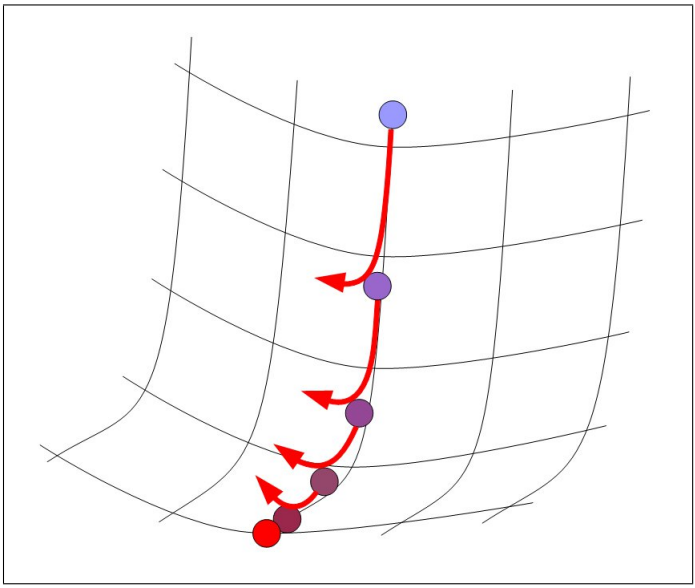
# Interior Point Methods with Second Derivatives for Solving Large Scale Nonlinear Programming Problems

Diploma Thesis



Tobias Werner

July 14, 2010

Lehrstuhl Angewandte Informatik 7
Universität Bayreuth
95440 Bayreuth, Germany

# Table of Contents

**1 Introduction** **3**
1.1 Abstract . . . 3
1.2 Motivation . . . 3
1.3 Overview . . . 5

**2 Mathematical Background** **7**
2.1 Nonlinear Programs . . . 7
2.2 Nonlinear Convex Programs . . . 9
2.2.1 Convex and Concave Sets . . . 9
2.2.2 Convex and Concave Functions . . . 10
2.2.3 Convex Problems . . . 11
2.3 Optimality Conditions . . . 11
2.3.1 Lagrange Function . . . 11
2.3.2 KKT Conditions . . . 17
2.4 Newton Methods . . . 18
2.4.1 Root Finding . . . 19
2.4.2 Quasi-Newton Methods . . . 21
2.4.3 BFGS Updates . . . 23
2.5 Sequential Quadratic Programming . . . 25
2.5.1 Quadratic Programs . . . 26
2.5.2 Algorithm Derivation . . . 27
2.5.3 Algorithm Outline . . . 29
2.5.4 Convergency . . . 31
2.6 Interior Point Methods . . . 31
2.6.1 Introduction . . . 32
2.6.2 Barrier Methods . . . 32
2.6.3 Algorithm Derivation . . . 35
2.6.4 Realization . . . 38
2.6.5 Convergency . . . 39
2.7 Enhancements . . . 39
2.7.1 Quasi-Newton Adaptions . . . 40
2.7.2 Line Search . . . 41
2.7.3 Stopping Conditions . . . 41

**3 Implementation** **43**
3.1 NLPIP . . . 43
3.1.1 Clientside Interface . . . 43
3.1.2 Internal Structure . . . 44
3.1.3 LINSLV module . . . 44
3.2 HSSSLV module . . . 46
3.2.1 Fortran Gradient Structure . . . 46
3.2.2 Module Routines . . . 48

**4 Tests** **51**
4.1 Testing Scenarios . . . 51

4.1.1 Standard Tests . . . 51
4.1.2 Free Material Optimization Tests . . . 53
4.2 Results . . . 55
4.2.1 Standard Tests . . . 55
4.2.2 Free Material Optimization Tests . . . 61
4.2.3 Results Table . . . 74
4.3 Evaluation . . . 75

**5 Conclusion** **77**

**Appendices** **78**
A Bibliography . . . 78
B HSSSLV source code . . . 80
C Maple to FORTRAN 70 Conversion Tool . . . 87
D Compact Disc Contents . . . 92
E Deutschsprachige Zusammenfassung . . . 94
E.1 Mathematische Grundlagen . . . 94
E.2 Implementierung . . . 99
E.3 Anwendung in der freien Materialoptimierung . . . 100
E.4 Tests . . . 101
F Software Tools . . . 102
G Erklärung zur Authentizität . . . 103

# 1 Introduction

## 1.1 Abstract

**Topic of this diploma thesis is the optimization of an existing solver for nonlinear programs in the context of sparse, large-scale problems by the introduction of analytically calculated second derivatives. In particular, the existing solver iteratively approximates the nonlinear program by a sequence of quadratic programs, and then solves these with an inner interior point method alongside a limited memory BFGS update to accommodate for large input problems. In the proceedings of this thesis, the solver was extended to allow for analytical second derivatives in a sparse representation as input instead of their BFGS approximation. Both the original and the optimized implementation have been tested and compared for convergency performance on a set of standard problems. Finally, both implementations have been put to trial in a higher level solver designed for free material optimization.**

## 1.2 Motivation

The motivation for this thesis comes from the field of free material optimization (FMO) in aerospace applications [RS10].

One of the most important aspects of aircraft development is reducing weight. Lighter aircraft need less fuel, thus travel more economically. Side benefits are better maneuverability and longer gliding times, in particular in the context of emergency situations.

In classical aircraft design, most parts of the aircraft are made from metals and alloys of various kind. Weight then can be optimized, given certain constraints on the tolerable deflection, by changing the basic structural topology of the aircraft. Particularly strained parts may be made from thicker sheets and plates, while material may be conserved for less stressed components. Figure 1.1 shows one such optimized material distribution.

In this context, metals and alloys exhibit a material property called isotropy: Material is distorted indifferently to the direction the material is stressed in. See figure 1.2 for an illustrative example.

There are many renowned methods for calculating and optimizing topologies for isotropic materials, readily available either as standalone software or as plugins into commercial CAD applications.

However, with recent advances in technology, carbon fiber-reinforced polymers have more and more become a viable, if expensive, alternative for the construction of lightweight and durable aircraft components. Compared to traditional steel or aluminium, carbon composites offer a much higher stiffness to weight ratio. Furthermore, carbon composites are not affected by additional wear when under dynamic

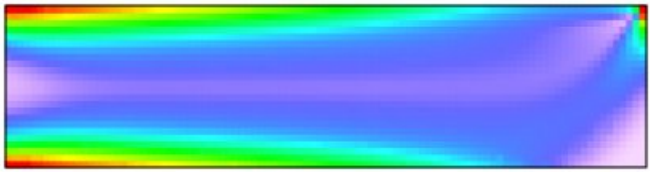


Figure 1.1: An optimal material distribution on a plate element. The force comes from the upper right-hand side, while the element is fixed on the left-hand side. Red colours indicate particularly strained segments, where additional material is required. Blue and white colours represent unstrained sections, where material may be conserved. Picture courtesy of [SE09].

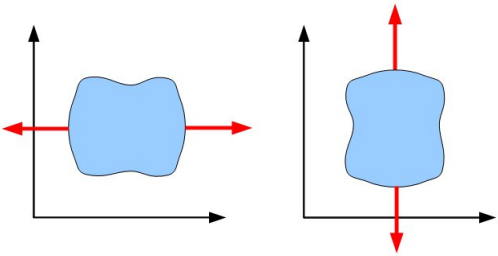


Figure 1.2: Isotropic materials react to stress without regarding the stress direction.

load changes. Yet, fiber-reinforced materials also exhibit anisotropic reactions to stress. Load tolerance in direction of the fibers usually is much higher than tolerance for load perpendicular to the fibers. This is demonstrated in figure 1.3.

Due to this, optimization problems for anisotropic materials exhibit a more complicated structure than those for isotropic ones. Thus, traditional FMO techniques cannot be used for carbon composites without modifications. Alternative optimization methods for anisotropic materials have not been investigated into extensively.

For this reason, the EU funded a research project by the name of Plato-N. The Plato-N project aimed at the development of mathematical methods and associated software tools for material optimizations on carbon composite components. Project staff was formed by a joint venture of select universities, renowned aircraft manufacturers, and independent software producers.

In the context of the Plato-N project, the University of Bayreuth was tasked with the design of an optimizer that could handle the large scale, sparse, nonlinear programs that represent material optimizations on anisotropic materials.

The resulting FMO optimizer is implemented in a program named SCPF [CZ04] [SE09]. Within SCPF, a sequential convex programming method is used to iteratively

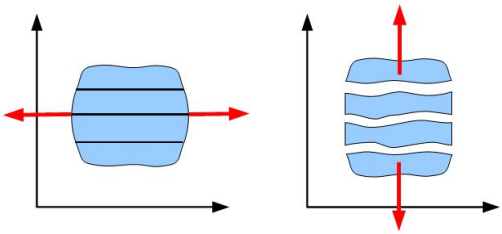


Figure 1.3: Anisotropic materials are more resistant to stresses from a predefined direction - such as the alignment direction of fibers in a carbon composite.

approach a solution of the original FMO problem by generating and solving convex subproblems.

For solving the convex subproblems, an existing nonlinear solver by the name of NLPIP [BS08] [BS09] is employed. NLPIP offers two optimization methods: An outer solver with a sequential quadratic programming approach, and an encapsulated inner solver, based on the interior point method.

In both methods, NLPIP originally only used analytical first derivatives of the objective function. Second derivatives were numerically approximated by a limited memory BFGS update strategy whenever required. Yet, the objective function for the FMO problem allows for exact, analytical calculation of second derivatives as well. It is intuitively understood that accurate second derivatives then might be exploited to improve convergency of the entire optimization process.

Thus the topic of this thesis: **Interior Point Methods with Second Derivatives for Solving Large Scale Nonlinear Programming Problems**.

## 1.3 Overview

As described in the leading abstract and motivation sections, the goal of this diploma thesis was the extension of an existing solver for nonlinear optimization problems by the introduction of analytically calculated second derivatives. In particular, an existing FORTRAN program by the name of NLPIP was to be modified so that sparse, analytical Hessian matrices could be used as an alternative to its default limited memory BFGS update strategy.

This thesis composition provides the background for the actual FORTRAN implementation. The following pages include mathematical foundations, implementation notes, test setups, testing results, and context information for the major application scenario of free material optimization.

As a rough outline, the composition splits into three main parts:

- **Mathematical Background**

  First, the mathematical background specific to nonlinear optimization with sequential quadratic programming and interior point methods is presented. This both includes a review over Newton and quasi-Newton methods, and algorithmic details for the later implementation.

- **Implementation**

  Using the precedingly established notations and conventions, the FORTRAN implementation of the base nonlinear solver NLPIP and the modifications for second derivatives are explained.

- **Tests**

  In the last part, testing scenarios for both the original and the modified solver are described and the test results are presented and evaluated.

# 2 Mathematical Background

This chapter describes the mathematical background for the later implementation of NLPIP. In particular, the basic problem formulation for general nonlinear and convex nonlinear optimization problems is given, and conditions for optimal points within a convex nonlinear program are derived. Then, sequential quadratic programming and interior point methods for solving nonlinear problems are introduced.

All findings are not only described mathematically, but also explained extensively with examples and illustrations. Transferring an intuitive understanding of concepts, implications, and interrelationships is the focus point of this thesis.

Both the actual mathematical presentation as well as the accompanying examples are based on [KS08], [FJ04], [IG04], and [BS08]. However, contents have been adapted to fit together fluently. Additional sources are cited where applicable.

## 2.1 Nonlinear Programs

Nonlinear programming means finding an optimal (minimal or maximal) point on a nonlinear, real and twice continously differentiable objective function

$$f : \mathbb{R}^m \to \mathbb{R}, \qquad f \in C^2(\mathbb{R}^m). \tag{2.1}$$

A set of $n := p + q$ nonlinear, real and twice continously differentiable functions

$$h_i : \mathbb{R}^m \to \mathbb{R}, \qquad h_i \in C^2(\mathbb{R}^m) \qquad i = 1, \ldots, n \tag{2.2}$$

is used to define $p$ boundary equations and $q$ boundary inequalities

$$h_i(x) = 0 \qquad \text{for } i = 1, \ldots, p \tag{2.3}$$

$$h_i(x) \geq 0 \qquad \text{for } i = p+1, \ldots, p+q = n \tag{2.4}$$

that constrain the acceptable solutions on the objective function.

The nonlinear minimization problem can thus be formulated as:

**Definition 2.1. Nonlinear Program (NLP)**

$$\min_{x \in \mathbb{R}^m} f(x) \tag{2.5}$$

$$\text{so that:} \quad h_i(x) = 0 \quad \text{for } i = 1, \ldots, p$$

$$h_i(x) \geq 0 \quad \text{for } i = p+1, \ldots, n.$$

In the following text, only minimization problems are considered. A maximization problem for an objective function $\tilde{f}$ can be replaced by a minimization problem on $f := -\tilde{f}$.

Boundary conditions that do not match the above canoical form can be normalized by various methods. For instance, a condition of the form $\tilde{h}(x) \leq c$ may be replaced

by a normalized condition $h(x) := c - \tilde{h}(x) \geq 0$. See [FJ04] for further normalization strategies, in particular for strategies on the subset of linear problems.

Points from the domain of the objective function $f$ can be qualified in regards to the set of boundary constraints:

**Definition 2.2.** Given (NLP), a point $x \in \mathbb{R}^m$ is called feasible if $h_i(x) = 0$ for $i = 1, \ldots, p$ and $h_i(x) \geq 0$ for $i = p+1, \ldots, n$. Points that are not feasible are termed infeasible.

The optimization domain

$$P := \{x \in \mathbb{R}^m \mid h_1(x) = 0, \ldots, h_p(x) = 0, h_{p+1} \geq 0, \ldots, h_n(x) \geq 0\} \tag{2.6}$$

represents the set of all feasible points.

The remaining text only considers non-pathological constellations of boundary conditions. In particular, boundary conditions always are required to be non-contradictory, so that the optimization domain $P$ never is empty.

Finally, boundary conditions can also be qualified in regards to some fixed point:

**Definition 2.3.** Given (NLP), any boundary condition $h_i, i = 1, \ldots, n$ is termed active at a given point $x \in \mathbb{R}^m$ if $h_i(x) = 0$. A boundary condition that is not active is termed inactive.

See figure 2.1 for an example nonlinear minimization problem.

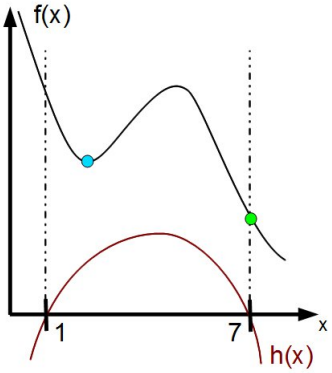



Figure 2.1: In this example, nonlinear minimization means finding either the local or the global minimum on the nonlinear function $f(x)$. There is a single, nonlinear boundary inequality $h(x) \geq 0$ which confines the optimization domain to the interval $[1, 7]$.

## 2.2 Nonlinear Convex Programs

Deriving intuitive optimality conditions for general, nonlinear problems is difficult. Consequently, program formulations often are restricted to the subset of nonlinear convex optimization problems. The interior point method of the NLPIP implementation also is motivated by convex optimization.

Thus, this section gives base definitions for convexity and concavity, provides properties of concave sets and concave functions, and finally defines nonlinear convex problems.

### 2.2.1 Convex and Concave Sets

Sets are classified into convex and concave sets:

**Definition 2.4.** A set $M \subset \mathbb{R}^m$ is called convex if

$$\lambda x + (1-\lambda)y \in M \qquad \forall x, y \in M \quad \text{and} \quad \forall \lambda \in [0,1] \tag{2.7}$$

A set $M \subset \mathbb{R}^m$ is called concave if it is not convex.

In colloquial english, a set is convex if a line segment can be drawn between any two points from the set without crossing the set's boundary. Example convex and concave sets are depicted in figure 2.2.

The definition of concavity implies that any set $M \subset \mathbb{R}^m$ is concave if there are $\lambda \in [0,1]$, $x, y \in M$, so that $\lambda x + (1-\lambda)y \notin M$. Thus, the empty set $\emptyset$ is concave, while the $\mathbb{R}^m$ is convex.

Convex sets are closed under intersection operations, as the intersection of two convex sets yields another convex set:

**Lemma 2.1.** *Let $A$ and $B$ be two convex sets, then the intersection $A \cap B$ is convex.*

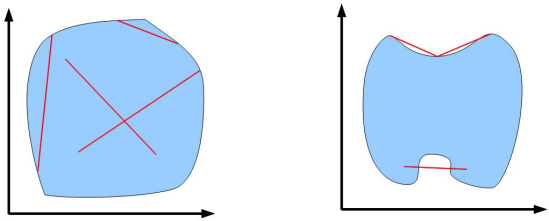


Figure 2.2: A convex set on the left-hand side and a concave set on the right-hand side. The coloured lines are examples for line segments completely inside the convex set and partially outside the concave set.

*Proof.* By contradiction: If $A \cap B$ were not convex, there were at least two points $x, y \in A \cap B$ and a $\lambda \in [0,1]$ so that $p := \lambda x + (1-\lambda)y \notin A \cap B$. However, $x, y \in A \cap B \Rightarrow x, y \in A$ and $x, y \in B \Rightarrow p \in A$ and $p \in B \Rightarrow p \in A \cap B$. $\lightning$ □

### 2.2.2 Convex and Concave Functions

As with sets, there also is a connotion of concavity and convexity for functions:

**Definition 2.5.** The function $g : D \to \mathbb{R},\ D \subset \mathbb{R}^m$, is called convex on $D$ if

$$\lambda g(x) + (1-\lambda)g(y) \quad \geq \quad g(\lambda x + (1-\lambda)y) \qquad \forall x, y \in D \quad \text{and} \quad \forall \lambda \in [0,1]\,. \tag{2.8}$$

The function $g$ is called concave on $D$ if $-g$ is convex on $D$.

An alternative, algebraical formulation for concave functions based on the above definition is as such:

**Lemma 2.2.** *The function $g : D \to \mathbb{R},\ D \subset \mathbb{R}^m$, is concave on $D$ if $\lambda g(x)+(1-\lambda)g(y) \leq g(\lambda x + (1-\lambda)y) \quad \forall x, y \in D$ and $\forall \lambda \in [0,1]$.*

*Proof.* Substitution of g(x) by -g(x) in the definition of convex functions. □

In an intuitive understanding, a function is convex if any line segment connecting two arbitrary points on the function surface always is on or above the function surface. Likewise, a function is concave if any such line segment always is below or on the function surface. An example for convex and concave functions is given in figure 2.3.

As a relation to sets, it is directly seen from the above definitions that the area above a convex function and the area below a concave function form convex sets. In this context, there also is an important difference between sets and functions: Sets always are either concave or convex, while functions need be neither. The polynomial $x^3$ for instance does not exhibit concavity or convexity over $\mathbb{R}$.

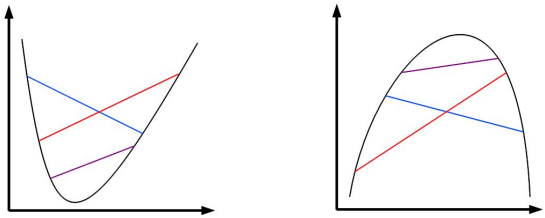


Figure 2.3: A convex function on the left-hand side and a concave function on the right-hand side. The coloured lines are examples for line segments completely above or below the respective function.

### 2.2.3 Convex Problems

When considering the original formulation of the nonlinear problem, the optimization domain over the objective function is defined by a series of boundary equations and inequalities. If all boundary functions are diligently selected, the resulting optimization domain gains particularly desireable properties, as well:

**Lemma 2.3.** *Given the boundary functions $h_1, \ldots, h_n$ and associated $p$ equations and $q$ inequalities from (NLP), let $h_1, \ldots, h_p$ be affine, and let $h_{p+1}, \ldots, h_n$ be concave.*

*Then the optimization domain $P$ is convex.*

*Proof.* For just a single, affine boundary equation or a single, concave boundary inequality with index $i$, the set of feasible points $P_i$ is convex.

For any affine boundary equation $h_i$, this follows from affinity: Affine functions can be expressed as $h_i(x) = a^T x + b$ for some $a \in \mathbb{R}^m, a \neq 0$, and some $b \in \mathbb{R}$. Consider arbitrary $x, y$ with $h_i(x) = 0$, and $h_i(y) = 0$, furthermore any $\lambda \in [0, 1]$. On the one hand $h_i(x) = a^T x + b = 0$, $h_i(y) = a^T y + b = 0 \Rightarrow x - y = 0$. On the other hand $h_i\big(\lambda x + (1-\lambda)y\big) = a^T\big(\lambda x + (1-\lambda)y\big) + b = \lambda a^T(x-y) + (a^T y + b) = 0$. Thus any point $\lambda x + (1-\lambda)y \in P_i$.

For any concave boundary constraint $h_i$, this results from convexity of the area under the function: For arbitrary $x, y$ with $h_i(x) \geq 0$ and $h_i(y) \geq 0$ and for arbitrary $\lambda \in [0, 1]$, all $h_i\big(\lambda x + (1-\lambda)y\big) \geq \lambda h_i(x) + (1-\lambda)h_i(y) \geq 0$, thus any point $\lambda x + (1-\lambda)y \in P_i$.

As shown in lemma 2.1, the intersection of convex sets is convex. The set $P$ is formed by the points that are valid for each single boundary condition, thus by the intersection of all convex $P_i$, thus $P$ is convex. □

If the objective function $f$ and the optimization domain both are convex, then the entire nonlinear problem is named convex. As seen from lemma 2.3, this is the case if all boundary equations refer to affine functions, and if all boundary inequalities are defined on concave functions.

## 2.3 Optimality Conditions

In this section, the convex programs introduced in the previous section are investigated for intuitive optimality conditions.

### 2.3.1 Lagrange Function

One important concept for classifying any given point within the optimization domain is the Lagrange function of the optimization problem:

**Definition 2.6. Lagrange Function**

The function

$$L : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$$
$$L(x, u) := f(x) - \sum_{j=1}^{n} u_j h_j(x) \tag{2.9}$$

is termed the Lagrange function of the nonlinear optimization problem.

The components of the vector $u := \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix}$ are known as the Lagrange multipliers.

The gradient of the Lagrange function represents the balance between the objective function and the boundary constraints at some given point in the optimization domain. An algorithm may use this balance as a basis for deciding whether an optimal point has been found: A stationary point of the Lagrange function hints at an optimum.

As an illustration for this property [WPa], consider only a single equality constraint $h$ on some objective function $f$. Then, investigate possible stationary points on the Lagrange function:

$$\nabla_x L(x, u) = \nabla f(x) - u\nabla h(x) = 0 \quad \Leftrightarrow \quad \nabla f(x) = u\nabla h(x) \tag{2.10}$$

Thus, a stationary point $x'$ on the Lagrange function can only be achieved if the gradients of $f$ and $h$ are linearly dependent. Intuitively, this behaviour can be explained as such: If $\nabla f(x')$ and $\nabla h(x')$ are linearly dependent, then further, small movement along the boundary equation $h(x) = 0$ and perpendicular to the gradient $\nabla h(x')$ also moves perpendicular to the gradient $\nabla f(x')$. As a result, movement is confined to a single niveau value on the objective function, the objective function does not change by the movement. This indicates that $x'$ either is an optimum, or at least a saddle point of the constrained objective function. Yet, if gradients $\nabla f(x')$ and $\nabla h(x')$ are linearly independent, further movement alongside $h(x)' = 0$ yields either a descent or an ascent in $f$, so $x'$ can not be the position of an optimum. This is visualized in figure 2.4.

An alternative, more algebraic understanding is explained in [FJ04]:

Let a nonlinear optimization problem on an objective function $f$ and a single boundary inequality $h(x) \geq 0$ be given. The corresponding Lagrange function is $L(x, u) = f(x) - uh(x)$. Now, consider minimization problems of the form

$$\min_{x \in \mathbb{R}} L(x, u) = \min_{x \in \mathbb{R}} f(x) - uh(x) \tag{2.11}$$

for fixed values $u \in \mathbb{R}, u \geq 0$. Values for $u < 0$ are not considered, since these turn the concave boundary equations into convex ones.

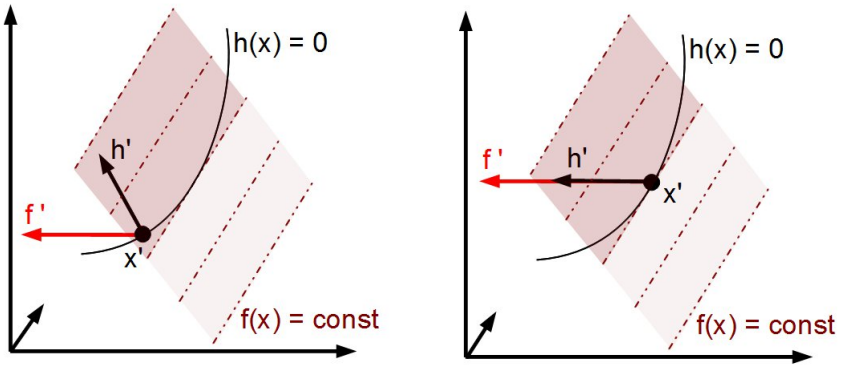



Figure 2.4: Both images display an objective function $f$ and an associated equality constraint $h(x) = 0$ on a constraint function $h$. $f'$ and $h'$ respectively are the gradients of $f$ and $h$ at the point $x$. In the left image, both gradients are linearly independent. Thus further movement along $h(x) = 0$ allows for further descent on the function, and there is no minimum at $x'$. On the right image, the gradients are linearly dependent. In the example, this is a necessary condition for an optimal value at $x'$.

If $u$ is close to zero, the associated problem is $\min_{x \in \mathbb{R}} f(x)$. The resulting solution then provides a global minimum for $f$ on $\mathbb{R}$, but probably violates the boundary inequality $h$. If a very large value of $u \gg 0$ is chosen, then $f$ is neglectably small in the minimization problem, and the problem may be approximated as $\min_{x \in \mathbb{R}} -uh(x) = \max_{x \in \mathbb{R}} h(x)$. This problem provides a solution that satisfies the boundary condition, but does not provide any optimum on the original function. Now, let $u$ smoothly vary in between those two extremes. At some point, there then obviously exists a balance between the optimization of $f$ and the restriction to $h$. That point may probably both be a minimum for $f$ on the optimization domain, and a feasible point for $h$.

In total, a minimum on the Lagrange function for a certain Lagrange multiplier $u$ may be the solution to the nonlinear minimization problem.

Before a more formal representation of this intuitive explanation can be derived, another restriction needs to be made on the optimization problem. In particular, a degenerate case exists where all feasible points are located only on the niveau line $h_i(x) = 0$ of any boundary inequality. In terms of the above example, this means the Lagrange function cannot balance this inequality with the original function, since any balancing actions immediately yield but infeasible points.

There are several constraints that can be introduced into the nonlinear problem to regularize such situations. One of the more popular of these is Slater's constraint qualification (or Slater's condition for short):

**Definition 2.7. Slater's constraint qualification (SCQ)**

Given (NLP), the additional requirement

$$\exists x' \in P \qquad \text{so that } h_i(x') > 0 \text{ for } i = p+1, \ldots, n \tag{2.12}$$

is called Slater's constraint qualification.

Intuitively, Slater's constraint qualification requires that there is a feasible point somewhere within the convex optimization domain that is not touched by any of the boundary inequalities. In other words, (SCQ) ensures that there is some feasible space in between the boundary inequalities that may be used for balancing by the Langrange function. This is visualized in figure 2.5.

Under the above requirement, the example statement on the minimum of the Lagrange function for a certain pair of Lagrange multipliers may then be formalized as such [FJ04]:

**Proposition 2.1.** *Given a convex (NLP) that satisfies (SCQ).*
*Let there be a finite* $\alpha := inf\{f(x) \mid x \in P\} \in \mathbb{R}$.

*Then there is a vector* $u \in \mathbb{R}^n$ *of Lagrange multipliers with* $u \geq 0$ *so that:*

$$L(x,u) \;=\; f(x) - \sum_{i=1}^{n} u_i h_i(x) \;\geq\; \alpha \qquad\qquad \forall x \in \mathbb{R}^m. \tag{2.13}$$

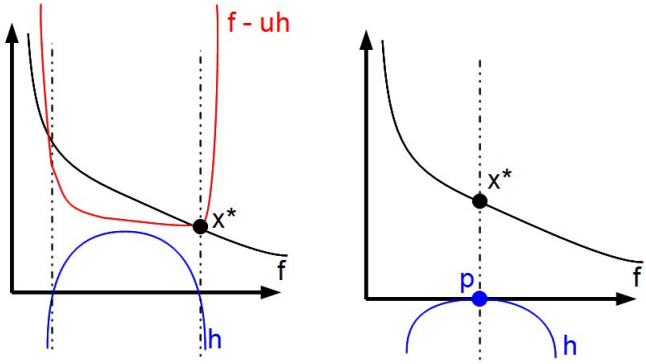



Figure 2.5: These two images show the importance of Slater's condition. On the left, Slater's condition is satisfied by any points in the optimization domain. Thus, an $u > 0$ may be found so that the Lagrange function $L(x,u) = f - uh(x)$ has a minimum exactly at the minimum $x^*$ of the optimization problem. On the right, just a single point $p$ is inside the optimization domain, and $h(p) = 0$. In this case, Slater's condition is not satisfied. Thus, it is not possible to find an appropriate $u > 0$ so that the Lagrange function has a minimum at $x^*$.

*Proof.* The concept of the following proof is to transform the problem into a higher-dimensional, combinatorical space. There, all possible constellations of boundary condition and objective function values form a convex set. Due to convexity and the fact that the origin is outside this set, one may find a not-null vector that points to a combination representing a feasible minimum on the nonlinear problem. Homogenizing this vector then gives the Lagrange multipliers $u$.

Let

$$\begin{aligned} A := \{v = (v_0, \ldots, v_n)^T \in \mathbb{R}^{n+1} \quad | \quad & \exists x \in \mathbb{R}^m : \\ & v_0 > f(x) - \alpha, \\ & v_i = h_i(x) \text{ for } i = 1, \ldots, p, \\ & v_i \leq h_i(x) \text{ for } i = p+1, \ldots, n\}. \end{aligned} \tag{2.14}$$

Analogous to lemma 2.3, $A$ can be shown to be convex. $A$ is not empty, since the optimization domain is not empty. Finally, $0 \notin A$, because $v_0 > f(x) - \alpha \underset{\alpha \text{ inf.}}{\geq} 0$.

Based on these properties of $A$, separability theory for convex sets[1] states that there is a $z = (z_0, \ldots, z_n) \in \mathbb{R}^{n+1} \neq 0$, so that $z^T v \geq 0$ for all $v \in A$ and so that $z^T \tilde{v} > 0$ for some $\tilde{v} \in A$.

By the definition of $A$, $v_0$ has a lower, but no upper bound, while $v_{p+1}, \ldots, v_n$ have upper, but no lower bounds. Then, satisfying

$$z^T v \geq 0 \;\Leftrightarrow\; z_0 v_0 + \ldots + z_{p+1} v_{p+1} + \ldots + z_n v_n \geq 0 \tag{2.15}$$

for all possible $v \in A$ requires $z_0 \geq 0$ and $z_i \leq 0$ for $i = p+1, \ldots, n$.

There also is a

$$v_\epsilon(x) := \begin{pmatrix} f(x) + \epsilon - \alpha \\ h_1(x) \\ \vdots \\ h_n(x) \end{pmatrix} \in A \tag{2.16}$$

for all $\epsilon \geq 0$ and all $x \in \mathbb{R}^m$.

Thus

$$0 \;\leq\; z^T v_\epsilon \;=\; z_0\big(f(x) + \epsilon - \alpha\big) + \sum_{i=1}^{n} z_i h_i(x) \qquad \forall \epsilon > 0,\ \forall x \in \mathbb{R}^m \tag{2.17}$$

gives, together with continuity of involved functions,

$$z_0\big(f(x) - \alpha\big) + \sum_{i=1}^{m} z_i h_i(x) \quad \geq \quad 0 \qquad \forall\, x \in \mathbb{R}^m \tag{2.18}$$

for $\epsilon \to 0$.

Furthermore, $z_0$ is not 0. This is proven by contradiction:
Assume $z_0 = 0$. Let $x' \in \mathbb{R}^m$ be the point from (SCQ) with $h_i(x') > 0$ for $i = p+1, \ldots, n$. Since $x' \in P$, $h_i(x') = 0$ for $i = 1, \ldots, p$. Then, consider any

$$\bar{v} := \begin{pmatrix} f(x') + 1 - \alpha \\ h_1(x') = 0 \\ \vdots \\ h_p(x') = 0 \\ \bar{v}_{p+1} < h_{p+1}(x') \\ \vdots \\ \bar{v}_n < h_n(x') \end{pmatrix} \in A. \tag{2.19}$$

Again, $\bar{v}_{p+1}, \ldots, \bar{v}_n$ have no lower bound, but since $z_0 = 0$, only $z_{p+1} = z_{p+2} = \ldots = z_n = 0$ satisfy $z^T \bar{v} \geq 0$ in this case.

Inserting this $z$ into the separability results yields

$$\begin{aligned} z^T v &= z_1 v_1 + \ldots + z_p v_p \\ &= z_1 h_1(x) + \ldots + z_p h_p(x) \geq 0 \qquad \forall\, x \in \mathbb{R}^m, \end{aligned} \tag{2.20}$$

$$\begin{aligned} z^T \tilde{v} &= z_1 \tilde{v}_1 + \ldots + z_p \tilde{v}_p \\ &= z_1 h_1(\tilde{x}) + \ldots + z_p h_p(\tilde{x}) > 0 \qquad \text{for some } \tilde{x} \in \mathbb{R}^m. \end{aligned} \tag{2.21}$$

Now, consider a point $x' - \epsilon(\tilde{x} - x') \in \mathbb{R}^m$ for some small $\epsilon > 0$. Substituting in (2.20), exploiting possible linearity of the equation constraints, and considering (2.21) gives

$$\begin{aligned} 0 \leq z^T v \;&=\; \sum_{i=1}^{p} z_i h_i\big(x' - \epsilon(\tilde{x} - x')\big) \;=\; \sum_{i=1}^{p} z_i \Big(h_i(x') - \epsilon\big(h_i(\tilde{x}) - h_i(x')\big)\Big) \\ &=\; \sum_{i=1}^{p} z_i - \epsilon h_i(\tilde{x}) \;=\; -\epsilon \left(\sum_{i=1}^{p} z_i h_i(\tilde{x})\right) \;<\; 0 \qquad \lightning \end{aligned} \tag{2.22}$$

Finally, since $z_0 > 0$,

$$\begin{aligned} z_0(f(x) - \alpha) + \sum_{i=1}^{n} z_i h_i(x) \quad &\geq \quad 0 \qquad \Leftrightarrow \\ f(x) - \alpha + \sum_{i=1}^{n} \frac{z_i}{z_0} h_i(x) \quad &\geq \quad 0 \qquad \Leftrightarrow \\ f(x) - \sum_{i=1}^{n} \left(-\frac{z_i}{z_0} h_i(x)\right) \quad &\geq \quad \alpha. \end{aligned} \tag{2.23}$$

Substituting

$$u := \left(-\frac{z_1}{z_0}, \ldots, -\frac{z_n}{z_0}\right)^T \tag{2.24}$$

in (2.23) gives (2.13), and thus the above proposition. □

### 2.3.2 KKT Conditions

Another, more common formulation for the results of the last subsection is given as [KS08]:

**Proposition 2.2.** *Given a convex (NLP) that satisfies (SCQ), and the minimal point $x^*$ within $P$.*

*Then, there exists a vector of Lagrange multipliers $u^* \in \mathbb{R}^n$, so that the following conditions are satisfied:*

$$u_j^* \geq 0 \qquad \text{for } j = p+1, \ldots, n \qquad \text{(Positivity)}$$
$$\nabla_x L(x^*, u^*) = 0 \qquad \text{(Stationarity)}$$
$$u_j^* h_j(x^*) = 0 \qquad \text{for } j = p+1, \ldots, n. \qquad \text{(Complementarity)}$$

*Proof.* If $x^*$ is a minimal point, then according to proposition 2.1, there is an $u^* \geq 0$ so that

$$f(x) - \sum_{i=1}^{n} u_i^* h_i(x) \;\geq\; f(x^*) \qquad \forall x \in P \tag{2.25}$$

This $u^*$ already satisfies the (Positivity) condition.

Now, consider the convex and differentiable function

$$\phi : \mathbb{R}^n \to \mathbb{R}$$
$$\phi(x) := f(x) - f(x^*) - \sum_{i=1}^{n} u_i^* h_i(x). \tag{2.26}$$

Since $x^* \in P$, $h_1(x^*) = 0, \ldots, h_p(x^*) = 0$ and $h_{p+1}(x^*) \geq 0, \ldots, h_n(x^*) \geq 0$, and thus $\phi(x^*) = -\sum_{i=1}^{n} u_i^* h_i(x^*) \leq 0$. Since additionally $\phi(x) \geq 0$ for all $x \in \mathbb{R}^n$, $\phi(x^*) = 0$, and $x^*$ also is the minimum of $\phi$. Thus,

$$\nabla\phi(x^*) = \nabla f(x^*) - \sum_{i=1}^{n} u_i^* \nabla h_i(x^*) = \nabla_x L(x^*, u^*) = 0 \tag{2.27}$$

gives the (Stationarity) condition.

[1] It is intuitively understood that there is a separating hyperplane with a normal vector $z$ that satisfies $z^T v \geq 0$ and $z^T \tilde{v} > 0$ for each convex set that does not contain the origin. However, the associated mathematical proof in separability theory is rather extensive, and outside the scope of this thesis. Indepth explanation of the fundamental rules can be found in [FJ04] or in [WW06].

Assuming there were any $u_j^* h_j(x^*) \neq 0$ for $j = p+1, \ldots, n$ leads to a contradiction, as in this case $\phi(x^*) = \sum_{i=p+1}^{n} - u_i^* h_i(x^*) > 0$. Thus, (Complementarity) also is ensured. □

The three conditions from the above proposition are known as the KKT conditions[2].

Disregarding mathematical precision, an intuitive understanding of KKT conditions may be achieved by considering the gradients of the objective function and constraint functions in an infinitesimally small region around the point $x^*$. In particular, the stationarity condition can be expanded into

$$\nabla_x L(x^*, u^*) = \nabla f(x^*) - \sum_{j=1}^{n} u_j^* \nabla h_j(x^*) = 0 \qquad \Leftrightarrow$$

$$\nabla f(x^*) = \sum_{j=1}^{n} u_j^* \nabla h_j(x^*). \tag{2.28}$$

Thus, stationarity requires that the gradient of the objective function at the point $x^*$ can be written as a linear combination of the gradients of the restriction functions at $x^*$. Due to complementarity, only the gradients of active inequality constraints may be considered in this linear combination. The positivity condition enforces positive linear multipliers for all inequality constraints. In total, this only can be satisfied by an objective function gradient that points into the feasible optimization domain. Since a convex optimization space is required, the entire negative half-space behind the objective function gradient is infeasible. Thus, no further descent is possible without violating any constraint, and $x^*$ is a minimum. This explanation also is visualized in figure 2.6.

KKT conditions are a vital tool in qualifing the optimality of a point. In particular, under the preceding requirements, KKT conditions provide a necessary condition for the existence of a minimum on a convex optimization problem.

As such, they form the basis for many mathematical appropaches to nonlinear optimization problems. For instance, the sequential quadratic programming and interior point methods that are the topic of this thesis rely heavily on KKT conditions.

## 2.4 Newton Methods

As derived in the preceding section, the KKT conditions give a set of necessary conditions for a minimum on a convex, nonlinear programming problem. In particular, a series of equations such as $\nabla_x L(x, u) = 0$ or $u_j h_j(x) = 0$ must be satisfied for some given $x$ and $u$.

[2]The renowned KKT conditions have been named after their inventors, W. Karush, H. Kuhn, and A. W. Tucker. Based on earlier work by W. Karush, the KKT conditions were first introduced to the public in 1951 on a symposium at Berkeley University, California [HK51].

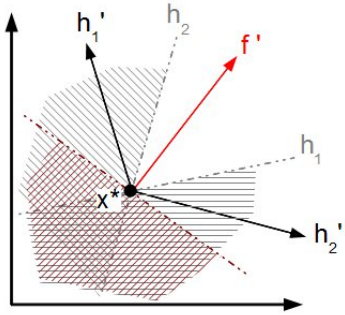



Figure 2.6: This figure shows the gradient $f'(x^*)$ of an objective function $f$ and the gradients $h_1'(x^*)$ and $h_2'(x^*)$ of active, convex boundary inequalities $h_1$ and $h_2$. Grey-hatched space is infeasible, un-hatched space represents the optimization domain. The KKT conditions require that $f'$ be a positive linear combination of $h_1'$ and $h_2'$, thus be anywhere in valid optimization domain. The negative halfspace of $f'$, indicated by red hatches, then always is outside feasible space. Thus, no further descent is possible without violating any restriction, and $x^*$ is a local minimum.

As all involved equations require some expression to vanish, an intuitive approach to determine appropriate $x$ and $u$ implies finding a root on the equations. Albeit actual strategies typically are more involved, root finding still acts as a motivation for many of these - in particular, for the later sequential quadratic programming and interior point methods.

In this context, the following section provides an overview over the well-known Newton method for root finding.

Additionally, some foundations are given on alternative, quasi-Newton methods that give performance benefits at the expense of accuracy. As an example of one such method, the BFGS update strategy is introduced.

### 2.4.1 Root Finding

Newton's root finding method provides a general way to find a zero point on any twice continuously differentiable function. Albeit an unmodified Newton method does not exhibit desireable convergency behaviour for certain problems, it still can be used to motivate more elaborate approaches, such as the later sequential quadratic programming method.

As the simplemost scenario, consider finding the root on a nonlinear, continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$.

Given some starting point $x_k \in \mathbb{R}$, the algorithm begins with finding an analytical approximation to $f$ in $x_k$. The approximation is chosen such that its root is easily

calculated by algebraic means. The algorithm continues to locate the zero point $x_{k+1} \in \mathbb{R}$ on the approximated function. If the approximation was any good, $x_{k+1}$ is now closer to the root of $f$ than $x_k$ was. The algorithm may run yet another iteration with $x_{k+1}$ as the new starting point to get even closer to the desired zero point of $f$. This continues until the function value for the current iteration point finally is near enough to zero.

In mathematical sense, a sequence of iterate points $(x_k)$ is generated from a given initial point $x_0$. At each iteration, the analytical approximation of $f$ is constructed as the first order taylor polynomial $\tilde{f}_{x_k}$ through $x_k$:

$$\tilde{f}_{x_k}(\Delta x_k) := f(x_k) + f'(x_k)\Delta x_k. \tag{2.29}$$

Since $\tilde{f}_{x_k}$ is linear, its zero point is easily found by algebraic methods. This yields an update step

$$\tilde{f}_{x_k}(\Delta x_k) = f(x_k) + f'(x_k)\Delta x_k = 0 \qquad \Leftrightarrow \qquad \Delta x_k = -\frac{f(x_k)}{f'(x_k)}. \tag{2.30}$$

Finally, the resulting step $\Delta x_k$ may be applied to the preceding point $x_k$:

$$x_{k+1} := x_k + \Delta x_k = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{2.31}$$

As an enhancement, Newton's descent can be extended to any function

$$f : \mathbb{R}^m \to \mathbb{R}^n \qquad f(x) := \begin{pmatrix} f_1(x_1, \ldots, x_m) \\ \vdots \\ f_n(x_1, \ldots, x_m) \end{pmatrix}. \tag{2.32}$$

In this case, linearizing the function $f$ involves calculating the Jacobian matrix $J_f(x)$. The resulting affine approximation is given by

$$\tilde{f}_{x_k}(\Delta x_k) := f(x_k) + J_f(x_k)\Delta x_k. \tag{2.33}$$

Finally, the update step that gives a root on the approximation can be found by solving the equation system

$$J_f(x_k)\Delta x_k = -f(x_k). \tag{2.34}$$

While this approach to root finding sounds rather reasonable, it has many disadvantages. For instance, for certain functions $f$ or for certain starting points $x_0$, there may be a cycle in the sequence $(x_k)$. Thus the algorithm never comes close enough to a function value of zero, but rather loops endlessly. An alternative problem exists for any saddle points or optima on $f$, where $J_f(x_k) = 0$. The naive approach does not handle these correctly, either.

Examples for both convergent and cycling Newton's descent are depicted in figure 2.7.

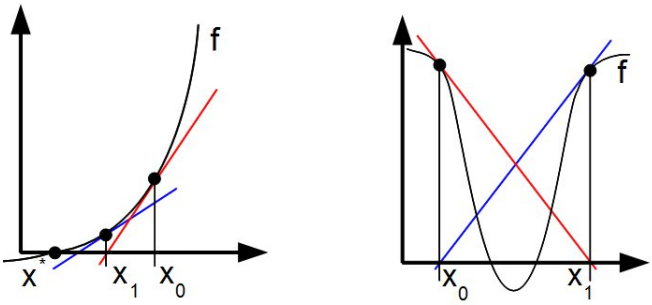



Figure 2.7: In the left-hand side image, Newton's descent correctly converges to the root $x^*$ of $f$. The coloured lines indicate linear approximations at each iteration point. On the right-hand side, the iteration cycles endlessly between two points $x_0$ and $x_1$.

However, under certain restrictions on the involved function and variables, Newton's descent always converges. In particular, quadratical convergency to a root is ensured if the starting point is near enough to the root already and if the root is not a saddle point or an optimum of $f$. This intuitive connotion is presented in the following mathematical statement:

**Lemma 2.4.** *Given a function $f : \mathbb{R}^m \to \mathbb{R}^n$. Let there be a point $x^* \in \mathbb{R}^m$ with $f(x^*) = 0$ and $J_f(x^*) \neq 0$.*

*Then, there exists a $\delta > 0$, so that any sequence $(x_k)$ with $x_{k+1} := x_k - {J_f}^{-1}(x_k) f(x_k)$ converges to $x^*$ for any $x_0$ with $\|x^* - x_0\| < \delta$.*

*For some constant $c > 0$, any two consecutive elements of any such sequence satisfy*

$$\|x_{k+1} - x^*\| \leq c\|x_k - x^*\|^2. \tag{2.35}$$

*Proof.* An elaborate proof is outside the scope of this thesis. See, for instance, [WW06] or [FJ04]. □

### 2.4.2 Quasi-Newton Methods

In the following, consider root finding on a difficult to compute function $f : \mathbb{R}^m \to \mathbb{R}^m$ for very large $m$.

In this case, one major problem with Newton's descent becomes obvious. In particular, the Jacobian matrix $J_f(x_k) \in \mathbb{R}^{m,m}$ needs to be recalculated for every step of the iteration. Since function evaluations of $f$ are difficult and $J_f$ has many elements, such recalculations become prohibitively costy.

Approaches known as quasi-Newton methods avoid complete recalculations at the expense of accuracy: Instead of the Jacobian, an approximation $B_k \approx J_f(x_k)$ is used

at each step. Then, $B_{k+1}$ may be derived from $B_k$ via a more performantly calculated update.

The update that derives $B_{k+1}$ from $B_k$ needs to be chosen diligently. In particular, a vital requirement is to keep up about the same level of superlinear convergency the base Newton method exhibits.

One such set of rules that describes conditions for the upkeep of convergency with an approximated Jacobian is given in the following proposition:

**Proposition 2.3.** *Given a function $f : \mathbb{R}^m \to \mathbb{R}^m$.*

*Let there be an $x^* \in \mathbb{R}^m$, so that $f(x^*) = 0$ and $J_f(x^*) \neq 0$.*

*Further, let $\|J_f(x) - J_f(x^*)\| \leq c\|x - x^*\|$ for some constant $c > 0$.*

*Finally, let there be a sequence $(x_k)$ with $\lim\limits_{k\to\infty} x_k = x^*$ and $x_{k+1} := x_k - B_k^{-1} f(x_k)$ for nonsingular matrices $B_k \in \mathbb{R}^{m,m}$.*

*With $\Delta x := x_{k+1} - x_k$ and $\Delta y := y_{k+1} - y_k$, the following conditions are equivalent:*

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0 \tag{2.36}$$

$$\lim_{k\to\infty} \frac{\|\big(B_k - J_f(x^*)\big)\Delta x\|}{\|\Delta x\|} = 0 \tag{2.37}$$

$$\lim_{k\to\infty} \frac{\|B_k \Delta x - \Delta y\|}{\|\Delta x\|} = 0. \tag{2.38}$$

*Proof.* Again, an elaborate proof is outside the scope of this thesis. For instance, a proof is provided in [FJ04]. □

Rule (2.36) of the above proposition indicates the desired convergency behaviour. In colloquial english, this rule means that, for large $k$, each new point $x_{k+1}$ in the current iteration is much closer to the actual zero point than the preceding sequence element $x_k$ if an approximation $B_k$ to the Jacobian is used. To achieve this behaviour, it is sufficient to satisfy any of the other rules.

Thus, rule (2.38) provides the starting point for deriving an appropriate matrix approximation $B_k$. Ideally, matrix $B_k$ should be chosen so that $B_k \Delta x = \Delta y$. However, $B_k$ also is involved in the calculation of $\Delta x = -B_k^{-1} f(x_k)$. Consequently, a-priori selection of an appropriate $B_k$ is not an easy task.

Instead, an update requirement

$$B_{k+1} \Delta x = \Delta y \tag{2.39}$$

is applied to the Jacobian approximation. As this requirement only partially fulfills rule (3), it does not enforce the desired convergency. Yet, it comes reasonably close, in particular if the $B_k$s do not drastically change in between iterations.

The above update requirement (2.39) represents the core of quasi-Newton methods: While there exist many approaches to actually determine the matrix $B_{k+1}$ under this requirement, any of these inherit quasi-Newtonian convergency properties, similar to the original Newton's descent.

### 2.4.3 BFGS Updates

A very well known quasi-Newton approach - and an approach that found its way into the NLPIP nonlinear solver in modified form - is the BFGS update[3] strategy.

In particular, the corresponding matrix update rule is described as

$$A_+ = A + \frac{(p - Aq)p^T + p(p - Aq)^T}{p^T q} - \frac{(p - Aq)^T q}{(p^T q)^2} pp^T \tag{2.40}$$

where $A := B_k^{-1}$,
$A_+ := B_{k+1}^{-1}$,
$p := \Delta x$,
and $q := \Delta y$.

Using an approximation to the inverse of $B_k^{-1}$ yields an important advantage. In particular, for performing a single Newton step, no matrix inversion is required anymore, but the step $\Delta x_k$ directly calculates as

$$\Delta x_k = -B_k^{-1} f(x_k) = -Af(x_k). \tag{2.41}$$

Another benefit of the BFGS update rule is its upkeep of symmetry. This is important in minimization scenarios, where root finding is applied on the first derivative of some objective function. Then, the Jacobian of the root finding problem matches up with the Hessian matrix of the Lagrange function. The original Hessian matrix is symmetric, and the approximation should reflect that property.

The derivation of the BFGS approximation $A_+$ and its properties is based on the following proposition [FJ04]:

**Proposition 2.4.** *Let $M \in \mathbb{R}^{m,m}$ be a symmetric, nonsingular matrix. Further let $p, q \in \mathbb{R}^m$, $q \neq 0$, and $c := M^{-2}q$. Finally, let $A \in \mathbb{R}^{m,m}$ be a symmetric matrix.*

*Then,*

$$\min_{\tilde{A}} \{ \left\| M(\tilde{A} - A)M \right\|_F \mid \tilde{A} = \tilde{A}^T,\ \tilde{A}q = p \} \tag{2.42}$$

*is satisfied by*

$$A_+ := A + \frac{(p - Aq)c^T + c(p - Aq)^T}{c^T q} - \frac{(p - Aq)^T q}{(c^T q)^2} cc^T. \tag{2.43}$$

[3] The BFGS update was named after its inventors, C. G. Broyden, R. Fletcher, D. Goldfarb, and D. F. Shanno, who independently suggested similar strategies.

*Proof.* $A_+$ is symmetric, as all matrix addends in the defining equation (2.43) are symmetric. $A_+$ also complies to the condition $A_+q = p$, proof by substitution:

$$\begin{aligned} A_+q &= Aq + (p - Aq)\frac{c^Tq}{c^Tq} + c\frac{(p - Aq)^Tq}{c^Tq} - \frac{(p - Aq)^Tq}{(c^Tq)^2}cc^Tq \\ &= Aq + (p - Aq) + c\frac{(p - Aq)^Tq}{c^Tq} - \frac{(p - Aq)^Tq}{c^Tq}c = p \end{aligned} \tag{2.44}$$

Thus $A_+$ belongs to the matrix set (2.42) that is minimized over.

Now, consider an arbitrary orthonormal base $\{u_1, \ldots, u_m\}$ of the $\mathbb{R}^m$. Since the $u_i$ form an orthonormal base, $u_i^Tu_j = \delta_{ij}$ for all $i, j \in \{1, \ldots, m\}$. Let $U := (u_1, \ldots, u_m) \in \mathbb{R}^{m,m}$ be the matrix associated with this base.

Then each matrix $E = (e_1, \ldots, e_m) \in \mathbb{R}^{m,m}$ satisfies

$$\begin{aligned} \|E\|_F^2 = \|E^T\|_F^2 = \|(e_1, \ldots, e_m)\|_F^2 = \sum_{i=1}^m \|e_i\|_2^2 = \sum_{i=1}^m \|U^Te_i\|_2^2 = \\ \|(U^Te_1, \ldots, U^Te_m)\|_F^2 = \|U^TE^T\|_F^2 = \|EU\|_F^2 = \\ \|(Eu_1, \ldots, Eu_n)\|_F^2 = \sum_{i=1}^m \|Eu_i\|_2^2. \end{aligned} \tag{2.45}$$

Let $\tilde{E} := M(\tilde{A} - A)M$ for some $\tilde{A}$ within the minimization set (2.42), let $E_+ := M(A_+ - A)M$, and let $z := Mc = M^{-1}q$. For these,

$$\begin{aligned} (p - Aq)^TM = p^TM - q^TAM = (\tilde{A}q)^TM - q^TAM = q^T\tilde{A}M - q^TAM = \\ q^T(\tilde{A} - A)M = q^TM^{-1}M(\tilde{A} - A)M = z^T\tilde{E}, \end{aligned} \tag{2.46}$$

$$c^Tq = (M^{-2}q)^Tq = (q^TM^{-1})(M^{-1}q) = z^Tz, \tag{2.47}$$

$$Mcc^TM = zz^T. \tag{2.48}$$

Simplifying the definition for $E_+$ with these results yields

$$\begin{aligned} E_+ &= M(A_+ - A)M \\ &= MAM + \frac{M(p - Aq)c^TM + Mc(p - Aq)^TM}{c^Tq} \\ &\qquad - \frac{M(p - Aq)^TqM}{(c^Tq)^2}cc^T - MAM \\ &= \ldots \\ &= \frac{\tilde{E}zz^T + zz^T\tilde{E}}{z^Tz} - \frac{z^T\tilde{E}z}{(z^Tz)^2}zz^T. \end{aligned} \tag{2.49}$$

Multiplying some $v$ with $v^T z = 0$ onto the above gives

$$\begin{aligned}\|E_+ v\|_2 &= \left\| z \frac{z^T \tilde{E} v}{z^T z} \right\|_2 \\ &\leq \left\| z z^T \right\|_2 \left\| \frac{\tilde{E} v}{z^T z} \right\|_2 = z^T z \frac{\|\tilde{E} v\|_2}{z^T z} = \|\tilde{E} v\|_2 \end{aligned} \tag{2.50}$$

Additionally, the definition of $z$ indicates

$$\begin{aligned} &E_+ z = E_+ M^{-1} q = M(A_+ - A)q = M(\tilde{A} - A)q = \tilde{E} z \Rightarrow \\ &\quad \|E_+ z\|_2 = \|\tilde{E} z\|_2 \end{aligned} \tag{2.51}$$

Finally, find another $m-1$ vectors $v_i \in \mathbb{R}^m$, $i = 1, \ldots, m-1$, that complement $\frac{z}{\|z\|}$ to an orthonormal base. Let $U$ above be this orthonormal base. Then, the proposition is proven:

$$\begin{aligned}\|E_+\|_F &= \sum_{i=1}^{m} \|E_+ u_i\|_2^2 = \|E_+ z\|_2^2 + \sum_{i=1}^{m-1} \|E_+ v_i\|_2^2 \\ &\leq \|\tilde{E}_+ z\|_2^2 + \sum_{i=1}^{m} \|\tilde{E} v_i\|_2^2 = \|\tilde{E}\|_F \end{aligned} \tag{2.52}$$

□

Essentially, the above proposition states that, given a certain symmetric weighting matrix $M$, it is possible to find a symmetric matrix $A_+$ that is close to the preceding iteration matrix $A$ in regards to the Frobenius Norm. This new matrix $A_+$ then also satisfies the update requirement for quasi-Newton methods. Various update strategies may thus be derived by applying various weighting matrices $M$.

The BFGS update is achieved by choosing an $M$ with $p = M^{-2} q$. Substituting this into the more general definition of $A_+$ provided in the preceding proposition then gives the associated update rule (2.40). Finally, the proposition also guarantees the symmetry of $A_+$ and thus of $B_{k+1}$ for the BFGS update, based on the assumption that the matrix $A$ respective $B_k$ of the preceding iteration was symmetric as well.

The only remaining point concerns the selection of the first matrix approximation $B_0$ or $A_0$. While there also are various methods here - albeit more influenced by storage and sparsity requirements - the most naive approach is to simply let $A_0^{-1} := B_0 := J_f(x_0)$.

## 2.5 Sequential Quadratic Programming

In the previous sections, optimality conditions for convex problems have been derived. These can be used to determine if any given point on the optimization domain actually is a minimum. Additionally, simple root finding strategies have been introduced. On

this background, the following section provides an actual algorithm for finding a minimal point on a convex problem.

In particular, the sequential quadratic programming method is discussed. In this method, the base nonlinear problem is iteratively approximated by quadratic problems. The solution of each quadratic problem then is used to determine a descent direction within the original nonlinear problem.

In the following, quadratic programs first are introduced in a general form. Then, a general sequential quadratic programming algorithm is derived. Finally, mathematical formulations are given for the concrete algorithm of the NLPIP implementation.

### 2.5.1 Quadratic Programs

The category of quadratic programs forms a special case of nonlinear programs. For quadratic programs, the objective function is restricted to a quadratic form

$$f : \mathbb{R}^m \to \mathbb{R}$$
$$f(d) := d^T A d + b^T d \tag{2.53}$$

where $A \in R^{m,m}$ is a symmetric matrix and $b \in R^m$ is an arbitrary vector. All boundary functions $h_1, \ldots, h_n$ need to be affine within a quadratic problem. Thus all restrictions can be expressed as

$$H_e d + c_e = 0 \tag{2.54}$$
$$H_i d + c_i \geq 0 \tag{2.55}$$

with a matrix $H_e \in \mathbb{R}^{m,p}$ and a vector $c_e \in \mathbb{R}^p$ for the $p$ equation constraints, and a matrix $H_i \in \mathbb{R}^{m,q}$ and a vector $c_i \in \mathbb{R}^q$ for the $q$ inequalities.

The resulting, abstract problem can then be formulated as:

**Definition 2.8. Quadratic Program (QP)**

$$\min_{d \in \mathbb{R}^m} d^T A d + b^T d \tag{2.56}$$
$$\text{so that:} \quad H_e d + c_e = 0$$
$$H_i d + c_i \geq 0.$$

If the matrix $A$ additionally is known to be positive semidefinite, then $f$ is convex. Since the linearized boundary conditions already are concave, respectively affine, the entire problem is convex in this case.

For some minimal point $d^* \in \mathbb{R}^m$ and associated Lagrange multipliers $\lambda^* \in \mathbb{R}^n$, the KKT conditions of the quadratic problem are given as

$$2Ad^* + b - (H_e \oplus H_i)^T \lambda^* = 0, \tag{2.57}$$
$$\lambda^* \geq 0, \tag{2.58}$$
$$\lambda^* \odot (H_i\ d^*) = 0. \tag{2.59}$$

### 2.5.2 Algorithm Derivation

As an entry point into the actual sequential quadratic programming method (SQP), reconsider the KKT conditions presented in proposition 2.2:

Let there be given a convex, nonlinear problem that satisfies Slater's constraint qualification. Motivated by KKT conditions and the original problem formulation, define a function

$$\Gamma : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^{n+1}$$

$$\Gamma(x,u) := \begin{pmatrix} \nabla_x L(x,u) = \nabla f(x) - \sum_{i=1}^{n} u_i \nabla h_i(x) \\ -h_1(x) \\ \vdots \\ -h_p(x) \\ -u_{p+1} h_{p+1}(x) \\ \vdots \\ -u_n h_n(x) \end{pmatrix}. \tag{2.60}$$

The minus signs for equality constraints and complementarity conditions have artificially been introduced so that the Jacobian of $\Gamma$ becomes symmetric. Expanding the Jacobian thus gives the symmetric matrix

$$J_\Gamma(x,u) = \tag{2.61}$$

$$\begin{pmatrix} \nabla_x^2 L(x,u) & -\nabla h_1(x) & \dots & -\nabla h_p(x) & -\nabla h_{p+1}(x) & \dots & -\nabla h_n(x) \\ -\nabla h_1^T(x) & & & & & & \\ \vdots & & 0 & & & 0 & \\ -\nabla h_p^T(x) & & & & & & \\ -u_{p+1}\nabla h_{p+1}^T(x) & & & & -h_{p+1}(x) & & 0 \\ \vdots & & 0 & & & \ddots & \\ -u_n \nabla h_n^T(x) & & & & 0 & & -h_n(x) \end{pmatrix}.$$

As $\Gamma$ represents part of the KKT conditions and part of the original nonlinear problem formulation, it is directly seen that any given minimum $x^* \in P$ satisfies the equation $\Gamma(x^*, u^*) = 0$ for some Lagrange multipliers $u^* \in \mathbb{R}^n$, $u^* \geq 0$.

The basic concept now is to find some point $x \in \mathbb{R}^m$ and associated Lagrange multipliers $u \in \mathbb{R}^n$ so that $\Gamma(x,u) = 0$. The point $x$ then may probably be a minimum.

For this, an adapted Newton's descent is applied onto the function $\Gamma$, similar to the original Newton's method described in subsection 2.4.1. In particular, a sequence of iterates $(x_k, u_k)$ is generated from starting values $x_0 \in P$ and $u_0 \geq 0$ by the step

$$(x_{k+1}, u_{k+1}) := (x_k, u_k) + (\Delta x_k, \Delta u_k). \tag{2.62}$$

$\Delta x_k$ and $\Delta u_k$ are determined by a slightly modified update rule

$$\big(J_\Gamma(x_k, u_{k+1})\big)(\Delta x_k, \Delta u_k) = -\Gamma(x_k, u_k). \tag{2.63}$$

This gives a nonlinear equation system for $\Delta x_k$ and $\Delta u_k$.

To ensure that the resulting $x_{k+1}$ and $u_{k+1}$ still are feasible in the original problem, additional requirements

$$h_i(x_{k+1}) \;=\; h_i(x_k) + \nabla h_i^T(x_k)\Delta x_k \;\geq\; 0 \qquad \text{for } i = p+1, \ldots, n \tag{2.64}$$

$$u_{k+1\,i} \;=\; u_{k\,i} + \Delta u_{k\,i} \;\geq\; 0 \qquad \text{for } i = 1, \ldots, n \tag{2.65}$$

are imposed. In a later implementation, these are enforced by picking an appropriate step length into the direction of $\Delta x_k$ and $\Delta u_k$.

Substituting the Jacobian at $(x_k, u_{k+1} = u_k + \Delta u_k)$ into the adapted update rule (2.63) yields the equation system

$$\nabla_x^2 L(x_k, u_{k+1})\, \Delta x_k - \sum_{i=1}^{n} \nabla h_i(x_k) \Delta u_{k\,i} \;=\; -\nabla f(x_k) + \sum_{i=1}^{n} u_{k\,i} \nabla h_i(x_k), \tag{2.66}$$

$$-\nabla h_i^T(x_k)\Delta x_k \;=\; h_i(x_k) \qquad \text{for } i = 1, \ldots, p, \tag{2.67}$$

$$-(u_{k\,i} + \Delta u_{k\,i})\nabla h_i^T(x_k)\Delta x_k - h_i(x_k)\Delta u_{k\,i} \;=\; u_{k\,i} h_i(x_k) \qquad \text{for } i = p+1, \ldots, n. \tag{2.68}$$

After simplification of the above, the final equation system is

$$\nabla_x^2 L(x_k, u_{k+1})\, \Delta x_k + \nabla f(x_k) - \sum_{i=1}^{n} u_{k+1\,i} \nabla h_i(x_k) \;=\; 0 \tag{2.69}$$

$$h_i(x_k) \;+\; \nabla h_i^T(x_k)\Delta x_k \;=\; 0 \qquad \text{for } i = 1, \ldots, p, \tag{2.70}$$

$$u_{k+1}\big(h_i(x_k) \;+\; \nabla h_i^T(x_k)\Delta x_k\big) \;=\; 0 \qquad \text{for } i = p+1, \ldots, n. \tag{2.71}$$

Substitution by

$$d^* := \Delta x_k, \qquad \lambda^* := u_{k+1}, \qquad A := \frac{1}{2}\nabla_x^2 L(x_k, u_{k+1}), \qquad b := \nabla f(x_k),$$

$$H_e := \begin{pmatrix} \nabla h_1^T(x_k) \\ \vdots \\ \nabla h_p^T(x_k) \end{pmatrix}, \qquad c_e := \begin{pmatrix} h_1(x_k) \\ \vdots \\ h_p(x_k) \end{pmatrix},$$

$$H_i := \begin{pmatrix} \nabla h_{p+1}^T(x_k) \\ \vdots \\ \nabla h_n^T(x_k) \end{pmatrix}, \qquad c_i := \begin{pmatrix} h_{p+1}(x_k) \\ \vdots \\ h_n(x_k) \end{pmatrix},$$

in requirement (2.65) and in conclusions (2.69) and (2.71) yields the KKT conditions for the general quadratic problem (QP) introduced earlier.

In other words, $\Delta x_k$ and $u_{k+1}$ match up with a point $d$ and associated Lagrange multiplier $\lambda$ that are the solution of

**Definition 2.9. Quadratic Approximation (QA)**

$$\begin{aligned} \min_{d \in \mathbb{R}^m} \; & \frac{1}{2}\, d^T\, \nabla_x^2 L(x_k, u_{k+1})\, d \;+\; \nabla f(x_k)^T\, d && (2.72) \\ \text{so that: } & \nabla h_i^T(x_k)\, d \;+\; h_i(x_k) \;=\; 0 && \text{for } i = 1, \ldots, p, \\ & \nabla h_i^T(x_k)\, d \;+\; h_i(x_k) \;\geq\; 0 && \text{for } i = p+1, \ldots, n. \end{aligned}$$

Thus, $\Delta x_k$ and $u_{k+1}$ can be obtained by iteratively solving a corresponding quadratic minimization problem.

One remaining issue concerns the Hessian of the Lagrange function. In particular, solving the quadratic problem in the above form requires the Hessian at $(x_k, u_{k+1})$ to determine $u_{k+1}$. This induces a circular dependency, similar to the one for quasi-Newton methods. As with quasi-Newton approaches, this is alleviated by approximating $\nabla_x^2 L(x_k, u_{k+1})$ with $\nabla_x^2 L(x_k, u_k)$.

Under the assumption that the Hessian matrix $\nabla_x^2 L(x_k, u_k)$ of the Lagrange function is positive semidefinite, the quadratic minimization problem then even is convex. In this case, the KKT conditions provide a sufficient condition for a minimum value.

Figure 2.8 provides an intuitive example for the idea behind sequential quadratic programming.

### 2.5.3 Algorithm Outline

Following the above derivation, a basic algorithm could be implemented as such [FJ04]:

```
Initialization:
Choose x_0 ∈ R^m,
       u_0 ∈ R^n with u_0 ≥ 0.

Iterate over k = 0, 1, ...
  Determine step direction Δx and u_{k+1} by solving (QPA)
  Set x_{k+1} := x_k + Δx
Next k
```

This outline only provides a rough overview over the entire approach. Certain points still remain to be discussed in the following sections.

Most importantly, the resulting quadratic subprogram has to be solved in some way or another. One possible approach to this, based on an inner solver with an interior point method, is described in the next section.

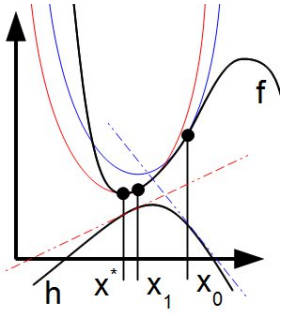



Figure 2.8: In this example for sequential quadratic programming, an objective function $f : \mathbb{R} \to \mathbb{R}$ is to be minimized under a single constraint $h : \mathbb{R} \to \mathbb{R}$. At each iteration point, a quadratic approximation of the objective function and a linearized representation of the constraint is generated, and the resulting minimization problem is solved to find the next iteration point. Approximations are indicated in colour in the above image.

In terms of enhancements, one popular adaption to this algorithm concerns the Hessian matrix $\nabla_x^2 L(x, u)$ of the Lagrange function. In particular, it is possible to replace this matrix with some per-step approximation. On the one hand, this avoids recalculating a costy and possibly large Hessian in each step of the iteration. On the other hand, it gives an opportunity to enforce positive semidefiniteness of the matrix. Positive semidefiniteness is important for convexity and thus for convergency in the quadratic problem, yet may be lost due to numerical instabilities. For instance, a BFGS update as described in subsection 2.4.3 may be used, with special provisions to ensure semidefiniteness. This correction strategy is detailed in the later enhancements section 2.7.

Further extensions include a better choice for the step length with a line search algorithm, and some heuristic for an appropriate stopping condition. Both of these topics are outside the scope of this thesis, but short reviews are presented in later sections, as well.

Including all these enhancements, the resulting outer SQP iteration becomes [BS08]:

Initialization:
Choose $x_0 \in \mathbb{R}^m$,
$u_0 \in \mathbb{R}^n$ with $u_0 \geq 0$.

Iterate over $k = 0, 1, \ldots$
If stopping criterion is satisfied $\rightarrow$ finish.

Determine step direction $\Delta x$ and $u'_{k+1}$ by solving (QPA).

Find step size $\alpha_k$ by line search along $\Delta x$,

```
    so that boundary constraints and positivity of the
    Lagrange multipliers are satisfied.

  Set x_{k+1} := x_k + α_k Δx
  Set u_{k+1} := u_k + α_k (u'_{k+1} − u_k)

  Perform BFGS updates on the approximation of ∇²_x L(x, u)
Next k
```

This closely matches the approach used in the actual implementation of NLPIP.

### 2.5.4 Convergency

Proving convergency of general SQP methods is difficult. In particular, there are many possible enhancements and implementation details that may influence convergency behaviour.

For instance, convergency analysis may be based on a BFGS update strategy instead of the more direct Newton's approach, with an enhancement to correct any indefinite points of the Hessian matrix on the way. Then, convergency behaviour is characterized by the following, Newton-like proposition [FJ04]:

**Proposition 2.5.** *Given (SQP) with a BFGS update and definiteness correction as in [MP78a]. Let* $z^* := (x^*, u^*)$ *be the minimum and associated Lagrange multipliers of the problem.*

*Then, convergency is locally superlinear, in that there are* $\epsilon > 0$, $\delta > 0$, *so that: If, for a sequence* $(z_k) := \big((x_k, u_k)\big)$, *it holds that* $\|z_0 - z^*\| \leq \delta$ *and* $\|B_0 - \nabla_x^2 L(x^*, u^*)\| < \epsilon$, *then all quadratic subprograms have a solution, and*

$$\lim_{k \to \infty} \|z_{k+1} - z^*\|^{\frac{1}{k}} = 0.$$

*Proof.* The corresponding proof is outside the scope of this thesis. An example approach may be found in [MP78b]. □

## 2.6 Interior Point Methods

As explained in the previous section, the convex, nonlinear program can be approximated by a series of quadratic programs. These can then be solved, and the resulting solution is used as the step direction for the next iteration.

This section deals with finding the actual solution within the quadratic program. In particular, the interior point method used within NLPIP is detailed here.

### 2.6.1 Introduction

Interior point methods originally had been introduced as an approach to linear programming, competing with the linear simplex method. The linear simplex method follows the boundaries of the optimization domain in search for an optimum. Thus it is affected by the associated combinatorical complexity of the problem space. Interior point methods - as the name already induces - apply a different strategy. Their optimization iterates lead through the interior of the optimization domain.

The fundamental concept behind this behaviour is the application of barrier terms. Barrier terms replace the harsh boundary inequalities of the original problem. Instead, barriers act upon the objective function, and penalize any approach to the borders of the optimization domain by increasing the associated objective function values. In particular, barrier terms provide finite penalty values within the domain. Towards the domain borders, the penalty implied by the barriers increases, and reaches infinity outside the optimization domain. Following the way of least penalization, the solution iterates for optimization on a barrier problem thus are led to the domain center.

While this concept had already been developed much earlier, its potential had not been recognized until the first interior point method was released as Karmarkar's algorithm in [NK84]. The associated merit, in particular, was the encoding of an arbitrary convex optimization domain into barrier terms. Then, the maximum complexity of the algorithm may be given as a polynomial that depends on the desired accuracy of the end result.

Although barrier terms originally were developed for linear programming, they can also successfully be applied onto nonlinear programming.

### 2.6.2 Barrier Methods

In the following, general barrier methods are explained as a motivation for the later interior point approach.

First, a more mathematical definition of the already introduced barrier terms is required. Barrier terms usually are formed based on an appropriate barrier function that models the penalty associated with leaving the optimization domain:

**Definition 2.10.** A barrier function is a real-valued, convex, continuously differentiable, strictly monotonically decreasing function $b : \mathbb{R}^+ \to \mathbb{R}$, where $\lim_{t \to 0+0} b'(t) = -\infty$ and where $\lim_{t \to 0+0} b(t) = \infty$.

Examples for such functions are $b(t) = -ln(t)$, $b(t) = \frac{1}{t}$ or $b(t) = \frac{1}{t^\alpha}$ [FJ04].

Barrier functions now are applied onto the optimization problem to form the actual barrier terms. In particular, a series of real-valued barrier functions is combined to create a higher-dimensional barrier over an entire convex optimization domain.

For this, let be given convex (NLP) on a function $f$, with associated $n = p + q$ restrictions based on $p$ boundary equations for functions $h_1, \ldots, h_p$, and $q$ boundary inequalities for functions $h_{p+1}, \ldots, h_n$.

As part of the barrier approach, a new objective function $\phi : \mathbb{R}^m \to \mathbb{R}$ is constructed from the original objective function $f$ by introducing barrier terms for some barrier function $b$:

$$\phi(x) := f(x) + \mu \sum_{i=p+1}^{n} b\big(h_i(x) - d_i\big) \tag{2.73}$$

where $\mu \in \mathbb{R}$, $\mu > 0$ represents a weighting factor for the barrier terms, and the vector $d \in \mathbb{R}^q$, $d \geq 0$ provides barrier offsets.

Intuitively speaking, the barrier offsets $d$ can be used to shrink or enlarge the area of influence for the barrier terms. For large $d$, the feasible area defined by $h_i(x) - d_i$ shrinks, as do the zero niveau lines. Thus, the barrier functions return $-\infty$ for many more points inside the original optimization domain, and increase the penalty for points that already are deep inside the feasible area.

Based on the new objective function $\phi$, the associated barrier minimization problem is given as:

**Definition 2.11. Barrier Problem (BP)**

$$\begin{aligned} &\min_{x \in \mathbb{R}^m} \ \phi(x) && && (2.74)\\ &\quad \text{so that:} && h_i(x) = 0 \qquad \text{for } i = 1, \ldots, p. \end{aligned}$$

Note that the boundary inequalities are not contained in this problem anymore, but have been replaced by the barrier terms.

An important observation on this minimization problem is the shape of the replacement objective function [FJ04]:

**Lemma 2.5.** *If $f$ is convex and $h_{p+1}, \ldots, h_n$ are concave, then $\phi$ is convex.*

*Proof.* First, show that $\varphi_i(x) := b\big(h_i(x) - d_i\big)$ is convex for all $i = p + 1, \ldots, n$.

For $\rho \in [0, 1]$ and for all $x \in \mathbb{R}^m, y \in \mathbb{R}^m$:

$$\begin{aligned} h_i\big(\rho x + (1-\rho)y\big) - d_i &\geq \big(\rho h_i(x) + (1-\rho)h_i(y)\big) - d_i \\ &= \big(\rho h_i(x) + (1-\rho)h_i(y)\big) - \rho d_i + \rho d_i - d_i \\ &= \big(\rho h_i(x) + (1-\rho)h_i(y)\big) - \rho d_i - (1-\rho)d_i \\ &= \rho\big(h_i(x) - d_i\big) + (1-\rho)\big(h_i(y) - d_i\big) \end{aligned} \tag{2.75}$$

and

$$\begin{aligned}\rho\varphi_i(x) + (1-\rho)\varphi_i(y) &= \rho b\big(h_i(x) - d_i\big) + (1-\rho) b\big(h_i(y) - d_i\big) \\ &\geq b\Big(\rho\big(h_i(x) - d_i\big) + (1-\rho)\big(h_i(y) - d_i\big)\Big) \\ &\geq b\Big(h_i\big(\rho x + (1-\rho)y\big) - d_i\Big) \\ &= \varphi_i\big(\rho x + (1-\rho)y\big), \end{aligned} \tag{2.76}$$

both under the use of the convexity and concavity definitions, as well as monotonic function properties. Thus, the $\varphi_i$ are convex.

Finally, if $g$, $\tilde{g}$ are convex functions, then $\lambda g + \nu\tilde{g}$ is convex for all $\lambda > 0, \nu > 0$. This is seen directly from the definition of convex functions. As $\phi$ is formed as such from the convex function $f$ and the convex functions $\mu\varphi_i = \mu b(h_i(x) - d_i)$ with $\mu > 0$, $\phi$ is convex. □

Thus, the barrier-based optimization problem again is a convex optimization problem, and appropriate optimality conditions such as the KKT conditions apply.

There also is a relation between the solution of the barrier-enhanced problem and the original minimization problem. In particular, if the barrier offsets $d$ are steadily decreased, and at the same time, the barrier weight $\mu$ is reduced, the optimal solution of the barrier problem approaches the optimal solution of the unmodified problem. This is represented by the following proposition:

**Proposition 2.6.** *Let be given a convex (NLP) with (SCQ) and a barrier function $b$ which satisfies $\lim_{t\to\infty} b(t) = 0$. Further, let $x^* \in \mathbb{R}^m$ be the solution for the optimization problem.*

*Then, the barrier-enhanced problem on the objective function*

$$\phi(x) = f(x) + \mu \sum_{i=p+1}^{n} b(h_i(x) - d_i)$$

*has a minimum for all $\mu > 0$ and all $d \geq 0$.*

*Furthermore, the minima $x^*(\lambda\mu, \lambda d)$ of appropriately parametrized barrier-enhanced problems approach the minimum of the original problem:*

$$\lim_{\lambda\to 0+0} \big(\|x^* - x^*(\lambda\mu, \lambda d)\|\big) = 0.$$

*Proof.* This proposition has been adapted from [FJ04] without proof. □

The above proposition forms the concept of a potential algorithm for solving any convex nonlinear programming problem with a simple barrier method. In particular, one could generate an iterative series of barrier problem approximations, and solve these. For each of the approximated problems, the barrier terms push the possible

solution away from the borders. The strength of this effect can be controlled by the offset vector $d$ and the general weight $\mu$. Usually, one knows that a solution to one barrier problem possibly is at least little closer to the solution of the original problem than the solution to the preceding barrier problem. Thus, at each iteration, the barrier terms may slightly be weakened by decreasing $d$ and $\mu$. This allows solutions to be dragged a little more towards the borders of the optimization domain in case the actual optimum is there. The resulting path then leads through the interior of the optimization domain. This also is visualized in figure 2.9.

### 2.6.3 Algorithm Derivation

Analogous to the above fundamental barrier method, a more sophisticated interior point algorithm may be constructed.

Starting point for the derivation of the interior point algorithm again is a convex (NLP) with its objective function and restrictions. Additionally, the requirement of (SCQ) is imposed.

Instead of directly applying barrier functions, the problem inequalities first are transformed into equalities by the introduction of $q$ slack variables $w_i \in \mathbb{R}$, $w_i \geq 0$,

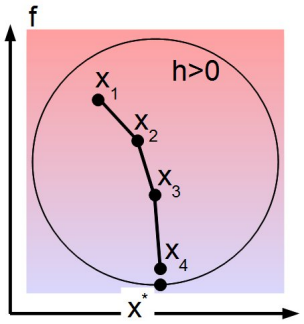



Figure 2.9: In this example for simple barrier methods, an objective function $f : \mathbb{R}^2 \to \mathbb{R}$ is to be minimized. Lower values of the objective function are represented by more bluish colors. There is a single inequality constraint $h : \mathbb{R}^2 \to \mathbb{R}$ that defines the circular optimization domain. $h$ is applied via a barrier term. In the first iteration, the barrier term influence is still large. Thus, iterate $x_2$ is pushed towards the center of the optimization domain. Then, barrier influence is gradually decreased, so $x_3$ and $x_4$ can again approach the border of the domain and the optimum $x_*$.

$i = p+1, \ldots, n$. The equivalent problem becomes

$$\begin{aligned} &\min_{x \in \mathbb{R}^m} f(x) && && (2.77)\\ \text{so that:} \quad &h_i(x) = 0 && \text{for } i = 1, \ldots, p \\ &h_i(x) - w_i = 0 && \text{for } i = p+1, \ldots, n \\ &w_i \geq 0 && \text{for } i = p+1, \ldots, n. \end{aligned}$$

Then, a barrier term is constructed from the inequalities on the slack variables, based on the barrier function $-ln(t)$. The resulting, barrier-enhanced objective function $\phi : \mathbb{R}^m \times \mathbb{R}^q \to \mathbb{R}$ is given as

$$\phi(x, w) := f(x) - \mu \sum_{i=p+1}^{n} ln(w_i), \tag{2.78}$$

where $\mu \in \mathbb{R}$, $\mu > 0$ again is a weighting coefficient. Compared to the earlier barrier method, the slack variables avoid the need for an explicit offset vector $d$ within the enhanced objective function.

Finally, the convex barrier minimization problem

$$\begin{aligned} &\min_{x \in \mathbb{R}^m, w \in \mathbb{R}^q} \phi(x, w) && && (2.79)\\ \text{so that:} \quad &h_i(x) = 0 && \text{for } i = 1, \ldots, p \\ &h_i(x) - w_i = 0 && \text{for } i = p+1, \ldots, n \end{aligned}$$

yields equivalent solutions to the original problem for a sufficiently small $\mu$. Note that minimization here is performed over both the original variables $x$ and the slack variables $w = (w_{p+1}, \ldots, w_n)^T$.

For further reference, the Lagrange function of the above problem is given as

$$L(x, w, u) := \phi(x, w) - \sum_{i=1}^{p} u_i h_i(x) - \sum_{i=p+1}^{n} u_i \big(h_i(x) - w_i\big). \tag{2.80}$$

Now, the KKT conditions and the boundary equalities are utilized to find an optimal point on the slack-enhanced, convex barrier problem. First, the requirements $u_i \geq 0$, $i = 1, \ldots, n$, and $w_i \geq 0$, $i = p+1, \ldots, n$, are imposed. These later on are enforced by an appropriate step selection. Then, an optimal point $x^*$, slack variables $w^*$, and associated Lagrange multipliers $u^*$ satisfy the following equations:

$$\begin{aligned} &\nabla_x L(x^*, w^*, u^*) = \nabla f(x^*) - \sum_{i=1}^{n} u_i^* \nabla h_i(x^*) = 0, && (2.81)\\ &\nabla_w L(x^*, w^*, u^*) = \begin{pmatrix} -\frac{\mu}{w_{p+1}^*} + u_{p+1}^* \\ \vdots \\ -\frac{\mu}{w_n^*} + u_n^* \end{pmatrix} = 0, \end{aligned}$$

$$
\begin{aligned}
h_i(x^*) &= 0 && \text{for } i = 1, \ldots, p, \\
h_i(x^*) - w_i^* &= 0 && \text{for } i = p+1, \ldots, n.
\end{aligned}
$$

As with the previously discussed SQP approach, the concept now is to locate $x$, $w$ and $u$ that satisfy the above conditions. For these, further simplification yields the equivalent system

$$\nabla f(x) - \sum_{i=1}^{n} u_i \nabla h_i(x) = 0, \tag{2.82}$$

$$
\begin{aligned}
&-\mu e + WUe = 0 \\
&h_i(x) = 0 && \text{for } i = 1, \ldots, p, \\
&h_i(x) - w_i = 0 && \text{for } i = p+1, \ldots, n.
\end{aligned}
$$

Here, $e := (1, \ldots, 1) \in \mathbb{R}^q$ is a vector of ones, whereas $W := diag(w_{p+1}, \ldots, w_n)$ and $U := diag(u_{p+1}, \ldots, u_n)$ are diagonal matrices created from slack variables and lagrange multipliers, respectively.

Finally, applying standard Newton's descent on this equation system results in an update rule

$$\begin{pmatrix} \nabla_x^2 L(x,w,u) & 0 & -J_H^T(x) \\ 0 & U & W \\ J_{H_2}(x) & -I & 0 \\ J_{H_1}(x) & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta w \\ \Delta u \end{pmatrix} = \begin{pmatrix} -\nabla f(x) + J_H^T(x) u \\ \mu e - WUe \\ -H_2(x) + w \\ -H_1(x) \end{pmatrix}, \tag{2.83}$$

where

$$H_1(x) := \begin{pmatrix} h_1(x) \\ \vdots \\ h_p(x) \end{pmatrix}, \qquad H_2(x) := \begin{pmatrix} h_{p+1}(x) \\ \vdots \\ h_n(x) \end{pmatrix}, \qquad H(x) := \begin{pmatrix} H_1(x) \\ H_2(x) \end{pmatrix}. \tag{2.84}$$

Iterative application of Newton's descent then gives the following, naive interior point algorithm [FJ04]:

```
Initialization:
Choose x_0 ∈ ℝ^m,
       w_0 ∈ ℝ^q with w_0 ≥ 0,
       u_0 ∈ ℝ^n with u_0 ≥ 0.

Iterate over k = 0,1,...

  Determine L(x_k, w_k, u_k), then solve the
  above system to get (Δx, Δw, Δu)

  Select a step length α ∈ (0,1] with
```

$$w_k + \alpha\Delta w \geq 0$$
$$\text{and } u_k + \alpha\Delta u \geq 0$$

Set $(x_{k+1}, w_{k+1}, u_{k+1}) := (x_k, w_k, u_k) + \alpha(\Delta x, \Delta w, \Delta u)$

Next $k$

### 2.6.4 Realization

In the context of NLPIP, the above algorithm is applied onto the quadratic subprogram created by the SQP algorithm detailed in section 2.5.3.

The corresponding interior point equation system for NLPIP can then be presented as [BS08]

$$\phi(d, w, u) := \begin{pmatrix} \nabla_x^2 L d + \nabla f(x) + J_H(x)^T u \\ H_1(x) + J_{H_1}(x) d \\ H_2(x) + J_{H_2}(x) d + w \\ WUe - \sigma\mu e \end{pmatrix} = 0. \tag{2.85}$$

Here, $x \in \mathbb{R}^m$ is the current iteration point in SQP, while $d \in \mathbb{R}^m$ and associated slacks $w \in \mathbb{R}^q$ are the optimization variables of the interior point algorithm. As in (2.83), $H : \mathbb{R}^m \to \mathbb{R}^n$, $H_1 : \mathbb{R}^m \to \mathbb{R}^p$ and $H_2 : \mathbb{R}^m \to \mathbb{R}^q$ provide shorthand for the boundary functions, while $W$ and $U$ are diagonal matrices containing the slack variables and Lagrange multipliers. Finally, $\sigma \in [0, 1]$ is a centering parameter. This parameter is important for determining the actual step direction, and is explained after the general system solving instructions.

Applying Newton's descent on the above equation system yields a new system of the form

$$\begin{pmatrix} \nabla_x^2 L & J_H(x)^T & 0 \\ J_{H_1}(x) & 0 & 0 \\ J_{H_2}(x) & 0 & I \\ 0 & W & U \end{pmatrix} \begin{pmatrix} \Delta d \\ \Delta u \\ \Delta w \end{pmatrix} = - \begin{pmatrix} \nabla_x^2 L d + \nabla f(x) + J_H(x)^T u \\ H_1(x) + J_{H_1}(x) d \\ H_2(x) + J_{H_2}(x) d + w \\ WUe - \sigma\mu e \end{pmatrix}. \tag{2.86}$$

Simplifying this system with $\Delta w = U^{-1}(\sigma\mu e - W\Delta u) - We$ yields

$$\begin{pmatrix} \nabla_x^2 L & J_{H_1}^T(x) & J_{H_2}^T(x) \\ J_{H_1}(x) & 0 & 0 \\ J_{H_2}(x) & 0 & -WU^{-1} \end{pmatrix} \begin{pmatrix} \Delta d \\ \Delta u \end{pmatrix} = - \begin{pmatrix} \nabla_x^2 L + \nabla f(x) + J_H(x)^T u \\ H_1(x) + J_{H_1}(x) d \\ H_2(x) + J_{H_2}(x) d + \sigma\mu U^{-1} e \end{pmatrix}. \tag{2.87}$$

The above equations can now be solved for a new step direction $(\Delta d, \Delta u, \Delta w)$.

This is currently handled by a predictor-corrector method, based on the already introduced centering parameter $\sigma$. First, Newton descent is performed on a system

where $\sigma = 0$. The resulting step suggestion $(\Delta d_p, \Delta u_p, \Delta w_p)$, also called a predictor step, is evaluated in regards to the so-called complementary gap

$$\mu_p = \frac{(w_k + \Delta w_p)^T (u_k + \Delta u_p)}{n}. \tag{2.88}$$

In particular, $\mu_p$ indicates how much the $u_{k+1}$ and $w_{k+1}$ that result from the predictor step violate the requirement $W_k U_k e - \sigma\mu = W_k U_k e = 0$.

If $\mu_p > \mu$, then the associated step violates complementarity on the original problem, and only a small step length $\lambda$ can be picked.

Thus, an additional centering step $(\Delta d_{ce}, \Delta u_{ce}, \Delta w_{ce})$ is calculated by solving the equation system with $\sigma = 0.25$. Finally, a corrector step $(\Delta d_{cp}, \Delta u_{cp}, \Delta w_{cp})$ is determined by solving system (2.86) for $-(0, 0, \Delta W_p \Delta U_p e)^T$.

The entire step $(\Delta d, \Delta u, \Delta w)$ is then found by combining the predictor, centering, and corrector steps, and an associated step length $\lambda$ is found that maintains the positivity of $u$ and $w$.

### 2.6.5 Convergency

As with the sequential quadratic programming method, deriving general convergency rules for an interior point method is not easily accomplished. In particular, convergency depends on the choice of step directions and step length, on the level of approximation on involved matrices, and last but not least on the actual nonlinear optimization problem.

An example convergency analysis can be found in the rather extensive work [IG04]. In particular, a primal-dual interior point method similar to the one realized within NLPIP was discussed. Several additional requirements were made on the objective function and on the constraint functions, such as Lipschitz continuity for both the Hessian matrix of the Lagrange function and the Jacobian matrices of the boundaries. The feature set is complemented by an augmented Lagrange function and two merit functions to control convergency. The resulting system was then approached with a standard Newton descent on the KKT conditions. For this algorithmic setup, general convergency has been proven.

## 2.7 Enhancements

In the previous sections, the core nonlinear optimization progress within NLPIP has been presented in its entirety. However, as already noted in the SQP algorithm overview 2.5.3, there are many possible methods to enhance on these basic concepts. Some of these remaining finetuning details are briefly discussed in this section.

### 2.7.1 Quasi-Newton Adaptions

Up to now, the Hessian used within the algorithms has been assumed to be readily available. This setup matches the behaviour of Newton's descent method. The descent direction depends on both the gradient and the Hessian, and directly uses both of these. This kind of descent has been implemented into the actual solver in the context of this thesis. The user then is required to provide a Hessian of the Lagrange function alongside constraint gradients any time the current position in optimization changes. The Hessian provides the interior point algorithm with exact shaping information on the descent function. As such, it is intuitively understood that this yields good convergency.

However, in certain cases, recomputing the Hessian in each step of the algorithm may not be feasible. For instance, the Hessian may be too costy to compute, the Hessian may be too large to fit in memory, or there may not be an analytical form of the Hessian readily available. In such cases, a quasi-Newton method may be used to update the Hessian matrix after each step of the iteration. For instance, the BFGS update strategy that was described in subsection 2.4.3 is the default setting for NLPIP.

In this context, both sequential quadratic programming and interior point algorithms may be adapted to any quasi-Newton method by using a positive semidefinite matrix $B_k \approx \nabla_x^2 L(x_k, u_k)$ as a replacement for the original Hessian matrix of the Lagrange function. $B_{k+1}$ then is not produced by re-evaluating the Lagrange function, but by some quasi-Newton update.

As an additional enhancement, there are certain approaches that try to enforce positive semidefiniteness on the approximating matrix $B_k$, even if the non-approximated Hessian of the Lagrange function loses that property. One popular approach [MP78a] is to test for an almost violated regularization condition

$$\Delta x^T v < 0.2 \Delta x^T B_k \Delta x, \tag{2.89}$$

where $v := \nabla_x^2 L(x_{k+1}, u_{k+1})(x_{k+1} - x_k)$. In this case, the quasi-Newton update may be replaced by an alleviated update

$$B_{k+1}\Delta x = \alpha v + (1 - \alpha) B_k \Delta x \tag{2.90}$$

with

$$\alpha := 0.8 \frac{\Delta x^T B_k \Delta x}{\Delta x^T B_k \Delta x - \Delta x^T v}. \tag{2.91}$$

For the resulting matrix $B_{k+1}$, positive semidefiniteness can be proven, at the cost of losing part of the quasi-Newton convergency properties [FJ04].

Finally, in applications with very large problem dimensions, modified BFGS methods may be used that store only a subset or an approximation of the already approximative matrix $B_k$. The NLPIP solver provides such limited memory BFGS updates. In particular, only a set of $p < m$ helper vectors needs to be stored instead of the full $m \times m$ BFGS matrix. Wherever required, the approximated Hessian may then

implicitly be reconstructed from these pairs with just little costs. More details are provided in the NLPIP algorithm guide [BS08].

Note that limited memory BFGS updates put a slight bias on large-scale, dense matrices. In contrast, the NLPIP extension created alongside this thesis aimed at the introduction of exact, but sparse Hessian matrices for large-scale problems.

### 2.7.2 Line Search

Once an appropriate descent direction for the sequential quadratic problem has been found, the SQP algorithm still has to determine how far to travel along this direction. In particular, even small changes to the travelled distance may have serious implications on the optimality of the resulting point. For this reason, specialized line search methods have been developed. In particular, given certain requirements on the involved problem, line search algorithms even may enforce global convergency.

There exist several possible approaches to line search, and detailing these is outside the scope of this thesis. Instead, the basics for the the line search performed by NLPIP are presented here.

The method that is applied in NLPIP is based on the augmented Lagrange function

$$L(x,u) \;:=\; f(x) - \sum_{i\in I}\big(u_i h_i(x) + \frac{1}{2}\rho_i h_i(x)^2\big) - \frac{1}{2}\sum_{i\in J}\frac{u_i^2}{\rho_i} \tag{2.92}$$

where $I := \{i \in \{1,\dots,n\} \mid u_i + \rho_i h_i(x) \geq 0\}$, $J := \{1,\dots,n\} \setminus I$. The $\rho_i \in \mathbb{R}$, $i = 1,\dots,n$ are penalty parameters.

The idea now is to find a step length $\alpha$ that corresponds to a minimum on the augmented Lagrange function. This can, for instance, again be handled by solving a minimization problem with a Newton-like method.

In this context, the parameters $\rho_i$ are updated on each iteration of the SQP algorithm. Similar to the barrier method described above, penalty parameters introduce a higher influence on the Lagrange function for boundaries that have been violated. Thus, if several steps approach or even exceed some restriction on the optimization domain, this restriction gains more weight as its associated $\rho_i$ is increased within each parameter update. Then any later iterates are moved away from the restriction and back into feasible domain.

### 2.7.3 Stopping Conditions

Another point of vital influence on the implementation is the selection of a stopping condition. Stopping conditions indicate whether convergency has been reached or another iteration of the solver is to be performed. As with line search methods, detailed information on stopping conditions is outside the topic of this thesis. Thus, only an overview over the stopping condition relevant for the later implementation chapter is given.

In particular, NLPIP uses a set of three stopping criteria, based on the slack variables, the Lagrange function, and the boundary constraints,

$$\|\nabla_x L(x_k, u_k)\|^2 \leq \epsilon, \tag{2.93}$$

$$\frac{w^T u}{n} \leq \epsilon, \tag{2.94}$$

$$\text{and} \sum_{i=p+1}^{n} |\max\left(0, h_i(x_k)\right)| \leq \sqrt{\epsilon} \tag{2.95}$$

for some user-defined tolerance $\epsilon$.

These respectively model a minimum on the Lagrange function, thus a satisfied KKT condition, a minimum complementary slackness, and a minimum violation of boundary inequalities, thus a feasible point.

Finally, for the later tests, an additional stopping criterion has been introduced in the form of a maximum iteration cap that may be set at a global scope.

# 3 Implementation

Based on the mathematic findings of the previous chapter, this chapter describes the accompanying FORTRAN implementation. The description both includes the existing, limited memory BFGS update, and the extension for analytically provided second derivatives.

## 3.1 NLPIP

In this section, the user interface that NLPIP provides to clients as well as a rough outline of the programm flow inside of NLPIP are presented. In particular, code sections that needed to be modified for the support of second order derivatives are indicated.

All descriptions in this section rely on the NLPIP user manual [BS09] and on the associated source code comments.

Since this thesis is not meant as a complete code documentation, only vital excerpts of code are presented. For more in-depth information, the original source code should be consulted.

### 3.1.1 Clientside Interface

NLPIP provides the method NLPIPB as the clientside entry point into the actual optimization algorithm. The interface for NLPIPB is provided in listing 3.1.

On the first call to NLPIPB, the user needs to provide an initial starting point $X$, the associated value of the objective function $F$, values of all active restrictions in $G$, and derivatives of these as desired within the $DF$ and $DG$ parameters. Other parameters need to be set appropriately. For instance, the problem dimension needs to be set, and correctly sized working arrays need to be allocated.

For communication with the client during the optimization process, NLPIP uses a reverse communication scheme. In this scheme, the optimizer returns on certain events, and signals the particular event in the output parameter $IFAIL$. Such events include success in locating a minimum, terminal failure at minimization, or changes in the current iteration point. In the latter event, the user needs to provide new

```
      SUBROUTINE NLPIPB
     /     (M, ME, MW, MWMAX, N, X, F, G, DF, DG, Y, SL,
     /      XL, XU, ACTIVE, P, ACC, ACCQP,
     /      MAXFUN, MAXIT, IPRINT, IOUT,
     /      IPARAM, IFAIL, WORK, LWORK, IWORK, LIWORK)
```

Figure 3.1: Client-side interface for NLPIPB.

objective and constraint function values, as well as any associated derivatives. Then, NLPIP is called again, and optimization continues.

### 3.1.2 Internal Structure

Internally, NLPIP is structured into three layered modules, each with a specific functionality: NLPIP, QPSLV, and LINSLV.

The top-level module is the core NLPIP module, containing the main entry point into optimization. Within the NLPIP procedure, the user-provided nonlinear program is handled by the sequential quadratic algorithm as described within section 2.5.3. In particular, the original nonlinear program is approximated by a quadratic program. The quadratic program then is handled via a call to the QPSLV module. NLPIP uses the resulting, minimal value on the quadratic problem as the step direction within the original nonlinear program. The actual step length is determined by a linesearch algorithm. Finally, convergency is tested, and the call returns to the client.

The QPSLV module is called upon by NLPIP whenever a quadratic problem approximation is to be solved. QPSLV employs an interior point method for this, following the algorithm given in section 2.6.4. During this process, certain linear operations on the constraint gradients or the Hessian of the Lagrange function are transferred to the LINSLV module.

Additionally, QPSLV carries out limited memory BFGS updates on the NLPIP-internal approximation of the Lagrange Hessian. Since these updates are harmful if there is an user-provided, analytical Hessian, appropriate code segments had to be changed during the implementation of this thesis. In particular, a new parameter was introduced into the global NLPIP parameter structure to indicate that a user-provided Hessian is available. In this case, any BFGS updates in NLPIP are disabled.

The LINSLV module encapsules all linear operations on the first and second derivatives of the Lagrange function. Most importantly, this includes solving the linear equation system of the interior point method detailed in 2.6.4. Since LINSLV is not part of the NLPIP distribution but needs to be provided by the user, it is discussed on its own in the next subsection.

### 3.1.3 LINSLV module

NLPIP itself has been designed to work with arbitrary representations of the boundary condition gradients $DG$. For instance, smaller problems may benefit in performance if the gradients are provided in a dense format, while large-scale problems need to provide sparse gradients to avoid memory overflows.

This is realized by relocating all of the calculations that involve the first and possibly any second derivatives to an external module. This module must offer a single entry point, a procedure called LINSLV, with the interface presented in listing 3.3. The associated semantics are detailed in table 3.2.

| Mode | Operation |
|---|---|
| 1 | Calculate $X := I \cdot DG^T \cdot Y + X$<br>$I$ scalar multiplier.<br>$Y$ vector of size $M$.<br>$X$ vector of size $N$. |
| 2 | Calculate $X := I \cdot DG \cdot Y + X$<br>$I$ scalar multiplier.<br>$Y$ vector of size $N$.<br>$X$ vector of size $M$. |
| 3 | Set $X(1) := DG(I) \cdot Y$<br>$I$ row index in $DG$.<br>$Y$ vector of size $N$. |
| 4 | Full factorization of $C := \begin{pmatrix} H + diag(X) & DG^T \\ DG & -diag(Y) \end{pmatrix}$<br>The factorization of $C$ is stored internally for later steps (mode 5-7).<br>Assumes $X$ and $Y$ contain parts of the diagonal.<br>$X$ vector with $N$ elements.<br>$Y$ vector with $M$ elements. |
| 5 | Update matrix C<br>Equal to mode 4, but only the diagonal of $C$ changed.<br>Only numerical factorization needs to be updated. |
| 6 | Solve $Y := C \setminus Y$<br>$C$ is the internal matrix stored by mode 4.<br>$Y$ contains a list of variables to solve for,<br>and is required to be of size $N + M$ |
| 7 | Multisolve $Y(1, ..., MMAX) := C \setminus Y1, ..., MMAX$<br>$MMAX$ indicates the number of vectors to solve for.<br>$Y$ is an $N + M$ on $MMAX$ matrix. |
| 8 | Calculate $X := H \cdot Y$<br>$X$, $Y$ vectors of size $N$.<br>$H$ the Hessian from $DG$.<br>This mode only is used if BFGS updates are disabled. |

Figure 3.2: LINSLV operation modes.

Each client must implement a custom version of a LINSLV module and link that module on compilation.[1]

## 3.2 HSSSLV module

As described in the previous section, the entire functionality for calculating a descent direction has been outsourced from the main NLPIP code base into a user-provideable LINSLV module.

In this section, implementation details are given for HSSSLV, a LINSLV implementation module. In particular, HSSSLV has been designed to fullfill the requirements of the superset free material optimization application. As such, HSSSLV optionally may use an externally provided, sparse Hessian matrix for the Lagrange function instead of a limited memory BFGS approximation. This also gives a derivation of the module name HSSSLV: HeSSian-based linear SoLVer.

The entire source code of HSSSLV is available on the accompanying compact disc. Additionally, a comment-stripped version of HSSSLV can be found in appendix B.

### 3.2.1 Fortran Gradient Structure

The type-agnostic gradient input variable *DG* from the basic LINSLV interface has been replaced by the structure definition shown in listing 3.4. Since NLPIP does not read from *DG* outside of LINSLV if BFGS updates are disabled, this change is opaque to the remainder of the solver.

The gradient structure itself is rather intuitive. Both the gradients of the constraints and the Hessian matrix of the Lagrange at the current position within the optimization domain are provided in a sparse column/row/value format. In this format, only nonzero matrix elements have to be provided by the user[2]. For the symmetric Hes-

```
      SUBROUTINE LINSLV(MODE, M, MMAX, N, I, DG, X, Y)
        INTEGER MODE, M, MMAX, N, I
!       TYPE(ARBITRARY) DG
        DOUBLEPRECISION X,Y
        DIMENSION X(N + M), Y(N + M, MMAX)
```

Figure 3.3: LINSLV procedure interface.

[1] In the implementation of analytical second derivatives, this facility originally has been used to call different linear equation solvers for either BFGS updates or Newton methods. However, later on, both of these have been united into a single module.

```
MODULE GRADIENTDEF
  TYPE :: GRADIENT
    INTEGER :: LENG_GRAD
    INTEGER, POINTER :: COL_GRAD(:), ROW_GRAD(:)
    DOUBLEPRECISION, POINTER :: VAL_GRAD(:)

    LOGICAL :: USE_HESSIAN

    INTEGER :: LENG_HESSIAN
    INTEGER, POINTER :: COL_HESSIAN(:), ROW_HESSIAN(:)
    DOUBLEPRECISION, POINTER :: VAL_HESSIAN(:)

    INTEGER, POINTER :: IWA(:)
    DOUBLEPRECISION, POINTER :: DWA(:)
  END TYPE
END MODULE
```

Figure 3.4: Fortran gradient definition.

sian matrix, only elements in the upper right triangle and on the diagonal are to be specified.

Each matrix element is represented by a triplet of column index, row index, and the associated value. The triplets are split over three separate arrays for each input matrix. For the gradients, input values are provided by the $COL\_GRAD$, $ROW\_GRAD$ and $VAL\_GRAD$ pointers. For the Hessian matrix, elements are stored in the $COL\_HESSIAN$, $ROW\_HESSIAN$ and $VAL\_HESSIAN$ pointers. Additional integer values $LENG\_HESSIAN$ and $LENG\_GRAD$ indicate the respective total number of triplets.

For the gradients, matrix triplets need not be sorted by column or row indices, but rather may come in an arbitrary sequence.

However, the Hessian matrix arrays always need to hold the entire diagonal in the first triplets, and then provide nondiagonal values within consequent fields in a row-first, column-second sorting. This is a requirement induced by the superset SCPF solver. An example Hessian matrix and its representation in the required sparse column/row/value format is given in figure 3.5.

The boolean value $USE\_HESSIAN$ indicates whether the HSSSLV module works in Newton- or in quasi-Newton mode. In Newton mode, the provided Hessian is used as is. In quasi-Newton mode, the provided Hessian is ignored, and a 0 matrix is used in all calculations. Typically, $USE\_HESSIAN$ needs to reflect $IPARAM(15)$

[2]The restriction on nonzero elements is not strictly enforced. This allows for a more intuitive construction of the input matrices, while still exploiting sparse matrix fillings. For instance, in the FMO implementation, only elements that are zero for all possible positions on the objective function are pruned from the matrices. Elements that may both be zero and nonzero always are included.

$$\begin{pmatrix} 1 & 32 & 0 & 6 \\ 32 & 15 & 10 & 0 \\ 0 & 10 & 17 & 2 \\ 6 & 0 & 2 & 5 \end{pmatrix} \qquad \begin{array}{lcl} \text{LENG\_HESSIAN} & = & 8 \\ \text{COL\_HESSIAN} & = & [1, 2, 3, 4, 2, 4, 3, 4] \\ \text{ROW\_HESSIAN} & = & [1, 2, 3, 4, 1, 1, 2, 3] \\ \text{VAL\_HESSIAN} & = & [1, 15, 17, 5, 32, 6, 10, 2] \end{array}$$

Figure 3.5: Symmetric Hessian matrix in sparse column/row/value format.

of the NLPIP parameter field. Then, limited memory BFGS updates are calculated alongside all LINSLV calls, and compensate the missing Hessian matrix. Manually passing the parameter $USE\_HESSIAN$ through within $DG$ is required, since NLPIP is agnostic to the gradient structure, but LINSLV is not provided with the actual NLPIP parameter array $IPARAM(:)$.[3]

Finally, there also is a set of preallocated working arrays $IWA$ and $DWA$. These avoid dynamic memory allocation within the HSSSLV module. The size of the working arrays again depends on the maximum number of nonzero elements in the matrices.

New gradient and Hessian values are provided by client code once the NLPIP call returns. For instance, the SCPF solver uses Maple-generated code as a base for calculating complicated analytical first and second derivates. See appendix C for details on this setup.

### 3.2.2 Module Routines

The actual entry point into the HSSSLV module - as dictated by the LINSLV interface - is the $LINSLV$ subroutine. This routine does not hold any relevant code itself. It rather dispatches the incoming parameters to a series of helper routines, depending on the operation mode provided in the $MODE$ parameter. See table 3.2 for a list of supported operations.

The implementation of matrix/vector multiply-and-add and matrix/scalar multiplication operations in the sparse column/row/value format for modes 1 to 3 and mode 8 is straightforward. The only problem concerns dealing with unsorted matrix elements. This can intuitively be solved by inverting the operations in question: Instead of one pass over the output structure and direct calculation of output values, the output first is reset to zero, and results of a single pass over both input structures are accumulated in the output. See the example matrix/vector multiplication excerpt in listing 3.6 for an implementation suggestion. The original source code contains further optimizations for automatic transposition[4] and for symmetric matrices.

[3] The $USE\_HESSIAN$ flag also provides an intuitive debugging method for the Hessian matrices. If set to false while still running in Newton-mode, convergency drastically decreases, but errors in Hessian calculation can be excluded.

[4] **Attention:** The NLPIP and SCPF solvers keep their constraint gradients in transposed format relative to each other. Thus, some care is required when handling the input matrices.

```
!       Input matrix, input vectors, interpolation factors,
!       output vector
        INTEGER ROWCNT, NUMELE
        INTEGER, POINTER :: M_COLS(:), M_ROWS(:)
        DOUBLEPRECISION, POINTER :: M_VALS(:)
        DOUBLEPRECISION, POINTER :: X(:), Y(:)
        DOUBLEPRECISION ALPHA, BETA
        DOUBLEPRECISION, POINTER :: Z(:)

!       Reset output
        DO I = 1, ROWCNT
          Z(I) = 0
        ENDDO

!       Accumulate matrix * X within Z
        DO I = 1, NUMELE
          Z(M_ROWS(I)) =
                  Z(M_ROWS(I)) + M_VALS(I) * X(M_COL(I))
        ENDDO

!       Apply multiplier ALPHA and additive vector BETA * Y
        DO I = 1, ROWCNT
          Z(I) = ALPHA * Z(I) + BETA * Y(I)
        ENDDO
```

Figure 3.6: Calculation of $Z := \alpha\, M\, x + \beta\, y$ in the sparse column/row/value format.

In LINSLV modes 4 to 7, a sparse, linear equation system for a single Newton step within the interior point method needs to be solved. For this, the MUMPS library[5] [PA00] is utilized.

In particular, a MUMPS parameter structure is initialized on the first entry into the handler routine for either of the LINSLV operation modes. A single call to MUMPS is required in this case so that MUMPS may allocate all the internal fields it requires for storing the matrix factorization. MUMPS uses an internal heuristic to determine the storage space for preallocation. Since the default storage is exceeded for some of the larger SCPF problems, the associated MUMPS parameters have been finetuned to allow for a certain overallocation.

Then, depending on the actual mode, either a new matrix factorization is built from the input structure $DG$, or a solution for the current factorization and some input

[5] In a previous implementation, the PARDISO solver [OS04] had been used. Initial problems with predictability had been solved by upgrading from the solver delivered with the Intel MKL library to a more recent version. The PARDISO solver then offered performance that vastly exceeded the later MUMPS library. However, licensing rules prohibited the continued use of PARDISO in the scope of the entire project. For future reference, the PARDISO-based implementation still is available in the file `"HSSSLV_PARDISO.FOR"`

vector is determined.

Matrix factorization requires some preceding work to build a complete matrix from the separate constraint gradients, the Hessian matrix, and from the provided diagonal elements.

The solution handler of the linear equation system also implements a slight improvement over a naive approach: For semidefinite problems that are formed during finite material optimization, semidefiniteness may be lost due to optimization. As an intuitive cure, a series of small, increasing factors $\alpha > 1$ is multiplied onto the diagonal of the matrix until all eigenvalues of the matrix become positive again [KL44]. This is supported by the MUMPS feature to report negative eigenvalues of the input matrix.

Further implementation details and usage examples for the custom HSSSLV module can be found in the heading comment within the source file `"HSSSLV.FOR"` on the accompanying compact disc.

# 4 Tests

After the previous chapters provided an understanding of the implementation of NLPIP and its fundamental principles, this chapter details the tests that were used to qualify and quantify the performance gain through analytical second order derivatives of the Lagrange function.

In the course of this chapter, each testing scenario is described, and associated parameters are indicated. Then, the testing results for each test case are provided. In the last section, the results are evaluated and compared.

## 4.1 Testing Scenarios

Generally, testing was split up into two distinct types of tests:

- **Standard Tests**

  There are several collections of standard test problems for nonlinear programming methods. One of the more renowned ones is the Hock-Schittkowski test suite [KS09]. For assessing the NLPIP extension created alongside this thesis, a select few problems from this compilation have been used.

- **Free Material Optimization Tests**

  Further tests come from the set of toy problems that have been used to verify the correctness and convergency of the higher-level SCPF free material optimizer developed in terms of the Plato-N project.

**Attention:** One must note that both the SCPF solver and the encapsulated NLPIP solver are still work in progress. In particular, FMO tests reflect software state from January 2010, while standard tests work on an update from June 2010. As such, the testing setup described hereafter only serves as a means to gauge the relative performance of analytical second derivatives versus respective BFGS approximations. No statement is made on the maximum possible, absolute convergency speed and accuracy of the entire optimization process.

### 4.1.1 Standard Tests

In this subsection, all chosen standard tests from the Hock-Schittkowski suite [KS09] are listed under their respective problem number in the associated publication.

For each test problem, an objective function $f$ is defined, and any constraints on functions $h_i$ are presented. All constraints have already been adjusted to match with NLPIP's conventions of the feasible halfspace.

In addition to the actual functions, analytical second order derivatives of the Lagrange function and gradients for the constraints are provided. Since the original tests come

without analytical second order derivatives, this influenced the choice of tests: sample problems with intuitively determined derivatives have deliberately been picked.

Finally, the optimum $x^*$ and the associated value $f(x^*)$ of the objective function is given as ground truth data for verifying the results of the test runs.

- **Problem 3**
  $f(x) = x_2 + 10^{-5}(x_2 - x_1)^2$
  $h(x) = -x_2 \ \leq \ 0$
  $\nabla^2_x L = \begin{pmatrix} 2.0e-5 & -2.0e-5 \\ -2.0e-5 & 2.0e-5 \end{pmatrix}^T$
  $\nabla h = (0, -1)^T$
  $x^* = (0, 0)^T$
  $f(x^*) = 0$
- **Problem 4**
  $f(x) = \frac{1}{3}(x_1 + 1)^3 + x_2$
  $h_1(x) = 1 - x_1 \leq 0$
  $h_2(x) = -x_2 \leq 0$
  $\nabla^2_x L = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}^T$
  $\nabla h_1 = (-1, 0)^T$
  $\nabla h_2 = (0, -1)^T$
  $x^* = (1, 0)^T$
  $f(x^*) = \frac{8}{3}$
- **Problem 12**
  $f(x) = \frac{1}{2}x_1^2 + x_2^2 - x_1 x_2 - 7x_1 - 7x_2$
  $h(x) = 4x_1^2 + x_2^2 - 25 \leq 0$
  $\nabla^2_x L = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}^T$
  $\nabla h = (8x_1, 2x_2)^T$
  $x^* = (2, 3)^T$
  $f(x^*) = -30$
- **Problem 21**
  $f(x) = 0.01x_1^2 + x_2^2 - 100$
  $h(x) = -10x_1 + x_2 + 10 \leq 0$
  $2 \leq x_1 \leq 50$
  $-50 \leq x_2 \leq 50$

$\nabla_x^2 L = \begin{pmatrix} 0.02 & 0 \\ 0 & 2 \end{pmatrix}^T$
$\nabla h = (-10, 1)^T$
$x^* = (2, 0)^T$
$f(x^*) = -99.96$

- **Problem 22**
  $f(x) = (x_1 - 2)^2 + (x_2 - 1)^2$
  $h_1(x) = x_1 + x_2 - 2 \leq 0$
  $h_2(x) = x_1^2 - x_2 \leq 0$
  $\nabla_x^2 L = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}^T$
  $\nabla h_1 = (1, 1)^T$
  $\nabla h_2 = (2x_1, -1)^T$
  $x^* = (1, 1)^T$
  $f(x^*) = 1$

Testing was carried out by a small wrapper application around the NLPIP core optimizer. The wrapper application simply reacts to NLPIP reverse communication attempts and provides new analytical objective and boundary function values, as well as any associated derivatives. The corresponding calculations have been hardcoded into the application for simplicity.

Standard tests included but two runs of the NLPIP solver on each input problem, one for analytical second derivatives and one for BFGS updates. Optimization was performed until an optimum had been found with a set accuracy of $\epsilon_{SQP} = 10^{-12}$ in SQP, and a set accuracy of $\epsilon_{IPM} = 10^{-16}$ in IPM. Additionally, a maximum of 200 SQP iterations were allowed to enforce termination. The total number of internal iterations within the interior point method up to either of these events was counted, and the corresponding running time was measured. Finally, the resulting optimum point and the corresponding value on the objective function were recorded to verify correctness against ground truth data in the later interpretation.

### 4.1.2 Free Material Optimization Tests

Each of the finite material optimization tests calculates minimum material distribution on a certain base topology so that a maximum topology displacement is not exceeded under one or more given loads. For this, a given coarse input topology is tesselated into finer-grained finite elements and associated boundary nodes. The number of finite elements then determines the number of optimization variables and boundary constraints. Both the original topology, as well as the tesselation vary over the individual tests.

The actual calculations for the free material optimization tests have been performed using a variety of collaborating software. In particular, MOPED [MK99] has been

used to preprocess the input topologies. Then, the sequential convex programming solver SCPF [SE09] was employed to put together an iterative series of convex, nonlinear programs. These, in turn, have been solved by NLPIP. Thus, there have been several cycles of the SQP algorithm, each producing a new sequence of quadratic problems.

As with standard tests, optimization in the FMO scenarios was performed until a set accuracy for the resulting optimum point has been reached on each of the three solver levels. In particular, the outer SCPF solver required an accuracy of $\epsilon_{SCPF} = 10^{-5}$ for the entire FMO problem, while the encapsulated NLPIP solver used accuracies of $\epsilon_{SQP} = 10^{-8}$ and $\epsilon_{IPM} = 10^{-9}$. Additionally, a maximum iteration cap was introduced at each level of optimization to improve stability. A maximum 25 interior point iterations were allowed to accurately solve any quadratic problem with an interior point approach. Sequential quadratic programming within NLPIP also was restricted to a maximum of 25 quadratic approximations. Finally, after 25 iterations of SCPF, optimization finished.

Then, the total number of innermost interior point iterations required up to this maximum have been counted for both the BFGS and the analytical second derivatives approach. The corresponding time has been measured, and the respective objective function values at the final location have been noted. After each testing run, MOPED was used again to postprocess the resulting material distributions into graphical representations.

Note that testing results for the FMO scenarios strictly speaking allow no direct comparison between the limited memory BFGS update and the exact analytical approach. In particular, only the first quadratic subproblem is the same for both approaches. Since solutions for BFGS and analytical second derivatives do not exactly match for the first problem, later iterations of the SCPF algorithm use different starting points for convex approximation, and thus provide different convex problems to either of the two methods. As such, the testing results rather are expected to indicate a rough trend on the more optimal approach for free material optimization.

In addition to the scenarios used for analytical-to-BFGS comparison, a final test problem was solved using the later Plato-N calculation pipeline. In particular, the Plato-N optimization framework has been used both for pre- and post-processing the material topology. Within this test, only analytical second derivatives have been examined. However, instead of introducing an artificial iteration cap, complete convergence was allowed. This scenario acts as a full example for the later application of the implementation that was created alongside this thesis.

All parameters for the FMO tests are summarized in the following table:

| Name | Nodes | Elems | Loads | Vars | Constrs |
|---|---|---|---|---|---|
| Cantchen | 50 | 36 | 1 | 217 | 38 |
| Canti | 400 | 351 | 1 | 2107 | 353 |
| Canti3 | 2500 | 2376 | 1 | 14257 | 2378 |
| SmallEx1 | 450 | 406 | 2 | 2437 | 409 |
| SmallEx4 | 1250 | 1176 | 2 | 7057 | 1179 |
| Plato-N | - | - | 1 | 45000 | 22502 |

## 4.2 Results

This section presents the results of the tests described in the previous section. For each testing scenario, the NLPIP protocol file is provided, once for approximated and once for analytical second order derivatives. Finally, figure 4.2.3 provides an overview over the most important outcomes for each test.

### 4.2.1 Standard Tests

The following result tables have been obtained from the standard tests.

For each testing scenario, the iterations of the interior point method within the first quadratic program are presented. The column `"IT"` provides the iteration counter. Within `"KKT"`, the derivative of the Lagrange function at the current iterate is given. `"FS"` gives the current feasibility. `"MU"` and `"SIG"` indicate the respective parameters $\mu$ and $\sigma$ from the interior point algorithm.

After the per-iteration information, global measurements are given: the total number of inner or outer iterations, the resulting optimum position, and the final value of the objective function.

For compactness reasons, only an excerpt of each protocol is presented. The complete log files are available on the accompanying compact disc.

**Problem 3 - BFGS approximation**

| IT | KKT | FS | MU | SIG |
|---|---|---|---|---|
| 1 | 0.000000D+00 | 0.000000D+00 | 0.291572D+00 | 0.564039D-02 |
| 2 | 0.130691D-15 | 0.156163D-15 | 0.514613D-01 | 0.147522D-01 |
| 3 | 0.890249D-16 | 0.000000D+00 | 0.132250D-01 | 0.136851D-01 |
| 4 | 0.435360D-16 | 0.156163D-15 | 0.331721D-02 | 0.137684D-01 |
| 5 | 0.618076D-16 | 0.142109D-15 | 0.834216D-03 | 0.136932D-01 |
| 6 | 0.517585D-16 | 0.156163D-15 | 0.209702D-03 | 0.137287D-01 |
| 7 | 0.262086D-16 | 0.139322D-15 | 0.527318D-04 | 0.136760D-01 |
| 8 | 0.372838D-16 | 0.256051D-15 | 0.132559D-04 | 0.137155D-01 |
| 9 | 0.364366D-16 | 0.121431D-16 | 0.333300D-05 | 0.136528D-01 |
| 10 | 0.208886D-17 | 0.290566D-16 | 0.837446D-06 | 0.136376D-01 |
| 11 | 0.395448D-16 | 0.278644D-15 | 0.210139D-06 | 0.133538D-01 |
| 12 | 0.333405D-16 | 0.491144D-16 | 0.523770D-07 | 0.124975D-01 |
| 13 | 0.595271D-16 | 0.142109D-15 | 0.127320D-07 | 0.948287D-02 |
| 14 | 0.104373D-15 | 0.525296D-16 | 0.280461D-08 | 0.346941D-02 |
| 15 | 0.649059D-18 | 0.139322D-15 | 0.432720D-09 | 0.133121D-03 |
| 16 | 0.182430D-19 | 0.139322D-15 | 0.221317D-10 | 0.571592D-07 |
| 17 | 0.111022D-15 | 0.142109D-15 | 0.847717D-13 | 0.117964D-04 |
| 18 | 0.202211D-20 | 0.142109D-15 | 0.503815D-17 | 0.198486D+00 |
| 19 | 0.202016D-20 | 0.169407D-16 | 0.100000D-17 | 0.000000D+00 |

| | |
|---|---|
| Total inner point iterations: | 85 |
| Total SQP iterations: | 11 |
| Resulting optimum: | -3.376673771736008E-014, 1.000000000000000E-018 |
| Final objective function: | 1.000000000000011E-018 |

**Problem 3 - Analytical second derivatives**

| IT | KKT | FS | MU | SIG |
|---|---|---|---|---|
| 1 | 0.000000D+00 | 0.000000D+00 | 0.291572D+00 | 0.148071D-03 |
| 2 | 0.311185D-16 | 0.139322D-15 | 0.153585D-01 | 0.278051D-03 |
| 3 | 0.779580D-16 | 0.156163D-15 | 0.895756D-03 | 0.189905D-05 |
| 4 | 0.285717D-16 | 0.142109D-15 | 0.105531D-04 | 0.134927D-06 |
| 5 | 0.474504D-17 | 0.278644D-15 | 0.531271D-07 | 0.125049D-06 |
| 6 | 0.172268D-16 | 0.284217D-15 | 0.265649D-09 | 0.376437D-08 |
| 7 | 0.144799D-16 | 0.988895D-17 | 0.132825D-12 | 0.752869D-05 |
| 8 | 0.292945D-20 | 0.173033D-16 | 0.725742D-17 | 0.000000D+00 |

| | |
|---|---|
| Total inner point iterations: | 8 |
| Total SQP iterations: | 2 |
| Resulting optimum: | 5.329070518200751E-015, 0.000000000000000E+000 |
| Final objective function: | 2.839899258795643E-034 |

**Problem 4 - BFGS approximation**

| IT | KKT | FS | MU | SIG |
|---|---|---|---|---|
| 1 | 0.141330D-16 | 0.875000D+00 | 0.193614D+02 | 0.000000D+00 |
| 2 | 0.123292D-15 | 0.273251D+00 | 0.610401D+01 | 0.000000D+00 |
| 3 | 0.208391D-16 | 0.152284D-02 | 0.121543D+00 | 0.000000D+00 |
| 4 | 0.229363D-15 | 0.902650D-05 | 0.296616D-02 | 0.000000D+00 |
| 5 | 0.105046D-15 | 0.460099D-07 | 0.354280D-04 | 0.000000D+00 |
| 6 | 0.726662D-16 | 0.230046D-09 | 0.183010D-06 | 0.000000D+00 |
| 7 | 0.553198D-16 | 0.115030D-12 | 0.916673D-10 | 0.000000D+00 |
| 8 | 0.149009D-15 | 0.579818D-15 | 0.458346D-12 | 0.000000D+00 |
| 9 | 0.139583D-15 | 0.140528D-15 | 0.229959D-15 | 0.000000D+00 |
| 10 | 0.628395D-22 | 0.114286D-17 | 0.100000D-17 | 0.000000D+00 |

| | |
|---|---|
| Total inner point iterations: | 40 |
| Total SQP iterations: | 4 |
| Resulting optimum: | 1.00000000000000, 1.000000000000016E-018 |
| Final objective function: | 2.66666666666667 |

**Problem 4 - Analytical second derivatives**

| IT | KKT | FS | MU | SIG |
|---|---|---|---|---|
| 1 | 0.141330D-16 | 0.875000D+00 | 0.193614D+02 | 0.000000D+00 |
| 2 | 0.123070D-15 | 0.273251D+00 | 0.610313D+01 | 0.000000D+00 |
| 3 | 0.207853D-16 | 0.152284D-02 | 0.121015D+00 | 0.000000D+00 |
| 4 | 0.362250D-16 | 0.912685D-05 | 0.146040D-02 | 0.000000D+00 |
| 5 | 0.486367D-16 | 0.458676D-07 | 0.845552D-05 | 0.000000D+00 |
| 6 | 0.129640D-15 | 0.229327D-09 | 0.423213D-07 | 0.000000D+00 |
| 7 | 0.985414D-16 | 0.114661D-12 | 0.211618D-10 | 0.000000D+00 |
| 8 | 0.127131D-15 | 0.577635D-15 | 0.105811D-12 | 0.000000D+00 |
| 9 | 0.631957D-19 | 0.616380D-17 | 0.536486D-16 | 0.000000D+00 |
| 10 | 0.618096D-22 | 0.100000D-17 | 0.100000D-17 | 0.000000D+00 |

| | |
|---|---|
| Total inner point iterations: | 20 |
| Total SQP iterations: | 3 |
| Resulting optimum: | 1.00000000000000, 0.000000000000000E+000 |
| Final objective function: | 2.66666666666667 |

**Problem 12 - BFGS approximation**

| IT | KKT | FS | MU | SIG |
|---|---|---|---|---|
| 1 | 0.000000D+00 | 0.000000D+00 | 0.280100D+03 | 0.000000D+00 |
| 2 | 0.572232D-15 | 0.000000D+00 | 0.172188D+02 | 0.000000D+00 |
| 3 | 0.349999D-15 | 0.355271D-14 | 0.161726D+00 | 0.000000D+00 |
| 4 | 0.352158D-15 | 0.355271D-14 | 0.808923D-03 | 0.000000D+00 |
| 5 | 0.563074D-16 | 0.000000D+00 | 0.404616D-06 | 0.000000D+00 |
| 6 | 0.180839D-15 | 0.140702D-15 | 0.202308D-10 | 0.000000D+00 |
| 7 | 0.271683D-15 | 0.355271D-14 | 0.101160D-13 | 0.000000D+00 |
| 8 | 0.424005D-20 | 0.000000D+00 | 0.110548D-17 | 0.000000D+00 |

| | |
|---|---|
| Total inner point iterations: | 35343 |
| Total SQP iterations: | 101 |
| Resulting optimum: | 2.16923265436954, 3.14831383596689 |
| Final objective function: | -31.7875854474057 |

**Problem not solved. Maximum iteration count exceeded.**

**Problem 12 - Analytical second derivatives**

| IT | KKT | FS | MU | SIG |
|---|---|---|---|---|
| 1 | 0.000000D+00 | 0.000000D+00 | 0.280100D+03 | 0.000000D+00 |
| 2 | 0.142398D-14 | 0.000000D+00 | 0.351868D+02 | 0.000000D+00 |
| 3 | 0.501356D-15 | 0.355271D-14 | 0.109471D+01 | 0.000000D+00 |
| 4 | 0.852857D-15 | 0.355271D-14 | 0.245626D-02 | 0.000000D+00 |
| 5 | 0.850562D-15 | 0.000000D+00 | 0.123843D-05 | 0.000000D+00 |
| 6 | 0.342960D-15 | 0.355271D-14 | 0.619228D-08 | 0.000000D+00 |
| 7 | 0.996463D-16 | 0.355271D-14 | 0.309614D-11 | 0.000000D+00 |
| 8 | 0.745567D-17 | 0.140702D-15 | 0.155544D-15 | 0.000000D+00 |
| 9 | 0.439377D-20 | 0.140702D-15 | 0.100000D-17 | 0.000000D+00 |

| | |
|---|---|
| Total inner point iterations: | 1550 |
| Total SQP iterations: | 29 |
| Resulting optimum: | 1.99999993256129, 3.00000017983661 |
| Final objective function: | -30.0000000000001 |

**Problem 21 - BFGS approximation**

| IT | KKT | FS | MU | SIG |
|---|---|---|---|---|
| 1 | 0.000000D+00 | 0.200000D+02 | 0.122845D+03 | 0.000000D+00 |
| 2 | 0.459996D-05 | 0.163902D+02 | 0.100802D+03 | 0.000000D+00 |
| 3 | 0.276663D-13 | 0.355271D-14 | 0.455751D+01 | 0.000000D+00 |
| 4 | 0.307519D-15 | 0.142109D-15 | 0.699442D-02 | 0.000000D+00 |
| 5 | 0.130270D-15 | 0.355271D-14 | 0.557178D-04 | 0.000000D+00 |
| 6 | 0.172020D-15 | 0.355271D-14 | 0.279030D-05 | 0.000000D+00 |
| 7 | 0.167919D-15 | 0.355271D-14 | 0.139546D-07 | 0.000000D+00 |
| 8 | 0.170132D-15 | 0.355271D-14 | 0.698825D-09 | 0.000000D+00 |
| 9 | 0.278658D-15 | 0.177636D-14 | 0.349418D-11 | 0.000000D+00 |
| 10 | 0.196242D-15 | 0.177636D-14 | 0.174775D-14 | 0.000000D+00 |
| 11 | 0.523451D-20 | 0.838926D-19 | 0.100000D-17 | 0.000000D+00 |

| | |
|---|---|
| Total inner point iterations: | 2579 |
| Total SQP iterations: | 100 |
| Resulting optimum: | 2.00000000000000, -5.373651061452947E-008 |
| Final objective function: | -99.9600000000000 |

**Problem 21 - Analytical second derivatives**

| IT | KKT | FS | MU | SIG |
|---|---|---|---|---|
| 1 | 0.000000D+00 | 0.200000D+02 | 0.122845D+03 | 0.000000D+00 |
| 2 | 0.107883D-05 | 0.173912D+02 | 0.106804D+03 | 0.000000D+00 |
| 3 | 0.211482D-12 | 0.888178D-14 | 0.500014D+01 | 0.000000D+00 |
| 4 | 0.253832D-15 | 0.177636D-14 | 0.455342D-02 | 0.000000D+00 |
| 5 | 0.145322D-15 | 0.177636D-14 | 0.155428D-03 | 0.000000D+00 |
| 6 | 0.151178D-15 | 0.142109D-15 | 0.778764D-05 | 0.000000D+00 |
| 7 | 0.419136D-16 | 0.177636D-14 | 0.397788D-07 | 0.000000D+00 |
| 8 | 0.129699D-15 | 0.429888D-17 | 0.199252D-08 | 0.000000D+00 |
| 9 | 0.137197D-15 | 0.136643D-15 | 0.996280D-11 | 0.000000D+00 |
| 10 | 0.191629D-16 | 0.177636D-14 | 0.498216D-14 | 0.000000D+00 |
| 11 | 0.206406D-19 | 0.177636D-14 | 0.100734D-17 | 0.000000D+00 |

| | |
|---|---|
| Total inner point iterations: | 11 |
| Total SQP iterations: | 3 |
| Resulting optimum: | 2.00000000000000, 0.000000000000000E+000 |
| Final objective function: | -99.9600000000000 |

**Problem 22 - BFGS approximation**

| IT | KKT | FS | MU | SIG |
|---|---|---|---|---|
| 1 | 0.198746D-16 | 0.300000D+01 | 0.854014D+02 | 0.000000D+00 |
| 2 | 0.483966D-15 | 0.319341D-01 | 0.171594D+01 | 0.000000D+00 |
| 3 | 0.617706D-16 | 0.370799D-03 | 0.201274D-01 | 0.000000D+00 |
| 4 | 0.304869D-16 | 0.247895D-05 | 0.134636D-03 | 0.000000D+00 |
| 5 | 0.313841D-16 | 0.123946D-07 | 0.673191D-06 | 0.000000D+00 |
| 6 | 0.630925D-16 | 0.619826D-11 | 0.336620D-09 | 0.000000D+00 |
| 7 | 0.258680D-16 | 0.309764D-13 | 0.168314D-11 | 0.000000D+00 |
| 8 | 0.804454D-16 | 0.333067D-15 | 0.842400D-15 | 0.000000D+00 |
| 9 | 0.555113D-16 | 0.111022D-15 | 0.100000D-17 | 0.000000D+00 |

| | |
|---|---|
| Total inner point iterations: | 46 |
| Total SQP iterations: | 4 |
| Resulting optimum: | 0.999999989341166, 1.00000001065883 |
| Final objective function: | 1.00000002131767 |

**Problem 22 - Analytical second derivatives**

| IT | KKT | FS | MU | SIG |
|---|---|---|---|---|
| 1 | 0.198746D-16 | 0.300000D+01 | 0.854014D+02 | 0.000000D+00 |
| 2 | 0.232343D-15 | 0.845250D+00 | 0.244690D+02 | 0.000000D+00 |
| 3 | 0.181348D-15 | 0.341541D-03 | 0.199791D+00 | 0.000000D+00 |
| 4 | 0.119102D-15 | 0.353870D-06 | 0.126560D-02 | 0.000000D+00 |
| 5 | 0.364065D-16 | 0.232096D-08 | 0.196491D-04 | 0.000000D+00 |
| 6 | 0.147545D-15 | 0.115875D-09 | 0.992005D-06 | 0.000000D+00 |
| 7 | 0.415050D-16 | 0.579981D-12 | 0.498752D-08 | 0.000000D+00 |
| 8 | 0.884717D-16 | 0.297540D-13 | 0.249847D-09 | 0.000000D+00 |
| 9 | 0.302896D-16 | 0.137969D-15 | 0.124926D-11 | 0.000000D+00 |
| 10 | 0.134003D-18 | 0.222045D-15 | 0.625423D-15 | 0.000000D+00 |
| 11 | 0.740149D-16 | 0.444089D-15 | 0.100000D-17 | 0.000000D+00 |

| | |
|---|---|
| Total inner point iterations: | 22 |
| Total SQP iterations: | 4 |
| Resulting optimum: | 1.00000000000000, 1.00000000000000 |
| Final objective function: | 1.00000000000000 |

### 4.2.2 Free Material Optimization Tests

The following pages present the results obtained from the FMO tests.

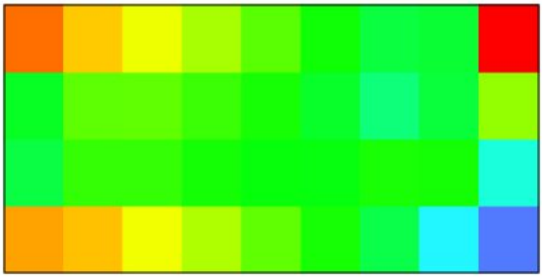
For each testing sample, the resulting material distribution is displayed. Note that due to the maximum SCPF iteration cap, full convergency was not reached. As such, the images do not represent competitive FMO results. The only intention here is to provide a more figurative comparison between limited memory BFGS updates and exact second derivatives.

The actual optimization progress for each FMO test is shown in an associated result table. In particular, the result tables list the first ten iterations of the outer SCPF solver. The associated SCPF iteration counter is given in the column `"IT"`. `"QIT"` indicates the total number of NLPIP iterations. The feasibility of the current iterate is provided as `"FEAS"`. The `"OBJ"` column represents the value of the objective function at the end of the current iteration. The step length within the optimization domain is given by `"NDX"`. The derivative of the Lagrange function is represented by `"NLX"`.

After the rough iteration outline, global parameters such as timing information and the resulting objective function value for the complete optimization run are listed.

Compared to all original protocols, irrelevant data columns have been removed. The original, full logs are available on the accompanying compact disc.

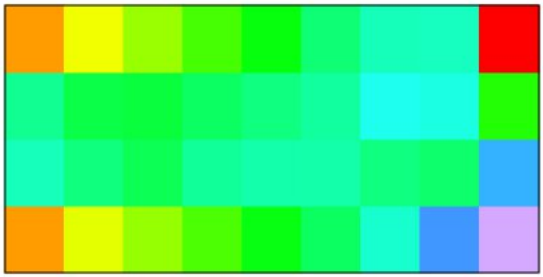
**Cantchen - BFGS approximation**



| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.7050D+00 | 0.64D+01 | 0.78D+00 |
| 2 | 25 | 0.00D+00 | 0.4058D+00 | 0.10D+02 | 0.70D+00 |
| 3 | 25 | 0.00D+00 | 0.2283D+00 | 0.12D+02 | 0.46D+00 |
| 4 | 25 | 0.00D+00 | 0.1695D+00 | 0.87D+01 | 0.16D+00 |
| 5 | 25 | 0.00D+00 | 0.4206D+00 | 0.26D+01 | 0.81D+01 |
| 6 | 25 | 0.00D+00 | 0.2019D+00 | 0.47D+01 | 0.59D+00 |
| 7 | 25 | 0.00D+00 | 0.1272D+00 | 0.78D+01 | 0.26D+00 |
| 8 | 25 | 0.00D+00 | 0.1038D+00 | 0.88D+01 | 0.78D-01 |
| 9 | 25 | 0.00D+00 | 0.9109D-01 | 0.74D+01 | 0.41D-01 |
| 10 | 25 | 0.00D+00 | 0.7946D-01 | 0.10D+02 | 0.30D-01 |

| | |
|---|---|
| Total running time: | 54sec |
| Total inner point iterations: | 12257 |
| Final objective function: | 0.3392D-01 |

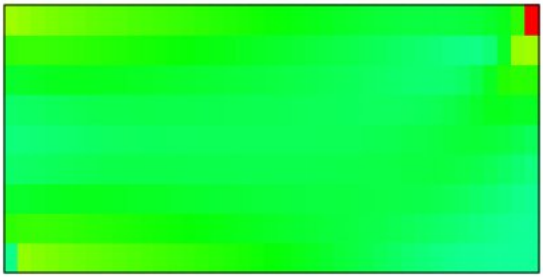
**Cantchen - Analytical second derivatives**



| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.7071D+00 | 0.63D+01 | 0.77D+00 |
| 2 | 25 | 0.00D+00 | 0.3968D+00 | 0.11D+02 | 0.72D+00 |
| 3 | 20 | 0.00D+00 | 0.2108D+00 | 0.14D+02 | 0.48D+00 |
| 4 | 18 | 0.00D+00 | 0.1501D+00 | 0.14D+02 | 0.17D+00 |
| 5 | 17 | 0.00D+00 | 0.1153D+00 | 0.13D+02 | 0.87D-01 |
| 6 | 21 | 0.00D+00 | 0.9247D-01 | 0.14D+02 | 0.51D-01 |
| 7 | 25 | 0.00D+00 | 0.7614D-01 | 0.14D+02 | 0.32D-01 |
| 8 | 25 | 0.00D+00 | 0.6384D-01 | 0.14D+02 | 0.21D-01 |
| 9 | 25 | 0.00D+00 | 0.5495D-01 | 0.14D+02 | 0.13D-01 |
| 10 | 22 | 0.00D+00 | 0.4766D-01 | 0.14D+02 | 0.95D-02 |

| | |
|---|---|
| Total running time: | 11sec |
| Total inner point iterations: | 2672 |
| Final objective function: | 0.2842D-01 |

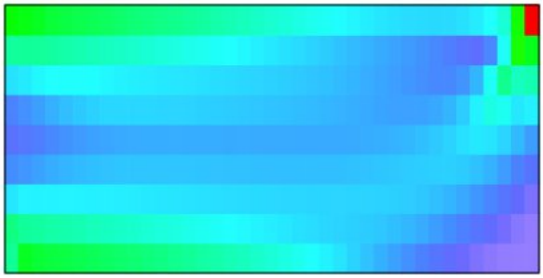
**Canti - BFGS approximation**



| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.7435D+00 | 0.28D+02 | 0.68D-01 |
| 2 | 25 | 0.00D+00 | 0.4398D+00 | 0.25D+02 | 0.71D+00 |
| 3 | 25 | 0.00D+00 | 0.2640D+00 | 0.25D+02 | 0.46D+00 |
| 4 | 25 | 0.00D+00 | 0.1859D+00 | 0.26D+02 | 0.21D+00 |
| 5 | 25 | 0.00D+00 | 0.1434D+00 | 0.24D+02 | 0.11D+00 |
| 6 | 25 | 0.00D+00 | 0.1228D+00 | 0.34D+02 | 0.46D-01 |
| 7 | 1 | 0.00D+00 | 0.1228D+00 | 0.79D-05 | 0.22D+04 |
| 8 | 25 | 0.00D+00 | 0.1101D+00 | 0.14D+02 | 0.44D-01 |
| 9 | 25 | 0.00D+00 | 0.1041D+00 | 0.82D+01 | 0.24D-01 |
| 10 | 25 | 0.00D+00 | 0.9961D-01 | 0.98D+01 | 0.45D-01 |

| | |
|---|---|
| Total running time: | 10min 19sec |
| Total inner point iterations: | 10405 |
| Final objective function: | 0.6523D-01 |

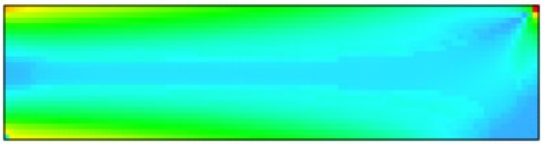
**Canti - Analytical second derivatives**



| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.7037D+00 | 0.19D+02 | 0.78D+00 |
| 2 | 25 | 0.00D+00 | 0.4297D+00 | 0.23D+02 | 0.68D+00 |
| 3 | 25 | 0.00D+00 | 0.2775D+00 | 0.23D+02 | 0.41D+00 |
| 4 | 1 | 0.00D+00 | 0.2775D+00 | 0.44D-05 | 0.35D+04 |
| 5 | 25 | 0.00D+00 | 0.2346D+00 | 0.10D+02 | 0.16D+00 |
| 6 | 25 | 0.00D+00 | 0.2040D+00 | 0.13D+02 | 0.12D+00 |
| 7 | 25 | 0.00D+00 | 0.1772D+00 | 0.13D+02 | 0.89D-01 |
| 8 | 25 | 0.00D+00 | 0.1537D+00 | 0.17D+02 | 0.69D-01 |
| 9 | 25 | 0.00D+00 | 0.1340D+00 | 0.16D+02 | 0.51D-01 |
| 10 | 24 | 0.00D+00 | 0.1185D+00 | 0.17D+02 | 0.35D-01 |

| | |
|---|---|
| Total running time: | 3min 13sec |
| Total inner point iterations: | 3567 |
| Final objective function: | 0.4100D-01 |

**Canti3 - BFGS approximation**



| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.7658D+00 | 0.38D+02 | 0.66D-01 |
| 2 | 25 | 0.00D+00 | 0.4901D+00 | 0.38D+02 | 0.68D+00 |
| 3 | 25 | 0.00D+00 | 0.3241D+00 | 0.38D+02 | 0.44D+00 |
| 4 | 25 | 0.00D+00 | 0.2456D+00 | 0.55D+02 | 0.23D+00 |
| 5 | 25 | 0.00D+00 | 0.1999D+00 | 0.30D+02 | 0.12D+00 |
| 6 | 25 | 0.00D+00 | 0.1691D+00 | 0.30D+02 | 0.68D-01 |
| 7 | 25 | 0.00D+00 | 0.1460D+00 | 0.39D+02 | 0.45D-01 |
| 8 | 1 | 0.00D+00 | 0.1460D+00 | 0.19D-05 | 0.82D+03 |
| 9 | 25 | 0.00D+00 | 0.1356D+00 | 0.29D+02 | 0.35D-01 |
| 10 | 25 | 0.00D+00 | 0.1289D+00 | 0.14D+02 | 0.27D-01 |

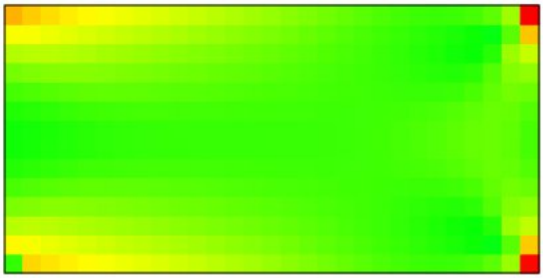
| | |
|---|---|
| Total running time: | 49min 59sec |
| Total inner point iterations: | 13757 |
| Final objective function: | 0.5515D-01 |

**Canti3 - Analytical second derivatives**

| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.7382D+00 | 0.40D+02 | 0.74D+00 |
| 2 | 25 | 0.00D+00 | 0.4772D+00 | 0.52D+02 | 0.66D+00 |
| 3 | 25 | 0.00D+00 | 0.3243D+00 | 0.36D+02 | 0.41D+00 |
| 4 | 1 | 0.00D+00 | 0.3243D+00 | 0.14D-05 | 0.34D+04 |
| 5 | 25 | 0.00D+00 | 0.2650D+00 | 0.22D+02 | 0.21D+00 |
| 6 | 15 | 0.00D+00 | 0.2443D+00 | 0.21D+02 | 0.50D+00 |
| 7 | 10 | 0.00D+00 | 0.2300D+00 | 0.25D+02 | 0.15D+01 |
| 8 | 8 | 0.00D+00 | 0.2250D+00 | 0.29D+02 | 0.34D+01 |
| 9 | 12 | 0.00D+00 | 0.2106D+00 | 0.33D+02 | 0.26D+01 |
| 10 | 10 | 0.00D+00 | 0.1855D+00 | 0.38D+02 | 0.19D+00 |

| | |
|---|---|
| Total running time: | 15min 2sec |
| Total inner point iterations: | 2737 |
| Final objective function: | 0.5898D-01 |

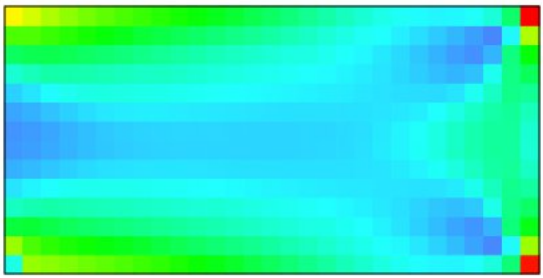
**SmallEx1 - BFGS approximation**



| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.7107D+00 | 0.20D+02 | 0.77D+00 |
| 2 | 25 | 0.00D+00 | 0.4415D+00 | 0.25D+02 | 0.67D+00 |
| 3 | 25 | 0.00D+00 | 0.3765D+00 | 0.83D+01 | 0.46D+00 |
| 4 | 25 | 0.00D+00 | 0.2900D+00 | 0.12D+02 | 0.30D+00 |
| 5 | 25 | 0.00D+00 | 0.2540D+00 | 0.11D+02 | 0.22D+00 |
| 6 | 25 | 0.00D+00 | 0.2210D+00 | 0.14D+02 | 0.97D-01 |
| 7 | 25 | 0.00D+00 | 0.3399D+00 | 0.41D+01 | 0.87D+00 |
| 8 | 25 | 0.00D+00 | 0.2214D+00 | 0.68D+01 | 0.38D+00 |
| 9 | 25 | 0.00D+00 | 0.1909D+00 | 0.90D+01 | 0.11D+00 |
| 10 | 25 | 0.00D+00 | 0.1689D+00 | 0.14D+02 | 0.74D-01 |

| | |
|---|---|
| Total running time: | 17min 41sec |
| Total inner point iterations: | 13531 |
| Final objective function: | 0.7407D-01 |

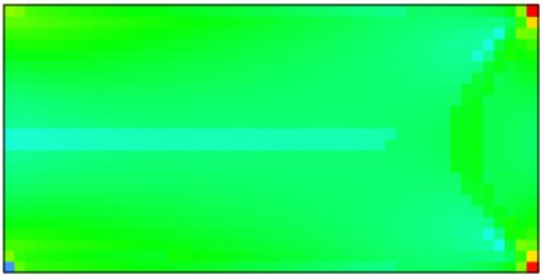
**SmallEx1 - Analytical second derivatives**



| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.7112D+00 | 0.20D+02 | 0.77D+00 |
| 2 | 25 | 0.00D+00 | 0.4428D+00 | 0.25D+02 | 0.67D+00 |
| 3 | 25 | 0.00D+00 | 0.3052D+00 | 0.24D+02 | 0.38D+00 |
| 4 | 25 | 0.00D+00 | 0.2471D+00 | 0.25D+02 | 0.19D+00 |
| 5 | 25 | 0.00D+00 | 0.1781D+00 | 0.26D+02 | 0.17D+00 |
| 6 | 21 | 0.00D+00 | 0.1404D+00 | 0.26D+02 | 0.83D-01 |
| 7 | 20 | 0.00D+00 | 0.1154D+00 | 0.26D+02 | 0.49D-01 |
| 8 | 25 | 0.00D+00 | 0.9830D-01 | 0.25D+02 | 0.29D-01 |
| 9 | 25 | 0.00D+00 | 0.8575D-01 | 0.26D+02 | 0.19D-01 |
| 10 | 25 | 0.00D+00 | 0.7578D-01 | 0.25D+02 | 0.13D-01 |

| | |
|---|---|
| Total running time: | 3min 23sec |
| Total inner point iterations: | 3112 |
| Final objective function: | 0.3469D-01 |

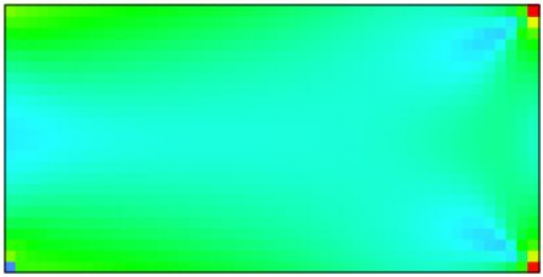
**SmallEx4 - BFGS approximation**



| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.6986D+00 | 0.32D+02 | 0.78D+00 |
| 2 | 25 | 0.00D+00 | 0.4387D+00 | 0.39D+02 | 0.65D+00 |
| 3 | 1 | 0.00D+00 | 0.4387D+00 | 0.66D-05 | 0.30D+04 |
| 4 | 25 | 0.00D+00 | 0.3245D+00 | 0.20D+02 | 0.37D+00 |
| 5 | 25 | 0.00D+00 | 0.2834D+00 | 0.86D+01 | 0.26D+00 |
| 6 | 25 | 0.00D+00 | 0.2483D+00 | 0.17D+02 | 0.13D+00 |
| 7 | 25 | 0.00D+00 | 0.2172D+00 | 0.18D+02 | 0.10D+00 |
| 8 | 25 | 0.00D+00 | 0.2059D+00 | 0.68D+01 | 0.67D-01 |
| 9 | 25 | 0.00D+00 | 0.1981D+00 | 0.14D+02 | 0.20D-01 |
| 10 | 25 | 0.00D+00 | 0.1867D+00 | 0.16D+02 | 0.39D-01 |

| | |
|---|---|
| Total running time: | 54min 24sec |
| Total inner point iterations: | 13671 |
| Final objective function: | 0.5331D-01 |

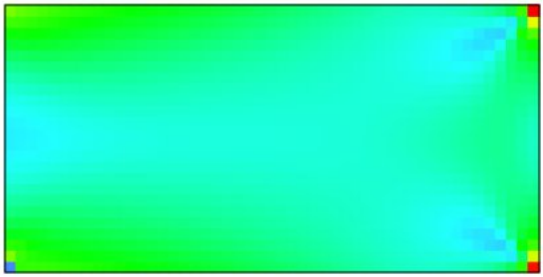
**SmallEx4 - Analytical second derivatives**



| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.7317D+00 | 0.32D+02 | 0.75D+00 |
| 2 | 25 | 0.00D+00 | 0.4746D+00 | 0.39D+02 | 0.65D+00 |
| 3 | 23 | 0.00D+00 | 0.3198D+00 | 0.38D+02 | 0.42D+00 |
| 4 | 25 | 0.00D+00 | 0.2475D+00 | 0.43D+02 | 0.20D+00 |
| 5 | 1 | 0.00D+00 | 0.2475D+00 | 0.90D-06 | 0.17D+05 |
| 6 | 10 | 0.00D+00 | 0.2955D+00 | 0.71D+01 | 0.10D+05 |
| 7 | 1 | 0.00D+00 | 0.2955D+00 | 0.53D-05 | 0.31D+04 |
| 8 | 15 | 0.00D+00 | 0.2955D+00 | 0.16D-11 | 0.30D+04 |
| 9 | 25 | 0.00D+00 | 0.2462D+00 | 0.14D+02 | 0.15D+00 |
| 10 | 15 | 0.00D+00 | 0.2090D+00 | 0.11D+02 | 0.41D+00 |

| | |
|---|---|
| Total running time: | 19min 34sec |
| Total inner point iterations: | 4451 |
| Final objective function: | 0.5576D-01 |

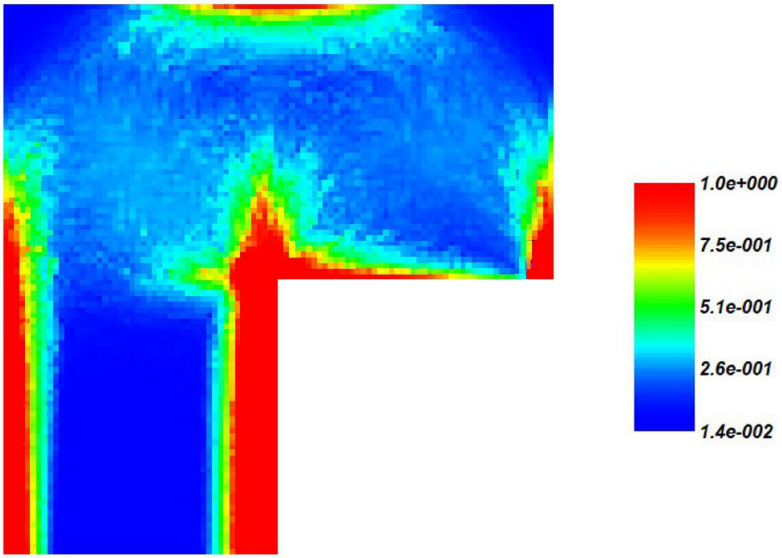
**SmallEx4 - Analytical second derivatives**



| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 25 | 0.00D+00 | 0.7317D+00 | 0.32D+02 | 0.75D+00 |
| 2 | 25 | 0.00D+00 | 0.4746D+00 | 0.39D+02 | 0.65D+00 |
| 3 | 23 | 0.00D+00 | 0.3198D+00 | 0.38D+02 | 0.42D+00 |
| 4 | 25 | 0.00D+00 | 0.2475D+00 | 0.43D+02 | 0.20D+00 |
| 5 | 1 | 0.00D+00 | 0.2475D+00 | 0.90D-06 | 0.17D+05 |
| 6 | 10 | 0.00D+00 | 0.2955D+00 | 0.71D+01 | 0.10D+05 |
| 7 | 1 | 0.00D+00 | 0.2955D+00 | 0.53D-05 | 0.31D+04 |
| 8 | 15 | 0.00D+00 | 0.2955D+00 | 0.16D-11 | 0.30D+04 |
| 9 | 25 | 0.00D+00 | 0.2462D+00 | 0.14D+02 | 0.15D+00 |
| 10 | 15 | 0.00D+00 | 0.2090D+00 | 0.11D+02 | 0.41D+00 |

| | |
|---|---|
| Total running time: | 19min 34sec |
| Total inner point iterations: | 4451 |
| Final objective function: | 0.5576D-01 |

**Plato-N - Analytical second derivatives**




| IT | QIT | FEAS | OBJ | NDX | NLX |
|---|---|---|---|---|---|
| 1 | 57 | 0.00D+00 | 0.5934D+05 | 0.46D+05 | 0.84D+00 |
| 2 | 59 | 0.00D+00 | 0.3404D+05 | 0.25D+05 | 0.84D+00 |
| 3 | 68 | 0.00D+00 | 0.1811D+05 | 0.16D+05 | 0.72D+00 |
| 4 | 62 | 0.00D+00 | 0.1189D+05 | 0.63D+04 | 0.21D+01 |
| 5 | 66 | 0.00D+00 | 0.1112D+05 | 0.16D+04 | 0.23D+01 |
| 6 | 64 | 0.00D+00 | 0.1065D+05 | 0.16D+04 | 0.19D+01 |
| 7 | 69 | 0.00D+00 | 0.1039D+05 | 0.17D+04 | 0.16D+01 |
| 8 | 70 | 0.00D+00 | 0.1019D+05 | 0.19D+04 | 0.17D+01 |
| 9 | 71 | 0.00D+00 | 0.9998D+04 | 0.19D+04 | 0.23D+01 |
| 10 | 61 | 0.00D+00 | 0.9739D+04 | 0.17D+04 | 0.65D+01 |

| | |
|---|---|
| Total running time: | 22min 37sec |
| Total inner point iterations: | 4451 |
| Final objective function: | 0.5576D-01 |

### 4.2.3 Results Table

The following table provides an overview over all testing outcomes. For each scenario, the total number of QP iterations `"QIT"` over the entire process, the final objective function value `"OBJ"`, and the total running time `"TIME"` in minutes are presented.

| | Approximated | | | Analytical | | |
|---|---|---|---|---|---|---|
| **Name** | QIT | OBJ | TIME | QIT | OBJ | TIME |
| **Standard problems** | | | | | | |
| 3 | 85 | 1.0000D-18 | < 0:01 | 8 | 2.8398D-34 | < 0:01 |
| 4 | 40 | 0.2667D+01 | < 0:01 | 20 | 0.2667D+01 | < 0:01 |
| 12 | 35343 | -0.3178D+02 | < 0:01 | 1550 | -0.3000D+02 | < 0:01 |
| 21 | 2579 | -0.9996D+02 | < 0:01 | 11 | -0.9996D+02 | < 0:01 |
| 22 | 46 | 0.1000D+01 | < 0:01 | 22 | 0.1000D+01 | < 0:01 |
| **SCPF problems** | | | | | | |
| Cantchen | 12257 | 0.3392D-01 | 0:54 | 2672 | 0.2842D-01 | 0:11 |
| Canti | 10405 | 0.6523D-01 | 10:19 | 3567 | 0.4100D-01 | 3:13 |
| Canti3 | 13757 | 0.5515D-01 | 49:59 | 2737 | 0.5898D-01 | 15:02 |
| SmallEx1 | 13531 | 0.7407D-01 | 17:41 | 3112 | 0.3469D-01 | 3:23 |
| SmallEx4 | 13671 | 0.5331D-01 | 54:24 | 4451 | 0.5576D-01 | 19:34 |
| Plato-N | - | - | - | 1563 | 0.9177D+04 | 22:37 |

## 4.3 Evaluation

In the following, an overview over important testing results is given, and appropriate conclusions are outlined.

At first, results for the standard test problems are considered.

For analytical second derivatives, all standard tests succeeded. The resulting optimum function values always were within acceptable tolerance to the ground truth data. The maximum inner or outer iteration counts were rarely ever hit, but most quadratic subproblems have been solved based on accuracy estimations.

BFGS updates solved four out of five test problems without hitting the maximum allowed iterations. For the unsolved problem, cycling occurred in later QP iterations, but at least an approximation of the actual solution was found. Perhaps this scenario would have benefitted from a raise in maximum allowed iterations for the earlier iterations.

When comparing iterations spent within any interior method problem solving, exact second derivatives vastly outperform BFGS updates on three problems. Still, runtime differences have been negligible for both approaches, with measurements well below timer resolution.

While the standard test problems already show a certain bias for analytical second derivatives, these problems also are rather small and do not necessarily reflect behaviour in the later application environment.

Thus, a more practical indication on the benefits or disadvantages of either method is given by the much larger toy problems from free material optimization.

In the FMO tests, analytical second derivatives require but few inner iterations and few quadratic subprograms to hit the minimum accuracy at each of the outer SCPF iterations. Consequently, they arrive at the maximum outer iteration cap much earlier than BFGS update strategies, and thus provide a solution in much less time.

BFGS updates, on the other hand, required more inner iterations, and frequently hit the maximum inner iteration cap instead of the set minimum accuracy.

Behaviour of both methods thus matches up with the observations made for the standard test problems.

However, as stated in the test introduction, testing was not necessarily fair in the application problems. In particular, the BFGS approach was forced to continue based on a less than optimal point when hitting the iteration cap, which may have impeded convergency at later iterations of the SCPF solver.

Considering the actual running time, the BFGS updates on average take around three times longer than analytical updates to reach the outer iteration cap.

Another interpretation comes from the stopping criterion of the actual application: In a scenario with a fixed calculation time span - such as when results are required at a set time because of project dependencies - about three times as many outer iterations of the analytical approach may be performed. Thus a better overall minimization may be provided. In a setting where a certain minimum accuracy is required for

the resulting problem solution, the analytical approach still produces results with compareable accuracy much earlier than the BFGS update strategy.

That said, the intuitive idea that higher order derivatives improve optimization has empirically been validated.

# 5 Conclusion

Within this thesis, an overview was given over a combined method for nonlinear optimization, based on an outer sequential quadratic programming approach, and an inner, interior point method.

In this context, two different strategies for handling the Hessian matrix of the Lagrange function have been discussed:

On the one hand, a BFGS update strategy keeps but an approximation of the Hessian matrix in memory. Its limited memory extension provides reduced storage space for large problems at the cost of further accuracy by implicitly reconstructing the Hessian approximation from a smaller set of vectors wherever required.

On the other hand, an exact, analytical matrix may be provided to the optimization algorithm. Both strategies may be used in determining a descent direction on the optimization domain. However, since the analytically derived Hessian matrix is more exact than its BFGS counterpart, it was intuitively believed that the analytical Hessian could be exploited for improving convergency.

Thus, in the implementation phase, an existing FORTRAN optimizer that utilized the described setup was modified to allow for analytical Hessian matrices as input in addition to its default BFGS update strategy.

Finally, both approaches have been compared in terms of convergency based on certain testing scenarios. As expected, the more exact, analytically derived Hessian significantly reduces the required number of iterations. In this context, particular speed improvements have been gained in the target application scenario of free material optimization. As such, the goal set for this thesis has been reached.

However, while much has been achieved, there still are many outstanding followup tasks.

One particular problematic solution that has not completely been solved is the degeneration of the SCPF optimization restrictions for certain FMO test cases. As described in the implementation section, this currently is treated ad hoc in the linear solver module by modifying the diagonal of the Hessian matrix if any negative eigenvalues have been found. Perhaps handling at a higher level might give better results.

Finally, one could also consider other application scenarios for future tests. For instance, the rather simple second order Newton method for robotics path planning discussed in [TW07] may be replaced by the more sophisticated solver described in this thesis.

# Appendices

## A Bibliography

The following sources have been considered in the creation of this diploma thesis:

[BS08] *"An SQP Interior Point Algorithm for Solving Large Scale Nonlinear Optimization Problems"*
B. Sachsenberg
PhD Thesis, work in progress, University of Bayreuth, November 18, 2008

[BS09] *"NLPIP: A Fortran Implementation of an SQP Interior Point Algorithm for Solving Large Scale Nonlinear Optimization Problems - User's Guide"*
B. Sachsenberg
PhD Thesis, work in progress, University of Bayreuth, April 22, 2009

[CZ04] *"Software Manual for SCPIP 3.0"*
C. Zillober
Technical Report, University of Bayreuth, 2004

[FJ04] *"Optimierung"*
F. Jarre, J. Stoer
Springer-Verlag, 2004

[HK51] *"Nonlinear Programming"*
H. W. Kuhn, A. W. Tucker
Proceedings of 2nd Berkeley Symposium, Berkeley University of California Press, pp. 481–492, 1951

[IG04] *"Convergence Analysis of a Primal-Dual Interior-Point Method for Nonlinear Programming"*
I. Griva, D. F. Shanno, R. J. Vanderbei
July 21, 2004

[KL44] *"A Method for the Solution of Certain Non-linear Problems"*
K. Levenberg
Quarterly of Applied Mathematics, 2(2), pp. 164–168, July, 1944

[KS08] *"Wissenschaftliches Rechnen"*
K. Schittkowski
Lecture Notes, Winter Term 2008-2009

[KS09] *"Test Examples for Nonlinear Programming Codes - All Problems from the Hock-Schittkowski Collection"*
K. Schittkowski
Report, Department of Computer Science, University of Bayreuth, 2009

[MK99] *"MOPED User's Guide, Version 1.02"*
M. Kocvara, M. Stingl, R. Werner
Research Report 262, Institute of Applied Mathematics, University of Erlangen, 1999

[MP78a] *"A Fast Algorithm for Nonlinearly Constrained Optimization Calculations"*

M. J. D. Powell
Lecture Notes in Matehmatics 630, Springer-Verlag, Berlin, pp. 144-157, 1978

[MP78b] *"The Convergence of Variable Metric Methods for Nonlinearly Constrained Optimization Calculations"*
M. J. D. Powell
Nonlinear Programming 3, Academic Press, New York, pp. 27-63

[NK84] *"A New Polynomial Time Algorithm for Linear Programming"*
N. Karmarkar
Combinatorica, Volume 4, number 4, pp. 373–395, 1984

[OS04] *"Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO"*
O. Schenk, K. Gärtner
Journal of Future Generation Computer Systems, pp. 475-487, 2004

[PA00] *"Multifrontal Parallel Distributed Symmetric and Unsymmetric Solvers"*
P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent
Computational Methods in Applied Mechanical Engineering, pp. 501-520, 2000

[RS10] *"Topologieoptimierung mit Faserverbundwerkstoffen"*
RISC Software GmbH
Web Release on `http://industry.risc.uni-linz.ac.at`, July, 2010

[RV00] *"On Formulating Semidefinite Programming Problems as Smooth Convex Nonlinear Optimization Problems"*
R. J. Vanderbei, H. Y. Benson
Princeton University, January 27, 2000

[SE06] *"Anwendung von Filtermethoden auf das Optimierungsverfahren SCP"*
S. Ertel
Diploma Thesis, University of Bayreuth, August 6, 2006

[SE09] *"A Feasible Sequential Convex Programming Method for Free Material Optimization"*
S. Ertel, K. Schittkowski, C. Zillober
8th World Congress on Structural and Multidisciplinary Optimization, Lisbon, Portugal, June 1-5, 2009

[TW07] *"Optimal Specialization of Underdetermined Paths in Robotics"*
T. Werner
Bachelor Thesis, University of Bayreuth, February, 2007

[WPa] *"Lagrange Multipliers"*
Wikipedia, the free encyclopedia
Web Release on `http://en.wikipedia.org/wiki/Lagrange_multipliers`, May 3, 2010

[WW06] *"Mathematik für Physiker"*
W. von Wahl, H. Kerner
Springer-Verlag, 2006

## B HSSSLV source code

The following section provides the source code for the HSSSLV module for reference. Note that source code comments have been stripped for compactness. Appropriate documentation can be found in the implementation chapter 3, or in the original source on the accompanying compact disc.

```
      MODULE GRADIENTDEF
        TYPE :: GRADIENT
             INTEGER :: LENG_GRAD
             INTEGER, POINTER :: COL_GRAD(:)
             INTEGER, POINTER :: ROW_GRAD(:)
             DOUBLEPRECISION, POINTER :: VAL_GRAD(:)

             LOGICAL :: USE_HESSE

             INTEGER :: LENG_HESSE
             INTEGER, POINTER :: COL_HESSE(:)
             INTEGER, POINTER :: ROW_HESSE(:)
             DOUBLEPRECISION, POINTER :: VAL_HESSE(:)

             INTEGER, POINTER :: IWA(:)
             DOUBLEPRECISION, POINTER :: DWA(:)
        END TYPE
      END MODULE


      SUBROUTINE LINSLV(MODE, M, MMAX, N, I, DG, X, Y)
        USE GRADIENTDEF
        IMPLICIT NONE
        INTEGER MODE, M, N, I
        INTEGER MMAX
        TYPE(GRADIENT) DG
        DOUBLEPRECISION X,Y
        DIMENSION X(N + M),Y(N + M, MMAX)
        INTEGER J

        GOTO (1,2,3,4,5,6,7,8), MODE

!       Execute appropriate LINSLV operation.
1       CONTINUE
          CALL HSSDCOOMV(N, M, DG%LENG_GRAD,
     /                   DG%VAL_GRAD, DG%ROW_GRAD, DG%COL_GRAD,
     /                   DBLE(I), Y, 1D0, X, .FALSE., .FALSE.,
     /                   DG%DWA)
        RETURN
2       CONTINUE
          CALL HSSDCOOMV(N, M, DG%LENG_GRAD,
     /                   DG%VAL_GRAD, DG%ROW_GRAD, DG%COL_GRAD,
     /                   DBLE(I), Y, 1D0, X, .TRUE., .FALSE.,
```

```
     /                              DG%DWA)
        RETURN
3       CONTINUE
          CALL HSSSKALMUL(M, N, I, DG, X, Y)
        RETURN
4       CONTINUE
          CALL HSSSL6(1,M,N,X,Y,DG,I)
        RETURN
5       CONTINUE
          CALL HSSSL6(2,M,N,X,Y,DG,I)
        RETURN
6       CONTINUE
          CALL HSSSL6(3,M,N,X,Y,DG,I)
        RETURN
7       CONTINUE
          DO J=1,MMAX
            CALL HSSSL6(3,M,N,X,Y(1,J),DG,I)
          ENDDO
        RETURN
8       CONTINUE
          CALL HSSMLT(N, DG, Y, X)
        RETURN
      END


!     Linear factorization and solution with MUMPS
      SUBROUTINE HSSSL6(MODE,M,N,D1,D2,DG,I)
        USE GRADIENTDEF
        IMPLICIT NONE
        INCLUDE 'dmumps_struc.h'
        INTEGER MODE,M,N,I
        DOUBLEPRECISION, TARGET :: D1, D2
        DIMENSION D1(N + M), D2(N + M)
        TYPE(GRADIENT) DG
        INTEGER J, IERR, NNZ
        DOUBLE PRECISION FAKT
        LOGICAL, SAVE :: MUMPSINIT = .FALSE.
        TYPE(DMUMPS_STRUC), SAVE :: MUMPS_PAR

!       Hessian toggle for BFGS updates
        IF (DG%USE_HESSE) THEN
          NNZ = DG%LENG_GRAD + DG%LENG_HESSE + M
        ELSE
          NNZ = DG%LENG_GRAD + N + M
        ENDIF

!       Mumps initialization
        IF (NOT(MUMPSINIT)) THEN

          MUMPS_PAR%JOB = -1
```

```
          MUMPS_PAR%COMM = 0
          MUMPS_PAR%PAR = 1
          MUMPS_PAR%SYM = 1

          CALL MPI_INIT(IERR)
          CALL DMUMPS(MUMPS_PAR)
          CALL MPI_FINALIZE(IERR)

!         Enable overallocation
          IF (MUMPS_PAR%INFO(1) .EQ. -9)THEN
            MUMPS_PAR%ICNTL(14) = 50
          ENDIF

          MUMPS_PAR%ICNTL(3) = 0
          MUMPS_PAR%ICNTL(13) = 2
          MUMPS_PAR%ICNTL(5) = 0
          MUMPS_PAR%ICNTL(18) = 0

          MUMPSINIT = .TRUE.
        ENDIF

        MUMPS_PAR%N = N + M
        MUMPS_PAR%IRN => DG%IWA(NNZ + 1: 2 * NNZ)
        MUMPS_PAR%JCN => DG%IWA(1:NNZ)
        MUMPS_PAR%A => DG%DWA(1:NNZ)
        MUMPS_PAR%NZ = NNZ

!       Build complete, sparse matrix C
        IF (MODE.EQ.1 .OR. MODE.EQ.2) THEN
          IF (DG%USE_HESSE) THEN
            DO J = 1, N
              MUMPS_PAR%A(J) = DG%VAL_HESSE(J) + D1(J)
              MUMPS_PAR%JCN(J) = J
              MUMPS_PAR%IRN(J) = J
            ENDDO
            DO J = N+1, DG%LENG_HESSE
              MUMPS_PAR%A(J) = DG%VAL_HESSE(J)
              MUMPS_PAR%JCN(J) = DG%COL_HESSE(J)
              MUMPS_PAR%IRN(J) = DG%ROW_HESSE(J)
            ENDDO
            DO J = 1, DG%LENG_GRAD
              MUMPS_PAR%A(DG%LENG_HESSE+J)=DG%VAL_GRAD(J)
              MUMPS_PAR%JCN(DG%LENG_HESSE+J)=DG%COL_GRAD(J)+N
              MUMPS_PAR%IRN(DG%LENG_HESSE+J)=DG%ROW_GRAD(J)
            ENDDO
            DO J = 1, M
              MUMPS_PAR%A(DG%LENG_GRAD+DG%LENG_HESSE+J)=-D2(J)
              MUMPS_PAR%JCN(DG%LENG_GRAD+DG%LENG_HESSE+J)=N+J
              MUMPS_PAR%IRN(DG%LENG_GRAD+DG%LENG_HESSE+J)=N+J
            ENDDO
```

```
            ELSE
              DO J = 1, N
                MUMPS_PAR%A(J) = D1(J)
                MUMPS_PAR%JCN(J) = J
                MUMPS_PAR%IRN(J) = J
              ENDDO
              DO J = 1, DG%LENG_GRAD
                MUMPS_PAR%A(N + J) = DG%VAL_GRAD(J)
                MUMPS_PAR%JCN(N + J) = DG%COL_GRAD(J) + N
                MUMPS_PAR%IRN(N + J) = DG%ROW_GRAD(J)
              ENDDO
              DO J = 1, M
                MUMPS_PAR%A(N + DG%LENG_GRAD + J) = -D2(J)
                MUMPS_PAR%JCN(N + DG%LENG_GRAD + J) = N + J
                MUMPS_PAR%IRN(N + DG%LENG_GRAD + J) = N + J
              ENDDO
            ENDIF
          ENDIF
          CONTINUE

!         Actual solver call for current solution mode
          IF (MODE.EQ.1) THEN
            MUMPS_PAR%JOB = 4
          ELSEIF (MODE.EQ.2) THEN
            MUMPS_PAR%JOB = 2
          ELSEIF (MODE.EQ.3) THEN
            MUMPS_PAR%JOB = 3
            MUMPS_PAR%RHS => D2
            MUMPS_PAR%NRHS = 1
            MUMPS_PAR%LRHS = N + M
          ENDIF

          CALL MPI_INIT(IERR)
          CALL DMUMPS(MUMPS_PAR)
          CALL MPI_FINALIZE(IERR)

!         Correction for non-positive-semidefinite matrices
          IF (MODE .EQ. 3) THEN
            J = 0
            DO
      /     WHILE(MUMPS_PAR%INFO(12).GT.DG%COL_GRAD(DG%LENG_GRAD)
      /                 .AND. J .LE. 4)
              J = J+1
              FAKT = 1.D1
              DO I=1,J-1
                FAKT = 1.D1*FAKT
              ENDDO
              DO I=1, N
                MUMPS_PAR%A(I) = MUMPS_PAR%A(I) + FAKT
              ENDDO
```

```
            MUMPS_PAR%JOB = 2
            CALL MPI_INIT(IERR)
            CALL DMUMPS(MUMPS_PAR)
            CALL MPI_FINALIZE(IERR)
            MUMPS_PAR%JOB = 3
            CALL MPI_INIT(IERR)
            CALL DMUMPS(MUMPS_PAR)
            CALL MPI_FINALIZE(IERR)
          ENDDO
        ENDIF

        I = MUMPS_PAR%INFOG(1)
      END


!     Calculate B = Hessian(in DG) * DL
      SUBROUTINE HSSMLT(N, DG, DL, B)
            USE GRADIENTDEF
        IMPLICIT NONE
        INTEGER N
        TYPE(GRADIENT) DG
        DOUBLEPRECISION DL(N), B(N)
        DOUBLEPRECISION, POINTER :: TMP(:)
        INTEGER J

        TMP => DG%DWA(1:N)
        DO J=1,N
          TMP(J) = 0.0D0
        ENDDO
        CALL HSSDCOOMV(N, N, DG%LENG_HESSE,
     /                 DG%VAL_HESSE, DG%ROW_HESSE, DG%COL_HESSE,
     /                 1D0, DL, 0D0, TMP, .FALSE., .TRUE.,
     /                 DG%DWA)
        DO J=1,N
          B(J) = TMP(J)
        ENDDO
      END


!     Calculate X(1) = DG(I) * Y
      SUBROUTINE HSSSKALMUL(M, N, I, DG, X, Y)
        USE GRADIENTDEF
        IMPLICIT NONE
        INTEGER M, N, I
        TYPE(GRADIENT) DG
        DOUBLEPRECISION X,Y
        DIMENSION X(N + M), Y(N + M)
        INTEGER J

        X(1) = 0.0D0
```

```
          DO J=1,DG%LENG_GRAD
            IF (DG%COL_GRAD(J) .EQ. I) THEN
              X(1) = X(1) + DG%VAL_GRAD(J) * Y(DG%ROW_GRAD(J))
              IF (DG%COL_GRAD(J) .GT. I) THEN
                GOTO 11
              ENDIF
            ENDIF
          ENDDO
11        CONTINUE
        END


!       Multiplication Y = ALPHA * M * X + BETA * Y
        SUBROUTINE HSSDCOOMV
       /          (M, K, NUMELE, VALS, ROWS, COLS,
       /           ALPHA, X, BETA, Y, TRANSPOSE, SYMMETRIC, WORK)
          IMPLICIT NONE
          INTEGER M, K, NUMELE
          DOUBLEPRECISION VALS(1:NUMELE)
          INTEGER ROWS(1:NUMELE), COLS(1:NUMELE)
          DOUBLEPRECISION ALPHA, BETA
          DOUBLEPRECISION X(1:M+K), Y(1:M+K)
          LOGICAL TRANSPOSE, SYMMETRIC
          DOUBLEPRECISION WORK(M + K)
          INTEGER I, ROWCNT, COLCNT, CURROW, CURCOL

          IF (TRANSPOSE .EQ. .TRUE.) THEN
            ROWCNT = K
            COLCNT = M
          ELSE
            ROWCNT = M
            COLCNT = K
          ENDIF

          DO I = 1, ROWCNT
            WORK(I) = 0.0D0
          ENDDO

          DO I = 1, NUMELE
            IF (TRANSPOSE) THEN
              CURROW = COLS(I)
              CURCOL = ROWS(I)
            ELSE
              CURROW = ROWS(I)
              CURCOL = COLS(I)
            ENDIF
            WORK(CURROW) = WORK(CURROW) + VALS(I) * X(CURCOL)
            IF (SYMMETRIC) THEN
              IF (CURROW .NE. CURCOL) THEN
                WORK(CURCOL) = WORK(CURCOL) + VALS(I) * X(CURROW)
```

```
            ENDIF
          ENDIF
        ENDDO

        DO I = 1, ROWCNT
          Y(I) = ALPHA * WORK(I) + BETA * Y(I)
        ENDDO
      END
```

# C Maple to FORTRAN 70 Conversion Tool

For several problems, such as the superset SCPF solver for free material optimization, calculating analytical second derivatives is possible, yet hard to perform by hand. Thus, Maple has been used to determine the derivatives. The resulting equations have been exported to FORTRAN 70 code with Maple's automatic code generation utility.

However, there are several quirks in the Maple-generated FORTRAN code. For instance, temporary working variables are chosen with almost random indices and there is no way to offset or resort the column and row indices of the output matrices. Optimizing a few thousand lines of autogenerated code by hand is a tedious occupation. For that reason, a conversion utility for optimizing Maple output has been written in the context of this thesis.

The main screen of the tool is shown in figure C.1.

In a typical application scenario, the user pastes the Maple-generated code into the central text entry box. Then, the conversion process is configured with the upper-hand options. For instance, the variable names from Maple are to be indicated, and a replacement name may be selected for each of these. After a click on the left-hand conversion button, the provided Maple-generated code is transformed into optimized FORTRAN 70 under consideration of the current options. This includes the following operations:

- Variable names are replaced with the provided new identifiers.
- Matrices are split up into the sparse column-row-value format of SCPF that has been described in the implementation chapter 3. Diagonal elements are stored in the first array fields, followed by other elements from the upper right matrix triangle. As a symmetry optimization for hessian matrices, the lower left triangle is discarded.
- User-defined array offsets are appended where desired. This allows to store multiple matrices in one continuous storage space.
- Temporary working variables are compacted into a single, continuous array.
- Faulty line breaks in the original code are removed, and reinserted at the FORTRAN 70 maximum width of 72 characters.

The resulting code is placed into the lower-hand output pane and can be copied into an appropriate FORTRAN source file.

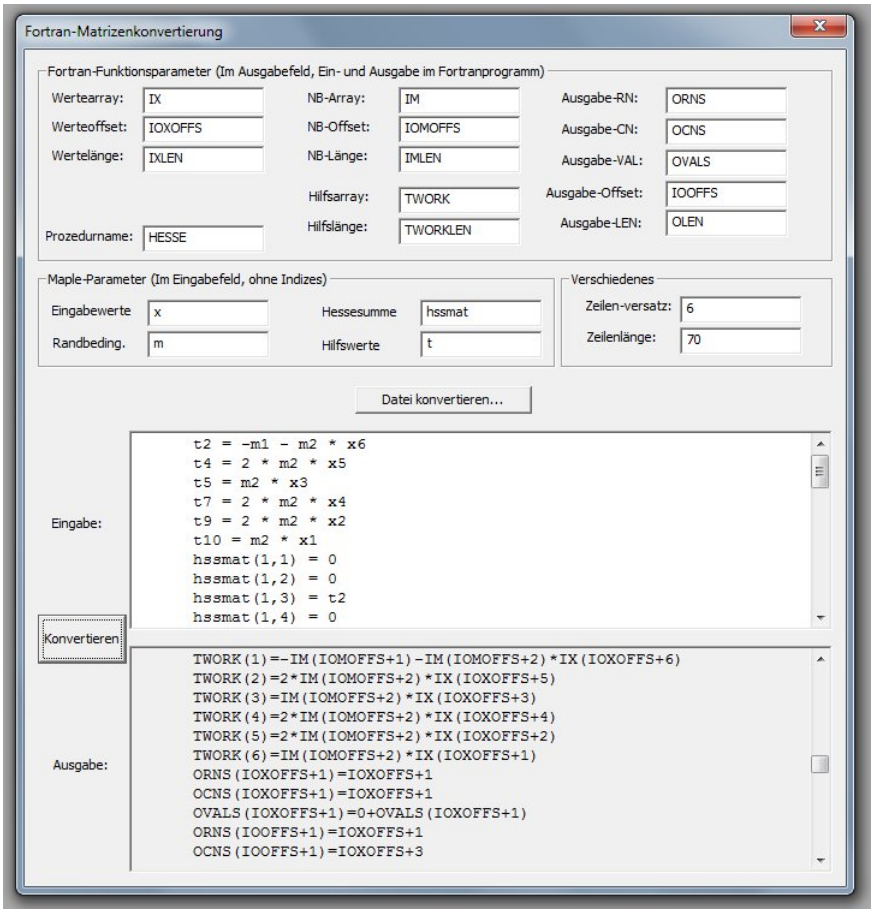



Figure C.1: The main screen of the Maple to FORTRAN 70 conversion tool.

For instance, Maple generates the following code excerpt for the Hessian of the Lagrange function in SCPF:

```
t2 = -m1 - m2 * x6
t4 = 2 * m2 * x5
t5 = m2 * x3
t7 = 2 * m2 * x4
t9 = 2 * m2 * x2
t10 = m2 * x1
hssmat(1,1) = 0
hssmat(1,2) = 0
hssmat(1,3) = t2
hssmat(1,4) = 0
hssmat(1,5) = t4
hssmat(1,6) = -t5
hssmat(2,1) = 0
hssmat(2,2) = -2 * t2
hssmat(2,3) = 0
hssmat(2,4) = -t4
hssmat(2,5) = -t7
hssmat(2,6) = t9
hssmat(3,1) = t2
hssmat(3,2) = 0
hssmat(3,3) = 0
hssmat(3,4) = t7
hssmat(3,5) = 0
hssmat(3,6) = -t10
hssmat(4,1) = 0
hssmat(4,2) = -t4
hssmat(4,3) = t7
hssmat(4,4) = 2 * t5
hssmat(4,5) = -t9
hssmat(4,6) = 0
hssmat(5,1) = t4
hssmat(5,2) = -t7
hssmat(5,3) = 0
hssmat(5,4) = -t9
hssmat(5,5) = 2 * t10
hssmat(5,6) = 0
hssmat(6,1) = -t5
hssmat(6,2) = t9
hssmat(6,3) = -t10
hssmat(6,4) = 0
hssmat(6,5) = 0
hssmat(6,6) = 0
```

The Maple-to-FORTRAN converter then transforms the above excerpt into the below FORTRAN 70 routine:

```
      SUBROUTINE HSUM33(IX, IOXOFFS, IXLEN, TWORK,
     #TWORKLEN, IM, IOMOFFS, IMLEN, OCNS, ORNS, OVALS,
     #IOOFFS, OLEN)
      IMPLICIT NONE
      DOUBLEPRECISION IX, TWORK, IM, OVALS
      INTEGER IOXOFFS, IXLEN, TWORKLEN, IOMOFFS, IMLEN, OCNS,
     #ORNS, IOOFFS, OLEN
      DIMENSION IX(IXLEN), TWORK(TWORKLEN), IM(IMLEN),
     #OCNS(OLEN), ORNS(OLEN), OVALS(OLEN)
      TWORK(1)=-IM(IOMOFFS+1)-IM(IOMOFFS+2)*IX(IOXOFFS+6)
      TWORK(2)=2*IM(IOMOFFS+2)*IX(IOXOFFS+5)
      TWORK(3)=IM(IOMOFFS+2)*IX(IOXOFFS+3)
      TWORK(4)=2*IM(IOMOFFS+2)*IX(IOXOFFS+4)
      TWORK(5)=2*IM(IOMOFFS+2)*IX(IOXOFFS+2)
      TWORK(6)=IM(IOMOFFS+2)*IX(IOXOFFS+1)
      ORNS(IOXOFFS+1)=IOXOFFS+1
      OCNS(IOXOFFS+1)=IOXOFFS+1
      OVALS(IOXOFFS+1)=0+OVALS(IOXOFFS+1)
      ORNS(IOOFFS+1)=IOXOFFS+1
      OCNS(IOOFFS+1)=IOXOFFS+3
      OVALS(IOOFFS+1)=TWORK(1)
      ORNS(IOOFFS+2)=IOXOFFS+1
      OCNS(IOOFFS+2)=IOXOFFS+5
      OVALS(IOOFFS+2)=TWORK(2)
      ORNS(IOOFFS+3)=IOXOFFS+1
      OCNS(IOOFFS+3)=IOXOFFS+6
      OVALS(IOOFFS+3)=-TWORK(3)
      ORNS(IOXOFFS+2)=IOXOFFS+2
      OCNS(IOXOFFS+2)=IOXOFFS+2
      OVALS(IOXOFFS+2)=-2*TWORK(1)+OVALS(IOXOFFS+2)
      ORNS(IOOFFS+4)=IOXOFFS+2
      OCNS(IOOFFS+4)=IOXOFFS+4
      OVALS(IOOFFS+4)=-TWORK(2)
      ORNS(IOOFFS+5)=IOXOFFS+2
      OCNS(IOOFFS+5)=IOXOFFS+5
      OVALS(IOOFFS+5)=-TWORK(4)
      ORNS(IOOFFS+6)=IOXOFFS+2
      OCNS(IOOFFS+6)=IOXOFFS+6
      OVALS(IOOFFS+6)=TWORK(5)
      ORNS(IOXOFFS+3)=IOXOFFS+3
      OCNS(IOXOFFS+3)=IOXOFFS+3
      OVALS(IOXOFFS+3)=0+OVALS(IOXOFFS+3)
      ORNS(IOOFFS+7)=IOXOFFS+3
      OCNS(IOOFFS+7)=IOXOFFS+4
      OVALS(IOOFFS+7)=TWORK(4)
      ORNS(IOOFFS+8)=IOXOFFS+3
      OCNS(IOOFFS+8)=IOXOFFS+6
```

```
OVALS(IOOFFS+8)=-TWORK(6)
ORNS(IOXOFFS+4)=IOXOFFS+4
OCNS(IOXOFFS+4)=IOXOFFS+4
OVALS(IOXOFFS+4)=2*TWORK(3)+OVALS(IOXOFFS+4)
ORNS(IOOFFS+9)=IOXOFFS+4
OCNS(IOOFFS+9)=IOXOFFS+5
OVALS(IOOFFS+9)=-TWORK(5)
ORNS(IOXOFFS+5)=IOXOFFS+5
OCNS(IOXOFFS+5)=IOXOFFS+5
OVALS(IOXOFFS+5)=2*TWORK(6)+OVALS(IOXOFFS+5)
ORNS(IOXOFFS+6)=IOXOFFS+6
OCNS(IOXOFFS+6)=IOXOFFS+6
OVALS(IOXOFFS+6)=0+OVALS(IOXOFFS+6)
IOXOFFS=IOXOFFS+6
IOMOFFS=IOMOFFS+2
IOOFFS=IOOFFS+9
END
```

Internally, the conversion tool has been written in C++, using Visual Studio and the MFC library. A hand-written, FLEX-based FORTRAN parser is used as a frontend for analyzing the Maple input.

Output correctness has been ensured by a series of tests. For instance, random number sequences have been used as input vectors to generate hessian matrices from the optimized FORTRAN code. The same random number sequences have been used to generate numerical approximations of the Hessian matrices from within Maple. The resulting matrices have been compared, and were found to be almost identical, with but small rounding-induced errors.

# D Compact Disc Contents

This section details the contents of the enclosed compact disc with the source code and compiled applications that accompany this diploma thesis:

- `"Thesis"`

  A folder containing the latex files and jpeg images that compose this thesis paper. Also provided is a TeXnicCenter project `"DiplomaThesis.tcp"`, and a Windows-only building helper script `"FetchResult.bat"`.

- `"Thesis/Build"`

  Any intermediate files and the resulting PDF file are generated within the Build subfolder. `"FetchResult.bat"` is automatically called by the TeXnicCenter project to copy the PDF back into the root `"Thesis"` folder.

- `"Thesis/Figures"` **and** `"Thesis/Results"`

  JPEG graphics and OpenOffice drawing files for the figures within this thesis are stored here.

- `"Bibliography"`

  Any available electronical versions of the literature sources that have been used in the creation of this thesis.

- `"Software"`

  All software source code written to accompany this thesis is stored within this directory. Note that required third party software components have not been included for licensing reasons.

- `"Software/Hessian"`

  Source code of the Maple to Fortran conversion utility and binaries for Windows x64 reside within the `"Vc"` subdirectory. Example input and result files for the SCPF problem are respectively stored within the `"Input"` and `"Results"` subdirectories. Refer to the Maple code transformation appendix C for further details.

- `"Software/Helper"`

  Various scripts that have been employed to perform automated test evaluation.

- `"Software/Derivatives"`

  Source code for the modified LINSLV module HSSSLV. Additionally, a PARDISO-based legacy version HSSSLV_PARDISO is provided for reference. Consult the implementation chapter 3 for further details.

- `"Software/StandardTests"`

  Source code for the NLPIP wrapper used for the standard tests. Refer to the tests chapter 4 for further details.

- `"Protocols"`

  Result and log files for all testing runs from chapter 4 are provided within this directory. Files with an `".eps"` extension are images of the resulting material distribution, and can be displayed in near to every popular image viewer. Files with any other extension provide various types of human-readable data, and can be opened with any text editor. For instance, the `"fort.32"` file contains iteration results from both inner and outer iterates of the SCPF problem.

**Attention:** The compact disc is only provided for convenience, and has not been approved for public release.

# E Deutschsprachige Zusammenfassung

Nach Prüfungsordnung ist Diplomarbeiten in einer anderen als der deutschen Sprache eine deutschsprachige Zusammenfassung beizulegen. In diesem Abschnitt werden daher die Ergebnisse meiner Arbeit kurz auf Deutsch erläutert.

Ziel der Arbeit war die Optimierung eines bestehenden Verfahrens zur Lösung nichtlinearer Programme durch die Einbindung analytisch berechneter zweiter Ableitungen. Das ursprüngliche Lösungsverfahren setzt dabei innerhalb einer sequentiell-quadratischen Methode ein Innere-Punkte-Verfahren ein, in dem zweite Ableitungen der Lagrangefunktion durch ein BFGS-Update approximiert werden. Statt dieses BFGS-Updates waren nun analytisch berechnete, zweite Ableitungen zu verwenden. Dazu musste das bestehende FORTRAN-Programm NLPIP mit der Implementierung des Verfahrens angepasst werden. Anschliessend wurde das Konvergenzverhalten beider Programmversionen verglichen.

Im Folgenden wird zunächst der mathematische Hintergrund der nichtlinearen Programmierung stichpunkthaft dargelegt. Dabei wird insbesondere auf sequentiell-quadratische Programmierung, Innere-Punkte-Verfahren, sowie Newton- und Quasi-Newton-Methoden eingegangen. Die theoretische Grundlage basiert dabei auf [KS08], [FJ04], sowie [IG04].

Anschliessend wird die vorgenommene FORTRAN-Programmierung kurz erläutert, entsprechende Quellen finden sich bei [BS08], [BS09], [SE09], und [CZ04].

Nach Beschreibung der Implementierung wird das für die späteren Tests bedeutende Anwendungsbeispiel der freien Materialoptimierung vorgestellt.

Schliesslich wird ein Überblick der durchgeführten Tests auf Basis von [SE09] und [KS09] geboten. Darauf folgt eine Zusammenfassung der wichtigsten Testergebnisse. Abschliessend wird die Erweiterung des Verfahrens anhand dieser Ergebnisse beurteilt.

## E.1 Mathematische Grundlagen

Allgemein ist das Ziel der nichtlinearen Optimierung das Finden eines Optimalpunktes - also eines Minimums oder eines Maximums - auf einer beliebigen, reellen Funktion $f : \mathbb{R}^m \to \mathbb{R}$.

Der Optimierungsbereich $P$ über der Funktion $f$ wird dabei durch eine Menge an Gleichheits- bzw. Ungleichheitsbedingungen auf weiteren reellen Funktionen

$$h_i : \mathbb{R}^m \to \mathbb{R} \qquad \text{für } i = 1, \ldots, n = p + q \tag{1}$$

festgelegt.

Ohne Beschränkung der Allgemeinheit lässt sich schliesslich das zugehörige Optimie-

rungsproblem als Minimierungsproblem

$$\begin{aligned} &\min_{x \in \mathbb{R}^m} (f(x)) && && (2)\\ &\quad \text{so dass:} && h_i(x) = 0 && \text{für } i = 1, \ldots, p \\ &\quad && h_i(x) \geq 0 && \text{für } i = p+1, \ldots, n \end{aligned}$$

formulieren.

Im weiteren Text seien die Funktionen $h_i$ dabei so belegt, dass der pathologische Fall eines leeren Optimierungsbereichs ausgeschlossen ist.

Ein spezielles Teilgebiet der nichtlinearen Optimierungsaufgaben bildet die konvexe Optimierung. Hierbei ist die Zielfunktion konvex. Gleichheitsbedingungen werden als affin, Ungleichheitsbedingungen als konkav vorrausgesetzt. Durch diese Einschränkungen wird die mathematische Bearbeitung des nichtlinearen Problems erleichtert. Im Folgenden werden daher lediglich konvexe Optimierungsprobleme betrachtet.

Eine grosse Rolle bei der Identifikation von Optimalwerten kommt dabei der Lagrange-Funktion

$$\begin{aligned} &L : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R} \\ &L(x, u) = f(x) - \sum_{j=1}^{n} u_j g_j(x) && (3) \end{aligned}$$

zu. Die Komponenten des Vektors $u$ sind gemeinhin als Lagrange-Multiplikatoren bekannt.

Anschaulich gesprochen spiegelt die Lagrange-Funktion das Gleichgewicht oder Ungleichgewicht zwischen Zielfunktion und Beschränkungsfunktionen wider. Für illustrierte Beispiele sei auf die Abbildungen 2.4 und 2.5 im englischsprachigen Teil der Arbeit verwiesen.

Mathematisch präzisiert wird die Aussage der Lagrange-Funktion durch die KKT-Bedingungen. Man sagt, ein Punkt $x \in \mathbb{R}^m$ erfüllt die KKT-Bedingungen, falls es einen Vektor $u$ aus Lagrange-Multiplikatoren gibt, so dass:

| | | |
|---|---|---|
| $u_j \geq 0$ | für $j = p+1, \ldots, n$ | (Positivität) |
| $\nabla_x L(x, u) = 0$ | | (Stationarität) |
| $u_j h_j(x) = 0$ | für $j = p+1, \ldots, n$ | (Komplementarität) |

Eine graphische Darstellung der Bedeutung der KKT-Bedingungen findet sich in Abbildung 2.6 im englischsprachigen Hauptteil.

Mathematisch gesehen sind jedoch auch die KKT-Bedingungen alleine weder notwendig, noch hinreichend für die Existenz eines Minimums an einem gegebenen Punkt.

Durch weitere Bedingungen, sogenannte Regularisierungsforderungen, lässt sich die Aussagekraft der KKT-Bedingungen stärken. Es gibt eine Reihe solcher Anforderungen, mit verschieden starken Einschränkungen an den Punkt $x$ und die beteiligten

Beschränkungsfunktionen. Beispielhaft sei hier die Slater-Bedingung angeführt. Diese benötigt für eine stärkere Aussage die Existenz eines Punktes $x^* \in P$, der nicht auf einer der Funktionen für die Beschränkungsungleichungen liegt - also $h_i(x^*) > 0$ für alle $j = p+1, \ldots, n$. Gilt die Slater-Bedingung für ein Optimierungsproblem, so ist die Erfüllung der KKT-Bedingungen an einem Punkt zumindest eine notwendige Bedingung für ein Minimum.

Basierend auf den KKT-Bedingungen lässt sich nun das sequentielle, quadratische Programmierverfahren motivieren. Dazu sollen in einem iterativen Verfahren $x_k$ und $u_k$ bestimmt werden, so dass diese gegen eine Lösung des nichtlinearen Problems konvergieren. Insbesondere wird eine Funktion $\Gamma : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^{n+1}$ aus den KKT-Bedingungen zusammengesetzt:

$$\Gamma(x,u) := \begin{pmatrix} \nabla_x L(x,u) = \nabla f(x) - \sum_{i=1}^{n} u_i \nabla h_i(x) \\ -h_1(x) \\ \vdots \\ -h_p(x) \\ -u_{p+1} h_{p+1}(x) \\ \vdots \\ -u_n h_n(x) \end{pmatrix}. \tag{4}$$

Für einen zulässigen Punkt, der die KKT-Bedingungen erfüllt, liegt bei $\Gamma$ eine Nullstelle vor, es handelt sich womöglich um das gesuchte Minimum der Zielfunktion.

Auf $\Gamma$ wird daher nun eine Nullstellen-Suche nach Newton durchgeführt. Statt dem eigentlichen Newton-Schritt der Nullstellensuche wird jedoch ein modifiziertes Update angewandt. Dabei gilt die Update-Regel

$$\big(J_\Gamma(x_k, u_{k+1})\big)(\Delta x_k, \Delta u_k) = -\Gamma(x_k, u_k), \tag{5}$$

die in $\Delta x_k$ und $\Delta u_k$ ein nichtlineares Gleichungssystem ist. Durch Einsetzen, Umformen und Vereinfachen lässt sich diese Regel auf ein alternatives, nichtlineares Problem

$$\min_{d \in \mathbb{R}^m} \frac{1}{2} d^T \nabla_x^2 L(x_k, u_{k+1})\, d \;+\; \nabla f(x_k)^T\, d \tag{6}$$

$$\begin{aligned} \text{so dass: } \nabla h_i^T(x_k)\, d \;+\; h_i(x_k) \;&=\; 0 && \text{für } i = 1, \ldots, p, \\ \nabla h_i^T(x_k)\, d \;+\; h_i(x_k) \;&\geq\; 0 && \text{für } i = p+1, \ldots, n. \end{aligned}$$

zurückführen. Dabei ergeben sich $\Delta x_k$ und $u_{k+1}$ als Lösung $d^*$ und zugehörige Lagrangemultiplikatoren $u^*$ dieses Minimierungsproblems.

Unter der Vorraussetzung, dass die Hessematrix $\nabla_x^2 L$ der Lagrange-Funktion positiv definit ist, handelt es sich bei dem neuen Problem um ein quadratisches, konvexes Problem: Die Zielfunktion ist eine Quadrik, die Nebenbedingungen sind affin respektive konkav.

Im SQP-Verfahren werden nun iterativ nach diesem Schema quadratische Probleme aufgestellt. Deren Lösung wird an einen anderen Algorithmus delegiert, die Ergebnispunkte dienen im Anschluss als Abstiegsrichtung im ursprünglichen Problem. Der

SQP-Algorithmus wird in Abschnitt 2.5.3 im englischsprachigen Teil der Arbeit in Pseudocode wiedergegeben.

Die Lösung des quadratischen Problems wird dabei in der Implementierung NLPIP von einem Innere-Punkte-Verfahren ermittelt.

Beim Innere-Punkte-Verfahren wird ebenfalls eine Newton-Suche auf Basis der KKT-Bedingungen durchgeführt, diesmal auf der quadratischen Approximation des Ursprungsproblems. Statt jedoch die Newton-Abstiegsregel anzupassen, wird bei der Inneren-Punkte-Methode die Problemformulierung verändert. Insbesondere werden Schlupfvariablen $w = (w_{p+1}, \ldots, w_n) \in \mathbb{R}^q$ mit $w \geq 0$, $h_i() - w_i = 0$ für $i = p+1, \ldots, n$ in das ursprüngliche Problem eingeführt. Zudem wird die Zielfunktion um Barriereterme erweitert, die an die Stelle der früheren Ungleichungsrestriktionen treten. Es ergibt sich eine neue Zielfunktion $\phi : \mathbb{R}^m \times \mathbb{R}^q \to \mathbb{R}$,

$$\phi(x, w) := f(x) - \mu \sum_{i=p+1}^{n} ln(w_i), \tag{7}$$

für einen Barriere-Gewichtungsparameter $\mu > 0$. Das zugehörige Minimierungsproblem ist

$$\begin{aligned} &\min_{x \in \mathbb{R}^m, w \in \mathbb{R}^q} \phi(x, w) && && (8)\\ &\text{so dass:} && h_i(x) = 0 && \text{für } i = 1, \ldots, p \\ &&& h_i(x) - w_i = 0 && \text{für } i = p+1, \ldots, n. \end{aligned}$$

Analog zum SQP-Verfahren wird nun ein Newton-Abstieg auf den KKT-Bedingungen und den Gleichheitsbedingungen dieses Problems durchgeführt. So ergibt sich das Gleichungssystem

$$\begin{pmatrix} \nabla_x^2 L(x,w,u) & 0 & -J_H^T(x) \\ 0 & U & W \\ J_{H_2}(x) & -I & 0 \\ J_{H_1}(x) & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta w \\ \Delta u \end{pmatrix} = \begin{pmatrix} -\nabla f(x) + J_H^T(x)u \\ \mu e - WUe \\ -H_2(x) + w \\ -H_1(x) \end{pmatrix}, \tag{9}$$

wobei zur Vereinfachung die Definitionen

$$H_1(x) := \begin{pmatrix} h_1(x) \\ \vdots \\ h_p(x) \end{pmatrix}, \qquad H_2(x) := \begin{pmatrix} h_{p+1}(x) \\ \vdots \\ h_n(x) \end{pmatrix}, \qquad H(x) := \begin{pmatrix} H_1(x) \\ H_2(x) \end{pmatrix} \tag{10}$$

eingesetzt wurden.

Anwendung dieses Verfahrens auf das zuvor erläuterte, quadratische Optimierungsproblem ergibt das Gleichungssystem

$$\begin{pmatrix} \nabla_x^2 L & J_{H_1}^T(x) & J_{H_2}^T(x) \\ J_{H_1}(x) & 0 & 0 \\ J_{H_2}(x) & 0 & -WU^{-1} \end{pmatrix} \begin{pmatrix} \Delta d \\ \Delta u \end{pmatrix} = - \begin{pmatrix} \nabla_x^2 L + \nabla f(x) + J_H(x)^T u \\ H_1(x) + J_{H_1}(x)d \\ H_2(x) + J_{H_2}(x)d + \sigma\mu U^{-1}e \end{pmatrix}. \tag{11}$$

Die Lösung dieses Systems ergibt nun ein Schritt $\lambda(\Delta d, \Delta u, \Delta w)$, dessen Anwendung auf den aktuellen Iterationspunkt einen Abstieg in Richtung Minimum bedeutet.

Zugrunde gelegt wurden bei dieser Herleitung die Vorraussetzungen $w \geq 0$ und $u \geq 0$. Die Einhaltung der Voraussetzungen wird zumeist über eine Liniensuche in der gefundenen Abstiegsrichtung und dementsprechend über eine passende Schrittweite $\lambda$ erzwungen.

Über iterative Anwendung des Innere-Punkte-Verfahrens findet sich nun eine Lösung $d*$ des quadratischen Problems, und somit eine Abstiegsrichtung $\Delta x = d*$ im ursprünglichen, nichtlinearen Problem.

Eine wesentliche Erweiterung, sowohl am SQP- als auch am IPM-Verfahren lässt sich durch die Verwendung sogenannter quasi-Newton-Updates erzielen. Bei diesen Verfahren wird statt der Hessematrix $\nabla_x^2 L$ in jedem Schritt eine Näherungsmatrix $B_k$ eingesetzt. Über die quasi-Newton-Forderung

$$B_{k+1}\Delta x = \Delta u \tag{12}$$

an die Matrizenfolge der $B_k$ wird dabei ein Newton-ähnliches Konvergenzverhalten gewährleistet. Dadurch kann eine unter Umständen teuere Auswertung der Hessematrix der Lagrangefunktion entfallen. Weitere Vorteile ergeben sich bei sehr grossen Problemstellungen durch Speichereinsparungsmöglichkeiten in der entsprechenden Approximation.

Die im Optimierer NLPIP ursprünglich eingesetzte BFGS-Update-Strategie ist ein sehr bekannter Vertreter der Quasi-Newton-Updates. Insbesondere beschreibt das BFGS-Update eine Aktualisierung für die Inverse $A_k = B_k^{-1}$ zur ursprünglichen Matrix $B_k$ nach folgendem Schema:

$$A_+ = A + \frac{(p - Aq)p^T + p(p - Aq)^T}{p^T q} - \frac{(p - Aq)^T q}{(p^T q)^2} pp^T \tag{13}$$

mit $A := B_k^{-1}$,
$A_+ := B_{k+1}^{-1}$,
$p := \Delta x$,
und $q := \Delta u$.

Einerseits erhält die auf diese Weise erzeugte Folge von $A$s eine anfängliche Symmetrie - beispielsweise durch Initialisierung von $A_0 := \nabla_x^2 L(x_0, u_0)$ gegeben. Andererseits lässt sich durch Approximation der Inversen von $B$ die spätere Invertierung innerhalb des Quasi-Newton-Schritts einsparen.

Da die BFGS-Update-Strategie nur mit Näherungen rechnet, ist insgesamt ein Geschwindigkeitsgewinn bei gleichzeitigem, leichten Genauigkeitsverlust zu erwarten.

Eine weitere Ergänzung am BFGS-Update-Verfahren in NLPIP ist der Einsatz einer Limited-Memory-Strategie als Optimierung für grosse Eingabeprobleme. Das bedeutet, das nur eine gegebene Anzahl an Vektoren verwendet wird, aus denen die Gesamtmatrix $A$ implizit erzeugt werden kann. Dadurch sinkt der Speicherplatzbedarf auf Kosten der Genauigkeit um eine Grössenordnung.

Inwiefern akkurate Hessematrizen mit anderen Optimierungen für dünnbesetzte Strukturen hier Vorteile bieten, wurde in den Testfällen empirisch untersucht.

## E.2 Implementierung

Bereits vor Beginn dieser Arbeit wurde an der Universität Bayreuth das nichtlineare Optimierungsprogramm NLPIP [BS08] [BS09] entwickelt. Dabei handelt es sich bei NLPIP um die FORTRAN-Implementierung eines sequentiellen, quadratischen Programmierungsverfahrens mit Innerer-Punkte-Methode zur Lösung der quadratischen Probleme. Innerhalb der Inneren-Punkte-Methode kommt ein BFGS-Update zum Quasi-Newton-Abstieg mit approximierter zweiter Ableitung der Lagrange-Funktion zum Einsatz. Grosse Eingabeprobleme wurden innerhalb der BFGS-Update-Strategie durch einen Limited-Memory-Ansatz abgefangen.

In der Implementierungsphase dieser Arbeit war das bestehende Programm zu erweitern, so dass alternativ zum BFGS-Update auch vom Nutzer zur Verfügung zu stellende, zweite Ableitungen eingebunden werden können. Eine parallele Erweiterung des Programmes diente der Verwendung dünnbesetzter Matrizenstrukturen zur Anbindung an den späteren Testfall der freien Materialoptimierung.

Im Folgenden wird ein grober Überblick über die Struktur von NLPIP geboten, ehe die durchgeführten Programmierarbeiten erläutert werden. Für einen tieferen Einblick sei auf das englischsprachige Kapitel 3 in dieser Arbeit verwiesen, ebenso auf die Benutzerdokumentation für NLPIP [BS08] und die Kommentare im zugehörigen Quellcode.

Der für den Benutzer sichtbare Haupteinstiegspunkt in den NLPIP-Optimierer wird durch die Prozedur NLPIPB geboten. Dabei übergibt der Benutzer beim ersten Aufruf einen entsprechenden Startpunkt, zugehörige Werte der Zielfunktion, die Werte der Grenzfunktionen, sowie alle benötigten Ableitungen.

Zur Kommunikation mit Benutzercode kommt im folgenden Programmablauf das sogenannte 'Umgekehrte Kommunikationsverfahren' zum Einsatz: Der Aufruf von NLPIPB kehrt nicht nur zum Benutzer zurück, wenn die Optimierung erfolgreich abgeschlossen oder fehlgeschlagen ist, sondern auch, wenn neue Eingaben verlangt werden. Beispielsweise muss der Benutzer Funktionswerte und Gradienten neu berechnen, wenn eine interne Iteration des Optimierers zu einem neuen Eingabepunkt geführt hat. Genauen Aufschluss über die erwartete Benutzerreaktion gibt dabei eine Zustandsvariable in einem Ausgabeparameter.

Innerhalb von NLPIPB gibt es drei verschiedene Abarbeitungsschichten, vertreten durch die Module NLPIP, QPSLV, und LINSLV.

In der äussersten Schicht - repräsentiert durch das Modul NLPIP - wird die iterative, quadratische Approximierung des Optimierungsproblems vorgenommen. Die Behandlung des quadratischen Problems wird an die Innere-Punkte-Methode im Modul QPSLV weitergereicht. Zurück erhält NLPIP einen Minimalpunkt im quadratischen Problem. Dieser Minimalpunkt wird als Basisvektor für die Abstiegsrichtung im ursprünglichen Problem eingesetzt. Dabei wird die konkrete Schrittweite über ein

Liniensucheverfahren bestimmt. Der Erfolg des Gesamtverfahrens wird nach Abstieg über ein toleranzgesteuertes Abbruchkriterium überprüft.

Durch das Modul QPSLV werden die von NLPIP approximierten quadratischen Probleme mit einem Innere-Punkte-Verfahren gelöst. Ebenso wird ein BFGS-Update auf der approximierten Hessematrix durchgeführt. Verschiedene Operationen, insbesondere im Zusammenhang mit den Gradienten der Nebenbedingungen oder Hessematrix der Lagrange-Funktion werden wiederum an das Module LINSLV weitergegeben.

Auf der letzten Abarbeitungsschicht steht das Modul LINSLV. Innerhalb von LINSLV sind dabei alle Operationen realisiert, die von Nebenbedingungs-Gradienten oder der Hessematrix der Lagrange-Funktion abhängen. Dadurch wird eine Entkopplung zwischen NLPIP und einer bestimmten Darstellung der unter Umständen grossen Eingabematrizen erzielt. NLPIP kennt dabei weder das genaue Format der Jacobi-Matrix noch das Format der Hesse-Matrix. Beide Formate müssen vom Benutzer festgelegt werden. Ebenso ist LINSLV folglich nicht Teil von NLPIP, sondern muss vom Benutzer geschrieben und mit dem eigentlichen Programm verknüpft werden. Lediglich die Schnittstelle zwischen LINSLV und NLPIP ist festgeschrieben, entsprechende Darstellungen finden sich in Abbildungen 3.2 und 3.3 im englischsprachigen Teil der Arbeit.

Hier greift der erste Teil der vorgenommenen Anpassungen am Quellcode: Aufgrund der zahlreichen Variablen und Nebenbedingungen, die bei der freien Materialoptimierung auftreten, durften nur dünnbesetze Matrizen in NLPIP verwendet werden. Weiterhin sollten die Hessematrizen der Lagrange-Funktion analytisch bestimmt und zur Konvergenzsteigerung verwendet werden. Dazu wurde ein neues LINSLV-Modul mit dem Namen HSSSLV entwickelt. Dieses Modul setzt statt den dichtgepackten Gradienten in der ursprünglichen Implementierung von LINSLV einen strukturierten Gradiententyp ein. Dieser Typ beinhaltet sowohl Gradienten der Nebenbedingungen, als auch die Hessematrix der Lagrangefunktion. Beide Matrizen werden dabei im vorgegebenen, dünnbesetzten Format des SCPF-Lösers zur freien Materialoptimierung im Speicher gehalten. Ebenso implementiert HSSSLV dünnbesetzte Versionen der von der LINSLV-Schnittstelle geforderten Operationen. Schliesslich lässt sich durch einen Parameter das Verhalten von HSSSLV auf einen BFGS-konformen Operationsmodus umstellen, die vom Benutzer übergebene Hessematrix wird in diesem Fall ignoriert. Der Quellcode zum Modul HSSSLV findet sich auf dem beiliegenden Datenträger, sowie in reduzierter Form in Anhang B.

Der zweite Teil an Programmänderungen betrifft die NLPIP- und QPSLV-Module direkt. Zwar ist die Behandlung der Gradienten und Hessematrizen aus NLPIP gelöst, die BFGS-Aktualisierung ist jedoch fest integriert. Daher musste ein zusätzlicher Parameter eingefügt werden, mit dem das BFGS-Update bei Bedarf abgeschaltet werden kann. Entsprechend dem Parameter mussten schliesslich über den Quelltext verteilte Limited-Memory-Berechnungen deaktiviert werden.

## E.3 Anwendung in der freien Materialoptimierung

Ein wesentliches Anwendungsbeispiel für die Konvergenzsteigerung durch analytisch berechnete, zweite Ableitungen liegt auf dem Gebiet der freien Materialoptimierung.

Dieser Anwendungsfall stellt die Grundlage späterer, praktischer Verfahrenstests, und wird daher im Folgenden kurz beschrieben [RS10].

Ziel der freien Materialoptimierung ist die Bestimmung einer Mindest-Materialverteilung über ein Bauteil in Abhängigkeit einer gegebenen Lastverteilung, so dass eine maximale Auslenkung nicht überschritten wird. Belastete Stellen müssen dabei in einem dickeren Material ausgeführt werden, an unbelasteten Punkten kann Material eingespart werden.

Für traditionelle isotrope Materialien, so zum Beispiel Hochleistungsstahl oder Aluminium, existieren bereits zahlreiche erfolgreiche Lösungsstrategien zur Materialoptimierung.

Im Flugzeugbau werden jedoch zunehmend anisotrope Materialen, beispielsweise Kohlenfaserverbundswerkstoffe, eingesetzt. Für diese Materialien und die resultierenden grossen Optimierungsprobleme sind die bestehenden Verfahren nur eingeschränkt nutzbar. Im Rahmen des von der EU finanzierten Projektes Plato-N wurde daher an der Universität Bayreuth die freie Materialoptimierung speziell für anisotrope Materialien untersucht.

In Folge dieser Forschungen wurde ein neues Verfahren zur freien Materialoptimierung auf Basis der sequentiellen, konvexen Programierung entwickelt [CZ04] [SE09]. Bei diesem Verfahren wird das ursprüngliche Optimierungsproblem iterativ durch nichtlineare, konvexe Probleme approximiert. Diese wiederum werden über NLPIP mit dem kombinierten Verfahren aus sequentieller, quadratischer Programmierung und der Innere-Punkte-Methode aus dem vorhergehenden Abschnitt gelöst.

## E.4 Tests

Nach erfolgreicher Implementierung wurden das ursprüngliche Optimierungsverfahren mit BFGS-Approximation der zweiten Ableitungen und das Optimierungsverfahren mit analytisch bestimmten zweiten Ableitungen einander gegenübergestellt. Dabei wurde ein Vergleich im Hinblick auf Konvergenzgeschwindigkeit, Ausführungszeit, und Qualität der Ergebniswerte vorgenommen.

Als Testszenarien dienten einerseits eine Sammlung an nichtlinearen Standardproblemen [KS09], andererseits eine Menge an Spielproblemen aus dem zuvor vorgestellten Anwendungsgebiet der freien Materialoptimierung [SE09]. Eine Auflistung aller Testparameter findet sich im englischsprachigen Kapitel 4. Für die Standard-Testverfahren ist insbesondere der Abschnitt 4.1.1 relevant. Die Beispiele aus der freien Materialoptimierung sind in Abschnitt 4.1.2 zusammengefasst.

In der Durchführung wurde der Optimierungsprozess für jedes Testszenario und beide Optimierungsverfahren protokolliert. Entsprechende Protokolle finden sich auf dem mitgelieferten Datenträger. Für einen Überblick über die Testergebnisse sei zudem auf Darstellung 4.2.3 im englischsprachigen Teil der Arbeit verwiesen.

Als Fazit ergab sich bei den Tests, dass die Anwendung analytischer zweiter Ableitungen die Konvergenzgeschwindigkeit und somit die Ausführungszeit deutlich verbessert. Insbesondere für die freie Materialoptimierung erfüllt die Erweiterung um analytische zweite Ableitungen also die bestehenden Erwartungen.

## F Software Tools

The following software tools have been used in the creation of this thesis document and the accompanying programs:

- **Visual Studio 2008**
  `http://www.microsoft.com/germany/visualstudio`
- **Intel Visual Fortran 11**
  `http://software.intel.com/en-us/intel-compilers`
- **GIMP 2.6**
  `http://www.gimp.org`
- **OpenOffice 3**
  `http://www.openoffice.org`
- **LaTeX 2$\epsilon$**
  `http://www.latex-project.org`
- **TeXnicCenter**
  `http://www.texniccenter.org`

Thank you for your great work!

## G Erklärung zur Authentizität

Hiermit bestätige ich, Tobias Werner, dass ich diese Diplomarbeit inklusive zugehöriger Programmquellen selbständig angefertigt habe. Lediglich die unter Anhang A genannten Quellen wurden bei Erstellung der Arbeit einbezogen. Entsprechende Verwendungsstellen im laufenden Text sind ausdrücklich mit einer Referenz versehen.

Unterschrift: _____

(Tobias Werner)