\Gamma_N} g \cdot \mathrm{u} \, dx,$$

i.e. the deformation energy. It is desirable to minimize both quantities, deformation and area, simultaneously but less material leads to larger deformations usually. In order to merge both quantities into one performance function, we multiply them with weight factors α and β to gain

more flexibility, for example

$$\tilde{I}(\Omega) = \alpha \sum_i |\mathrm{u}(x_i)|^2 + \beta \int_\Omega \, dx.$$

The goal is now to minimize the performance function $I = I(\Omega)$. The shape parameters p directly describe the domain Ω thus we can write

$$\tilde{I}(\Omega(p)) =: I(p) \to \min \quad \text{with} \quad p \in P \subset \mathbb{R}^n. \tag{3.2}$$

3.1. Optimization method

The prime task is the optimization of (3.2). Assuming that I is a twice differentiable function in P, we can use a Quasi-Newton method to solve the optimization problem. To introduce the idea and to keep notation simple, we assume that $P = \mathbb{R}^n$, that is that any direction Δp will not lead out of P. Instead of using the usual gradient descent $-\nabla I$ we also consider the second derivative,

$$I(p_k + \Delta p) \approx I(p_k) + \nabla I(p_k)^T \Delta p + 1/2 \Delta p^T H(p_k) \Delta p,$$

where H is an approximation of the Hessian matrix of I. We differentiate this approximation with respect to Δp and get the gradient

$$\nabla I(p_k + \Delta p) \approx \nabla I(p_k) + H(p_k) \Delta p.$$

By setting this gradient to zero we obtain

$$\Delta p = -H^{-1}(p_k) \nabla I(p_k),$$

Newtons method for optimization. As we will see later, calculating an analytic Hessian is very expensive. Therefore we use the Inverse Broyden-Feltcher-Goldfarb-Shanno (BFGS) method to update $B = H^{-1}$ in each step, so we do not need to solve this system of linear equations. Starting with the identity matrix as B_0, Algorithm 3 performs the optimization process.
If P is bounded and Δp can lead out of it, the direction Δp has to be determined by minimizing the quadratic local model with linearized constraints (SQP)

$$1/2 \Delta p^T H(p_k) \; \Delta p + \Delta p^T \; \nabla I(p_k) \longrightarrow \min.$$

The Quasi-Newton method offers a faster convergence than the deepest decent method, so less

Algorithm 3 Quasi-Newton methd for $I(p) \to \min$

Input: p_0, B_0, TOL are given
$k = 0$
while $|\nabla I(p_k)| > TOL$ **do**
  Gain the direction $p_k = -B_k \nabla I(p_k)$
  Obtain a stepsize α_k by a line search
  Update $p_{k+1} = p_k + \alpha_k p_k$
  Update B_{k+1} by the inverse BFGS method
  $k = k + 1$.
end while

iterations are required to get the optimal solution. The dimension of the optimization problem is very small compared to the node number of the FE mesh. Therefore the computational costs for the additional calculations in the Quasi-Newton method are negligible, but since less Lamé problems need to be solved, the overall computational cost is reduced.
As it can be seen, only the function value $I(p_k)$ and the gradient $\nabla I(p_k)$ with respect to the shape parameters are required for the algorithm that is performed by the program *DONLP2* from Peter Spellucci, [11] and [10]. There is also a matrix-free version of the BFGS update in the case that the number of shape parameters is very large [7].

3.2. Discrete Adjoint Method

Determining the gradient ∇I analytically is very difficult. A simple approach to determine ∇I numerically is a standard difference quotient, that is to obtain an approximation of the gradient by evaluating $\frac{\partial I}{\partial p_i}(p) \approx \frac{I(p+he_i)-I(p)}{h}$, where e_i is the i-th vector of the standard basis. However, the drawback of this approach is the great computational cost because for each component of the gradient it is requires to solve a Lamé problem, including assembling and solving the system of linear equations. The latter needs to be done with a relatively high accuracy because the stepsize h should be sufficiently small to obtain a good approximation for the gradient ∇I. Also, the choice of h is difficult because too large values of h result in a vague approximation while too small values cause numerical cancellation that perturbs the gradient too.
An attractive alternative is the discrete adjoint method (DAM) [8], whose idea and derivation we repeat here for completeness. Let I be a given performance function that is differentiable with respect to the shape parameters and discretized by cubature formulas, for example

$$\begin{aligned}\tilde{I}(\Omega) &= \alpha \sum_i |u(x_i)|^2 + \beta \int_\Omega dx \\ \approx I(p) &= \sum_i |u(x_i)|^2 + \beta \sum_{T\in\mathbb{T}} \sum_k |\det(J(\widehat{x}_k))|\end{aligned}$$

This discretisized performance function $I(p)$ depends on the shape parameters p and the components of the deformation vector $\underline{u} := [u_i^r]_{(i,r)}$, which itself depend on the shape parameters,

$$I = I(\underline{u}(p), p),$$

where $\underline{u}(p)$ is implicitly given by the system

$$K(p)\ \underline{u}(p) - \underline{f}(p) =: R(\underline{u}(p), p) = 0.$$

We remark that both the stiffness matrix K and the load vector $\underline{f}$ depend on p because the integration domains in (2.9) are affected by p. Let the perturbation δp be small, so terms of higher order can be neglected as we are only looking for the first derivatives. We get

$$\delta I = \frac{\partial I}{\partial \underline{u}} \delta \underline{u} + \frac{\partial I}{\partial p} \delta p. \tag{3.3}$$

The partial derivative $\frac{\partial}{\partial \underline{u}}$ refers to the first derivate with respect to the components of $\underline{u}$. With the condition that $\underline{u} + \delta\underline{u}$ has to solve the perturbed problem, it yields that

$$\delta R = \frac{\partial R}{\partial \underline{u}} \delta \underline{u} + \frac{\partial R}{\partial p} \delta p = 0. \tag{3.4}$$

With this equation, δu is implicitly given. As $\delta R = 0$, it can be multiplied by an arbitrary term Ψ^T and the new product is subtracted from the right hand side of (3.3). Rearranging gives

$$\begin{aligned}\delta I &= \frac{\partial I}{\partial \underline{u}} \delta \underline{u} + \frac{\partial I}{\partial p} \delta p - \Psi^T \left(\frac{\partial R}{\partial \underline{u}} \delta \underline{u} + \frac{\partial R}{\partial p} \delta p \right) \\ &= \left(\frac{\partial I}{\partial \underline{u}} - \Psi^T \frac{\partial R}{\partial \underline{u}} \right) \delta \underline{u} + \left(\frac{\partial I}{\partial p} - \Psi^T \frac{\partial R}{\partial p} \right) \delta p. \end{aligned} \tag{3.5}$$

Now, we want to avoid the evaluation of $\delta\underline{u}$ because it is expensive and has to be done for each pertubation direction δp seperately. Therefore, we choose Ψ^T such that the first term of (3.5) vanishes,

$$\begin{aligned} \frac{\partial I}{\partial \underline{u}} - \Psi^T \frac{\partial R}{\partial \underline{u}} &\stackrel{!}{=} 0 \\ \Leftrightarrow \left[\frac{\partial R}{\partial \underline{u}}\right]^T \Psi &= \left[\frac{\partial I}{\partial \underline{u}}\right]^T, \end{aligned}$$

so δI can be evaluated using only Ψ and δp. As Ψ is independent of δp, the gradient becomes

$$\nabla I = \frac{\partial I}{\partial p} - \Psi^T \frac{\partial R}{\partial p}.$$

The chain rule with respect to the node positions s_i allows the evaluation of the partial derivatives $\frac{\partial}{\partial p}$,

$$\frac{\partial}{\partial p} = \sum_i \frac{\partial s_i}{\partial p} \frac{\partial}{\partial s_i}.$$

The derivatives $\frac{\partial}{\partial u}$ and $\frac{\partial}{\partial s}$ are calculated directly from the cubature formulas as shown in the next section. Casteljau's algorithm determines the node positions of the nodes on the shape segments and can be differentiated to obtain $\frac{\partial s}{\partial p}$, [8,].

The cubature formulas for I and R

To gain the required derivatives for the discrete adjoint method, we have to analyze the cubature formulas for I and R. To simplify notation in the following, we drop the (x_k) of the functions since most of them are evaluated in the integration points x_k. The partial derivative $\frac{\partial}{\partial x_r}(\cdot)$ with respect to the r-th dimension on the world element is shortened by $(\cdot)_{,r}$ and $[.]_r$ denotes the r-th component of a vector. For example, $[u]_{x_1,x_2}$ is the partial derivative of the x_1-component of u with respect to x_2. We recall that φ_i is the basis function in the node i while φ_i^r is the test function that consists of φ_i in the r-th component and zero functions in the other components. The test functions for edge integrals and domain integrals are the same yet the integration points are different. We do not use a different notation because it is obvious whether it is an integral over an edge or a domain. For an exemplary discretized performance function we have

$$\begin{aligned} I &= \alpha \sum_i |u(x_i)|^2 + \beta \int_\Omega dx \\ &= \alpha \sum_i |u(x_i)|^2 + \beta \sum_{T \in \mathbb{T}} \sum_k w_k \left|\det(\mathrm{J})\right|. \end{aligned} \tag{3.6}$$

For the parts $K\underline{u}$ and $\underline{f}$ of the residuum $R = K\underline{u} - \underline{f}$ we obtain

$$
\begin{aligned}
[Ku]_{(i,r)} &= \sum_{(j,s)} K_{(i,r),(j,s)} u_{(j,s)} = \sum_{(j,s)} a(\varphi_j^s, \varphi_i^r) u_{(j,s)} \\
&= a(\sum u_{(j,s)} \varphi_j^s, \varphi_i^r) = a(u, \varphi_i^r) \\
&= \int_{\Omega'} \lambda(\nabla \cdot u)(\nabla \cdot \varphi_i^r) + 2\mu \sum_{i,j=1}^{2} \varepsilon_{ij}(u) \varepsilon_{ij}(\varphi_i^r) \; dx \\
&= \sum_{el} \sum_{k} w_k \left| \det(\mathrm{J}) \right| \left\{ \lambda([u]_{x,x} + [u]_{y,y}) \varphi_{i,r} + 2\mu \left([u]_{r,r} \varphi_{i,r} + \frac{1}{2}([u]_{x,y} + [u]_{y,x}) \varphi_{i,\overline{r}} \right) \right\}, \\
-\left[\underline{f}\right]_{(i,r)} &= b(\varphi_i^r) = \int_{\Gamma_N} [g]_r \cdot \varphi_i \;\; \| t(\widehat{x}_k) \|_2 = \sum_{bd} \sum_{k} w_k \;\; [g]_r \;\; \varphi_i \;\; \| t(\widehat{x}_k) \|_2 \,,
\end{aligned}
$$

where

$$
\overline{r} = \begin{cases} 2 & \text{for} \quad r = 1 \\ 1 & \text{for} \quad r = 2. \end{cases}
$$

The derivatives with respect to the components of $\underline{u}$

Assuming that the points of interest x_i are also nodes of the mesh, the derivatives of the cubature formulas for the discretized performance function (3.6) with respect to the components of $\underline{u}$ are

$$
\begin{aligned}
\frac{\partial I}{\partial \left[\underline{u}\right]_{(j,s)}} &= \alpha \sum_i \frac{\partial}{\partial \left[u_j\right]_s} \left| u(x_i) \right|^2 \\
&= \alpha \sum_i 2 \left[\underline{u}\right]_{(i,s)} \delta_{ij}.
\end{aligned}
$$

The residual $R = K\underline{u} - \underline{f}$ is obviously linear with respect to $\underline{u}$ as K and $\underline{f}$ are independent of $\underline{u}$, thus

$$
\left[\frac{\partial R}{\partial \left[\underline{u}\right]_{(j,s)}} \right]_{(i,r)} = K_{(i,r),(j,s)}, \quad \text{or} \quad \frac{\partial R}{\partial \underline{u}} = K.
$$

The derivatives with respect to the node positions x_i

Following the approach in [9], one can observe that the derivatives with respect to the node positions x_i can be reduced to the derivatives of the four terms

$$
\left| \det(\mathrm{J}) \right|, \quad \varphi_{i,r}, \quad \| t(\widehat{x}_k) \|_2 \,, \quad [u]_{t,r},
$$

because the remaining terms do not depend on the node positions. We remark that it is a matter of the modeling whether the value of the load g depends on the shape parameters or not. In this work we assume that g does not, otherwise corresponding terms have to be added. According to [9] we have

$$\begin{aligned} \frac{\partial \varphi_{i,r}}{\partial [x_j]_s} &= -\varphi_{j,r}\varphi_{i,s} \\ \frac{\partial |\det(\mathrm{J})|}{\partial [x_j]_s} &= \left[|det(J)|\, J^{-T} \widehat{\triangle} \widehat{\varphi_j} \right]_s . \end{aligned}$$

For the basis function φ_i, and therefore also for u, the derivatives with respect to the node positions vanish,

$$\begin{aligned} \frac{\partial \varphi_i(x_k)}{\partial [x_j]_s} &\equiv 0, \\ \frac{\partial u(x_k)}{\partial [x_j]_s} &\equiv 0, \end{aligned}$$

as all functions are evaluated at x_k whose position ob the reference element is independent of $[x_j]_s$.
Differentiating the definition of $\underline{u}$(2.14) we obtain

$$\frac{\partial [u]_{t,r}}{\partial [x_j]_s} = -\varphi_{j,r}[u]_{t,s},$$

and with the chain rule and the tangent vector t from (2.18) it yields

$$\frac{\partial \left\| t(\widehat{x}_k) \right\|_2}{\partial [x_j]_s} = \frac{1}{\left\| t(\widehat{x}_k) \right\|_2} [t(\widehat{x}_k)]_s \, \varphi_j.$$

With these formulas the derivatives of the performance function can be written as

$$\begin{aligned} \frac{\partial I}{\partial [x_j]_s} &= \frac{\partial}{\partial [x_j]_s} \left[\sum_{T \in \mathbb{T}} \sum_k w_k \beta \, |\det(\mathrm{J})| \right] \\ &= \sum_{T \in \mathbb{T}} \sum_k w_k \beta \frac{\partial \, |\det(\mathrm{J})|}{\partial [x_j]_s}. \end{aligned}$$

For the benefit of clarity the residual R is split into three parts, the right hand side, the λ- and the μ-part, the whole derivation can be seen in Appendix A.1.

$$
\begin{aligned}
& + \frac{\partial}{\partial \left[x_j\right]_s} \left[\int\limits_{\Omega'} \lambda (\nabla \cdot u)(\nabla \cdot \varphi_i^r) \; dx \right] \\
= \quad & |\det(\mathrm{J})| \left\{ -([u]_{x,x} + [u]_{y,y}) \varphi_{j,r} \varphi_{i,s} + (-\varphi_{j,x} [u]_{x,s} - \varphi_{j,y} [u]_{y,s}) \varphi_{i,r} \right\} \quad ,
\end{aligned}
$$

$$
\begin{aligned}
& + \frac{\partial}{\partial \left[x_j\right]_s} \left[\int\limits_{\Omega'} 2\mu \sum_{i,j=1}^{2} \varepsilon_{ij}(u) \varepsilon_{ij}(\varphi_i^r) \; dx \right] \\
= \quad & \sum_{el} \sum_{k} w_k 2\mu \left[\frac{\partial \left|\det(\mathrm{J})\right|}{\partial \left[x_j\right]_s} \left([u]_{r,r} \varphi_{i,r} + \frac{1}{2} ([u]_{x,y} + [u]_{y,x}) \varphi_{i,\overline{r}} \right) \right. \\
& \qquad + |\det(\mathrm{J})| \left(-[u]_{r,r} \varphi_{j,r} \varphi_{i,s} - \varphi_{j,r} [u]_{r,s} \varphi_{i,r} \right. \\
& \qquad \left. \left. - \frac{1}{2} ([u]_{x,y} + [u]_{y,x}) \varphi_{j,\overline{r}} \varphi_{i,s} - \frac{1}{2} (\varphi_{j,y} [u]_{x,s} + \varphi_{j,x} [u]_{y,s}) \varphi_{i,\overline{r}} \right) \right] \quad ,
\end{aligned}
$$

$$
\begin{aligned}
& - \frac{\partial}{\partial \left[x_j\right]_s} \left[\int\limits_{\Gamma_N} [g]_r \cdot \varphi_i \;\; \left\| t(\widehat{x}_k) \right\|_2 \right] \\
= \quad & \sum_{bd} \sum_{k} w_k \;\; [g]_r \, \widehat{\varphi}_i(\widehat{x}_k) \frac{\partial \left\| t(\widehat{x}_k) \right\|_2}{\partial \left[x_j\right]_s} \quad .
\end{aligned}
$$

4. Mesh refinement

The initial mesh may not offer the necessary accuracy since the approximation due to the discretization is not sufficient. Refining the mesh uniformly the discretization allows a wider space of functions, improving the approximation of the solution. This quickly leads to very large node numbers which increases the computational cost so that even very good solvers still results in unreasonable large computational time.
A different idea is the adaptive mesh design where the mesh is refined or coarsened locally, based on an error estimator. There are several types of adaptivity, in this work we use the h-adaptivity without coarsening since it an be easily implemented in the already existing FE-code. Only the regions where the approximation is poor are refined, resulting in a much smaller number of degrees of freedom than the uniform mesh refinement. One discretized Lamé problem has to be solved in each refinement step. The current solution is interpolated to the new mesh to start with a better initial guess for the iterative solver of (2.17). The interpolated solutions are better guesses, thus fewer iterations are needed on the subsequent level.
This motivates the following algorithm: Starting with a coarse mesh, we solve the Lamé problem (including assembling and solving (2.17) and estimate the error of the solution. Next, the mesh is refined (uniform or adaptive) and the solution is interpolated on the new mesh. We now repeat solving the Lamé problem, estimation and refinement until a given refinement level is reached.

4.1. Red and Baensch-Green refinement

There are two main ways to refine a triangle, the uniform refinement and the bisection of a triangle.
The uniform refinement[1], also known as red refinement, splits a triangle into four similar triangles (compare Figure 4.1). In the case of quadratic triangles, the mid-nodes become the new corner nodes and thus are reused.
If the initial triangles are well shaped the refined triangles keep this shape. The red refinement

[1] Uniform mesh refinement means globally, that is all elements are refined, whereas uniform refinement is meant locally.

is easily applied for the uniform mesh refinement and multiplies the node number by about four in each step. However, there is a problem if one uses the red refinement for the h-adaptivity. If an element is refined but its neighbor is not, hanging nodes come into existence which (unless treated properly) cause non-conforming basis functions $\varphi_i \notin H^1(\Omega)$. Another solution would be a bisection of the neighboring element but this can lead to degenerating triangles. Also, the mesh structure may require certain features to be able to handle coarsening.

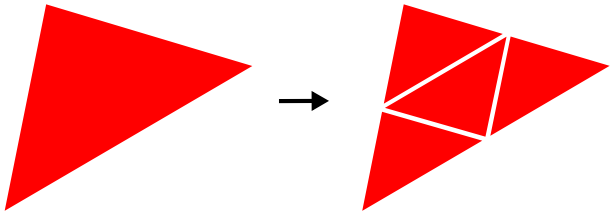


Figure 4.1.: Red refinement of a triangle

Another refinement of triangles is the bisection, also called green refinement. A given edge, called the split edge, is split into two edges and the mid node becomes a corner node of the new triangles (compare Figure 4.2).

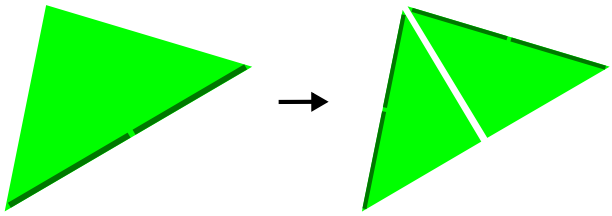


Figure 4.2.: Green refinement of a triangle with a given split edge

We want to make sure that elements do not degenerate by choosing the new split edges according to a certain pattern. An easy applied method was presented by Eberhard Baensch in [1]. The two old edges that were not split become the split edges of the two new elements (marked in dark green in Figure 4.2). Regardless of the refinement of a neighboring element, an element is only split at its split edge. To avoid hanging nodes, we continue to refine elements with hanging nodes until all hanging nodes have disappeared (compare Figure 4.3). Baensch proves in [1] that this algorithm is finite and the triangles do not degenerate. Another method is the longest-edge refinement where the longest edge is chosen to be the split edge. The Baensch-Green refinement may increase the maximal interior angle of an initial element, but degenerating elements are avoided [1].

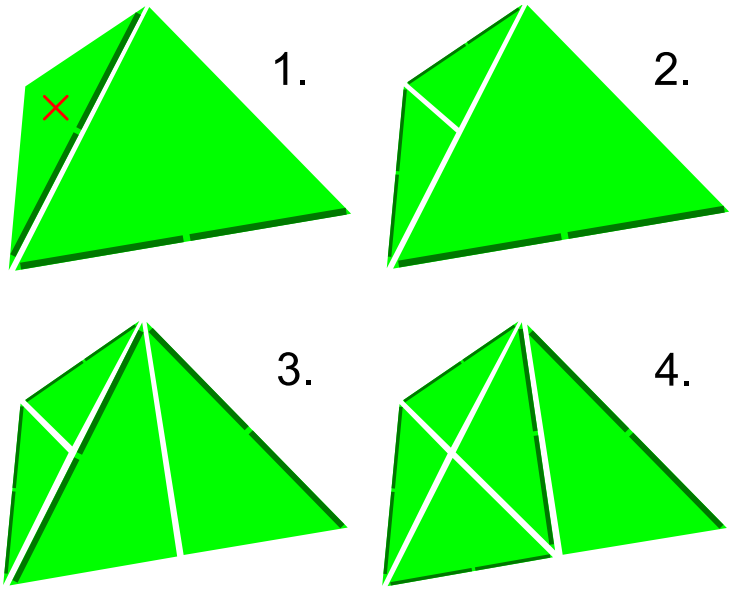


Figure 4.3.: We start with a marked element that is refined and yield a hanging node on its neighbor. Since the hanging node does not belong to the split edge the element is refined without resolving the hanging node. This is achieved in the next step when its child element is refined again and the hanging node removed.

4.2. Residual-based error estimator

Besides the refinement, the other essential part of an h-adaptive mesh design is the error estimator. It should mirror the actual error well enough to have a good guess which elements need to be refined. A common and wide-spread choice is the residual-based error estimator [2, chapter 10]. Let $e = \mathrm{u} - u_h$ be the error between the the analytic solution u of the Lamé problem (2.9) and the numerical approximation u_h. We want to estimate the energy norm of the error

$$\|e\|_a := \sqrt{a(e,e)}.$$

To derive the residual-based error estimate, let $v \in V$ be chosen arbitrarily. Writing the integration over the whole domain Ω as a sum of integrals over the elements T, we have

$$
\begin{aligned}
a(e,v) &= a(\mathrm{u},v) - a(u_h,v) = b(v) - a(u_h,v) \\
&= \int_\Gamma g \cdot v \; ds - \int_\Omega \varepsilon(v) : C\varepsilon(u_h) \; dx \\
&= \sum_{T \in \mathbb{T}} \left\{ \int_{\partial T \cap \Gamma} g \cdot v \; ds - \int_T \varepsilon(v) : C\varepsilon(u_h) \; dx \right\}.
\end{aligned}
$$

Integrating the termsn over the elements T by parts, as seen in parts Appendix (A), yields

$$
a(e,v) = \sum_{T \in \mathbb{T}} \left\{ \int_{\partial T \cap \Gamma} g \cdot v \; ds + \int_T \nabla \cdot C\varepsilon(u_h) \cdot v \; dx - \int_{\partial T} n \cdot C\varepsilon(u_h) \cdot v \; ds \right\}. \tag{4.1}
$$

In order to choose which elements are to be refined, we seek to obtain quantities that quantify the error on an element T_i and need to allocate the edge integrals to their elements. The jump discontinuity between the element T and its neighboring element T' is denoted by

$$
[C\varepsilon(u_h)] := C\varepsilon(u_h)|_{T'} - C\varepsilon(u_h)|_T \,.
$$

Distributing the jumps $[C\varepsilon(u_h)]$ of an edge to its adjoining elements, we can rewrite $a(e,v)$ as integrals that belong to a certain element,

$$
a(e,v) = \sum_{T \in \mathbb{T}} \left\{ \int_T R_h \cdot v \; dx + \int_{\partial T} r_h \cdot v \; ds \right\}, \tag{4.2}
$$

with the interior element residual $R_h := \nabla \cdot C\varepsilon(u_h)$ and the edge residuals

$$
r_h|_\gamma := \begin{cases} \frac{1}{2} n \cdot [C\varepsilon(u_h)] \,, & \text{if } \gamma \subset \partial T \setminus \partial\Omega, \\ g - n \cdot C\varepsilon(u_h), & \text{if } \gamma \subset \Gamma_N, \\ 0, & \text{if } \gamma \subset \Gamma_D. \end{cases} \tag{4.3}
$$

Since the error e satisfies the Galerkin orthogonality property $a(e, I_h v) = 0$, with I_h being the Clément interpolation operator to the subspace S_h, it can be added to (4.2), obtaining

$$
a(e,v) = \sum_{T \in \mathbb{T}} \left\{ \int_T R_h \cdot (v - I_h v) \; dx + \int_{\partial T} r_h \cdot (v - I_h v) \; ds \right\}. \tag{4.4}
$$

The Cauchy-Schwarz inequality yields

$$a(e,v) \leq \sum_{T\in\mathbb{T}} \left\{ \|R_h\|_{0,T} \|v - I_h v\|_{0,T} + \|r_h\|_{\partial T} \|v - I_h v\|_{0,\partial T} \right\}.$$

For the Clément interpolation operator there are the following inequalities, [3, chapter 3.8],

$$\begin{aligned} \|v - I_h v\|_{0,T} &\leq c h_T \|v\|_{0,\tilde{\omega}_T}, & (4.5)\\ \|v - I_h v\|_{0,\gamma} &\leq c h_\gamma^{1/2} \|v\|_{0,\tilde{\omega}_T}, & (4.6) \end{aligned}$$

with a constant c, the element diameter h_T and the edge length h_γ. The patch $\tilde{\omega}_T$ is a neighborhood of the element T. We are assuming a quasi-uniform triangulation, thus h_γ can be replaced with h_T because they have the same order of magnitude. Also, $\bigcup\{\tilde{\omega}_T : T \in \mathbb{T}\}$ overlaps the domain Ω only a finite number of times and incorporating this fact into the constant c, we can write T instead of $\tilde{\omega}_T$. Applying the Cauchy-Schwarz inequality and inserting the Clément approximations (4.5) and (4.6) gives

$$a(e,v) \leq c \left\{ \sum_{T\in\mathbb{T}} h_T^2 \|R_h\|_{0,T}^2 \|v\|_{0,T}^2 + h_T \|r_h\|_{0,\partial T} \|v\|_{0,T}^2 \right\}^{1/2}.$$

Finally, due to the coercivity of the bilinear form $a(\cdot,\cdot)$, [3, chapter 6.3], it follows that $\|v\|_{0,T} \leq C \|v\|_a$ and inserting e in place of v results in the a posteriori error estimate

$$(\|e\|_a)^2 \leq C \left\{ \sum_{T\in\mathbb{T}} h_T^2 \|R_h\|_{0,T}^2 + h_T \|r_h\|_{0,\partial T} \right\}. \quad (4.7)$$

Besides the constant C all quantities of the right-hand side can be calculated once we have an approximate solution u_h. For easier reading, the right-hand side is regrouped

$$\begin{aligned} \|e\|_a^2 &\leq C \sum_{T\in\mathbb{T}} \eta_T^2 & (4.8)\\ \text{with} \quad \eta_T^2 &= h_T^2 \|R_h\|_{0,T}^2 + h_T \|r_h\|_{0,\partial T}^2. & (4.9) \end{aligned}$$

The local quantity η_T may not necessarily be a good estimate for the true local error $|||e|||_T$ of the element T but it is its contribution to the global error.
Once we have the η_T for each element, we refine all elements whose η_T has a large influence on the global error, for example

$$\left\{ T \in \mathbb{T} : \eta_T \geq TOL * \max_{T \in \mathbb{T}} |\eta_T| \right\}, \tag{4.10}$$

where $TOL \in (0, 1)$ is a given tolerance. This has the drawback that sometimes only a few elements are refined what is not desirable because we need more refinement steps to reduce the error. More refinement steps means more memory usage for the stiffness matrices $K^{(l)}$ for each level l (required for the multigrid preconditioner) and more smoothing steps in the multigrid preconditioner than necessary. Lowering the tolerance does not need to be a good idea because regions with rather small errors may be over refined then. This can be avoided by sorting the error contributions η_T first and then choose a certain percentage beginning with the largest η_T. The drawback of doing this is that the sorting algorithm normally needs $O(N \log N)$ comparisons what may significantly increase the computational costs for large numbers of elements. However our experiments indicate that the quality of the mesh is not necessarily better due to the sorting. Another method for the Baensch-green refinement is to refine the marked elements twice, that is that the children of a marked element are also refined. This accelerates the growth of the number of nodes resulting in fewer refinement steps compared to a standard refinement. However, it may still happen that the number of new elements strongly varies from refinement step to step.

5. Numerical results

In this final chapter we present some numerical examples to show how the adaptive mesh design preforms, in particular compared to the uniform mesh refinement. Also, results from shape optimization and possible drawbacks are discussed. For all examples, the Lamé constants are chosen to be $\lambda = 1.15 \cdot 10^5$ and $\mu = 7.7 \cdot 10^4$ which would match e.g. steel. The PDE solution was computed by the FE-solver *FEINS* from R. Schneider. The adaptive mesh design including the Baensch-Green refinement and the error estimator, as well as the DAM for the Lamé problem have been implemented in *FEINS* for the purpose of this work.

5.1. Tests to compare uniform and adaptive mesh refinement

If we want to compare the uniform and adaptive mesh refinement we have to keep in mind that the accuracy of the adaptive mesh design does not improve as fast with the number of refinement steps as for the uniform mesh refinement. Thus, comparing the two refinements methods according to the number of refinement steps is not appropriate. Instead one should compare either according to overall computational effort, or for simplicity according to the number of degrees of freedom in the discretization. As discussed in section 4.2, The adaptive mesh design does not only require an error estimator that has to be evaluated, but also the local refinement may be more difficult than for a global uniform refinement. Still, the efficiency of the adaptive refinement makes up for these inconveniences. In order to test the adaptive mesh design we consider a two-dimensional disc with a crack on the left side. As shown in Figure 5.1 the lower and right edges are clamped and there is a external force acting on the right half of the upper edge.

Figure 5.2 shows the mesh after 16 adaptive mesh refinement steps with the residual-based error estimator. The singularity at the tip of the crack is strongly refined as well as the intersection point between the Dirichlet and Neumann boundary condition in the upper right corner. In addition, another singularity is at the midpoint of the upper edge, where the value of the Neumann boundary condition changes from the external force to stress free. Other regions like the lower right or upper left corner where the stresses are smaller and simpler are still coarse because the discretization is already sufficient with coarser elements.

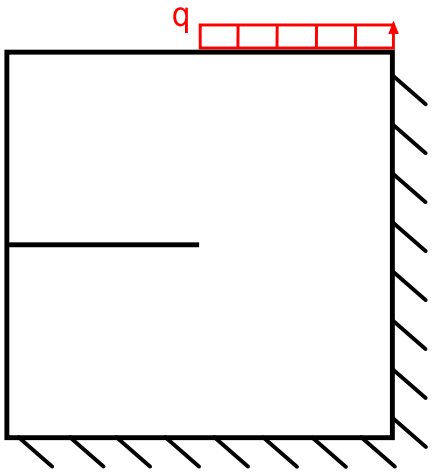


Figure 5.1.: Disc with a crack

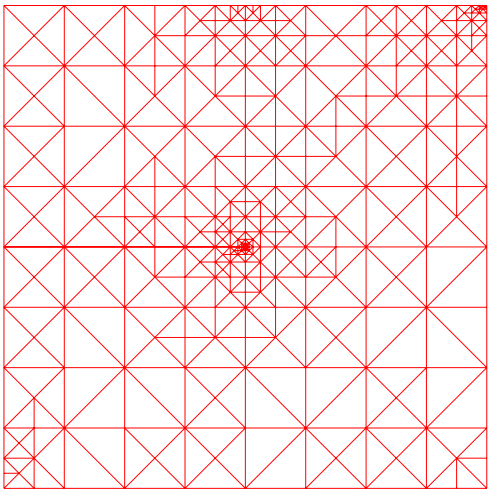


Figure 5.2.: Mesh of the disc with a crack after 16 adaptive mesh refinement steps

Convergence of a performance function I

We now investigate the convergence of an exemplary performance function, comparing uniform and adaptive mesh refinement. The computations were performed on a Intel Xeon Dual Core

CPU 3.0 GHz with 64 GB RAM on openSUSE Linux 11.1 (x86_64). The structural element is shown in Figure 5.3, being loaded with a constant external force on the upper edge and clamped on the two left ends. The performance function I is the displacement evaluated in the mid x_0 and the right corner x_1 of the upper edge,

$$I = |u(x_0)|^2 + |u(x_1)|^2 .$$

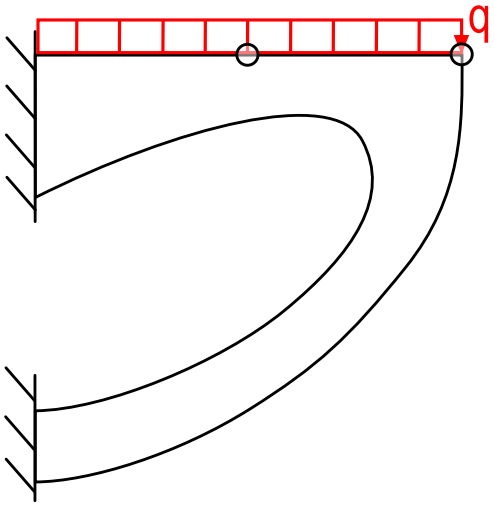


Figure 5.3.: Example of a beam with additional support

Eight levels of uniform mesh refinement steps are performed, as can be seen in Table 5.1, and 40 levels of adaptive mesh refinements, see Table 5.2. The node number N increases about the factor 4 with each uniform refinement which leads to great costs in memory usage while the adaptive mesh refinement increases N only moderately.
The iterations of the PCG-solver stay constant due to the multigrid preconditioner thus the total computational time is only $O(N)$ per refinement step. The variation in the iteration numbers for the adaptive mesh design is mainly caused by the irregular numbers of refined elements. The more elements are refined, the more iterations are required to gain the required accuracy, starting from the interpolated solution from the previous mesh. It is not unlikely to happen that large percentages of the elements ($30-50\%$) are refined in an adaptive refinement step, what is a sign that the error is well distributed among the elements. It is also possible that no iterations are performed because the interpolated solution already satisfies the specific stop criteria in the PCG-solver. However, during the computations there occured problems with the PCG-solver, the multigrid perconditioner and the adaptive mesh refinement which will be discussed later.
The residual-based error estimator provides an estimate for the global error η and the maximum of the contributions η_T. The adaptive mesh design decreases the error faster but not as steady

as the uniform mesh refinement. It may even happen that the estimate for the error becomes larger when a large percentage of elements is refined.

Since there are no analytical solutions to this model problem, the function value I is compared to a reference value of $I^* = 1.0733543293$ obtained by an 100 level adaptive mesh refinement. The relative error e_I of I compared to this reference value decreases on average with an order of magnitude of approximately $O(N^{-0.8})$ for the uniform and $O(N^{-2.2})$ for the adaptive mesh refinement. Table 5.3 shows for each uniform mesh refinement step a level of adaptive mesh design that yields a comparable relative error of the performance function value.

level	N	$I(p)$	e_I	$e_{\nabla I}$	estimate	max$\{\eta_T\}$	time (s)	iter	order
0	493	1.0621724	1.0E-02	0.0990	6.65E+03	2.34E+03	0.0369	1	
1	1,785	1.0708423	2.3E-03	0.0723	1.88E+03	9.07E+02	0.0678	5	-1.16
2	6,769	1.0725388	7.6E-04	0.0532	6.77E+02	3.75E+02	0.2550	4	-0.84
3	26,337	1.0730578	2.8E-04	0.0390	2.71E+02	1.58E+02	1.1537	4	-0.74
4	103,873	1.0732439	1.0E-04	0.0286	1.14E+02	6.69E+01	5.2992	5	-0.72
5	412,545	1.0733129	3.9E-05	0.0208	4.84E+01	2.85E+01	23.4874	5	-0.71
6	1,644,289	1.0733387	1.5E-05	0.0151	2.08E+01	1.22E+01	98.6824	5	-0.70
7	6,565,377	1.0733483	5.6E-06	0.0108	8.92E+00	5.26E+00	400.8150	5	-0.69
8	26,237,953	1.0733518	2.3E-06	0.0076	3.84E+00	2.27E+00	1655.5513	5	-0.65

Table 5.1.: Convergence of the function value I for uniform mesh refinement for the example from Figure 5.3

Discontinuity due to the meshing

In this work, the mesh generator *Triangle* from Jonathan Richard Shewchuk is used to generate a mesh based on a given boundary which in return depends on the shape parameters. In the optimization, a remshing is performed for each shape parameter, which may cause discontinu-

level	N	$I(p)$	e_I	$e_{\nabla I}$	estimate	max$\{\eta_T\}$	time (s)	iter	order
0	493	1.0621724	1.0E-02	0.0990	6.65E+03	2.33E+03	0.0369	1	
5	1,053	1.0717514	1.5E-03	0.0554	1.72E+03	3.59E+02	0.1910	6	-2.56
10	2,097	1.0728899	4.3E-04	0.0318	6.96E+02	6.74E+01	0.5476	4	-1.80
15	3,181	1.0730989	2.4E-04	0.0230	3.61E+02	5.68E+01	1.1327	2	-1.44
20	4,301	1.0732776	7.2E-05	0.0167	2.32E+02	1.74E+01	2.3581	2	-3.99
25	5,761	1.0732992	5.1E-05	0.0130	1.62E+02	7.84E+00	4.1975	2	-1.13
30	8,810	1.0733370	1.6E-05	0.0093	9.02E+01	6.41E+00	7.7245	1	-2.73
35	13,013	1.0733444	9.2E-06	0.0080	5.15E+01	1.89E+00	12.9089	3	-1.43
40	17,268	1.0733489	5.0E-06	0.0059	3.68E+01	2.95E+00	19.9245	0	-2.18

Table 5.2.: Convergence of the function value I for adaptive mesh refinement for the example from Figure 5.3

level (uni)	N	rel. error	time (s)	level (adap)	N	rel. error	time (s)
1	1,785	2.3E-03	0.07	5	1,053	2.1E-03	0.19
2	6,769	7.6E-04	0.26	9	1,946	7.6E-04	0.50
3	26,337	2.8E-04	1.15	14	3,154	4.9E-04	1.11
4	103,873	1.0E-04	5.30	18	3,917	9.9E-05	2.13
5	412,545	3.9E-05	23.49	27	7,987	1.9E-05	5.67
6	1,644,289	1.5E-05	98.68	33	12,483	1.0E-05	10.42
7	6,565,377	5.6E-06	400.82	39	17,217	5.6E-06	19.01
8	26,237,953	2.3E-06	1655.55	50	30,303	1.4E-06	46.38

Table 5.3.: Comarision of the convergnce for the uniform and adaptive mesh refinement for the example from Figure 5.3. The levels for the adaptive mesh refinement have been chosen such that the relative error was comparable to the error of the uniform mesh refinement.

ities: If there are small changes in the shape parameters, the mesh may change discontinuously, even the number of triangles and nodes changes. This discontinuity may result in jumps in the displacement u_h which can cause artificial, discretization induced, discontinuities in the performance function and therefore problems for the optimization algorithm. Discontinuities may cause the performance function value to increase even for small changes in the descent direction indicated by the gradient. The optimization process may terminate because it cannot reduce the performance function value by the line search in the search direction.

We show these discontinuities with an example. Consider a beam with dimensions 10×2 that is clamped on the left end and loaded with an external force on the upper side. There is a hole with radius 0.9 inside the beam and its center is located at $x_m(t) = 1 + 8t$ along the length of the beam with the parameter $t \in [0, 1]$. With increasing t the hole moves from the left to the right, compare Figure 5.4. The performance function I is the displacement evaluated in the upper right corner, $I = |u(x_0)|^2$.

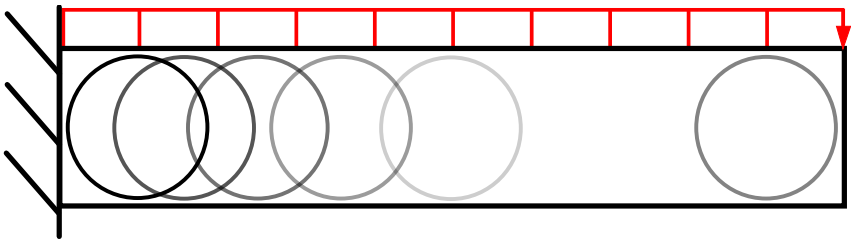


Figure 5.4.: Beam with a moving hole

Figure 5.5 shows the function value $I(t)$ depending on the parameter t (i.e the position of the hole) evaluated for the initial mesh, one and two uniform mesh refinements. On the initial mesh, level 0, the perturbation is not only discontinuous but also seems to have periodic character which

is due to the meshing. While the hole passes to the right, the mesh suddenly changes when more or less triangles are needed in the thin regions under and above the hole. This process does not need to be continuous and hence discontinuities can be caused in the performance function. Due to the convergence $\lim_{h \to 0} u_h = \mathrm{u}$ we expect that the discontinuities vanish when the mesh is refined, which is also demonstrated in Figure 5.5.

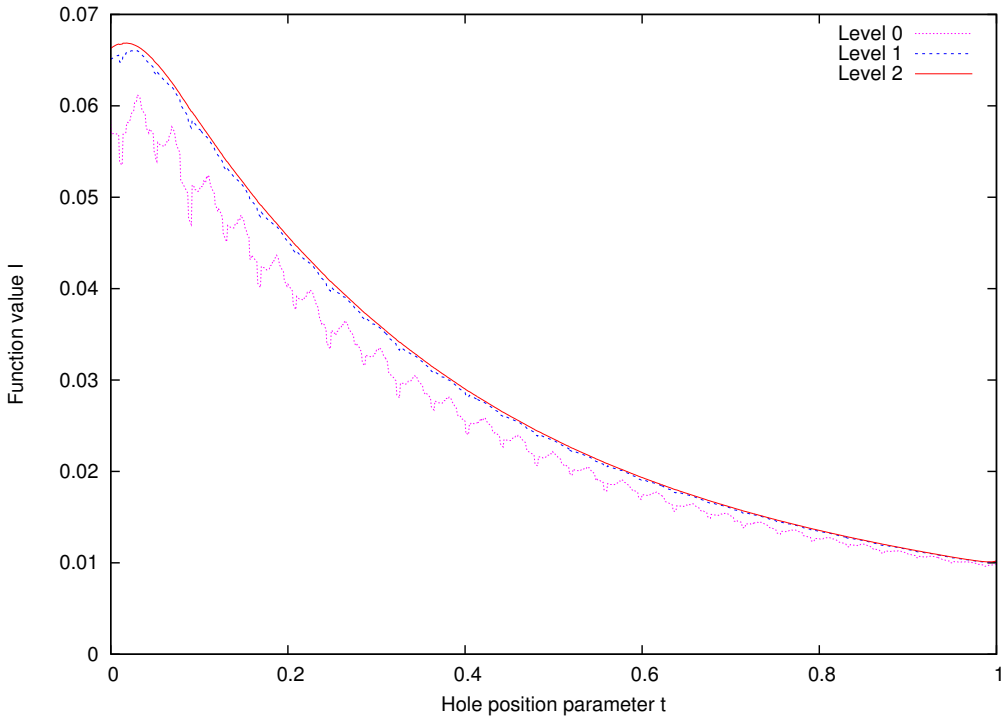


Figure 5.5.: Perfomance function depending on the parameter t for unifrom mesh refinement

The adaptive mesh design yields similar results with fewer nodes. While two uniform mesh refinements require about[1] 7956 dof, six adaptive mesh refinements give only about 922 dof and an error of comparable magnitude, see Figure 5.6.

As mentioned earlier, computing the gradient ∇I with the help of finite differences has the problem that the stepsize h has to be chosen. Too large stepsizes may be a poor approximation of the derivative while too small stepsizes give rise to numerical problems like cancellation. We demonstrate this with the bridge example in Figure 5.8 and a single component of the gradient. We investigate the derivative with respect to the x_2-coordinate of the left control point of the

[1] The node number changes slightly, depending on the position parameter t

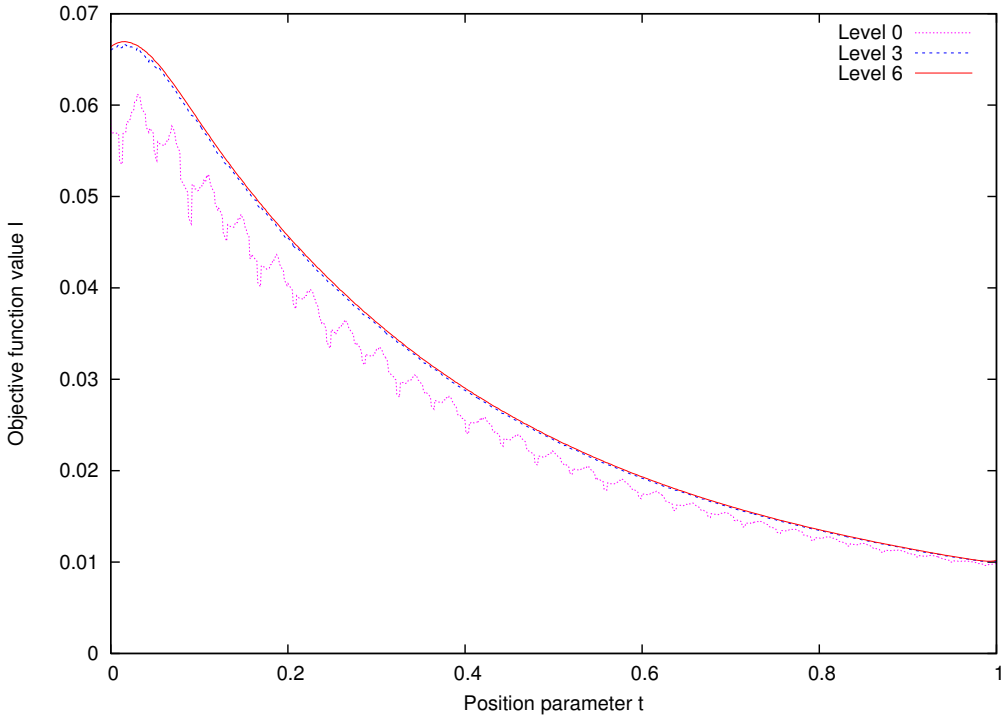


Figure 5.6.: Perfomance function depending on the parameter t for adaptive mesh refinement

left shape segment (compare the diamond in Figure 5.7). Increasing this quantity will add a small bulge on the left part of the upper edge. Table 5.4 shows the convergence of the derivative for several stepsizes. The cancellation effects, that occur on the coarse mesh for small stepsizes, can be reduced with a number of mesh refinements. This is of course a result of the artificial discontinuities described in this section. Still, the DAM already provides a very reasonable approximation of $[\nabla I]_4 = -0.0824001979$ on the coarse mesh, which clearly shows the benefits of this method.

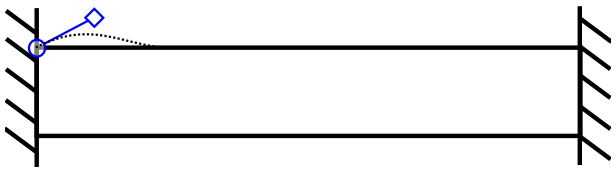


Figure 5.7.: Bézier control point of interest. We investigate the convergence of the finite difference of the derivative with respect to its x_2-component.

h	$[\nabla I]_4$ (lvl 0)	$[\nabla I]_4$ (lvl 20 adap)
1.0E-01	-0.0824012980	-0.0824091573
1.0E-02	-0.0823717628	-0.0824092121
1.0E-03	-0.0821142862	-0.0824118467
1.0E-04	-0.0795394861	-0.0823893906
1.0E-05	-0.0537914824	-0.0822128801
1.0E-06	0.2036885523	-0.0847328643

Table 5.4.: Convergence of the forth component of ∇I with decreasing stepsize h, compared on the coarse mesh and 20 adaptive mesh refinements

Problems with the multigrid preconditioner

While performing calculations with adaptive Baensch-Green refinement, the multigrid preconditioner caused divergence in the PCG solver for several examples. This is most likely caused by errors in the Implementation. However, even though considerable effort was spent, no such errors could be found, and all comparisions with hand-calcultions for small test cases indicated correctness. Usually, the Lamé problem does not require many smoothing steps in the multigrid. Yet, increasing the number of smoothing steps can fix the problem and cause convergence again. However, the divergence occurs usually only in a single level and after further adaptive refiment steps, the usual fast convergence is observed. Also, no systematic pattern could be found, which meshes are prone to this problem.

Table 5.5 shows the development in the PCG-solver for 10 adaptive refinement steps for an example that occurred during the shape optimization process of the hook model in Figure 5.14. The maximal number of iterations was set to 100 and two pre- and two post-smoothing steps were applied in the multigrid preconditioner. Already the first refinement step causes a relatively large number of iterations even though only two elements were refined. The next levels have rather normal behavior until the fifth refinement step. The residual diverge on the next four levels and on the ninth level the PCG finally converge. The usual fast convergence is observed on the tenth level again.

5.2. Tests for Shape Optimization

We now demonstrate the Shape optimization and its results with a few examples. The optimization itself is perfomed by the program *DONLP2* from Peter Spellucci. It calls the FE-solver *FEINS* for a given vector of shape parameters to compute the performance function value and gradient.

Level	N	$\|res\|$	iter
0	330	1.2E-12	1
1	334	7.1E-08	30
2	345	5.1E-08	10
3	366	4.2E-08	9
4	386	5.3E-08	6
5	415	1.7E-02	100
6	451	3.9E-07	100
7	529	7.9E-05	100
8	602	1.1E-02	100
9	779	7.6E-08	46
10	918	4.2E-08	6

Table 5.5.: Convergence of the forth component of ∇I with decreasing stepsize h, compared on the coarse mesh and 20 adaptive mesh refinements

Bridge model

Our first example is a beam of dimension 9×1 that is shown in Figure 5.8. The upper and lower edge are described by three Bézier spline segments each and a constant load acts on the upper edge. Figure 5.8 shows the initial configuration with two straight edges. We consider the performance function

$$I(\Omega) = \alpha \int_{\Omega} \frac{1}{2}\varepsilon(\mathrm{u}) : C\varepsilon(\mathrm{u})\ dx + \alpha \int_{\Gamma_N} g \cdot \mathrm{u}\ dx + \beta \int_{\Omega} dx, \tag{5.1}$$

i.e. the potential energy and area.

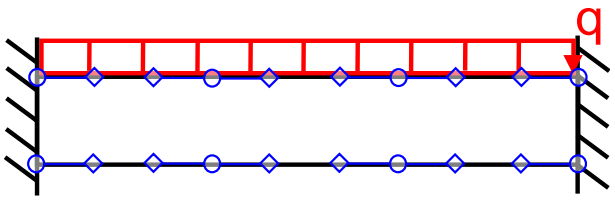


Figure 5.8.: Shape optimization: Initial configuration of the bridge model

For example, we choose the parameters $\alpha = 0.025$ and $\beta = 0.1$ and perform the shape optimization on an Intel(R) Core(TM) i7 CPU 920 with *DONLP2* . Ten adaptive mesh refinements are applied for each parameter vector solving the Lamé problem using a *LAPACK* direct solver for banded matrices. The optimization process with 202 Lamé problems solved takes about 110 seconds and the result is illustrated in Figure 5.9.

The bulge of the bridge is remarkable as it is a widespread architecture feature especially for older bridges or archways. It is also surprising because, due to the modeling that the load g does

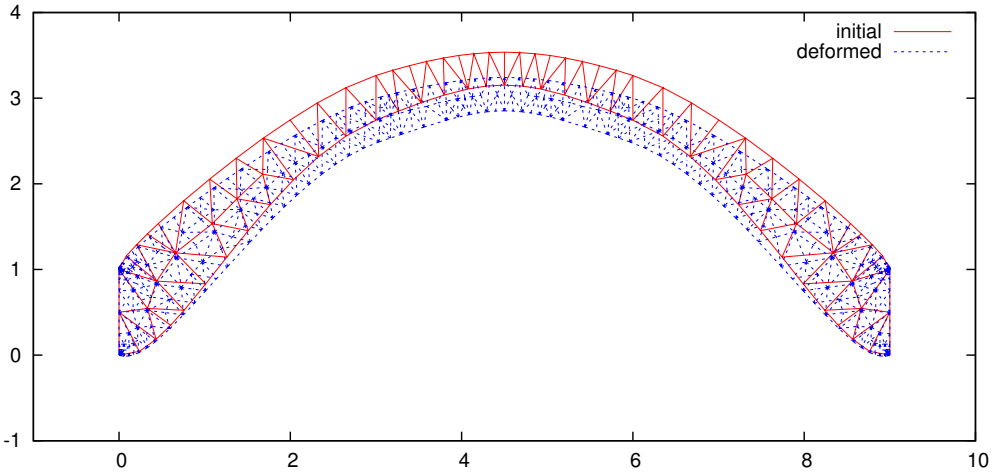


Figure 5.9.: Result of the shape optimization with $\alpha = 0.025$ and $\beta = 0.1$. The red mesh is the undeformed coarse mesh without any refinement. The blue mesh shows the ten adaptive mesh refinements and the displacement (multiplied by ten for better visibility) is added to it too. In particular, the corners, the interceptions between Dirichlet and Neumann boundary condition, are more refined than the rest of the otherwise evenly refined mesh.

not depend on the shape parameters, a longer upper edge also yields a larger total force. One might expect the optimization to have a tendency to keep the length of the upper edge small and hence also the total force. Yet, the total force acting in the result is about a fifth larger than for the initial structure. The reason lies in the differences of material behavior for normal and shear forces. For example, steel acts much stiffer when a normal force is applied, compared to a shear force of the same magnitude. Therefore the potential energy is smaller for normal forces than for shear forces. By redirecting the load as a normal force inside the structure, the structure has a smaller potential energy and can even compensate a larger total force.

We now compare the uniform and adaptive mesh refinement and its influence on the shape optimization. Tables 5.6 and 5.7 show the results of the shape optimization for several uniform and adaptive mesh refinements. The node number N refers to the node number of the

final mesh of the optimal parameters and the errors e_I and e_{p^*} show the relative error of the perfomance function value and the optimal vector of parameters compared to a reference value of $I^* = 1.2420313233$ and its parameters that were obtained by a shape optimization using 30 adaptive mesh refinements. Besides the shrinking errors, the number of evaluations of I and ∇I (that is solving a Lamé problem) decreases. This indicates that the discretized performance function becomes smoother and the gradient is better approximated which accelerates the optimization process. Discontinuities on coarse meshes (compare Figure 5.5 and 5.6) may cause the optimization to terminate without satisfying the optimality conditions. The reasons for tertimination in the optimization solver *DONLP2* is shown in the tables, where 'ok' means that a local minimum was found and the default tolerances of *DONLP2* are fulfilled, and 'slow' means "'that the progress in the preformance function value is very slow or the problem is ill-conditioned"'. This is most likely caused by insufficient accuracy or discontinuities in the performance function or its gradient.

Level	N	$I(p^*)$	e_I	e_{p^*}	evals	time (s)	Stop
0	277	1.2357135013	5.1E-03	5.2E-02	183	24	slow
1	945	1.2403272793	1.4E-03	5.7E-02	197	50	slow
2	3,457	1.2401745251	1.5E-03	1.6E-02	178	254	slow
3	12,673	1.2415616948	3.8E-04	1.1E-02	135	444	slow
4	49,409	1.2418020303	1.8E-04	9.2E-03	100	1476	ok

Table 5.6.: Convergence of the function value I and p^* for uniform mesh refinement for the bridge model

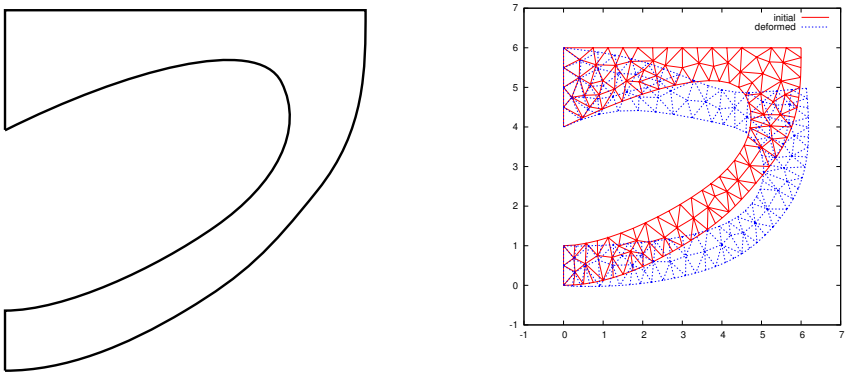
Level	N	$I(p^*)$	e_I	e_{p^*}	evals	time (s)	Stop
0	277	1.2357135013	5.1E-03	5.2E-02	183	24	slow
5	508	1.2413346125	5.6E-04	8.9E-03	163	37	slow
10	1,350	1.2418196462	1.7E-04	9.8E-03	112	67	ok
15	2,552	1.2419641986	5.4E-05	4.3E-03	103	173	ok
20	3,990	1.2420299810	1.1E-06	2.8E-03	85	383	ok

Table 5.7.: Convergence of the function value I and p^* for adaptive mesh refinement for the bridge model

Beam with additional supporter

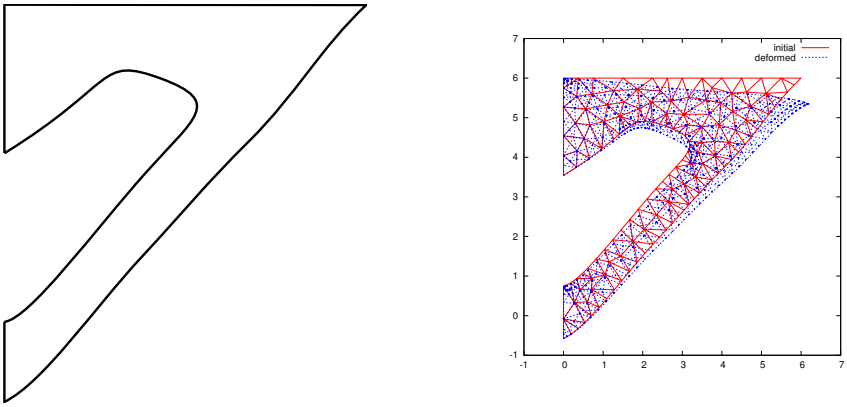
Next, we have a look on the convergence example in Figure 5.3 again. The upper edge and the x_1-coordinates of the Dirichlet boundary are chosen to be the only parts that are not affected by the shape parameters. With the energy performance function (5.1), the shape optimization is performed for $\alpha = 0.02$, $\beta = 0.1$ and $\alpha = 0.01$, $\beta = 0.2$ and its results are illustrated in Figures

5.10, 5.11 and 5.12. The red mesh in the Figures is the undeformed coarse mesh where the blue mesh shows the adaptive mesh refinements and the displacement (multiplied by ten for better visibility) is added to it too.



(a) Shape (b) Meshing and deformation

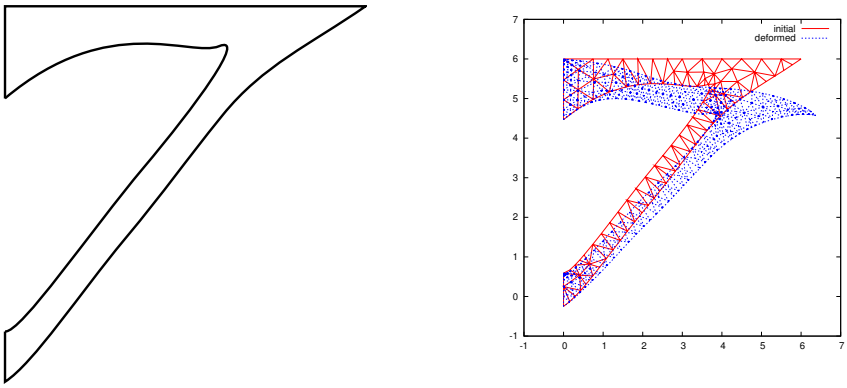
Figure 5.10.: Initial shape of the beam example



(a) Optimized shape (b) Meshing and deformation

Figure 5.11.: Shape optimization of a beam with an additional supporter for $\alpha = 0.02$, $\beta = 0.1$.

Both meshes have still some similarities. At first, the supporter has an almost uniform thickness and is inclined by an angle of roughly 50° to the horizontal axis. Larger angles would lengthen the supporter too much and its deformation would increase, while smaller angles would cause larger shear forces that cause larger deformations too. This optimal position of the supporter is a compromise between the deformations caused by its lengthening and the shear forces. Thanks to the supporter, a large amount of the load can be redirected as a normal force, causing a



(a) Optimized shape (b) Meshing and deformation

Figure 5.12.: Shape optimization of a beam with an additional supporter for $\alpha = 0.01$, $\beta = 0.2$.

smaller potential energy. The upper beam is also strengthened on the left clamping, reducing the shear forces.

Importance of the initial guess

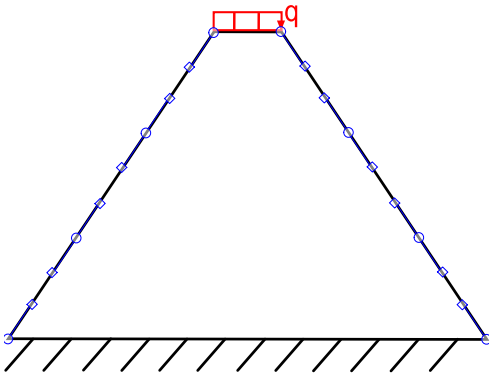
Like in many nonlinear optimization problems, the initial guess may have a strong influence on the process and the local minimum that is found. It is possible that certain shape parameters result in invalid meshes, that is e.g. crossing edges. This can be avoided if certain subsets of the set P of all vectors of shape parameters are declared not admissable for the optimization. Since it is rather difficult to obtain equations that characterize if given shape parameter provides a valid mesh, we utilize a penalization approach and assign it with the initial performance function value to guarantee that infeasible solutions are not chosen as a local minimum. Still, it is possible that the gradient points in a direction that leads to shape parameters that do not provide valid meshes. In this case the optimization process gets stuck and terminates without satisfying the optimality conditions.

We demonstrate the dependence on initial guess by the example of with a frustum in Figure 5.13 with the performance function

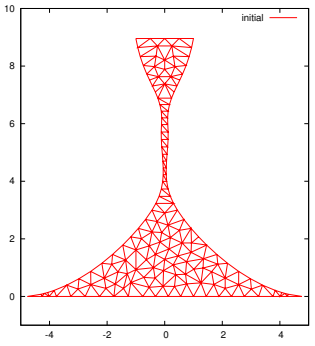
$$I(\Omega) = \alpha \left|\mathrm{u}(x_0)\right|^2 + \beta \int_\Omega dx.$$

First, we start with a frustum that has a wide base. The optimization terminates with the result of a structure that has a thin middle part but still a very wide base. One might guess that this wide base is unnecessary and only increases the performance function value of $I = 0.71839$.

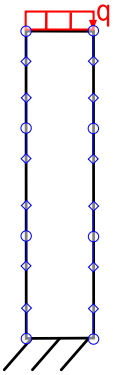
Indeed, a rectangle as initial guess yields an optimal shape that has a far smaller performance function value of $I = 0.24136$.



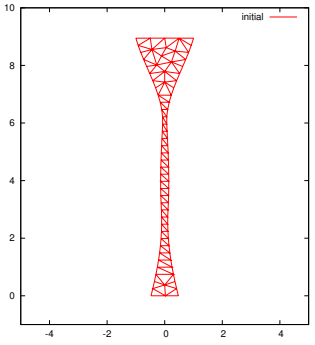
(a) Initial frustum



(b) Optimized shape to a). $I = 0.71839$



(c) Initial rectangle



(d) Optimized shape to b). $I = 0.24136$

Figure 5.13.: Shape optimization of a frustum and its dependence on the initial guess.

Hook model

We consider a structure like a hook shown in Figure 5.14. It is clamped on the left side and a constant load acts on the right upper edge. The two connecting edges are described with three Bézier spline segments each and the performance function is the potential energy 5.1. First, we compare the results depending on how the left clamped edge is constrained. Its x_1-position is constrained to stay at $x_1 = 0$ in both cases, but the x_2-position of the upper node is restricted to stay lower than $x_2 = 1$ in the first case and has no constraints in the second case. The lower node is unconstrained in both cases. Figure 5.15 shows the results of the two different models. The first case yields a result that consists of two parts: a thick horizontal beam that

compensates shear forces and a very thin vertical beam where only normal forces act. In the second case, it is possible that the clamped edge moves up, eliminating the vertical part. It also has more options to shape the upper edge. The clustering of elements near the Neumann boundary results from the optimization process. While the clamped edge slowly moves up, the vertical part is still formed. This means that more shape segments are concentrated near the Neumann boundary. Due to the meshing, each shape segment is refined a few times to ensure that the edges approximate the shape segment well enough. If short shape segments cluster on the right side, so do the elements. Besides that, the two horizontal beams seem to look alike.

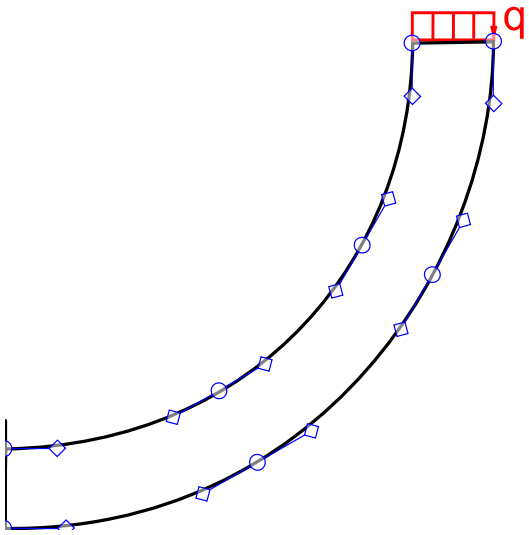
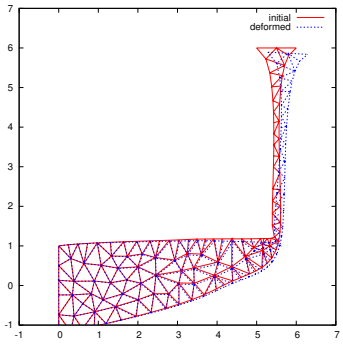
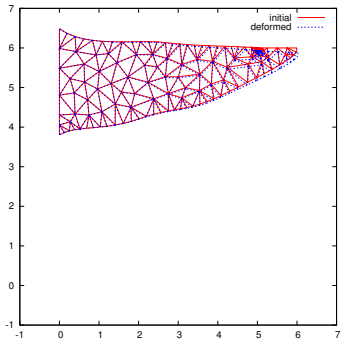


Figure 5.14.: Shape optimization: Initial configuration of the hook model

The results also show that the optimal solution strongly depends on the acting force. We demonstrate that with two more examples. We add an equal x_1-component to the load q. In one case, it points in the negative and in the other case it points in the positive x_1-direction. The results are illustrated in Figure 5.16. A thin beam is sufficient to compensate the normal force in the first case but the shear force in the second case requires a rather thick structure. We see that the results strongly depend on the load q. In order to obtain a rather robust structure we consider the performance function

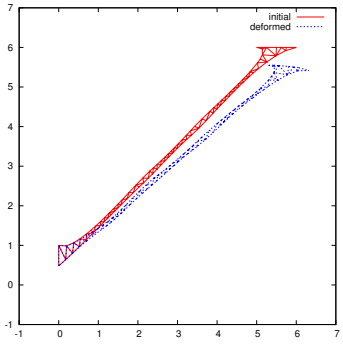


(a) Clamped edge is constrained to stay lower than $x_2 = 1$

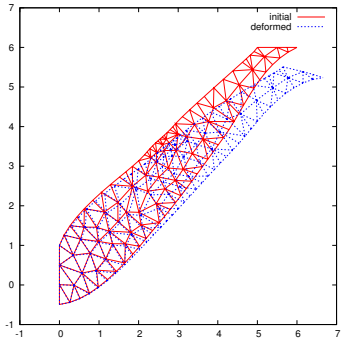


(b) Clamped edge can move along the x_2-axis

Figure 5.15.: Shape optimization of a hook and the dependence of the constraints.



(a) External force points to the clamped edge.

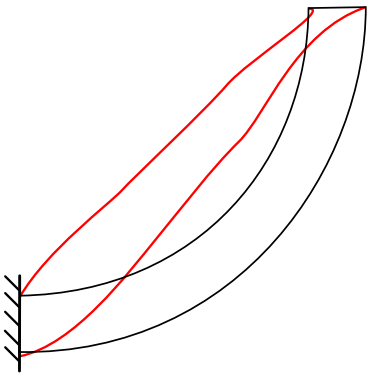


(b) External force is perpendicular to the line between clamped and loaded edge.

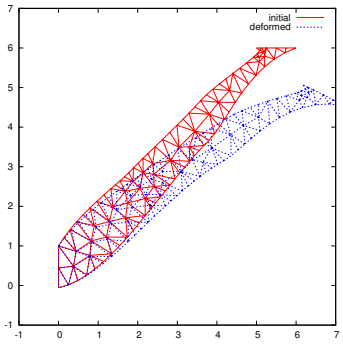
Figure 5.16.: Shape optimization of a hook and the dependence of the external force.

$$I = I_1 + I_2 \quad \text{with} \quad I_i = \alpha \int_\Omega \frac{1}{2}\varepsilon(\mathrm{u}^{(i)}) : C\varepsilon(\mathrm{u}^{(i)})\, dx + \alpha \int_{\Gamma_N} g \cdot \mathrm{u}^{(i)}\, dx + \beta \int_\Omega dx,$$

where $\mathrm{u}^{(1)}$ and $\mathrm{u}^{(2)}$ are the solution to two different external forces, $g^{(1)}$ acting in x_1- and $g^{(2)}$ x_2-direction. The result can be seen in Figure 5.17.



(a) Optimized shape (red) in comparison to the initial shape.



(b) Deformation of the optimized shape.

Figure 5.17.: Shape optimization for the combined performance function $I = I_1 + I_2$, considering two different external forces.

A. Appendix

Derivation of the Derivative of the Residual R with respect to the node positions

$$
\begin{aligned}
& \frac{\partial}{\partial [x_j]_s} \left[\int_{\Omega'} \lambda (\nabla \cdot u)(\nabla \cdot \varphi_i^r) \, dx \right] \\
= \; & \frac{\partial}{\partial [x_j]_s} \left[\sum_{el} \sum_{k} w_k \lambda \left| \det(\mathrm{J}) \right| ([u]_{x,x} + [u]_{y,y}) \varphi_{i,r} \right] \\
= \; & \sum_{el} \sum_{k} w_k \lambda \left[([u]_{x,x} + [u]_{y,y}) \varphi_{i,r} \frac{\partial \left| \det(\mathrm{J}) \right|}{\partial [x_j]_s} + \left| \det(\mathrm{J}) \right| \frac{\partial ([u]_{x,x} + [u]_{y,y}) \varphi_{i,r}}{\partial [x_j]_s} \right] \\
= \; & \sum_{el} \sum_{k} w_k \lambda \left[([u]_{x,x} + [u]_{y,y}) \varphi_{i,r} \frac{\partial \left| \det(\mathrm{J}) \right|}{\partial [x_j]_s} + \right. \\
& \qquad \left. \left| \det(\mathrm{J}) \right| \{ -([u]_{x,x} + [u]_{y,y}) \varphi_{j,r} \varphi_{i,s} + (-\varphi_{j,x} u_{x,s} - \varphi_{j,y} u_{y,s}) \varphi_{i,r} \} \right],
\end{aligned}
$$

$$
\begin{aligned}
& \frac{\partial}{\partial [x_j]_s} \left[\int_{\Omega'} 2\mu \sum_{i,j=1}^{2} \varepsilon_{ij}(u) \varepsilon_{ij}(\varphi_i^r) \, dx \right] \\
= \; & \frac{\partial}{\partial [x_j]_s} \left[\sum_{el} \sum_{k} 2\mu w_k \left| \det(\mathrm{J}) \right| \left(u_{r,r} \varphi_{i,r} + \frac{1}{2} (u_{x,y} + u_{y,x}) \varphi_{i,\overline{r}} \right) \right] \\
= \; & \sum_{el} \sum_{k} 2\mu w_k \left[\frac{\partial \left| \det(\mathrm{J}) \right|}{\partial [x_j]_s} \left(u_{r,r} \varphi_{i,r} + \frac{1}{2} (u_{x,y} + u_{y,x}) \varphi_{i,\overline{r}} \right) \right. \\
& \qquad + \left| \det(\mathrm{J}) \right| (-u_{r,r} \varphi_{j,r} \varphi_{i,s} - \varphi_{j,r} u_{r,s} \varphi_{i,r} \\
& \qquad \left. - \frac{1}{2} (u_{x,y} + u_{y,x}) \varphi_{j,\overline{r}} \varphi_{i,s} - \frac{1}{2} (\varphi_{j,y} u_{x,s} + \varphi_{j,x} u_{y,s}) \varphi_{i,\overline{r}}) \right],
\end{aligned}
$$

$$
\begin{aligned}
& \frac{\partial}{\partial [x_j]_s} \left[\int_{\Gamma_1} [g]_r \cdot \varphi_i^{(1)} \, ds \right] \\
= \; & \frac{\partial}{\partial [x_j]_s} \left[\sum_{bd} \sum_{k} w_k \; \left\| t(\widehat{x}_k) \right\|_2 \; [g]_r \; \widehat{\varphi}_i(\widehat{x}_k) \right] \\
= \; & \sum_{bd} \sum_{k} w_k \; [g]_r \, \widehat{\varphi}_i(\widehat{x}_k) \frac{\partial \left\| t(\widehat{x}_k) \right\|_2}{\partial [x_j]_s}.
\end{aligned}
$$

Integrating $a(e,v)$ by parts

To derive (4.1) we need the formula for integration in higher dimensions and a generalization of it,

$$\begin{aligned}\int_T \frac{\partial v}{\partial \mathrm{x}_i} f \; dx &= \int_{\partial T} v \; f \; [n]_i \; ds - \int_T v \frac{\partial f}{\partial \mathrm{x}_i} \; dx, \\ \int_T \nabla v \cdot f \; dx &= \int_{\partial T} v \; n \cdot f \; ds - \int_T \; v \; \nabla \cdot f \; dx,\end{aligned}$$

where $f, v \in H^1(T)$, and n being the normal vector of the surface ∂T. With the strain being $\varepsilon(v) = 1/2 \; (\nabla v + \nabla v^T)$ and the tension $\sigma = C\varepsilon(u_h)$ being symmetrical, we have

$$\begin{aligned}\varepsilon(v) : C\varepsilon(u_h) &= tr\left(1/2 \; (\nabla v + \nabla v^T)(C\varepsilon(u_h))^T\right) \\ &= 1/2 \; tr(\nabla v C\varepsilon(u_h)) + 1/2 \; tr(\nabla v^T C\varepsilon(u_h)).\end{aligned}$$

We insert this in $a(e,v)$ and obtain

$$\begin{aligned}a(e,v) &= \sum_{T \; \in \mathbb{T}} \left\{ \int_{\partial T \cap \Gamma} g \cdot v \; ds - \int_T \varepsilon(v) : C\varepsilon(u_h) \; dx \right\} \\ &= \sum_{T \; \in \mathbb{T}} \left\{ \int_{\partial T \cap \Gamma} g \cdot v \; ds - \int_T 1/2 \; tr(\nabla v C\varepsilon(u_h)) + 1/2 \; tr(\nabla v^T C\varepsilon(u_h)) \; dx \right\}.\end{aligned}$$

We now examine the two parts of the integral over Ω. Applying (A.1) to the first part gives

$$\begin{aligned}& 1/2 \int_T tr(\nabla v C\varepsilon(u_h)) \; dx \\ = \; & 1/2 \int_{\partial T} n \cdot \varepsilon(u_h) \cdot v \; ds - 1/2 \int_T \nabla \cdot C\varepsilon(u_h) \cdot v \; dx.\end{aligned}$$

For the second part, applying (A.1) yields

$$
\begin{aligned}
& 1/2 \int_T tr(\nabla v^T C\varepsilon(u_h))\ dx \\
= \ & 1/2 \int_T \sum_i [\nabla v^T C\varepsilon(u_h)]_{ii}\ dx \\
= \ & 1/2 \int_T \sum_i \sum_j [\nabla v^T]_{ij} [C\varepsilon(u_h)]_{ji}\ dx \\
= \ & 1/2 \sum_i \sum_j \int_T \frac{\partial [v]_j}{\partial \mathrm{x}_i} [C\varepsilon(u_h)]_{ji}\ dx \\
= \ & 1/2 \sum_i \sum_j \left\{ \int_{\partial T} [v]_j [C\varepsilon(u_h)]_{ji} n_i\ ds - \int_T [v]_j \frac{\partial [C\varepsilon(u_h)]_{ji}}{\partial \mathrm{x}_i}\ dx \right\} \\
= \ & 1/2 \left\{ \int_{\partial T} v \cdot (n \cdot C\varepsilon(u_h))\ ds - \int_T \nabla \cdot C\varepsilon(u_h) \cdot v\ dx \right\}.
\end{aligned}
$$

Combining and inserting these equations gives the result (4.1),

$$
a(e,v) = \sum_{T \in \mathbb{T}} \left\{ \int_{\partial T \cap \Gamma} g \cdot v\ ds + \int_T \nabla \cdot C\varepsilon(u_h) \cdot v\ dx - \int_{\partial T} n \cdot C\varepsilon(u_h) \cdot v\ Sds \right\}.
$$

References

[1] E. Baensch. Local mesh refinement in 2 and 3 dimensions. *Impact of Computing in Science and Engineering*, 3:181–191, 1991.

[2] W. Bangerth, R. Rannacher. *Adaptive Finite Element Methods for Differential Equations*. Birkhäuser Verlag, Basel-Boston-Berlin, 2003.

[3] D. Braess. *Finite Elemente - Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer Lehrbuch, Berlin, 3. Auflage, 2003.

[4] R. Cools. Monomial cubature rules since "Stroud": a compilation – part 2. *Journal of Computational and Applied Mathematics*, 48:309–326, 1993.

[5] R. Cools. An encyclopaedia of cubature formulas. *Journal of Complexity*, 19:445–453, 2003.

[6] R. Cools, P. Rabinowitz. Monomial cubature rules since "Stroud": a compilation. *Journal of Computational and Applied Mathematics*, 112(1-2):21–27, 1999.

[7] J. Nocedal, S. Wright. *Numerical Optimization*. Springer Series in Operations Research, 1999.

[8] R. Schneider. *Applications of the Discrete Adjoint Method in Computational Fluid Dynamics*. PhD thesis, The University of Leeds School of Computing, 2006.

[9] R. Schneider, P. Jimack. On the evaluation of finite elements sensitivities to nodal coordinates. *Electronic Transactions on Numerical Analysis (ETNA)*, 32:134–144, 2009.

[10] P. Spellucci. A new technique for inconsistent problems in the SQP method. *Mathematical Methods of Operations Research*, 47:355–400, 1998.

[11] P. Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. *Mathatical Programming*, 82:413–448, 1998.

Conclusions

The application of the discrete adjoint method and the adaptive mesh refinement have been the main topic of this work. The DAM allows an efficient omputation of the derivate of a given performance function I with respect to parameters p, in case I depends on p indirectly. For example, a quantity $u(p)$ that directly depends on p though its evaluation is expensive. This holds true for the Lamé problem where the potential energy depends on the displacement u which is the solution of a partial differential equation. The advantages of the adaptive mesh refinement have been shown regarding the discretization error, computational costs and accelerating the optimization process.

Theses (english)

1. In the course of the considered shape optimization of mechanical structures, there are to contradicting goals: a high stiffness and a small material volume. We look for a compromise between both quantities.

2. The linear elasticity theory is sufficient to model small deformations.

3. The displacements in the linear elasticity theory are described by the Lamé equation, which is an elliptical partial differential equation of order two.

4. The Lamé equation can be solved approximately with the help of the finite element method.

5. To gain a suitable distribution of the mid nodes on a spline segment, it is necessary to determine the spline parameter t for each node on the spline.

6. The uniform mesh refinement reduces the discretization error, though that results in very large computational costs.

7. The adaptive mesh refinement can achieve comparable discretization errors with far smaller computational costs. This is a huge advantage for the shape optimization which demands a large number of evaluations of the performance function.

8. Since the number of unknowns increase moderately with the adaptive mesh refinement, direct solvers are still practical.

9. The gradient of the performance function can be computed with the help of the discrete adjoint method.

10. Artificial discontinuities due to the meshing are resolved by mesh refinements which accelerates the optimization.

11. Normal forces cause smaller potential energy than shear forces. Due to this, the shape optimization preferres structures where the external force is redirected as normal forces inside the structure.

12. Even the fact that the adaptive mesh refinement is a discontinuous process, it weakens discontinuities in the solution u_h and thus in the performance function I.

Thesen (deutsch)

1. Im Rahmen der hier betrachteten Formoptimierung für mechanische Bauteile gibt es zwei sich widersprechende Ziele, eine hohe Steifigkeit und eine geringer Materialvolumen. Gesucht ist ein Kompromiss zwischen beiden Größen.

2. Die lineare Elastizitätstheorie ist für die Modellierung von kleinen Deformationen ausreichend.

3. Die Verschiebungen in der lineare Elastizitätstheorie werden durch die Lamé'sche Differentialgleichung beschrieben, die eine elliptische partielle Differentialgleichung zweiter Ordnung ist.

4. Mit Hilfe der Finiten-Elemente-Methode kann die Lamé'sche Gleichung näherungsweise gelöst werden.

5. Um eine brauchbare Verteilung der Knoten auf den Spline-Segmenten zu erhalten, muss der Spline-Parameter t für jeden Knoten auf dem Spline einzeln bestimmt werden.

6. Die uniforme Netzverfeinerung verringert den Diskretisierungsfehler, was allerdings sehr hohe Rechenkosten zur Folge hat.

7. Die adaptive Netzverfeinerung kann vergleichbare Diskretisierungsfehler mit sehr viel geringeren Rechenkosten erzielen. Dies ist von entscheidenden Vorteil für die Formoptimierung, die viele Zielfuntionsauswertungen erfordert.

8. Da bei der adaptiven Netzverfeinerung die Zahl der Unbekannten moderat bleibt, sind direkte Löser praktikabel.

9. Mit Hilfe der Methode der Diskreten Adjungierten kann der Gradient der Zielfunktion bestimmt werden.

10. Künstliche Unstetigkeiten durch die Vernetzung werden durch Netzverfeinerungen beseitigt, was die Optimierung beschleunigt.

11. Die potentielle Energie ist bei Normalkräften sehr viel kleiner als bei Scherkräften. Dies bewirkt, das die Formoptimierung Strukturen bevorzugt, wo die äußere Belastung als Normalkraft umgeleitet wird.

12. Obwohl die adaptive Netzverfeinerung ein unstetiger Prozess ist, schwächt sie doch die Unstetigkeiten in der Lösung u_h und damit der Zielfunktion I.

Eidesstattliche Versicherung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe angefertigt habe. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Chemnitz, den 22. Januar 2010