INFEASIBILITY AND NEGATIVE CURVATURE IN OPTIMIZATION

A DISSERTATION
SUBMITTED TO THE PROGRAM IN SCIENTIFIC COMPUTING AND COMPUTATIONAL MATHEMATICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Erik G. Boman
February 1999

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Walter Murray
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Gene H. Golub

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Michael A. Saunders

Approved for the University Committee on Graduate Studies:

Abstract

Infeasibility and Negative Curvature in Optimization.
Erik G. Boman, Stanford University, 1999.

It may seem reasonable that only problems with a solution can be solved. However, in practice it may be that some type of "solution" is needed even when the problem is ill-posed and no solution exists. Our concern is with constrained optimization problems that do not have any feasible points. (We then say the problem is *infeasible*.) It may be that a small perturbation of some of the constraints would yield a solution, and determining such a perturbation might produce the approximate solution required. However, by allowing a slightly larger pertubation in the constraints, we might find a much improved value of the objective function. It is quite possible that a "slightly infeasible solution" of this kind would be welcomed by the user. In this thesis we address such issues of infeasibility.

Sequential quadratic programming (SQP) methods are a class of methods for solving constrained nonlinear optimization problems. They have been very successful in practice, and are widely regarded as the most efficient direct approach to constrained optimization. As their name suggests, SQP methods are iterative and solve a quadratic program (QP) at each iteration. In convergence analysis it is usually assumed that the QP subproblems always have a feasible point. This assumption is not warranted even when the original nonlinear problem has a feasible point. When the original problem is infeasible, it is almost inevitable that the QP subproblems will be infeasible. One may think this acceptable because if there is no solution to the problem posed, it should not be surprising that an algorithm might break down. However, in practice we would still like a point that in some sense is "good". This thesis addresses the issue of how to proceed when the QP subproblems are infeasible and how to determine an approximate solution acceptable to a user when the problem does not have a feasible point.

The approach taken is to introduce additional variables into the problem. This can be done in several ways. Each option has different properties, for example with respect to the linear algebra operations involved to solve the QP subproblem. A key issue is how to measure infeasibility and feasibility. Some information provided by the user is helpful here because the measure will differ with the application. We describe and analyze two variants of the so-called *elastic* approach. The first variant is based on the l_1 norm, and the second on the l_∞ norm.

A concern when elastic variables are added is that we may converge to a point in the elastic formulation satisfying the first-order optimality conditions that is *not* feasible for the original problem. At such a point, no direction of descent exists for the measure of infeasibility being used. The only way to proceed is to determine a direction of negative curvature. In Chapters 4 and 5 we focus on directions of negative curvature and how they can be computed efficiently.

Our treatment of negative curvature is not directly based on the motivation just described. Rather, we examine directions of negative curvature in the more general context of unconstrained optimization. When second derivatives are known we wish to be able to determine a point that satisfies the second-order (necessary) conditions for a minimizer. It can be shown that an essential feature of such algorithms is to be able to compute a direction of negative curvature for a symmetric matrix. We describe some new work on how to compute such directions. In particular, we show how, with little effort, some iterative methods can be used to obtain a *good* direction of negative curvature from a poor direction.

Acknowledgements

I would like to express my deep gratitude towards my advisor, Walter Murray, for his guidance, inspiration, and enthusiasm. His influence is apparent in every part of this thesis. Almost all I know about optimization is due to him. A busy professor, he would always generously take time to talk with me, though frequently we digressed onto topics like English football or skiing!

I am deeply indebted to Gene Golub, who was the director and driving force behind the SCCM Program for many years. Gene not only taught me a lot about numerical linear algebra and gave me intellectual inspiration, but also showed a unique personal generousity that made me feel at home at Stanford.

Michael Saunders did a superb job serving on my reading committee, suggesting numerous improvements in this thesis. His detailed understanding is very impressive, and in hindsight I realize that I should have used him as a resource more often.

I would also like to thank Petter Bjørstad at the University of Bergen, who first awoke my interest in scientific computing. After I finished my Cand. Scient. (M.S.) degree in Bergen, he cleared the way for me to come to Stanford. I am grateful to Gene Golub and Mary Washburn for organizing what must have been one of the quickest admissions ever.

Thanks to all my fellow students and postdocs in SCCM who made these years at Stanford so enjoyable, of which there are too many to mention by name. Best wishes to all my good friends from the Hammarskjöld House and the Stanford Outing Club who provided me a happy social life.

This thesis was financially supported in part by the Research Council of Norway under grant no. 110860/410. I am also grateful to the Stanford School of Engineering for awarding me a fellowship in my first year at Stanford.

Last, but not least, I would like to thank my parents, who prepared me well for the long and difficult task of earning a PhD.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We consider the nonlinear optimization problem

$$\begin{array}{lll} \textbf{NLP:} & \displaystyle\min_{x\in\mathbb{R}^n} & F(x) \\ & \text{s.t.} & c(x) \geq 0, \end{array} \tag{1.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$. We will assume that both $F(x)$ and $c(x)$ are smooth, i.e. they are at least twice differentiable. Although only inequality constraints are specified, in general we may also have some equality constraints. In theory, an equality $h(x) = 0$ can be transformed into a pair of inequalities $h(x) \geq 0$ and $-h(x) \geq 0$, but in practice it is usually better to treat equalities separately. Another formulation of NLP that is commonly used in software packages is

$$\begin{array}{ll} \displaystyle\min_{x\in\mathbb{R}^n} & F(x) \\ \text{s.t.} & l \leq \begin{pmatrix} x \\ Ax \\ c(x) \end{pmatrix} \leq u, \end{array} \tag{1.2}$$

where $F : \mathbb{R}^n \to \mathbb{R}$, $A : \mathbb{R}^n \to \mathbb{R}^{m_1}$ is linear, $c : \mathbb{R}^n \to \mathbb{R}^{m_2}$ is nonlinear, and the lower and upper bounds l and u are vectors of appropriate length. Since these two formulations are equivalent we will use them interchangeably. The former version is most convenient to analyze from a theoretical point of view, but since we are also concerned with efficiency in practical algorithms we sometimes need the latter version as well.

Many types of algorithms have been proposed to solve problems of the type NLP. Good general references are [20] and [27]. In this dissertation we focus on SQP (Sequential Quadratic Programming) methods. We note that NLP can be a very difficult problem to solve; for instance, it is NP-hard in the traditional complexity model [80]. Nevertheless, there are several algorithms for nonlinear programming that work quite well in practice. Most algorithms attempt to solve NLP by solving a sequence of simpler optimization problems (subproblems). In this thesis we are only concerned with finding local minima. Global optimization is a much more difficult problem, beyond the scope of this thesis.

Although the work presented here is targeted at SQP methods, we believe many of the principles are applicable to other algorithms that are based on solving a sequence of subproblems.

1.1 Infeasibility issues

1.1.1 Our motivation

Given an instance of NLP it is hard (in fact NP-hard) to decide even whether a solution exists. A solution exists if and only if there is a *feasible point*. If a feasible point exists, we say that the problem is *feasible*, otherwise it is *infeasible*.

Our focus is on two different, but related, issues related to feasibility and infeasibility. For the first issue, we assume an optimization method that solves a sequence of subproblems (e.g. SQP, see Section 1.3). Even if the original nonlinear problem is feasible, some of the subproblems may be infeasible. The algorithm should not stop in this case. Consequently, some type of "approximate solution" of the infeasible subproblems is needed. In other words, we need to do something reasonable with the inner problem to ensure that the outer iteration can continue. An obvious approach is to modify the subproblems. This is the topic of Chapters 2 and 3. A brief overview of earlier work is given in Section 1.4.

The second issue is what to do when the problem itself is infeasible, not just a subproblem. That is, we cannot assume the infeasibility will disappear, since no feasible point exists. Curiously, it may even happen that all subproblems are feasible. Optimization problems that model real-world situations are usually feasible, but this is not always the case. It may happen that there is no solution to the real problem, or the mathematical model formulation has no solution. There is no simple way to detect when this happens. Typically, the user tries to solve the model problem not knowing whether a solution exists or not. The output from an optimization routine will normally (but not always) be a local optimum if the problem is feasible. But when the problem is infeasible (or some other difficulty arises), many software packages simply terminate with a message saying "no feasible point" or similar. Such behavior is not very helpful since the user often needs some "answer", even when none exists. This is the case in many real-time and on-line situations; some decision always has to be made (not necessarily optimal). One possibility, implemented in some codes, is to switch to minimizing the infeasibilities (in some norm) when it appears that no feasible point exists. This is a bit more useful. However, what the user often would like is an "approximate solution" that is near feasible and at the same time produces a reasonably good objective value. We aim to satisfy such users. Our proposed method will not abruptly switch from one objective function to another, but rather change the objective smoothly. If a "near feasible" point is not found, this is an indication that the model from which the problem was derived may be poor. Even if one has a good mathematical model of a real-life system, the resulting optimization problem may be infeasible under some circumstances. A similar but slightly different situation is when the real-life problem has a solution but the mathematical problem does not because of inaccuracies in the model. In both these cases our method can prove valuable.

A third issue that is not generally appreciated is that even if a subproblem is feasible, we may not wish to move to a feasible point. This may seem counter-intuitive, but we show an example in Section 1.1.3.

Finally, we give an example of an application area. Utility companies must plan the operation of power plants and the distribution of electrical power to their customers. This is often split into two optimization problems, where the first decides which plants and generators to turn on and how to operate them, while the second determines the distribution of the power over a network to the customers. The goal may be to minimize the overall cost or to maximize profit. The constraints are of many types. One category is physical laws like Ohm's and Kirchhoff's laws, while a different type is conditions like "the end-user voltage should be 110–115 volts". It can happen that the problem is infeasible; e.g., there could be insufficient resources to satisfy the demand. Clearly, one must do something reasonable and not just give up. Some constraints cannot be violated, for example the physical (electromagnetic) laws, so some of the user-imposed constraints must be relaxed or violated. One option is to let the voltage fall below the lowest threshold (a so-called *brown-out*). An alternative is to cut the power to some customers completely. Which is the better solution depends on the metric used to measure the constraint violations.

1.1.2 Inconsistent constraints

In constrained optimization, it may happen that no feasible point exists. If so, we say the constraints are *inconsistent* and the problem is *infeasible*. (Informally we may say the constraints are infeasible.) Since this is an important topic in this thesis, we summarize some well-known properties about infeasibility and inconsistency in this section.

If all constraints are bounds on the variables, i.e.,

$$l \leq x \leq u, \tag{1.3}$$

it is trivial to determine if a feasible point exists. The jth constraint is inconsistent if and only if $l_j > u_j$.

Consider the case where all constraints are linear. We need to distinguish between equality and inequality constraints. Consider first the equality constraints

$$Ax = b, \tag{1.4}$$

where A is m by n. When $m \leq n$ and A has full rank, a feasible region always exists. This feasible region is isomorphic to $\mathbb{R}^{n-m}$. A special case is when $m = n$, where the feasible region is the single point $A^{-1}b$. In general, the constraints are inconsistent if and only if $\operatorname{rank}([A\ b]) > \operatorname{rank}(A)$, i.e., there exists a vector y such that $A^T y = 0$ and $b^T y \neq 0$.

Consider next linear inequalities,

$$Ax \geq b. \tag{1.5}$$

The feasible region is a convex polyhedron, possibly empty. As with equalities, a feasible region always exists when $\operatorname{rank}(A) = m \leq n$. The constraints are inconsistent if and only if there exists a vector $y \geq 0$ such that $A^T y = 0$ and $b^T y > 0$. The most common approach to finding a feasible point is to solve an LP that minimizes the sum of infeasibilities.

With nonlinear constraints, there is no simple characteristic that tells us whether a feasible point exists. The feasibility problem is as hard as NLP. Since SQP algorithms for nonlinear optimization linearize the nonlinear constraints, the linear case is most important to us.

Some work has been done on identifying (irreducibly) inconsistent sets of constraints, mostly for linear inequalities [78, 8], but also for nonlinear programs [9]. One motivation is

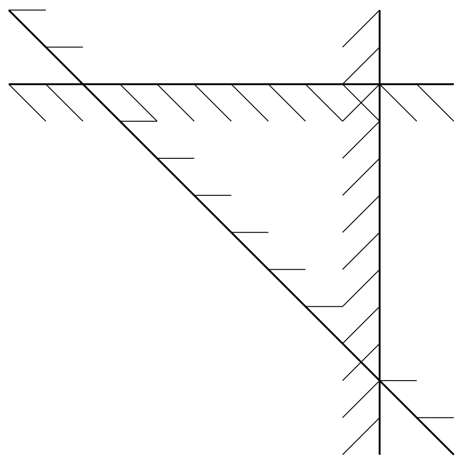

Figure 1.1: An infeasible problem.

that there may be errors in the formulation of a large optimization problem, and a tool that identifies a small set of inconsistent constraints can assist the user in detecting errors in the model. The same methods can also be used to determine how to modify the constraints such that a feasible point exists.

1.1.3 Geometric examples

We show some examples of infeasible problems. For now we ignore the objective and show only the (inconsistent) constraints.

Example 1.1.1

$$x_1 \geq 0 \tag{1.6}$$

$$x_2 \geq 0 \tag{1.7}$$

$$-x_1 - x_2 - 1 \geq 0 \tag{1.8}$$

These linear inequalities are illustrated in Figure 1.1. The first two constraints imply a feasible point must lie in the first quadrant, while the last constraint defines an infeasible half-plane that contains the entire first quadrant. Consequently, no point is feasible with respect to all three constraints. However, if any one of the constraints is removed, the problem becomes feasible. We are interested in finding points that are "near-feasible". Such points can be defined in various ways. Two approaches discussed in the next chapters are to minimize the constraint violations in the l_∞ and l_1 norms, respectively. In this example, one can show the l_∞-norm of the constraint violations is minimized by $x = (-1/3, -1/3)$, while any point inside the triangle given by the vertices $(0, -1)$, $(-1, 0)$ and $(0, 0)$ is a minimizer in the l_1-norm.

Example 1.1.2

$$x_1^2 + (x_2 - 2)^2 \leq 1 \tag{1.9}$$

$$x_1^2 + (x_2 + 2)^2 \leq 1 \tag{1.10}$$

These nonlinear constraints define two unit circles centered at $(0, -2)$ and $(0, 2)$, respectively. Clearly, no point can lie inside both circles because they do not intersect, so the constraints are inconsistent. The origin, $x = (0, 0)$, is a least infeasible point in any of the measures we consider.

As we shall see in Section 1.3, SQP methods create subproblems with constraints that are linearizations of the nonlinear constraints. We therefore consider different linear approximations of (1.9)–(1.10).

Suppose the current iterate is $x = (0, 0)$. Then the geometry of the problem is symmetric around the line $x_1 = 0$, and a linearization of the two circles produces two parallel lines as shown on the top of Figure 1.2. The QP subproblem must then be infeasible.

Suppose next that the current iterate is slightly above the line $x_2 = 0$. The two resulting lines for the linearized constraints are no longer parallel but slightly tilted. They intersect at a point where x_2 is large and negative. This intersection point is a vertex that defines the "top" of the feasible region (see bottom part of Figure 1.2). This demonstrates that a subproblem may be feasible even if the outer problem is not. However, in this example it would be unwise to move to a feasible point because the feasible region lies far from the region of interest with small constraint violations for the nonlinear problem.

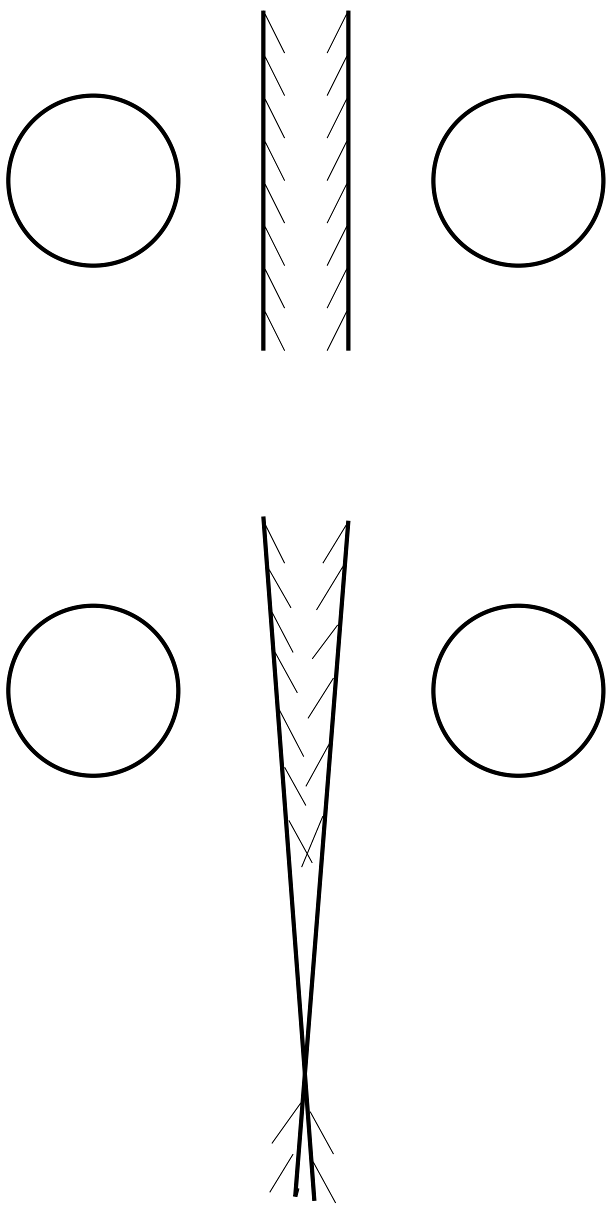

Figure 1.2: Top subproblem is infeasible; bottom subproblem is feasible, but it is not advisable to move to a feasible point.

1.2 Optimality conditions for NLP

We review the necessary and sufficient conditions for optimality. Most optimization algorithms are designed to try to satisfy these *optimality conditions.* Before we state the conditions, we need some terminology. An inequality constraint $c_i(x) \geq 0$ is *active* at x if $c_i(x) = 0$. If $c_i(x) > 0$, it is *inactive.* By convention, all equality constraints $c_i(x) = 0$ are active at a feasible point.

A point x^* is a *local minimizer* (solution) to NLP if $c(x^*) \geq 0$ and there exists a $\delta > 0$ such that $F(x) \geq F(x^*)$ for all x satisfying

$$\|x - x^*\| \leq \delta \quad \text{and} \quad c(x) \geq 0. \tag{1.11}$$

We will usually (implicitly) assume that the Jacobian of the active constraints at x^*, $\hat{J}$, has full rank. We note that this condition does not always hold, but it is a common assumption in convergence proofs.

Necessary conditions for x^* to be a local minimizer are that there exist multipliers λ^* such that

$$c(x^*) \geq 0, \tag{1.12}$$

$$\hat{J}^T\lambda^* = g(x^*), \quad [\text{or } Z^T g(x^*) = 0] \tag{1.13}$$

$$\lambda^* \geq 0, \tag{1.14}$$

$$Z^T W(x^*, \lambda^*) Z \text{ is positive semidefinite,} \tag{1.15}$$

where $g = \nabla F(x)$, Z is a basis for the nullspace of $\hat{J}$, and $W(x, \lambda)$ is the Hessian of the Lagrangian $L(x, \lambda)$. These conditions are called the *second-order KKT (Karush-Kuhn-Tucker) conditions.* The first-order conditions are (1.12–1.14).

If we replace (1.14) and (1.15) by

$$\lambda^* > 0, \tag{1.16}$$

$$Z^T W(x^*, \lambda^*) Z \text{ is positive definite} \tag{1.17}$$

we obtain sufficient conditions for optimality.

We note that the methods in the first three chapters of this dissertation do not use exact (analytical) second derivatives, and convergence can only be proven to first-order optimality points. In the last chapter we discuss second-order methods that use directions of negative curvature.

1.3 SQP algorithms

SQP algorithms can be viewed as generalizations of Newton's method to the (inequality-) constrained case. The basic idea is to construct a constrained quadratic model around the current iterate x_k. Minimization of this quadratic model gives us an update p_k that approximates the error $x^* - x_k$. Our new iterate is $x_{k+1} = x_k + \alpha_k p_k$, where α_k is usually determined by a line search. Trust-region methods can also be used.

In other words, SQP methods solve a sequence of quadratic programs (QPs). Each QP has linear constraints and a quadratic objective. The QP constraints are linearizations of the nonlinear constraints at x_k. It is important to note that the quadratic objective is an approximation to the Lagrangian $L(x, \lambda)$, and not to the objective function $F(x)$. The kth QP subproblem for NLP can be stated as

$$\begin{array}{lll} \textbf{QP:} & \min\limits_{p} & g_k^T p + p^T H_k p \\ & \text{s.t.} & J_k p \geq -c_k, \end{array} \tag{1.18}$$

where $g_k = \nabla F(x_k)$, J_k is the Jacobian of c at x_k, and H_k is an approximation to the Hessian of the Lagrangian.

1.3.1 Brief SQP history

The origins of SQP methods can be traced back to Wilson's doctoral thesis in 1963 [82]. Wilson was only concerned with convex problems and his algorithm required exact Hessians. In 1969, Murray [49] proposed an SQP algorithm that used a quasi-Newton approximation of the Hessian. He also suggested using a merit function with a line search.

SQP methods received a lot of attention in the 1970s. Of particular significance are the papers by Biggs [3], Han [33] and Powell [61, 62]. While most SQP methods produce QPs with inequality constraints, Biggs developed an algorithm with equality-constrained QPs. Han proved convergence for a class of SQP algorithms, and Powell described an SQP algorithm for which he could prove (local) superlinear convergence.

1.3.2 Merit function and line search

Any algorithm for constrained optimization needs to ensure that the algorithm converges to a point that is both feasible and a (constrained) minimizer. To measure how good a given point is, we cannot simply look at the objective function because that does not say anything about how close to the feasible set we are. The standard approach today is to define a *merit function*, which combines both the objective and the constraint violations into one function. Merit functions are closely related to penalty functions, which we will study in Section 2.2. Here we assume for simplicity that all constraints are equalities of the form $c(x) = 0$, and briefly mention that the two most popular merit functions for SQP methods are the l_1 merit function

$$M(x) = F(x) + \rho \|c(x)\|_1, \tag{1.19}$$

and the augmented Lagrangian

$$M(x, \lambda) = F(x) - \lambda^T c(x) + \frac{\rho}{2} c(x)^T c(x), \tag{1.20}$$

where $\rho > 0$ is a penalty parameter and λ is a set of Lagrange multiplier estimates. In practice, a more complicated version of the augmented Lagrangian merit function is used.

We focus on algorithms that use a *line search* (the main alternative is the trust-region approach). That is, after a search direction p has been determined, we search along the line $x + \alpha p$ in order to find a good step length α. The merit function is used in the line search to determine the step α. Most convergence proofs for SQP methods rely on proving a sufficient decrease in the merit function at each iteration.

1.3.3 Multiplier estimates

The SQP methods that we focus on rely on Lagrange multiplier estimates. Instead of only searching in the primal space for x^*, these methods generate iterates (x_k, λ_k) that we want to converge to (x^*, λ^*). Hence we are searching in a higher-dimensional space than that of x. This basic idea is also the foundation of primal-dual methods. It is important that we compute accurate multiplier estimates, since x cannot converge to x^* any faster than $\lambda \to \lambda^*$.

In SQP methods we need to distinguish between the multipliers of the nonlinear problem and those of the QP. In each QP minor iteration, the QP multiplier estimates are updated. After solving a QP, the QP multipliers can then be used to define the search direction in the Lagrange multiplier (dual) space.

1.3.4 Quasi-Newton update of the Hessian

Most SQP methods do not require exact second derivatives because these may be unavailable or expensive to compute. Another potential problem is that the exact Hessian may be indefinite. Standard SQP methods assume H is positive semidefinite and thus the QP subproblems are convex, but an algorithm with nonconvex QPs has recently been developed [28]. We focus on SQP methods where at each iteration, a positive definite *quasi-Newton* approximation of the Hessian is used. For an overview of quasi-Newton methods, see for example [13].

1.3.5 Active-set approach

There are two main strategies for handling constraints in optimization problems. In *interior-point* (barrier) methods, the iterates x are strictly feasible with respect to inequality constraints. The problem of finding an initial feasible point remains. If no such point is known, the problem is typically augmented with an artificial variable that creates an obvious feasible point for the augmented problem. When both equalities and inequalities are present, not all interior-point methods are feasible-point methods; infeasible-interior-point methods also exist.

A different approach is taken in *active-set* methods. The idea is to predict which constraints are active (that is, equality holds) at the solution x^*. The algorithm maintains a set of constraints that are forced to be active at each iteration. This set is called the *working set*. The working set may change at each iteration. However, active-set methods are often very efficient in practice because the working set changes little when the iterates are close to a minimizer.

1.3.6 A simple SQP algorithm

Based on the previous subsections, we outline a pedagogical (simplified) SQP algorithm. Implementations of SQP algorithms tend to be much more complicated.

Algorithm 1.3: *Simple SQP algorithm.*

Choose x_0, λ_0 and H_0. Set $k = 0$.
while the KKT conditions are not satisfied
  Set up a QP subproblem at x_k. Compute a search direction p_k
    and Lagrange multipliers μ_k.
  Compute a step length α that reduces the merit function.
  Update $x_{k+1} = x_k + \alpha p_k$.
  Use μ_k to update the multiplier estimates λ_k.
  Set $k = k + 1$.
  Evaluate $c(x_k)$, $g(x_k)$, $J(x_k)$.
  Update H_k, the Hessian approximation.
end

1.4 Related infeasibility work

It was realized quite early that a deficiency with SQP algorithms is that the QP subproblems may not have a feasible point. Many ways to remedy this problem have been suggested, though the topic has never been a central theme in the field of SQP algorithms.

Consider the standard NLP with inequality constraints (1.1). Powell [60] suggested modifying the linearized constraints in the following way:

$$\xi c_i(x_k) + a_i^T p \geq 0, \quad i \in V, \tag{1.21}$$

$$c_i(x_k) + a_i^T p \geq 0, \quad i \notin V, \tag{1.22}$$

where $a_i = \nabla c_i(x_k)$ and V is the set of constraints that are violated at x_k. ξ is a scalar between 0 and 1. When $\xi = 1$, these are the standard QP constraints, which may not be feasible. With $\xi < 1$, the violated constraints are relaxed. One attempts to use a ξ that is close to the largest ξ for which a feasible point of (1.21) exists.

Several authors (Fletcher [18, 19], Powell [63]) have proposed replacing the constrained QP with an unconstrained QP. The most popular approach has been to add a l_1 penalty term to the QP objective and remove the constraints. Such a method, with the addition of a trust-region constraint to bound the size of the search direction, was proposed by Fletcher as the Sl_1QP method.

Tone [77] introduced additional variables to represent the violation in each constraint.

He proposed adding a *big M* penalty term in the objective in order to keep the constraint violations small. He also suggested a way to combine this method with that of Powell. The approach we later suggest is similar in spirit to Tone's, but on a detailed level it is quite different.

Burke and Han [6] developed a robust SQP method in which the QPs are modified such that a feasible point exists. In order to compute the modification, an additional optimization problem has to be solved. This problem is either an LP or a QP, depending on the norm used in their algorithm. This approach can be fairly expensive, but is important because the authors provided a convergence proof for their algorithm under quite general assumptions. They show an example where their robust SQP method works but the Sl_1QP method fails. An algorithm very similar to the Burke and Han method has been described by Zhou [84].

Recently, Spellucci [74, 73] has proposed an SQP algorithm that handles infeasible QPs, using an approach similar to that of Tone. This method has been implemented in the code DONLP2 (available from http://plato.la.asu.edu/donlp2.html).

The approach described in Chapter 2 is closely related to that incorporated in the large-scale SQP code SNOPT [25]. SNOPT incorporates something called *elastic mode*, which allows certain constraints to be violated. We describe this approach in detail in the next chapter.

A basic question underlying all strategies that attempt to modify infeasible subproblems is whether one first attempts to solve the original subproblem before deciding to modify, or one always modifies the subproblem. The first strategy is inefficient in the sense that one has to solve two subproblems instead of one. A way to avoid the doubling of work is to solve the feasibility problem as a special subproblem first. In SQP methods, the feasibility problem for the QP subproblem is an LP, which in general is faster to solve than a QP. An alternative is to solve only QP subproblems that are known to have a solution. The "always modify" approach falls into this category. In some algorithms, computing the required modification is a costly optimization problem in itself. We seek a method where the modification is so simple that the extra cost is negligible.

A closely related topic is the analysis of SQP methods in the case where the QP subproblems are degenerate, and possibly the NLP itself. For example, a standard assumption is that the Jacobian of the active constraints is nonsingular, but this is not always the case. Wright [83] has described and analyzed an inexact SQP algorithm designed for such cases.

Temporary and permanent infeasibility

The efforts described in the previous section are mainly concerned with "temporary" infeasibility. That is, even when a subproblem in an optimization algorithm is determined to be infeasible, it is assumed that the original problem is feasible. How the infeasibility is overcome (e.g., by modifying the subproblem) is not very important because eventually the infeasibility will disappear.

On the other hand, if the original NLP itself is infeasible, one would expect infeasible subproblems to be a persistent phenomenon. This can be viewed as "permanent" infeasibility. In such cases, an approximate solution that is (slightly) infeasible is the best answer one can find. Such an approximate solution is the solution to a nearby perturbed problem. However, in general it is not known how the problem should be perturbed. Different modifications of the subproblems will clearly produce different "solutions".

Our goal is to devise a single strategy that addresses both types of infeasibility simultaneously.

Chapter 2

The l_1-elastic approach

We are interested in two situations. In the first case, we have a nonlinear problem (NLP) whose feasible set may be empty. The second case concerns the occurrence of infeasible subproblems within an outer iteration of a nonlinear optimization algorithm; in particular, infeasible QPs within an SQP method. Our treatment will focus mainly on the second problem, but we later show the same approach is useful in the first setting as well.

We propose a strategy involving some additional variables (which we will call *elastic* variables), and allowing the original constraints to be violated. This is combined with a penalty function method. An important distinction from some earlier work (like the Sl_1QP method by Fletcher [19]) is that the penalty function is used only to obtain a search direction and not as a merit function. The strategy is closely related to the one recently adopted in SQOPT and SNOPT [25]. There the term *elastic bound* is used, because the bounds on the constraints can be viewed as being elastic and no additional variables are introduced. In our presentation we treat the elastic variables as if they were regular variables, but in an implementation it may be possible to represent such variables implicitly in some cases.

In this chapter we focus on the l_1 penalty function, while in the next chapter we consider the l_∞ norm.

2.1 Elastic variables and a composite objective

Consider the nonlinear problem (1.1). We propose handling the infeasibility issue by modifying the original problem so that

- a feasible point always exists;

- if the original problem has a nonempty feasible set we obtain the same solution;
- the modified problem is not harder to solve than the original.

One way to achieve the first goal is to introduce some extra variables into the problem representing the constraint violations (infeasibilities). This leads to

$$\begin{array}{ll} \min\limits_{x,t} & F(x) + \gamma P(t) \\ \text{s.t.} & c(x) + t \geq 0, \quad t \geq 0, \end{array} \tag{2.1}$$

where γ is a positive parameter, t is a vector, and P is some penalty function. The intuition behind this approach is that we allow the constraints to be violated but add a penalty to the objective function for doing so. We can always find a feasible point for the modified problem (2.1) for any given x by adjusting elements of t until they are sufficiently large. We call the t-variables *elastic* variables because they allow the constraints to be "stretched" (violate the bounds). The term *elastic* is borrowed from linear programming where it has a similar meaning[1].

Several issues need to be addressed:

1. Precisely how to introduce the elastic variables? One can think of many variations of the above formulation.
2. What type of penalty term to use? We want the penalty function to be differentiable and ideally also to be exact for a finite γ.
3. How is the solution of the problem affected by this penalty term?
4. How is the computational complexity of the problem affected?

2.1.1 Introducing elastic variables

The best way to introduce elastic variables will depend on the formulation of the optimization problem. One concern is to keep the number of constraints and the number of variables low to avoid unnecessary work. Another concern is that we need to introduce the variables in such a way that we can develop some theory regarding convergence properties.

[1] This use of the word "elastic" is sometimes attributed to G. Brown.

For inequalities, the obvious way to allow constraint violations is

$$c(x) \geq 0 \;\Rightarrow\; c(x) + t \geq 0, \quad t \geq 0, \tag{2.2}$$

as above. This introduces one new t–variable for each constraint. An alternative is

$$c(x) \geq 0 \;\Rightarrow\; c(x) + \tau e \geq 0, \quad \tau \geq 0. \tag{2.3}$$

where τ is a scalar and e as usual is a vector of ones. The latter version only captures the most violated constraint, while the former takes into account all the constraint violations. It is not immediately obvious whether one formulation is better than the other. We study the t version (one-norm) in this chapter, and the τ version (max-norm) in the next chapter.

2.2 Penalty functions

We have added new variables t or τ to the problem. Ideally we wish t or τ to become zero and thereby provide a solution to the original problem. To achieve this we must add a penalty term to the objective. Since penalty functions have been extensively studied in the literature [2, 20], we only briefly review the topic here.

2.2.1 Penalty functions

One way to transform a constrained optimization problem into an unconstrained problem, is to use a penalty function. Suppose we wish to solve the NLP (1.1). Let $c^-(x)$ be defined by $c_i^-(x) = \max(0, -c_i(x))$. We refer to c^- as the constraint violations. The main idea is to solve the unconstrained problem

$$\min_x F(x) + P(c^-(x), \rho) \tag{2.4}$$

instead of the constrained problem. Here $\rho > 0$ is a penalty parameter and $P(y, \rho)$ is a penalty term. The penalty term should satisfy $P(0, \rho) = 0$ and increase monotonically in both y and ρ, so that the more violated the constraints are, the higher the penalty. When ρ is sufficiently large, the unconstrained minimizer of (2.4) should equal the constrained minimizer of (1.1). The exact meaning of "sufficiently large" depends on the penalty function. We describe the three most important penalty functions.

The l_1 penalty function

A simple way to create a penalty function is to take the absolute values (one-norm) of the violated constraints. This gives

$$P_1(x,\rho) = F(x) + \rho\|c^-(x)\|_1 = F(x) + \rho\sum_i |c_i^-(x)|. \tag{2.5}$$

It can be shown that P_1 is *exact* in the sense that the minimizer of (2.5) is the constrained minimizer for a finite value of ρ. However, a major drawback of the absolute value penalty function is that it is not differentiable. The derivatives do not exist when $c_i(x) = 0$.

The l_∞ penalty function

The max-norm (or l_∞) penalty function is

$$P_\infty(x,\rho) = F(x) + \rho\|c^-(x)\|_\infty = F(x) + \rho\max\left\{0, \max_i(-c_i(x))\right\}. \tag{2.6}$$

If there is only one constraint, then $P_\infty(x,\rho) \equiv P_1(x,\rho)$. In the general case with m constraints, the l_∞ penalty function shares many properties with the l_1 penalty function; they are both nondifferentiable, and exact for a finite ρ. We now turn to differentiable penalty functions.

The quadratic penalty function

The quadratic penalty function is

$$P_2(x,\rho) = F(x) + \frac{\rho}{2}\|c^-(x)\|_2^2 = F(x) + \frac{\rho}{2}c^-(x)^T c^-(x). \tag{2.7}$$

This function is differentiable, although only once differentiable (not twice) when $c_i(x) = 0$ for some i. In order to find the constrained minimizer, we may solve (2.7) for an increasing sequence of values for ρ, for example $1, 10, 10^2, 10^3, \ldots$. A drawback is that we are only guaranteed to obtain the constrained minimizer in the limit as $\rho \to \infty$, and the penalty function is usually increasingly hard to minimize..

The augmented Lagrangian

There are several versions of the augmented Lagrangian. For equality constrained problems, we define the augmented Lagrangian to be

$$L_A(x,\lambda,\rho) = F(x) - \lambda^T c(x) + \frac{\rho}{2} c(x)^T c(x), \tag{2.8}$$

where λ is a vector of Lagrange multiplier estimates. This function is differentiable. One can prove that, when $\lambda = \lambda^*$, the augmented Lagrangian is an exact penalty function for a finite ρ. For inequalities several variations exist. Perhaps the most common is

$$L_A(x,\lambda,\rho) = F(x) + \sum_i \begin{cases} -\lambda_i c_i(x) + \frac{\rho}{2} c_i(x)^2 & \text{if } c_i(x) \le \lambda_i/\rho \\ -\frac{\rho}{2}\lambda_i^2 & \text{if } c_i(x) > \lambda_i/\rho \end{cases} \tag{2.9}$$

which was suggested by Rockafellar [65].

A procedure for constrained minimization is

For $k = 0, 1, 2, \ldots$

Compute a local minimizer, $x(\lambda^k)$, to $\min_x L_A(x, \lambda^k, \rho)$.

Update λ^{k+1} such that $\{\lambda^k\} \to \lambda^*$.

Terminate when $c(x(\lambda^k))$ is sufficiently small.

End

Under standard assumptions, one can show that this procedure will converge to the desired minimizer for a finite penalty parameter ρ. In practice, we do not know what ρ should be a priori, so ρ has to be adjusted in the iteration scheme.

2.2.2 Elastic NLP theory

In SQP methods the subproblems are quadratic, so it is natural for us to focus on penalty functions that have only linear and quadratic terms. Mainly we study what we call an *elastic* NLP,

$$\begin{aligned} &\min_{x,t} F(x) + \gamma e^T t \\ &\text{s.t. } c(x) + t \ge 0, \quad t \ge 0. \end{aligned} \tag{2.10}$$

Observe that this is (2.1) with the l_1 penalty function.

Lemma 1 *Assume the NLP (1.1) has a nonempty feasible set. If (x^*, λ^*) is a first-order KKT point of (1.1), then $(x^*, 0, \lambda^*)$ is a first-order KKT point of (2.10) when*

$$\gamma > \max_i \lambda_i^*.$$

Proof: The lemma follows from the optimality conditions of (1.1). The proof is essentially the same as the proof showing that the absolute penalty function is exact. We give a proof here because it provides some helpful insight.

The first-order optimality conditions for (2.10) are

$$\begin{pmatrix} \hat{J}^T & 0 \\ I & I \end{pmatrix} \begin{pmatrix} \lambda^* \\ \mu^* \end{pmatrix} = \begin{pmatrix} g^* \\ \gamma e \end{pmatrix}, \ \lambda^* \geq 0, \ \mu^* \geq 0, \tag{2.11}$$

where $g^* = \nabla F(x^*)$, $\hat{J}$ is the Jacobian of the active constraints at x^*, and we have split the Lagrange multipliers into two parts corresponding to the nonlinear constraints and the bounds on t. Expanding this, we get

$$\hat{J}^T \lambda^* = g^*, \tag{2.12}$$

$$\lambda^* + \mu^* = \gamma e, \ \lambda^* \geq 0, \ \mu^* \geq 0, \tag{2.13}$$

and the multipliers λ^* and μ^* are uniquely determined when $\hat{J}$ has full row rank. Note that the multipliers λ^* are equal to the optimal multipliers of (1.1), since (2.12) are the first-order optimality conditions for the original NLP. From $\mu^* = \gamma e - \lambda^*$, it follows that $\mu^* \geq 0$ when $\gamma \geq \lambda^*_{\max}$. If all μ^* are positive, all the bounds on t must be active; hence $t^* = 0$. This completes the proof. $\square$

Corollary 2 *Assume the NLP (1.1) is convex and has a nonempty feasible set. If x^* is a minimizer of (1.1), then $(x^*, 0)$ is a minimizer of (2.10) when*

$$\gamma \geq \max_i \lambda_i^*,$$

where λ^* are the Lagrange multipliers for (1.1) at x^*.

Proof: Follows directly from Lemma 1 and the fact that for a convex function, any KKT point is a minimizer. □

This corollary applies to SQP methods with positive definite Hessian approximations.

The required value of γ is not so important because the optimal multipliers are not known *a priori*. Most important is that γ is finite provided the Lagrange multipliers are bounded (a standard assumption). It is straightforward to generalize Lemma 1 to NLPs with equality constraints.

The reverse of Lemma 1 or Corollary 2 does not hold, since $F(x) + \gamma e^T t$ may have KKT points with $t \neq 0$ that are not KKT points of $F(x)$. This is one of the concerns addressed in this thesis.

We give two examples to illustrate the results in this section.

Example 2.2.1

Consider the QP

$$\min x_1^2 + x_2^2 \tag{2.14}$$

$$\text{s.t. } x_1 + x_2 - 1 \geq 0. \tag{2.15}$$

The minimizer is $x^* = (.5, .5)$ with multiplier $\lambda^* = 1$. The elastic QP is

$$\min x_1^2 + x_2^2 + \gamma t \tag{2.16}$$

$$\text{s.t. } x_1 + x_2 + t - 1 \geq 0 \tag{2.17}$$

$$t \geq 0. \tag{2.18}$$

The first-order KKT conditions for this problem are

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \pi^* \\ \mu^* \end{pmatrix} = \begin{pmatrix} 2x_1 \\ 2x_2 \\ \gamma \end{pmatrix}, \quad \pi^*, \mu^* \geq 0, \tag{2.19}$$

which gives the solution

$$\gamma < 1: \quad x^* = (\gamma/2, \gamma/2),\ t^* = 1 - \gamma,\ \pi^* = \gamma,\ \mu^* = 0, \tag{2.20}$$

$$\gamma \geq 1: \quad x^* = (.5, .5),\ t^* = 0,\ \pi^* = 1,\ \mu^* = \gamma - 1. \tag{2.21}$$

Example 2.2.2
Consider the NLP

$$\min 0 \tag{2.22}$$
$$\text{s.t. } x_1^2 + x_2^2 \geq 4, \tag{2.23}$$
$$2x_1x_2 \leq 1. \tag{2.24}$$

This is a pure feasibility problem because the objective is constant. The feasible regions are shown in Figure 2.2.2. The elastic NLP is then

$$\min \gamma(t_1 + t_2) \tag{2.25}$$
$$\text{s.t. } x_1^2 + x_2^2 + t_1 - 4 \geq 0 \tag{2.26}$$
$$-2x_1x_2 + t_2 + 1 \geq 0 \tag{2.27}$$
$$t_1 \geq 0 \tag{2.28}$$
$$t_2 \geq 0. \tag{2.29}$$

We examine the case where the two nonlinear constraints are active but the bounds on t are not. The first-order KKT conditions are

$$\begin{pmatrix} 2x_1 & -2x_2 \\ 2x_2 & -2x_1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \pi^* = \begin{pmatrix} 0 \\ 0 \\ \gamma \\ \gamma \end{pmatrix}, \quad \pi^* \geq 0 \tag{2.30}$$

From this we get $\pi^* = (\gamma, \gamma)$ and $x_1 = x_2$. Since we assumed that the two nonlinear constraints were active, the possible range for x^* is $x^* = (\alpha, \alpha)$, $\sqrt{2}/2 \leq \alpha \leq \sqrt{2}$. Consider for example $x = (1, 1)$. Although $x = (1, 1), t = (2, 1)$ is a (first-order) KKT point for the elastic program, $x = (1, 1)$ is not a KKT point for the original program. Note that the value of γ plays no role in this example.

2.2.3 Nonuniform penalty weights

For a particular application, the user may know that some constraints are more important than others. Ideally an algorithm should take such information into account. This can

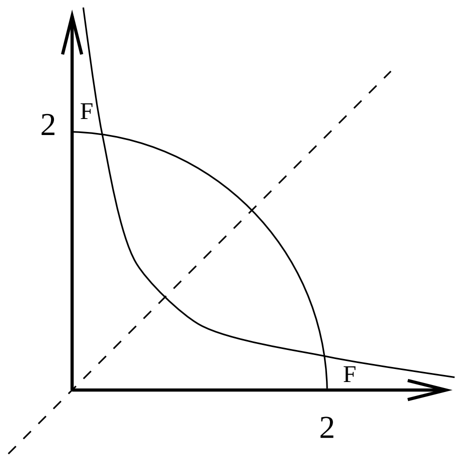

Figure 2.1: The feasible regions are marked with 'F'

easily be done by replacing the vector e with a nonuniform vector w ($w \geq 0$):

$$\min_{x,t} \; F(x) + \gamma w^T t \tag{2.31}$$

$$\text{s.t. } c(x) + t \geq 0 \tag{2.32}$$

The more important a constraint is, the greater weight w_i it should have. We assume the constraints are scaled such that they are of the same order of magnitude, otherwise this has to be taken into account when deciding the weights.

With weights, Lemma 1 has to be modified, but only the value of the bound on γ changes.

2.3 An elastic SQP method

To simplify the presentation, we will, unless stated otherwise, assume only inequality constraints are present. We show later that equality constraints (and constraints with both lower and upper bounds) can be treated in a similar manner. Thus we assume the QP subproblems generated by SQP methods for the problem NLP are of the form

$$\begin{array}{ll} \min\limits_{p} & g^T p + \frac{1}{2} p^T H p \\ \text{s.t.} & Jp \geq -c, \end{array} \tag{2.33}$$

where as usual $g = \nabla F(x_k)$, $J = \nabla c(x_k)$, and H is an approximation to the Hessian of the Lagrangian, $L(x, \lambda) = F(x) - \lambda^T c(x)$.

A standard assumption in the analysis of SQP methods is that a feasible point for the QP always exists. Unfortunately, in practice the QP subproblem may be infeasible even if the original problem is not. There have been a number of suggestions on how to remedy this in an algorithm. The SQP code NPSOL works around the problem by having a *phase I* minimize the (possibly weighted) sum of infeasibilities. This is not quite satisfactory because we are not guaranteed to obtain a descent direction for the merit function of the NLP when the subproblem is infeasible. A different approach is to use a composite objective in NLP that incorporates both the original objective function and the constraint violations (e.g. a penalty function). We have to be careful to do this in a manner such that the algorithm is not slowed down when the QP subproblem is actually feasible.

Consider the l_1 penalty function for NLP:

$$P(x, \gamma) = F(x) + \gamma \|c^-(x)\|_1 \tag{2.34}$$

where $c_i^- = \max(0, -c_i)$. By introducing elastic variables t, we get

$$\begin{array}{ll} \displaystyle\min_{x,t} & F(x) + \gamma e^T t \\ \text{s.t.} & c(x) + t \geq 0 \\ & t \geq 0. \end{array} \tag{2.35}$$

A feasible point x of (2.35) that has $t = 0$ is clearly also a feasible point for the original problem. The parameter γ weights the emphasis of becoming feasible versus improving the objective value. As we increase γ, we put more weight on finding a feasible point. We know from Lemma 1 that if there exists a feasible point for NLP (1.1) then (2.35) will find it when γ is sufficiently large (but finite). If $\gamma \to \infty$ and there is still no solution, then the original problem NLP has no feasible point (or the KKT conditions do not hold at the solution).

The corresponding elastic QP subproblem is:

$$\begin{array}{ll} \displaystyle\min_{p,q} & g^T p + \frac{1}{2} p^T H p + \gamma e^T q \\ \text{s.t.} & Jp + q \geq -(c + t) \\ & q \geq -t. \end{array} \tag{2.36}$$

The main difficulty in an SQP algorithm based on the elastic QP (2.36) is to choose good values for γ. Ideally, we would like to choose γ larger than $\lambda^*_{\max}$ but not too large. In practice, this is impossible since we do not know λ^*. Therefore, we have to start with an initial guess $\gamma = \gamma_0$ and dynamically adjust γ. If we solve (2.36) and obtain $t > 0$ we may wish to increase γ. We need to have some cut-off limit $\gamma_{\max}$ where we "give up" when $\gamma \geq \gamma_{\max}$. In these cases we will not know whether the NLP is infeasible or it has a large multiplier at the solution. In our method γ remains constant inside a QP subproblem, but might change between subproblems.

We know that $\gamma = \lambda^*_{\max}$ is the smallest γ for which a solution to the elastic problem may be a solution to the original problem. However, if we "overestimate" γ such that $\gamma > \lambda^*_{max}$ the solution x^* does not change. Thus one could be tempted to set γ to some extremely large value. The effect is that the term from the objective becomes negligible while $t \neq 0$, so essentially the result is the standard "phase I" approach of minimizing the sum of infeasibilities.

2.3.1 Switching to elastic mode

It is not necessary to use the elastic formulation all the time. A standard SQP method (no elastic variables) can be employed until it is deemed necesssary or appropriate to switch to elastic mode. We suggest several circumstances under which such a switch should take place. The first is obviously when a QP subproblem is determined to be infeasible. Another is when the multipliers become large, and a third is when the Jacobian of the active constraints becomes ill-conditioned.

2.3.2 Multiplier "feedback"

From the discussion in Section 2.3 we realize that an algorithm may have to increase the value of γ until it is sufficiently large. Lemma 1 shows that the value of γ required grows with the largest multiplier. The optimal multipliers of the elastic problem are equivalent to the optimal multipliers of the original problem when γ is large enough that $t = 0$. Let us examine what happens to the multipliers when γ is smaller and $t > 0$. When the bounds on t are not active, the optimality conditions are

$$\begin{pmatrix} \hat{J}^T \\ I \end{pmatrix} \lambda^* = \begin{pmatrix} g^* \\ \gamma e \end{pmatrix}. \tag{2.37}$$

It follows that $\lambda_i^* = \gamma$ for all i. When we increase γ, the multipliers (for the modified problem) increase correspondingly, which again forces us to increase γ, and so on. This can be compared to a feedback loop. The multipliers and γ drive each other up.

Fortunately there is a simple way around this problem. Instead of scaling the penalty term by γ and leaving the original objective intact, we scale both terms. Let ω be a scalar between 0 and 1. Our new composite objective is

$$\omega F(x) + (1-\omega)e^T t. \tag{2.38}$$

Dividing through by ω we see that this corresponds to $\gamma = \frac{1-\omega}{\omega}$, or equivalently, $\omega = \frac{1}{1+\gamma}$. The main advantage of this formulation is that the Lagrange multipliers are now $O(1)$, not $O(\gamma)$. A second benefit is that while the old objective goes towards ∞ as $\gamma \to \infty$, the new objective goes towards $e^T t$ as $\omega \to 0$.

Therefore we will use the objective (2.38) in our algorithms, but from a theoretical point of view it is simpler to analyze the γ formulation.

2.4 Bounding the elastic variables

In SQP methods, the iterates x_k do not in general satisfy the nonlinear constraints. A standard assumption needed to prove convergence is therefore that the objective function is bounded below on an *extension* of the feasible set. Such an extension can be written

$$\|c^-(x)\|_1 \le \beta, \tag{2.39}$$

where β is a positive scalar. A different norm will produce a different region. The extension of the feasible set in the one-norm can alternatively be written

$$e^T c^-(x) \le \beta. \tag{2.40}$$

In elastic mode the constraints $c(x) \ge 0$ are changed to

$$c(x) + t \ge 0, \quad t \ge 0. \tag{2.41}$$

We want to examine how conditions (2.39) and (2.41) change when elastic variables are employed. Condition (2.39) is then

$$\| (c(x) + t)^{-} \| \leq \beta. \tag{2.42}$$

A crucial observation here is that nothing prevents $c(x) \to -\infty$ and $t \to \infty$! The penalty term in the objective function is intended to keep t small, but once we go outside the region where (2.39) holds, the objective $F(x)$ may go to $-\infty$ while x converges to an infeasible point. This observation also holds in other norms.

It follows from this argument that we cannot prove convergence for the original problem if we simply add the elastic variables to the problem and solve as usual. The problem formulation has to be modified such that the t variables are bounded in some way. Just putting simple bounds on t,

$$t \leq u,$$

is not satisfactory because there then is no guarantee a feasible point exists for the elastic QP. The existence of a feasible point is a key property of the elastic subproblem that should be preserved. We suggest bounding t by enforcing

$$\|t\|_1 \leq \beta, \tag{2.43}$$

or equivalently,

$$e^T t \leq \beta. \tag{2.44}$$

When elastic variables are introduced, such a β can be derived from the current $c(x)$. This β is not the same β that defined the extension of the feasible region, but we bypass this difficulty by assuming the derived β is smaller so all our assumptions hold. Let q denote the search direction in the t direction. Constraint (2.43) implies

$$e^T q \leq \beta - e^T t. \tag{2.45}$$

Condition (2.43) and $t \geq 0$ imply that the right-hand side of (2.45) is nonnegative. Without the constraint (2.45), the QP subproblem always has a feasible point. (Pick any p and make q large enough.) Adding (2.45) limits the choice of q so it is no longer obvious that a QP feasible point exists. However, in the next paragraph we show that the new QP is indeed feasible.

If we only wish to solve a single QP, adding bounds of the type $q \leq \tilde{u}$ would be a valid option. But we want to solve a *sequence* of QPs. In that case, we might have to let $u \to \infty$ in order to ensure feasible points exist. An advantage of our approach is that β is constant for the whole sequence of QPs.

2.4.1 The modified QP is feasible

We prove that there always exists a feasible point for the elastic QP, even after we add the extra constraint on t.

Lemma 3 *Assume c, J, t and β are given, where $t \geq 0$, $e^T t \leq \beta$ and $e^T c^- \leq \beta$. Then there exists a feasible point for the system of linear inequalities*

$$Jp + q \geq -(c + t), \tag{2.46}$$

$$q \geq -t, \tag{2.47}$$

$$e^T q \leq \beta - e^T t. \tag{2.48}$$

Furthermore, when $\hat{J}^T e \neq 0$ (where $\hat{J}$ corresponds to the rows of J for which $c \leq 0$) there is a feasible point with $e^T q < \beta - e^T t$.

Proof: Let $\hat{c}$ denote the constraints that are violated or active at the current point x (that is, $c_i(x) \leq 0$), and let $\bar{c}$ denote the constraints that are strictly satisfied at the current point. We apply the same hat and bar notation to J, t and q. We shall assume $\bar{t} = 0$, because if $\bar{t} > 0$ the merit function can be reduced by setting $\bar{t} = 0$ without violating any constraints.

We have seen that without (2.48) a feasible point exists for any p (just make q sufficiently large). We now prove that the system is feasible even with the additional constraint (2.48).

Using the notation defined above, consider the LP

$$\min_{p,q}\ e^T q = e^T \hat{q} + e^T \bar{q} \tag{2.49}$$

$$\text{s.t.}\ \ \hat{J}p + \hat{q} \geq -(\hat{c} + \hat{t}),\ \hat{c} \leq 0, \tag{2.50}$$

$$\bar{J}p + \bar{q} \geq -\bar{c},\ \bar{c} > 0, \tag{2.51}$$

$$\hat{q} \geq -\hat{t}, \tag{2.52}$$

$$\bar{q} \geq 0. \tag{2.53}$$

A feasible point exists for this LP because the constraints are equivalent to (2.46–2.47). Let (p^*, q^*) be a minimizer of the LP (2.49–2.53). If we can show that $e^T q^* \leq \beta - e^T t$, then (p^*, q^*) is also feasible with respect to (2.46–2.48). Recall that β is an upper bound on the constraint violations, so that $-e^T\hat{c} \leq \beta$ and $e^T t \leq \beta$.

Define p_0 and q_0 by

$$p_0 = 0,\ \bar{q}_0 = 0,\ \hat{q}_0 = -(\hat{c} + \hat{t}). \tag{2.54}$$

It is straightforward to verify that (p_0, q_0) is an initial feasible point (p_0, q_0) of (2.49–2.53). The objective value is $e^T q_0 = -e^T(\hat{c} + \hat{t})$. Observe that

$$e^T q_0 = -e^T(\hat{c} + \hat{t}) \leq \beta - e^T t. \tag{2.55}$$

Hence (p_0, q_0) satisfies (2.46–2.48) and we can already conclude a feasible point exists for the modified elastic QP. Naturally, $p = p_0$ is not quite satisfactory because $p_0 = 0$ and x would remain unchanged after the update. We would like to obtain a $p^* \neq 0$ and show that

$$e^T q^* < e^T q_0, \tag{2.56}$$

i.e., the sum of the constraint violations is strictly decreasing. One way to accomplish this is to start with p_0 and find an update in the nullspace of the active set that decreases the objective. Let A denote the full Jacobian of the constraints in (2.49–2.53), and let $\tilde{A}$ be the

Jacobian of the active set. We have that

$$A = \begin{pmatrix} \hat{J} & I & 0 \\ \bar{J} & 0 & I \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix}. \tag{2.57}$$

From the way we split the constraints, we know that the top (block) row is active, while the second (block) row is not. The constraints in the third (block) row correspond to the bounds on $\hat{q}$, which cannot be active because $\hat{q} > 0$ by definition. The constraints in the last (block) row may be either active or non-active. Consequently, the active-set Jacobian can be written

$$\tilde{A} = \begin{pmatrix} \hat{J} & I & 0 \\ 0 & 0 & I' \end{pmatrix}, \tag{2.58}$$

where the prime indicates that a subset of the rows has been selected.

Let

$$Z = \begin{pmatrix} I \\ -\hat{J} \\ 0 \end{pmatrix}. \tag{2.59}$$

Then Z has full rank and $\tilde{A}Z = 0$, so Z is a basis for the nullspace of $\tilde{A}$. Furthermore, let g denote the gradient of $e^T q$ with respect to $p, \bar{q}, \hat{q}$, i.e. $g = (0, e, e)$. The steepest descent direction projected onto the nullspace is given by

$$v = -ZZ^T g. \tag{2.60}$$

The descent along v is

$$g^T v = -g^T Z Z^T g \tag{2.61}$$

$$= -e^T \hat{J} \hat{J}^T e \tag{2.62}$$

$$\leq 0. \tag{2.63}$$

From this it follows there is strict descent, except when $Z^T g = 0$, or equivalently,

$$\hat{J}^T e = 0. \tag{2.64}$$

Consider the function $f(x) = e^T \hat{c}(x)$, that is, the sum of the violated constraints. A point that satisfies $\hat{J}^T e = 0$ is a first-order optimality point for f. At such points, one cannot expect to reduce the objective unless second-order information is used to compute a direction of negative curvature. When $\hat{J}^T e \neq 0$, then $v^T g < 0$ so v is a feasible direction of (strict) descent, and it follows that

$$e^T q^* < \beta - e^T t. \tag{2.65}$$

□

2.4.2 Warm start and initial feasible point

We showed that a feasible point of the QP exists, but the procedure described may not be suitable in an algorithm. In a practical algorithm one would want to find not only a feasible point but preferably a "good" one. Also we would like to find such a point quickly. In an active-set method, we wish to exploit the information from the previous QP in the next QP. This is done by retaining the working set from the previous QP—a so-called *warm start*. The use of warm starts can dramatically reduce the number of QP iterations, and is necessary for convergence when QP subproblems are nonconvex.

In the elastic SQP method, warm starts can be performed as in a standard SQP method, except we operate on the extended problem with the t-variables. The only special case is the first time we switch to elastic mode, perhaps because the regular (inelastic) QP is infeasible. We can determine a search direction p from the constraints in the working set. Fortunately, it is easy to determine a q such that all constraints but (2.48) are satisfied:

$$q = \max\left(-(Jp + c + t), -t\right). \tag{2.66}$$

2.5 Updating the parameters γ and t

2.5.1 Updating the penalty parameter γ

With the modification introduced in Section 2.4, the infeasibilities are kept small by two mechanisms: by the penalty parameter γ, and by the constraint $e^Tt \le \beta$. The latter ensures the elastic variables are bounded, while a sufficiently large γ will make them go to zero (if a feasible point exists). So far we have not discussed how to adjust γ in a practical algorithm. We now show how the Lagrange multiplier from the nontrivial constraint on t can be used to determine the update for γ.

Consider the elastic nonlinear problem

$$\min_x \; F(x) + \gamma e^T t \tag{2.67}$$

$$\text{s.t.} \;\; c(x) + t \ge 0 \tag{2.68}$$

$$t \ge 0 \tag{2.69}$$

$$e^T t \le \beta. \tag{2.70}$$

Let λ, μ and ν be the Lagrange multipliers for the three groups of constraints, respectively. The first-order optimality conditions are

$$\begin{pmatrix} J^T & 0 & 0 \\ I & I & -e \end{pmatrix} \begin{pmatrix} \lambda^* \\ \mu^* \\ \nu^* \end{pmatrix} = \begin{pmatrix} g^* \\ \gamma e \end{pmatrix}, \tag{2.71}$$

$$\lambda^*, \mu^*, \nu^* \ge 0, \tag{2.72}$$

which gives

$$J^T \lambda^* = g^*, \tag{2.73}$$

$$\lambda^* + \mu^* = (\gamma + \nu^*) e. \tag{2.74}$$

If $\nu^* > 0$, the constraint (2.70) is active, so we have to increase γ. Based on (2.74), a natural update would be

$$\gamma_{new} = \gamma + \nu^*. \tag{2.75}$$

This strategy tries to produce an x such that $e^T t \leq \beta$. However, it does not attempt to reduce the elastic variables further. Recall that if a feasible point exists for the original problem, we wish to find a point with $t = 0$ for the elastic problem. This indicates we need to increase γ by more than (2.75), so we propose

$$\gamma_{new} = 2\left(\gamma + \nu^*\right), \tag{2.76}$$

where the constant two is somewhat arbitrary (must be at least one).

We impose an upper limit on γ, say γ_{max}. If $\gamma > \gamma_{max}$ the problem is quite likely infeasible and we stop increasing γ. Because the value of γ is at least doubled every time we increase γ, the cut-off γ_{max} is reached in $O(\log(\gamma_{max}))$ steps. Thus only a constant number of adjustments of γ will be made.

Note that in an SQP method the nonlinear multipliers are not known, but the QP multipliers can be used as estimates. QP multipliers are always available in our elastic methods because the elastic QPs are feasible. In a standard SQP method a QP may be infeasible and then no multipliers are available.

2.5.2 More on γ and the constraint on t

We have seen that the constraint on $e^T t$ plays a role similar to controlling the penalty parameter γ. The magnitude of t is implicitly bounded by γ. A larger γ will produce a smaller t in some norm. Thus it is not necessary to introduce an explicit constraint on t such as (2.44) in an elastic SQP method, but doing so has several advantages. It is highly unlikely that the user would have any idea about how to choose a value for γ, while a reasonable bound β on $e^T t$ is much more likely to be known. Recall that we assumed the objective and constraint functions are well-defined within an extension of the feasible region where $e^T c^-(x) \leq \beta$. This β is not necessarily small; in fact it could be infinite.

Lemma 4 *A minimizer (p^*, q^*) of the elastic QP*

$$\min_{p,q}\ g^T p + \frac{1}{2} p^T H p + \gamma e^T q \tag{2.77}$$

$$s.t.\ \ Jp + q \geq -(c + t) \tag{2.78}$$

$$q \geq -t \tag{2.79}$$

satisfies

$$e^T q^* \leq 0 \tag{2.80}$$

when γ is sufficiently large.

Proof: We have shown that (2.77) has a feasible point, even when the constraint

$$e^T q \leq \beta - e^T t \tag{2.81}$$

is added. When $\beta = e^T t$, this reduces to (2.80).

Consider the case when γ is large. The minimum of (2.77) will go towards ∞ as $\gamma \to \infty$ if $e^T q > 0$. Clearly the minimum of (2.77) is less than or equal to the minimum of the same QP with the additional constraint (2.80). Because $e^T q^* \leq 0$, it follows that the minimum of (2.77) does not increase with γ. Consequently, $e^T q \leq 0$ for γ sufficiently large. $\square$

2.5.3 Adjustment of s and t based on the merit function

Earlier we defined the augmented Lagrangian merit function as

$$M(x, \lambda) = F(x) - \lambda^T c(x) + \frac{1}{2}\rho \|c(x)\|_2^2, \tag{2.82}$$

where $\rho \geq 0$ is a penalty parameter. This formulation is only valid for equality constraints ($c(x) = 0$). One way to generalize it to inequalities ($c(x) \geq 0$), is to introduce a set of *slack* variables s. These variables can be represented implicitly in a code, and hence do not require extra storage. A merit function is then

$$M(x, s, \lambda) = F(x) - \lambda^T (c(x) - s) + \frac{1}{2}(c(x) - s)^T D_\rho (c(x) - s), \tag{2.83}$$

where D_ρ is a diagonal matrix of penalty parameters ρ_i. This merit function was analyzed in [26]. In an SQP algorithm such as implemented in NPSOL and SNOPT, search directions for x, s, and λ are computed in each QP subproblem and then a "better" value of s is chosen between QP subproblems. Consider x and λ to be fixed. Since the slack variables all occur separately in the merit function, it is possible to choose each s_i such that it minimizes

$M(x,s,\lambda)$ subject to $s_i \geq 0$. The result is

$$s_i = \begin{cases} \max(0, c_i(x)) & \text{if } \rho_i = 0, \\ \max(0, c_i(x) - \lambda_i/\rho_i) & \text{if } \rho_i > 0. \end{cases} \tag{2.84}$$

The same principle applies in the elastic SQP method. We can adjust s and t in order to minimize the merit function. For the elastic NLP (2.10), we have

$$\tilde{F}(x,t) = F(x) + \gamma e^T t, \tag{2.85}$$

$$\tilde{c}(x,t) = c(x) + t, \tag{2.86}$$

$$\tilde{M}(x,s,t,\lambda) = F(x) + \gamma e^T t - \lambda^T(c(x) - s + t) + \frac{1}{2}(c(x) - s + t)^T D_\rho (c(x) - s + t). \tag{2.87}$$

A convenient way to minimize $\tilde{M}(x,s,t,\lambda)$ with respect to s and t is to minimize for each pair (s_i,t_i) separately. This is a valid strategy because there are no cross-terms. Consider therefore the two-dimensional problem

$$\min_{s_i,t_i} \ \phi(s_i,t_i) \equiv \gamma t_i - \lambda_i(c_i - s_i + t_i) + \frac{1}{2}(c_i - s_i + t_i)^2 \tag{2.88}$$

$$\text{s.t. } \ s_i, t_i \geq 0. \tag{2.89}$$

There are four possible cases because each of s_i and t_i may be zero or nonzero. We now show that $s_i > 0$ and $t_i > 0$ is never optimal, hence $s_i = 0$ or $t_i = 0$.

Suppose $s_i > 0$ and $t_i > 0$ at the minimum of $\phi(s_i,t_i)$. By taking partial derivatives of $\phi(s_i,t_i)$ we obtain the optimality conditions

$$\frac{\partial \phi}{\partial s_i} = \lambda_i - \rho_i(c_i - s_i + t_i) = 0, \tag{2.90}$$

$$\frac{\partial \phi}{\partial t_i} = \gamma - \lambda_i + \rho_i(c_i - s_i + t_i) = 0. \tag{2.91}$$

These two equations are only consistent if $\gamma = 0$. Since $\gamma > 0$, we conclude there is no unconstrained minimizer of $\phi(s_i,t_i)$.

When $t_i = 0$ the problem reduces to the standard (nonelastic) case and the optimal s_i is given by (2.84). Consider next the case where $s_i = 0$. Then condition (2.91) reduces to

$$\gamma - \lambda_i + \rho_i(c_i + t_i) = 0, \tag{2.92}$$

which gives

$$t_i = -\left(c_i(x) + \frac{\gamma - \lambda_i}{\rho_i}\right) \tag{2.93}$$

when $\rho_i > 0$. We observe that when $\gamma > \lambda_i$, then $t_i < -c_i$. When $\rho_i = 0$, we can simply set $t_i = -c_i(x)$. We need to enforce $t_i \geq 0$, so in general the optimal t_i is given by

$$t_i = \begin{cases} \max(0, -c_i(x)) & \text{if } \rho_i = 0, \\ \max(0, -c_i(x) - (\gamma - \lambda_i)/\rho_i) & \text{if } \rho_i > 0. \end{cases} \tag{2.94}$$

It is easy to tell whether $s_i = 0$ or $t_i = 0$ by examining the value of $c(x_i)$. When $c_i(x) \geq 0$ then $t_i = 0$ and s_i is given by (2.84). Otherwise, when $c_i(x) < 0$ then $s_i = 0$ and t_i is given by (2.94).

2.6 Relation to the SNOPT approach

We have mentioned that the elastic method we suggest is closely related to the method implemented in SNOPT. We will elaborate on this shortly, but first we describe how equalities can be handled.

2.6.1 Equality constraints

So far we have assumed that all the constraints are inequalities. For equalities, there are two main strategies for modifying the constraints to allow for elasticity. The obvious elastic version of $c(x) = 0$ is

$$c(x) + t = 0, \tag{2.95}$$

where t may have both positive and negative elements. A drawback of this formulation is that a sign change in t causes discontinuous derivatives in the absolute value (l_1) penalty function. We therefore insist on the elastic variables being nonnegative. Fortunately, the difficulty can be avoided by using two variables instead of one:

$$c(x) + t - v = 0, \quad t, v \geq 0. \tag{2.96}$$

The number of elastic variables has now doubled, but we note that there always exists a solution where complementarity in t and v holds, i.e. $t_i v_i = 0$ for all i. This property can be exploited in an implementation: only one vector (the length of t) need be stored.

Equality constraints are a special case of inequality constraints with both lower and upper bounds. Using the same approach as for equalities, the constraints $l \leq c(x) \leq u$ can be transformed to

$$l \leq c(x) + t - v \leq u, \quad t, v \geq 0. \tag{2.97}$$

Another option is to transform all the general constraints (equalities and inequalities) into equalities by adding new slack variables. The new slack variables will have simple bounds, and we can add elastic variables on these. More formally, $l \leq c(x) \leq u$ will be transformed to

$$c(x) - s = 0, \quad l \leq s \leq u. \tag{2.98}$$

This is not an elastic formulation, but just a restatement of the original problem. The formulation can be made elastic by allowing the bounds on s to be violated. This type of strategy has been implemented in SNOPT and SQOPT.

2.6.2 SNOPT

SNOPT [25] is an SQP code developed by Gill, Murray, and Saunders for solving large and sparse nonlinear optimization problems. Of particular interest to us is that SNOPT has incorporated an elastic formulation similar to the ones we have studied. More specifically, the l_1-elastic formulation is employed but all the elastic variables are handled implicitly. SNOPT treats linear and nonlinear constraints differently. First, a linear elastic program is solved to compute a point that satisfies the linear constraints. If this linear problem is infeasible, then SNOPT stops and never evaluates the nonlinear constraints. Otherwise, all the following iterates stay feasible with respect to the linear constraints and the simple bounds. An l_1-elastic problem is solved where only the nonlinear constraints are elastic.

2.6.3 SQOPT

SQOPT is the QP solver employed by SNOPT. This is a sparse, active-set QP algorithm that switches to l_1-elastic mode when necessary. In fact, SQOPT can treat any subset of the constraints as elastic, but SNOPT uses these features only as described above.

2.6.4 A quick comparison to our approach

We propose to allow all types of constraints to be elastic, that is, simple bounds and linear constraints as well as nonlinear constraints. This is slightly more general than what is done in SNOPT, though one could achieve much the same effect with SNOPT by simply specifying the linear constraints as nonlinear.

SNOPT only supports the l_1 norm for constraint violations, while we propose to use the l_∞ norm in the next chapter. We will show that the l_∞ norm has some advantages in the case of simple bounds.

Another difference is that SNOPT puts no explicit upper bound on the magnitude of the constraint violations, $e^T t$, though the code attempts to keep this quantity small in the line search.

2.7 A model elastic SQP algorithm

We summarize our findings so far by describing a simple elastic SQP algorithm.

Algorithm 2.7: *Simple elastic SQP algorithm.*

Initialize variables.
repeat
  Find an initial feasible point for the elastic QP (2.77).
  Solve the elastic QP to obtain search directions p, q and multipliers π.
  Determine α by a line search using the augmented Lagrangian
    merit function applied to $F(x) + \gamma e^T t$.
  Set $x = x + \alpha p$.
  Set $t = t + \alpha q$.
  Set $\lambda = \lambda + \alpha(\pi - \lambda)$.
  Update t.
  Update penalty parameters γ and ρ, if necessary.
  if $\gamma > \gamma_{\max}$ **then** the NLP is likely infeasible, *exit*
  Perform quasi-Newton update of the approximate Hessian H.
until $t = 0$ and x satisfies the optimality conditions.

2.8 Numerical examples and results

We have not yet implemented the elastic SQP algorithm in full as described. However, one can demonstrate the behavior of such an algorithm by changing the problem to contain explicit elastic variables. We have written a program in MATLAB that takes an optimization problem as input and transforms the problem to include elastic variables. The elastic problem is then solved by calling NPSOL (or E04UCF, the corresponding NAG routine).

First we look at an infeasible problem. Consider the following version of the example with two circles from Section 1.1.3:

$$\min_x \|x - (0,1)\|^2 \tag{2.99}$$

$$\text{s.t. } \|x - (2,0)\|^2 \leq 1 \tag{2.100}$$

$$\|x - (-2,0)\|^2 \leq 1. \tag{2.101}$$

We ran NPSOL 5.0 with explicit elastic variables and a fixed value of γ. The results for the original problem and the elastic problem with various values of γ are shown in Table 2.1.

Since there is no feasible point, there is no solution or "correct answer" to this problem.

γ	x^*	# iter.	# func. eval.
original	(.447, 0)	14	43
10^{-3}	(0, .998)	3	6
1	(0, .333)	6	9
10^3	(0, .5E-3)	8	13
10^6	(0, .5E-6)	8	11

Table 2.1: Results for the l_1-elastic algorithm on an infeasible problem.

γ	original	10^{10}	$100\lambda^*_{\max}$	$10\lambda^*_{\max}$	$1.01\lambda^*_{\max}$
HS27	20/28	37/84	22/27	18/27	27/40
HS93	11/14	10/13	11/14	11/14	12/16
HS104	21/26	15/17	17/19	17/19	22/29
HS108	$1/2^\dagger$	18/32	16/29	13/17	16/20
HS116	14/15	$5/6^*$	11/13	19/24	22/28

Table 2.2: Number of iterations and function evaluations for the l_1-elastic algorithm on some standard test problems.

If we ignore the constraints, the unconstrained minimizer is $(0, 1)$. Any point along the line between the points $(-1, 0)$ and $(1, 0)$ minimizes the l_1-norm of infeasibilities, while $(0, 0)$ is a minimizer of the l_∞-norm of the infeasibilities. Taking all this into account, one can argue that $(0, 0)$ is the best point.

We observe from the empirical results in Table 2.1 that a small value of γ leads to a point near the unconstrained minimizer, while the elastic solution approaches $(0, 0)$ with increasing γ. This behavior is what we desired for infeasible problems.

We now turn to feasible optimization problems. Two concerns when we add elastic variables are that the transformed problem is more expensive to solve per iteration, and the number of iterations increase. However, from the discussion in this chapter we have seen that the additional work per iteration is small. We have not developed any theory concerning the number of (major) iterations, so this must be studied empirically.

We show the results for some test problems taken from Hock and Schittkowski [34] in Table 2.2. Note that we did not add elastic variables to simple bounds. Equalities were not treated as a special case, but essentially as two inequalities.

The numbers in the table are the number of iterations and function evaluations. The dagger (†) indicates that no solution was found, while an asterix (*) means that the code converged to a point that satisfies the optimality conditions of the elastic problem but not those of the original problem. (In theory this should not happen, but it may happen in practice owing to scaling effects.)

Test problems HS27 and HS108 are particularly interesting. We see that if γ is very large (10^{10}), the number of iterations and function evaluations increase significantly. But when $\gamma = 10\lambda^*_{\max}$, the number of iterations (and function evaluations) is actually smaller than for the original problem. This indicates that the elastic approach may accelerate convergence even when there are no infeasible QP subproblems.

The number of iterations do not vary as much for the other problems. Since $\lambda^*_{\max}$ is the smallest value of γ that gives a solution with $t = 0$, it is natural that the number of iterations should increase a little when γ is close to this value. This strengthens our belief that it is advisable to overestimate γ.

These numerical results should be interpreted as examples of what may happen. More extensive testing on a proper implementation of the elastic algorithm is left as future work.

Chapter 3

The l_∞-elastic approach

3.1 Introduction

In the previous chapter we studied elastic SQP methods using a vector of elastic variables and the one-norm in the penalty term. There are many alternatives. Of particular interest is the max-norm, since then only one elastic variable is required. This approach is therefore potentially more efficient in terms of the work per iteration, though the difference is negligible when the elastic variables are handled implicitly. The l_∞ norm is worth studying because this norm may be more relevant in some applications. For example, the maximum violation of any single constraint may be an important quantity to the user, while the sum of all the constraint violations gives little information.

Replacing t by τe in the elastic formulation, where τ is scalar, we obtain

$$\begin{aligned} \min_{x,\tau} \quad & F(x) + \gamma\tau \\ \text{s.t.} \quad & c(x) + \tau e \geq 0, \\ & \tau \geq 0. \end{aligned} \tag{3.1}$$

We call this the τ version or the max-norm version. Note that we may wish to use a nonuniform vector of penalty weights in place of e; see the following section. For simplicity we only analyze the case with uniform weights (e) in this chapter, though we believe it is straightforward to generalize the formulae to the nonuniform case.

The QP corresponding to (3.1) is

$$\begin{aligned} \min_{p,\theta}\ & \frac{1}{2}p^T H p + g^T p + \gamma\,\theta \\ \text{s.t.}\ & Jp + \theta e \geq -(c + \tau e), \\ & \theta \geq -\tau, \end{aligned} \tag{3.2}$$

where θ is the update in the τ dimension.

3.1.1 Nonuniform penalty weights

In the previous chapter we showed how to generalize the elastic one-norm formulation by incorporating penalty weights. In the max-norm formulation we cannot have weights in the penalty term because there is a single scalar elastic variable. However, we can put weights in the constraints:

$$\min_{x,\tau}\ F(x) + \gamma\tau \tag{3.3}$$

$$\text{s.t.}\ c(x) + \tau v \geq 0. \tag{3.4}$$

Note that the weights v are inversely related to the weights w from the one-norm formulation in Section 2.2.3. That is, a small v_i corresponds to a large w_i, and vice versa. In particular, $v_i = 0$ implies that the constraint $c_i \geq 0$ is *hard* and cannot be violated.

Note that the scaling of the constraints is important in this formulation, and this should be taken into account when the weights are chosen.

3.2 Theory

Exactly as in Section 2.2.2, we derive the following results.

Lemma 5 *Assume the NLP (1.1) has a nonempty feasible set. If (x^*, λ^*) is a first-order KKT point of (1.1), then $(x^*, 0, \lambda^*)$ is a first-order KKT point of (3.1) when*

$$\gamma \geq \sum_i \lambda_i^*.$$

The only thing that has changed from Lemma 1 is the value of the bound for γ. We skip the proof because it is essentially the same as for Lemma 1. Again, we have a corollary in the convex case.

Corollary 6 *Assume the NLP (1.1) is convex and has a nonempty feasible set. If x^* is a minimizer of (1.1), then $(x^*, 0)$ is a minimizer of (3.1) when*

$$\gamma \geq \sum_i \lambda_i^*,$$

where λ^* are the Lagrange multipliers for (1.1) at x^*.

3.3 Bounding τ

As described earlier, we assume the objective and constraints are well-defined and bounded on an extension of the feasible region,

$$\|c^-(x)\|_\infty \leq \beta, \tag{3.5}$$

where we this time consider the max-norm. This condition is equivalent to $c(x) \geq -\beta e$. It is therefore desirable to bound the elastic variable τ by $\tau \leq \beta$. Analogously to Section 2.4, we need to show that the elastic QP has a nonempty feasible region also when this additional constraint is present.

Lemma 7 *Assume c, J, τ and β are given, where $0 \leq \tau \leq \beta$ and $c_i \geq -\beta$ for all i. Then there exists a feasible point for the system of linear inequalities*

$$Jp + \theta e \geq -(c + \tau e), \tag{3.6}$$

$$\theta \geq -\tau, \tag{3.7}$$

$$\theta \leq \beta - \tau. \tag{3.8}$$

Furthermore, when J has full row rank, there is a feasible point with $\theta < \beta - \tau$.

Proof: Consider the LP

$$\min_{p,\theta} \ \theta \tag{3.9}$$

$$\text{s.t. } Jp + \theta e \geq -(c + \tau e), \tag{3.10}$$

$$\theta \geq -\tau. \tag{3.11}$$

We wish to show that a minimizer θ^* satisfies $\theta^* \leq \beta - \tau$. It is easy to verify that $p_0 = 0$, $\theta_0 = \max\{-\tau, \max_i -(c_i + \tau)\}$ is a feasible point. By assumption, $c_i \geq -\beta$ so $\theta_0 \leq \beta - \tau$. If θ^* is a minimizer, then $\theta^* \leq \beta - \tau$. This proves the main part of the lemma.

Next we prove the last part of the lemma. Let $\hat{J}$ correspond to the working set, a set of active constraints ($c_i = -\tau$). The working matrix for (3.10)–(3.11) is of the form

$$\tilde{J} = \begin{pmatrix} \hat{J} & e \end{pmatrix}. \tag{3.12}$$

If the lower bound on θ is active then $p = 0$, $\theta = -\tau$ is a feasible point, so we assume in the following that the bound is not active.

The working set contains at most n constraints, where n is the dimension of p; hence the null space of $\tilde{J}$ is nonempty. Let $\hat{Z}$ be a basis for the null space of $\hat{J}$. A corresponding null space for $\tilde{J}$ is of the form

$$Z = \begin{pmatrix} \hat{Z} & y \\ 0 & -1 \end{pmatrix}, \tag{3.13}$$

provided there exists a y such that $\hat{J}y = e$. Suppose this is the case. Define the search direction v by

$$v = -ZZ^T g, \tag{3.14}$$

where $g = (0, \dots, 0, 1)^T$ is the gradient of the objective. The descent along v is

$$g^T v = -g^T Z Z^T g = -1, \tag{3.15}$$

so v is a direct of strict descent, which is what we wanted. The only case where there is no descent is when $Z^T g = 0$. This can only occur when there is no y such that $\hat{J}y = e$, which implies that $\hat{J}$ has low row rank. Because $\hat{J}$ is a submatrix of J, J must also have low row rank if $\hat{J}$ has. $\square$

3.4 Updating γ

In this section we show that γ in the max-norm variant should be updated the same way as for the one-norm version.

Consider the elastic nonlinear problem

$$\min_x F(x) + \gamma\tau \tag{3.16}$$

$$\text{s.t. } c(x) + \tau e \geq 0 \tag{3.17}$$

$$\tau \geq 0 \tag{3.18}$$

$$\tau \leq \beta. \tag{3.19}$$

Let λ be the Lagrange multipliers for the nonlinear constraints and let μ and ν be the multipliers for the bounds on τ. The first-order optimality conditions are

$$\begin{pmatrix} J^T & 0 & 0 \\ e^T & 1 & -1 \end{pmatrix} \begin{pmatrix} \lambda^* \\ \mu^* \\ \nu^* \end{pmatrix} = \begin{pmatrix} g^* \\ \gamma \end{pmatrix}, \tag{3.20}$$

$$\lambda^*, \mu^*, \nu^* \geq 0, \tag{3.21}$$

which gives

$$J^T\lambda^* = g^*, \tag{3.22}$$

$$e^T\lambda^* + \mu^* = \gamma + \nu^*. \tag{3.23}$$

If $\nu^* > 0$, constraint (3.19) is active, so we have to increase γ. Based on (3.23), a natural update would be

$$\gamma_{new} = \gamma + \nu^*, \tag{3.24}$$

but as we have argued earlier, it is better to choose γ larger than than the estimated minimum value. Thus we suggest the same update as in (2.76):

$$\gamma_{new} = 2(\gamma + \nu^*). \tag{3.25}$$

3.5 Updating s and τ based on the merit function

The augmented Lagrangian merit function for the l_∞-elastic problem is

$$\tilde{M}(x, s, \tau, \rho) = F(x) + \gamma\tau + \lambda^T(c - s + \tau e) + \frac{1}{2}(c - s + \tau e)^T D_\rho (c - s + \tau e). \tag{3.26}$$

We wish to find nonnegative τ and s that minimize M, assuming all other variables are fixed. Unfortunately, this optimization problem does not decompose into several two-dimensional problems that can easily solved, like with the l_1 norm. Similarly to the result for the l_1 case, we show that either $\tau = 0$ or at least one $s_i = 0$.

Suppose the bounds $s = 0$ and $\tau = 0$ are not active. The optimality conditions are

$$\frac{\partial \tilde{M}}{\partial s} = -\lambda - D_\rho(c - s + \tau e) = 0, \tag{3.27}$$

$$\frac{\partial \tilde{M}}{\partial \tau} = \gamma + \lambda^T e + e^T D_\rho(c - s + t) = 0. \tag{3.28}$$

If we take the inner product of (3.27) and e and compare with (3.28), it becomes clear that these equations can only have a solution if $\gamma = 0$, which is false.

If $\tau = 0$, then each optimal s_i can be computed separately from (3.27). A more difficult case is when $\tau > 0$. For a fixed value of τ, again the optimal s can be computed from (3.27). Consequently, we can treat s as a function of τ and we only need to solve a one-dimensional optimization problem to determine τ.

3.6 Efficient elastic SQP methods

We have described a simple elastic SQP method for the case when there are only general inequalities. In a real code, we need to emphasize efficiency. In this section we study how to make the algorithm more efficient and how to handle equality constraints and distinguish between various types of constraint (simple bounds, linear, nonlinear). In order to analyze these aspects, we need to go into much more detail than we have done so far.

3.6.1 Working-set Jacobian and null space representations

In this section we review standard methods for representing the working-set Jacobian, $\hat{J}$, and its corresponding null space. Different representations are used in the small, dense case and the large, sparse case.

The small, dense case (NPSOL)

At each QP iteration, a *working set* of constraints is identified. All the constraints in the working set are linearly independent. A constraint in the working set can correspond to a (simple) bound constraint, a linear constraint, or a linearized nonlinear constraint. NPSOL uses the TQ factorization of $\hat{J}$, given by

$$\hat{J}Q = \begin{pmatrix} 0 & T \end{pmatrix}, \tag{3.29}$$

where T is a square, nonsingular, and reverse-triangular matrix, and Q is orthogonal. The *transformed* Hessian is defined by

$$H_Q = Q^T H Q. \tag{3.30}$$

Let the columns of Q be partitioned so that

$$Q = \begin{pmatrix} Z & Y \end{pmatrix}, \tag{3.31}$$

where the columns of Z form a basis for the null space of $\hat{J}$. The matrix Z is needed to compute the reduced gradient $Z^T g$. NPSOL maintains the Cholesky factor R of H_Q, such that $R^T R = H_Q$. Note that the reduced Hessian, $H_Z = Z^T H Z$, is the upper left portion of the transformed Hessian. This is used (and updated) by NPSOL's QP solver LSSOL.

When a bound constraint is active, the corresponding variable is *fixed*. In this case, we can simply eliminate the corresponding column in the Jacobian $\hat{J}$, since that variable cannot change.

The large, sparse case (SNOPT)

For large and sparse problems, $\hat{J}$ is typically stored as a sparse matrix, and Z may be given only implicitly. At an iteration in an active-set method there are some linear and nonlinear constraints active, and also some variables are fixed on their bounds. This gives rise to a working-set Jacobian of the form

$$\hat{J} = \begin{pmatrix} B & S & N \\ 0 & 0 & I \end{pmatrix}, \tag{3.32}$$

where B is square and nonsingular. Possibly the variables (columns) must be permuted to obtain this partitioned form. Borrowing terms from linear programming, we say the variables corresponding to the columns of B are *basic*, the variables corresponding to the columns of N are *nonbasic* (or fixed), while the variables corresponding to the columns of S are called *superbasic*. A basis for the null space of $\hat{J}$ is given by

$$Z = \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}. \tag{3.33}$$

The matrices Z and B^{-1} are not computed explicitly because only the actions of Z and Z^T are required in optimization algorithms. Thus it is sufficient to factor B such that $Bx = y$ and $B^Tx = y$ can be solved efficiently for any vector y.

The reduced Hessian Z^THZ is computed and stored as a dense matrix in SNOPT's QP solver SQOPT, since the dimension of the null space is assumed to be small. Note that Z^THZ may be much denser than Z or H.

3.6.2 Equality constraints

So far we have considered only inequality constraints. The simplest elastic formulation in the max-norm would be

$$c(x) + \tau e - \nu e = 0, \quad \tau, \nu \geq 0, \tag{3.34}$$

but this is not a valid elastic formulation. There are no scalars τ and ν such that equality holds in every component. Consequently, with scalar elastic variables we must change to a formulation based on inequalities. The natural max-norm elastic formulation for a constraint $c_i(x) = 0$ is

$$|c_i(x)| \le \tau, \tag{3.35}$$

which can be written

$$c(x) + \tau e \ge 0, \tag{3.36}$$

$$-c(x) + \tau e \ge 0, \quad \tau \ge 0, \tag{3.37}$$

for a set of constraints $c(x) = 0$.

Equality constraints are a special case of inequality constraints with both lower and upper bounds. The max-norm elastic formulation of $l \le c(x) \le u$ is

$$c(x) + \tau e \ge l, \tag{3.38}$$

$$-c(x) + \tau e \ge -u, \quad \tau \ge 0. \tag{3.39}$$

Another option is to transform all the general constraints (equalities and inequalities) into equalities by adding new slack variables. The new slack variables will have simple bounds, and we can add elastic variables on these. More formally,

$$l \le c(x) \le u \tag{3.40}$$

will be transformed to

$$c(x) - s = 0, \quad l \le s \le u. \tag{3.41}$$

We allow the bounds on s to be violated, but not the equalities. This strategy is only useful if simple bounds can be made elastic in an efficient way. That is the topic of the next section. Note that the slack variables s can often be represented implicitly in an implementation.

3.6.3 Bound constraints

Adding elastic variables to the general (linear or nonlinear) constraints does not change the nature of these constraints. But adding variables t or τe to the simple bounds on x transforms these to general linear constraints, which may require more work. One way to avoid this problem is to insist on always satisfying the simple bounds. However, this is not quite satisfactory because there is little reason to believe the simple bounds are always more important to satisfy than the general constraints. In fact, we claim that elastic simple bounds would be quite useful. In many applications, the general constraints correspond to physical laws (which are not elastic), while simple bounds are often chosen by the user and may be flexible. Nevertheless, many simple bounds are of the type $x_j \geq 0$, and such nonnegativity bounds are almost always hard bounds. The user should therefore be given the opportunity to specify which constraints and bounds should be hard (inelastic) and which should be soft (elastic).

Simple bounds need to be treated separately because when a simple bound is active we can eliminate the corresponding column in the Jacobian, and similarly for the Hessian. In effect, simple bounds let us reduce the problem size. When the additional elastic variables are introduced, it appears that this attractive property is lost. However, there are ways around this snag. One approach has been implemented in SNOPT and SQOPT, while we show a different approach (based on the max-norm) in which the special structure of the elastic problem is taken advantage of.

3.6.4 Only simple bound constraints are elastic

A special formulation of NLP is when the general constraints are equalities and the simple bounds are the only inequalities:

$$\begin{aligned} \min_{x,t} \quad & F(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & l \leq x \leq u \end{aligned} \tag{3.42}$$

Any NLP can be written in this form with the help of slack variables. In this section, we consider the case where only the simple bounds are allowed to become elastic.

For bound constraints we cannot afford to introduce many new variables, so we choose to focus on the variant where the bounds $l \leq x \leq u$ are changed into

$$x + \tau e \geq l, \quad x - \tau e \leq u, \quad \tau \geq 0. \tag{3.43}$$

Here τ is a scalar elastic variable.

For purpose of exposition, in the following we ignore the upper bounds and assume only lower bounds are present. (Upper bounds do not change the nature of the problem but make the formulae more complicated.) We are left with the optimization problem

$$\begin{aligned} \min_{x,t} \quad & F(x) + \gamma\tau \\ \text{s.t.} \quad & c(x) = 0 \\ & x + \tau e \geq l \\ & \tau \geq 0. \end{aligned} \tag{3.44}$$

This system has a special structure because the linear constraints involve only two variables each. The Jacobian has a sparse structure that should be exploited.

The QP corresponding to (3.44) is

$$\begin{aligned} \min_{p,\theta} \quad & \tfrac{1}{2}p^THp + g^Tp + \gamma\theta \\ \text{s.t.} \quad & Jp = -c \\ & p + \theta e \geq \bar{l} \\ & \theta \geq -\tau, \end{aligned} \tag{3.45}$$

where $\bar{l} = l - x - \tau e$. The Jacobian for all the constraints is

$$\begin{pmatrix} J & 0 \\ I & e \\ 0 & 1 \end{pmatrix}. \tag{3.46}$$

We are interested in $\hat{J}$, the Jacobian of the working set. Since J corresponds to equality constraints, they are all active. Only some of the elastic simple bounds will be active. (Since a bound pair can be violated only in one direction at any given point, the working set would contain at most one row from each bound pair. Thus we have lost no generality by considering only lower bounds.) We will assume the constraint $\theta \geq -\tau$ is *not* active, since otherwise we can set $\tau = 0$ in the next iteration, and a feasible point is obtained. Hence the Jacobian of the working set must have the following structure:

$$\hat{J} = \begin{pmatrix} J & 0 \\ \hat{I} & e \end{pmatrix}, \tag{3.47}$$

where $\hat{I}$ is a subset of the rows of the identity matrix, and e is a correspondingly shorter vector of ones. We now reorder the free variables before the fixed variables. Redefine $\hat{J}$ such that

$$\hat{J} = \begin{pmatrix} J_1 & J_2 & 0 \\ 0 & I & \tilde{e} \end{pmatrix}, \tag{3.48}$$

where $[J_1 \ J_2]$ is a column permutation of J.

In dense methods, we assume an orthogonal basis Z_1 for the null space of J_1 is available. This is reasonable because such a Z_1 would have been computed by a standard algorithm. We seek a Z that is a basis for the null space of $\hat{J}$. It is easy to verify that

$$Z = \begin{pmatrix} Z_1 & v \\ 0 & -e \\ 0 & 1 \end{pmatrix} \tag{3.49}$$

satisfies $\hat{J}Z = 0$ when v is chosen such that

$$J_1 v = J_2 e. \tag{3.50}$$

In order for the columns of Z to be orthogonal, $Z_1^T v = 0$ is required. Consequently, we define v to be the solution of

$$\begin{pmatrix} J_1 \\ Z_1^T \end{pmatrix} v = \begin{pmatrix} J_2 e \\ 0 \end{pmatrix}. \tag{3.51}$$

A simple scaling of the last column in Z such that it has unit norm makes Z orthogonal. Specifically, let $\alpha = \left(v^T v + k + 1\right)^{-1/2}$, where k is the number of active bound constraints. Then

$$Z = \begin{pmatrix} Z_1 & \alpha v \\ 0 & -\alpha e \\ 0 & \alpha \end{pmatrix} \tag{3.52}$$

is an orthogonal basis for the null space of the Jacobian.

Null-space based optimization algorithms require access to the reduced gradient, $Z^T g$, and the reduced Hessian, $Z^T H Z$. These quantities can be calculated fairly simply in our case. Assume as before that we order the *free* variables before the *fixed* variables, and that the elastic variable comes last. Assume the Hessian approximation H is of the form

$$H = \begin{pmatrix} H_{11} & H_{21}^T & 0 \\ H_{21} & H_{22} & 0 \\ 0 & 0 & H_\tau \end{pmatrix}, \tag{3.53}$$

where some blocks are zero because there is no coupling between τ and the original variables. Similarly, let the gradient g be partitioned as

$$g = \begin{pmatrix} g_1 \\ g_2 \\ \gamma \end{pmatrix}. \tag{3.54}$$

We find that

$$Z^T H Z = \begin{pmatrix} Z_1^T H_{11} Z_1 & \alpha Z_1^T (H_{11} v - H_{21}^T e) \\ \alpha (H_{11} v - H_{21}^T e)^T Z_1 & \alpha^2 \left(v^T H_{11} v - 2e^T H_{21} v + e^T H_{22} e + H_\tau \right) \end{pmatrix}, \tag{3.55}$$

and the reduced gradient is

$$Z^T g = \begin{pmatrix} Z_1^T g_1 \\ \alpha \left(v^T g_1 - e^T g_2 + \gamma \right) \end{pmatrix}. \tag{3.56}$$

These quantities cost only slightly more to compute than $Z_1^T H_{11} Z_1$ and $Z_1^T g_1$, which are computed in a standard algorithm.

3.7 More on elastic bounds

Consider the NLP

$$\min_x F(x) \tag{3.57}$$

$$\text{s.t. } l_c \le c(x) \le u_c \tag{3.58}$$

$$l_x \le x \le u_x, \tag{3.59}$$

where we have separated the simple bounds and the general constraints. This formulation is equivalent to (1.2), where the linear constraints have been merged into $c(x)$. By introducing slack variables s, we get

$$\min_x F(x) \tag{3.60}$$
$$\text{s.t. } c(x) - s = 0 \tag{3.61}$$
$$l_c \le s \le u_c, \tag{3.62}$$
$$l_x \le x \le u_x, \tag{3.63}$$

The simple bounds can now be made elastic. Making the bounds on s elastic corresponds to elastic bounds on the nonlinear constraints (3.58).

At first one may think that elastic variable bounds are no different from elastic slack variable bounds, but this is not quite true. We compare several cases. First consider the following simple version of NLP:

$$\min_x F(x) \tag{3.64}$$
$$\text{s.t. } c(x) - s = 0 \tag{3.65}$$
$$s \ge 0. \tag{3.66}$$

Let A be the Jacobian of $c(x)$. The working-set Jacobian, $\hat{J}_1$, is

$$\hat{J}_1 = \begin{pmatrix} A & -I \\ 0 & \bar{I} \end{pmatrix}, \tag{3.67}$$

where $\bar{I}$ is a row subset of the identity matrix I. We want to find Z_1, a basis for the null space of $\hat{J}_1$. This can be done in a standard fashion by identifying sets of basic, superbasic, and nonbasic variables. With a suitable permutation P, we have

$$\hat{J}_1 = \begin{pmatrix} A_B & -I_B & A_S & -I_N \\ 0 & 0 & 0 & I \end{pmatrix} P, \tag{3.68}$$

where A and $-I$ have been partitioned into two submatrices each, and $B = (A_B,\ -I_B)$ is square and nonsingular. From Section 3.6.1 we know that a basis matrix for the null space of $\hat{J}_1$ is

$$Z_1 = P \begin{pmatrix} -B^{-1}A_S \\ I \\ 0 \end{pmatrix}. \tag{3.69}$$

Consider next the elastic variation

$$\min_{x,t} \; F(x) + \gamma e^T t \tag{3.70}$$

$$\text{s.t. } c(x) - s = 0 \tag{3.71}$$

$$s + t \geq 0 \tag{3.72}$$

$$t \geq 0. \tag{3.73}$$

The slack bounds (essentially, the nonlinear constraints) have been made elastic, while there are no simple bounds on x. The working-set Jacobian, $\hat{J}_2$, is

$$\hat{J}_2 = \begin{pmatrix} A & -I & 0 \\ 0 & \bar{I} & \bar{I} \end{pmatrix}. \tag{3.74}$$

An advantage of this variation is that there is a null space matrix, Z_2, that has a simple form derived from (3.69):

$$Z_2 = \tilde{P} \begin{pmatrix} Z_1 & 0 \\ 0 & \bar{I}' \\ 0 & 0 \end{pmatrix}, \tag{3.75}$$

where $\bar{I}'$ is a modification of the identity matrix such that $\bar{I}\bar{I}' = 0$. $\bar{I}'$ may easily be constructed by taking the identity matrix I and eliminating all the rows contained in $\bar{I}$. Let us verify that $\hat{J}_2 Z_2 = 0$. Note that $\hat{J}_2 = \begin{pmatrix} \hat{J}_1 & \hat{I} \end{pmatrix}$, where $\hat{I}^T = \begin{pmatrix} 0 & \bar{I}^T \end{pmatrix}$. Then

$$\hat{J}_2 Z_2 = \begin{pmatrix} \hat{J}_1 & \hat{I} \end{pmatrix} \begin{pmatrix} Z & 0 \\ 0 & \bar{I}' \end{pmatrix} = \begin{pmatrix} 0 & \hat{I}\bar{I}' \end{pmatrix} = 0, \tag{3.76}$$

because $\hat{I}\bar{I}' = \bar{I}\bar{I}' = 0$.

The number of columns of Z_2 is greater than that of Z_1, and consequently the size of the reduced Hessian Z^THZ will increase. However, the additional work introduced in a reduced Hessian algorithm is modest because of the simple structure of Z.

Consider next the case where there are simple bounds on x and the nonlinear constraints are equalities (slack variables s may be introduced to handle inequalities):

$$\min_{x,t} F(x) \tag{3.77}$$

$$\text{s.t. } c(x) = 0 \tag{3.78}$$

$$x \geq 0. \tag{3.79}$$

Suppose we wish the simple bounds be elastic, but not the general nonlinear constraints. This gives

$$\min_{x,t} F(x) + \gamma e^T t \tag{3.80}$$

$$\text{s.t. } c(x) = 0 \tag{3.81}$$

$$x + t \geq 0 \tag{3.82}$$

$$t \geq 0. \tag{3.83}$$

Now the working-set Jacobian has the form

$$\hat{J}_3 = \begin{pmatrix} A & 0 \\ \bar{I} & \bar{I} \end{pmatrix}. \tag{3.84}$$

Unlike with (3.67) or (3.74), it is not simple to obtain a sparse or structured Z, because A is coupled with $\bar{I}$. Compared to the original problem, m_b elastic variables have been added, where m_b is the number of simple bounds on variables. Let k be the number of active bounds. Typically, $k \ll m_b$. The size of the null space Z and the reduced Hessian Z^THZ increases by roughly $m_b - k$. Consequently, algorithms that operate on the reduced Hessian may incur (substantially) more work and run slower than in standard (nonelastic) mode.

So far in this section, we have discussed only l_1-elastic formulations. Let us consider the

l_∞-elastic version of (3.80)–(3.83).

$$\min_{x,\tau} F(x) + \gamma\tau \tag{3.85}$$

$$\text{s.t. } c(x) = 0 \tag{3.86}$$

$$x + \tau e \geq 0 \tag{3.87}$$

$$\tau \geq 0. \tag{3.88}$$

Now the working-set Jacobian has the form

$$\hat{J}_4 = \begin{pmatrix} A & 0 \\ \bar{I} & e \end{pmatrix}. \tag{3.89}$$

In this case, only one elastic variable is introduced, and the size of the null space increases by at most one. This holds no matter how large m_b and k become. We conclude that the l_∞ formulation has an advantage over the l_1 formulation when it comes to simple bounds. A simple example illustrates our point. Consider an optimization problem of the form (3.77) with n variables, $n/2$ nonlinear constraints, and $n/2$ violated bounds. In the l_1 version, $n/2$ elastic variables are added, so the degrees of freedom (number of variables minus number of active constraints) are $n+n/2-n = n/2$. In comparison, with the l_∞ version only one elastic variable is introduced, hence there is only one degree of freedom (because $n + 1 - n = 1$).

Another way to appreciate the advantage of the l_∞ approach is to realize that all the violated simple bounds have the same optimal value. In other words, instead of a possibly large number of t_i values, only a single scalar τ is required.

Chapter 4

Directions of negative curvature

4.1 Overview

In this chapter, we consider the unconstrained minimization problem

$$\min_x F(x), \quad x \in \mathbb{R}^n, \tag{4.1}$$

where the function $F(x)$ is assumed to be at least twice differentiable. As before, we denote the gradient of F by g and the Hessian (or an approximation thereof) by H. It is implied that these quantities are evaluated at the same point (the current point).

For many optimization algorithms, one can only prove convergence to *first-order* points. That is, we may find a saddle point while we are really seeking a minimizer. Any method for minimization that can be shown to converge to a point that satisfies the *second-order* necessary conditions must explicitly or implicitly compute a direction of negative curvature of an indefinite matrix. In this and the following chapter we study how to compute *good* directions of negative curvature and show that this problem is closely related to the symmetric eigenvalue problem.

This chapter is only loosely related to the earlier chapters, and is largely self-contained. Nevertheless, there is a link to the preceding work. When trying to reduce the infeasibilities we may end up at a saddle point for the infeasibilities, so there is no descent direction. We then need to compute a direction of negative curvature to move away from the saddle point. Using directions of negative curvature earlier in the process helps avoid such saddle points, and may aid faster convergence to minimizers.

We begin this chapter by describing the importance of directions of negative curvature when we want to find second-order points. In Section 4.2 we examine the distribution of the Rayleigh quotient, which tells us how difficult it is to find a direction of negative curvature. This leads us to the symmetric eigenvalue problem (Section 4.3). In Chapter 5 we describe algorithms for computing directions of negative curvature. We consider both direct and iterative methods, and suggest a new hybrid based on combining a direct and iterative approach.

4.1.1 Why do we need directions of negative curvature?

Given a function $F(x) \in C^2, x \in \mathbb{R}^n$, with gradient $g = g(x) = \nabla F(x)$, a descent direction d has the property that

$$g^T d < 0.$$

Similarly, let the Hessian $\nabla^2 F(x)$ be denoted by H (or $H(x)$ when the argument is relevant). A direction of negative curvature, d, has the property that

$$d^T H d < 0.$$

Any method for minimization (such as a linesearch method, a trust-region method, or a gradient flow approach) for which convergence to second-order optimality conditions can be shown, must either explicitly or implicitly compute a direction of negative curvature. Suppose the current estimate is a point satisfying the first-order but not the second-order conditions. To proceed, such algorithms must determine a point in the neighborhood of the saddle point that has a lower value than the current iterate. Such points must lie along a direction of negative curvature. It is also our contention that the relevance of computing directions of negative curvature grows with the dimension of the problem. This latter hypothesis is based on the probability that a direction of negative curvature occurring by chance decreases significantly (exponentially) with increasing dimension. For example, we will show that the probability of a random direction being a direction of negative curvature for a matrix with 99 eigenvalues of 1 and one of -1 is less then 10^{-15}.

In this chapter we discuss methods to compute directions of negative curvature when n is large. We desire the effort to compute the direction to be similar to or less than that of computing the direction of descent.

An interesting property of directions of negative curvature not possessed by directions of descent, is that we can define precisely what is meant by a good direction simply by comparing it with the best possible. The best possible is the one that minimizes d^THd/d^Td, and that is given by the eigenvector corresponding to the smallest (leftmost) eigenvalue of H. Clearly the problem of computing a direction of negative curvature is similar to that of computing an eigenvector corresponding to the smallest eigenvalue. It differs in two ways; first we do not need it to be accurate, and secondly we may have additional information as a result of computing a direction of descent. Moreover, we shall be concerned with computing a sequence of directions, and the matrices that arise typically have very few negative eigenvalues.

A key property that is required to prove convergence to second-order KKT points is that of a *sufficient* direction of negative curvature. Formally, a *sequence* of directions, $\{d_k\}$, is said to be sufficient if any nonzero d_k is a non-ascent direction of negative curvature, i.e., $g_k^Td_k \le 0$, $d_k^TH_kd_k < 0$, where x_k is the kth iterate. Further, $\|d_k\|$ must be bounded and the curvature must be sufficient in the sense that

$$\lim_{k\in S} d_k^TH_kd_k = 0 \quad\Longrightarrow\quad \liminf_{k\in S} \lambda_{\min}(H_k) \ge 0 \text{ and } \lim_{k\in S} d_k = 0, \tag{4.2}$$

where S is any subsequence. In practice, $\lambda_{\min}$ is not known, so this has come to require that if d_k satisfies $d_k^Td_k = 1$ then

$$d_k^TH_kd_k \le \beta\lambda_{\min}, \tag{4.3}$$

where β is some scalar between zero and one (e.g., $\beta = 0.1$) and $\lambda_{\min}$ is the smallest eigenvalue of H_k. In other words, relative to the best possible direction of negative curvature, d_k is not arbitrarily poor. Computing such a direction is not easy. Both direct and iterative methods have been proposed, but this process is usually only a small part of an optimization algorithm, so we do not wish to spend much effort. This implies we are not willing to use pivoting in a direct method (reordering affects the sparsity pattern), and we may only be willing to perform relatively few matrix–vector products in an iterative method.

It should be noted that the strictures placed on the direction of negative curvature by the need to prove convergence in a linesearch method, while important, are not overriding. The likelihood of the modified Cholesky algorithm failing to obtain a direction of sufficient negative curvature is small (because β may be small). Our interest is in obtaining *good* directions of negative curvature, if this can be done routinely. By "good" we mean that β is not too small, say $\beta \geq 0.1$. We believe that this will enhance the efficiency of minimization algorithms that use such directions.

4.2 The probability distribution of the Rayleigh quotient

4.2.1 Motivation

Let A be a real symmetric matrix. If we pick a random vector v, what is the probability that $v^TAv < 0$? Clearly this depends on the eigenvalues $\lambda(A)$ of A. Our interest in this question stems from wanting to know how difficult it is to find a direction of negative curvature. More specifically, we would also like to know how likely it is that by chance a direction of non-ascent that has been computed is a direction of negative curvature. These questions lead to a more general question. In linear algebra, the *Rayleigh quotient* of v with respect to A is defined as $R(v) = v^TAv/v^Tv$. The *field of values* is the range of the function $R(v)$. A natural next question is what is the probability density function (pdf) of $R(v)$? And what is the cumulative density function (cdf)? We will assume v is uniformly distributed on the unit sphere. (We can always normalize v because the Rayleigh quotient is invariant under scaling of v.) The answer to the questions posed can be deduced from the study of quadratic forms in random variables. These results have received little attention in the numerical linear algebra and optimization communities. In the following sections, we review some results from statistics and reformulate them in terms of linear algebra. We then apply these results to the questions of interest.

4.2.2 Random vectors on the sphere

In order to proceed, we need to know some properties about random vectors of unit length.

Lemma 8 *Pick $x \in \mathbb{R}^n$ such that each x_i is chosen independently from the standard normal distribution (i.e., the mean is zero and the variance is one). Then $v = x/\|x\|$ is a unit random vector uniformly distributed on the unit sphere $S^{n-1} \subset \mathbb{R}^n$.*

Proof: Let Q be any orthogonal matrix of dimension n. Let x be a vector with normally distributed elements as described above, and $v = x/\|x\|$. Define $\hat{x} = Qx$. Then $\hat{v} = \hat{x}/\|\hat{x}\| = Qx/\|x\|$. Clearly v and $\hat{v}$ have the same distribution on the unit sphere. Since Q can be any orthogonal matrix, the distribution must be uniform. $\square$

This result has been known at least since 1956 [40, p.130]. The uniform distribution over an (n-dimensional) sphere is sometimes referred to as the *Haar* distribution. The lemma immediately gives an algorithm for how to construct random unit vectors. (We assume a pseudo-random number generator is available.) There is a generalization of Lemma 8 for random orthogonal matrices. To generate a random orthogonal matrix, start with a square matrix with normally distributed elements and compute the QR factors. The resulting Q is a random orthogonal matrix. This algorithm requires $O(n^3)$ operations, but it is possible to reduce the work to $O(n^2)$ by using Householder transformations [75].

A consequence of Lemma 8 is that we can assume in our analysis that the random vectors have each component independently chosen from a standard normal distribution (because the Rayleigh quotient is invariant under scaling).

4.2.3 Quadratic forms in random variables

Statisticians have studied quadratic forms in random variables extensively; see [47] and [37, Ch.29] for an overview. A very extensive bibliography on the distribution of quadratic forms in normal variables has been collected by Dumais and Styan [14]. We first show that the distribution of the Rayleigh quotient can be derived from the distribution of a quadratic form. Observe that

$$Pr\left(\frac{x^TAx}{x^Tx} \leq \alpha\right) = Pr\left(x^T(A-\alpha I)x \leq 0\right). \tag{4.4}$$

We are interested in the case where x_i are normally distributed with mean zero. No assumptions on the definiteness of A are made. The interesting case is when $A - \alpha I$ is indefinite, otherwise the probability is trivially zero or one.

The distribution function (cdf), $F_R(\alpha)$, for the Rayleigh quotient is

$$F_R(\alpha) \equiv Pr\left(\frac{x^TAx}{x^Tx} \leq \alpha\right). \tag{4.5}$$

Many distributions have no simple closed-form distribution function, but there is often a formula for the corresponding characteristic function ϕ, defined by

$$\phi(t) \equiv \int_{-\infty}^{\infty} e^{ity} dF(y), \tag{4.6}$$

where $i = \sqrt{-1}$ and t is real. It has been shown [47] that the characteristic function for the quadratic form $Q(x) = \sum_{j=1}^{n} \lambda_j x_j^2$ is

$$\phi_Q(t) = \prod_{j=1}^{n} (1 - 2i\lambda_j t)^{-\frac{1}{2}}. \tag{4.7}$$

To determine the corresponding distribution function F_Q we need to invert this formula. Gil-Pelaez [23] has derived the formula

$$F_Q(y; \lambda) = \frac{1}{2} - \frac{1}{\pi} \int_0^\infty t^{-1} \mathfrak{Im} \left[e^{-ity} \phi_Q(t) \right] dt. \tag{4.8}$$

We examine the case $y = 0$, in which case the previous formula expands to

$$F_Q(\lambda) = \frac{1}{2} - \frac{1}{\pi} \int_0^\infty t^{-1} \mathfrak{Im} \left[\prod_{j=1}^{n} (1 - 2i\lambda_j t)^{-\frac{1}{2}} \right] dt. \tag{4.9}$$

This formula is centered at the middle of the distribution and is not suitable for numerical evaluation when $F_Q \approx 0$ or $F_Q \approx 1$ due to cancellations. We are interested in the cases when F_Q is small and would therefore like a formula expanded around the left tail. When $\lambda > 0$, the quadratic form is positive definite, and thus $F_Q(\lambda) = 0$. From (4.9) it follows that

$$\int_0^\infty t^{-1} \mathfrak{Im} \left[\prod_{j=1}^{n} (1 - 2i\lambda_j t)^{-\frac{1}{2}} \right] dt = \frac{\pi}{2} \tag{4.10}$$

when all $\lambda_j > 0$. After we combine (4.9) and (4.10) and group terms with positive and negative eigenvalues, we find that

$$F_Q(\lambda) = \frac{2}{\pi} \int_0^\infty t^{-1} \mathfrak{Re} \left[\prod_{\lambda_j > 0} (1 - 2i\lambda_j t)^{-\frac{1}{2}} \right] \mathfrak{Im} \left[\prod_{\lambda_j < 0} (1 + 2i\lambda_j t)^{-\frac{1}{2}} \right] dt. \tag{4.11}$$

Imhof [35] found a formula that only involves real arithmetic and is more suitable for numerical integration:

$$F_Q(\lambda) = \frac{1}{2} - \frac{1}{\pi}\int_0^\infty \frac{\sin\left(\frac{1}{2}\sum_{j=1}^n \tan^{-1}(\lambda_j u)\right)}{u\prod_{j=1}^n \left(1+\lambda_j^2 u^2\right)^{\frac{1}{4}}} du. \tag{4.12}$$

Again, we exploit the fact that $F_Q(\lambda) = 0$ when all $\lambda_j > 0$ and obtain

$$\begin{aligned}
F_Q(\lambda) &= \frac{1}{\pi}\int_0^\infty \frac{\sin\left(\frac{1}{2}\sum \tan^{-1}(|\lambda_j|u)\right) - \sin\left(\frac{1}{2}\sum \tan^{-1}(\lambda_j u)\right)}{u\prod_{j=1}^n \left(1+\lambda_j^2 u^2\right)^{\frac{1}{4}}} du && (4.13)\\
&= \frac{1}{\pi}\int_0^\infty \frac{\sin(a+b) - \sin(a-b)}{u\prod_{j=1}^n \left(1+\lambda_j^2 u^2\right)^{\frac{1}{4}}} du, \quad \begin{cases} a = \frac{1}{2}\sum_{\lambda_j>0} \tan^{-1}(\lambda_j u), \\ b = \frac{1}{2}\sum_{\lambda_j<0} \tan^{-1}(-\lambda_j u), \end{cases} && (4.14)\\
&= \frac{1}{\pi}\int_0^\infty \frac{2\cos(a)\sin(b)}{u\prod_{j=1}^n \left(1+\lambda_j^2 u^2\right)^{\frac{1}{4}}} du && (4.15)\\
&= \frac{2}{\pi}\int_0^\infty \frac{\cos\left(\frac{1}{2}\sum_{\lambda_j>0} \tan^{-1}(\lambda_j u)\right)\sin\left(\frac{1}{2}\sum_{\lambda_j<0} \tan^{-1}(-\lambda_j u)\right)}{u\prod_{j=1}^n \left(1+\lambda_j^2 u^2\right)^{\frac{1}{4}}} du. && (4.16)
\end{aligned}$$

We are now ready to state our main result.

Theorem 9 *Let A be a real symmetric matrix with eigenvalues $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_n)$. Then the probability distribution function $F_R(\alpha; \lambda)$ of the Rayleigh quotient over all vectors $x \in \mathbb{R}^n$ with unit norm (i.e., x is uniformly distributed on a sphere) is*

$$F_R(\alpha;\lambda) = \frac{1}{2} - \frac{1}{\pi}\int_0^\infty \frac{\sin\left(\frac{1}{2}\sum_{j=1}^n \tan^{-1}((\lambda_j-\alpha)u)\right)}{u\prod_{j=1}^n (1+(\lambda_j-\alpha)^2 u^2)^{\frac{1}{4}}} du. \tag{4.17}$$

This theorem follows directly from the Imhof formula (4.12). Formula (4.17) is well suited for numerical integration because the integrand is smooth and decays rapidly as $u \to \infty$. However, a drawback is that the numerical round-off error can be significant when F_R is small because the formula is centered around one half. Therefore, we propose the equivalent formula

$$F_R(\alpha;\lambda) = \frac{2}{\pi}\int_0^\infty \frac{\cos\left(\frac{1}{2}\sum_{\lambda_j>\alpha}\tan^{-1}((\lambda_j-\alpha)u)\right)\sin\left(\frac{1}{2}\sum_{\lambda_j<\alpha}\tan^{-1}((\alpha-\lambda_j)u)\right)}{u\prod_{j=1}^{n}\left(1+(\lambda_j-\alpha)^2u^2\right)^{\frac{1}{4}}}du. \tag{4.18}$$

Formula (4.18) has better numerical properties when F_R is small. This formula can be evaluated by any standard method for numerical integration, for example (Gaussian) quadrature. Since the integrand decays rapidly, the domain of integration can be truncated to a finite interval and the truncation error will be small. The formula is still not perfectly suited for numerical evaluation, since in some cases the integrand oscillates around zero and again cancellations may give numerical errors. However, for our purposes it proved good enough.

4.2.4 Numerical experiments

As an illustration we evaluate the probability that $v^TA_nv < 0$ when A_n has one eigenvalue at -1 and the remaining $n-1$ at 1. We evaluated formulae (4.17) and (4.18) numerically using the NAG library. In both cases we asked for a relative accuracy of 10^{-2}. We used the integration routine D01AKF, a Gauss-Kronrod quadrature method, and the integration interval was truncated at 100. The results are shown in Table 4.1. The error estimates shown come from the NAG routine. We see that expression (4.18) is the more stable one and we were able to determine the probabilities up to $n = 100$ before too much precision was lost. If we plot the probabilities as a function of n, we get an almost straight line on a logarithmic scale (Figure 4.1). From these data, we have estimated that the probability of negative curvature, $F_R(A_n)$, is approximately proportional to $n^{0.70}$. Clearly for large n, encountering a direction of negative curvature by chance is exceedingly small.

4.3 The symmetric eigenvalue problem as an optimization problem

The linear eigenvalue problem $Ax = \lambda x$ is one of the most studied problems in mathematics. The problem of finding the smallest (or largest) eigenvalue can be viewed as an optimization problem. We are only concerned with the case when A is real and symmetric, and the

n	Formula (4.17)	Formula (4.18)
10	1.50e-02 ± 3.8e-04	1.50e-02 ± 2.2e-07
20	3.38e-04 ± 2.4e-05	3.38e-04 ± 5.7e-09
30	8.70e-06 ± 2.2e-07	8.70e-06 ± 9.9e-15
40	2.37e-07 ± 2.5e-04	2.37e-07 ± 1.2e-10
50	6.63e-09 ± 2.8e-08	6.63e-09 ± 2.7e-16
60	1.90e-10 ± 9.9e-07	1.90e-10 ± 2.6e-16
70	5.49e-12 ± 2.7e-05	5.50e-12 ± 2.5e-16
80	1.57e-13 ± 1.8e-03	1.61e-13 ± 2.3e-16
90	5.53e-13 ± 2.2e-03	4.74e-15 ± 2.0e-16
100	2.78e-16 ± 1.9e-08	1.36e-16 ± 1.8e-16

Table 4.1: Probabilities for negative curvature of A_n with one negative eigenvalue.

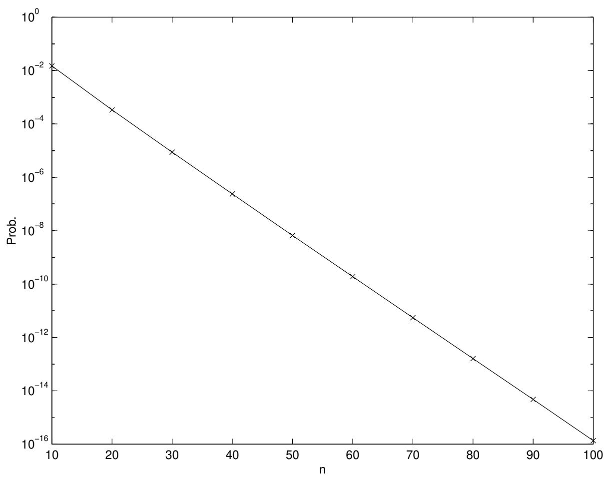

Figure 4.1: Plot of probabilities for negative curvature of A_n with one negative eigenvalue.

optimization problem is then

$$\min_{x \neq 0} \ R(x) \equiv \frac{x^T A x}{x^T x}, \tag{4.19}$$

where $R(x)$ is known as the Rayleigh quotient. We note that the scaling of the eigenvector x does not matter. Any multiple of a given x will give the same Rayleigh quotient and hence the same eigenvalue. Thus any minimizer for $R(x)$ is a weak minimizer, and the Hessian at a minimizer must be singular. This creates difficulties for many optimization algorithms, so we would like to change the problem to have a unique minimizer. A standard way of achieving this is to add a constraint on the norm of x, typically

$$\min_{x} \ x^T A x \quad \text{s.t. } x^T x = 1. \tag{4.20}$$

There is now a unique minimizer, assuming that the smallest eigenvalue is a simple eigenvalue. The Hessian has full rank at the minimum. Only the smallest eigenvalue produces a minimum; the others produce saddle points, except for the largest eigenvalue which gives a maximum.

The use of (4.19) or (4.20) to compute an extreme eigenvalue has been extensively studied in the literature [81, 58, 66]. A third option that is less studied is to use a penalty function to convert the constrained problem (4.20) into an unconstrained problem. The l_1, quadratic, and augmented Lagrangian penalty functions give

$$\begin{aligned} P_1(x) &= x^T A x + \rho |x^T x - 1|, && (4.21)\\ P_2(x) &= x^T A x + \rho (x^T x - 1)^2, && (4.22)\\ L_A(x,\mu) &= x^T A x - \mu(x^T x - 1) + \rho (x^T x - 1)^2. && (4.23) \end{aligned}$$

In the last equation we have called the Lagrange multiplier μ to avoid confusion with the eigenvalue λ (though at the optimum they have the same value).

In the following sections we review various methods for solving these unconstrained optimization problems. We remark that the most popular method today for finding extreme eigenvalues, the *Lanczos* algorithm, is not derived from the optimization point of view, though it has some optimal properties.

4.3.1 Line search methods

Some popular methods for unconstrained minimization are steepest descent, conjugate gradients, and Newton's method (see e.g. [27] or [46]). All these methods are iterative and of the form $x_{k+1} = x_k + \alpha_k p_k$, where p is called the search direction and α is the step length. (As usual, we skip subscripts when these are not important.) In the method of steepest descent p is the negative gradient, while in conjugate gradient methods the search directions are orthogonal with respect to an inner product related to the Hessian, H. In the standard Newton method, $p = -H^{-1}g$ and $\alpha = 1$. All the three methods mentioned can be made more flexible by choosing other values for α. This is called line search. An *exact* line search is when we select α to minimize the function along the line $x + \alpha p$. The value of α will therefore in general change for every iteration. For general problems, solving this one-dimensional minimization problem may be quite expensive, so *inexact* line search is preferred.

However, the Rayleigh quotient $R(x)$ is a special function where a closed form expression for the optimal step length α exists. This is easy to show. Dropping the subscripts, we want to minimize $R(x + \alpha p)$ for given x and p.

$$R(x + \alpha p) = \frac{(x + \alpha p)^T A (x + \alpha p)}{(x + \alpha p)^T (x + \alpha p)} \tag{4.24}$$

$$= \frac{x^T A x + 2p^T A x \alpha + p^T A p \alpha^2}{x^T x + 2p^T x \alpha + p^T p \alpha^2}. \tag{4.25}$$

We differentiate this with respect to α and solve $\frac{d}{d\alpha} R(x + \alpha p) = 0$. Keeping only the numerator we find that

$$\begin{aligned}\left(p^T A p\, p^T x - p^T p\, p^T A x\right) \alpha^2 + \left(p^T A p\, x^T x - p^T p\, x^T A x\right) \alpha \\ + \left(p^T A x\, x^T x - x^T A x\, p^T x\right) = 0.\end{aligned} \tag{4.26}$$

This is a quadratic equation in α so it can be solved analytically. Note that for certain choices of p, some terms are zero and the expression simplifies. Solving this equation yields two roots but only one of them corresponds to a minimizer.

4.3.2 Line search on the sphere

It is easy to adapt line search methods from unconstrained optimization to the case where there are linear constraints. The search direction must lie in the null space of the constraints. But in nonlinear optimization, there is no way to stay feasible with respect to a nonlinear constraint while moving along a straight line. There are several ways to tackle this problem. One approach is to accept that the iterates do not satisfy the nonlinear constraints during the iteration process, as long as all constraints are satisfied in the limit. A different approach is to ensure feasibility at each iteration by projecting the next iterate onto the feasible set in some way. Examples of this approach are gradient projection methods and the generalized reduced gradient method, GRG. These methods tend to work well when the constraints are nearly linear but often work poorly with highly nonlinear constraints.

A spherical constraint, e.g. $\|x\| = 1$, is a special case because the feasible set is a smooth surface (a differentiable manifold). The analogue of a line search is to search along a great circle on the sphere, also called geodesic search. This way the iterates x_k will always stay on the sphere and remain feasible. We now describe a geodesic search on the sphere for the Rayleigh quotient. Note that on the sphere the Rayleigh quotient is simply a quadratic form.

Assume we are given a point x, $\|x\| = 1$, and a search direction $\bar{p}$. Locally around x, the surface of the sphere is almost linear (flat) and the tangent space orthogonal to x. Therefore we can orthogonalize $\bar{p}$ against x and obtain a corresponding search direction on the sphere. More formally, we seek the projection, p, of $\bar{p}$ onto the tangent space of the sphere at x. This is given by

$$p = \frac{\bar{p} - (\bar{p}^T x)x}{\|\bar{p} - (\bar{p}^T x)x\|}. \tag{4.27}$$

We wish to search along the sphere from x in the direction p. The great circle of interest is uniquely defined by the points x, p, and the origin. A parametric equation for this curve is

$$s(\theta) = x\,\cos\theta + p\,\sin\theta, \quad \theta \in (-\pi, \pi]. \tag{4.28}$$

We could also have derived this equation by saying we want to search the intersection of the plane $\mathbf{span}\,\{x, \bar{p}\}$ and the sphere $\|x\| = 1$. Noting that $\mathbf{span}\,\{x, \bar{p}\} = \mathbf{span}\,\{x, p\}$, we arrive at (4.28).

The quadratic minimization problem on the curve (4.28) is

$$\min_{-\pi<\theta\leq\pi} (x\cos\theta + p\sin\theta)^T A(x\cos\theta + p\sin\theta) \tag{4.29}$$

$$= x^T Ax\cos^2\theta + 2p^T Ax\ \cos\theta\sin\theta + p^T Ap\ \sin^2\theta \tag{4.30}$$

We differentiate with respect to θ and get

$$-2\,x^T Ax\ \cos\theta\sin\theta + 2\,p^T Ax(\cos^2\theta - \sin^2\theta) + 2\,p^T Ap\ \sin\theta\cos\theta = 0 \tag{4.31}$$

$$\Rightarrow (p^T Ap - x^T Ax)\sin 2\theta + 2p^T Ax\cos 2\theta = 0 \tag{4.32}$$

$$\Rightarrow \tan 2\theta = \frac{2p^T Ax}{x^T Ax - p^T Ap}, \tag{4.33}$$

which gives

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2p^T Ax}{x^T Ax - p^T Ap}\right). \tag{4.34}$$

Note that $\theta \pm \pi$ are also possible solutions. In the special case $x^T Ax = p^T Ap$, $\cos 2\theta = 0$, so $\theta = \pi/4$.

In general this curvilinear search will yield a different point than the regular line search followed by a projection. Observe that since the projection of any point onto the unit sphere does not alter the Rayleigh quotient, we may at each iteration of a line search method obtain a point on the sphere by projection, with no change in the objective value. We wish to examine how this strategy compares to the geodesic search. The projection of a line onto a sphere is a geodesic curve on the "visible" half of the sphere. It is easily recognized that this curve is (4.28) with $-\pi/2 < \theta < \pi/2$. The search curve for the projected line search method is the same as for the geodesic search, except when the search goes longer than $\pi/2$ (we ignore this case because it will rarely happen reasonably close to a minimizer). It follows that an exact line search procedure will generate the same iterates if the objective function has the property that $F(x) = F(\beta x)$ for any β, i.e. F is radially invariant. This property holds for the Rayleigh quotient $R(x)$. Thus we conclude that performing the exact line search (4.26) and projecting the step onto the sphere is equivalent to doing the exact line search on the sphere (4.34). Note that an inexact line search procedure will in general *not* produce the same iterates when the parametrizations of the two search curves are different.

There is a simpler way to do quadratic minimization on a k-dimensional sphere. Consider again the problem of minimizing the Rayleigh quotient over the subspace spanned by two vectors x and p. Let $\vec{\alpha} = (\alpha_1, \alpha_2)$ be a vector of unit length.

$$\min_{\vec{\alpha}} R(\alpha_1 x + \alpha_2 p) = \min_{\vec{\alpha}} \frac{\vec{\alpha}^T Y^T A Y \vec{\alpha}}{\vec{\alpha}^T Y^T Y \vec{\alpha}}, \tag{4.35}$$

where $Y = [x, p]$. $Y^T Y$ is diagonal when x and p are orthogonal. This minimization problem is a generalized eigenvalue problem with the solution

$$\vec{\alpha} = \lambda_{\min}(Y^T A Y, Y^T Y). \tag{4.36}$$

This technique can easily be generalized to k dimensions. In this case, $\vec{\alpha}$ is determined by a generalized eigenvalue problem of dimension k.

4.3.3 Coordinate search

A simple method that is rarely considered for eigenvalue problems is to search along the coordinate axes, i.e., $p = e_j$ for some j. This is called coordinate search or coordinate relaxation. The method was applied to the eigenvalue problem in the 1950s and has been analyzed in [17, Sec.61] and [38]. Convergence can be proven under mild assumptions, but the convergence rate may be poor.

For our purposes, an attractive feature of the method is that each iteration is very inexpensive because taking inner products with e_j is trivial. Inserting $p = e_j$ into (4.26) we get the following equation for the exact step length α:

$$\left(a_{jj} x_j - a_j^T x\right) \alpha^2 + (a_{jj} \mu \rho)\, \alpha + \rho a_j^T x - \mu x_j = 0, \tag{4.37}$$

where $\mu = x^T A x$ and $\rho = x^T x$. A key observation is that the value of μ can easily be updated at each iteration:

$$\mu_{new} = (x + \alpha e_j)^T A (x + \alpha e_j) \tag{4.38}$$

$$= \mu + 2\alpha x^T a_j + \alpha^2 a_{jj}. \tag{4.39}$$

Consequently, no matrix–vector product is required in this algorithm. The most expensive step is to compute the inner product $x^T a_j$, which is $O(n)$.

Exploiting the special properties of the search directions e_j, we obtain the following algorithm:

Algorithm 4.3.3: *Coordinate search.*

Input: x, an initial vector
$\mu = x^T A x$
$\rho = x^T x$
for $k = 1, 2, 3 \ldots$
    Pick a search direction e_j
    $\gamma = x^T a_j$
    Solve for α: $(a_{jj} x_j - \gamma)\alpha^2 + (a_{jj}\rho - \mu)\alpha + (\gamma\rho - \mu x_j) = 0$
    Select the better α
    $\rho = \rho + 2\alpha x_j + \alpha^2$
    $\mu = \mu + 2\alpha\gamma + \alpha^2 a_{jj}$
    $x = x + \alpha e_j$
end

We have dropped the subscript k on the iterates, so x_j refers to the jth component of x. In this algorithm there are two possible roots for α and we need to choose the one that corresponds to a minimizer.

For small n, the roots of the quadratic equation become relatively expensive to compute compared to the other operations. Therefore it has been suggested to use an approximate value of α that is faster to compute [38, 17]. Such an inexact line search method is faster overall if the number of iterations is not much higher than for exact line search.

A remaining question is how to pick the search directions e_j. Not much is known about this, but the convergence proofs rely on cycling through all the indices. Two natural choices are to cycle through the indices from 1 to n, or backwards from n down to 1. In the so-called *Aitken double sweep method* one searches along $e_1, e_2, \ldots, e_n$ and then comes back in the order $e_{n-1}, \ldots, e_1$. It has also been suggested that one search along the axis where the gradient has its largest component. This strategy requires that the gradient be accessible at each iteration.

The work in n iterations of coordinate search, commonly called a sweep, is about the same as one matrix–vector product. This estimate holds also when A is sparse because then the columns a_j are sparse, too, and sparse inner products can be utilized.

Finally we discuss the start-up cost. Before the iteration starts, we need to compute

$x^T x$ and $x^T Ax$ for the initial x. In general this requires one matrix–vector product and two inner products, but often the initial x has some structure we can exploit. One case of particular interest is when $x = d$, the direction of negative curvature determined by the modified Cholesky factorization. (We are jumping ahead here; modified Cholesky is described in Section 5.1.) The modified Cholesky factorization is of the form $A + E = LL^T$, where $E \geq 0$ is diagonal. The vector d is defined by

$$L^T d = \gamma e_s, \tag{4.40}$$

where γ is such that $\|d\| = 1$. It follows that

$$d^T Ad = d^T LL^T d - d^T Ed = \gamma^2 - \sum_i e_i d_i^2. \tag{4.41}$$

We only need to sum over the terms where $e_i \neq 0$, that is, the modified elements. Hence the cost of this calculation is negligible. Since we assume d has already been scaled to have unit norm, we know that $x^T x = d^T d = 1$.

4.3.4 Overrelaxation and SOR

The coordinate search method described in the previous section is known as a *relaxation* method in numerical linear algebra. For linear systems, it is well known that successive overrelaxation (SOR) is usually more effective than exact relaxation. The key feature of overrelaxation is that one takes steps that are larger than the (locally) optimal α. It is natural to ask if overrelaxation is useful for eigenproblems as well. The answer is yes. SOR methods for the symmetric eigenvalue problem were described and analyzed by Ruhe in [67]. He derived the optimal relaxation parameter, ω, when the matrix has property A. He showed the method converges for $0 < \omega < 2$. His numerical experiments indicate that overrelaxation is faster than exact search for many matrices if you choose a good relaxation parameter ω, typically 1.6 to 1.8, even for matrices that do not have property A. However, the gain is modest. Both the analysis and the empirical results were based on the assumption one wants high accuracy and can spend many sweeps to achieve this. We are interested in the low accuracy case where only few sweeps (or even less) are performed. We are unaware of any analysis of this case, but it is evident that in the first few (minor) iterations the best Rayleigh quotient is achieved with exact search (relaxation). Therefore we decided not to consider SOR-type methods any further.

4.3.5 Steepest descent

A well known optimization algorithm is the method of steepest descent, in which the search direction p is chosen to be in the direction where the function decreases the most. This direction is the negative gradient. For the Rayleigh quotient $R(x)$, we have that

$$\nabla R(x) = \frac{2}{x^T x}\left(Ax - \frac{x^T Ax}{x^T x}x\right) = \frac{2}{x^T x}\left(Ax - R(x)x\right). \tag{4.42}$$

Thus we define our search direction $p = Ax - R(x)x$. Computing p requires one matrix–vector product and one inner product. Note that no matter how we choose α_k, the iterates x_k will lie in a k-dimensional Krylov space generated by A.

An exact line search requires the solution of

$$\min_{\alpha} \; R(x + \alpha p). \tag{4.43}$$

From (4.42) we see that $p^T x = 0$. It follows that

$$p^T Ax = p^T(Ax - R(x)x) = p^T p. \tag{4.44}$$

Applying these two formulae in (4.26), we obtain

$$\left(p^T p\right)^2 \alpha^2 - \left(p^T Ap\, x^T x - p^T p\, x^T Ax\right)\alpha - p^T p\, x^T x = 0, \tag{4.45}$$

and dividing through by $p^T p$ and $x^T x$ yields

$$\frac{p^T p}{x^T x}\,\alpha^2 + \left(R(x) - R(p)\right)\alpha - 1 = 0. \tag{4.46}$$

Solving this quadratic equation gives the exact line search parameter α. One can show that the smallest root should be chosen when one wants to minimize the Rayleigh quotient. Convergence for other values of α (inexact line search methods) on this problem has been studied in [17, Sec.74]. A special case is $\alpha = 1/R(x)$, which gives the power method.

Convergence

The asymptotic convergence properties of steepest descent are well known and were described by Kantorovich [39, Ch.XV]. We quote a main result for a matrix A with eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ and corresponding eigenvectors $v_1, \ldots, v_n$. Let x_0 be an initial vector, and let x_k be the kth iterate in the steepest descent method (with exact line search).

Theorem 10 *If x_0 is not orthogonal to the eigenspace corresponding to λ_1, then $R(x_k) \to \lambda_1$ and $x_k \to v$, where v is an eigenvector of λ_1. Furthermore, the convergence rate is such that*

$$R(x_k) - \lambda_1 \leq \left(\frac{a-b}{a+b}\right)^2 (1+\alpha_k)(R(x_{k-1}) - \lambda_1), \quad k = 1, 2, \ldots$$
$$\textit{where } a = \lambda_n - \lambda_1,\ b = \lambda_2 - \lambda_1,$$

and $\{\alpha_k\}$ is a monotonically decreasing sequence that tends to zero.

It has later been shown [64] that the convergence rate of $R(x_k)$ to λ_1 is indeed that of a geometric progression.

We are not only interested in the convergence of the eigenvalue approximation, but also that of the eigenvector. For simplicity, assume that λ_1 is simple, such that v_1 is unique (up to a scale factor). Let ϕ_k be the angle between x_k and v_1. Knyazev and Skorokhodov [42] proved the following result.

Theorem 11 *If $\tan \phi_0^2 < \infty$, then*

$$\tan \phi_k^2 \leq \left(1 - \frac{b}{a}\right)^{2k} \tan \phi_0^2,$$
$$\textit{where } a = \lambda_n - \lambda_1,\ b = \lambda_2 - \lambda_1.$$

This could be a useful result with respect to finding a direction of negative curvature because if $\lambda_1 < 0$, there exists an angle δ such that any vector d with angle $(d, v_1) < \delta$ is a direction of negative curvature.

4.3.6 Conjugate gradients

There are several algorithms called "conjugate gradients" (one linear and various nonlinear versions) and this sometimes causes confusion. Also, for the eigenvalue problem there is a close relation to the Lanczos algorithm. The excellent paper by Edelman and Smith [16] presents the relations between these methods. We summarize their findings here using their terminology. Edelman and Smith group the CG methods into three categories:

LCG: Linear Conjugate Gradients. A standard method for solving $Ax = b$ where A is symmetric positive definite. See [29, Ch.10] for a description. The method may also be viewed as a minimization algorithm for the quadratic function $F(x) = \frac{1}{2}x^TAx - b^Tx$. The search directions p_k are all A–orthogonal (or conjugate). No explicit line search is required because there is a closed form expression for the optimal step length.

NCG: Nonlinear Conjugate Gradients. A well known algorithm for unconstrained minimization, see [27, 20, 46]. Two consecutive search directions are conjugate with respect to the current Hessian or some approximation thereof (different inner products give rise to different versions). An inexact line search method is usually used.

ICG: Idealized Conjugate Gradients. A fictional algorithm introduced in [16] for purposes of exposition. ICG is defined as an algorithm that generates iterates x_k that satisfy $F(x_k) = \min_{x\in\mathcal{K}_k} F(x)$, where $\mathcal{K}_k$ is a k-dimensional Krylov space. The search directions $p_k = x_{k+1} - x_k$ are not necessarily conjugate with respect to any fixed inner product, so the name is slightly misleading. "Optimal Krylov Method" might have been a better name.

While the LCG and ICG algorithms are uniquely defined (given the initial data), there are several versions of NCG. The most popular ones are the Fletcher-Reeves and the Polak-Ribière algorithms [13]. For the Rayleigh quotient the Hessian can be computed explicitly, so one can then make the directions conjugate through the (free space) Hessian. A more sophisticated approach is to take the unit length constraint into account and compute the so-called constrained Hessian. One can then impose conjugacy through this constrained Hessian, which is nonsingular.

The three CG algorithms are all equivalent when the objective is a convex quadratic function (i.e., the Hessian is constant and positive definite). They all find the exact minimizer in at most n steps. No such property holds in the nonlinear case. While LCG and

NCG are ICG methods in the special convex quadratic case, no NCG algorithm is an ICG method for general nonlinear functions.

When we apply CG-like methods to the Rayleigh quotient $R(x)$ and compare to Lanczos (which is described in Section 5.2.1), we find that:

- The Lanczos algorithm (including computation of the smallest eigenvalue of the tridiagonal matrix) is an ICG algorithm for the Rayleigh quotient. Lanczos will find the exact eigenvalue in at most n steps.
- An NCG algorithm for the Rayleigh quotient will compute iterates in a Krylov space, but the iterates are not necessarily optimal in their subspace. An NCG algorithm is not an ICG algorithm. Such an algorithm is therefore not equivalent to Lanczos.
- There is no LCG algorithm for the Rayleigh quotient, but there is a connection between LCG applied to the quadratic form x^TAx and the Lanczos algorithm. The residual vectors computed by LCG are (up to normalization) Lanczos vectors that tridiagonalize the matrix A. The link is through the tridiagonalization and not the eigenvalue computation.

It is often claimed that CG is equivalent to Lanczos. This is only correct if the interpretation is as in the last item. We emphasize that there is no CG-type method applied to the Rayleigh quotient that is equivalent to Lanczos (apart from ICG, which is not a practical algorithm).

A detailed historical overview of CG methods for eigen-like problems is given in [16].

4.3.7 Penalty function methods for x^TAx

Three common penalty functions applied to (4.20) give

$$\begin{aligned} P_1(x) &= x^TAx + \rho|x^Tx - 1| && (4.47)\\ P_2(x) &= x^TAx + \rho(x^Tx - 1)^2 && (4.48)\\ L_A(x,\mu) &= x^TAx - \mu(x^Tx - 1) + \rho(x^Tx - 1)^2. && (4.49) \end{aligned}$$

The l_1 penalty function P_1 is exact for sufficiently large ρ, but it is nondifferentiable and hence not suited for smooth optimization algorithms. We will not consider it any further. The quadratic penalty function P_2 is smooth but not exact for a fixed, finite ρ. For it to be exact, we require $\rho \to \infty$. However, for our purposes a moderately large ρ should suffice. It can be shown that

$$\|x^* - v\| = O(1/\rho), \tag{4.50}$$

where x^* is the minimizer of (4.48) and v is the solution of (4.20), i.e. the sought eigenvector. An error of $O(1/\rho)$ in x leads to an error of $O(1/\rho^2)$ in the Rayleigh quotient.

The augmented Lagrangian L_A is an exact penalty function for suffiently large ρ. We can eliminate the multiplier μ by expanding it as a function of x. At an optimum the gradient of the Lagrangian is zero, hence

$$2Ax - 2\mu x = 0 \;\Rightarrow\; \mu = \frac{x^T Ax}{x^T x}. \tag{4.51}$$

This is not surprising because the Rayleigh quotient is an eigenvalue when x is an eigenvector. If we treat μ as a function of x and substitute $\mu = R(x)$ into (4.49) we get

$$P_3(x) = \frac{x^T Ax}{x^T x} + \rho(x^T x - 1)^2. \tag{4.52}$$

This is precisely the quadratic penalty function for the Rayleigh quotient. A minimizer of $P_3(x)$ is an eigenvector for any value of ρ. The penalty parameter only affects the scaling of x^*. The standard approach is to minimize just the Rayleigh quotient, but adding the penalty term makes the Hessian nonsingular at the solution, so this may have some virtue.

The unit length constraint on the eigenvector can be formulated in different ways. For example, instead of $x^T x = 1$ we could just as well write $\sqrt{x^T x} - 1 = 0$ or $\sqrt{x^T x - 1} = 0$. The augmented Lagrangian penalty function for the latter formulation is

$$P_4(x) = x^T Ax - \mu\sqrt{x^T x - 1} + \rho(x^T x - 1). \tag{4.53}$$

Solving $\nabla_x L(x) = 0$, where $L(x) = x^T Ax - \mu(x^T x - 1)$ is the Lagrangian, yields

$$\mu = 2\sqrt{x^T x - 1}\,\frac{x^T Ax}{x^T x}. \tag{4.54}$$

Eliminating μ, we get the function

$$P_4(x) = 2\frac{x^TAx}{x^Tx} - x^TAx + \rho(x^Tx - 1). \tag{4.55}$$

Observe that when $x^Tx = 1$, $P_4(x) = x^TAx$, as one would expect.

Lanczos is optimal

Suppose we apply the method of steepest descent to the penalty functions $P_i(x)$, $i > 1$. The gradients are

$$\nabla P_2(x) = 2Ax + 4\rho(x^Tx - 1)x, \tag{4.56}$$

$$\nabla P_3(x) = \frac{2}{x^Tx}\,(Ax - R(x)x) + 4\rho(x^Tx)x, \tag{4.57}$$

$$\nabla P_4(x) = \frac{4}{x^Tx}\,(Ax - R(x)x) - 2Ax + 2\rho x, \tag{4.58}$$

where $R(x)$ is the Rayleigh quotient as usual. An important observation is that the plane spanned by x and $\nabla P_i(x)$, $i > 1$, is $\mathbf{span}\{x, Ax\}$ for all values of ρ. It follows from Section 4.3.2 that the corresponding search curves on the sphere are exactly the same, independent of ρ. But the functions $P_i(x)$ are not independent of $\|x\|$. We conclude that steepest descent on $P_i(x)$ will generate iterates in the same Krylov subspace for all ρ. However, the line searches will in general be different, so the resulting iterates will then also be different.

Both steepest descent and nonlinear CG generate iterates in the same Krylov space as Lanczos, when starting with the same initial vector. Since Lanczos is known to be optimal in the Krylov space (ICG), we know that steepest descent and NCG can never do better than Lanczos in the same number of iterations, no matter how we tune the penalty parameter. Any implementation of steepest descent or NCG on the Rayleigh quotient will require (at least) one matrix–vector product per iteration. Thus the cost per iteration is comparable to that of Lanczos. Consequently, methods based on unconstrained minimization of penalty functions appear to be inferior to Lanczos. The only obvious advantage they have is that you automatically obtain the desired vector, while in Lanczos one has to store the Lanczos vectors or else regenerate them.

4.3.8 Gradient-based methods based on other formulations

Algorithms based on gradient or conjugate-gradient methods may have other objective functions than the ones we have described. One such algorithm was suggested by Blum and Rodrigue [4]. They sought stationary points for the functional

$$\frac{1}{2}\|Ax - R(x)x\|^2. \tag{4.59}$$

Any stationary point is an eigenvector. The stationary points of this functional are not isolated; therefore a gradient method to find so-called "C-stationary" points was developed in [4].

There are surely other optimization problems that are equivalent to the linear eigenproblem. We chose to focus on the ones that have simple and smooth formulations and/or are closely related to penalty methods.

Chapter 5

Computing directions of negative curvature

The focus of this chapter is on algorithms for computing directions of negative curvature. Specifically, we wish to develop algorithms that obtain *good* directions with relatively little effort. First we review some well known direct and iterative methods, and then we demonstrate that these techniques can be combined to form efficient hybrid algorithms.

5.1 Direct methods

All algorithms for computing eigenvalues are iterative, but if we are satisfied with finding a direction of negative curvature, direct methods exist. These methods are based on either a Cholesky-like positive definite factorization or an indefinite factorization.

When H is indefinite, it has no Cholesky factorization LL^T because any matrix of the form LL^T is positive semidefinite. Gill and Murray suggested using a *modified* Cholesky factorization [24], later refined in [27]. The main idea is to perform the standard outer-product Cholesky algorithm until the diagonal element of the Schur complement, $\hat{h}_{kk}$, is negative or smaller than ϵ (where ϵ is small and positive). In such cases the diagonal element is replaced by a positive number, typically $\max\left\{|\hat{h}_{kk}|, \epsilon\right\}$, and the normal Cholesky algorithm is resumed. However, this procedure may not be sufficient. Even if $\hat{h}_{kk} > \epsilon$ it may be necessary to increase the diagonal element of the modified factor to ensure that subsequent diagonal elements are adequately bounded. This is achieved by ensuring that all the off-diagonal elements in column k are bounded in magnitude by some constant β.

The Gill-Murray algorithm produces a factorization

$$H + E = LL^T, \tag{5.1}$$

where $E \geq 0$ is diagonal. If H is sufficiently positive definite (not almost singular), $E = 0$, and the algorithm is equivalent to the standard Cholesky algorithm. Let s be the smallest index for which $\hat{h}_{ss} \leq \hat{h}_{kk}$, $k = 1, \ldots, n$. Let e_s be the vector with a one in position s and zeros otherwise. Define d by

$$L^T d = e_s. \tag{5.2}$$

Then d can be proven to be a direction of negative curvature when $\epsilon = 0$ [24, 27]. In practice, d is usually a direction of negative curvature also when ϵ is small but positive. Unfortunately, d can be a very poor direction of negative curvature. In fact, the curvature can be arbitrarily close to zero even when the smallest eigenvalue is bounded away from zero, as was shown in [48].

Moré and Sorensen [48] suggested using an indefinite factorization $H = LBL^T$, where B is block diagonal with block sizes one or two. The most popular such factorizations are the Bunch-Parlett and Bunch-Kaufman factorizations. From this it is possible to derive a direction of negative curvature d that satisfies (4.3). A drawback of this method is that one needs to employ (at least) partial pivoting, which in general would increase the density of the factors. Just as critical is that the sparsity structure would vary at each iteration. Recently, Cheng and Higham [7] described a modified Cholesky factorization based on the bounded Bunch-Kaufman (BKK) method. One might also consider an algorithm based on Aasen's method [1], but no efficient way is known to compute a direction of negative curvature from the tridiagonal matrix generated by Aasen's method.

Forsgren, Gill, and Murray [21] proposed a method based on a *partial* Cholesky factorization. This method also requires partial pivoting but is usually less expensive than the method in [48]. It has been proven that the resulting direction of negative curvature d satisfies (4.3) with $\beta = O(4^{-n_1})$, where n_1 is the number of rows (columns) factored before a nonpositive diagonal pivot was encountered. The problem with this bound is that n_1 can be large (up to n), which gives a β close to zero. Although this bound is usually unduly pessimistic, computational tests show that poor directions may be found.

Recently, Neumaier [53] has proposed a method that is based on a modified Cholesky factorization similar to the Gill-Murray factorization. The main difference is he has devised a more complicated procedure for obtaining the vector d. This scheme is more expensive but produces a better (or equally good) direction of negative curvature. It is shown[1] that d satisfies (4.2), but the direction may still be relatively poor compared to the smallest eigenvector because (4.3) is not shown to hold.

It is our belief that one can never guarantee a good direction of negative curvature unless some type of pivoting is employed. We do not wish to do so for efficiency reasons. A natural question is whether a direct–iterative hybrid method is possible: first do some factorization *without* pivoting to obtain a direction of negative curvature, then improve it by an iterative process. We describe this approach in Section 5.2.7 after we first examine various iterative methods.

5.2 Iterative methods

Recall that the optimal direction of negative curvature is an eigenvector corresponding to the smallest (leftmost) eigenvalue. An obvious approach is therefore to use an iterative algorithm for finding extreme eigenvalues as an algorithm to compute a direction of negative curvature. This strategy is appealing because we can rely on existing methods. As we do not need much accuracy, the iteration process can be stopped much sooner than for the eigenvalue problem. A difficult case when trying to compute a particular eigenvector is when several eigenvalues are clustered closely together. In this case the algorithm may converge to the "wrong" eigenvalue. This is not a concern for us, since if the eigenvalues are very close we do not care which one we find. The curvature will be about the same for all of them. A special case that can cause trouble is when the smallest eigenvalues are clustered around zero, so we may find a positive eigenvalue when a negative one exists. However, even this case is not of concern for us, because finding a direction of negative curvature is only important if there exists a good one.

The two preferred algorithms today for finding extreme eigenvalues are the Lanczos method and the Jacobi-Davidson method. If we have an estimate for the smallest eigenvalue, inverse iteration or Rayleigh quotient iteration may also be used. We have also developed a special version of the Chebyshev algorithm for this problem. In addition, all the iterative

[1]At the time of this thesis, there is some uncertainty about the validity of his proof because a counter-example appears to exist.

methods described in Section 4.3 can be used.

5.2.1 Lanczos

The Lanczos algorithm operates on a Krylov subspace, and computes optimal approximations in those subspaces (more precisely, it is a Rayleigh-Ritz procedure). A key feature of the method is that one quickly obtains good estimates of the extreme eigenvalues.

Some authors distinguish between the Lanczos *process*, which constructs an orthogonal basis for the Krylov subspace $\mathcal{K}_j(A)$, and the Lanczos *algorithm*, which also estimates the eigenvalues of A. When we write just *Lanczos* we will normally mean the latter. We show a simple version of the Lanczos algorithm for a symmetric matrix A below.

Algorithm 5.2.1: *Simple Lanczos.*

Input: A, the matrix, and r_0, an initial vector.
Output: Q_j, the Lanczos basis, and $\lambda(T_j)$,
    estimates of the extreme eigenvalues of A.

$\beta_0 = \|r_0\|$
$q_0 = 0$
for $j = 1, 2, \ldots$
    $q_j = r_{j-1}/\beta_{j-1}$
    $r_j = Aq_j - \beta_{j-1}q_{j-1}$
    $\alpha_j = q_j^T r_j$
    $r_j = r_j - \alpha_j q_j$
    $\beta_j = \|r_j\|$
    Compute selected eigenpairs of $T_j = \text{tridiag}(\beta, \alpha, \beta)$ as desired.
    if satisfied **then** *exit.*
end

This algorithm reduces A to tridiagonal form assuming exact arithmetic and $\beta_j > 0$. After n steps

$$T = Q^T AQ, \tag{5.3}$$

where T is tridiagonal and Q is orthogonal. Normally we perform much fewer than n

iterations, and we have that

$$AQ_j - Q_jT_j = r_je_j^T, \tag{5.4}$$

where $Q_j = [q_1, \dots, q_j]$ is orthogonal and T_j is tridiagonal. The eigenvalues θ_j of the tridiagonal matrix T_j are called the Ritz values. It is inexpensive to compute the Ritz values because usually $j \ll n$. Using QR iteration to compute the p extreme eigenvalues of T_j requires only $O(pj)$ operations. Selected eigenvalues of a tridiagonal matrix can also be computed by bisection or by the divide-and-conquer method of Gu and Eisenstat [32]. The extreme Ritz values approximate the extreme eigenvalues of the original matrix A. It can be shown that

$$\theta_{\min}(T_j) = \min_{y=Q_jz} \frac{y^TAy}{y^Ty}, \tag{5.5}$$

and similarly for $\theta_{\max}(T_j)$. Hence the Lanczos method is optimal in the Krylov subspace (see also Section 4.3.6).

We note that the simple Lanczos algorithm described above requires us to store only three vectors. This is sufficient to produce the desired eigenvalues. If one also wants one or more eigenvectors, one either has to store all the Lanczos vectors or repeat the whole Lanczos process to regenerate them. If only one or a few eigenvectors are sought, these can alternatively be found by inverse iteration (see Section 5.2.3).

The Lanczos vectors are orthogonal in exact arithmetic, but in floating-point arithmetic they can lose orthogonality quite rapidly. Hence, in practice some type of reorthogonalization has to be used. Since full reorthogonalization is very expensive, several types of selective or partial reorthogonalization have been developed [58, 10].

It has long been folklore in numerical analysis circles that the Lanzcos and similar algorithms quickly determine the eigenvectors corresponding to extreme eigenvalues. Quite how quickly is often omitted. Calculating only a few extreme eigenvalues and eigenvectors is usually cheaper than determining all the eigenvectors, but it can typically take about $2\sqrt{n}$ iterations [58, p.259] to compute them to machine precision. While that is low compared to n it is more work than we wish to expend. For negative curvature we require less accuracy, but numerical experiments indicate $O(\sqrt{n})$ iterations may still be necessary (see Section 5.2.8).

A detailed convergence analysis of Lanczos based on Chebyshev polynomials was first done by Kaniel and Paige [55]. A more recent paper is [68].

5.2.2 Jacobi-Davidson

A newer eigenvalue method that has gained popularity lately is the Jacobi-Davidson method by Sleijpen and Van der Vorst [72]. This method combines features from the Davidson method and an old algorithm proposed by Jacobi. One advantage of the method is that it allows for preconditioning. The preconditioner is applied to a linear subproblem, not the eigenvalue problem itself.

We give a very brief sketch of the Jacobi-Davidson method. Suppose we seek the eigenvalue of A closest to μ and the corresponding eigenvector (extreme eigenvalues can be found similarly). In each iteration of Jacobi-Davidson, we approximate the desired eigenvector in a subspace. Let u_k be the Ritz vector after k iterations. We then seek to update u_k with a correction w in $u_k^\perp$, the orthogonal complement of u_k. This strategy is called JOCC (Jacobi's orthogonal complement computation). The restriction of A with respect to $u_k^\perp$ is

$$B = (I - u_k u_k^T) A (I - u_k u_k^T), \ \|u_k\| = 1. \tag{5.6}$$

Then we solve

$$(B - \mu I) w = -r, \tag{5.7}$$

where $r = Au_k - \mu u_k$ is the kth residual. w is used to update u_k, that is, $u_{k+1} = u_k + \alpha_k w$. A key point in Jacobi-Davidson is that the equation for w need only be solved approximately, typically by using only a few iterations of a Krylov subspace method. Note that the JOCC is just a linear system, so we can use a preconditioner if we wish.

It can be shown [72] that the Jacobi-Davidson converges cubically for symmetric matrices in a region around the solution. However, far from the solution the method can be quite slow. One possible remedy is to start with another method and switch to Jacobi-Davidson when a reasonably good approximation has been found [30]. In our application we expect to have a fairly good initial guess, so such a strategy should not be necessary.

The Jacobi-Davidson method can be interpreted as an inexact Newton scheme [71].

5.2.3 Inverse iteration and RQI

The preferred methods to compute the eigenvalue closest to a given number μ are inverse iteration and Rayleigh-quotient iteration (RQI). These are both based on the power method. The power method finds the largest (in magnitude) eigenvalue of a matrix, so the idea is to apply it to the matrix $(A - \mu I)^{-1}$.

Input: x_0, an initial vector
and μ, an approximation to the desired eigenvalue

for $k = 1, 2, \ldots$
  Solve $(A - \mu I)y_k = x_{k-1}$
  $x_k = y_k / \|y_k\|$
end

In plain inverse iteration the shift μ is constant throughout the process. In RQI μ is set to the Rayleigh quotient of x_k at each iteration. As the process converges, this will be a better approximation to the eigenvalue of interest than the initial μ.

The convergence properties of inverse iteration [59, 58, 36] and RQI [54, 57] have been well studied. Suppose we seek λ_1, and assume that λ_1 is the eigenvalue closest to μ. As $k \to \infty$, $x_k \to v_1$ linearly with convergence rate at worst $|\mu - \lambda_1|/|\mu - \lambda_2|$, where v_1 is an eigenvector of λ_1. (We do not consider the case where λ_1 is a multiple eigenvalue, since that is much more complicated.) The asymptotic convergence rate is of little importance when low accuracy is required. Ipsen [36] has shown that a single iteration is usually sufficient to obtain an approximate eigenvector with the same order of accuracy as μ approximates λ_1.

It has been shown that RQI converges asymptotically cubically to an eigenvector [54, 58]. One difficulty with RQI is that the method is more sensitive to the initial vector, that is, it converges fast but may converge to a different eigenvector than desired. This is a concern in eigenvalue computations where the eigenvector is unknown, but is less of a concern to us when we already have a good approximate direction.

In inverse iteration, the linear system can be solved either by direct factorization (LU or Cholesky) or by an iterative procedure. Since the shift is fixed, the same matrix factors can be reused througout the procedure. Therefore a direct method is typically employed. With RQI the shift changes at every iteration, so either a new factorization is required, or more realistically, an iterative solver has to be used for the linear system. One might think an iterative solver is inappropriate since the system is very ill-conditioned when the shift is close to an eigenvalue, but in fact the vector y_k tends to blow up in the desired direction. See [59, 58] for a discussion of this.

We note that the matrix $A-\mu I$ is symmetric but may be indefinite. Therefore we cannot use CG but need a method like MINRES, SYMMLQ, or QMR. The overall algorithm is an inner–outer algorithm, and as usual the question is how accurate do we need to solve the inner problem. This has been studied in [76] and more recently in [22]. The answer appears to be that a relatively loose tolerance in the inner problem is sufficient. Typically, only two or three iterations are performed in each inner iteration.

RQI is not a Krylov subspace method. It is related to Newton's method [59]. Inverse iteration is a Krylov method with respect to $(A-\mu I)^{-1}$ if the linear system is solved exactly.

In the negative curvature problem, we want to approximate the most negative eigenvector but usually we do not have a good estimate of the eigenvalue a priori. One possible strategy is to fix μ at say $\mu = -1$. If $\lambda_{\min} > -1$ the algorithm will converge to $\lambda_{\min}$. Otherwise, $\lambda_{\min} < -1$ and we may find a different eigenvalue λ_j, where $\lambda_{\min} < -1 < \lambda_j$. This is acceptable if λ_j is negative and bounded away from zero, because we then obtain a reasonably good direction of negative curvature. The bad case is when we converge to a value that is around zero or positive. This has to be considered a failure, so this simple approach is not quite satisfactory.

In an optimization algorithm, if we have already computed a descent direction p we can use this to choose a better shift for inverse iteration. We would like to determine a direction of negative curvature, d, that gives better decrease in the objective function than the descent direction. A possible strategy is therefore to choose the shift for inverse iteration, μ, depending on the predicted function decrease g^Tp. Unfortunately, this does not easily produce an estimate for μ because that requires an estimate for the function decrease in the quadratic model in d, and an appropriate scaling of d is usually not known.

It has been suggested that inverse iteration be combined with RQI [76]. The argument is that one should start with inverse iteration because the method is quite robust when the initial vector is a poor approximation, and later switch to RQI to take advantage of the fast local converge of RQI. This conclusion does not apply directly to the problem of finding good directions of negative curvature. In a purely iterative scheme we have no estimate of the eigenvector at first and should use inverse iteration initially. We expect the vectors obtained from inverse iteration to be sufficiently good for our purpose, so the RQI phase is not necessary. On the other hand, if we have a vector at the beginning that is close to a desired eigenvector in some measure, then it might be better to do only RQI.

Although we believe inverse iteration and RQI may be suitable algorithms for finding directions of negative curvature in some situations, we have chosen to focus on other methods.

Finally, we remark that the problem of finding the smallest eigenpair is a special case. Earlier we stated that an indefinite solver has to be applied to the matrix $A - \mu I$. This is not true when the smallest eigenvalue is desired. Suppose we apply CG to the system $A - \mu I$. If $\mu > \lambda_{\min}$, then $A - \mu I$ is indefinite and CG may not work. If a CG search direction has negative or zero curvature, we say that CG becomes *unreliable* since the standard convergence theory no longer holds. ("unreliable" is not a standard term; some people say CG *breaks down* in this case, but that is not quite accurate because the iteration process may sometimes continue.) It should be mentioned that CG occasionally works on indefinite systems; for example, it will converge to a solution if all iterates lie in a subspace where the quadratic form is convex. Interestingly, if CG becomes unreliable, it will produce a direction of negative curvature of $A - \mu I$. It follows that $\lambda_{\min} < \mu$ and the direction of negative curvature is a vector that can be used as the next iterate. For more details on CG applied to indefinite systems, see Section 5.3.1 and Algorithm 5.2.6. We have not analyzed how such a procedure compares to inverse iteration (or RQI) with a standard indefinite solver.

5.2.4 Chebyshev iteration

We derive our Chebyshev algorithm from looking at polynomials over a Krylov subspace. Any Krylov subspace method starting with x_0 can be expressed as

$$x_k = P_k(A)x_0, \tag{5.8}$$

where P_k is a polynomial of degree k and $P_0 = 1$.

We seek a polynomial P_k that gives good negative curvature, as measured by the Rayleigh quotient. Let $\lambda_1 \leq \ldots \leq \lambda_n$ be the eigenvalues of A. We are interested in the case where A is indefinite, so we can assume $\lambda_1 < 0$ and $\lambda_n > 0$. Let $v_1, \ldots, v_n$ be the corresponding unit eigenvectors. We normalize x_0 to have unit length, and define $\phi_i = x_0^T v_i$.

Then $x_0 = \sum_i \phi_i v_i$, and

$$
\begin{aligned}
x_k^T x_k &= x_0^T P_k(A)^2 x_0 && (5.9)\\
&= \sum_{i=1}^{n} \phi_i^2 v_i^T P_k(A)^2 v_i && (5.10)\\
&= \sum_{i=1}^{n} \phi_i^2 P_k(\lambda_i)^2. && (5.11)
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
x_k^T A x_k &= x_0^T P_k(A) A P_k(A) x_0 && (5.12)\\
&= \sum_{i=1}^{n} \phi_i^2 v_i^T P_k(A)^2 A v_i && (5.13)\\
&= \sum_{i=1}^{n} \phi_i^2 P_k(\lambda_i)^2 \lambda_i, && (5.14)
\end{aligned}
$$

and we get

$$
\begin{aligned}
R(x_k) &= \frac{x_k^T A x_k}{x_k^T x_k} && (5.15)\\
&= \frac{\sum_{\lambda_i < 0} \phi_i^2 P_k(\lambda_i)^2 \lambda_i}{\sum_{i=1}^{n} \phi_i^2 P_k(\lambda_i)^2} + \frac{\sum_{\lambda_i \geq 0} \phi_i^2 P_k(\lambda_i)^2 \lambda_i}{\sum_{i=1}^{n} \phi_i^2 P_k(\lambda_i)^2} && (5.16)
\end{aligned}
$$

The first aggregate term contains all the negative terms and the second contains the positive terms. To get good negative curvature, we seek a polynomial that is small on the positive eigenvalues but blows up for $\lambda < 0$. The obvious candidate is a Chebyshev polynomial.

The Chebyshev polynomials $C_k(z)$ can be defined by

$$
\begin{aligned}
C_k(z) &= \cos\left(k \cos^{-1}(z)\right), \quad -1 \leq z \leq 1, && (5.17)\\
&= \cosh\left(k \cosh^{-1}(z)\right), \quad |z| > 1. && (5.18)
\end{aligned}
$$

but a more practical definition is the recursive formula

$$
\begin{aligned}
C_0 = 1, \quad C_1 = z, && (5.19)\\
C_{k+1}(z) = 2zC_k(z) - C_{k-1}(z). && (5.20)
\end{aligned}
$$

These polynomials are bounded in magnitude by one on $[-1, 1]$, but grow very rapidly outside this interval. In our case, we seek polynomials P_k that are small on $[0, \hat{\rho}]$, where $\hat{\rho}$ is an upper bound for λ_n. Consequently, we define P_k to be the Chebyshev polynomial of degree k on $[0, \hat{\rho}]$, given by

$$P_k(z) = C_k\left(-1 + \frac{2z}{\hat{\rho}}\right). \tag{5.21}$$

The three-term recurrence for P_k is

$$P_{k+1}(z) = \left(-2 + \frac{4}{\hat{\rho}}z\right)P_k(z) - P_{k-1}(z), \tag{5.22}$$

with initial conditions $P_0 = 1$, $P_1(z) = \frac{2}{\hat{\rho}}z - 1$. Applying this to A gives the following *Chebyshev iteration* algorithm:

$$x_1 = \frac{2}{\hat{\rho}}Ax_0 - x_0, \tag{5.23}$$

$$x_{k+1} = \frac{4}{\hat{\rho}}Ax_k - 2x_k - x_{k-1},\ k > 0. \tag{5.24}$$

This formula is very efficient because each step only requires one matrix–vector product and no inner products. Also note we only need to store three vectors.

In our experiments we used $\hat{\rho} = \|A\|_1$ because the one–norm of a matrix is inexpensive to compute, but it might be better to spend more effort to get a tighter bound. This should give faster convergence. We believe a good choice is to run a few iterations of Lanczos, for instance, and use the eigenvalue estimate obtained from that in a subsequent Chebyshev procedure.

Another promising strategy is to use Chebyshev polynomials to accelerate some other iterative method. This is known as the Chebyshev *semi-iterative* method and is described in [79, Ch.5].

5.2.5 Practical use of direction of negative curvature

We wish to analyze and compare the cost of different ways to compute a direction of negative curvature. In practice we rarely perform this procedure for its own sake, but rather as a step within an optimization algorithm. Hence we need to assess what else is done in the algorithm. For example, one should be aware which quantities have been computed and thus are readily available.

The standard approach in line search methods is first to compute a Newton-type descent direction p by solving $Hp = -g$, where $H \approx \nabla^2 F(x)$. If H is determined to be indefinite, a direction of negative curvature is also computed. Typically, a direct solver is used and some factorization of H is computed. This factorization is then available when a direction of negative curvature is sought. For very large problems, iterative solvers are usually preferred. In this case we can extract useful information from the iterative solver. In the next section we show how a direction of negative curvature can be found from the CG algorithm.

5.2.6 Negative curvature in CG

Suppose we attempt to solve $Hp = -g$ using CG when H is indefinite. In such cases, CG may fail. If all iterates lie in a subspace where the quadratic form is convex, CG may find a satisfactory solution and we will not determine whether the matrix was definite or indefinite. However, if one of the CG search directions has negative curvature, then the CG process becomes unreliable (and may even break down). But in this case we have found a direction of negative curvature, though it could be poor. When we use the CG algorithm, we will either solve $Hp = -g$ satisfactorily, or obtain a direction of negative curvature. The algorithm is a straightforward modification of the standard CG algorithm. A minor deficiency in the algorithm below is that it may terminate with a direction of zero curvature. However, it is highly unlikely this will occur in floating-point arithmetic. Furthermore, even a direction of zero curvature may be satisfactory if we later perform some type of iterative improvement to obtain a better direction of negative curvature; see the next section.

Algorithm 5.2.6: *CG for possibly indefinite systems.*

Input: H, g, ϵ

Output: When H is spd: p_k, a solution of $Hp = -g$ with tolerance ϵ,
otherwise: d, a direction of negative curvature.

$p_0 = 0;\ r_0 = -g;\ k = 0$
while $\|r_k\| > \epsilon(\|g\| + \|H\|\|p\|)$
  $k = k + 1$
  if $k = 1$
    $c_1 = r_0$
  else
    $\beta_k = r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2}$
    $c_k = r_{k-1} + \beta_k c_{k-1}$
  end
  if $c_k^T A c_k \leq 0$
    $d = c_k$; *exit*
  end
  $\alpha_k = r_{k-1}^T r_{k-1} / c_k^T A c_k$
  $p_k = p_{k-1} + \alpha_k c_k$
  $r_k = r_{k-1} - \alpha_k A c_k$
end

There is a close relation between CG and Lanczos (Section 5.3.1), so it is possible to recover the Lanczos vectors and the tridiagonal matrix from the CG vectors and coefficients. One can then compute the smallest Ritz value and vector, which are optimal in the Krylov subspace we work in. However, the work and storage requirements then become similar to those of Lanczos, so we do not study that option any further.

5.2.7 A hybrid approach

A fairly natural idea for computing a direction of negative curvature is to combine a direct method and an iterative method. A direct method like modified Cholesky will produce a direction of negative curvature but it may be poor. We can then use this vector as the initial vector in an iterative method. Murray reported in 1993 [50] that even one or two iterations of the steepest descent algorithm could dramatically improve a poor direction of negative curvature when a good one exists. This has encouraged us to study such hybrid methods in more depth. The first stage in a hybrid method does not necessarily have to be a direct method, but could be an iterative method like CG that terminates with a (poor) direction of negative curvature.

Obviously, any iterative method can be used in the refinement step. Since we already have a fairly good initial vector we expect to do only a few iterations. A relative inexpensive iterative method may be sufficient. We examine Lanczos, steepest descent, Chebyshev, and coordinate search.

We expect iterative methods to perform quite well when the initial vector has close to zero curvature. The main reason is that when H has only few negative eigenvalues, a vector d with near-zero curvature is likely to have strong components in the direction of the smallest eigenvectors. We prove this in Section 5.2.9. We know that this property leads to quick convergence for many iterative methods, e.g., Chebyshev iteration and inverse iteration (RQI).

5.2.8 Numerical results

We have tested the methods on a set of randomly generated indefinite matrices. (Some preliminary results were reported in [5].) Within an optimization method it is essential to obtain a relatively good direction of negative curvature whenever a significant direction exists. When the smallest eigenvalue is close to zero it is often the case that a direction of negative curvature need not be computed. Nonetheless we have included this case because it represents a potentially hard case for the algorithms and may occasionally be required. Since many iterative methods work extremely well on matrices whose eigenvalues are clustered, such distributions of the eigenvalues were avoided. We also expect the hard case for iterative methods to be when the negative eigenvalues are small in magnitude and there are also some small but positive eigenvalues.

We tested a set of $n \times n$ random symmetric indefinite matrices with specified spectra. Each test matrix was of the form $H = Q\Lambda Q^T$ with Q a random orthogonal matrix and Λ a diagonal matrix with t negative elements. We used the following distributions for the eigenvalues; the semi-uniform distribution,

$$\lambda_i = \begin{cases} -\alpha i/t, & i = 1, \dots, t \\ (i-t)/(n-t), & i = t+1, \dots, n \end{cases} \tag{5.25}$$

where $0 < \alpha < 1$, and the semi-geometric distribution,

$$\lambda_i = \begin{cases} \beta^{i-1}, & i = 1, \dots, n-t \\ -\beta^{i-1}, & i = n-t+1, \dots, n \end{cases} (), \tag{5.26}$$

where $0 < \beta < 1$. Similar distributions were used in [21].

For each distribution we generated 10 different random matrices corresponding to $t = 1, \dots, 10$. A new Q was generated for each H. We used the matrix dimension $n = 200$ for most tests. This is not very large, but our matrices are dense because a random orthogonal matrix is dense. In fact, the time to set up the problems was longer than the time to solve them. To test very large n one would need to generate matrices with similar properties but for which the Cholesky factors were sparse. A sparse modified Cholesky code would be required.

We tested three different types of initial vector:

1. Random initial vector.

2. Initial vector with negative curvature from modified Cholesky.

3. Initial vector with negative curvature from CG.

For the iterative improvement stage, we compared four of the methods described earlier: coordinate search, steepest descent, Lanczos, and Chebyshev iteration. All the algorithms were implemented in MATLAB. The Lanczos method was a simple implementation with no reorthogonalization. (We also tried complete reorthogonalization and observed no difference because the numbers of iterations are small.) All the methods except coordinate search require one matrix–vector product per iteration. For coordinate search we counted one sweep (n minor iterations) as one iteration because the work involved is comparable to that of a matrix-vector product. For dense matrices, the work per iteration is roughly the same for all the four methods considered here because vector-vector operations are lower order terms and become insignificant. However, for sparse matrices, each iteration of the Chebyshev method is cheaper than those of steepest descent or Lanczos because fewer scalar products and saxpy operations are required.

In the figures we show the ratio of the Rayleigh quotient $d_k^T H d_k / d_k^T d_k$ to $\lambda_{\min}(H)$ as a function of the number of iterations k. This ratio should go to one as $k \to \infty$. For each method, we plotted the minimum and maximum ratio over the ten trials. If the ratio was negative, the curvature was positive and the data points were not included in the graphs. Each method is normally described by a pair of lines that shows the boundary of the closure (envelope) of all the test runs. We chose to present the experiments this way because plotting all the data would clutter the graphs.

In Figure 5.1 we show the semi-uniform distribution with $\alpha = 1$ and $\alpha = 10^{-3}$ starting with a random initial vector. We see that all the methods managed to find a direction of negative curvature in 5 iterations or less with $\alpha = 1$. In the harder case ($\alpha = 10^{-3}$), the number of iterations goes up dramatically. (Note that the scale on the horizontal axes is different.) In the worst case, only Lanczos produces a good direction of negative curvature in a reasonable number of iterations. For coordinate search and steepest descent, only the best case shows up on the graph. This implies that in the worst case, negative curvature was not found in 100 iterations. Chebyshev iteration did not show up even in the best case.

Figure 5.2 shows the same examples starting with the direction from modified Cholesky. In this case, all four methods do well. Two peculiar effects are visible in the rightmost graph. First, we observe that there was a big improvement for coordinate search in the very first iterations. Second, we note that Chebyshev iteration does not always produce a monotone curve. Unlike the other methods, Chebyshev is not a descent method for the Rayleigh quotient.

In Figure 5.3 the initial vector is the one obtained from CG. The graph for $\alpha = 1$ displays the same behavior as in Figure 5.2, while the leftmost graph ($\alpha = 10^{-3}$) looks different. The initial curvature is weaker than that from modified Cholesky, but then the relative improvement after a few iterations is better.

In Figure 5.4 we used the semi-geometric distribution with $\beta = 0.95$. This example is an extremely hard case for any algorithm because the negative eigenvalues are very close to zero (of order -10^{-6}). For smaller values of β the matrix becomes numerically positive semidefinite in floating-point arithmetic, and eventually both modified Cholesky and our iterative methods fail to give a direction of negative curvature.

From these experiments we conclude that when starting with a poor initial vector (e.g. a random vector), our problem is similar to the symmetric eigenvalue problem and we prefer the Lanczos method. But if we already have a direction of negative curvature, any of the iterative methods we have considered will give much better negative curvature after a few iterations. We prefer steepest descent over the Lanczos method because it requires less memory and we obtain the desired vector directly (no need to store or regenerate

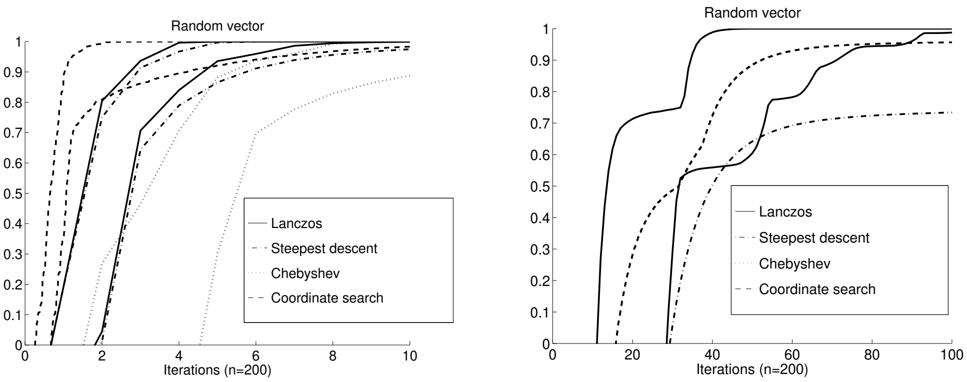

Figure 5.1: Semi-uniform distribution, $\alpha = 1$ (left) and $\alpha = 10^{-3}$ (right). Random initial vector.

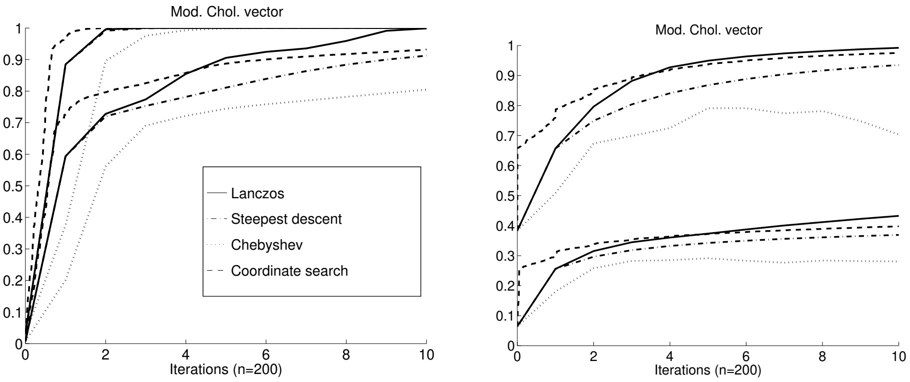

Figure 5.2: Semi-uniform distribution, $\alpha = 1$ (left) and $\alpha = 10^{-3}$ (right). Initial vector from Modified Cholesky.

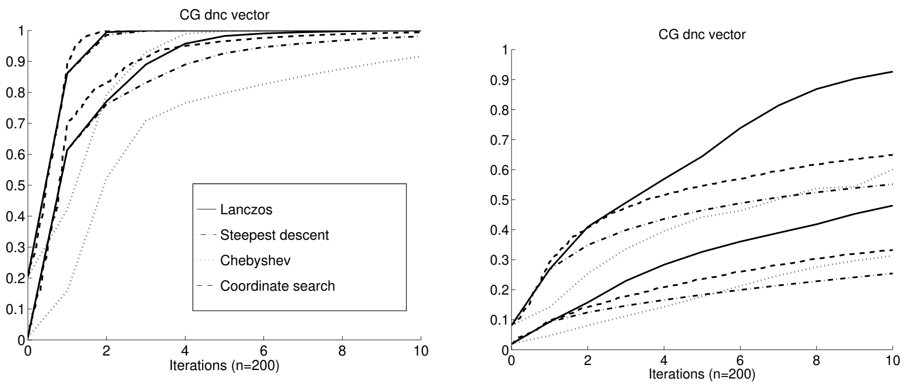

Figure 5.3: Semi-uniform distribution, $\alpha = 1$ (left) and $\alpha = 10^{-3}$ (right). Initial vector from CG.

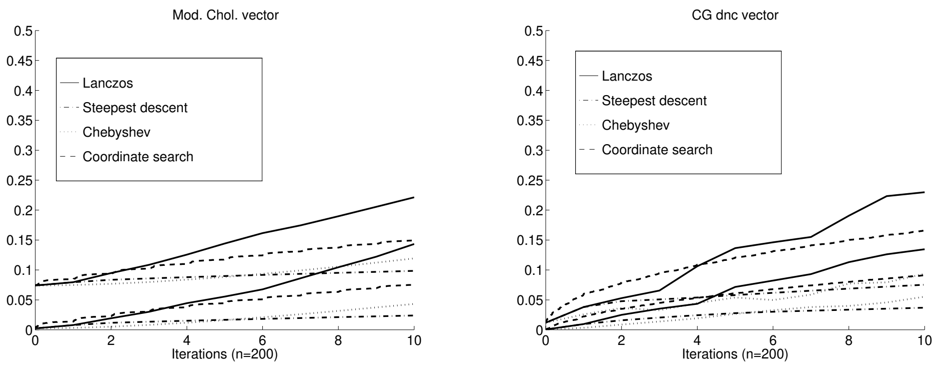

Figure 5.4: Semi-geometric distribution with $\beta = 0.95$. Modified Cholesky initial vector (left) and CG vector (right).

the Lanczos basis). The Chebyshev iteration is similar in cost to steepest descent but the Rayleigh quotient is in general not monotonically decreasing, a desirable property that holds for all the other methods. Overall Chebyshev iteration was the worst of the methods we tried. The coordinate search method performed surprisingly well and was in fact usually better than Lanczos early in the iteration process.

Consequently, coordinate search followed by steepest descent look to be the most attractive algorithms as a refinement step in a hybrid algorithm of the methods considered here. When starting from a random vector, a more powerful method like Lanczos is required. However, in an optimization algorithm we never start from a random direction.

5.2.9 The spectral composition of d

The matrices of interest have only one or a few negative eigenvalues. Assume that we somehow obtain a vector d with negative or near-zero curvature, that is,

$$d^T H d \leq \epsilon, \tag{5.27}$$

where $\epsilon > 0$ is small. Let $\lambda_1 \leq \ldots \leq \lambda_n$ be the eigenvalues of H, and let $v_1, \ldots, v_n$ be the corresponding eigenvectors. In the following, we write the spectral decomposition of d as

$$d = \sum_{i=1}^{n} \phi_i v_i. \tag{5.28}$$

Theorem 12 *Let H be an n by n matrix with only one negative eigenvalue λ_1. Suppose* *all the other eigenvalues are bounded away from zero such that*

$$\lambda_i \geq \theta|\lambda_1|, \; i > 1, \tag{5.29}$$

for some $\theta > 0$ independent of n. If

$$\frac{d^T H d}{d^T d} \leq \epsilon, \tag{5.30}$$

then

$$E\left[\left(\frac{\phi_i}{\phi_1}\right)^2\right] \to 0 \tag{5.31}$$

for $i > 1$ as $n \to \infty$, where E denotes the expectation.

Proof: Assume without loss of generality that $\|d\| = 1$. From (5.30) we have

$$\begin{aligned} \sum_{i=1}^{n} \phi_i^2 \lambda_i &\leq \epsilon \\ \sum_{i=2}^{n} \left(\frac{\phi_i}{\phi_1}\right)^2 \frac{\lambda_i}{|\lambda_1|} &\leq 1 + \frac{\epsilon}{|\lambda_1|\phi_1^2}. \end{aligned}$$

Exploiting (5.29), we get

$$\begin{aligned} \theta \sum_{i=2}^{n} \left(\frac{\phi_i}{\phi_1}\right)^2 &\leq 1 + \frac{\epsilon}{|\lambda_1|\phi_1^2} \\ \frac{1}{n-1} \sum_{i=2}^{n} \left(\frac{\phi_i}{\phi_1}\right)^2 &\leq \frac{1}{\theta(n-1)} \left(1 + \frac{\epsilon}{|\lambda_1|\phi_1^2}\right) \\ E\left[\left(\frac{\phi_i}{\phi_1}\right)^2\right] &\leq \frac{1}{\theta(n-1)} \left(1 + \frac{\epsilon}{|\lambda_1|\phi_1^2}\right) \\ &\to 0 \text{ as } n \to \infty \end{aligned}$$

□

A corollary of this theorem is that $|\phi_i/\phi_1| \to 0$ for $i > 1$ with probability one as $n \to \infty$.

Note that Theorem 12 also holds under certain weaker conditions. For instance, we can allow some eigenvalues to be arbitrarily close to zero as long as there is a finite number

λ_1	mean($\lvert\phi_1\rvert$)	std($\lvert\phi_1\rvert$)
–1	0.54	0.03
–0.1	0.81	0.03
–0.01	0.94	0.02
–0.001	0.995	0.005

Table 5.1: Distribution of $|\phi_1|$ for the modified Cholesky direction from matrices of order $n = 100$ with one negative eigenvalue

of them. Various extensions of the theorem hold when there is more than one negative eigenvalue.

The theorem above tells us that, under the stated assumptions, any direction of negative curvature will have a strong component in the direction of the eigenvector of the negative eigenvalue when n is large. Another observation that can be made is that when $\lambda_{\min}$ is small in magnitude, any direction of negative curvature must have a strong component in the direction of an eigenvector of $\lambda_{\min}$. We have observed this fact empirically using random test matrices with only one negative eigenvalue (semi-uniform distribution from Section 5.2.8). See Table 5.1. The positive eigenvalues were spread out between 0 and 1.

5.3 A modified Newton algorithm

In an optimization algorithm, computing a direction of negative curvature is only a small part of the overall algorithm. It is important to look at the whole approach. In this section we discuss a modified Newton method that relies on computing descent pairs (s, d) where s is a direction of descent and d is a direction of negative curvature. If a good descent direction is found, we do not need to compute a direction of negative curvature. When the descent direction is poor, one should take advantage of the calculations done so far when a direction of negative curvature is computed. Otherwise, a direction of negative curvature is not required.

We are interested in the case where H is large enough to require iterative methods. In *truncated* Newton algorithms, the direction p is found approximately by solving $Hp = -g$ iteratively. When H is determined to be indefinite, we wish to find a direction of negative curvature.

5.3.1 Lanczos and CG

After k iterations of the Lanczos method applied to the matrix A we have

$$AV_k = V_k T_k + \beta_k v_{k+1} e_k^T, \tag{5.32}$$

where $V_k = [v_1 \dots v_k]$ is n by k and orthogonal, and T_k is tridiagonal. The Lanczos algorithm computes eigenvalues, but with minor modifications it can also be used as an iterative method for linear systems.

Consider the general linear system $Ax = b$. Define the residual r_k to be $b - Ax_k$. Lanczos is a Krylov subspace method, so we can write

$$x_k = x_0 + V_k y_k, \tag{5.33}$$

where y_k is undefined for now. To find the best possible approximation to the true solution at every iteration, we impose the Galerkin condition

$$V_k^T r_k = 0, \tag{5.34}$$

that is, the current residual be orthogonal to the current Krylov subspace. We have

$$\begin{aligned} r_k &= b - Ax_0 - AV_k y_k && (5.35)\\ &= V_k \beta_0 e_1 - \left(V_k T_k + \beta_k v_{k+1} e_k^T\right) y_k && (5.36)\\ &= V_k \left(\beta_0 e_1 - T_k y_k\right) - \beta_k \eta_k v_{k+1}, && (5.37) \end{aligned}$$

where $\eta_k \equiv e_k^T y_k$. Choosing y_k to satisfy

$$T_k y_k = \beta_0 e_1 \tag{5.38}$$

gives $r_k = -\beta_k \eta_k v_{k+1}$, and then the Galerkin condition $V_k^T r_k = 0$ holds if v_{k+1} is orthogonal to V_k, or if $\eta_k = e_k^T y_k \approx 0$. In exact arithmetic v_{k+1} is truly orthogonal to V_k, but this may not hold in practice. Equation (5.38) defines a tridiagonal system for y_k of order k, which is fast to solve. The corresponding x_k is given by (5.33).

If A is symmetric and positive definite, T_k also has those properties. Consequently, the tridiagonal system can be solved by Cholesky factorization. Instead of waiting until the end to solve the tridiagonal system, one can incrementally update the current solution x_k. The resulting algorithm is equivalent to the conjugate gradients (CG) method. Lanczos and CG are intimately related [56, 29]. Let R_k be the CG residual matrix after k iterations. From the CG algorithm (5.2.6) we have $c_j = r_{j-1} + \beta_j c_{j-1}$, and it follows immediately that

$$R_k = C_k B_k, \tag{5.39}$$

where C_k is the matrix of CG search directions and B_k is upper unit bidiagonal with (β_j) on the superdiagonal. One can further show [29, Sec. 10.2.6] that the CG residuals and the Lanczos vectors are identical up to a constant factor. More precisely,

$$V_k = R_k \Delta_k^{-1}, \tag{5.40}$$

where Δ_k is diagonal with diagonal elements

$$\Delta_{ii} = \pm \|r_i\|_2. \tag{5.41}$$

Note that the Lanczos method for linear systems works for indefinite matrices, while CG in general does not. In nonlinear optimization we seldom know in advance whether the Hessian is positive definite or indefinite. It has been suggested [31, 45] to start with CG and switch to Lanczos when necessary. This strategy is possible because of the close relationship between the CG search directions and the Lanczos vectors.

An alternative is to start with the CG algorithm and, if a direction of negative curvature is found, then switch to Lanczos but restart using the direction of negative curvature. More precisely, suppose that we apply CG to a matrix H that is indefinite (but presumably has few negative eigenvalues). As long as we stay in a subspace where the Hessian is positive definite, CG works fine. Possibly the CG solver finds an adequate approximation to the solution, p, for the descent direction without detecting that H is indefinite. On the other hand, if CG generates a search direction c_k such that $c_k^T H c_k \leq 0$, then CG becomes unreliable but a direction of negative curvature is obtained. Here c_k is the search direction within CG; the search direction for the optimization algorithm would be $p_k = \sum_{i=1}^{k-1} \alpha_i c_i$. Note that p_k has no component along c_k, which has negative curvature.

One advantage of starting with CG instead of Lanczos is that we do not need to store or regenerate the Lanczos vectors in order to construct a Ritz vector (eigenvector). The drawback is that we usually obtain a direction of poorer negative curvature. Observe that c_k lies in the same k-dimensional Krylov subspace in which the Ritz vector from Lanczos is optimal. Hence the Ritz vector will always have stronger negative curvature than the vector obtained from the CG method. However, in practice the difference in quality appears to be rather small (typically less than a factor two in the Rayleigh quotient), so both vectors can be a useful starting point for further iterative improvement.

We show that the number of iterations needed to find a direction of negative curvature is exactly the same with CG as with Lanczos.

Lemma 13 *Consider the Lanczos algorithm (Table 5.2.1) and the CG algorithm (Table 5.2.6) applied to an indefinite matrix H with the same initial vector. If the kth iteration is the first for which the smallest Ritz value in Lanczos becomes nonpositive, then this is also the first iteration for which $c_k^T H c_k \le 0$ in CG, and vice versa.*

Proof: The smallest Ritz value in a subspace is less than or equal to $c_k^T H c_k / (c_k^T c_k)$ for any c_k in that subspace. This proves one direction of the theorem. Let k be the smallest index such that the Ritz value $\lambda_{\min}(T_k) \le 0$, where T_k is the tridiagonal matrix generated by Lanczos. Then T_{k-1} is positive definite, while T_k is not positive definite. Consequently, CG will work normally for the first $k-1$ iterations, but will become "unreliable" in iteration k with $c_k^T H c_k \le 0$. $\square$

Another way to obtain a direction of negative curvature from CG/Lanczos was proposed by Nash [52]. He suggested using the spectral decomposition of the bottom 2×2 block of a factorization of the tridiagonal matrix T_k. This can be viewed as a compromise between using the whole of T_k and just the last search direction.

5.3.2 Modified Lanczos and truncated Newton

In a Newton-type method we minimize a local quadratic model $Q(p) = \frac{1}{2} p^T H p + g^T p$. When H is positive definite, the minimizer can be obtained by solving the linear system $Hp = -g$. Note that when H is indefinite, such a p is of little interest because it is not a minimizer of $Q(p)$. Furthermore, $Q(p)$ is unbounded below, so we would like to find a direction of negative curvature for H. Let p_k be the projection of $p = -H^{-1} g$ onto the Krylov subspace $\mathcal{K}_k$, i.e., the current iterate after k iterations of Lanczos/CG. If T_k is positive definite then p_k is a descent direction with positive curvature. But if T_k is indefinite, then p_k may not be a descent direction. Fortunately, in this case it is simple to find a direction of negative curvature. Let $\lambda_1 < 0$ be the smallest eigenvalue of T_k and u the corresponding eigenvector. Then u is an optimal direction of negative curvature for T_k. The corresponding direction of negative curvature for H is $V_k u$.

One remaining question is how to obtain a descent direction when T_k is indefinite. A simple approach is to let $p = p_{j-1}$, where j is the smallest index such that T_j is indefinite. However, we may wish to continue the iterative procedure in order to obtain a better descent direction and direction of negative curvature from a higher-dimensional subspace. Assume we perform the Lanczos process k steps, where $k > j$ (and j is defined as above).

To obtain a descent direction, we need to modify T_k to become sufficiently positive definite. This can be done in many ways, but two approaches are particularly natural. The first is to apply a modified Cholesky algorithm to T_k. Because T_k is tridiagonal, one can devise a special algorithm for this case. This was first done by Nash [52]. The other strategy is to shift the eigenvalues of T_k by a positive scalar θ, i.e. let

$$T_k(\theta) = T_k + \theta I, \tag{5.42}$$

where θ is 0, if T_k is positive definite, and $\theta > -\lambda_1$ otherwise. The latter case is the one of interest. We wish to ensure that $T_k(\theta)$ is not ill-conditioned, so the shifted eigenvalues should be kept away from the origin. We suggest using

$$\theta = \max\left\{0, -2\lambda_1(T_k), -\lambda_1(T_k) + \epsilon\right\}, \tag{5.43}$$

where ϵ is a small positive number. This choice ensures that the eigenvalues of the modified matrix are greater than or equal to ϵ, and does not make any unnecessary modifications. We can easily compute the search direction

$$p_k(\theta) = p_0 + V_k T_k^{-1}(\theta)\|r_0\|e_1 \tag{5.44}$$

because all we need to do is factorize and solve for (factorize) the tridiagonal matrix $T_k(\theta)$.

The direction $p_k(\theta)$ can be interpreted as a combination of the Newton direction and the steepest descent direction in a subspace. In fact, the method described above attempts to compute a Newton direction p subject to the constraint

$$p^T p \leq \theta, \tag{5.45}$$

i.e., a trust-region constraint. By using CG/Lanczos to solve for p, we obtain an *inexact* or *truncated* Newton method. These methods have been studied extensively in the literature [11, 12, 52].

Let us summarize the overall algorithm. We attempt to solve $Hp = -g$ by a Lanczos process. If we successfully obtain a good descent direction, we are done. If at some point we detect that the tridiagonal matrix T_k is indefinite, we can obtain a direction of negative curvature from T_k and the Lanczos basis. We may either stop the iteration when this occurs, or we can continue the Lanczos process in order to determine a better direction of negative curvature. A descent direction is then available from (5.44).

5.3.3 Initial vector in Lanczos

We have not discussed the choice of initial vector for the Lanczos process, but tacitly assumed it is a multiple of the right hand side $-g$. There are some advantages in this choice, the most important being numerical stability. It has been shown [56] that the choice $-g$ is essential if the Lanczos vectors are not truly orthogonal, which is the case when no reorthogonalization is used.

A difficult case, often called *the hard case*, is when p is nearly orthogonal to g. In exact arithmetic, any Krylov method (like CG or Lanczos) will fail if the initial vector has no component in the direction of the solution. This principle also holds when we seek an eigenvector (or a direction of negative curvature). In the hard case, it would be better to start with a different initial vector, for example a random vector. The probability that a random vector is orthogonal to the desired solution vector is zero.

5.3.4 Practical Lanczos methods

Several times we have mentioned that a drawback of the Lanczos algorithm is that the eigenvector is not easily obtained because access is required to all the Lanczos vectors. Consequently, one must either store all the Lanczos vectors, or regenerate them. The first option is usually prohibited by memory limits, while the second option almost doubles the cost of the method. An alternative is to compute the desired eigenvector by inverse iteration. When we have a good estimate of the desired eigenvector, only a few (outer) iterations are needed to compute the eigenvector to high accuracy. Since we only seek one vector, this approach is attractive.

Another option is to use an *implicitly restarted* Lanczos method. Implicitly restarted Lanczos and Arnoldi methods [43, 44] have become very popular over the last few years. They work well when only a few eigenvalues/vectors are desired and memory is limited. The algorithms require $O(kn)$ storage to compute k eigenpairs. However, these methods are fairly complex and constitute a research field of their own, so they are beyond the scope of this thesis.

5.4 Preconditioning for eigen-like problems

For linear systems, it is well known that convergence can be accelerated by preconditioning the problem. The fundamental idea is to solve $M^{-1}Ax = M^{-1}b$ instead of $Ax = b$, where M is called the preconditioner. M is an approximation to A chosen such that $M^{-1}A$ is a "better" matrix than A in some sense. In the CG algorithm both A and M should be positive definite, and one would like the eigenvalues of $M^{-1}A$ to be clustered around one. Preconditioning for linear systems is quite well understood by now, especially the positive definite case.

It is not easy to precondition eigenvalue problems; in fact, some people claim it is impossible. (See [41] for an interesting discussion of preconditioned eigensolvers.) When you precondition a matrix the eigenvalues change, and thus you have changed the problem itself! However, once an eigenvalue approximation $\mu \approx \lambda$ is determined, one can precondition to solve for the eigenvector:

$$M^{-1}(A - \mu I)\, v = 0. \tag{5.46}$$

We are interested in computing good directions of negative curvature. As mentioned before, this process usually follows that of solving a linear system with the same matrix. Suppose we have solved $Hp = -g$ using a Krylov method like CG or Lanczos with a preconditioner. That is, we have solved $\tilde{H}p = -\tilde{g}$, where $\tilde{H} = M^{-1}H$. We require that M be positive definite while H can possibly be indefinite. In practice we wish to preserve symmetry, so consider instead $\tilde{H} = C^T HC$, sometimes called split preconditioning. Left, right, and split preconditioning are essentially equivalent when the matrix is symmetric

[29, 69]. If we apply the methods we have described to $\tilde{H}$, we find a direction of negative curvature $\tilde{d}$ for $\tilde{H}$. A direction of negative curvature for H is given by $d = C\tilde{d}$, since

$$d^T H d = \tilde{d}^T C^T H C \tilde{d} = \tilde{d}^T \tilde{H} \tilde{d}. \tag{5.47}$$

Unfortunately, d may be a poor direction of negative curvature for H even if $\tilde{d}$ is a good direction of negative curvature for $\tilde{H}$. The reason for this is that a small eigenvalue of $\tilde{H}$ does not necessarily correspond to a small eigenvalue of H after preconditioning. A simple example illustrates this. Let

$$H = \begin{pmatrix} -\theta & \theta & 0 \\ \theta & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \tag{5.48}$$

where θ is small and positive. Let $C = \mathbf{diag}(\theta^{-1/2}, 1, 1)$. Then

$$\tilde{H} = C^T H C = \begin{pmatrix} -1 & \theta^{1/2} & 0 \\ \theta^{1/2} & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}. \tag{5.49}$$

A good approximation to the smallest eigenvector of $\tilde{H}$ is $\tilde{d} = \left(1, -\frac{1}{2}\theta^{1/2}, 0\right)$, because $\tilde{d}^T \tilde{H} \tilde{d}^T / (\tilde{d}^T \tilde{d}) = -1 - O(\theta)$. This gives $d = C\tilde{d} = (\theta^{-1/2}, -\frac{1}{2}\theta^{1/2}, 0)$ and $d^T H d/(d^T d) = -\theta - O(\theta^2)$. Clearly $\lambda_{\min}(H) = -1$ with eigenvector $v = (0, 0, 1)$, so d becomes an arbitrarily poor direction of negative curvature for H as $\theta \to 0$.

5.4.1 Preconditioned gradient methods

Several authors have suggested gradient-type methods with a preconditioner for calculating a desired eigenpair, see for instance Samokish [70], Knyazev [42], and D'yakonov [15, Ch.9].

The preconditioned steepest descent algorithm for the Rayleigh quotient can be written as

$$x_{k+1} = x_k - \alpha_k M^{-1} \nabla R(x_k), \tag{5.50}$$

where M is some preconditioner. This method is closely related to steepest descent on the Rayleigh quotient of the preconditioned matrix $M^{-1}H$. The main difference is that the line search is performed with respect to the Rayleigh quotient of H and not that of $M^{-1}H$.

We observe that when M is the Hessian $\nabla^2 R(x)$, this method is a variant of Newton's method. One difference from Newton's method is that M stays fixed and is not updated at every step.

5.4.2 More remarks on preconditioning

Just as standard Lanczos is closely related to CG with no preconditioning, there is a preconditioned Lanczos method analogous to preconditioned CG. The (preconditioned) Lanczos vectors are no longer orthogonal but M-orthogonal (where M is a positive definite preconditioner), that is, $V_k^T M V_k = I$. The tridiagonal matrix $T_k = V_k^T H V_k$ does not approximate the eigenvalues of H, but rather those of the preconditioned matrix $M^{-1}H$. Consequently, we cannot simply look at the extreme eigenpair of T_k to find a good direction of negative curvature for H. Morgan and Scott [51] have suggested applying Lanczos to the matrix $W = L^{-1}(A - \mu I)L^{-T}$, where μ is an estimate of the desired eigenvalue and LL^T is the factorization of the preconditioner M. This technique can also be used to compute a direction of negative curvature once an estimate of the smallest eigenvalue is known.

The Jacobi-Davidson method (described in Section 5.2.2) is an eigenvalue algorithm that allows for preconditioning. A key feature of the method is that a linear subproblem is preconditioned, not the eigenproblem itself. Hence, this method may be a good choice if one needs an iterative method to compute a direction of negative curvature and little information is known. However, we view this as an unlikely scenario because normally a linear system first has to be solved to find a descent direction.

When a modified Cholesky factorization

$$H + E = LL^T$$

is available, then $M = LL^T$ is an obvious choice of preconditioner. The quality of this preconditioner will depend on E. In the extreme case, when $E = 0$, we have a "perfect" preconditioner.

However, if the modified Cholesky factors are available, a direction of negative curvature can be obtained cheaply by a single backsolve, and preconditioning would only be of interest in an iterative improvement phase. Since few iterations are performed in this phase, we expect the effect of preconditioning to be rather small.

A topic we have not studied is the possibility of using an incomplete modified Cholesky factorization.

5.5 Summary and conclusions

Directions of negative curvature play an important role when second-order optimality points are sought. We have shown that such directions are closely related to the smallest eigenvalue and eigenvector pair of the Hessian, and the Rayleigh quotient. We have described the distribution function of the Rayleigh quotient, and have shown how unlikely it is to find *by chance* a direction of negative curvature for an indefinite matrix with only a few negative eigenvalues. Several direct and iterative methods for computing a direction of negative curvature have been described.

We have shown that a dramatic improvement in the quality of a direction of negative curvature obtained from a direct method can be achieved using an iterative procedure. The direct factorization used is usually available at no extra cost because it is also needed to find a descent direction. No attempt was made to precondition the iterative procedures using the available Cholesky factorization of $H + E$, though that is a good candidate for future research. For problems with few negative eigenvalues it is typically the case that E is of very low rank, which suggests the (modified) Cholesky factorization will be an excellent preconditioner within a method such as the Jacobi-Davidson method [72].

For problems of the size studied here the use of a preconditioner is hardly merited except for pathological examples, but for larger problems that may not be the case. For very large problems, any explicit factorization will require too much storage, so a purely iterative method must be used. In this case, an iterative method has normally already been invoked to compute a descent direction and it should be possible to use information from this iterative process when computing a direction of negative curvature. We have shown how to exploit, for instance, the CG-Lanczos connection to achieve this.

We have discussed negative curvature issues in the context of unconstrained minimization. More research is needed to determine how much of these findings apply to constrained optimization.

Bibliography

[1] J. O. Aasen. On the reduction of a symmetric matrix to tridiagonal form. *BIT (Nordisk Tidskr. Informationsbehandling)*, 11:233–242, 1971.

[2] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods.* Academic Press, New York and London, 1982. ISBN 0-12-093480-9.

[3] M. C. Biggs. Constrained minimization using recursive equality quadratic programming. In F. A. Lootsma, editor, *Numerical Methods for Nonlinear Optimization*, pages 411–428. Academic Press, London and New York, 1972.

[4] E. K. Blum and G. H. Rodrigue. Solution of eigenvalue problems in Hilbert spaces by a gradient method. *J. of Computer and System Sciences*, 8:220–237, 1974.

[5] E. G. Boman and W. Murray. An iterative approach to computing directions of negative curvature. In *Proceedings of the Fifth Copper Mountain Conference on Iterative Methods.* University of Colorado, 1998.

[6] J. V. Burke and S-P. Han. A robust sequential quadratic programming algorithm. *Math. Prog.*, 43:277–303, 1989.

[7] S. H. Cheng and N. Higham. A modified Cholesky algorithm based on a symmetric indefinite factorization. *SIAM J. Matrix Anal. Appl.*, 19(4):1097–1110, 1998.

[8] J. W. Chinneck. MINOS(IIS) : Infeasibility analysis using MINOS. *Computers & Operations Research*, 21(1):1–9, 1994.

[9] J. W. Chinneck. Analyzing infeasible nonlinear programs. *Computational Optimization and Appl.*, 4(2):67–179, 1995.

[10] J. K. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol.1.* Birkhauser, 1985.

[11] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[12] R. S. Dembo and T. Steihaug. Truncated-Newton algorithms for large-scale unconstrained optimization. *Math. Prog.*, 26:190–212, 1983.

[13] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[14] M. Dumais and G. P. H. Styan. A bibliography on the distribution of quadratic forms in normal variables with special emphasis on the Craig-Sakamoto theorem and on Cochran's theorem. In *The Seventh International Workshop on Matrices and Statistics, in Celebration of T.W. Anderson's 80th Birthday*, December 1998. Bibliography available from styan@math.mcgill.ca.

[15] E. G. D'yakonov. *Optimization in Solving Elliptic Problems.* CRC Press, Boca Raton, FL., 1996.

[16] A. Edelman and S. T. Smith. On conjugate gradient-like methods for eigen-like problems. *BIT*, 36(1):1–16, 1996.

[17] D. K. Fadeev and V. N. Faddeeva. *Computational Methods of Linear Algebra.* W. H. Freeman and Co., 1963. Translated from Russian.

[18] R. Fletcher. *Practical Methods of Optimization.* Volume 2: Constrained Optimization. John Wiley and Sons, Chichester and New York, 1981.

[19] R. Fletcher. An ℓ_1 penalty method for nonlinear constraints. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel, editors, *Numerical Optimization 1984*, pages 26–40, Philadelphia, 1985. SIAM.

[20] R. Fletcher. *Practical Methods of Optimization.* John Wiley and Sons, Chichester, New York, Brisbane, Toronto and Singapore, second edition, 1987.

[21] A. Forsgren, P. E. Gill, and W. Murray. Computing modified Newton directions using a partial Cholesky factorization. *SIAM J. on Scientific Computing*, 16:139–150, 1995.

[22] W. N. Gansterer and C. W. Ueberhuber. Inner-outer inverse iteration for eigenvector computation. In *Proceedings of the Fifth Copper Mountain Conference on Iterative Methods*. University of Colorado, 1998.

[23] J. Gil-Pelaez. Note on the inversion theorem. *Biometrika*, 38:481–482, 1951.

[24] P. E. Gill and W. Murray. Newton-type methods for unconstrained and linearly constrained optimization. *Math. Prog.*, 7:311–350, 1974.

[25] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. Numerical Analysis Report 97-1, Department of Mathematics, University of California, San Diego, La Jolla, CA, 1997.

[26] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Some theoretical properties of an augmented Lagrangian merit function. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 101–128. North Holland, 1992.

[27] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, 1981.

[28] Meredith J. Goldsmith. *Sequential quadratic programming methods based on indefinite Hessian approximations.* PhD thesis, Dept. of EES and Oper. Res., Stanford University, 1999.

[29] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins Univ. Press, third edition, 1996.

[30] Gene H. Golub and Yong Sun, 1998. Personal communication.

[31] N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Exploiting negative curvature directions in linesearch methods for unconstrained optimization. Technical Report 97–18, Dept. of Math., FUNDP, Belgium, Nov. 1997.

[32] M. Gu and S. C. Eisenstat. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM J. Matrix Anal. Appl.*, 16:172–191, 1995.

[33] S. P. Han. Superlinearly convergent variable metric algorithms for general nonlinear programming problems. *Math. Prog.*, 11:263–282, 1976.

[34] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes.* Lecture Notes in Economics and Mathematical Systems 187. Springer Verlag, Berlin, Heidelberg and New York, 1981.

[35] J. P. Imhof. Computing the distribution of quadratic forms in normal variables. *Biometrika*, 48:419–426, 1961.

[36] Ilse C. F. Ipsen. Computing an eigenvector with inverse iteration. *SIAM Review*, 39(2):254–291, June 1997.

[37] Norman L. Johnson and Samuel Kotz. *Continuous Univariate Distributions – 2.* Houghton Mifflin Company, 1970.

[38] W. Kahan. Numerical linear algebra. *Canad. Math. Bull.*, 9:757–801, 1966.

[39] L. V. Kantorovich and G. P. Akilov. *Functional Analysis.* Pergamon Press, 1964. Translated from Russian.

[40] D. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, Reading, MA, USA, second edition, 1981.

[41] A. V. Knyazev. Preconditioned eigensolvers—an oxymoron? *Electronic Transactions on Numerical Analysis*, 7:104–123, 1998. Available from http://etna.mcs.kent.edu.

[42] A. V. Knyazev and A. L. Skorokhodov. On exact estimates of the convergence rate of the steepest ascent method in the symmetric eigenvalue problem. *LAA*, 154:245–257, 1991.

[43] R. B. Lehoucq. *Analysis and Implementation of an Implicitly Restarted Iteration.* PhD thesis, Dept. of Comp. and Appl. Math., Rice University, May 1995.

[44] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK User's Guide.* SIAM Publications, Philadelphia, 1998. Also available from http://www.caam.rice.edu/software/ARPACK.

[45] S. Lucidi and M. Roma. Numerical experience with new truncated Newton methods in large scale unconstrained optimization. *Computational Optimization and Applications*, 7(1):71–87, 1997.

[46] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, second edition, 1984.

[47] A. M. Mathai and S. B. Provost. *Quadratic Forms in Random Variables.* Marcel Dekker Inc, 1992.

[48] J. J. Moré and D. C. Sorensen. On the use of directions of negative curvature in a modified Newton method. *Math. Prog.*, 16:1–20, 1979.

[49] W. Murray. An algorithm for constrained minimization. In R. Fletcher, editor, *Optimization.* Academic Press, London/New York, 1969.

[50] W. Murray. Recent advances in nonlinear optimization. Presented at the PWACM workshop in Caracas, Venezuela, 1993.

[51] B. A. Murtagh and M. A. Saunders. MINOS 5.4 User's Guide. Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, CA, 1993.

[52] S. G. Nash. Newton-type minimization via the Lanczos method. *SIAM J. Numer. Anal.*, 21:770–788, 1984.

[53] A. Neumaier. On satisfying second-order optimality conditions using imodified Cholesky factorzations. Revised May 1997., 1996.

[54] A. M. Ostrowski. On the convergence of the Rayleigh quotient iteration for the computation of characteristic roots and vectors, i-vi. *Arch. Rational Mech. Anal.*, 1–4:233–241, 423–428, 325–340, 341–347, 472–481, 153–156, 1958–59.

[55] C. C. Paige. *The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices.* PhD thesis, London University, London, UK, 1971.

[56] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[57] B. N. Parlett. The Rayleigh quotient iteration and some generalizations for nonnormal matrices. *Math. Comp.*, 28:679–693, 1974.

[58] B. N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1980.

[59] G. Peters and J. H. Wilkinson. Inverse iteration, ill-conditioned equations, and Newton's method. *SIAM Rev.*, 21:339–360, 1979.

[60] M. J. D. Powell. A fast algorithm for nonlinearly constrained optimization calculations. Technical Report 77/NA 2, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 1977.

[61] M. J. D. Powell. Algorithms for nonlinear constraints that use Lagrangian functions. *Math. Prog.*, 14:224–248, 1978.

[62] M. J. D. Powell. The convergence of variable metric methods for nonlinearly constrained optimization calculations. In O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, editors, *Nonlinear Programming 3*, pages 27–63. Academic Press, London and New York, 1978.

[63] M. J. D. Powell. Variable metric methods for constrained optimization. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 288–311. Springer Verlag, London, Heidelberg, New York and Tokyo, 1983.

[64] V. G. Prikazchikov. Strict estimates of the rate of convergence of an iterative method of computing eigenvalues. *USSR Comp. Math. Math. Phys.*, 15(5):235–239, 1975. Translated from Russian.

[65] R. T. Rockafellar. A dual approach to solving nonlinear programming problems by unconstrained optimization. *Math. Prog.*, 5:354–373, 1973.

[66] A. Ruhe. Iterative eigenvalue algorithms for large symmetric matrices. In *Numerische Behandlung von Eigenwertaufgaben Oberwolfach 1972*, volume 24 of *Intl. Series Num. Math.*, pages 97–115, 1974.

[67] A. Ruhe. SOR methods for the eigenvalue problem with large sparse matrices. *Math. Comp.*, 28:695–710, 1974.

[68] Y. Saad. On the rates of convergence of the Lanczos and the block Lanczos methods. *SIAM J. Numer. Anal.*, 17:687–706, 1980.

[69] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, Boston, MA, 1996.

[70] B. A. Samokish. The steepest descent method for an eigenvalue problem with semi-bounded operators. *Izv. Vusov Math.*, 5:105–114, 1958. In Russian.

[71] G. L. G. Sleijpen and H. A. Van der Vorst. The Jacobi-Davidson method for eigenvalue problems as an accelerated inexact Newton scheme. Technical report, Mathematical Institute, Utrecht University, 1995.

[72] G. L. G. Sleijpen and H. A. Van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17:401–425, 1996.

[73] P. Spellucci. A new technique for inconsistent QP problems in the SQP method. *Math. Meth. Oper. Res.*, 47(3):355–400, 1998.

[74] P. Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. *Math. Prog.*, 82(3):413–448, 1998.

[75] G. W. Stewart. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM J. Numer. Anal.*, 17:403–409, 1980.

[76] D. B. Szyld. Criteria for combining inverse and Rayleigh quotient iteration. *SIAM J. Numer. Anal.*, 25(6):1369–1375, 1988.

[77] K. Tone. Revision of constraint approximations in the successive QP method for nonlinear programming problems. *Math. Prog.*, 26:144–152, 1983.

[78] J. van Loon. Irreducibly inconsistent systems of linear inequalities. *European J. of Operations Research*, 8:283–288, 1981.

[79] R. S. Varga. *Matrix Iterative Analysis.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1962.

[80] S. A. Vavasis. *Nonlinear optimization : Complexity issues.* Number 8 in International series of monographs on computer science. Oxford University Press, 1991.

[81] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, UK, 1965.

[82] R. B. Wilson. *A simplicial method for convex programming.* PhD thesis, Harvard University, 1963.

[83] S. J. Wright. Modifying SQP for degenerate problems. Technical Report ANL/MCS-P699-1097, MCS Division, Argonne National Laboratory, 1997.

[84] G. L. Zhou. A modified SQP method and its global convergence. *J. of Global Optimization*, 11(2):193–205, 1997.