Investigation of Different Solvers for Radiotherapy Treatment Planning Problems

Yvonne Ruckstuhl
Investigation of Different Solvers for Radiotherapy Treatment Planning Problems

MASTER OF SCIENCE THESIS

For the degree of Master of Science in Applied Mathematics at Delft University of Technology

Yvonne Ruckstuhl

November 17, 2014

Faculty of Electrical Engineering, Mathematics and Computer Science · Delft University of Technology
The work in this thesis was supported by the Department of Radiation Oncology, Erasmus Medical Center - Cancer Institute, Rotterdam, The Netherlands. Their cooperation is hereby gratefully acknowledged.
Delft University of Technology
Department of
Applied Mathematics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled

Investigation of Different Solvers for Radiotherapy Treatment Planning Problems

by

Yvonne Ruckstuhl

in partial fulfillment of the requirements for the degree of

Master of Science Applied Mathematics

Dated: November 17, 2014

Supervisors:  

Dr. ir. M. Keijzer

Dr. ir. S. Breedveld

Readers:  

Prof. dr. ir. A.W. Heemink

Dr. ir. D.J.P. Lahaye
Radiotherapy treatment planning involves solving inequality constrained minimization problems. The currently used interior point solver performs well, but is considered relatively slow. In this thesis we investigate two different solvers based on the logarithmic barrier method and Sequential Quadratic Programming (SQP) respectively. We argue that the behaviour of the logarithmic barrier solver is uncertain, thereby making it generally unreliable in this context. In addition we substantiate that the performance of the SQP solver is solid, but lacks efficiency in computing the minimizers of its related quadratic sub-problems. We conclude that without serious improvements, none of the solvers investigated are faster than the currently used interior point optimizer.
Contents

1 Introduction ................................................................. 1
   1-1 Treatment Plan ..................................................... 1
       1-1-1 Optimizing the Number of Beams and their Angles ............ 2
       1-1-2 Optimizing the Beam Intensity Profile ........................... 2
   1-2 Goal of this Research Project .................................... 7
   1-3 Assumptions and Notation ........................................ 7
   1-4 Outline of this Thesis ............................................. 8

2 Optimization .............................................................. 9
   2-1 KKT Conditions ..................................................... 9
       2-1-1 Equality Constrained Case .................................... 9
       2-1-2 General Case ................................................ 10
   2-2 Minimization Problem Versus KKT Conditions ....................... 12

3 Motivation to Investigate SQP and the Barrier Method ................. 13
   3-1 Interior Point Methods .......................................... 13
   3-2 Logarithmic Barrier Method ...................................... 15
   3-3 Sequential Quadratic Programming ................................ 16

4 Test Cases .................................................................. 17
   4-1 Structures of the Test Cases ..................................... 17
   4-2 Results for the Interior Point Method ........................... 19
## 5 Logarithmic Barrier Method

- **5-1 Central Path**
- **5-2 Computing a Central Point**
- **5-3 Algorithm**
- **5-4 Implementation**
  - **5-4-1 Initial Values**
  - **5-4-2 Subproblems**
- **5-5 Newton-Barrier Method**
  - **5-5-1 Tolerance Levels \( \epsilon \) and \( \tilde{\epsilon} \)**
  - **5-5-2 Initial Penalty Parameter \( t_1 \)**
  - **5-5-3 Updating Parameter \( \mu \)**
  - **5-5-4 Summary**
- **5-6 BFGS-Barrier Method**
- **5-7 Results**
- **5-8 Possible Improvements**
- **5-9 Analysis**
- **5-10 Conclusion**

## 6 Sequential Quadratic Programming

- **6-1 Theory**
  - **6-1-1 Quadratic Subproblems**
  - **6-1-2 Merit Function**
  - **6-1-3 Algorithm**
  - **6-1-4 Convergence Properties**
- **6-2 Implementation**
  - **6-2-1 Initial Values**
  - **6-2-2 Improving the Runtime**
  - **6-2-3 Numerical Problems**
- **6-3 Adjusting the Algorithm based on Results**
  - **6-3-1 Exact Hessian Versus Approximate Hessian**
  - **6-3-2 Merit Function**
- **6-4 Results**
- **6-5 Analysis**
  - **6-5-1 Outer Iterations**
  - **6-5-2 Inner Iterations**
- **6-6 Possible Improvements**
  - **6-6-1 Number of Inner Iterations**
  - **6-6-2 Implementation of Active Set Method**
- **6-7 Best Case Scenario**
- **6-8 Recommendations**
- **6-9 Conclusion**
Contents

7 Penalty Methods 75
   7-1 Equality Constrained Case ........................................ 75
   7-2 Inequality Constrained Case ..................................... 78

8 Conclusions and Recommendations 81

A Unconstrained Optimization 83
   A-1 Newton’s Method for Optimization ............................... 83
   A-2 Quasi-Newton Methods ........................................... 84
      A-2-1 BFGS ............................................................ 85
      A-2-2 Limited Memory BFGS Method ............................... 86

B Active Set Method for QP 89
   B-1 Search Direction .................................................. 90
   B-2 Stepsize .................................................................. 91
   B-3 Practical Algorithm ................................................ 91

C Results of the Logarithmic Barrier Method 93

   Bibliography 99
Radiotherapy is one of the main methods to treat cancer. It uses special equipment to direct high-energy photon beams to the tumor, causing destruction of cancer cells. To prevent damage to healthy tissue, it is important to keep the amount of absorbed dose\(^1\) in normal cells to a minimum. Since the radiation scatters inside the body, the beams inevitably harm healthy tissue in the process of reaching the tumor. The goal is therefore to deliver a sufficient amount of dose to the tumor, while protecting healthy tissue as much as possible. Available degrees of freedom to influence the distribution of the dose include the number of beams, their angles, and their intensities, which are each represented by a non-uniform 2D fluence profile. Optimizing these components results in a treatment plan. Each case is unique, therefore a different treatment plan is made for every patient.

1-1 Treatment Plan

One way to make a treatment plan is to simply try all possible combinations and choose the best one. Unfortunately this will take far too long, as the number of possible combinations is practically infinite. Therefore, a different approach is used at the Erasmus MC. The problem is divided into two parts: An optimization of the number of beams and their angles, and an optimization of the beam intensity profiles.

\(^1\)Absorbed dose is measured in gray (Gy) and quantifies the radiation energy (Joule) absorbed in a unit mass (kilogram).
1-1-1 Optimizing the Number of Beams and their Angles

This part is based on a sequential selection: first, only one beam is considered. Its angle is varied according to some chosen mesh and for each angle the beam profiles are optimized. Subsequently the beam is fixed at the angle which resulted into the best treatment plan. Then a second beam is added. Now, the best treatment plan is sought by varying the angle of the second beam. If adding the second beam has led to an improvement, a third beam is added, etc. This is repeated until no progress was made by adding a beam, or until a maximum number of beams is selected.

This process basically consists of a sequence of optimization problems with fixed number of beams and angles, leaving the intensity profile of each beam as the only variables. In the next subsection we explain how this is optimized.

1-1-2 Optimizing the Beam Intensity Profile

To translate the optimization problem into a practical form, a beam is discretized into \( n \) beamlets, creating the fluence matrix \( X \). Each element of \( X \) represents the mean intensity of the beamlet it corresponds to (see Figure 1-1). The patient’s body is discretized as well, into \( N \) volume elements called voxels. We introduce the discretized dose distribution \( D \) as a 3D matrix whose elements represent the dose delivered to each voxel respectively. The relation between the fluence and the dose is linear, so it can be expressed as \( D = \tilde{A}X \), where \( \tilde{A} \) is some matrix of appropriate size. For mathematical simplicity we rearrange the elements of this equation to obtain

\[
d = Ax
\]

where \( d \in \mathbb{R}^N_+ \) and \( x \in \mathbb{R}^n_+ \) are vectors representing the dose distribution and fluence respectively. The matrix \( A \in \mathbb{R}^{N \times n} \) is called the dose deposition matrix.

Now we concretize the goal of the optimization. First, the voxels are grouped into different volumes according to clinical characteristics. The volume that includes the tumor is called the PTV (Planning Target Volume). Other volumes are called OARs (Organs At Risk). We define \( A_l, d_l \) and \( N_l \) as the dose deposition matrix, the dose distribution and the number of voxels corresponding to the \( l^{th} \) volume respectively, such that \( d_l = A_l x \in \mathbb{R}^{N_l}_+ \).

The dose in each volume is assigned certain objectives, which are formulated as cost functions. We discuss three different types of cost functions.
Figure 1-1: Example of a continuous fluence profile, where the discrete fluence matrix $X$ has been converted to a continuous function by interpolation [1].

1. **Linear**: $f(x) = \| A_i x \|_\infty$ or $f(x) = \frac{1}{N_l} \| A_i x \|_1$
   A linear cost function is used when the goal is to minimize the maximum dose or the mean dose in a volume. Sometimes it is also used to maximize the minimum dose or mean dose of the PTV.

2. **EUD**: $f(x) = \left( \frac{1}{N_l} \sum_{k=1}^{N_l} [(A_i x)_k]^p \right)^{1/p}$
   Note that an EUD-type cost function resembles the $L_p$ norm. We see that it equals the mean dose when $p = 1$ and the maximum dose when $p = \infty$. Intuitively it follows that the parameter $p \geq 1$ defines a trade-off between the mean dose and the maximum dose. For $p \leq -1$ the value of the EUD lies between the mean and the minimum dose [2]. So an EUD with $p \geq 1$ is used for OARs, whereas for $p \leq -1$ it is suitable for PTVs.

3. **LTCP**: $f(x) = \frac{1}{N_l} \sum_{k=1}^{N_l} e^{-\alpha[(A_i x)_k - p]}$
   LTCP-type cost functions are meant for PTVs only. They penalize doses lower than the prescribed dose $p$ exponentially. For doses higher than $p$ the LTCP slowly approaches zero. So the LTCP strongly stimulates meeting the prescribed dose, while only little can be gained by "overdosing" the tumor.
4. **Quadratic**: \( f(x) = x^T G x + b^T x + c \)

Due to discretization, the fluence map can become highly discontinuous, which is not realizable in practice. A quadratic cost function is used to smoothen the fluence such that clinical feasibility is obtained. The exact characteristics of the smoothing are incorporated into the matrix \( G \) and vectors \( b \) and \( c \). Linear smoothing functions also exist.

On some volumes we impose *hard constraints*, which may not be violated. These constraints are used for example if the dose delivered to the PTV needs to be higher than a certain level to be effective, or when an OAR cannot withstand more than some critical dose. For each volume the cost functions and constraints are determined based on clinical characteristics. We take an iterative approach to find a good tradeoff between the minimization of the different objectives. This approach is described in the remainder of this section.

The objectives, formulated as cost functions \( f_j(x) \), are assigned a goal \( g_j \). The goal is an indication of the value we wish the objective to attain. The objectives are ranked according to a priority system, which is determined clinically. This information is summarized in a *wish-list*. An example of a wish list for a cervical cancer patient is given in table 1-1.

<table>
<thead>
<tr>
<th>Hard Constraints</th>
<th>Type</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTV</td>
<td>maximum</td>
<td>49.22 Gy</td>
</tr>
<tr>
<td>PTV Shell 1.5 cm</td>
<td>maximum</td>
<td>41.4 Gy</td>
</tr>
<tr>
<td>PTV Shell 4 cm</td>
<td>maximum</td>
<td>36.8 Gy</td>
</tr>
<tr>
<td>Unspecified Tissue</td>
<td>maximum</td>
<td>49.22 Gy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objectives</th>
<th>Rank</th>
<th>Volume</th>
<th>Type</th>
<th>Goal</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>PTV</td>
<td>minimize LTCP</td>
<td>1</td>
<td>( p = 46, \alpha = 0.75 )</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Skin Ring 3 cm</td>
<td>minimize maximum</td>
<td>33.2 Gy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Bowel</td>
<td>minimize mean</td>
<td>40 Gy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Sigmoid</td>
<td>minimize mean</td>
<td>40 Gy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Rectum</td>
<td>minimize mean</td>
<td>40 Gy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Bladder</td>
<td>minimize mean</td>
<td>40 Gy</td>
<td></td>
</tr>
</tbody>
</table>

Table 1-1: Wish-list for cervical cancer case.
Let us assume that a given wish-list has $m_1$ hard constraints $c_i(x)$ and $m_2$ objectives, each corresponding to a cost function $f_j(x)$, where $j$ indicates the rank. Optimizing the elements of the wish-list is done in two phases. We use $\ast$ and $\ast\ast$ to distinguish between the first and the second phase respectively.

In the first phase, the objectives are optimized one by one. The cost function corresponding to the objective with the highest rank, $f_1(x)$, is optimized first, creating the following minimization problem:

$$\min_x f_1(x) \quad \text{subject to } c_i(x) \leq 0, \quad i \in \{1, 2, ..., m_1\}$$

(1-2)

When the solution $x_1^\ast$ of this problem is found, the objective is converted into a constraint, such that $f_1(x) \leq \delta_1^\ast$, where $\delta_1^\ast$ assumes the value of goal $g_1$ provided that this target has been reached with some margin $\epsilon_1^\ast$, and is a little over the attained value otherwise. Formally $\delta_j^\ast$ is defined as follows:

$$\delta_j^\ast = \begin{cases} g_j \\ f_j(x_j^\ast) + \epsilon_j^\ast \end{cases}$$

(1-3)

The purpose of adding parameter $\delta_j^\ast$ is to flexibilize the process. Indeed, we do not want to impose constraints that are unnecessarily harsh, as this will limit the margins for objectives with a lower rank. Parameter $\epsilon_j^\ast$ serves merely to guarantee at least some flexibility so that the algorithm does not get stuck before finalizing the first phase. It is usually chosen as 3% of $f_j(x_j^\ast)$.

After the first objective has been converted into a constraint, the next cost function $f_2(x)$ is minimized subject to the hard constraints and the current constrained objective $f_1(x) \leq \delta_1^\ast$, etc. In general, the following minimization problem is solved at the $k^{th}$ iteration:

$$\min_x f_k(x) \quad \text{subject to } c_i(x) \leq 0, \quad i \in \{1, 2, ..., m_1\} \\
\quad \quad \quad \quad f_j(x) \leq \delta_j^\ast, \quad j \in \{1, 2, ..., k - 1\}$$

(1-4)

When all objectives have been converted to constraints, phase two is started.

In the second phase the constrained objectives are again optimized one by one using the same priority system. The key difference is that the objectives are constrained regardless
of their goal values: $\delta_{j}^{**} = f_{j}(x_{j}^{**}) + \epsilon_{j}^{**}$. In general, the following problem is solved at the $k^{th}$ iteration:

$$
\begin{align*}
& \min_{x} f_{k}(x) \\
& \text{subject to } c_{i}(x) \leq 0, \quad i \in \{1, 2, ..., m_{1}\} \\
& \quad f_{j}(x) \leq \delta_{j}^{**}, \quad j \in \{1, 2, ..., k - 1\} \\
& \quad f_{j}(x) \leq \delta_{j}^{*}, \quad j \in \{k + 1, k + 2, ..., m_{2}\}
\end{align*}
$$

(1-5)

Each cost function $f_{k}$ that already has been optimized to the fullest in the first phase, so $k \in \{j : \delta_{j}^{*} = \delta_{j}^{**}\}$, does not have to be optimized again, because the outcome of problem (1-5) yields $x_{k}^{**} = x_{k-1}^{**}$ and $\delta_{k}^{**} = \delta_{k}^{*}$.

The above described method to formulate a treatment plan is called iCycle. For more details on iCycle we refer to [3]. Figure 1-2 illustrates the beam angles and dose distribution of a possible treatment plan generated by iCycle for the cervical cancer case corresponding to wish-list 1-1 [3].
1-2 Goal of this Research Project

Creating a treatment plan as described above can be time consuming. The algorithm basically consists of solving a sequence of problems of the following general form:

$$\min_x f(x)$$
$$\text{subject to } c_i(x) \leq 0, \quad i \in \{1, 2, ..., m\}$$

(1-6)

Therefore, its runtime is directly related to the efficiency with which problem (1-6) can be solved. This is currently done using an interior point method. Even though interior point methods have been proven successful in this context, there might be methods that would perform even better. The goal of this thesis is to investigate different methods to solve problem (1-6) and compare their practical performance to that of the interior point method. In particular we investigate the logarithmic barrier method and Sequential Quadratic Programming (SQP). This will lead to a recommendation on whether the optimization method that is currently used should be replaced.

1-3 Assumptions and Notation

A minimization problem of type (1-6) is called convex if both the objective function and the constraints are convex [4]. Since all types of cost functions described in section 1-1 are convex, we assume throughout this thesis that problem (1-6) is convex and that the solution $x^*$ is unique.

Furthermore, we assume that problem (1-6) is feasible, i.e. there is an $x$ which satisfies $c_i(x) \leq 0, i \in \{1, 2, ..., m\}$. Unless stated otherwise, we also assume that there are no redundant constraints. For linear constraints this translates into their gradients being linearly independent. Finally, we consider the constraints satisfied if for all $i \in \{1, 2, ..., m\}$ holds $c_i(x) < 10^{-3}$.

Throughout this thesis we use $x^*$ to denote the unique solution of problem (1-6), unless stated otherwise.
1-4 Outline of this Thesis

We start with an introduction to convex optimization in chapter 2. Here we show that problem (1-6) can be reformulated into a so-called KKT system. In chapter 3 we motivate our choice to investigate the logarithmic barrier method and Sequential Quadratic Programming (SQP). To compare the performance of different solvers, 4 test cases are used. These are described in chapter 4. Chapter 5 and 6 include a full description, analysis and conclusion of the performance of the logarithmic barrier method and SQP respectively. In chapter 7 we briefly discuss the potential of penalty methods, which are believed to perform well when most constraints of problem (1-6) are linear [5]. In chapter 8 we summarize the important conclusions of this research, along with concrete recommendations.
This chapter is meant to provide the reader with some general insight in (convex) optimization. First we derive the KKT conditions, which summarize the requirements of a solution. Then we elaborate on the relation between the KKT conditions and the minimization problem.

2-1 KKT Conditions

To solve minimization problems we need to know how to identify a solution. To that end we derive which conditions a solution needs to meet. We call these the KKT (Karush–Kuhn–Tucker) conditions. First we derive the KKT conditions for equality constrained minimization problems. We then extend these conditions to the general case.

2-1-1 Equality Constrained Case

In this section we formulate the KKT conditions of the following equality constrained problem:

\[
\begin{align*}
\min_{x} & \quad f(x) \\
\text{subject to} & \quad h_j(x) = 0, \quad j \in \{1, 2, \ldots, \tilde{m}\}
\end{align*}
\]

In this section, \( x^* \) denotes the solution of problem (2-1). We are looking for characteristics of \( x^* \). Let us assume there is only one constraint, \( \tilde{m} = 1 \). So the goal is to find an \( x^* \) for which \( f(x) \) is minimized along \( h(x) = 0 \).
Suppose $x_0$ satisfies $h(x_0) = 0$. We want to establish whether the value of the objective function can be decreased by walking along the constraint in the neighborhood of $x_0$. If so, we continue to walk along the constraint until we find a point $x^*$ for which $f(x)$ can no longer be decreased along $h(x) = 0$. Figure 2-1 illustrates that at $x^*$ the respective gradients of the objective function and the constraint are parallel. In other words, for the solution $x^*$ the following equations hold

\[
\begin{align*}
\nabla f(x^*) & = -\tilde{\lambda} \nabla h(x^*) \\
h(x^*) & = 0
\end{align*}
\]

for some $\tilde{\lambda} \in \mathbb{R}$, which we call a Lagrange multiplier.

When there are more constraints, the righthand side of condition (2-2a) translates into a linear combination of the gradients of the constraints [5], yielding

\[
\begin{align*}
\nabla f(x^*) & = -\sum_{j=1}^{\tilde{m}} \tilde{\lambda}_j h_j(x^*) \\
h_j(x^*) & = 0, \quad j \in \{1, 2, \ldots, \tilde{m}\}
\end{align*}
\]

Note that the Lagrange multiplier is now a vector, $\lambda \in \mathbb{R}^{\tilde{m}}$. To incorporate these conditions into one equation we introduce the Lagrangian function, which is defined as follows:

\[
\mathcal{L}(x, \lambda) = f(x) + \sum_{j=1}^{\tilde{m}} \tilde{\lambda}_j h_j(x)
\]

The KKT conditions for problem (2-1) are then

\[
\nabla_{x,\lambda} \mathcal{L}(x, \tilde{\lambda}) = 0.
\]

### 2-1-2 General Case

In this section we formulate the KKT conditions for the general problem

\[
\begin{align*}
\min_x \quad & f(x) \\
\text{subject to} \quad & h_j(x) = 0, \quad j \in \{1, 2, \ldots, \tilde{m}\} \\
& c_i(x) \leq 0, \quad i \in \{1, 2, \ldots, m\}
\end{align*}
\]
In this section, \( x^* \) denotes the solution of problem (2-6). We denote the Lagrange multiplier of the inequality constraints by \( \lambda \) and of the equality constraints by \( \tilde{\lambda} \). We want to distinguish between active and inactive inequality constraints. An inequality constraint is called active when \( c_i(x) = 0 \) and inactive when \( c_i(x) < 0 \). The set of indices corresponding to the inequality constraints that are active at \( x^* \), is called the Active set, \( \mathcal{A} \).

The Lagrangian corresponding to problem (2-6) is given by

\[
\mathcal{L}(x, \lambda, \tilde{\lambda}) = f(x) + \sum_{j=1}^{\tilde{m}} \tilde{\lambda}_j h_j(x) + \sum_{i=1}^{m} \lambda_i c_i(x)
\]  

(2-7)

Let \( \lambda^* \) and \( \tilde{\lambda}^* \) denote the Lagrange multipliers of the inequality and equality constraints at the solution \( x^* \) respectively. From the previous subsection we know that

\[
\nabla_{x, \lambda_i \in \mathcal{A}, \tilde{\lambda}} \mathcal{L}(x^*, \lambda^*, \tilde{\lambda}^*) = 0
\]

(2-8)

In contrast to the equality constraint case, the value of the Lagrange multiplier \( \lambda \) is of some importance, in particular its sign. In order to intuitively understand the relevance of the sign of the Lagrange multiplier, we consider the case where \( m = 1 \) and \( \tilde{m} = 0 \).

Suppose that the inequality constraint is active at some \( x \) with corresponding Lagrange multiplier \( \lambda < 0 \), and that equation (2-8) holds, so \( \nabla f(x) = -\lambda \nabla c(x) \) and \( c(x) = 0 \). Then, the gradient of the objective function points in same direction as the gradient of the constraint, meaning that decreasing the constraint (i.e. moving towards the feasible region) leads to decrease in the objective function. Ergo, even though (2-8) holds, \( x \) is not the solution because of the sign of the Lagrange multiplier.

In general we conclude that the Lagrange multipliers corresponding to the active set are positive [5]. We therefore add another set of conditions:

\[
\lambda^*_i \geq 0, \quad i \in \mathcal{A}
\]

(2-9)

The necessary conditions (2-8) and (2-9) form the KKT conditions of problem (2-6) and can be summarized in the following manner;

\[
\nabla_{x, \tilde{\lambda}} \mathcal{L}(x^*, \lambda^*, \tilde{\lambda}^*) = 0
\]

(2-10a)

\[
c_i(x^*) \leq 0, \quad i \in \{1, 2, ..., m\}
\]

(2-10b)

\[
\lambda^*_i \geq 0, \quad i \in \{1, 2, ..., m\}
\]

(2-10c)

\[
\lambda^*_i c_i(x^*) = 0, \quad i \in \{1, 2, ..., m\}
\]

(2-10d)

Note that the formulation of these conditions implies that the Lagrange multipliers corresponding to the inactive constraints are zero.
2-2 Minimization Problem Versus KKT Conditions

The KKT conditions are not in general sufficient conditions for a solution. They merely indicate the necessary conditions for a solution. Fortunately, in the convex case the necessary conditions are also sufficient conditions. In fact, all $x$ that satisfy the KKT conditions

$$\nabla_{x,\lambda} \mathcal{L}(x, \lambda, \bar{\lambda}) = 0$$

(2-11a)

$$c_i(x) \leq 0, \quad i \in \{1, 2, ..., m\}$$

(2-11b)

$$\lambda_i \geq 0, \quad i \in \{1, 2, ..., m\}$$

(2-11c)

$$\lambda_i c_i(x) = 0, \quad i \in \{1, 2, ..., m\}$$

(2-11d)

are global solutions of problem (2-6) [4]. If the objective function is strictly convex, there is only one $x^*$ that satisfies (2-11), which is the unique solution of problem (2-6). So when dealing with convex problems, solving problem (2-6) is equivalent to solving system (2-11).

Some optimization methods are based on iteratively solving the minimization problem until its corresponding KKT conditions are satisfied, while others make use of algorithms which are designed to directly solve system (2-11). Both approaches will lead to the same solution. In fact, in most equality constrained cases they even result in exactly the same algorithm.

For a simple example consider Newton’s method to minimize a function $g(x)$. Since there are no constraints, the corresponding KKT conditions are simply $\nabla g(x) = 0$. The first approach would lead to minimizing a series of second order Taylor approximations of $g(x)$, whereas the second approach would lead to equating first order Taylor approximations of $\nabla g(x)$ to zero. Both approaches result in the same Newton step $p = -[\nabla^2 g(\tilde{x})]^{-1} \nabla g(\tilde{x})$. For more detail on Newton’s method see appendix A-1.
Motivation to Investigate SQP and the Barrier Method

The main difficulty of solving problem (1-6) is dealing with the inequalities. Indeed, when only equality constraints are involved, the KKT system (2-11) narrows down to

$$\nabla x_\lambda \mathcal{L}(x, \lambda) = 0$$ (3-1)

which can be solved using Newton’s method or a variant thereof. Most methods, including interior point methods, therefore reformulate problem (1-6) to implicitly eliminate all inequality constraints. In this chapter we cover the fundamentals of interior point methods and point out their weakness in iCycle context. This will lead to a motivation for the choice to investigate the Logarithmic Barrier Method and Sequential Quadratic Programming.

3-1 Interior Point Methods

Interior point methods use slack variables $s_i$ to convert inequality constraints into equality constraints. Problem (1-6) becomes

$$\begin{align*}
\min_x & \quad f(x) \\
\text{subject to} & \quad c_i(x) + s_i = 0, \quad i \in \{1, 2, ..., m\} \\
& \quad s_i \geq 0, \quad i \in \{1, 2, ..., m\}
\end{align*}$$ (3-2)
This formulation is still not inequality constraint free, because the slacks need to be positive. Instead of treating these conditions as inequality constraints, they are incorporated into the objective function, creating the following equality constrained problem:

\[
\begin{align*}
\min_{x,s} & \quad f(x) - \mu \sum_{i=1}^{m} \log s_i \\
\text{subject to} & \quad c_i(x) + s_i = 0, \quad i \in \{1, 2, \ldots, m\}
\end{align*}
\] (3-3)

where \(\mu > 0\) is a penalty parameter. The logarithm implicitly prevents the slacks from coming too close to zero since \((-\log s) \to \infty\) as \(s \downarrow 0\). The smaller \(\mu > 0\), the closer the solution of problem (3-3) is to \(x^*\). In fact, for \(\mu \downarrow 0\) problem (3-3) is equivalent to problem (1-6) [6]. Therefore, interior methods iteratively solve problem (3-3), where the parameter \(\mu\) is increased at each iteration until some predetermined accuracy has been reached. The search direction \((\Delta x, \Delta s, \Delta \lambda)^T\) is found by computing the Newton step corresponding to the KKT system of problem (3-3).

The Lagrangian of problem (3-3) is

\[
\mathcal{L}(x, s, \lambda) = f(x) - \mu \sum_{i=1}^{m} \log s_i + \sum_{i=1}^{m} \lambda_i (c_i(x) + s_i)
\] (3-4)

So the corresponding KKT system is

\[
\begin{align*}
\nabla f(x) + A^T \lambda &= 0 & \text{(3-5a)} \\
S\Lambda \mathbf{1} - \mu \mathbf{1} &= 0 & \text{(3-5b)} \\
c(x) + s &= 0 & \text{(3-5c)}
\end{align*}
\]

where \(S\) and \(\Lambda\) are diagonal matrices with \(S_{ii} = s_i\), \(\Lambda_{ii} = \lambda_i\), \(s = [s_1, s_2, \ldots, s_m]^T\), \(c(x) = [c_1(x), c_2(x), \ldots, c_m(x)]^T\), and \(A^T = [\nabla c_1(x), \nabla c_2(x), \ldots, \nabla c_m(x)]\). Note that the second equation has been multiplied by \(S\). The Newton step \((\Delta x, \Delta s, \Delta \lambda)^T\) is found by solving

\[
\begin{pmatrix}
H & 0 & A^T \\
0 & \Lambda & S \\
A & I & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta s \\
\Delta \lambda
\end{pmatrix}
= - \begin{pmatrix}
\nabla f(x) + A^T \lambda \\
SAe - \mu e \\
c + s
\end{pmatrix}
\] (3-6)

where \(H = \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 c_i(x)\). See appendix A-1 for the derivation of the Newton step. For concrete interior-point-type algorithms, where updating techniques for the parameter \(\mu\) are specified, we refer to [5].

The main weakness of interior point methods applied to problem (1-6) is that system (3-6) is very large. The number of unknowns can be reduced significantly by clever elimination.
of variables, but unfortunately this is also very expensive because of the size of the non-sparse matrix $A$.

The interior point method that is implemented in iCycle has been thoroughly optimized. The only remaining bottleneck lies in computing $A^T S^{-1} \Lambda A$, which is needed to eliminate variables. We therefore seek methods that do not require similar operations.

The KKT conditions (3-5) which led to system (3-6) are a direct result from the chosen reformulation in which slack variables were introduced to convert inequalities to equalities. It is therefore natural to wonder if there are methods which use a different reformulation. Preferably one that would lead to a smaller KKT system. An obvious choice is the logarithmic barrier method.

### 3-2 Logarithmic Barrier Method

The logarithmic barrier method, or simply barrier method, is very similar to interior point methods, but, in a way, less cumbersome. Instead of using slacks to create equality constraints and then incorporating the resulting slack inequalities ($s \geq 0$) into the objective, the barrier method directly incorporates the inequality constraints into the objective. So problem (1-6) is approximated as follows:

$$
\min_x f(x) - \mu \sum_{i=1}^m \log(-c_i) \tag{3-7}
$$

Let us define the barrier function $g(x) := f(x) - \mu \sum_{i=1}^m \log(-c_i)$. The corresponding KKT system is $\nabla g(x) = 0$, which is considerably smaller than system (3-5). However, computing the Hessian of the barrier function $\nabla^2 g(x)$ is expensive. Note that substituting $s_i = -c_i(x)$ into (3-3) results into problem (3-7), so in essence interior point methods and the barrier method are the same. More to the point, computation of $\nabla^2 g(x)$ also requires computation of $A^T D A$, where $D$ is a diagonal matrix (see chapter 5). It therefore seems that nothing can be gained. However, the barrier method solves a series of unconstrained minimization problems of type (3-7). The advantage lies in the existence of methods for unconstrained optimization, which circumvent computation of the Hessian $\nabla^2 g(x)$.

In this thesis we use the BFGS and Limited BFGS method (see appendix A-2) to solve problem (3-7). These methods use an approximation of the Hessian instead of the exact Hessian, thereby avoiding computation of $A^T D A$. We refer to chapter 5 for a full description of the logarithmic barrier method.
3-3 Sequential Quadratic Programming

Another interesting method is Sequential Quadratic Programming (SQP), where a sequence of quadratic problems, which model problem (1-6), is solved:

$$\min_p \quad \frac{1}{2} x^T Bx + c^T x$$
subject to $$Ax \leq b$$

(3-8)

Here $$Ax \leq b$$ represent linearized versions of $$c_i(x), i \in \{1, 2, ..., m\}$$ and $$B$$ and $$c$$ are chosen such that problem (3-8) represents problem (1-6) in some way. The gain lies in the available methods for solving quadratic problems, in particular active-set methods, see appendix B. These iteratively make educated guesses of the active set of the quadratic problem (3-8), and then solve

$$\min_p \quad \frac{1}{2} x^T Gx + c^T x$$
subject to $$A_k x = b_k$$

(3-9)

where $$A_k x = b_k$$ represents the constraints active at the $$k^{th}$$ iteration. The constraints whose indices are not in the current guess of the active set, are temporarily ignored. Solving problem (3-9) comes down to solving a symmetric linear system of size $$n + m_k$$, where $$m_k$$ is the number of indices in the $$k^{th}$$ guess of the active set. This system is relatively small considering $$m_k$$ can never be larger than $$n$$. A downside of active-set methods is that they may need many guesses, i.e. iterations, before finding the active set. A thorough description of SQP is given in chapter 6.
To investigate if Sequential Quadratic Programming (SQP) and the logarithmic barrier method have potential to supersede interior point methods, we need test cases. During this research we used four different test cases, each with different characteristics. First we present the specifics of each test case. Then we present the results for these test cases using the current interior point solver.

4-1 Structures of the Test Cases

For each test case we have summarized the relevant information in tables 4-1 - 4-4. The upper part of each table shows the type and goal of the objective. The lower part lists the types of the different constraints, where the second column shows the number of constraints of the corresponding type.

The number of constraints $m$ is large because maximum dose constraints are sets of linear functions, i.e. $a_i x \leq b$, $i \in \{1, 2, ..., m\}$. Note that the intensity of a beam cannot be negative. Therefore, all test cases include an additional constraint, $x > 0$. This also has a consequence for the number of constraints $m$, which in reality is $m \leftarrow m + n$.

Test case 1 is really simple, since it contains linear constraints only, see table 4-1. Test case 2 is simple in the sense that it has a linear objective function, see table 4-2. Test case 3 is a bit more complex, having a nonlinear objective function and constraint, see table 4-3. Test case 4 is the most important test case. Both its size and complexity is realistic,
which is not true for test cases 1-3. So the first three cases merely serve as tools for our research, whereas our final goal is achieving a low runtime for test case 4. The specifics of test case 4 are presented in table 4-4.

<table>
<thead>
<tr>
<th>Table 4-1: Test Case 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
</tr>
<tr>
<td>LTCP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Constraints</strong></th>
<th>Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear (max dose)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( n = 696, \ m = 4986 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4-2: Test Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
</tr>
<tr>
<td>Linear</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Constraints</strong></th>
<th>Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear (max dose)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Linear (mean dose)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Linear (smoothing)</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Quadratic (smoothing)</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>( n = 696, \ m = 6385 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4-3: Test Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
</tr>
<tr>
<td>LTCP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Constraints</strong></th>
<th>Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear (max dose)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Linear (smoothing)</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>( n = 696, \ m = 22610 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4-2 Results for the Interior Point Method

Table 4-4: Test Case 4

<table>
<thead>
<tr>
<th>Objective</th>
<th>Type</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LTCP</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linear (max dose)</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Linear (mean dose)</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Linear (smoothing)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Quadratic (smoothing)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>EUD</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$n = 3571, m = 90184$</td>
<td></td>
</tr>
</tbody>
</table>

4-2 Results for the Interior Point Method

A summary of the results obtained using the current in-house developed interior point optimizer for test cases 1-4 is presented in table 4-5.

Table 4-5: Results

<table>
<thead>
<tr>
<th></th>
<th>Objective</th>
<th>Number of Iterations</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case 1</td>
<td>0.30</td>
<td>8</td>
<td>2.6 sec</td>
</tr>
<tr>
<td>Test Case 2</td>
<td>2.7</td>
<td>39</td>
<td>15 sec</td>
</tr>
<tr>
<td>Test Case 3</td>
<td>0.13</td>
<td>31</td>
<td>14 sec</td>
</tr>
<tr>
<td>Test Case 4</td>
<td>0.37</td>
<td>198</td>
<td>40 min</td>
</tr>
</tbody>
</table>

For each test case an initial guess $x_0$ of the solution $x^*$ is provided. This guess will serve as the initial value for the logarithmic barrier method as well as SQP, unless stated otherwise.
Chapter 5

Logarithmic Barrier Method

In this chapter we discuss the logarithmic barrier method, or simply barrier method, applied to problem (1-6)

\[
\begin{array}{ll}
\min_{x} & f(x) \\
\text{subject to} & c_i(x) \leq 0, \; i \in \{1, 2, \ldots, m\}
\end{array}
\]  \hspace{1cm} (5-1)

Instead of solving a sequence of constrained subproblems, the barrier method only deals with unconstrained subproblems, which can be solved using Newton methods. The idea is to rewrite the objective function in such a way that the original constraints are guaranteed to be satisfied at the solution. For instance, the solution of the unconstrained minimization problem

\[
\begin{array}{ll}
\min_{x} & f(x) + \sum_{i=1}^{m} I^- (c_i(x)), \quad \text{where} \quad I^- (y) = \begin{cases} 0 & y \leq 0 \\ \infty & y > 0 \end{cases}
\end{array}
\]  \hspace{1cm} (5-2)

coincides with the solution of problem (5-1) [6]. However, the objective function of problem (5-2) is not differentiable, which makes it unsuitable for Newton’s method. To circumvent this difficulty the barrier method approximates the indicator function \( I^- \) by the function

\[
\hat{I}^- (y) = \begin{cases} -\frac{1}{t} \log (-y) & y < 0 \\ \infty & y \geq 0 \end{cases}
\]  \hspace{1cm} (5-3)

where \( t \) is a parameter that determines the accuracy of the approximation. Figure 5-1 illustrates the effect of \( t \) on \( \hat{I}^- (y) \) [6]. Note that \( \hat{I}^- (y) \) is convex, nondecreasing and differentiable on \( \mathbb{R}^- \). Substitution of \( \hat{I}^- (y) \) into problem (5-2) yields

\[
\begin{array}{ll}
\min_{x} & f(x) - \frac{1}{t} \sum_{i=1}^{m} \log (-c_i(x))
\end{array}
\]  \hspace{1cm} (5-4)
We denote the solution of problem (5-4) by \( x^*(t) \) and define the barrier function

\[
g_t(x) = f(x) - \frac{1}{t} \sum_{i=1}^{m} \log(-c_i(x)) \tag{5-5}
\]

Intuition suggests, and we confirm this in the next section of this chapter, that the larger the parameter \( t \), the better the approximation \( x^*(t) \approx x^* \). On the other hand, for large values of \( t \), problem (5-4) is hard to solve because the Hessian of its objective function varies rapidly near the boundary of the feasible set. The logarithmic barrier method therefore solves a sequence of problems of type (5-4), gradually increasing the parameter \( t \) until some chosen accuracy has been reached.

\[\text{Figure 5-1: The dashed lines show the function } I^-(y), \text{ and the solid curves show } \hat{I}^-(y) \text{ for } t = 0.5, 1, 2. \text{ The curve for } t = 2 \text{ gives the best approximation.}\]
5-1 Central Path

The set of central points \( x^*(t), t > 0 \) is called the central path. Note that all points on the central path are strictly feasible with respect to the constraints and \( \nabla g_i(x^*(t)) = 0 \), so for all \( t > 0 \)

\[
\nabla f(x^*(t)) + \sum_{i=1}^{m} \frac{1}{-tc_i(x^*(t))} \nabla c_i(x^*(t)) = 0 \quad (5-6a)
\]

\[
c_i(x^*(t)) < 0, \quad i \in \{1, 2, ..., m\} \quad (5-6b)
\]

To get some sense of how close \( x^*(t) \) is to the solution \( x^* \), we refer to the KKT conditions of problem (5-1). The Lagrangian is given by

\[
\mathcal{L}(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i c_i(x) \quad (5-7)
\]

and the KKT conditions are given by

\[
\nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla c_i(x) = 0 \quad (5-8a)
\]

\[
c_i(x) \leq 0, \quad i \in \{1, 2, ..., m\} \quad (5-8b)
\]

\[
\lambda_i \geq 0, \quad i \in \{1, 2, ..., m\} \quad (5-8c)
\]

\[
\lambda_i c_i(x) = 0, \quad i \in \{1, 2, ..., m\} \quad (5-8d)
\]

Recall that the solution \((x^*, \lambda^*)\) of system (5-8) is unique. Comparing (5-6) to (5-8), we see that by defining

\[
\lambda_i^*(t) = \frac{1}{-tc_i(x^*(t))} > 0 \quad (5-9)
\]

and expressing (5-6a) as

\[
\nabla f(x^*(t)) + \sum_{i=1}^{m} \lambda_i^*(t) \nabla c_i(x^*(t)) = 0, \quad (5-10)
\]

the first three KKT conditions are satisfied at \( x = x^*(t) \), for \( \lambda_i = \lambda_i^*(t) \). The last conditions yield \( \lambda_i^*(t)c_i(x^*(t)) = -\frac{1}{t} \), which goes to zero as \( t \to \infty \).

Note that since \( \nabla_x \mathcal{L}(x^*(t), \lambda^*(t)) = 0 \), the Lagrangian for \( \lambda = \lambda^*(t) \) is minimized with respect to \( x \) at \( x = x^*(t) \), so

\[
\min_x \mathcal{L}(x, \lambda^*(t)) = \mathcal{L}(x^*(t), \lambda^*(t)) \leq \mathcal{L}(x^*, \lambda^*(t)) \quad (5-11)
\]

Writing out this inequality and substituting \( \sum_{i=1}^{m} \lambda_i^*(t)c_i(x^*(t)) = -\frac{m}{t} \), yields

\[
|f(x^*) - f(x^*(t))| \leq \frac{m}{t} \quad (5-12)
\]

This confirms the intuitive idea that \( x^*(t) \) converges to \( x^* \) as \( t \to \infty \).
5-2 Computing a Central Point

Finding a central point means solving an unconstrained minimization problem of type (5-4). In this thesis we consider three methods designed for unconstrained optimization. The first is the well-known Newton method, which uses the exact Hessian of $g_t(x)$ to find the solution.

$$\nabla^2 g_t(x) = \nabla^2 f(x) + \frac{1}{t} \sum_{i=1}^{m} \frac{1}{c_i(x)^2} \nabla c_i(x) \nabla c_i(x)^T - \frac{1}{t} \sum_{i=1}^{m} \frac{1}{c_i(x)} \nabla^2 c_i(x) \tag{5-13}$$

Computing this Hessian can become expensive when the problem is large. The term $\sum_{i=1}^{m} \frac{1}{c_i(x)^2} \nabla c_i(x) \nabla c_i(x)^T$ can be written as $A^T DA$, where $A^T = [\nabla c_1(x), \nabla c_2(x), ..., \nabla c_m(x)]$ and $D$ is a diagonal matrix with $D_{ii} = \frac{1}{c_i(x)^2}$.

In chapter 3 we stated that computing $A^T DA$ should be avoided, since it is the bottleneck of the currently implemented interior point method. We therefore also consider the quasi-Newton methods BFGS and L-BFGS, which use an approximation of the Hessian $\nabla^2 g_t(x)$. A description of these methods can be found in appendix A.

5-3 Algorithm

As mentioned before, the barrier method solves a sequence of problems of type (5-4), where the parameter $t$ is increased at each iteration. For the remainder of this chapter we refer to the iterations executed while solving a subproblem as inner iterations. We refer to computation of a central point as outer iterations. To formulate a practical algorithm we need to define how $t$ is increased and we need a good stopping criterion.

Let us start with the latter. We know that $|f(x^*(t) - f(x^*)| \leq \frac{m}{t}$ for all $t > 0$. We can therefore simply choose the accuracy we wish to achieve by imposing $|f(x^*(t) - f(x^*)| \leq \epsilon$, for some tolerance $\epsilon$, and stop the algorithm when $t > \frac{m}{\epsilon}$. Note that in practice the central point $x^*(t)$ is not computed exactly. The optimization methods used to determine the solution of problem (5-4) typically stop when the norm of the gradient of the objective function is smaller than some tolerance $\tilde{\epsilon}$. (We elaborate on this matter in the next section). Therefore, we cannot guarantee an $\epsilon$-accuracy in practice, but it can be shown that the error is negligible for sufficiently small $\tilde{\epsilon}$ [6].

A simple way to increase $t$ at each iteration is to multiply it by some constant $\mu > 1$. Note that by choosing $\mu$ fixed, the number of outer iterations is also fixed. Smaller values of
\( \mu \) lead to more outer iterations, but fewer inner iterations, because the previous central point is a good initial guess for the next central point. For large values of \( \mu \) the opposite is valid. So the choice of \( \mu \) involves a trade-off between inner and outer iterations. According to [6], the total number of inner iterations is more or less constant for \( 3 \leq \mu \leq 100 \), so the choice of \( \mu \) is not particularly critical. Values from around 10 to 20 are said to work well.

A practical algorithm based on the logarithmic barrier method has been summarized in algorithm 1.

**Algorithm 1 Logarithmic Barrier Algorithm**

Choose \( t_0 \) and strictly feasible initial point \( x_0 \);

for \( k = 0, 1, 2, \ldots \) do

Compute \( x^*(t_k) \) by solving (5-4) with initial point \( x_k \);

Set \( x_{k+1} \leftarrow x^*(t_k) \);

if \( m/t_k < \epsilon \) then

    stop with approximate solution \( x^*(t_k) \);

end if

Set \( t_{k+1} \leftarrow \mu t_k \);

end for

5-4 Implementation

In this section we deal with several matters regarding the implementation of the barrier method. In particular we elaborate on computing initial values and techniques to simplify the inner iterations.

5-4-1 Initial Values

A difficulty of the barrier method is the need for a strictly feasible initial point \( x_0 \). In this section we discuss how to circumvent this. We also determine a good initial penalty parameter \( t_1 \).

**Initial Point \( x_0 \)**

Let us first discuss an approach where we explicitly find a strictly feasible \( x_0 \), which serves as an initial point to solve problem (5-1) with the barrier method. Such points can be
found by solving

$$\min_s \quad s$$

subject to \( c_i(x) < s, \quad i \in \{1, 2, ..., m\} \) \tag{5-14}

Note that \( s_0 = \max_i c_i(x) + \delta \), yields a strictly feasible initial point of problem (5-14) for any \( x \) and \( \delta > 0 \). We can therefore solve this problem with the barrier method and stop when \( s < 0 \). The solution \( x_0 \) is then used as a strictly feasible initial value for problem (5-1). However, this approach may lead to high values of the objective function \( f(x_0) \), since it is completely left out of the equation. Also, solving two problems requires computation of the initial penalty parameter \( t_0 \) twice, which is undesirable considering the sensitivity of the barrier method with respect to this value (see next section).

To circumvent these undesirable properties, we combine problem (5-14) and problem (5-1) to obtain:

$$\min_{x,s} \quad f(x) + \nu s$$

subject to \( c_i(x) < s, \quad i \in \{1, 2, ..., m\} \)

\[ s \geq 0 \tag{5-15} \]

The solution of problem (5-15) coincides with \( x^* \) as long as the penalty parameter \( \nu \) is large enough. Also, by choosing \( s_0 = \max\{\max_i c_i(x), 0\} + \delta \), we satisfy all constraints for any \( x \). To determine the parameter \( \nu \) we use information of the objective function, in particular its goal \( g \) (see section 1-1). Recall that the goal \( g \) of an objective in iCycle context is the value we hope the minimization problem to achieve. We therefore set \( \nu = \max\{g, 1\} \). This way, the weight of meeting the constraints is approximately as high as that of minimizing the objective function.

**Initial Penalty Parameter \( t_1 \)**

A good initial penalty parameter \( t_1 \) is very important. If \( t_1 \) is too large, computation of the first central point \( x^*(t_1) \) requires too many iterations. On the other hand, choosing \( t_0 \) too small leads to more outer iterations than necessary, and, in addition, computation of \( x^*(t_1) \) may also be hard, because the weight put on minimizing the constraints is too large. A possible approach is to choose \( t_1 \) such that the distance between \( x_0 \) and \( x^*(t_1) \) is as small as possible. Obviously we do not know \( x^*(t_1) \), so we use a different measure instead. We know that \( \nabla g_t(x^*(t)) = 0 \) for all \( t > 0 \), so we want to find the \( t_1 \) which minimizes \( \|\nabla g_t(x_0)\|_2 \). Note that minimizing \( g_t(x) \) is equivalent to minimizing \( tg_t(x) \), so for simplicity we solve the following problem

$$\min_t \quad \|t\nabla f(x_0) + \nabla \phi(x_0)\|_2 \tag{5-16}$$
where $\phi(x) = -\sum_{i=1}^{m} \log(-c_i(x))$, so $\nabla \phi(x) = -\sum_{i=1}^{m} \frac{1}{c_i(x_0)} \nabla c_i(x_0)$ [6]. The solution yields

$$t_1 = -\frac{\nabla f(x_0)^T \nabla \phi(x_0)}{f(x_0)^T f(x_0)}$$

Choosing $t_1$ as equation (5-17) ensures that the distance between $x_0$ and $x^*(t_1)$ is minimized in some sense.

### 5-4-2 Subproblems

In this section we discuss some important issues regarding the implementation of the methods to solve the subproblems (5-4). First we discuss how to deal with infeasible iterates by introducing a step size $\alpha$. Then we determine a good stopping criterion.

#### Stepsize

A straightforward way to deal with infeasible iterates is to simply cut down the step $p$ by some stepsize $\alpha$ such that $c_i(x + \alpha p) < 0$ for all $i \in \{1, 2, ..., m\}$. However, if the iterates get too close to the boundary of the feasible region, the barrier function $g_t(x)$ blows up, causing its Hessian to become ill-conditioned. The performance of Newton’s method is not affected by this, but for quasi-Newton methods ill-conditioning is highly undesirable. We therefore want the iterates to stay at an acceptable distance from the feasible boundary. But how do we define this margin? As $t$ increases the iterates should be allowed closer to the feasible boundary, since the solution $x^*$ may have active constraints. We therefore determine $\alpha$ such that

$$c_i(x + \alpha p) \leq -\frac{1}{t}$$

(5-18)

Although the logarithm naturally keeps the constraints from coming too close to zero, it is not ruled out that $c_i(x^*(t)) > -\frac{1}{t}$ for some $i \in \{1, 2, ..., m\}$. In such cases enforcing (5-18) might yield $\alpha = 0$, thereby landing the algorithm in an infinite loop. To avoid this, we slightly relax condition (5-18) if the allowed stepsize $\alpha$ becomes too small.

#### Stopping Criterion

Usually, Newton-based methods terminate the algorithm when the norm of the gradient of the objective function is smaller than some tolerance $\tilde{\epsilon}$. Though this works generally well, we have the luxury that we do not need to fully optimize the subproblems, since central points only serve as initial guesses for their successor. It was proposed by Gonzaga.
and Powell to terminate the algorithm when the predicted difference between \( g_t(x) \) and \( g_t(x^*(t)) \) is less than some constant \( \sigma \) between 0.1 and 0.3 [7]:

\[
g_t(x) - g_t(x^*(t)) \approx \nabla g_t(x)^T [\nabla^2 g_t(x)]^{-1} \nabla g_t(x) \leq \sigma \tag{5-19}
\]

Here we have used that \( g_t(x^*(t)) \approx g_t(x) + [\nabla g_t(x)]^T p, \) where \( p = -[\nabla^2 g_t(x)]^{-1} \nabla g_t(x) \approx x^*(t) - x \) is the Newton step (see appendix A-1).

### 5-5 Newton-Barrier Method

We have already argued that by applying Newton’s method to solve the subproblems (5-4), the barrier method cannot in general supersede the interior point method that is currently used in iCycle, because it suffers from the same bottleneck. So we only use the Newton-barrier method as a reference with respect to the barrier method with the quasi-Newton approach. In particular we use the Newton-Barrier method to investigate the influence of different parameters.

We start with a discussion regarding the tolerance parameters \( \epsilon \) and \( \tilde{\epsilon} \). Then we move on to investigate the initial penalty parameter \( t_1 \). We close with an analysis of the updating parameter \( \mu \).

For the remainder of this chapter we use \( k \) to indicate the outer iterations and \( j \) to indicate the inner iterations. The results are generally presented in tables where each row shows information of one outer iteration. The first column numbers the outer iterations, \( k = 0, 1, \ldots \). The second column shows the value of the objective function \( f(x_k) \). The third column presents the highest value among the constraints, i.e. \( \max c(x_k) \). The fourth column gives the penalty parameter \( t_k \). The fifth columns shows the norm of the gradient of the barrier function at \( x_k \).

When Newton’s method is used to fully optimize an unconstrained minimization problem, the algorithm is typically terminated when the norm of the gradient of the objective function is lower than some tolerance \( \tilde{\epsilon} \). However, as discussed in section (5-4-2), we terminate the algorithm when

\[
\nabla [g_{t_k}(x_j)]^T p_j \leq \sigma \tag{5-20}
\]

for some fixed \( \sigma \in [0.1, 0.3] \). So the value shown in the fifth column is an indication of how accurate the approximation of the central point \( x_k \approx x^*(t_k) \) is. We label this column \( \tilde{\epsilon}_k \),
with \[ \tilde{\epsilon}_k = \|g_t(x_k)\|_2 \] (5-21)

The last column presents the number of inner iterations needed to compute the \(k^{th}\) \(\tilde{\epsilon}_k\)-accurate central point. We denote this number by \(N_k\). The last row summarizes information of the approximate solution. In particular, the last column of the last row shows the total number of inner iterations.

### 5-5-1 Tolerance Levels \(\epsilon\) and \(\tilde{\epsilon}\)

We start by presenting the results of test case 2 (table 5-1), which has a linear objective function and linear and quadratic constraints. The constraints decrease rapidly \((k = 0 - 5)\) and as soon as they are satisfied \((k = 6)\), the objective function strictly decreases. So the global behaviour of the barrier method on test case 2 seems satisfactory.

<table>
<thead>
<tr>
<th>(k)</th>
<th>(f(x_k))</th>
<th>max (c(x_k))</th>
<th>(t_k)</th>
<th>(\tilde{\epsilon}_k)</th>
<th>(N_k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.3274e-02</td>
<td>9.09e+01</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.6071e+01</td>
<td>2.77e+01</td>
<td>9.72e-01</td>
<td>3.74e-02</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>1.0420e+01</td>
<td>3.10e+01</td>
<td>2.92e+00</td>
<td>3.93e+00</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1.1653e+01</td>
<td>1.26e+01</td>
<td>8.75e+00</td>
<td>1.17e-02</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>1.0709e+01</td>
<td>3.32e+00</td>
<td>2.62e+01</td>
<td>1.32e-02</td>
<td>45</td>
</tr>
<tr>
<td>5</td>
<td>1.0927e+01</td>
<td>1.48e-01</td>
<td>7.87e+01</td>
<td>3.19e-02</td>
<td>39</td>
</tr>
<tr>
<td>6</td>
<td>7.4641e+00</td>
<td>0.00e+00</td>
<td>2.36e+02</td>
<td>2.97e-02</td>
<td>69</td>
</tr>
<tr>
<td>7</td>
<td>4.9010e+00</td>
<td>0.00e+00</td>
<td>7.09e+02</td>
<td>7.47e-02</td>
<td>82</td>
</tr>
<tr>
<td>8</td>
<td>3.5793e+00</td>
<td>0.00e+00</td>
<td>2.13e+03</td>
<td>3.76e-01</td>
<td>103</td>
</tr>
<tr>
<td>9</td>
<td>3.0109e+00</td>
<td>0.00e+00</td>
<td>6.38e+03</td>
<td>6.58e-01</td>
<td>129</td>
</tr>
<tr>
<td>10</td>
<td>2.8480e+00</td>
<td>0.00e+00</td>
<td>1.91e+04</td>
<td>1.72e+00</td>
<td>231</td>
</tr>
<tr>
<td>11</td>
<td>2.8171e+00</td>
<td>0.00e+00</td>
<td>5.74e+04</td>
<td>4.76e+00</td>
<td>395</td>
</tr>
<tr>
<td>12</td>
<td>2.8102e+00</td>
<td>0.00e+00</td>
<td>1.72e+05</td>
<td>1.50e+01</td>
<td>635</td>
</tr>
<tr>
<td>13</td>
<td>2.8081e+00</td>
<td>0.00e+00</td>
<td>5.17e+05</td>
<td>4.56e+01</td>
<td>939</td>
</tr>
<tr>
<td>14</td>
<td>2.8074e+00</td>
<td>0.00e+00</td>
<td>1.55e+06</td>
<td>1.35e+02</td>
<td>1193</td>
</tr>
</tbody>
</table>

| 14  | 2.8074e+00 | 0.00e+00 | 1.55e+06 | 1.35e+02 | 3893 |

**Table 5-1:** Newton-Barrier method applied to test case 2, with \(\mu = 3\), \(\epsilon = 0.01\) and \(\sigma = 0.3\).

However, looking at the last two columns we see that solving the subproblems becomes increasingly expensive as the penalty parameter \(t_k\) grows. Not only does the number of
inner iterations $N_k$ increase, but the accuracy of the solution $x_k$ of the subproblems decreases at an alarming rate as well. Lowering the value $\sigma$ does not change this behaviour. Comparing $f(x_{14}) = 2.81$ to $f(x^*) \leq 2.7$ (see chapter (4)), we conclude that indeed the solution $x_{14}$ could be more accurate. There are two ways to rectify this.

We can choose a smaller $\epsilon$. This leads to solving additional subproblems with an even higher penalty parameter than $t_{14}$. Since solving problems with a high penalty parameter is (apparently) very expensive, we have to consider if it is worth the effort. Let us look at the last few iterations of table 5-1. The value $f(x_k)$ decreases very slowly. This is probably because the iterates seem to strongly deviate from the central path. It is therefore reasonable to conclude that little can be gained by computing more outer iterations.

The other approach is thus to force the tolerance $\tilde{\epsilon}_k$ below some level, so that the deviation from the central path is bounded. This would yield a higher accuracy of the central points, in particular that of the last iteration, which is the final estimate of $x^*$. However, naturally, it requires more inner iterations. To give some sense of the number of extra iterations needed to obtain a slightly higher accuracy, we show the 12th iteration of test case 2 (table 5-1), if we force $\tilde{\epsilon}_k \leq 10$:

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\max c(x_k)$</th>
<th>$t_k$</th>
<th>$\tilde{\epsilon}_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>2.8171e+00</td>
<td>0.00e+00</td>
<td>5.74e+04</td>
<td>4.76e+00</td>
<td>395</td>
</tr>
<tr>
<td>12</td>
<td>2.7777e+00</td>
<td>0.00e+00</td>
<td>1.72e+05</td>
<td>1.00e+01</td>
<td>40250</td>
</tr>
</tbody>
</table>

Though the result of the 12th iteration is clearly more accurate than with a tolerance of $\tilde{\epsilon}_k = 15$, simply forcing the iterates closer to the central path is not realizable, because of the extreme number of extra inner iterations. Other test cases show similar behaviour.

To keep the barrier method in the race for superseding the interior point method, we simply have to accept a lower accuracy of our final solution. To that end, we set $\epsilon$ to a higher value in order to avoid computation of the last few iterations. For test case 2 with $\mu = 3$, choosing $\epsilon = 0.1$ corresponds to stopping after 12 iterations (see table 5-1). So the final solutions would yield $f(x_{12}) = 2.810$ instead of $f(x_{14}) = 2.807$, which we consider acceptable. We therefore set $\epsilon = 0.1$.

Fixing the same $\epsilon$ for all test cases is not optimal, since it is an upper bound for the absolute difference between $f(x^*(t_k))$ and $f(x^*)$. However, determining $\epsilon$ from a chosen relative tolerance is not straightforward, since we do not know $f(x^*)$. Furthermore, the
difficulties of the barrier method seem to depend on the value of the penalty parameter $t_k$. A tolerance level $\epsilon$ that is solely based on a chosen relative tolerance can become very low if the number of constraints $m$ is large, which causes (undesirably) high values of $t_k$. In short, determining a good tolerance $\epsilon$ for each individual problem is very complex. Besides, $\epsilon$ indicates only an upper bound, so the solution may be far more accurate than $f(x^*(t)) - f(x^*) = \epsilon$. Taking all this into account, we decide to fix $\epsilon = 0.1$. We justify this choice by noting that we merely seek to investigate the potential of the barrier method, and $\epsilon = 0.1$ turns out to be a reasonable choice for all test cases.

### 5-5-2 Initial Penalty Parameter $t_1$

We have discussed the tolerances $\epsilon$ and $\tilde{\epsilon}_k$. We continue our analysis by evaluating our choice of the initial penalty parameter $t_1$. Recall that we claimed that the performance of the barrier method is sensitive to this value. For test case 2, table 5-1 shows that computing $t_1$ according to (5-17) is suitable. The first iteration shows progress with respect to the initial value, while the number of inner iterations is low. For test case 3 however, the first outer iteration needs many inner iterations, see table 5-3. We therefore investigate the effect of varying the initial penalty parameter $t_1$.

We denote the new initial penalty parameter by $\tilde{t}_1$. To compare the results for different values of $\tilde{t}_1$ with those for $t_1$, we set $\tilde{t}_1 = \mu^n t_1$ with $n \in \{-3, -2, -1, 0, 1\}$. Note that choosing $\tilde{t}_1$ this way yields $\tilde{t}_k = t_2$, where $k = 2 - n$. The results for test case 3 are shown in table 5-2. The first column indicates the number of iterations to get to the second iteration of table 5-3 ($k = 2 - n$). The second column presents the initial penalty parameter $\tilde{t}_1$. The last column shows the total number of inner iterations, i.e. $N = \sum_{i=1}^k N_i$, where $N_i$ is the number of inner iterations corresponding to the $i^{th}$ outer iteration.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$\mu t_1$</th>
<th>$\mu^n t_1$</th>
<th>$N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$9780$</td>
<td>$-362$</td>
<td>$399$</td>
</tr>
<tr>
<td>2</td>
<td>$3260$</td>
<td>$3260$</td>
<td>$203$</td>
</tr>
<tr>
<td>3</td>
<td>$1087$</td>
<td>$1087$</td>
<td>$166$</td>
</tr>
<tr>
<td>4</td>
<td>$362$</td>
<td>$362$</td>
<td>$181$</td>
</tr>
<tr>
<td>5</td>
<td>$121$</td>
<td>$121$</td>
<td>$522$</td>
</tr>
</tbody>
</table>

**Table 5-2: Test Case 3**

We see that the optimum for the initial penalty parameter probably lies between $\tilde{t}_1 = 362$ and $\tilde{t}_1 = 3260$. Although we cannot conclusively state that even the optimal $\tilde{t}_1$ would yield poor results, the information in table 5-2 suggests that the large number of inner iterations corresponding to the first outer iteration is not (only) due to the initial penalty parameter. This problem may originate in the initial value $x_0$ and the very structure of the problem.

But before we analyse this issue in more detail, we want to provide the reader with some perspective. Compare for example $t_1 = 0.97$ for test case 2 with $t_1 = 3260$ for test case 3.
Table 5-3: Newton-Barrier method applied to test case 3, with \( \mu = 3, \epsilon = 0.1 \) and \( \sigma = 0.3 \).

By extrapolating the information of table 5-2, we can imagine that choosing \( t_1 \sim 1 \) like in test case 2, would lead to an immense number of unnecessary outer and inner iterations. Specifically, running test case 3 with \( \tilde{t}_1 = 1 \) requires 3380 inner iterations for the first outer iteration alone. The other way around would also be disastrous. Table 5-4 shows the effect of increasing the initial penalty parameter of test case 2.

We can establish that at least the order of the initial penalty parameter \( t_1 \) determined by (5-17) is suitable for both test cases. In fact, we have verified via similar analyses that this is true for all test cases. Even though test case 3 would slightly benefit from choosing \( \tilde{t}_1 \) = \( \frac{\mu}{\mu} \), it is not in general a better choice. Furthermore, since there is very little information available to determine an initial penalty parameter, we should be satisfied with the results obtained by computing \( t_1 \) according to (5-17).

Table 5-4: Test Case 2

5-5-3 Updating Parameter \( \mu \)

In contrast to what was predicted in section 5-3, the updating parameter \( \mu \) turns out to play a key role in our analysis. Note that \( \mu \) defines in some sense the quality of the initial value \( x_{k-1} \) for computing the next central point. Indeed, for smaller \( \mu \) the central points \( x^*(t_{k-1}) \) and \( x^*(t_k) \) are closer to each other. So it might make sense to choose smaller \( \mu \) for subproblems that are hard to solve.
Take for example test case 3 (table 5-3), where the iterations are relatively expensive. Table 5-5 shows the results of test case 3 for $\mu = \sqrt{3}$. Note that choosing $\mu$ this way results in twice as many outer iterations. We see that the results are significantly better than for $\mu = 3$. On the other hand, for test case 2, where the greater part of the subproblems seem easy to solve, choosing $\mu = \sqrt{3}$ is not beneficial (table C-2 in appendix C). The total number of inner iterations is more: 2109 instead of 1761 (table 5-1 at $k = 12$).

It appears that smaller $\mu$ are suitable when the subproblems are hard to solve, but when there are no complications the algorithm benefits from larger $\mu$. (We have verified that this statement is consistent with all test cases.) So, for optimal performance of the barrier method, the value $\mu$ should vary along with the complexity of the subproblems, i.e. $\mu = \mu(k)$. Can we find a pattern to base the function $\mu(k)$ on?

For most test cases the complications arise for large values of $t_k$, so we could reduce $\mu(k)$ when $t_k$ crosses some predetermined fixed value. Another option is to gradually reduce $\mu(k)$, where $\mu(1)$ is based on $t_1$. However, unfortunately we also have problems which already show difficulties in the first few iterations, see test case 4. Table 5-6 shows the results for $\mu = 3$. The third iteration for $\mu = 3$ is suddenly very expensive. For $\mu = \sqrt{3}$ this complication does not arise, see table C-4.

The results of test case 4 suggest that no (obvious) general pattern exists among our test cases, let alone among all problems of type (5-1). Since we do not know beforehand for

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\text{max } c(x_k)$</th>
<th>$t_k$</th>
<th>$\bar{\epsilon}_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.3015e-01</td>
<td>1.59e+00</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0372e-01</td>
<td>0.00e+00</td>
<td>3.26e+03</td>
<td>1.05e-01</td>
<td>178</td>
</tr>
<tr>
<td>2</td>
<td>5.2924e-01</td>
<td>0.00e+00</td>
<td>5.65e+03</td>
<td>4.27e-01</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>3.7909e-01</td>
<td>0.00e+00</td>
<td>9.79e+03</td>
<td>4.49e-01</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>2.8059e-01</td>
<td>0.00e+00</td>
<td>1.70e+04</td>
<td>5.38e-01</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>2.1762e-01</td>
<td>0.00e+00</td>
<td>2.94e+04</td>
<td>5.59e-01</td>
<td>49</td>
</tr>
<tr>
<td>6</td>
<td>1.7901e-01</td>
<td>0.00e+00</td>
<td>5.09e+04</td>
<td>6.40e-01</td>
<td>99</td>
</tr>
<tr>
<td>7</td>
<td>1.5759e-01</td>
<td>0.00e+00</td>
<td>8.81e+04</td>
<td>7.23e-01</td>
<td>155</td>
</tr>
<tr>
<td>8</td>
<td>1.4709e-01</td>
<td>0.00e+00</td>
<td>1.53e+05</td>
<td>1.10e+00</td>
<td>229</td>
</tr>
<tr>
<td>9</td>
<td>1.4254e-01</td>
<td>0.00e+00</td>
<td>2.64e+05</td>
<td>2.04e+00</td>
<td>340</td>
</tr>
<tr>
<td>9</td>
<td>1.4254e-01</td>
<td>0.00e+00</td>
<td>2.64e+05</td>
<td>2.04e+00</td>
<td>1107</td>
</tr>
</tbody>
</table>

Table 5-5: Newton-Barrier method applied to test case 3, with $\mu = \sqrt{3}$, $\epsilon = 0.1$ and $\sigma = 0.3$. 

Logarithmic Barrier Method

\[ f(x_k) \max\ c(x_k) \quad t_k \quad \tilde{\epsilon}_k \quad N_k \]

<table>
<thead>
<tr>
<th>( k )</th>
<th>( f(x_k) )</th>
<th>( \max\ c(x_k) )</th>
<th>( t_k )</th>
<th>( \tilde{\epsilon}_k )</th>
<th>( N_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9.1183e+00</td>
<td>1.14e+02</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.3397e-01</td>
<td>9.54e+00</td>
<td>1.04e+03</td>
<td>4.68e-03</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>2.3787e-01</td>
<td>1.87e+01</td>
<td>3.13e+03</td>
<td>8.44e-02</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4.3335e-01</td>
<td>1.33e+00</td>
<td>9.39e+03</td>
<td>1.49e-01</td>
<td>1202</td>
</tr>
<tr>
<td>4</td>
<td>4.5599e-01</td>
<td>0.00e+00</td>
<td>2.82e+04</td>
<td>6.11e-01</td>
<td>597</td>
</tr>
<tr>
<td>5</td>
<td>3.7093e-01</td>
<td>0.00e+00</td>
<td>8.45e+04</td>
<td>8.97e-01</td>
<td>503</td>
</tr>
<tr>
<td>6</td>
<td>3.5083e-01</td>
<td>0.00e+00</td>
<td>2.54e+05</td>
<td>1.65e+00</td>
<td>1141</td>
</tr>
<tr>
<td>7</td>
<td>3.4712e-01</td>
<td>0.00e+00</td>
<td>7.61e+05</td>
<td>4.84e+00</td>
<td>1251</td>
</tr>
<tr>
<td>8</td>
<td>3.4611e-01</td>
<td>0.00e+00</td>
<td>2.28e+06</td>
<td>1.48e+01</td>
<td>2132</td>
</tr>
<tr>
<td>8</td>
<td>3.4611e-01</td>
<td>0.00e+00</td>
<td>2.28e+06</td>
<td>1.48e+01</td>
<td>6850</td>
</tr>
</tbody>
</table>

Table 5-6: Newton-Barrier method applied to test case 4, with \( \mu = 3, \epsilon = 0.1 \) and \( \sigma = 0.3 \).

which values of \( t_k \) difficulties arise, we cannot benefit from a non-constant \( \mu(k) \), so we fix \( \mu(k) = \mu \).

As argued before, the constant \( \mu \) involves a trade-off between solving tough subproblems and extra outer iterations. For test cases 1-4, choosing \( \mu = \sqrt{3} \) works well: all subproblems are reasonably easy to solve, while the number of outer iterations remains acceptable. A summary of the results of the Newton-barrier algorithm for all test cases is given in table 5-7. The last column shows the total number of iterations corresponding the interior method. The full results are presented in appendix C.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( f(x_k) )</th>
<th>( \max\ c(x_k) )</th>
<th>( t_k )</th>
<th>( \tilde{\epsilon}_k )</th>
<th>( N )</th>
<th>( N_{IP} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case 1</td>
<td>9</td>
<td>1.4039e-01</td>
<td>0.00e+00</td>
<td>7.44e+04</td>
<td>4.44e-01</td>
<td>187</td>
</tr>
<tr>
<td>Test Case 2</td>
<td>23</td>
<td>2.8233e+00</td>
<td>0.00e+00</td>
<td>1.72e+05</td>
<td>1.63e+01</td>
<td>2109</td>
</tr>
<tr>
<td>Test Case 3</td>
<td>9</td>
<td>1.4254e-01</td>
<td>0.00e+00</td>
<td>2.64e+05</td>
<td>2.04e+00</td>
<td>1107</td>
</tr>
<tr>
<td>Test Case 4</td>
<td>14</td>
<td>3.4556e-01</td>
<td>0.00e+00</td>
<td>1.32e+06</td>
<td>8.76e+00</td>
<td>4215</td>
</tr>
</tbody>
</table>

Table 5-7: Results of the Newton-Barrier method.

To gain some extra insight in our findings, we quote from [6]:

"When the parameter \( \mu \) is chosen to give the best worst-case bound on the total number of Newton steps required, values of \( \mu \) near one are used."

The same source advices to choose \( 3 \leq \mu \leq 100 \), or even more specific \( 10 \leq \mu \leq 20 \). This suggests that if small values of \( \mu \) seem to work best, we are probably close to the worst
case scenario. This is plainly confirmed by comparing our results with those resulting from
the interior point method (see table 5-7). Keep in mind that the runtime of one iteration
computed by the interior point method is of the same order as the runtime of one inner
iteration computed by the Newton-barrier method. So we have to wonder whether the
barrier method is suitable for our type of problems.

Nevertheless, since iterations with quasi-Newton methods are significantly cheaper then
those with Newton’s method, we continue our research of the barrier method. We will see
that quasi-Newton methods are even more sensitive to the updating parameter $\mu$, in the
sense that the consequences of choosing $\mu$ too large are more severe.

5-5-4 Summary

Before we move on to quasi-Newton methods, we summarize our findings of the general
barrier method. First, we have concluded that we have to accept a relatively low accuracy
for our solution. We fix $\epsilon = 0.1$ and let $\tilde{\epsilon}$ depend on the barrier function, as is described in
section 5-4-2. Then we established that choosing the initial penalty parameter $t_1$ according
to equation (5-17) works well. Finally, we saw that reducing $\mu$ from 3 to $\sqrt{3}$ is necessary
to ensure that all test cases perform adequately.

5-6 BFGS-Barrier Method

In this section we discuss the BFGS-barrier method. The corresponding results will give us
some sense of the general performance of quasi-Newton methods in this context. Indeed,
the L-BFGS-method, which we discuss in the next section, is directly derived from the
BFGS method. In fact, for certain parameter choices the two are identical.

The most important difference between Newton and the BFGS method is that the latter
avoids computing the exact Hessian $\nabla^2 g_t(x)$. We saw in section 5-2 that computation of the
exact Hessian requires computation of $A^T DA$, where $A^T = [\nabla c_1(x), \nabla c_2(x), \ldots, \nabla c_m(x)]$
and $D$ is a diagonal matrix. This alone requires $O(mn^2)$ multiplications. In addition, the
BFGS method approximates the inverse of the Hessian directly, so we do not have to solve
a linear system to determine the search direction $p_j$. An iteration of the BFGS method
requires (apart from function evaluations) only $O(n^2)$ multiplications in total. Note that
for our problems usually $m \gg n$. As a result, the inner iterations of the BFGS-barrier
method are significantly less expensive than with Newton.
Logarithmic Barrier Method

The BFGS-barrier method (see appendix A-2-1) requires an additional initial value, \( H_0^k \), which is the estimate of the inverse Hessian \([\nabla^2 g_t(x_{k-1})]^{-1}\). The search direction \( p_0 \) of the first inner iteration of the \( k^{th} \) outer iteration is computed by 
\[ p_0 = -H_0^k \nabla g_t(x_{k-1}). \]
We want \( p_0 \) to be of the right order, so it is natural to define 
\[ H_0^k = \beta I, \]
where \( \beta = \| x_k - x_{k-1} \| / \| \nabla g_t(x_k - x_{k-1}) \| \), such that \( \| p_0 \| = \| x_k - x_{k-1} \| \). Several tests indicate that using the 2-norm yields the best results.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( f(x_k) )</th>
<th>( \max c(x_k) )</th>
<th>( t_k )</th>
<th>( \epsilon_k )</th>
<th>( N_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.3015e-01</td>
<td>1.59e+00</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.1024e-01</td>
<td>1.64e+00</td>
<td>3.26e+03</td>
<td>4.45e+06</td>
<td>72</td>
</tr>
<tr>
<td>2</td>
<td>4.8852e-01</td>
<td>0.00e+00</td>
<td>5.65e+03</td>
<td>5.37e+00</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>3.7747e-01</td>
<td>0.00e+00</td>
<td>9.79e+03</td>
<td>6.28e+02</td>
<td>210</td>
</tr>
<tr>
<td>4</td>
<td>2.7930e-01</td>
<td>0.00e+00</td>
<td>1.70e+04</td>
<td>2.71e+02</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>2.1466e-01</td>
<td>0.00e+00</td>
<td>2.94e+04</td>
<td>9.92e+01</td>
<td>93</td>
</tr>
<tr>
<td>6</td>
<td>1.7398e-01</td>
<td>0.00e+00</td>
<td>5.09e+04</td>
<td>1.07e+01</td>
<td>115</td>
</tr>
<tr>
<td>7</td>
<td>1.4879e-01</td>
<td>0.00e+00</td>
<td>8.81e+04</td>
<td>8.05e+03</td>
<td>202</td>
</tr>
<tr>
<td>8</td>
<td>1.3351e-01</td>
<td>0.00e+00</td>
<td>1.53e+05</td>
<td>2.93e+01</td>
<td>307</td>
</tr>
<tr>
<td>9</td>
<td>1.2606e-01</td>
<td>0.00e+00</td>
<td>2.64e+05</td>
<td>2.46e+03</td>
<td>442</td>
</tr>
<tr>
<td>9</td>
<td>1.2606e-01</td>
<td>0.00e+00</td>
<td>2.64e+05</td>
<td>2.46e+03</td>
<td>1497</td>
</tr>
</tbody>
</table>

Table 5-8: BFGS-Barrier method applied to test case 3, with \( \mu = \sqrt{3} \), \( \epsilon = 0.1 \) and \( \sigma = 0.3 \).

In table 5-8 the results are presented for test case 3. On first sight the results seem very good: the final value of the objective function is even smaller than that resulting from Newton’s method, and, in addition, the total number of inner iterations is not much higher, so we definitely gain a lot with regard to computation time. However, looking more closely, we see that the accuracy of the central points is critically low, including that of the final outer iteration. This is not unexpected, because the stopping criterium that we use for the subproblems is less precise than for the Newton-barrier method. Indeed, the BFGS-method uses \( H_j^k \) instead of the inverse Hessian \([\nabla^2 g_t(x_j)]^{-1}\). So why is our final result so good?

Apparently the values of the constraints are too high with respect to \( c(x^*(t_9)) \), which makes \( x_9 \) a poor approximation of \( x^*(t_9) \). But since the constraints are still all smaller or equal to zero it yields a good final solution for us. All test cases yield similar favourable results, except test case 2, see appendix C.

For test case 2, the implementation choice of the updating formula (A-10) turns out to have a significant influence on the results. This expression without subscripts:
5-6 BFGS-Barrier Method

$H \leftarrow \left( I - \frac{py^T}{p^Ty} \right) H \left( I - \frac{yp^T}{p^Ty} \right) + \frac{pp^T}{p^Ty}$ (5-22)

To save time we first compute $\rho = \frac{1}{p^Ty}$, $\gamma = y^THy$, $B_1 = py^THy$ and $B_2 = pp^T$. The updating formula for $H$ becomes

$H \leftarrow H - \rho \left[ B_1 + B_1^T - (\rho \gamma + 1)B_2 \right]$ (5-23)

Table 5-9: Effect of replacing implementation 5-23 by (5-22) for test case 2.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\max c(x_k)$</th>
<th>$t_k$</th>
<th>$\tilde{\epsilon}_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.3274e-02</td>
<td>9.09e+01</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.3349e+01</td>
<td>2.67e+01</td>
<td>9.72e-01</td>
<td>3.90e-02</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>9.2048e+00</td>
<td>3.60e+01</td>
<td>1.68e+00</td>
<td>1.86e-02</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>1.0732e+01</td>
<td>3.19e+01</td>
<td>2.92e+00</td>
<td>8.38e+00</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>1.3920e+01</td>
<td>2.16e+01</td>
<td>5.05e+00</td>
<td>1.52e+01</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>1.2510e+01</td>
<td>1.27e+01</td>
<td>8.75e+00</td>
<td>1.02e+02</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>1.4790e+01</td>
<td>6.94e+00</td>
<td>1.52e+01</td>
<td>1.96e+02</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>1.1876e+01</td>
<td>3.37e+00</td>
<td>2.62e+01</td>
<td>1.98e+01</td>
<td>60</td>
</tr>
<tr>
<td>8</td>
<td>1.1241e+01</td>
<td>1.32e+00</td>
<td>4.55e+01</td>
<td>2.83e+01</td>
<td>68</td>
</tr>
<tr>
<td>9</td>
<td>1.1192e+01</td>
<td>1.77e-01</td>
<td>7.87e+01</td>
<td>1.94e+01</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>9.3723e+00</td>
<td>0.00e+00</td>
<td>1.36e+02</td>
<td>4.36e+00</td>
<td>135</td>
</tr>
<tr>
<td>11</td>
<td>7.7548e+00</td>
<td>0.00e+00</td>
<td>2.36e+02</td>
<td>5.38e+02</td>
<td>74</td>
</tr>
<tr>
<td>12</td>
<td>6.0738e+00</td>
<td>0.00e+00</td>
<td>4.09e+02</td>
<td>1.68e+03</td>
<td>206</td>
</tr>
<tr>
<td>13</td>
<td>5.0490e+00</td>
<td>0.00e+00</td>
<td>7.09e+02</td>
<td>1.06e+03</td>
<td>98</td>
</tr>
<tr>
<td>14</td>
<td>1.4132e+00</td>
<td>1.77e-01</td>
<td>1.23e+03</td>
<td>2.28e+02</td>
<td>259</td>
</tr>
<tr>
<td>15</td>
<td>3.6147e+00</td>
<td>0.00e+00</td>
<td>2.13e+03</td>
<td>5.79e+02</td>
<td>128</td>
</tr>
<tr>
<td>16</td>
<td>3.1884e+00</td>
<td>0.00e+00</td>
<td>3.68e+03</td>
<td>4.69e+02</td>
<td>432</td>
</tr>
<tr>
<td>17</td>
<td>2.9705e+00</td>
<td>0.00e+00</td>
<td>6.38e+03</td>
<td>1.19e+03</td>
<td>262</td>
</tr>
<tr>
<td>18</td>
<td>2.8218e+00</td>
<td>0.00e+00</td>
<td>1.10e+04</td>
<td>7.54e+02</td>
<td>499</td>
</tr>
<tr>
<td>19</td>
<td>2.7625e+00</td>
<td>0.00e+00</td>
<td>1.91e+04</td>
<td>4.19e+03</td>
<td>232</td>
</tr>
<tr>
<td>20</td>
<td>2.7210e+00</td>
<td>0.00e+00</td>
<td>3.31e+04</td>
<td>3.98e+04</td>
<td>656</td>
</tr>
<tr>
<td>21</td>
<td>2.6978e+00</td>
<td>0.00e+00</td>
<td>5.74e+04</td>
<td>5.08e+04</td>
<td>869</td>
</tr>
<tr>
<td>22</td>
<td>2.6900e+00</td>
<td>0.00e+00</td>
<td>9.94e+04</td>
<td>5.80e+03</td>
<td>636</td>
</tr>
<tr>
<td>22</td>
<td>2.6900e+00</td>
<td>0.00e+00</td>
<td>9.94e+04</td>
<td>5.80e+03</td>
<td>4681</td>
</tr>
</tbody>
</table>

Formulation (5-23) requires far fewer multiplications than formulation (5-22). However, table 5-9 shows that using formulation (5-22) on test case 2 yields favourable results that
are comparable to those of all other test cases. The first 6 iterations are identical to those of table C-6. The $18^{th}$ iteration is the first to show serious effects of the implementation choice of $H$. Apparently, the round-off errors that are introduced with formulation (5-23) push some constraints too close to zero, causing large values of $\tilde{\epsilon}$. The other test cases also show effects of round-off errors, though they are not as severe as for test case 2.

We have shown that even test case 2 yields favourable results with the BFGS method, provided that the numerical effects of our implementation choices are kept under control. But can we rely on these results if the guaranteed accuracy is so low?

Before addressing this question we again discuss parameter $\mu$. As we already mentioned in the previous section, the results are very sensitive to this parameter, especially if it is chosen too large. The results of test case 4 with $\mu = 3$ are presented in table 5-10. We see that the deviation from the central path becomes problematic after the third iteration, i.e. large values of $\tilde{\epsilon}_k$ arise. Note that the results of the Newton-barrier method for test case 4 with $\mu = 3$ also show difficulty at the third iteration (table C-3). So we can conclude that for the BFGS-barrier method, it is essential that the parameter $\mu$ is not too large.

Without testing many more problems we cannot conclusively determine if the favourable results obtained with $\mu = \sqrt{3}$ are just lucky coincidences or not. However, our tests suggest that smaller $\mu$ result in higher accuracies. It is therefore not unreasonable to
presume that in general, the iterates $x_k$ converge to an acceptable estimate of the solution $x^*$ as long as $\mu$ is small enough.

5-7 Results

Even though the computation time of an iteration with the BFGS method is already drastically smaller than with Newton, it still requires $O(n^2)$ multiplications. An iteration of the L-BFGS algorithm requires $O(ln)$ multiplications, excluding function evaluations (see appendix A-2), where $l$ is the number of vectors stored. The L-BFGS method uses curvature information from only the $l$ most recent iterations to construct the Hessian approximation. So, the performance of the L-BFGS method with respect to the BFGS method depends on how relevant the curvature information from earlier iterations is.

In this section we present our final results for using quasi-Newton methods to compute the approximate central points $x_k$.

We denote the initial guess of $\nabla^2 g_t(x_j)$, by $H^0_j$. (For clarity we drop the subscript $k$.) First note that if for all outer iterations we choose $l \geq N_k$ and $H^0_j = H^0_k$, the L-BFGS and BFGS method are identical. To investigate how relevant the information from early iterations is, we apply the L-BFGS-barrier method on all test cases with $l = 10$, $l = 50$, $l = 100$ and $l = \infty$ respectively. For $j = nl, n \in \mathbb{N}$ we set $H^0_j = \beta I$, where $\beta$ is chosen according to (A-15). The results are presented in table 5-11. The last row shows the results of the BFGS-barrier method. Note that the runtime of an inner iteration varies with $l$, so we also compare the runtimes of the algorithm.

We see that the algorithm tends to be less reliable when $l$ is small. For test case 4, the values $l = 10$ and $l = 50$ even yield an unfeasible solution. However, comparing test cases 1,3 and 4, we see that the more complex the problem, the more influence increasing $l$ has on the final solution. In other words, for complex problems (test case 4) the curvature information from early iterations does seem relevant, whereas for easy problems (test case 1), it does not. We can go one layer deeper. It might be so that easy subproblems need less curvature memory than complex subproblems. Since for most test cases the subproblems become increasingly hard to solve as $t_k$ grows, we might benefit from increasing $l$ along with $k$. For test cases 1 and 3 this indeed yields very good results. However, test case 4 again puts a spanner in the works, because it already contains complex subproblems for small $t_k$. 
By choosing different parameters for each test case based on their performance, we can obtain results that are as good as, or even better than those corresponding to the interior point method. However, naturally, we cannot choose the parameters this way, since in general we do not have the required information prior to solving the problem. So we have to compromise with the results given in table 5-11.

<table>
<thead>
<tr>
<th>Test Case 1</th>
<th>$f(x)$</th>
<th>$\max c(x)$</th>
<th>$\tilde{\epsilon}$</th>
<th>$N$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l = 10$</td>
<td>1.4569e-01</td>
<td>0.00e+00</td>
<td>6.76e+03</td>
<td>155</td>
<td>2.9 s</td>
</tr>
<tr>
<td>$l = 50$</td>
<td>1.3916e-01</td>
<td>0.00e+00</td>
<td>8.73e+03</td>
<td>395</td>
<td>6.2 s</td>
</tr>
<tr>
<td>$l = 100$</td>
<td>1.3921e-01</td>
<td>0.00e+00</td>
<td>4.58e+03</td>
<td>454</td>
<td>11 s</td>
</tr>
<tr>
<td>$l = \infty$</td>
<td>1.3955e-01</td>
<td>0.00e+00</td>
<td>2.78e+03</td>
<td>513</td>
<td>16 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case 2</th>
<th>$f(x)$</th>
<th>$\max c(x)$</th>
<th>$\tilde{\epsilon}$</th>
<th>$N$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l = 10$</td>
<td>1.2902e+01</td>
<td>0.00e+00</td>
<td>2.83e+04</td>
<td>569</td>
<td>12 s</td>
</tr>
<tr>
<td>$l = 50$</td>
<td>4.4316e+00</td>
<td>7.52e-07</td>
<td>3.19e+12</td>
<td>1576</td>
<td>33 s</td>
</tr>
<tr>
<td>$l = 100$</td>
<td>3.1240e+00</td>
<td>6.37e-04</td>
<td>3.34e+11</td>
<td>2231</td>
<td>51 s</td>
</tr>
<tr>
<td>$l = \infty$</td>
<td>2.8451e+00</td>
<td>2.63e-07</td>
<td>1.44e+17</td>
<td>2127</td>
<td>64 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case 3</th>
<th>$f(x)$</th>
<th>$\max c(x)$</th>
<th>$\tilde{\epsilon}$</th>
<th>$N$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l = 10$</td>
<td>1.5869e-01</td>
<td>0.00e+00</td>
<td>4.09e+04</td>
<td>215</td>
<td>10 s</td>
</tr>
<tr>
<td>$l = 50$</td>
<td>1.3436e-01</td>
<td>0.00e+00</td>
<td>3.51e+03</td>
<td>677</td>
<td>28 s</td>
</tr>
<tr>
<td>$l = 100$</td>
<td>1.3086e-01</td>
<td>0.00e+00</td>
<td>1.41e+04</td>
<td>848</td>
<td>46 s</td>
</tr>
<tr>
<td>$l = \infty$</td>
<td>1.2836e-01</td>
<td>0.00e+00</td>
<td>8.42e+03</td>
<td>1197</td>
<td>60 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case 4</th>
<th>$f(x)$</th>
<th>$\max c(x)$</th>
<th>$\tilde{\epsilon}$</th>
<th>$N$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l = 10$</td>
<td>2.1012e-01</td>
<td>2.99e+01</td>
<td>9.32e+05</td>
<td>1001</td>
<td>17 m</td>
</tr>
<tr>
<td>$l = 50$</td>
<td>6.0536e-01</td>
<td>4.28e-01</td>
<td>3.47e+04</td>
<td>1612</td>
<td>23 m</td>
</tr>
<tr>
<td>$l = 100$</td>
<td>3.4368e-01</td>
<td>0.00e+00</td>
<td>8.63e+03</td>
<td>1531</td>
<td>23 m</td>
</tr>
<tr>
<td>$l = \infty$</td>
<td>3.1688e-01</td>
<td>0.00e+00</td>
<td>1.15e+04</td>
<td>4155</td>
<td>65 m</td>
</tr>
</tbody>
</table>

**Table 5-11:** Results of L-BFGS-Barrier method with varying $l$.

For test case 4 with $l = 100$, we have reached $f(x_k) \leq 0.37$ after 10 minutes, whereas the interior method resulted in $f(x_{198}) = 0.37$ in approximately 40 minutes. However,
choosing \( l = 100 \) is not necessarily the best choice for the other cases. Also, the choice of \( l \) can significantly influence the accuracy of the solution. In fact, it may happen that for some problems of type (5-1) the L-BFGS-barrier method is incapable of reaching an acceptable final solution. So again we cannot make conclusive statements regarding the reliability of the L-BFGS-barrier method.

### 5-8 Possible Improvements

Before we analyse the behaviour of the barrier method further, we comment on the efficiency of the implementation of the L-BFGS method. To that end we summarized some relevant information in table 5-12. The first and second column indicate the number of the test case and the value of \( l \) respectively. The third and fourth column show the total number of function evaluations and their total relative computation time (as a percentage of the total runtime of the algorithm). The fifth and sixth column present the same information for gradient evaluations. The last column shows the total time spent on function and gradient evaluations as a percentage of the total runtime of the algorithm.

<table>
<thead>
<tr>
<th>TC</th>
<th>( l )</th>
<th>Function Evaluations</th>
<th>Gradient Evaluations</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Total Number</td>
<td>Time</td>
<td>Total Number</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>446(m + 1)</td>
<td>59%</td>
<td>155(m + 1)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>762(m + 1)</td>
<td>48%</td>
<td>395(m + 1)</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>849(m + 1)</td>
<td>50%</td>
<td>454(m + 1)</td>
</tr>
<tr>
<td></td>
<td>∞</td>
<td>993(m + 1)</td>
<td>44%</td>
<td>541(m + 1)</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1941(m + 1)</td>
<td>65%</td>
<td>569(m + 1)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>4487(m + 1)</td>
<td>57%</td>
<td>1576(m + 1)</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>6084(m + 1)</td>
<td>53%</td>
<td>2231(m + 1)</td>
</tr>
<tr>
<td></td>
<td>∞</td>
<td>4414(m + 1)</td>
<td>38%</td>
<td>2406(m + 1)</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>870(m + 1)</td>
<td>69%</td>
<td>215(m + 1)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>1963(m + 1)</td>
<td>61%</td>
<td>677(m + 1)</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>3202(m + 1)</td>
<td>44%</td>
<td>848(m + 1)</td>
</tr>
<tr>
<td></td>
<td>∞</td>
<td>2825(m + 1)</td>
<td>46%</td>
<td>1481(m + 1)</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>4848(m + 1)</td>
<td>72%</td>
<td>1001(m + 1)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>6043(m + 1)</td>
<td>65%</td>
<td>1612(m + 1)</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>5037(m + 1)</td>
<td>59%</td>
<td>1531(m + 1)</td>
</tr>
<tr>
<td></td>
<td>∞</td>
<td>7158(m + 1)</td>
<td>35%</td>
<td>4222(m + 1)</td>
</tr>
</tbody>
</table>

**Table 5-12**: Number and relative computation time of function and gradient evaluations.
First note that the total number of gradient evaluations divided by $m + 1$, equals the total number of inner iterations (table 5-11). Indeed, at each iteration of a subproblem, the gradients of the $m$ constraints and the objective function have to be evaluated. The total number of function evaluations is higher. This is solely due to the linesearch that is done to find an appropriate stepsize $\alpha$. So the forth column with respect to the sixth column gives an indication of how much time is used on function evaluations to compute $\alpha$. We see that it indeed involves a large fraction of the total runtime, especially for small $l$. Therefore, assuming the function evaluations are done efficiently, reducing the number of function evaluations to compute $\alpha$ becomes a priority.

Currently, we simply do a linesearch to find a stepsize for which $c_i(x_j + \alpha p_j) \leq -\delta, i \in \{1, 2, ..., m\}, \delta > 0$. Each time there is an $i$ for which $c_i(x_j + \alpha p_j) > \delta$, we set $\alpha \leftarrow 0.5\alpha$ and compute all $m$ constraint evaluations to check whether the new stepsize yields a feasible iterate $x_{j+1} = x_j + \alpha p_j$. There are two ways to make this process more efficient. First, we can check after every constraint evaluation if $c_i(x_j + \alpha p_j) \leq -\delta$. If not, we can skip evaluating the rest of the constraints for this particular $\alpha$. Second, for violated linear constraints we can compute the allowed stepsize exactly, thereby avoiding unnecessary function evaluations. This stepsize can be computed in the same manner as a stepsize for active set methods, see appendix B. Unfortunately, this way of computing $\alpha$ is sensitive to rounding errors, so implementing this idea would require some modifications.

Naturally, the algorithm can be further improved by doing a more elaborate study on the different parameters. We already saw that the updating parameter $\mu$ has a significant influence on the efficiency of the algorithm. By introducing many more test cases it might be possible to categorize the minimization problems based on their performance with respect to different choices of $\mu$. That way, we might be able to avoid imposing the severe measures that are needed for complex problems on easy problems.

5-9 Analysis

One serious drawback of the logarithmic barrier method is the high nonlinearity of the barrier function $g_t(x)$. Since the level of nonlinearity increases along with $t_k$, the subproblems become harder to solve as $k$ increases. This is clearly reflected in the results of the Newton-barrier method. It seems that for large $t_k$, a quadratic Taylor polynomial cannot adequately capture the behavior of the barrier function.
We illustrate this phenomenon with an example (provided by [5]) of a 2-dimensional minimization problem, which consists of a quadratic objective function and simple linear constraints.

\[
\begin{align*}
\min_x \quad & (x_1 + 0.5)^2 + (x_2 - 0.5)^2 \\
\text{subject to} \quad & 0 \leq x \leq 1
\end{align*}
\] (5-24)

The barrier function corresponding to this example is given by

\[
g_t(x) = (x_1 + 0.5)^2 + (x_2 - 0.5)^2 - \frac{1}{t} \sum_{i=1}^{2} \left[ \log x_i + \log (1-x_i) \right]
\] (5-25)

Figure 5-2 shows the contours of function (5-25) for \( t = 100 \). The contours are almost straight along the left edge. As a result, Newton’s method may not show rapid convergence to the minimizer of \( g_t(x) \), except in a small neighbourhood of this point, where the contours are circular. Since outside this neighbourhood the convergence is likely to be slow, the number of iterations that is needed to minimize \( g_t(x) \) heavily depends on the initial guess of the solution.

![Figure 5-2: Contours of barrier function for a simple 2-D minimization problem.](image)

This is consistent with our analysis of the Newton-barrier method. We saw that for some test cases, a smaller \( \mu \) led to a significant decrease of the total number of inner iterations. Recall that \( \mu \) indirectly defines the quality of the initial guess of a subproblem. For small
\(\mu\), the subproblems generally start with an initial guess \(x_{k-1}\) that is relatively close to the solution \(x^*(t_k)\). So it appears that, similar to example (5-25), our test cases suffer from the logarithm in the barrier function \(g_t(x)\).

Example (5-25) also illustrates the drawbacks of applying quasi-Newton methods to solve the subproblems. The elongated nature of the contours in Figure 5-2 indicates bad scaling, to which quasi-Newton methods are sensitive. Indeed, the solution of the minimization problems yields \(x^* = [1, \frac{1}{2}]^T\), so only \(x_1\) is active at the solution. As a result, the barrier function (near its minimizer) is very sensitive to small changes of \(x_1\), whereas the influence of \(x_2\) is barely notable.

So the logarithm in the barrier function may cause the Hessian matrix \(\nabla^2 g_t(x)\) to contain a wide distribution of eigenvalues. In other words, the risk of the Hessian \(\nabla^2 g_t(x)\) being ill-conditioned is high. Ill-conditioning affects the performance of the quasi-Newton methods. This may explain why the values of \(\tilde{\epsilon}_k\) are much higher for the quasi-Newton methods than for Newton’s method. (Remember that \(\tilde{\epsilon}_k\) indicates the accuracy of \(x_k \approx x^*(t_k)\)). It may also explain the large impact of the implementation choice of expression (5-22). Indeed, round-off errors in ill-conditioned problems can influence the results significantly.

\section*{5-10 Conclusion}

In conclusion, both the Newton-barrier method and the quasi-Newton-barrier methods seem to suffer from the logarithm in the barrier function. Although the results of the L-BFGS-barrier method applied to the test cases are quite good (see table 5-11), the strong tendency to create ill-conditioned subproblems makes the barrier method generally unreliable.
Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) is an iterative method to solve problem (1-6):

\[
\begin{align*}
\min_x & \quad f(x) \\
\text{subject to} & \quad c_i(x) \leq 0, \quad i \in \{1, 2, \ldots, m\}
\end{align*}
\]  

(6-1)

Let \( x_k \) be the \( k^{th} \) estimate of the solution \( x^* \). An appropriate search direction \( p_x \) is found by solving a quadratic subproblem, which is assumed to reflect the original problem near \( x_k \) in some way. The step length \( \alpha \) is then determined by a so-called merit function \( \phi \), which measures the progress along the search direction. The next iterate is computed by setting \( x_{k+1} \leftarrow x_k + \alpha p_x \). This process is repeated until the KKT conditions corresponding to problem (6-1) are satisfied for some Lagrange multiplier \( \lambda^* \):

\[
\begin{align*}
\nabla_x \mathcal{L}(x^*, \lambda^*) & = 0 & \quad (6-2a) \\
c_i(x^*) & \leq 0, \quad i \in \{1, 2, \ldots, m\} & \quad (6-2b) \\
\lambda^*_i & \geq 0, \quad i \in \{1, 2, \ldots, m\} & \quad (6-2c) \\
\lambda^*_i c_i(x^*) & = 0, \quad i \in \{1, 2, \ldots, m\} & \quad (6-2d)
\end{align*}
\]

6-1 Theory

In this section we elaborate on the theoretical aspects of SQP. First we specify how the quadratic subproblems are formulated. Then we briefly discuss the use of a merit function. Finally we provide the reader with a practical algorithm followed by a brief discussion on its convergence properties.
6-1-1 Quadratic Subproblems

A quadratic problem consists of a quadratic objective function and linear constraints. A natural way to formulate the quadratic subproblem is to use Taylor polynomials to obtain

$$\begin{align*}
\min_p & \quad f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p \\
\text{subject to} & \quad c_i(x_k) + \nabla c_i(x_k)^T p \leq 0, \quad i \in \{1, 2, \ldots, m\}.
\end{align*}$$

However, when nonlinear constraints are involved, using (6-3) introduces problems. Suppose for example that a nonlinear convex constraint $c_i(x)$ is active at the $k^{th}$ iteration ($c_i(x_k) = 0$). Suppose as well that solving problem (6-3) results in a search direction $p_x$ along this linearized constraint. Then there is no positive step length $\alpha$ such that the next iterate $x_{k+1}$ is still feasible, as is illustrated in figure (6-1). Therefore, we have to somehow account for the nonlinearities of the constraints. Using a second-order approximation of the constraints, i.e. $c_i(x_k) + \nabla c_i(x_k)^T p + p^T \nabla^2 c_i(x_k)p \leq 0$, is not an option, because we want to keep the structure of a quadratic problem. In the literature (e.g in [5]), this difficulty is usually addressed by using the Lagrangian function, which is defined as in equation (6-4).

For problem (6-1) the Lagrangian is given by

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i c_i(x)$$

Incorporation of the respective Hessians of the nonlinear constraints into the objective function stimulates the terms $p^T \nabla^2 c_i(x_k)p$ to be small, thereby reducing the importance of the second-order terms in the approximation of the constraints.

A different motivation for using the Lagrangian is obtained by approaching the problem from a different point of view. We have already argued that solving problem (6-1) is equivalent to solving the corresponding KKT system (6-2). The first condition yields $\nabla_x \mathcal{L}(x^*, \lambda^*) = 0$, which, incidently, is equivalent to minimizing $\mathcal{L}(x, \lambda)$ with respect to $x$. It is therefore reasonable to use a second-order Taylor approximation of the Lagrangian as an objective function for the quadratic subproblem. However, since the first-order terms of the constraints are already incorporated as constraints in the quadratic subproblem, we leave these out of the objective. The quadratic subproblem becomes:

$$\begin{align*}
\min_p & \quad \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 \mathcal{L}(x_k, \lambda_k) p \\
\text{subject to} & \quad c_i(x_k) + \nabla c_i(x_k)^T p \leq 0, \quad i \in \{1, 2, \ldots, m\}
\end{align*}$$
The term \( f(x_k) \) was left out, since it has no effect on the minimization process.

The Hessian of the Lagrangian, \( \nabla^2 L(x_k, \lambda_k) \), can be expensive to compute. Many practical optimizers therefore use a symmetric, positive definite approximation \( B_k \approx \nabla^2 L(x_k, \lambda_k) \) instead of the exact Hessian [6]. The procedure that is used to determine \( B_k \) is inspired by the BFGS method, which is described in appendix A-2-1. The specific update formula is given by equation (A-11). In conclusion, the quadratic subproblem becomes

\[
\min_p \quad \nabla f(x_k)^T p + \frac{1}{2} p^T B_k p
\]

subject to \( c_i(x_k) + \nabla c_i(x_k)^T p \leq 0, \ i \in \{1, 2, \ldots, m\} \).

The solution of problem (6-6) consists of the optimal variable \( \tilde{p} \) and its corresponding Lagrange multiplier \( \tilde{\lambda} \). It is natural to choose \( p_x = \tilde{p} \) and \( p_\lambda = \tilde{\lambda} - \lambda_{k-1} \). In Appendix B we describe how problem (6-6) can be solved using an active set method.

It should be noted that taking a full step along \( p_x \) may lead to infeasible iterates or an increase of the objective function. Indeed, the quadratic subproblem may not be accurate far away from \( x_k \). To make sure each iterate is an improvement, a stepsize \( \alpha \) is introduced. This value is computed using a merit function.

### 6-1-2 Merit Function

The goal of a merit function is to measure the improvement towards the solution \( x^* \). To measure improvement, some kind of a trade-off between feasibility and the decrease in the objective function needs to be made. In this thesis the \( \ell_1 \) merit function is used:

\[
\phi(x; \mu) = f(x) + \mu \|c^+(x)\|_1
\]

(6-7)

were \( y^+ = \max(y, 0) \) and \( \mu \in \mathbb{R}^+ \) is called the penalty parameter, which defines the weight assigned to feasibility with respect to minimization of the objective. If \( \mu \) is large enough, local solutions of problem (6-1) coincide with local minima of \( \phi(x; \mu) \). In fact, it turns out that by choosing \( \mu > \|\lambda^*\|_\infty \), this desirable behaviour is guaranteed [5], [8]. A parallel goal is therefore to minimize the merit function.

Ideally we want the stepsize \( \alpha \in (0, 1] \) to be such that the decrease of the merit function is maximal. Unfortunately a full optimization over the interval \((0, 1]\) is too expensive. We therefore introduce the term sufficient decrease. A stepsize \( \alpha \) is accepted if the merit
function has produced a sufficient decrease over the step. A way to define this concept is that the decrease should not be too small with respect to the predicted change in the function. We may say that $\alpha$ is chosen such that

$$\phi(x + \alpha p_x; \mu) \leq \phi(x; \mu) + \eta \alpha D(\phi(x; \mu); p_x)$$

for some $\eta \in (0, 1)$. The term $D(\phi(x; \mu); p_x)$ denotes the directional derivative of the merit function along $p_x$ and is given by

$$D(\phi(x; \mu); p_x) = \nabla f(x_k)p_x - \mu \|c(x_k)^+\|_1$$

For a a full proof we refer to [5]. This sufficient decrease condition is called the Armijo condition [5]. The Armijo condition is illustrated in figure 6-2.

It can be shown that $p_x$ is a descent direction for the merit function if $p_x \neq 0$, $B_k$ is positive definite and $\mu > \|\tilde{\lambda}\|_\infty$, where $\tilde{\lambda}$ denotes the optimal Lagrange multiplier of the current quadratic subproblem [5]. We assume that the first two conditions are satisfied. First, if $p_x = 0$ the algorithm fails anyhow, since it would get stuck in an infinite loop, solving the same quadratic subproblem over and over again. Secondly, positive definiteness of $B_k$ is guaranteed because our problem is convex. This leaves the third condition, which can be satisfied by simply choosing $\mu$ such that $\mu > \|\tilde{\lambda}\|_\infty$.

**The Maratos Effect**

There is one major flaw in using the $\ell_1$ merit function to determine the stepsize $\alpha$. The $\ell_1$ merit function is not smooth at the feasible boundary, so it may happen that steps which make good progress towards the solution are rejected.

Note that the $\ell_1$ merit function equals the objective function in the feasible region and suddenly grows (with respect to the objective function) when constraints are violated, so the measured progress is discontinuous at the feasible boundary. As a result, the merit
function amplifies the difference in progress between complete feasibility and almost feasibility. In other words, slight violations of constraints are penalized relatively severely. This may cause unnecessary rejection of steps near the feasible boundary.

This phenomenon is called the Maratos effect [5]. There are several techniques to overcome the Maratos effect. An example is the watchdog strategy, which allows temporarily increase of the merit function as long as it has sufficiently decreased after a certain number of iterations.

### 6-1-3 Algorithm

The general idea of SQP has been covered. However, to implement a practical algorithm some further issues need to be discussed, in particular that of initial values.

Usually the initial estimate of the Hessian of the Lagrangian $B_0$ is set to a multiple of the identity matrix, $B_0 \leftarrow \beta I$, which reflects the scaling of the variables. Unfortunately there is no good general strategy to determine the scaling factor $\beta$. Therefore $B_0$ is often simply chosen as the identity matrix [5].

A good initial guess $x_0$ for the solution $x^*$ is equally hard to find. A possible strategy that is described in [9], is to find a good approximation $\tilde{x}$ of the solution of the related unconstrained problem and then solve the following linear problem

$$\min_x \|x - \tilde{x}\|_1$$
$$\text{subject to } Ax \leq b$$

(6-10)

where $A$ and $b$ correspond to the linear constraints of the original problem (6-1). The solution and corresponding Lagrange multipliers of problem (6-10) are then set as $x_0$ and $\lambda_0$ respectively. This way at least the linear constraints are satisfied at each iteration.

Finally, we need a good stopping criterion. A solution is found when the KKT conditions (6-2) are satisfied. The iterates $x_k$ and $\lambda_k$ should converge to $x^*$ and $\lambda^*$ respectively, so we terminate the algorithm when for some chosen tolerance $\epsilon > 0$

$$\max_i \left\{ \| \nabla_{x_i} \mathcal{L}(x_k, \lambda_k) \|_\infty, \| c(x_k) \|_\infty, \| \lambda_k c(x_k) \|_\infty \right\} \leq \epsilon$$

(6-11)

provided that all Lagrange multipliers are non-negative. The successive steps of the SQP method are summarized in algorithm 2.
Choose $x_0, \lambda_0, B_0, \nu \in (0,1)$ and $\omega > 0$;
Evaluate $\nabla f(x_0), c(x_0), \nabla c(x_0)$;
for $k = 0, 1, 2, \ldots$ do
  if convergence test is satisfied then
    stop with approximate solution $x^* = x_k$;
  end if
  Compute $p_x$ by solving quadratic subproblem (6-6) using the active-set method (appendix B);
  Let $\tilde{\lambda}$ be the corresponding Lagrange multiplier;
  Set $p_\lambda \leftarrow \tilde{\lambda} - \lambda_k$;
  Set $\mu_k \leftarrow \omega + \|\tilde{\lambda}\|_\infty$;
  Set $\alpha_k = 1$;
  while $\phi(x_k + \alpha_k p_x; \mu_k) \geq \phi(x_k; \mu_k) + \eta \alpha_k D(\phi(x_k; \mu_k); p_x)$ do
    Set $\alpha_k \leftarrow \tau \alpha_k$ for some $\tau \in (0,1)$;
  end while
  Set $x_{k+1} \leftarrow x_k + \alpha_k p_x$ and $\lambda_{k+1} \leftarrow \lambda_k + \alpha_k p_\lambda$;
  Evaluate $\nabla f(x_{k+1}), c(x_{k+1})$ and $\nabla c(x_{k+1})$;
  Compute $B_{k+1}$ using update formula (A-11);
end for
6-1-4 Convergence Properties

Unfortunately, applying the SQP algorithm as described in the previous section on an arbitrary minimization problem does not guarantee convergence from any starting point $x_0$. There have been many studies on how to modify the algorithm such that global convergence is guaranteed under certain conditions, see papers [10], [11], [12]. For us, the most concrete result is that the iterates $x_k, k > j$ converge R-superlinearly\(^1\) to the solution $x^*$ if the following conditions hold for some $j > 0$:

1. The exact Hessian of the Lagrangian at the solution $\nabla_x^2 \mathcal{L}(x^*, \lambda^*)$ is symmetric and positive definite.
2. The active set $\mathcal{A}$ is known.
3. The distances $\|x_j - x^*\|$ and $\|B_j - \nabla_x^2 \mathcal{L}(x^*, \lambda^*)\|$ are sufficiently small.
4. The merit function allows full step sizes.

In other words, if conditions 1-4 hold for some $j \geq 0$, then the sequence $\{x_k\}_{k>j}$ converges R-superlinearly to the solution [5], [8]. Note that the first condition always holds for convex problems.

If the second condition holds, the inequality constrained minimization problem becomes an equality constrained minimization problem, which is far easier to solve. For us this condition holds if the active sets of the quadratic subproblems $\mathcal{A}_k$ converge in a finite number of iterations to the active set of problem (6-1) $\mathcal{A}$, i.e. $\mathcal{A}_j = \mathcal{A}$ for some $j > 0$.

The third condition is to ensure that the quadratic subproblems model the behaviour of problem (6-1) near the solution adequately. If we use the exact Hessian $\nabla_x^2 \mathcal{L}(x_k, \lambda_k)$ and assume that $(x_k, \lambda_k) \rightarrow (x^*, \lambda^*)$ than the second condition should hold for some $j > 0$.

Finally, without the fourth condition, the Maratos effect could cause rejection of good steps, thereby making the second condition obsolete in the sense that the good quadratic representations of the original problem are not fully exploited. By using techniques such as the watchdog strategy (section 6-1-2), the Maratos effect can be avoided.

\[^1\]A sequence $\{x_k\}_{k>0}$ of positive real numbers converges R-superlinearly if $\lim \sqrt[k]{x_k} = 0$ [13].
6-2 Implementation

For the remainder of this chapter we refer to the iterations executed by the active set method to solve a quadratic subproblem as inner iterations. We refer to the computation of the iterates $x_k$ as outer iterations.

In this section we discuss the approach chosen to determine the initial values. Then we show how the runtime of the algorithm can be shortened by a clever use of past information and reformulation of the quadratic subproblem. Finally we explain how we handled some of the numerical problems that (inevitably) arose.

6-2-1 Initial Values

Some suggestions for the initial values have already been made in section 6-1-3. However, for the initial values $x_0$ and $\lambda_0$ we took a different approach. First of all, there is no good estimate for the solution of the related unconstrained problem available. Naturally we could put some effort into finding such an estimate, but then we are back to dealing with nonlinear functions. Furthermore, this estimate could still yield a poor approximation of the solution of the original problem due to a highly nonlinear objective function. We therefore solve a different problem instead:

$$
\min_{x,s} f(x) + \gamma s
$$

subject to $c_i(x) \leq s1, \ i \in \mathcal{N}$

where $\mathcal{N}$ is the set of indices corresponding to the linear constraints of problem (1-6) and $\gamma$ is some penalty parameter. Even though problem (6-12) is nonlinear, it is relatively easy to (partially) solve using the logarithmic barrier method (see chapter 5). The great advantage is that we do not need to solve the problem completely. Since the barrier method is a feasible method, i.e. all iterates are feasible, we can stop the algorithm when $s < 0$. The result yields an initial $x_0$ which is feasible with respect to the linear constraints, while keeping the objective function acceptably low.

For the initial approximation of the Hessian $B_0$ we do not differ from the suggestions given in section 6-1-3, so $B_0 = \beta I$. The only available relevant information is the scale of $x_0$, which gives us some sense of how large the step $p_x$ should be. In the unconstrained case $p_x$ would be computed by $p_x = -\frac{1}{\beta} \nabla f(x_0)$. We define $\beta$ such that $\frac{1}{\beta} \| \nabla f(x_0) \|_2 = \| x_0 \|_2$, so $\beta = \frac{1}{\| x_0 \|_2 \| \nabla f(x_0) \|_2}$. This way the step $p_x$ is in some sense of appropriate size.
6-2-2 Improving the Runtime

In general, the most time consuming part of the SQP method is solving the quadratic subproblems. Therefore, assuming the SQP algorithm converges nicely, its efficiency is directly related to the efficiency with which the quadratic subproblems are solved using the active set method. At each iteration of the active set method a guess of the active set is made, which we call the working set $W_k$. Subsequently the related equality constrained problem is solved, which provides the next guess $W_{k+1}$ of the active set. This process is continued until the active set $A$ is found. Note that solving an equality constrained quadratic problem is equivalent to solving a linear system, so once the active set is known, the solution is found in one iteration. In appendix B a detailed description of the active set method is given. We denote the initial working set and the active set of the $k^{th}$ subproblem by $W^0_k$ and $A_k$ respectively.

When no good estimate of the active set can be made, the number of iterations needed to solve problem (6-6) can be large. In fact, since only one index can be added or removed from the working set per iteration, the number of indices in the active set $A_k$ that differ from the initial working set $W^0_k$, i.e. the number of elements in the set $\{W^0_k \cup A_k\} \setminus \{W^0_k \cap A_k\}$, imposes a natural lower bound on the number of iterations. To improve the initial guess of the active set, warmstarting methods can be used [5]. For example, the active set of the $k^{th}$ subproblem can serve as the initial guess for the active set of the $k + 1^{th}$ subproblem, $W^0_{k+1} \leftarrow A_k$. Assuming global convergence of SQP, the initial guesses of the active sets should then improve as the solution is approached. Therefore, solving the quadratic subproblems can become economical as $k$ increases.

A drawback of the active set method is the need for a feasible initial point, i.e. the initial guess $p_0$ should satisfy $c_i(x_k) + \nabla c_i(x_k)^T p_0 = 0$, $i \in W^0_k$ and $c_i(x_k) + \nabla c_i(x_k)^T p_0 \leq 0$, $i \notin W^0_k$. When $x_k$ is feasible with respect to the current estimate of the active set $W^0_k$, we can simply choose $p_0 = 0$. However, this is not always the case, especially in early iterations. In fact, due to linearization of the constraints it is possible that a subproblem (6-6) is totally infeasible, i.e. there is no $p$ which satisfies the constraints. Consider for example the following problem:

$$\begin{align*}
    \min_x & \quad h(x) \\
    \text{subject to} & \quad x \geq 1 \quad (6-13) \\
    & \quad x^2 \leq -1
\end{align*}$$

Linearizing these constraints at $x_k = 1$ yields $p \geq 0$ and $p \leq -1$, which is inconsistent. To
overcome this difficulty we reformulate problem (6-6):

$$\min_p \quad \nabla f(x_k)^T p + \frac{1}{2} p^T B_k p + \tilde{\mu} \sum_{i \in I} s_i$$

subject to

$$c_i(x_k) + \nabla c_i(x_k)^T p \leq 0, \quad i \in \{1, 2, ..., m\} \setminus I$$

$$c_i(x_k) + \nabla c_i(x_k)^T p \leq s_i, \quad i \in I$$

$$s_i \geq 0, \quad i \in I$$

(6-14)

Here $\tilde{\mu}$ is some penalty parameter and $I$ is the set of indices corresponding to the constraints that are not satisfied at $x_k$. Problem (6-14) is always feasible and is equivalent to problem (6-6) if the latter was already feasible [5]. Furthermore, $p_0 = 0$ and $s_{0_i} = c_i(x_k)$ for $i \in I$ defines a feasible initial point for problem (6-14). Note that all constraints whose index $i \in I$, are active initially, so those indices that were not in the last active set $A_{k-1}$ should be added manually to the current initial working set $W_k^0$.

### 6-2-3 Numerical Problems

First of all, if the gradients of the constraints are not all linearly independent at $x_k$, quadratic subproblem (6-6) contains redundant or inconsistent constraints. The latter can be resolved using the technique described in the previous section. The former does not cause difficulties unless the set of linearized constraints contains duplicates. Indeed, problems only arise when the \textit{active} constraints are not linearly independent, and the index of a unique redundant constraint has no chance of entering the working set.

Unfortunately, probably due to round-off errors during the formulation of problem (6-1), the set of linearized constraints does sometimes contain duplicates. When non-unique constraints are added to the working set, the gradients of the \textit{active} constraints are no longer linearly independent, causing failure of the active set algorithm.

Since removing non-unique constraints beforehand is not straight forward, we applied a different strategy to deal with this problem. Each time the algorithm reports more than one, say two, blocking constraints, i.e the search direction intersects with two constraints at the same time, two indices are added to the working set. When this happens the rank of the active constraint matrix $A_k$ is checked. If the rank is smaller than the number of active constraints, we check if the two blocking constraints are identical. If so, one of them is set to zero and is removed from the working set, leaving the minimization problem uninfluenced by this redundant constraint.
Figure 6-3: The solid arrows show the steps taken, $\alpha p$. The dotted arrows show the search directions, $p$. Adding the wrong, in this case the blue, constraint to the working set, leads to violation of the green constraint, which leads to a negative stepsize $\alpha$, which leads to violation of the red constraint.

Round-off errors that occur during computation of the Lagrange multiplier can also cause an infinite loop, where one constraint is repeatedly added to the working set, just to be removed at the next iteration without taking a step, i.e. $\alpha = 0$. In this case, the algorithm is terminated and the current iterate $x_k$ is used as the search direction $p_k$ for the next outer iteration of the SQP algorithm.

6-3 Adjusting the Algorithm based on Results

Before we continue, we point out that test case 2 is not suitable for SQP as described in algorithm 2. Indeed, if all nonlinear constraints are inactive, the Hessian of the Lagrangian is zero, which makes the subproblem linear instead of quadratic. Since we have not implemented a linear solver, we do not consider test case 2 in this chapter.

In this section we argue that using the approximate Hessian $B_k$ is not beneficial for our problems. We also show that our test cases suffer from the Maratos effect (see section 6-1-2).

Results are generally presented in tables. The first column indicates the outer iterations, $k = 0, 1, 2, ..., K$, where $K$ is the total number of outer iterations. The second shows the value of the objective function $f(x_k)$. The third presents the maximum value among the
constraints, so \( \max c_i(x_k) \). The fourth column features the stepsize \( \alpha_k \), and the last shows the number of iterations to solve the \( k^{th} \) subproblem. The last row summarizes the results. In particular, the last row of the last column presents the total number of inner iterations, \( N = \sum_{k=1}^{K} N_k \).

### 6-3-1 Exact Hessian Versus Approximate Hessian

To get some sense of how well the matrices \( B_k \) represent the curvature information from problem (6-1), we first discuss some results where we use the exact Hessians \( \nabla^2 \mathcal{L}(x_k, \lambda_k) \) to formulate the quadratic subproblems. We then compare these results with those obtained from replacing the exact Hessian by \( B_k \).

#### Exact Hessian

We start with test case 3, see table 6-1. The first 4 columns imply that the subproblems are good representations of the original problem (6-1). Indeed, the iterates \( x_k \) seem to converge nicely towards the solution. In addition, the final value of the objective function \( f(x_8) = 0.12 \) is very good.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( f(x_k) )</th>
<th>( \max c(x_k) )</th>
<th>( \alpha_k )</th>
<th>( N_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0082e+00</td>
<td>1.90e+04</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.5742e-01</td>
<td>7.30e+03</td>
<td>1.0</td>
<td>530</td>
</tr>
<tr>
<td>2</td>
<td>1.2338e-01</td>
<td>1.98e+03</td>
<td>1.0</td>
<td>1343</td>
</tr>
<tr>
<td>3</td>
<td>1.1521e-01</td>
<td>9.31e+02</td>
<td>1.0</td>
<td>664</td>
</tr>
<tr>
<td>4</td>
<td>1.1637e-01</td>
<td>5.18e+02</td>
<td>0.9</td>
<td>298</td>
</tr>
<tr>
<td>5</td>
<td>1.1970e-01</td>
<td>1.23e+02</td>
<td>1.0</td>
<td>280</td>
</tr>
<tr>
<td>6</td>
<td>1.2089e-01</td>
<td>1.53e+01</td>
<td>1.0</td>
<td>96</td>
</tr>
<tr>
<td>7</td>
<td>1.2107e-01</td>
<td>5.59e-01</td>
<td>1.0</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>1.2108e-01</td>
<td>7.13e-04</td>
<td>1.0</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table 6-1:** SQP using the exact Hessian applied to test case 3.

Let us analyse the last column, which features the number of inner iterations per outer iteration. Apart from the first iteration, the number of inner iterations clearly decreases as \( k \) increases. This is consistent with what we discussed in section 6-2-2. Indeed, \( N_k \) measures the number of guesses the active set method makes before finding the active set. The results confirm that the initial guess \( \mathcal{W}_k^0 = \mathcal{A}_{k-1} \) of the active set \( \mathcal{A}_k \) of the \( k^{th} \)
subproblem becomes more accurate as $k$ grows. Consequently, the subproblems are easier to solve for large $k$.

Since there is no good initial estimate of the Lagrange multiplier $\lambda_0$ available, we use $B_0$ instead of $\nabla^2 \mathcal{L}(x_0, \lambda_0)$ for the first subproblem. As a result, the first subproblem does not represent the original problem (6-1) as well as the subsequent subproblems. In particular, its active set $\mathcal{A}_1$ may not yield a good initial guess for $\mathcal{A}_2$. We therefore ignore this information and let $\mathcal{W}_2^0$ be empty. This way, the second subproblems only requires $N_2 = 718$ iterations instead of $N_2 = 1343$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>max $c(x_k)$</th>
<th>$\alpha_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0082e+00</td>
<td>1.90e+04</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.5742e-01</td>
<td>7.30e+03</td>
<td>1.000</td>
<td>530</td>
</tr>
<tr>
<td>2</td>
<td>1.6185e-01</td>
<td>3.27e+03</td>
<td>1.000</td>
<td>415</td>
</tr>
<tr>
<td>3</td>
<td>1.5875e-01</td>
<td>1.68e+03</td>
<td>1.000</td>
<td>392</td>
</tr>
<tr>
<td>4</td>
<td>1.5666e-01</td>
<td>8.74e+02</td>
<td>1.000</td>
<td>267</td>
</tr>
<tr>
<td>5</td>
<td>1.5473e-01</td>
<td>4.63e+02</td>
<td>0.900</td>
<td>236</td>
</tr>
<tr>
<td>6</td>
<td>1.5192e-01</td>
<td>2.35e+02</td>
<td>1.000</td>
<td>190</td>
</tr>
<tr>
<td>7</td>
<td>1.4895e-01</td>
<td>1.29e+02</td>
<td>0.900</td>
<td>98</td>
</tr>
<tr>
<td>8</td>
<td>1.4629e-01</td>
<td>8.57e+01</td>
<td>0.656</td>
<td>135</td>
</tr>
<tr>
<td>9</td>
<td>1.4422e-01</td>
<td>6.47e+01</td>
<td>0.478</td>
<td>134</td>
</tr>
<tr>
<td>10</td>
<td>1.4250e-01</td>
<td>5.22e+01</td>
<td>0.387</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td></td>
<td>49</td>
<td>1.2674e-01</td>
<td>3.41e+00</td>
<td>0.052</td>
</tr>
<tr>
<td></td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>1.2292e-01</td>
<td>3.02e+00</td>
<td>0.590</td>
</tr>
<tr>
<td></td>
<td>81</td>
<td>1.2281e-01</td>
<td>3.02e+00</td>
<td>0.531</td>
</tr>
<tr>
<td></td>
<td>82</td>
<td>1.2271e-01</td>
<td>3.03e+00</td>
<td>0.729</td>
</tr>
<tr>
<td></td>
<td>83</td>
<td>1.2262e-01</td>
<td>2.33e+00</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>84</td>
<td>1.2253e-01</td>
<td>1.74e+00</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td></td>
<td>198</td>
<td>1.2108e-01</td>
<td>1.07e-03</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>199</td>
<td>1.2108e-01</td>
<td>9.39e-04</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>199</td>
<td>1.2108e-01</td>
<td>9.39e-04</td>
<td>21682</td>
</tr>
</tbody>
</table>

**Table 6-2:** SQP using the approximate Hessian $B_k$ applied to test case 3.
Approximate Hessian

We now present the results of replacing $\nabla^2 L(x_k, \lambda_k)$ with $B_k$. The results of test case 3 are shown in table 6-2. Clearly the total number of inner and outer iterations is a lot higher than for using the exact Hessian. We see that the merit function cuts down many steps. After the 6th iteration the step sizes gradually decrease to $\alpha_{49} = 0.052$. Then they gradually increase back to $\alpha_{83} = 1$. This might be due to the Maratos effect. We therefore also run the algorithm for fixed $\alpha_k = 1$. The results are shown in table 6-3.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\max c(x_k)$</th>
<th>$\alpha_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0082e+00</td>
<td>1.90e+04</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.5742e-01</td>
<td>7.30e+03</td>
<td>1</td>
<td>530</td>
</tr>
<tr>
<td>2</td>
<td>1.6185e-01</td>
<td>3.27e+03</td>
<td>1</td>
<td>415</td>
</tr>
<tr>
<td>3</td>
<td>1.5875e-01</td>
<td>1.68e+03</td>
<td>1</td>
<td>392</td>
</tr>
<tr>
<td>4</td>
<td>1.5666e-01</td>
<td>8.74e+02</td>
<td>1</td>
<td>267</td>
</tr>
<tr>
<td>5</td>
<td>1.5459e-01</td>
<td>4.63e+02</td>
<td>1</td>
<td>236</td>
</tr>
<tr>
<td>6</td>
<td>1.5190e-01</td>
<td>2.35e+02</td>
<td>1</td>
<td>157</td>
</tr>
<tr>
<td>7</td>
<td>1.4864e-01</td>
<td>1.30e+02</td>
<td>1</td>
<td>99</td>
</tr>
<tr>
<td>8</td>
<td>1.4510e-01</td>
<td>9.21e+01</td>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>9</td>
<td>1.4254e-01</td>
<td>6.57e+01</td>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>10</td>
<td>1.4024e-01</td>
<td>4.94e+01</td>
<td>1</td>
<td>68</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>159</td>
<td>1.2108e-01</td>
<td>1.34e-03</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>160</td>
<td>1.2108e-01</td>
<td>1.28e-03</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>161</td>
<td>1.2108e-01</td>
<td>1.01e-03</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>162</td>
<td>1.2108e-01</td>
<td>8.41e-04</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>162</td>
<td>1.2108e-01</td>
<td>8.41e-04</td>
<td>1</td>
<td>5667</td>
</tr>
</tbody>
</table>

Table 6-3: SQP with approximate Hessian $B_k$ and fixed step size $\alpha = 1$ applied to test case 3.

We see that the behaviour of SQP is better when full steps are taken. Not only is the total number of outer iterations smaller, the subproblems also require fewer iterations. The latter is because the warmstarting methods discussed in section 6-2-2, where the previous active set is used as an initial guess for the next active set, cannot be fully utilized if the step is cut down. Remember that active set methods require a feasible initial guess $p_k^0$ with respect to $W_k^0$. If $\alpha = 1$, it is guaranteed that the linear constraints are satisfied with respect to $A_{k-1}$ for $p_k^0 = 0$, whereas this is not necessarily true for smaller step sizes. As
6-3 Adjusting the Algorithm based on Results

a result, we cannot use information of the working set from previous iterations if $\alpha < 1$. What we can do is use the constraint evaluations at $x_k$ to define a good initial working set $W_{k+1}^0$. Unfortunately, due to round off errors active constraints are often not precisely zero, so this method is less efficient than warm starting methods.

**Comparison**

Although the results with the approximate Hessian $B_k$ are better for constant $\alpha_k = 1$, they still are poor in comparison to using the exact Hessian. So we need to weigh the expense of computing the exact Hessian against the total number of inner iterations. To that end we summarized some relevant information about the outer iterations in table 6-4. The time is given as a percentage of the total run time of the algorithm.

<table>
<thead>
<tr>
<th>Time</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>initializing</td>
<td>0.8%</td>
</tr>
<tr>
<td>solving subproblems</td>
<td>96%</td>
</tr>
<tr>
<td>computing step size</td>
<td>3%</td>
</tr>
<tr>
<td>computing Hessian</td>
<td>0.2%</td>
</tr>
</tbody>
</table>

*Table 6-4: Runtime distribution of outer iterations corresponding to test case 3 using the exact Hessian*

Only 0.2% of the computation time is used to compute the Hessian, so we can firmly state that the extra costs of computing this matrix are negligible in comparison to solving the subproblems. Since it is clear that using the exact Hessian yields by far the best results, we do not consider using $B_k$ any longer.

6-3-2 Merit Function

In the previous section we saw that for test case 3, fixing $\alpha_k = 1$ for all $k > 0$ led to better results when the approximate Hessian $B_k$ was used. In this section we draw the conclusion that SQP generally yields better results by always allowing full stepsizes.

We support this conclusion using test case 3, see table 6-5. The algorithm requires fewer inner iterations to obtain a (slightly) higher level of accuracy, while only one extra outer iteration is required.

A much stronger influence of using the merit function is detected by changing the initial value $x_0$. Recall that we compute $x_0$ by partly solving the following problem using the
Sequential Quadratic Programming

\[ k \mid f(x_k) \mid \max_i c(x_k) \mid N_k \]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0082e+00</td>
<td>1.91e+04</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.5742e-01</td>
<td>7.30e+03</td>
<td>530</td>
</tr>
<tr>
<td>2</td>
<td>1.2338e-01</td>
<td>1.98e+03</td>
<td>718</td>
</tr>
<tr>
<td>3</td>
<td>1.1521e-01</td>
<td>9.31e+02</td>
<td>664</td>
</tr>
<tr>
<td>4</td>
<td>1.1652e-01</td>
<td>5.24e+02</td>
<td>298</td>
</tr>
<tr>
<td>5</td>
<td>1.1936e-01</td>
<td>1.57e+02</td>
<td>203</td>
</tr>
<tr>
<td>6</td>
<td>1.2081e-01</td>
<td>2.16e+01</td>
<td>95</td>
</tr>
<tr>
<td>7</td>
<td>1.2107e-01</td>
<td>8.47e-01</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>1.2108e-01</td>
<td>1.81e-03</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>1.2108e-01</td>
<td>8.18e-09</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>1.2108e-01</td>
<td>8.18e-09</td>
<td>2530</td>
</tr>
</tbody>
</table>

Table 6-5: SQP applied to test case 3 with the exact Hessian, fixed \( \alpha = 1 \) and \( \gamma = 1 \).

barrier method (section 6-2-1):

\[
\min_x f(x) + \gamma s \\
\text{subject to } c_i(x) \leq s 1, \quad i \in \mathcal{N}
\]

(6-15)

The set \( \mathcal{N} \) contains the indices corresponding to the linear constraints of problem (6-1). The results given in table 6-1 correspond to \( \gamma = 1 \). By increasing the penalty parameter \( \gamma \), we put more weight on feasibility, thereby risking a higher value of \( f(x_0) \). On the other hand, the barrier algorithm is likely to be terminated after fewer iterations. So the penalty parameter \( \gamma \) involves a trade-off between the quality of the initial value \( x_0 \) and the cost of computing this point. Table 6-6 presents the results for test case 3 for \( \gamma = 10 \).

Similar to what we saw when we used the approximate Hessian \( B_k \), many steps are cut down. Fixing \( \alpha_k = 1 \) for all \( k > 0 \) yielded better results in that case. This was probably due to the Maratos effect and that the warmstarting method discussed in section 6-2-2 cannot (fully) be applied if \( \alpha < 1 \). The latter is also clearly reflected in table 6-7. Indeed, as soon as a full stepsize is made (\( k = 114 \)) the number of inner iterations required to solve the subproblems (\( N_{115} \)) drastically decreases. Table 6-7 shows the results for test case 3 with \( \gamma = 10 \) and fixed \( \alpha = 1 \).

When full stepsizes are taken, the number of outer iterations for test case 3 is more or less the same for both initial values, i.e. the convergence remains very satisfying. So it seems that our problems suffer from the Maratos effect. To support this theory we first elaborate on how the stepsize is computed.
### Table 6-6: SQP applied to test case 3 with the exact Hessian and $\gamma = 10$. The merit function is used to control the step sizes.

In section 6-1-2 we agreed that the stepsize $\alpha$ should satisfy

$$\phi(x + \alpha p_x; \mu) \leq \phi(x; \mu) + \eta \alpha D(\phi(x; \mu); p_x)$$  \hspace{1cm} (6-16)

An $\alpha$ which satisfies (6-16) is found by doing a simple linesearch. In practise it may happen that such an $\alpha$ is not found. Note that if $\alpha = 0$, the algorithm lands in an infinite loop, so we need $\alpha > 0$. To make sure that an $\alpha > 0$ is found, we terminate the linesearch if $\alpha < 10^{-10}$. When this happens, the $\alpha$ which resulted in the smallest difference between the left and right hand side of equation (6-16), is defined as the final stepsize.
Some stepsizes shown in table 6-6, including \( \alpha_8 \) where the feasible boundary is crossed, did not satisfy equation (6-16), i.e. none of the tried stepsizes \( \alpha \in \left[ 10^{-10}, 1 \right] \) led to a sufficient decrease of the merit function. This is because the initial value \( x_0 \) is feasible, so the merit function would not allow crossing the feasible boundary unless the decrease of the objective function is significant, which does not seem to be the case. So the iterates closely follow the only nonlinear (quadratic) constraint of test case 3. Comparing table 6-6 with table 6-7, we see that allowing the feasible boundary to be crossed yields far better convergence. This suggest that, although the merit function does not decrease 'sufficiently', the iterates \( x_k \) do approach the solution \( x^* \) at each iteration. In fact, they approach it fast. This phenomenon fits the description of the Maratos effect perfectly.

We have tested all test cases for different initial values and they all yield better results when full step sizes are taken. We therefore fix \( \alpha_k = 1 \) for all \( k \).

### 6-4 Results

In this section we provide the final results for the three test cases. We use \( \gamma = 1 \) and \( \alpha = 1 \). The results are presented in tables 6-8 - 6-10, where the total runtime is presented in the last row.
**Table 6-8:** SQP applied to test case 1 with the exact Hessian, fixed $\alpha = 1$ and $\gamma = 1$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\text{max } c(x_k)$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.2219e+00</td>
<td>-5.67e-01</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.1045e-01</td>
<td>0.00e+00</td>
<td>1024</td>
</tr>
<tr>
<td>2</td>
<td>1.0234e-01</td>
<td>0.00e+00</td>
<td>4739</td>
</tr>
<tr>
<td>2</td>
<td>1.0234e-01</td>
<td>0.00e+00</td>
<td>5763</td>
</tr>
</tbody>
</table>

Total Time: 641 sec

**Table 6-9:** SQP applied to test case 3 with the exact Hessian, fixed $\alpha = 1$ and $\gamma = 1$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\text{max } c(x_k)$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0082e+00</td>
<td>1.91e+04</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.5742e-01</td>
<td>7.30e+03</td>
<td>530</td>
</tr>
<tr>
<td>2</td>
<td>1.2338e-01</td>
<td>1.98e+03</td>
<td>718</td>
</tr>
<tr>
<td>3</td>
<td>1.1521e-01</td>
<td>9.31e+02</td>
<td>664</td>
</tr>
<tr>
<td>4</td>
<td>1.1652e-01</td>
<td>5.24e+02</td>
<td>298</td>
</tr>
<tr>
<td>5</td>
<td>1.1936e-01</td>
<td>1.57e+02</td>
<td>203</td>
</tr>
<tr>
<td>6</td>
<td>1.2081e-01</td>
<td>2.16e+01</td>
<td>95</td>
</tr>
<tr>
<td>7</td>
<td>1.2107e-01</td>
<td>8.47e-01</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>1.2108e-01</td>
<td>1.81e-03</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>1.2108e-01</td>
<td>8.18e-09</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>1.2108e-01</td>
<td>8.18e-09</td>
<td>2530</td>
</tr>
</tbody>
</table>

Total Time: 291 sec

**Table 6-10:** SQP applied to test case 4 with the exact Hessian, fixed $\alpha = 1$ and $\gamma = 1$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\text{max } c(x_k)$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.1583e+00</td>
<td>1.22e+05</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4.6972e-01</td>
<td>5.05e+04</td>
<td>779</td>
</tr>
<tr>
<td>2</td>
<td>3.3372e-01</td>
<td>1.47e+04</td>
<td>1063</td>
</tr>
<tr>
<td>3</td>
<td>2.9086e-01</td>
<td>8.84e+03</td>
<td>891</td>
</tr>
<tr>
<td>4</td>
<td>2.9164e-01</td>
<td>7.13e+03</td>
<td>496</td>
</tr>
<tr>
<td>5</td>
<td>3.0349e-01</td>
<td>3.15e+03</td>
<td>521</td>
</tr>
<tr>
<td>6</td>
<td>3.1224e-01</td>
<td>9.28e+02</td>
<td>257</td>
</tr>
<tr>
<td>7</td>
<td>3.1538e-01</td>
<td>1.75e+02</td>
<td>121</td>
</tr>
<tr>
<td>8</td>
<td>3.1593e-01</td>
<td>9.86e+00</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>3.1595e-01</td>
<td>3.86e-02</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>3.1595e-01</td>
<td>5.78e-07</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>3.1595e-01</td>
<td>5.78e-07</td>
<td>4177</td>
</tr>
</tbody>
</table>

Total Time: 296 min

**Table 6-10:** SQP applied to test case 4 with the exact Hessian, fixed $\alpha = 1$ and $\gamma = 1$.  

63
Comparing these results with the interior point method, we see that our solution is generally better, but the total runtime is far worse. So the SQP algorithm needs significant improvements. To that end we analyse the behaviour of SQP applied to the test cases in the next section.

6-5 Analysis

In this section we analyse the general behaviour of SQP applied to our test cases. Since solving the quadratic subproblems can be treated as a black box, we analyse the outer iterations and inner iterations separately.

6-5-1 Outer Iterations

In this section we investigate the outer iterations executed by the SQP algorithm for test cases 1, 3 and 4. We mainly focus on the number of outer iterations \( K \), which we refer to as the convergence of SQP.

The results in the previous section (tables 6-8 - 6-10) show that the total number of outer iterations \( K \) is small for all test cases. The results presented in table 6-7 suggest that the convergence of SQP is insensitive to the choice of the initial value \( x_0 \). Indeed, for test case 3 the total number of outer iterations equals \( K = 9 \) for \( \gamma = 1 \), and \( K = 12 \) for \( \gamma = 10 \), which we do not consider a significant difference, in the sense that for both initial values the convergence is good. We have tested for all test cases that indeed the total number of outer iterations \( K \) remains low for remote choices of \( \gamma \).

by considering the solver of the quadratic subproblems as a black box, the only choices for the user are the initial values and \( \epsilon \). The latter does not influence the algorithmic process, and \( B_0 \) and \( \lambda_0 \) only affect the first iteration. So it seems that the convergence of SQP is insensitive to parameter choices, which makes the algorithm very dependable in that sense. We therefore seek possibilities for improvement elsewhere.

6-5-2 Inner Iterations

We have established that the convergence of the SQP algorithm is very good. However, to decide if the algorithm is time efficient, we also need to take the solver of the subproblems
into account, in particular the number of inner iterations. We use $j$ to indicate the iterations executed by the active set method. Recall that subproblems are of the form

$$\min_p \frac{1}{2} p^T G p + c^T p$$

subject to $A p \leq b$ \hspace{1cm} (6-17)

A great advantage of the active set method applied to convex quadratic problems is that (in theory) finding the exact solution in a finite number of iterations is guaranteed. Indeed, if the active set is known, the solution is found in one iteration, and there is only a finite number of possible active sets.

At each iteration, the active set is guessed, $\mathcal{W}_j$, and the solution $p_j$ corresponding to $\mathcal{W}_j$ is computed. Then it is checked whether $p_j$ is the solution problem (6-17), i.e. whether $\mathcal{W}_j$ is the active set. If so, the algorithm terminates with solution $p^* = p_j$ and active set $\mathcal{A} = \mathcal{W}_j$. If not, another (clever) guess of the active set ($\mathcal{W}_{j+1}$) is made. This process continues until the active set is found. So the number of iterations equals the number of guesses needed to find the active set.

A drawback of the active set method is that subsequent guesses of the active set can only differ in one index. More specifically, only one index can be added or removed from the working set at each iteration. So the efficiency of the active set method strongly depends on the initial guess $\mathcal{W}_0$. If $\mathcal{W}_0$ is far from the active set, the active set method needs many iterations to find the solution, whereas if the initial guess $\mathcal{W}_0$ is good, the algorithm is done in just a few iterations.

We clearly saw this in the results for all the test cases (tables 6-8 - 6-10). Indeed, $\mathcal{W}_{k+1} = \mathcal{A}_k$ and $\mathcal{A}_k \to \mathcal{A}$, so the fact that $N_k$ clearly decreases as $k$ increases confirms that the quality of the initial guess is key to solving the subproblems efficiently. (In the last comment we used $\mathcal{A}$ to denote the active set of the original problem (6-1), and we used $\mathcal{W}_k^0$ and $\mathcal{A}_k$ to denote the initial guess and the active set of the $k^{th}$ subproblem respectively.)

### 6-6 Possible Improvements

We first discuss possible improvements based on the analysis conducted in the previous section. We then show how we can significantly improve the runtime of the active set method by adjusting its implementation.
6-6-1 Number of Inner Iterations

Clearly, the bottleneck of the SQP algorithm with the active set method, lies in solving the first few subproblems, where the initial guess of the active set is poor, causing large values of \( N_k \). In this section we try to partly circumvent this problem. As usual we denote the outer iterations by \( k \).

The active sets of the first few subproblems \( A_k \), are not necessarily good estimates of the active set of the original problem (6-1), \( A \). However, they do converge to \( A \), i.e. \( A_k \to A \). So, do we generally want a good initial guess for \( A_k \) or \( A \)? This really depends on the available information.

If we are confident that we can make a fairly good estimate of \( A \), we can choose to solve the first few subproblems according to this estimate. In other words, we fix the working set for the first few outer iterations, so that the active set method requires only one (or, when no feasible initial value \( p_0 \) is available, a few) iteration(s) to solve the quadratic subproblems.

A consequence of this approach is that linear constraints whose indices are not in the working set are ignored and may therefore be violated. When all constraints are considered during the solving of the subproblems, this does not happen, i.e. all linear constraints remain lower or equal to zero. Recall that the active set method requires a feasible initial value \( p_0 \), which is harder to find when many constraints are violated. In fact, the chosen working set may be infeasible with respect to the full problem, in the sense that no \( p_0 \) in the feasible region exists for which all constraints corresponding to the working set are active. In this case the working set would have to be adjusted. Since there is no straightforward way to do this, we would have to reset the working set entirely. In conclusion, this approach only potentially works if the initial guess of the active set \( A \) is very good, which is not the case for us.

If we do not have the resources to make a very good estimate of \( A \), it is beneficial to find a good estimate of \( A_k \). In this case we focus on fully solving the subproblems, and finding the active set \( A \) through the sequence \( \{A_k\}_{k>0} \).

For the first few outer iterations we cannot make good guesses of both \( A \) and \( A_k \). So we try yet a different approach. The subproblems are probably coarser models of the original problem far away from the solution than close to the solution, so it might be senseless to fully solve the first few subproblems. To that end, we terminate the active set algorithm.
when an imposed upper bound on the number of iterations, say $N_k \leq 500$, is reached. The results for test case 3 are given in table 6-11.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\max c(x_k)$</th>
<th>$N_k$</th>
<th>$f(x_k)$</th>
<th>$\max c(x_k)$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0082e+00</td>
<td>1.91e+04</td>
<td>-</td>
<td>1.0082e+00</td>
<td>1.91e+04</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.5742e-01</td>
<td>7.30e+03</td>
<td>500</td>
<td>1.5742e-01</td>
<td>7.30e+03</td>
<td>530</td>
</tr>
<tr>
<td>2</td>
<td>1.2347e-01</td>
<td>1.99e+03</td>
<td>500</td>
<td>1.2338e-01</td>
<td>1.98e+03</td>
<td>718</td>
</tr>
<tr>
<td>3</td>
<td>1.1525e-01</td>
<td>9.52e+02</td>
<td>500</td>
<td>1.1521e-01</td>
<td>9.31e+02</td>
<td>664</td>
</tr>
<tr>
<td>4</td>
<td>1.1643e-01</td>
<td>5.38e+02</td>
<td>500</td>
<td>1.1652e-01</td>
<td>5.24e+02</td>
<td>298</td>
</tr>
<tr>
<td>5</td>
<td>1.1935e-01</td>
<td>1.57e+02</td>
<td>302</td>
<td>1.1936e-01</td>
<td>1.57e+02</td>
<td>203</td>
</tr>
<tr>
<td>6</td>
<td>1.2082e-01</td>
<td>2.13e+01</td>
<td>95</td>
<td>1.2081e-01</td>
<td>2.16e+01</td>
<td>95</td>
</tr>
<tr>
<td>7</td>
<td>1.2107e-01</td>
<td>8.37e-01</td>
<td>14</td>
<td>1.2107e-01</td>
<td>8.47e-01</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>1.2108e-01</td>
<td>1.80e-03</td>
<td>3</td>
<td>1.2108e-01</td>
<td>1.81e-03</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>1.2108e-01</td>
<td>7.97e-09</td>
<td>2417</td>
<td>1.2108e-01</td>
<td>8.18e-09</td>
<td>2530</td>
</tr>
</tbody>
</table>

Table 6-11: SQP applied to test case 3 with the exact Hessian, fixed $\alpha = 1$ and $\gamma = 1$. The left part presents the results for partially solving the subproblems, $N_k \leq 500$, and the right part for fully solving the subproblems.

Comparing the left-hand side table with the right-hand side table, we see that the total number of inner iterations is reduced by $2530 - 2417 = 113$. Though, this is good news, we might have expected a larger reduction. Indeed, the total number of iterations exceeding $N_k = 500$ in the right table is 412. It seems that the quadratic subproblems model the original problem very well, in the sense that finding the exact solution is not irrelevant even for the first few subproblems. Tables 6-12 and 6-13 show the effect of bounding $N_k \leq 500$ for test cases 1 and 4 respectively.

Comparing the left-hand-side tables with the right-hand-side tables, we see that enforcing $N_k \leq 500$ yields better results for test case 1, but poorer results for test case 4. Enforcing $N_k \leq 250$ accentuates this effect, i.e. the results for test case 1 become even better and for test case 4 worse. So it seems that terminating the active set algorithm early, is not always wise. Or at least the bound on the number of inner iterations $N_k$ should be chosen according to certain characteristics of the minimization problems, perhaps the number of free dimensions $n$. Realizing that the gain is not significant, we do not investigate this suggestion further and continue our analysis using the results presented in the right-hand-side tables, i.e. the results where all the subproblems were fully solved.
In conclusion, the SQP algorithm is still far too slow to replace the interior point method, unless the performance of the active set method can be improved significantly. Unfortunately, we have not found a successful strategy to significantly reduce the total number of inner iterations. We therefore seek possible improvements of the implementation of the active set method instead.

Table 6-12: SQP applied to test case 1 with the exact Hessian, fixed $\alpha = 1$ and $\gamma = 1$. The left part presents the results for partially solving the subproblems, $N_k \leq 500$, and the right part for fully solving the subproblems.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\max c(x_k)$</th>
<th>$N_k$</th>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\max c(x_k)$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.2219e+00</td>
<td>-5.67e-01</td>
<td>-</td>
<td>0</td>
<td>1.2219e+00</td>
<td>-5.67e-01</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.1195e-01</td>
<td>0.00e+00</td>
<td>500</td>
<td>1</td>
<td>1.1045e-01</td>
<td>0.00e+00</td>
<td>1024</td>
</tr>
<tr>
<td>2</td>
<td>1.0716e-01</td>
<td>0.00e+00</td>
<td>500</td>
<td>2</td>
<td>1.0234e-01</td>
<td>0.00e+00</td>
<td>4739</td>
</tr>
<tr>
<td>3</td>
<td>1.0418e-01</td>
<td>0.00e+00</td>
<td>500</td>
<td>3</td>
<td>1.0368e-01</td>
<td>0.00e+00</td>
<td>2000</td>
</tr>
<tr>
<td>4</td>
<td>1.0368e-01</td>
<td>0.00e+00</td>
<td>2000</td>
<td>2</td>
<td>1.0234e-01</td>
<td>0.00e+00</td>
<td>5763</td>
</tr>
</tbody>
</table>

Table 6-13: SQP applied to test case 4 with the exact Hessian, fixed $\alpha = 1$ and $\gamma = 1$. The left part presents the results for partially solving the subproblems, $N_k \leq 500$, and the right part for fully solving the subproblems.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\max c(x_k)$</th>
<th>$N_k$</th>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\max c(x_k)$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.1583e+00</td>
<td>1.22e+05</td>
<td>-</td>
<td>0</td>
<td>1.1583e+00</td>
<td>1.22e+05</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.2502e-01</td>
<td>4.67e+04</td>
<td>500</td>
<td>1</td>
<td>4.6972e-01</td>
<td>5.05e+04</td>
<td>779</td>
</tr>
<tr>
<td>2</td>
<td>4.0344e-01</td>
<td>1.19e+04</td>
<td>500</td>
<td>2</td>
<td>3.3372e-01</td>
<td>1.47e+04</td>
<td>1063</td>
</tr>
<tr>
<td>3</td>
<td>3.0811e-01</td>
<td>7.29e+03</td>
<td>500</td>
<td>3</td>
<td>2.9086e-01</td>
<td>8.84e+03</td>
<td>891</td>
</tr>
<tr>
<td>4</td>
<td>2.8381e-01</td>
<td>9.84e+03</td>
<td>500</td>
<td>4</td>
<td>2.9164e-01</td>
<td>7.13e+03</td>
<td>496</td>
</tr>
<tr>
<td>5</td>
<td>3.0053e-01</td>
<td>3.74e+03</td>
<td>500</td>
<td>5</td>
<td>3.0349e-01</td>
<td>3.15e+03</td>
<td>521</td>
</tr>
<tr>
<td>6</td>
<td>3.1144e-01</td>
<td>1.31e+03</td>
<td>500</td>
<td>6</td>
<td>3.1224e-01</td>
<td>9.28e+02</td>
<td>257</td>
</tr>
<tr>
<td>7</td>
<td>3.1456e-01</td>
<td>4.50e+02</td>
<td>500</td>
<td>7</td>
<td>3.1538e-01</td>
<td>1.75e+02</td>
<td>121</td>
</tr>
<tr>
<td>8</td>
<td>3.1516e-01</td>
<td>2.28e+02</td>
<td>500</td>
<td>8</td>
<td>3.1593e-01</td>
<td>9.86e+00</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>3.1508e-01</td>
<td>3.49e+01</td>
<td>171</td>
<td>9</td>
<td>3.1595e-01</td>
<td>3.86e-02</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>3.1598e-01</td>
<td>1.38e-01</td>
<td>15</td>
<td>10</td>
<td>3.1595e-01</td>
<td>5.78e-07</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>3.1598e-01</td>
<td>3.91e-06</td>
<td>11</td>
<td>11</td>
<td>3.1595e-01</td>
<td>5.78e-07</td>
<td>4177</td>
</tr>
</tbody>
</table>

In conclusion, the SQP algorithm is still far too slow to replace the interior point method, unless the performance of the active set method can be improved significantly. Unfortunately, we have not found a successful strategy to significantly reduce the total number of inner iterations. We therefore seek possible improvements of the implementation of the active set method instead.
6-6 Possible Improvements

6-6-2 Implementation of Active Set Method

We have established that almost the entire computation time is spent on solving the sub-problems. We therefore investigate the bottlenecks of the active set method applied to the subproblems of test case 3 with parameters as in table 6-9. The information is summarized in table 6-14. We want to stress that the given number of calls and relative runtimes correspond to the sum of all 9 subproblems of test case 3. However, for clarity we do not indicate the outer iterations when we consider inner iterations. For example, we denote the $j^{th}$ search direction of the $k^{th}$ subproblem by $p_j$ instead of $p_j^k$.

<table>
<thead>
<tr>
<th>Calls</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search Direction</td>
<td>2530  86%</td>
</tr>
<tr>
<td>Step Size</td>
<td>1880  10%</td>
</tr>
<tr>
<td>Lagrange Multipliers</td>
<td>650  4%</td>
</tr>
</tbody>
</table>

Table 6-14: Information regarding the implementation of the active set method when applied to the subproblems of test case 3

The first row gives the number of search directions computed and their total relative computation time. The search direction $p_j$ can be found by solving a linear system, see appendix B, system (B-7):

\[ \begin{bmatrix} G & A_j^T \\ A_j & 0 \end{bmatrix} \begin{bmatrix} p_j \\ \lambda_j \end{bmatrix} = - \begin{bmatrix} Gx_j + c \\ 0 \end{bmatrix} \]

(6-18)

where $G = \nabla^2 L(x_k, \lambda_k)$, $c = \nabla f(x_k)$ and $A_j \in \mathbb{R}^{m_j \times n}$ is the matrix containing the gradients of the $m_j$ constraints active at the $j^{th}$ iteration. There are many possible methods to solve system (6-18). We use a variant of the null-space method. By a clever substitution of variables, the following two systems are solved instead of system (6-18):

\[ (Q_2^T G Q_2) p_j = -Q_2^T (G x_j + c) \]  
\[ (AQ_1)^T \lambda_j = Q_1^T [G(x_j + p_j) + c] \]

(6-19a) (6-19b)

where $Q_1 \in \mathbb{R}^{n \times m_j}$ and $Q_2 \in \mathbb{R}^{n \times (n-m_j)}$ result from the QR factorization of $A_j^T$:

\[ A_j^T = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} \]

(6-20)

[5]. Formulation (6-19) allows the search direction $p_j$ to be computed separately from the Lagrange multiplier $\lambda_j$. Note that the Lagrange multiplier is only used to check whether
an iterate is the solution when $p_j = 0$, so for $p_j \neq 0$, system (6-19b) need not be solved.

The second row of table 6-14 shows information about computation of the stepsize $\alpha_j$, see equation (B-8). The third row shows the same information about the computation of the Lagrange multipliers $\lambda_j$, which are found by solving system (6-19b).

First note that the number of search directions $p_j$ computed (2530) equals the total number inner iterations (table 6-9). Second, the sum of the number of stepsizes and Lagrange multipliers computed also equals the total number of inner iterations ($650 + 1880 = 2530$). Indeed, at each iteration a search direction $p_j$ is computed. Either $p_j = 0$, in which case the Lagrange multiplier $\lambda_j$ is computed, or $p_j \neq 0$, in which case the stepsize $\alpha_j$ is computed. From table 6-14 we see that finding the search direction is by far the most time consuming, so we focus on improving the efficiency of computing $p_j$.

**Implementation of Search Direction**

In table 6-15 we present the time distribution of the computation of the search direction $p_j$. For the $QR$ factorization and solving system (6-19a) using a Cholesky factorization, we use standard Matlab applications.

<table>
<thead>
<tr>
<th>Time Distribution</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>QR Factorization</td>
<td>70%</td>
</tr>
<tr>
<td>Multiplication $Q_2^TGQ_2$</td>
<td>23%</td>
</tr>
<tr>
<td>Solving System (6-19a)</td>
<td>5.4%</td>
</tr>
<tr>
<td>Multiplication $Q_2^T(Gx_j + c)$</td>
<td>1.6%</td>
</tr>
<tr>
<td>Total Time: 253s</td>
<td></td>
</tr>
</tbody>
</table>

**Table 6-15: Runtime distribution of computation of search direction.**

The most time is spent on the $QR$ factorization. This is good news, because here lies the main potential for improvement. Note that matrix $A_j$ only gains or loses at most one row at each iteration. Techniques exist to use the $QR$ factorization of $A_{j-1}^T$ to compute the $QR$ factorization of $A_j^T$. These updates are relatively cheap. The cost of computing the $QR$ factorization from scratch is $\mathcal{O}(n^2m_j)$, whereas clever updating techniques can reduce the costs to $\mathcal{O}(n(n - m_j))$. (Recall that $m_j$ denotes the number of indices in $\mathcal{W}_j$.) This also implies that the larger the number of active constraints, the more time we can save on $QR$ factorizations [5].
Computing $Q_2^T G Q_2$ also requires a relevant amount of time. We call this term the reduced Hessian. Since $Q_2 \in \mathbb{R}^{n \times (n-m_j)}$, the cost of computing the reduced Hessian increases as the number of active constraints lowers. So, if the $QR$ factorization is cheap, computing the reduced Hessian is expensive, and the other way around. It is therefore very beneficial to update the reduced Hessian in a similar way as the $QR$ factorization. Suppose that we have the Cholesky factorization of the current reduced Hessian, written as $Q_2^T G Q_2 = LL^T$. Then a series of inexpensive, elementary operations can be used to transform the Cholesky factor $L$ into the new factor $\tilde{L}$ for the new reduced Hessian $\tilde{Q}_2^T G \tilde{Q}_2$ [5].

We have not implemented these suggestions, but in the next section we present some numbers showing the potential of the updating techniques described in this section.

### 6-7 Best Case Scenario

For test case 3, the $QR$ factorization and computing the Reduced Hessian take up 81\% of the total runtime. So, by applying the update techniques as described in section 6-6-2, a significant relative amount of time can be saved. In this section we present a rough indication of potential runtimes for an improved implementation.

In table 6-16 we have summarized the relative and absolute time spent on $QR$ factorizations and computation of the reduced Hessians for all test cases. The relative time is with respect to the total runtime of the SQP algorithm. The last row of table 6-16 presents the total runtime without $QR$ factorizations and computation of reduced Hessians. This gives an indication of the best case scenario total computation times for a realistically improved implementation.

<table>
<thead>
<tr>
<th></th>
<th>Test Case 1</th>
<th>Test Case 3</th>
<th>Test Case 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$QR &amp; Q_2^T G Q_2$</td>
<td>86% 553 sec</td>
<td>81% 235 sec</td>
<td>78% 231 min</td>
</tr>
<tr>
<td>Rest</td>
<td>14% 88 sec</td>
<td>19% 56 sec</td>
<td>22% 65 min</td>
</tr>
</tbody>
</table>

**Table 6-16:** Potential runtimes.

We see that for all test cases the best case scenario runtimes are still not as good as those obtained from the interior point method. However, we must note that the SQP algorithm yields more accurate results, i.e. the final value of the objective value is for all test cases lower for SQP than for the interior point method. If we were to force the interior
point method to continue until comparable results are achieved, it might take a significant amount of time longer. Having said that, in radiotherapy the results usually do not need to be extremely accurate, so the high accuracy of SQP is not necessarily an advantage.

We also wish to point out that the runtime distribution of the implemented SQP solver, is totally different without the high computation times of the QR factorizations and the reduced Hessians. As a result, improving other parts of the implementation become a priority, such as computing the step size $\alpha_j$ of the active set method. Although we do not claim that our implementation is optimized to the fullest, we see no (obvious) possibilities for other significant improvements. As a result, it seems that the variant of SQP that was discussed in this chapter can be ruled out as a candidate to replace the interior point method.

### 6-8 Recommendations

Let us consider a quadratic problem of the form

$$\min_x \frac{1}{2} p^T G p + c^T p$$

subject to

$$\nabla a_i^T p = b_i, \quad i \in \{1, 2, ..., \tilde{m}\}$$

$$\nabla a_i^T p \leq b_i, \quad i \in \{\tilde{m} + 1, \tilde{m} + 2, ..., m\}$$

(6-21)

We assume $\tilde{m} \leq n$, otherwise problem (6-21) is infeasible. If $\tilde{m} = n$, the solution $p^*$ is fixed, i.e. $N = 0$, where $N$ is the number of iterations executed to solve problem (6-21). If $\tilde{m} = n - 1$, there is only one degree of freedom, so $N = 1$. In general, the upper bound for $N$ increases rapidly with the degrees of freedom, $n - \tilde{m}$. So, the the active method is most efficient when $\tilde{m}$ is large. Since for our quadratic subproblems $\tilde{m} = 0$, they have $n$ degrees of freedom, which yields a very high upper bound for $N$.

See for example test case 1, table 6-8, where $N_2 = 4739 \gg n = 696$. We also saw large values of $N_k$ for test case 3 when we used a different initial value $x_0$ (table 6-7). Apparently the working set often adds indices to the working set, only to be removed a few iterations later. This phenomenon, amplified by the many degrees of freedom, strongly affects the efficiency of the active-set method.

The degrees of freedom corresponding to a quadratic subproblem of problem (6-1) equals $\min(n, m)$. Since our problems often contain sets of linear constraints to keep the maximum dose of a volume under some value, i.e. $a_i x \leq b, i \in \{1, 2, ..., \tilde{m}\}$, we usually have
\( \min(n, m) = n. \) If we were to rewrite these sets of linear constraints into single (nonlinear) constraints, the total number of constraints becomes much lower than \( n \), thereby eliminating the discussed weakness of the active set method. A maximum dose constraint, \( d_i \leq b \) where \( d = Ax \) with \( A \in \mathbb{R}^{m \times n} \), can be remodelled as

\[
\sum_{i=1}^{\hat{m}} \exp(d_i) \leq \exp(b) \quad (6-22)
\]

[14]. Remodelling all maximum dose constraints would yield a very low total number of constraints \( m^* \). However, it does introduce bad scaling, so the implementation is not straightforward.

Another idea is to consider different solvers for the quadratic subproblems. Indeed, the convergence of SQP seems very good; the problem lies in the active set method. Tables 6-17 and 6-18 show the results of using the interior point method instead of the active set method to solve the quadratic subproblems. Note that the interior point method does not require a feasible initial point, so we do not need to (partially) solve a minimization problem to compute \( x_0 \).

<table>
<thead>
<tr>
<th>( k )</th>
<th>( f(x_k) )</th>
<th>( \max c(x_k) )</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.3015e-01</td>
<td>1.58e+00</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>3.0938e-01</td>
<td>-8.94e-04</td>
<td>6.6 sec</td>
</tr>
<tr>
<td>2</td>
<td>1.5011e-01</td>
<td>4.17e+03</td>
<td>3.4 sec</td>
</tr>
<tr>
<td>3</td>
<td>1.2934e-01</td>
<td>-1.06e-02</td>
<td>3.6 sec</td>
</tr>
</tbody>
</table>

Total Time: 14 sec

**Table 6-17:** SQP with interior point method applied to test case 3.

The runtimes of the subproblems seem very constant, which suggests that the performance of SQP with interior point method is stable and dependable. This is not the case for the active set method, where the first outer iterations are very expensive with respect to the last few. The total runtimes given in tables (6-17) and (6-18) show that for our test cases the interior point method performs better than the active set method. In fact, for test case 4 the total runtime is even lower than for the original interior point method.

To determine if this combination is in general better than the original interior point method, one has to consider if, for the interior point method, solving \( K \) quadratic problems is faster than solving 1 problem of type (6-1), where \( K \) denotes the number of outer iterations computed by SQP.
Sequential Quadratic Programming

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\max c(x_k)$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9.1183e+00</td>
<td>6.11e+01</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>6.4195e+01</td>
<td>-4.26e-06</td>
<td>431 sec</td>
</tr>
<tr>
<td>2</td>
<td>2.3638e+01</td>
<td>3.44e+04</td>
<td>171 sec</td>
</tr>
<tr>
<td>3</td>
<td>8.7676e+00</td>
<td>3.80e+04</td>
<td>184 sec</td>
</tr>
<tr>
<td>4</td>
<td>3.3378e+00</td>
<td>-5.37e-03</td>
<td>187 sec</td>
</tr>
<tr>
<td>5</td>
<td>1.3729e+00</td>
<td>-8.09e-03</td>
<td>172 sec</td>
</tr>
<tr>
<td>6</td>
<td>6.5665e-01</td>
<td>-7.55e-03</td>
<td>182 sec</td>
</tr>
<tr>
<td>7</td>
<td>3.9953e-01</td>
<td>-7.59e-03</td>
<td>176 sec</td>
</tr>
<tr>
<td>8</td>
<td>3.0650e-01</td>
<td>-4.81e-03</td>
<td>171 sec</td>
</tr>
<tr>
<td>9</td>
<td>2.7829e-01</td>
<td>-8.81e-03</td>
<td>169 sec</td>
</tr>
</tbody>
</table>

Total Time: 31 min

Table 6-18: SQP with interior point method applied to test case 4.

The interior point method uses slack variables to transform the inequality constraints into equality constraints. The slack variables are incorporated into the objective function, creating a minimization problem with $m$ equality constraints. The corresponding KKT system is solved using Newton’s method. As a result, the number of constraints $m$, which, along with $n$, defines the size of the KKT system, has a direct and predictable influence on the runtime of the algorithm.

The active set method on the other hand, is not necessarily slowed down if $m$ increases. The number that mainly influences the performance of the algorithm is the degrees of freedom $n$. So, presumably, the larger $m$ with respect to $n$, the better the performance of the active set method with respect to the interior point method may be. For which values of $m$ and $n$ (if any) the active set method supersedes the interior point method, we have not investigated. In any case, for our test cases 3 and 4, where already $m \gg n$, the interior point method is faster than the active method.

6-9 Conclusion

The many degrees of freedom along with poor initial guesses of the active set cause unpredictable performance of the active set method for the first few SQP iterations. As a result, the (potential) runtimes for test cases 1, 3 and 4 (table 6-16) are higher than those corresponding to the interior point method. We therefore recommend investigating rewriting the linear constraints as equation (6-22) to reduce the degrees of freedom, or using a different solver for the quadratic subproblems.
An approach that has not yet been considered is using penalty methods to solve problem (1-6).

\[
\min_x \quad f(x) \\
\text{subject to} \quad c_i(x) \leq 0, \quad i \in \{1, 2, \ldots, m\}
\]

(7-1)

According to [5] penalty methods show their strength when many of the constraints are linear. This may imply that penalty methods are suited for our problems. The goal of this chapter is to analyse the potential of penalty methods with respect to the interior point method and SQP. We want to stress that we have not implemented this method, so the contents discussed in this chapter are purely analytical and are merely used to decide if further research on penalty methods might be beneficial.

We start with a description of penalty methods applied to equality constrained problems. Then we show how penalty methods can be used for inequality constrained problems and analyse their potential behaviour on our problems.

### 7-1 Equality Constrained Case

Penalty methods are used to solve equality constrained problems of the form

\[
\min_x \quad f(x) \\
\text{subject to} \quad h_i(x) = 0, \quad i \in \{1, 2, \ldots, m\}
\]

(7-2)
For the remainder of this section we use $x^*$ to denote the solution of problem (7-2) and write $h(x) = [h_1(x), h_2(x), \ldots, h_m(x)]^T$. Recall that the KKT conditions corresponding to problem (7-2) are given by

\begin{align*}
\nabla f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) &= 0 \quad (7-3a) \\
h_i(x) &= 0 \quad (7-3b)
\end{align*}

Penalty methods combine the objective function and the constraints into one penalty function and reformulate problem (7-2) into

\[
\min_x P(x; \mu) \quad (7-4)
\]

where $P(x; \mu)$ is some penalty function with penalty parameter $\mu$. Minimization of penalty functions always involves a trade-off between minimizing the objective function $f(x)$ and satisfying the constraints.

We distinguish between exact and inexact penalty functions. The minimizer of an exact penalty function coincides with $x^*$ for any sufficiently large $\mu$, whereas this is generally not the case for inexact penalty functions. The $\ell_1$ merit function discussed in section 6-1-2 is an example of an exact penalty function. A disadvantage of exact penalty functions is that they are usually not smooth, which introduces difficulties for most solvers. We therefore consider smooth inexact penalty functions instead, which can be minimized using Newton’s method. An example is the quadratic penalty function:

\[
P(x; \mu) = f(x) + \frac{\mu}{2} \sum_{i=1}^{m} h_i(x)^2 \quad (7-5)
\]

The penalty parameter $\mu$ defines the severity with which violated constraints are penalized.

Intuition suggests that driving $\mu$ to infinity causes $h_i(x) \to 0$ at the minimizer of problem (7-5). This is confirmed by noting that the minimizer $x_k$ of the quadratic penalty function (7-5) satisfies

\[
\nabla f(x_k) + \mu \sum_{i=1}^{m} h_i(x_k) \nabla h_i(x_k) = 0 \quad (7-6)
\]

whereas $x^*$ satisfies the KKT conditions (7-3) for some Lagrange multiplier $\lambda^*$. Defining $\lambda_k = \mu h(x_k)$, we see that for $(x, \lambda) = (x_k, \lambda_k)$ condition (7-3a) is satisfied and $h(x_k) = \lambda_k / \mu$, which goes to zero as $\mu$ goes to infinity. So $(x_k, \lambda_k) \to (x^*, \lambda^*)$ as $\mu \to \infty$. 

So it makes sense to define a sequences \( \{\mu_k\}_{k>0} \) with \( \mu \to \infty \) as \( k \to \infty \) and solve sub-problem (7-4) for each \( k \) until the KKT conditions (7-3) are satisfied to some tolerance \( \epsilon \).

Note that \( h_i(x_k) \to \lambda_i^* / \mu_k \) as \( x_k \to x^* \), so for all \( i \in \{1, 2, ..., m\} \)

\[
h_i(x_k) \approx \frac{1}{\mu} \lambda_i^* \quad (7-7)
\]
in the neighbourhood of \( x^* \). This implies that there is generally no finite \( \mu \) for which problem (7-4) is equivalent to problem (7-2). As a result, high values of \( \mu_k \) are needed to obtain decent accuracy of the final solution. This is incidentally the main weakness of inexact penalty functions.

Unfortunately, problem (7-4) becomes ill-conditioned when the penalty parameter \( \mu \) is large, which may cause numerical problems [5]. Also, for large \( \mu \) the quadratic approximation on which Newton’s method is based, may not capture the behaviour of the quadratic penalty function adequately [5]. It is therefore beneficial to use a penalty function which does not systematically cause the perturbation \( h_i(x_k, s_k) \approx \lambda_i^* / \mu \), but instead stimulates the constraints to approach zero for moderate values of \( \mu \).

The augmented Lagrangian function has these desirable properties:

\[
P(x; \mu, \lambda) = \mathcal{L}(x, \lambda) + \frac{\mu}{2} \sum_{i=1}^{m} h_i(x)^2
\]

(7-8)

where \( \mathcal{L}(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) \) is the Lagrangian of the original problem (7-1). The minimizer \( x_k \) of the augmented Lagrangian function (7-8) satisfies

\[
\nabla f(x_k) + \sum_{i=1}^{m} [\lambda_i + \mu h_i(x_k)] \nabla h_i(x_k) = 0
\]

(7-9)

Using the same reasoning as for the quadratic penalty function, we find \( h_i(x_k) \to [\lambda_i^* - \lambda_i] / \mu \) as \( x_k \to x^* \), so for all \( i \in \{1, 2, ..., m\} \)

\[
h_i(x_k) \approx \frac{1}{\mu} [\lambda_i^* - \lambda_i] \quad (7-10)
\]
in the neighbourhood of the solution \( x^* \). If \( \lambda \) is a good approximation of \( \lambda^* \), the infeasibility in \( x_k \) is much smaller than \( 1/\mu \), rather than being proportional to \( 1/\mu \) as is the case for the quadratic penalty function.
For each subproblem

$$\min_{x} P(x; \mu_k, \lambda_k)$$

(7-11)

we have the opportunity to update the estimate \( \lambda = \lambda_k \) of the optimal Lagrange multiplier \( \lambda^* \). Relation (7-10) immediately suggests choosing

$$\lambda_{k+1} = \mu h(x_k) + \lambda_k$$

(7-12)

This discussion motivates algorithmic framework 3.

**Algorithm 3** Augmented Lagrangian Penalty Method

Choose initial values \( x_0, \lambda_0 \) and \( \mu_0 \);

for \( k = 0, 1, 2, \ldots \) do

Set \( x_{k+1} \) as the solution of problem (7-11) using Newton’s method, starting at \( x_k \);

if Stopping criterion is satisfied then

Stop with approximate solution \( x_{k+1} \);

end if

Compute \( \lambda_{k+1} \) from (7-12);

Choose new penalty parameter \( \mu_{k+1} > \mu_k \);

end for

7.2 Inequality Constrained Case

To use penalty methods for inequality constrained problems, slack variables are introduced to convert the inequality constraints into equality constraints. So problem (7-1) is reformulated into

$$\min_{x} f(x)$$

subject to \( c_i(x) + s_i = 0, \quad i \in \{1, 2, \ldots, m\} \)

$$x_i, s_i \geq 0, \quad i \in \{1, 2, \ldots, m\}$$

(7-13)

The subproblems corresponding to the penalty method for problem (7-13) are then

$$\min_{x,s} P(x, s; \mu_k, \lambda_k)$$

subject to \( x_i, s_i \geq 0, \quad i \in \{1, 2, \ldots, m\} \)

(7-14)

where \( P(x, s; \mu_k, \lambda_k) \) is the augmented Lagrangian function (7-8). The only difference between penalty methods applied to equality constrained problems, and problem (7-13), is
that the subproblems (7-14) are solved using SQP, instead of Newton’s method.

So what is the advantage of solving a sequence of inequality constrained problems of type (7-14) using SQP, with respect to solving only one inequality constrained problem of type (7-1) using SQP?

The only drawback of applying SQP to problem (7-1), is that $N_k$ is large for the first few subproblems. This is partly caused by poor initial guesses of the active set. Recall that a good initial guess of the working set is important, because at each iteration only one index may be added or removed from the working set. Indeed, adding multiple constraints that are violated for a full stepsize is likely to cause infeasible working sets, i.e. no feasible $x$ exists for which the constraints corresponding to the working set are active, see figure 7-1. This problem does not arise when the constraints are simple bounds, which is the case for problem (7-14), so we can safely use techniques to add multiple indices to the working set at once. As a result, the subproblems corresponding to (7-14) are less sensitive to the initial guess of the active set than those corresponding to problem 7-1).

To form an opinion on the potential of penalty methods, we have to consider two things. The costs of solving the subproblems, and the number of subproblems that need to be solved to obtain an acceptable approximate solution of problem (7-1).

The former may heavily depend on the latter. Indeed, large values of $\mu_k$ may introduce ill-conditioning, which can cause numerical problems. Also, the quadratic problems may not be able to capture the behaviour of problem (7-14) adequately for large $\mu_k$. So, if large values of $\mu_k$ are needed to obtain an acceptable approximate solution of problem (7-1), penalty methods are probably not suited for our purpose. However, if $\mu_k$ is kept at a moderate value, the active set method potentially needs few, relatively cheap iterations.
to find the minimizer $x_k$ of subproblem (7-14).

So it all comes down to the sequence $\{\mu_k\}_{k=0}^K$, where $K$ is the total number of outer iterations. We argued in the previous section that by using the quadratic penalty function the penalty parameter $\mu_K$ is likely to be large, because the feasibility conditions are perturbed. To avoid this, we introduced the augmented Lagrangian function (7-8), which stimulates the constraints to be satisfied for relatively small values of $\mu_k$, provided that the estimates of the optimal Lagrange multiplier are decent. This desirable property potentially results in good convergence of the penalty method, i.e. both $K$ and $\mu_K$ remain acceptably small.

Suppose that solving subproblem (7-14) using SQP requires $J_k$ quadratic subproblems. Let $N_j$ be the number of active set iterations to solve the $j^{th}$ quadratic subproblem of subproblem (7-14). Then the total number of active set iterations is $N = \sum_{k=1}^K \sum_{j=1}^{J_k} N_j$.

For penalty methods to perform better on our test cases than SQP, $N$ would have to be significantly smaller than the total number of inner iterations given in tables 6-8 - 6-10, because each formulation of a subproblem (7-14) requires computing the expensive expression $A^T A$, where $A$ is the matrix containing the gradients of the constraints $c_i(x) + s_i$. This also implies that for penalty methods to perform better than interior point methods, $K$ would have to be significantly smaller than the total number of outer iterations listed in tables 4-1 - 4-4. Indeed, the bottleneck of interior point methods lies in the compaction of $A^T D A$, where $D$ is a diagonal matrix.

If the estimates $\lambda_k, k > 0$ approximate the optimal Lagrange multiplier $\lambda^*$ well, the penalty method with augmented Lagrangian penalty function, potentially needs few outer iterations $K$ with moderates values of $\mu_k$ to find an acceptable solution of problem (7-1). In chapter 6 we saw that SQP requires few outer iterations to solve problem (7-1), so we might hope that this also applies for solving subproblem (7-14), i.e. $J_k$ is potentially small. (Recall that for moderate values of $\mu_k$ subproblem (7-14) is well-conditioned). Finally, because the constraints of problem (7-1) are simple bounds, they can potentially be solved efficiently using SQP, so $N_j$ is potentially small as well.

Clearly, the performance of penalty methods depends on many elements. If the discussed potentially favourable properties of even one of these elements turns out to be somewhat disappointing, the efficiency of the penalty methods is significantly reduced. We therefore cannot guarantee desirable behaviour of the penalty method without testing it on some test cases.
In chapter 5 we argued that the logarithmic barrier method is too unreliable to supersede the interior point method, so we advice against further research in this area.

In chapter 6 we saw that SQP has potential. For our test cases a very good final solution was found in relatively few (outer) iterations. Unfortunately, the active set method showed some weakness when applied to the subproblems of our test cases, which resulted in relatively high total runtimes. To take advantage of the good convergence properties of SQP we presented two ideas for further research in section 6-8: Reformulate the maximum dose constraints such that the total number of constraints decreases, or use a different solver for the quadratic subproblems.

In chapter 7 we discussed the potential of penalty methods. We saw that penalty methods have potential to be faster than SQP and the interior point method, but their efficiency depends on many different elements. To find out if the favourable theoretical behaviour of penalty methods appears in practice, the algorithm has to be implemented and tested, which is a lot of work. We believe that the uncertainties corresponding to the practical performance of penalty methods are high with respect to the possible gain, so we advice against further research of penalty methods.
Appendix A

Unconstrained Optimization

In this appendix we describe different methods to solve the following problem:

\[ \min_x g(x) \]  \hspace{1cm} \text{(A-1)}

where \( g(x) : \mathbb{R}^n \rightarrow \mathbb{R} \) is convex. The first section is dedicated to Newton’s method for optimization. Then we discuss quasi-Newton methods, which are modified versions of the original Newton method.

A-1 Newton’s Method for Optimization

Newton’s method for optimization iteratively approaches the solution \( x^* \) of problem (A-1) by subsequently minimizing second degree Taylor approximations of \( g(x) \).

Let us define \( g_k(p) \), where \( p = x - x_k \), as the second degree Taylor polynomial of \( g(x) \) around \( x_k \), and let \( m_k(p) \) be its derivative:

\[
\begin{align*}
g_k(p) &= g(x_k) + \nabla g(x_k)p + \frac{1}{2} p^T \nabla^2 g(x_k)p \\
m_k(p) &= \nabla g(x_k) + \nabla^2 g(x_k)p
\end{align*}
\]  \hspace{1cm} \text{(A-2, A-3)}

The minimum of \( g_k(x) \) is found by solving \( m_k(p) = 0 \). The solution defines the Newton step:

\[ p_k = - \left[ \nabla^2 g(x_k) \right]^{-1} \nabla g(x_k) \]  \hspace{1cm} \text{(A-4)}
Note that \( m_k(p) \) equals the first degree Taylor polynomial of \( \nabla g(x) \). Newton’s method for optimization is therefore nothing else than standard Newton\(^1\) applied to \( \nabla g(x) \). This insight is useful for the derivation of the quasi-Newton methods, which are discussed in the next section.

The algorithm is straightforward. At the \( k^{th} \) iteration the next iterate is found by computing the Newton step \( \mathbf{A}^{-1} \mathbf{b} \) and then setting \( x_{k+1} ← x_k + p_k \). The procedure is summarized in algorithm 4.

**Algorithm 4** Newton’s method for optimization

Choose initial value \( x_0 \); 

for \( k = 0, 1, 2, ... \) do 

Compute \( p_k \) by evaluating A-4; 

Set \( x_{k+1} ← x_k + p_k \); 

if Stopping criterium is satisfied then 

stop with solution \( x^* = x_{k+1} \); 

end if 

end for

\( \text{A-2 Quasi-Newton Methods} \)

The concept of quasi-Newton methods is similar to that of the original Newton method. The key difference is that instead of using exact Taylor polynomials, quasi-Newton methods approximate the Hessian \( \nabla^2 g(x_k) \approx B_k \), yielding the following modified versions of \( g_k(p) \) and \( m(x) \):

\[
\tilde{g}_k(p) = g(x_k) + \nabla g(x_k)p + \frac{1}{2}p^T B_k p \\
\tilde{m}_k(p) = \nabla g(x_k) + B_k p
\]

To maintain convexity of \( \tilde{g}_k(p) \), \( B_k \) is a symmetric positive definite matrix of appropriate size. The modified Newton step becomes

\[
p_k = -H_k \nabla g(x_k)
\]

where \( H_k = B_k^{-1} \). There are several ways to define \( B_k \), or, equivalently, \( H_k \). We discuss the BFGS (Broyden, Fletcher, Goldfarb, and Shanno) approach and the limited memory BFGS approach.

\(^{1}\)Newton’s method uses first order Taylor polynomials to iteratively solve \( f(x) = 0 \).
A-2-1 BFGS

Let us view Newton’s method as an algorithm which finds the solution of (A-1) by solving \( \nabla g(x) = 0 \), using first order Taylor polynomials of \( \nabla g(x) \). Since \( \tilde{m}_{k+1}(p) \) is meant as an approximation of \( \nabla g(x) \), it is reasonable to demand that \( \tilde{m}_{k+1}(p) \) should match \( \nabla g(x) \) at the latest two iterates \( x_k \) and \( x_{k+1} \). This translates mathematically into

\[
\tilde{m}_{k+1}(-p_k) = \nabla g(x_k) \\
\tilde{m}_{k+1}(0) = \nabla g(x_{k+1})
\]

Note that the second condition is satisfied automatically. The first condition yields

\[
B_{k+1}p_k = y_k \\
\iff \quad H_{k+1}y_k = p_k
\]  

(A-8)

where \( y_k = \nabla g(x_{k+1}) - \nabla g(x_k) \). There are infinitely many matrices that would satisfy equation (A-8). To determine \( H_{k+1} \) (and thereby \( B_{k+1} \)) uniquely, we impose as an additional condition on \( H_{k+1} \) that among all symmetric matrices satisfying equation (A-8), \( H_{k+1} \) is, in some sense, closest to the current matrix \( H_k \). So \( H_{k+1} \) would define the solution of the following problem

\[
\min_H \| H - H_k \| \\
\text{subject to} \quad H = H^T, \quad H y_k = p_k
\]  

(A-9)

Different norms can be used for problem A-9. Each norm gives rise to a different quasi-Newton method. The BFGS method uses the weighted Frobenius norm,

\[
\|A\|_W = \|W^{1/2}AW^{1/2}\|_F,
\]

where the Frobenius norm for a matrix \( C \) is defined as \( \|C\|_F = \sum_{i,j} c_{ij}^2 \). This yields the following update for \( H_k \) and \( B_k \):

\[
H_{k+1} = \left( I - \frac{p_k y_k^T}{p_k^T y_k} \right) H_k \left( I - \frac{y_k p_k^T}{p_k^T y_k} \right) + \frac{p_k p_k^T}{p_k^T y_k} \\
B_{k+1} = B_k - \frac{B_k p_k p_k^T B_k}{p_k^T B_k p_k} - \frac{y_k y_k^T}{p_k^T y_k}
\]  

(A-10)  

(A-11)

[5]. Note that since \( B_k \) is positive definite, equation (A-8) implicitly imposes the following constraint on \( p_k \) and \( y_k \):

\[
p_k^T y_k > 0
\]  

(A-12)
This can easily be verified by premultiplying equation A-8 by \( p_k^T \). Even though constraint A-12 is satisfied automatically at each iteration for convex \( g(x) \), it may be appropriate to enforce some margin on the constraint. Indeed, a low value of \( p_k^T y_k \) indicates weak convexity of \( \tilde{g}_k(p) \), which is undesirable. To that end we introduce

\[
  r_k = \theta_k y_k + (1 - \theta_k) B_k p_k \tag{A-13}
\]

where

\[
  \theta_k = \begin{cases} 
  1 & \text{if } p_k^T y_k \geq 0 \\
  0.8 p_k^T B_k p_k & \text{if } p_k^T y_k < 0.2 p_k^T B_k p_k \\
  \frac{p_k^T B_k p_k - p_k^T y_k}{p_k^T y_k} & \text{otherwise}
  \end{cases} \tag{A-14}
\]

such that \( p_k^T r_k \geq 0.2 p_k^T B_k p_k \geq 0 \). Substituting \( r_k \) for \( y_k \) in (A-10) and (A-11) yields the Damped BFGS Updating. Note that for \( \theta_k = 1 \) simply holds \( r_k = y_k \).

An overview of the BFGS method is given in algorithm 5. Writing out the updating formula (A-10) and noting that \( y_k^T H_k y_k \) is a scalar, we come to the conclusion that an iteration of the BFGS method requires \( O(n^2) \) multiplications (excluding function evaluations).

**Algorithm 5 BFGS Method**

Choose initial value \( x_0 \) and \( H_0 \);

for \( k = 0, 1, 2, \ldots \) do

- Compute \( p_k \) by evaluating (A-7);
- Set \( x_{k+1} \leftarrow x_k + p_k \);

  if Stopping criterium is satisfied then
  - stop with solution \( x^* = x_{k+1} \);
  else
  if Damped BFGS Updating is used then
  - Compute \( r_k \) using (A-13) and set \( y_k \leftarrow r_k \);
  end if
  - Compute \( H_{k+1} \) using (A-10);
  end if

end for

**A-2-2 Limited Memory BFGS Method**

The limited memory BFGS method, or L-BFGS method, is directly derived from the BFGS method, so we use the same terms as in the previous section.
For large-scale problems, the BFGS updating formula (A-10) can still be expensive with regard to the number of required multiplications and storage. To circumvent this problem, the L-BFGS method stores a modified version of $H_k$ implicitly, by storing the $l$ most recent vector pairs $(p_i, y_i), i \in \{k - l, k - l + 1, \ldots, k - 1\}$. The next search direction $p_k$ can be obtained by performing a sequence of inner products and vector summations involving these vector pairs. So the L-BFGS method only takes the curvature information of the last $l$ iterations into account.

First, we reformulate the BFGS method by fully writing out the updating formula (A-10). For clarity we substitute $V_k = I - \frac{y_k p_k^T}{p_k^T y_k}$ and $\rho_k = \frac{1}{p_k^T y_k}$.

$$H_k = V_{k-1}^T H_{k-1} V_{k-1} + \rho_{k-1} p_{k-1} p_{k-1}^T$$

Using this formulation, the value $-p_k = H_k \nabla f(x_k)$ can be obtained by the recursive procedure presented in algorithm 6 [5].

<table>
<thead>
<tr>
<th>Algorithm 6 BFGS two-loop recursion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q \leftarrow \nabla f(x_k)$;</td>
</tr>
<tr>
<td>for $i = k - 1, k - 2, \ldots, 0$ do</td>
</tr>
<tr>
<td>$\alpha_i \leftarrow \rho_i y_i^T q$;</td>
</tr>
<tr>
<td>$q \leftarrow q - \alpha_i y_i$;</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>$r \leftarrow H_0 q$;</td>
</tr>
<tr>
<td>for $i = 0, 1, \ldots, k - 1$ do</td>
</tr>
<tr>
<td>$\beta \leftarrow \rho_1 y_1^T r$;</td>
</tr>
<tr>
<td>$r \leftarrow r + p_i (\alpha_i - \beta)$;</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td><strong>stop</strong> with $H_k \nabla f(x_k) = r$.</td>
</tr>
</tbody>
</table>

Instead of computing this full two-loop recursion formula at each iteration, the L-BFGS method discards the curvature information of the first $k - l$ iterations. This can be justified by arguing that information from earlier iterations is likely to be less relevant than that from more recent iterations. So the search direction $p_k$ is computed by the two-loop recursion presented in algorithm 7.
Algorithm 7 L-BFGS two-loop recursion

Choose \( H_k^0 \);
\[ q \leftarrow \nabla f(x_k); \]
for \( i = k - 1, k - 2, \ldots, k - l \) do
\[ \alpha_i \leftarrow \rho_i p_i^T q; \]
\[ q \leftarrow q - \alpha_i y_i; \]
end for
\[ r \leftarrow H_k^0 q; \]
for \( i = k - l, k - l + 1, \ldots, k - 1 \) do
\[ \beta \leftarrow \rho_i y_i^T r; \]
\[ r \leftarrow r + p_i (\alpha_i - \beta); \]
end for
stop with \( p_k = -r \).

In this scheme, the matrix \( H_k^0 \) can be chosen freely. A popular method is to set
\[
H_k^0 = \frac{p_{k-1}^T y_{k-1}}{y_{k-1} y_{k-1}} I
\]  (A-15)

This choice helps the next \( p_k \) to be well-scaled. Without considering the multiplication of \( H_k^0 q \), scheme 7 requires \( 4ln \) multiplications only. The L-BFGS algorithm is summarized in algorithm 8.

Algorithm 8 L-BFGS Method

Choose initial value \( x_0 \);
for \( k = 0, 1, 2, \ldots \) do

Compute \( p_k \) according to scheme 7;
Set \( x_{k+1} \leftarrow x_k + p_k \);
if Stopping criterium is satisfied then
stop with solution \( x^* = x_{k+1} \);
else
if \( k > l \) then
Discard vector pair \((p_{k-l}, y_{k-l})\) from storage;
Save \( p_k \) and \( y_k = \nabla f(x_{k+1}) - \nabla f(x_k) \);
end if
end if
end if
In this section we discuss an active-set method, which can be applied to solve quadratic problems of the form
\[
\begin{aligned}
\min_x & \quad \frac{1}{2} x^T G x + c^T x \\
\text{subject to} & \quad A x \leq b
\end{aligned}
\]  
(B-1)

where \( G \in \mathbb{R}^{n \times n} \) is symmetric and positive definite, \( c \in \mathbb{R}^n \), \( A \in \mathbb{R}^{m \times n} \) and \( b \in \mathbb{R}^m \). We use \( x^* \) to denote the solution of problem (B-1), and we use \( a_i \) to denote the \( i^{th} \) row of matrix \( A \). The Lagrangian of problem (B-1) is given by
\[
\mathcal{L}(x, \lambda) = \frac{1}{2} x^T G x + c^T x + \lambda^T (Ax - b)
\]  
(B-2)

The corresponding KKT conditions are
\[
\begin{aligned}
G x + c + A^T \lambda &= 0 \quad &\text{(B-3a)} \\
A x &\leq b \quad &\text{(B-3b)} \\
\lambda_i &\geq 0, \quad i \in \{1, 2, ..., m\} \quad &\text{(B-3c)} \\
\lambda_i (a_i x - b_i) &= 0, \quad i \in \{1, 2, ..., m\} \quad &\text{(B-3d)}
\end{aligned}
\]

The active set method is based on iterative guesses of the active set \( \mathcal{A} \) corresponding to problem (B-1). Recall that \( \mathcal{A} \) is defined as the set of indices corresponding to the constraints which are active at the solution \( x^* \). So \( a_i x^* = b_i, i \in \mathcal{A} \) and \( a_i x^* < b_i, i \notin \mathcal{A} \). To distinguish between the guesses of the active set, and the true active set, we introduce the term Working set, which we denote by \( \mathcal{W} \). So \( \mathcal{W}_k \) denotes the \( k^{th} \) guess of \( \mathcal{A} \).
The idea of the active set method for quadratic programming is as follows. We start with an initial guess $W_0$ of the active set $A$. We choose $x_0$ to be feasible with respect to $W_0$, i.e. $a_i x_0 = b_i, i \in W_0$ and $a_i x_0 \leq b_i, i \notin W_0$. A search direction $p_0$ is determined by finding the solution $\tilde{x} = x_0 + p_0$ of the equality constrained subproblem

$$\min_x \quad \frac{1}{2} x^T G x + c^T x$$
subject to $A_k x = b_k$ \quad (B-4)

for $k = 0$. Here $A_k$ and $b_k$ denote the current active constraints, so the constraints whose indices are not in the working set are ignored here. We then check whether $\tilde{x}$ violates any of the inactive constraints. If so, a stepsize $\alpha$ is computed such that $a_j (x_0 + \alpha p_0) = b_j$, where $j \notin W_0$ is the first constraint to be violated along $p_0$. This constraint is referred to as the blocking constraint. We set $x_1 = x_0 + \alpha p_0$, making the $j^{th}$ constraint active. As a result, $j$ is added to the working set. If no constraints are violated, a full step $p_0$ is taken and the working set is left unchanged. The next search direction is found by solving (B-4) with the updated iterate and working set. This process is repeated until the KKT conditions (B-3) are satisfied.

### B-1 Search Direction

Let $x_k$ be the $k^{th}$ estimate of the optimal solution $x^*$ and let $W_k$ be its corresponding working set. As previously stated, we determine the search direction $p_k$ by minimizing the objective, subject to the equality constraints of the current working set. For practical reasons we substitute $x = p + x_k$ into problem (B-4) to obtain the equivalent problem

$$\min_p \quad \frac{1}{2} p^T G p + [G x_k + c]^T p$$
subject to $A_k p = 0$. \quad (B-5)

The KKT conditions corresponding to the solution $p_k$ of problem (B-5) are

$$G p_k + G x_k + c + A_k^T \lambda_k = 0 \quad (B-6a)$$
$$A_k p_k = 0 \quad (B-6b)$$

where $\lambda_k$ denotes the Lagrange multipliers corresponding to the active constraints. These KKT conditions can be reformulated in the following manner:

$$\begin{bmatrix} G & A_k^T \\ A_k & 0 \end{bmatrix} \begin{bmatrix} p_k \\ \lambda_k \end{bmatrix} = - \begin{bmatrix} G^T x_k + c \\ 0 \end{bmatrix} \quad (B-7)$$

The search direction $p_k$ and the Lagrange multipliers of the active constraints $\lambda_k$, are found by solving system (B-7).
B-2 Stepsize

When the search direction $p_k$ is found, the stepsize $\alpha$ needs to be determined. We want the stepsize to be as large as possible, but such that all constraints are satisfied at the next iterate, i.e. $\alpha = \max \{ s \in [0, 1] : A(x_k + sp_k) \leq b \}$. Note that the constraints corresponding to the current working set are satisfied regardless of the choice of $\alpha$. The same holds for all $i \notin W_k$ for which $a_ip_k \leq 0$, since $x_k$ is feasible. For the rest of the constraints we know that $a_i(x_k + \alpha p_k) \leq b_i$ only holds if $\alpha \leq \frac{b_i - a_ix_k}{a_ip_k}$. To summarize, the stepsize $\alpha$ is chosen as

$$\alpha = \min \left( 1, \min \left\{ \frac{b_i - a_ix_k}{a_ip_k} : i \notin W_k, a_ip_k > 0 \right\} \right)$$  \hspace{1cm} (B-8)

If $\alpha < 1$, a full step was impossible due to the so-called blocking constraint, i.e. the constraint for which $\frac{b_i - a_ix_k}{a_ip_k}$ is the smallest. Since $\alpha$ was chosen such that the blocking constraint is active at $x_{k+1} = x_k + \alpha p_k$, its index is added to the working set.

B-3 Practical Algorithm

Since the initial point $x_0$ is feasible and we always choose the stepsize $\alpha$ such that the next iterate is feasible, active set methods only allow feasible iterates. Furthermore, the Lagrange multipliers corresponding to inactive constraints are zero. We can therefore ignore both the second and the fourth of the KKT conditions (B-6), since they are satisfied automatically at each iteration. The remaining conditions are

$$Gx^* + c + A^T\lambda^* = 0$$  \hspace{1cm} (B-9a)

$$\lambda_i^* \geq 0, \quad i \in \{1, 2, ..., m\}$$  \hspace{1cm} (B-9b)

Note that the first condition is satisfied when the solution of subproblem (B-5) yields $p_k = 0$. This can be verified by substituting $p_k = 0$ into (B-6). (Bare in mind that the Lagrange multipliers corresponding to inactive constraints are zero.) So when $p_k = 0$, i.e. $x_{k+1} = x_k$, the Lagrange multipliers $\lambda_k$ are checked. If all of them are non-negative, the (exact!) solution of problem (B-1) is $x^* = x_{k+1}$. If there are negative multipliers, the constraint corresponding to the smallest Lagrange multiplier is removed from the working
set and the new search direction $p_{k+1}$ is computed. The active-set method is summarized in algorithm 9.

**Algorithm 9** Active Set Algorithm

Choose initial guess of the active set $W_0$;  
Choose an initial point $x_0$, which is feasible with respect to $W_0$;  
for $k = 0, 1, 2, \ldots$ do  
    Compute $p_k$ and $\lambda$ by solving B-7;  
    if $p_k = 0$ then  
        Let $i$ be the index corresponding to the smallest element of $\lambda$;  
        if $\lambda_i \geq 0$ then  
            stop with solution $x^* = x_k$;  
        else  
            Set $x_{k+1} \leftarrow x_k$;  
            Set $W_{k+1} \leftarrow W_k \setminus i$;  
        end if  
    else  
        Compute $\alpha_k$ from B-8;  
        Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$;  
        if $\alpha_k < 1$ then  
            Obtain $W_{k+1}$ by adding the index of the blocking constraint to $W_k$;  
        else  
            Set $W_{k+1} \leftarrow W_k$;  
        end if  
    end if  
end for
### Appendix C

#### Results of the Logarithmic Barrier Method

<table>
<thead>
<tr>
<th>( k )</th>
<th>( f(x_k) )</th>
<th>( \text{max } c(x_k) )</th>
<th>( t_k )</th>
<th>( \tilde{\epsilon}_k )</th>
<th>( N_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.6235e-01</td>
<td>5.00e+00</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>6.0049e-01</td>
<td>0.00e+00</td>
<td>9.19e+02</td>
<td>1.04e-02</td>
<td>48</td>
</tr>
<tr>
<td>2</td>
<td>6.1102e-01</td>
<td>0.00e+00</td>
<td>1.59e+03</td>
<td>3.61e-02</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>6.1682e-01</td>
<td>0.00e+00</td>
<td>2.76e+03</td>
<td>5.62e-01</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5.4219e-01</td>
<td>0.00e+00</td>
<td>4.77e+03</td>
<td>3.24e+01</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>3.8429e-01</td>
<td>0.00e+00</td>
<td>8.27e+03</td>
<td>1.28e+01</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>2.8051e-01</td>
<td>0.00e+00</td>
<td>1.43e+04</td>
<td>1.59e+01</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>2.1239e-01</td>
<td>0.00e+00</td>
<td>2.48e+04</td>
<td>2.24e+01</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>1.6820e-01</td>
<td>0.00e+00</td>
<td>4.30e+04</td>
<td>3.43e+01</td>
<td>28</td>
</tr>
<tr>
<td>9</td>
<td>1.4039e-01</td>
<td>0.00e+00</td>
<td>7.44e+04</td>
<td>4.44e+01</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( k )</th>
<th>( f(x_k) )</th>
<th>( \text{max } c(x_k) )</th>
<th>( t_k )</th>
<th>( \tilde{\epsilon}_k )</th>
<th>( N_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1.4039e-01</td>
<td>0.00e+00</td>
<td>7.44e+04</td>
<td>4.44e+01</td>
<td>187</td>
</tr>
</tbody>
</table>

**Table C-1**: Newton-Barrier method applied to test case 1, with \( \mu = \sqrt{3} \), \( \epsilon = 0.1 \), and \( \sigma = 0.3 \).
Results of the Logarithmic Barrier Method

Table C-2: Newton-Barrier method applied to test case 2, with $\mu = \sqrt{3}$, $\epsilon = 0.1$ and $\sigma = 0.3$
### Table C-3: Newton-Barrier method applied to test case 3, with $\mu = \sqrt{3}$, $\epsilon = 0.1$ and $\sigma = 0.3$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\text{max } c(x_k)$</th>
<th>$t_k$</th>
<th>$\tilde{\epsilon}_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$2.3015e-01$</td>
<td>$1.59e+00$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>$5.0372e-01$</td>
<td>$0.00e+00$</td>
<td>$3.26e+03$</td>
<td>$1.05e-01$</td>
<td>178</td>
</tr>
<tr>
<td>2</td>
<td>$5.2924e-01$</td>
<td>$0.00e+00$</td>
<td>$5.65e+03$</td>
<td>$4.27e-01$</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>$3.7909e-01$</td>
<td>$0.00e+00$</td>
<td>$9.79e+03$</td>
<td>$4.49e-01$</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>$2.8059e-01$</td>
<td>$0.00e+00$</td>
<td>$1.70e+04$</td>
<td>$5.38e-01$</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>$2.1762e-01$</td>
<td>$0.00e+00$</td>
<td>$2.94e+04$</td>
<td>$5.59e-01$</td>
<td>49</td>
</tr>
<tr>
<td>6</td>
<td>$1.7901e-01$</td>
<td>$0.00e+00$</td>
<td>$5.09e+04$</td>
<td>$6.40e-01$</td>
<td>99</td>
</tr>
<tr>
<td>7</td>
<td>$1.5759e-01$</td>
<td>$0.00e+00$</td>
<td>$8.81e+04$</td>
<td>$7.23e-01$</td>
<td>155</td>
</tr>
<tr>
<td>8</td>
<td>$1.4709e-01$</td>
<td>$0.00e+00$</td>
<td>$1.53e+05$</td>
<td>$1.10e+00$</td>
<td>229</td>
</tr>
<tr>
<td>9</td>
<td>$1.4254e-01$</td>
<td>$0.00e+00$</td>
<td>$2.64e+05$</td>
<td>$2.04e+00$</td>
<td>340</td>
</tr>
<tr>
<td>9</td>
<td>$1.4254e-01$</td>
<td>$0.00e+00$</td>
<td>$2.64e+05$</td>
<td>$2.04e+00$</td>
<td>1107</td>
</tr>
</tbody>
</table>

### Table C-4: Newton-Barrier method applied to test case 4, with $\mu = \sqrt{3}$, $\epsilon = 0.1$ and $\sigma = 0.3$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>$\text{max } c(x_k)$</th>
<th>$t_k$</th>
<th>$\tilde{\epsilon}_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$9.1183e+00$</td>
<td>$1.14e+02$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>$1.3397e-01$</td>
<td>$9.54e+00$</td>
<td>$1.04e+03$</td>
<td>$4.68e-03$</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>$1.7793e-01$</td>
<td>$2.02e+01$</td>
<td>$1.81e+03$</td>
<td>$1.76e-02$</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>$2.3773e-01$</td>
<td>$1.87e+01$</td>
<td>$3.13e+03$</td>
<td>$1.80e-02$</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>$3.3018e-01$</td>
<td>$5.58e+00$</td>
<td>$5.42e+03$</td>
<td>$1.17e-01$</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>$4.3341e-01$</td>
<td>$1.33e+00$</td>
<td>$9.39e+03$</td>
<td>$2.67e-01$</td>
<td>87</td>
</tr>
<tr>
<td>6</td>
<td>$5.0876e-01$</td>
<td>$1.33e-01$</td>
<td>$1.63e+04$</td>
<td>$4.92e+00$</td>
<td>163</td>
</tr>
<tr>
<td>7</td>
<td>$4.5547e-01$</td>
<td>$0.00e+00$</td>
<td>$2.82e+04$</td>
<td>$1.03e+00$</td>
<td>87</td>
</tr>
<tr>
<td>8</td>
<td>$4.0053e-01$</td>
<td>$0.00e+00$</td>
<td>$4.88e+04$</td>
<td>$1.15e+00$</td>
<td>114</td>
</tr>
<tr>
<td>9</td>
<td>$3.7110e-01$</td>
<td>$0.00e+00$</td>
<td>$8.45e+04$</td>
<td>$1.03e+00$</td>
<td>232</td>
</tr>
<tr>
<td>10</td>
<td>$3.5710e-01$</td>
<td>$0.00e+00$</td>
<td>$1.46e+05$</td>
<td>$1.10e+00$</td>
<td>372</td>
</tr>
<tr>
<td>11</td>
<td>$3.5060e-01$</td>
<td>$0.00e+00$</td>
<td>$2.54e+05$</td>
<td>$1.66e+00$</td>
<td>593</td>
</tr>
<tr>
<td>12</td>
<td>$3.4772e-01$</td>
<td>$0.00e+00$</td>
<td>$4.39e+05$</td>
<td>$2.88e+00$</td>
<td>677</td>
</tr>
<tr>
<td>13</td>
<td>$3.4627e-01$</td>
<td>$0.00e+00$</td>
<td>$7.61e+05$</td>
<td>$5.02e+00$</td>
<td>886</td>
</tr>
<tr>
<td>14</td>
<td>$3.4556e-01$</td>
<td>$0.00e+00$</td>
<td>$1.32e+06$</td>
<td>$8.76e+00$</td>
<td>932</td>
</tr>
<tr>
<td>14</td>
<td>$3.4556e-01$</td>
<td>$0.00e+00$</td>
<td>$1.32e+06$</td>
<td>$8.76e+00$</td>
<td>4215</td>
</tr>
</tbody>
</table>
Results of the Logarithmic Barrier Method

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>max $c(x_k)$</th>
<th>$t_k$</th>
<th>$\bar{\varepsilon}_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.6235e-01</td>
<td>5.00e+00</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>6.0771e-01</td>
<td>0.00e+00</td>
<td>9.19e+02</td>
<td>6.90e-01</td>
<td>193</td>
</tr>
<tr>
<td>2</td>
<td>6.1142e-01</td>
<td>0.00e+00</td>
<td>1.59e+03</td>
<td>1.35e+00</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>6.1380e-01</td>
<td>0.00e+00</td>
<td>2.76e+03</td>
<td>4.55e+00</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5.4242e-01</td>
<td>0.00e+00</td>
<td>4.77e+03</td>
<td>2.16e+01</td>
<td>37</td>
</tr>
<tr>
<td>5</td>
<td>3.8645e-01</td>
<td>0.00e+00</td>
<td>8.27e+03</td>
<td>6.75e+01</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>2.8214e-01</td>
<td>0.00e+00</td>
<td>1.43e+04</td>
<td>1.70e+03</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>2.1261e-01</td>
<td>0.00e+00</td>
<td>2.48e+04</td>
<td>1.71e+03</td>
<td>65</td>
</tr>
<tr>
<td>8</td>
<td>1.6892e-01</td>
<td>0.00e+00</td>
<td>4.30e+04</td>
<td>2.79e+02</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>1.3955e-01</td>
<td>0.00e+00</td>
<td>7.44e+04</td>
<td>2.78e+03</td>
<td>513</td>
</tr>
</tbody>
</table>

Table C-5: BFGS-Barrier method applied to test case 1, with $\mu = \sqrt{3}$, $\varepsilon = 0.1$, and $\sigma = 0.3$
<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>max $c(x_k)$</th>
<th>$t_k$</th>
<th>$\varepsilon_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.3274e-02</td>
<td>9.09e+01</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.3349e+01</td>
<td>2.67e+01</td>
<td>9.72e-01</td>
<td>3.90e-02</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>9.2048e+00</td>
<td>3.60e+01</td>
<td>1.68e+00</td>
<td>1.86e-02</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>1.0732e+01</td>
<td>3.19e+01</td>
<td>2.92e+00</td>
<td>8.38e+00</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>1.3920e+01</td>
<td>2.16e+01</td>
<td>5.05e+00</td>
<td>1.52e+01</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>1.2510e+01</td>
<td>1.27e+01</td>
<td>8.75e+00</td>
<td>1.02e+02</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>1.4790e+01</td>
<td>6.94e+00</td>
<td>1.52e+01</td>
<td>1.96e+02</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>1.2309e+01</td>
<td>3.36e+00</td>
<td>2.62e+01</td>
<td>4.43e+02</td>
<td>58</td>
</tr>
<tr>
<td>8</td>
<td>1.1211e+01</td>
<td>1.32e+00</td>
<td>4.55e+01</td>
<td>2.37e+01</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>1.1368e+01</td>
<td>1.59e-01</td>
<td>7.87e+01</td>
<td>6.43e+00</td>
<td>49</td>
</tr>
<tr>
<td>10</td>
<td>9.2919e+00</td>
<td>1.60e-04</td>
<td>1.36e+02</td>
<td>1.79e+02</td>
<td>117</td>
</tr>
<tr>
<td>11</td>
<td>7.7720e+00</td>
<td>0.00e+00</td>
<td>2.36e+02</td>
<td>4.94e+02</td>
<td>72</td>
</tr>
<tr>
<td>12</td>
<td>6.1659e+00</td>
<td>0.00e+00</td>
<td>4.09e+02</td>
<td>1.34e+02</td>
<td>142</td>
</tr>
<tr>
<td>13</td>
<td>4.9596e+00</td>
<td>0.00e+00</td>
<td>7.09e+02</td>
<td>5.01e+02</td>
<td>221</td>
</tr>
<tr>
<td>14</td>
<td>4.2565e+00</td>
<td>0.00e+00</td>
<td>1.23e+03</td>
<td>4.13e+02</td>
<td>83</td>
</tr>
<tr>
<td>15</td>
<td>3.5580e+00</td>
<td>0.00e+00</td>
<td>2.13e+03</td>
<td>8.54e+01</td>
<td>323</td>
</tr>
<tr>
<td>16</td>
<td>3.2454e+00</td>
<td>0.00e+00</td>
<td>3.68e+03</td>
<td>9.63e+02</td>
<td>151</td>
</tr>
<tr>
<td>17</td>
<td>2.9539e+00</td>
<td>0.00e+00</td>
<td>6.38e+03</td>
<td>1.80e+03</td>
<td>476</td>
</tr>
<tr>
<td>18</td>
<td>2.8869e+00</td>
<td>0.00e+00</td>
<td>1.10e+04</td>
<td>3.60e+16</td>
<td>75</td>
</tr>
<tr>
<td>19</td>
<td>2.8878e+00</td>
<td>0.00e+00</td>
<td>1.91e+04</td>
<td>3.00e+05</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>2.8805e+00</td>
<td>0.00e+00</td>
<td>3.31e+04</td>
<td>7.21e+16</td>
<td>19</td>
</tr>
<tr>
<td>21</td>
<td>2.8825e+00</td>
<td>0.00e+00</td>
<td>5.74e+04</td>
<td>1.00e+06</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>2.8451e+00</td>
<td>0.00e+00</td>
<td>9.94e+04</td>
<td>1.44e+17</td>
<td>145</td>
</tr>
<tr>
<td>22</td>
<td>2.8451e+00</td>
<td>0.00e+00</td>
<td>9.94e+04</td>
<td>1.44e+17</td>
<td>2127</td>
</tr>
</tbody>
</table>

Table C-6: BFGS-Barrier method applied to test case 2, with $\mu = \sqrt{3}$, $\epsilon = 0.1$ and $\sigma = 0.3$
### Results of the Logarithmic Barrier Method

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>max $c(x_k)$</th>
<th>$t_k$</th>
<th>$\epsilon_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.3015e-01</td>
<td>1.59e+00</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>5.0651e-01</td>
<td>1.68e+00</td>
<td>3.26e+03</td>
<td>9.01e+08</td>
<td>92</td>
</tr>
<tr>
<td>2</td>
<td>4.8622e-01</td>
<td>0.00e+00</td>
<td>5.65e+03</td>
<td>1.11e+01</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>3.7738e-01</td>
<td>0.00e+00</td>
<td>9.79e+03</td>
<td>5.36e+02</td>
<td>224</td>
</tr>
<tr>
<td>4</td>
<td>2.7960e-01</td>
<td>0.00e+00</td>
<td>1.70e+04</td>
<td>2.56e+02</td>
<td>39</td>
</tr>
<tr>
<td>5</td>
<td>2.1480e-01</td>
<td>0.00e+00</td>
<td>2.94e+04</td>
<td>1.60e+03</td>
<td>94</td>
</tr>
<tr>
<td>6</td>
<td>1.7689e-01</td>
<td>0.00e+00</td>
<td>5.09e+04</td>
<td>2.81e+03</td>
<td>46</td>
</tr>
<tr>
<td>7</td>
<td>1.4891e-01</td>
<td>0.00e+00</td>
<td>8.81e+04</td>
<td>4.22e+03</td>
<td>244</td>
</tr>
<tr>
<td>8</td>
<td>1.3501e-01</td>
<td>0.00e+00</td>
<td>1.53e+05</td>
<td>7.34e+03</td>
<td>236</td>
</tr>
<tr>
<td>9</td>
<td>1.2836e-01</td>
<td>0.00e+00</td>
<td>2.64e+05</td>
<td>8.42e+03</td>
<td>210</td>
</tr>
<tr>
<td>9</td>
<td>1.2836e-01</td>
<td>0.00e+00</td>
<td>2.64e+05</td>
<td>8.42e+03</td>
<td>1197</td>
</tr>
</tbody>
</table>

**Table C-7:** BFGS-Barrier method applied to test case 3, with $\mu = \sqrt{3}$, $\epsilon = 0.1$ and $\sigma = 0.3$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x_k)$</th>
<th>max $c(x_k)$</th>
<th>$t_k$</th>
<th>$\epsilon_k$</th>
<th>$N_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9.1183e+00</td>
<td>1.14e+02</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1.3275e-01</td>
<td>6.81e+00</td>
<td>1.04e+03</td>
<td>6.75e-03</td>
<td>82</td>
</tr>
<tr>
<td>2</td>
<td>1.7972e-01</td>
<td>9.10e+00</td>
<td>1.81e+03</td>
<td>1.83e-01</td>
<td>48</td>
</tr>
<tr>
<td>3</td>
<td>2.3838e-01</td>
<td>1.38e+01</td>
<td>3.13e+03</td>
<td>1.60e-01</td>
<td>76</td>
</tr>
<tr>
<td>4</td>
<td>3.2942e-01</td>
<td>5.69e+00</td>
<td>5.42e+03</td>
<td>9.26e-02</td>
<td>99</td>
</tr>
<tr>
<td>5</td>
<td>4.3369e-01</td>
<td>1.33e+00</td>
<td>9.39e+03</td>
<td>2.98e-01</td>
<td>157</td>
</tr>
<tr>
<td>6</td>
<td>5.0955e-01</td>
<td>1.27e-01</td>
<td>1.63e+04</td>
<td>4.21e+01</td>
<td>211</td>
</tr>
<tr>
<td>7</td>
<td>4.4907e-01</td>
<td>0.00e+00</td>
<td>2.82e+04</td>
<td>1.92e-01</td>
<td>209</td>
</tr>
<tr>
<td>8</td>
<td>3.9028e-01</td>
<td>0.00e+00</td>
<td>4.88e+04</td>
<td>9.41e+00</td>
<td>225</td>
</tr>
<tr>
<td>9</td>
<td>3.5547e-01</td>
<td>0.00e+00</td>
<td>8.45e+04</td>
<td>2.31e+01</td>
<td>330</td>
</tr>
<tr>
<td>10</td>
<td>3.5555e-01</td>
<td>0.00e+00</td>
<td>1.46e+05</td>
<td>2.31e+02</td>
<td>451</td>
</tr>
<tr>
<td>11</td>
<td>3.2506e-01</td>
<td>0.00e+00</td>
<td>2.54e+05</td>
<td>9.44e+01</td>
<td>518</td>
</tr>
<tr>
<td>12</td>
<td>3.2001e-01</td>
<td>0.00e+00</td>
<td>4.39e+05</td>
<td>5.54e-01</td>
<td>563</td>
</tr>
<tr>
<td>13</td>
<td>3.1786e-01</td>
<td>0.00e+00</td>
<td>7.61e+05</td>
<td>9.57e+02</td>
<td>596</td>
</tr>
<tr>
<td>14</td>
<td>3.1688e-01</td>
<td>0.00e+00</td>
<td>1.32e+06</td>
<td>1.15e+04</td>
<td>590</td>
</tr>
<tr>
<td>14</td>
<td>3.1688e-01</td>
<td>0.00e+00</td>
<td>1.32e+06</td>
<td>1.15e+04</td>
<td>4155</td>
</tr>
</tbody>
</table>

**Table C-8:** BFGS-Barrier method applied to test case 4, with $\mu = \sqrt{3}$, $\epsilon = 0.1$ and $\sigma = 0.3$. 


