UNIVERSITY OF CALIFORNIA, SAN DIEGO

Iterative Methods for Large-Scale Unconstrained Optimization

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Mathematics

by

Jennifer B. Erway

Committee in charge:

Professor Philip Gill, Chair
Professor Randolph Bank
Professor Robert R. Bitmead
Professor Michael Holst
Professor Bhaskar D. Rao

2006

The dissertation of Jennifer B. Erway is approved, and it is acceptable in quality and form for publication on microfilm:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2006

To B.W.

TABLE OF CONTENTS

LIST OF TABLES

ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance and support of my advisor, Philip Gill. His dedication to this research and his patience in support of me for many years was, and continues to be, the greatest factor in my accomplishments in mathematics.

I would also like to thank the committee members Randy Bank, Robert Bitmead, Mike Holst, and Bhaskar Rao for their support in writing this thesis.

I am grateful to the UCSD math department for their patience during this endeavor; in particular, I would like to thank the Computational and Applied Mathematics group. I am especially indebted to Mike Holst and Randy Bank for their unwavering encouragement and reassurance.

In the writing of this thesis, Josh Griffin's advice and suggestions were indispensable. And, I also wish to thank Kathleen Dillon who, one trying afternoon over a decade ago, meticulously criticized a paper I had written; on that day I learned some valuable lessons about writing.

Finally, I would like to thank my family and friends for their continued support and confidence in this undertaking. Without them, I would not be where I am today.

VITA

1997	B. A., *summa cum laude*, Claremont McKenna College
1999	M. A., University of California, San Diego
2006	Ph. D., University of California, San Diego

ABSTRACT OF THE DISSERTATION

Iterative Methods for Large-Scale Unconstrained Optimization

by

Jennifer B. Erway
Doctor of Philosophy in Mathematics
University of California San Diego, 2006
Professor Philip Gill, Chair

An unconstrained minimizer of a general nonlinear function may be found by solving a sequence of constrained subproblems in which a quadratic model function is minimized subject to a "trust-region" constraint on the norm of the change in variables. For the large-scale case, Steihaug has proposed an iterative method for the constrained subproblem based on the preconditioned conjugate-gradient (PCG) method. This method is terminated inside the trust region at an approximate minimizer or at the point where the iterates cross the trust-region boundary. When the iterates are terminated at the trust-region boundary, the final iterate is generally an inaccurate solution of the constrained subproblem. This may have an adverse affect on the efficiency and robustness of the overall trust-region method.

A PCG-based method is proposed that may be used to solve the trust-region subproblem to any prescribed accuracy. The method starts by using a modified Steihaug method. If the solution lies on the trust-region boundary, a PCG-based sequential subspace minimization (SSM) method is used to solve the constrained problem over a sequence of evolving low-dimensional subspaces. A new regularized sequential Newton method is used to define basis vectors for the subspace minimization. Several preconditioners are proposed for the PCG iterations. Numerical results suggest that, in general, a trust-region method based on the proposed solver is more robust and requires fewer function evaluations than Steihaug's method.

1

Introduction

1.1 Overview

Unconstrained optimization is fundamental to both the theory and practice of optimization. Many mathematical modeling and curve-fitting problems, ranging from modeling protein structure energy to designing automobile bodies, may be formulated as an unconstrained optimization problem. In addition, problems with constraints may be solved as a sequence of unconstrained problems, making advances in unconstrained optimization applicable to constrained optimization.

Unconstrained optimization involves finding the vector of variables x such that an objective function $f(x)$ achieves its minimum or maximum value. Thus, unconstrained optimization problems are usually categorized by the characteristics of the function f. This thesis focuses on techniques for solving general nonlinear unconstrained problems, assuming only that f is twice-continuously differentiable. As minimizing $f(x)$ is mathematically equivalent to maximizing $-f(x)$, all problems are formulated as minimization problems.

Trust-region methods are a class of methods for unconstrained minimization that define a sequence of iterates $\{x_j\}$ such that

$$x_{j+1} = x_j + s_j, \quad \text{with} \quad f(x_{j+1}) < f(x_j).$$

Each step s_j is the solution of a simpler *constrained* subproblem that involves minimizing a local quadratic model of f subject to a "trust-region" constraint on the norm of s_j. This *trust-region subproblem* has the form

$$\begin{aligned} \underset{s\in\mathbb{R}^n}{\text{minimize}} \quad & f(x_j) + \nabla f(x_j)^T s + \tfrac{1}{2} s^T \nabla^2 f(x_j) s \\ \text{subject to} \quad & \|s\| \leq \delta_j, \end{aligned} \tag{1.1}$$

where $\nabla f(x_j)$ and $\nabla^2 f(x_j)$ denote the gradient vector and Hessian matrix of f evaluated at x_j. The scalar δ_j is known as the *trust-region radius* and is chosen to provide the descent condition $f(x_{j+1}) < f(x_j)$. Trust-region methods are motivated by the presumption that the local quadratic model can be trusted only in some sufficiently small neighborhood of x_j. Trust-region methods accordingly adjust the trust-region radius δ_j to ensure that the reduction in f is comparable to the reduction predicted by the quadratic model.

The computational complexity of the trust-region subproblem (1.1) depends on the choice of norm used to restrict the length of s. This thesis will focus on the two-norm, which guarantees that any step satisfying the necessary and sufficient conditions for solution of (1.1) is a global minimizer of the subproblem (see Chapter 2). With this choice of norm, the computational complexity of problem (1.1) is equivalent to that of finding an extremal eigenpair of a symmetric matrix. However, the method used to solve each subproblem is for the most part little concerned with the ability to achieve anything near machine-precision accuracy. Of far greater importance to the overall solver is the ability to obtain *approximate* solutions of the subproblem efficiently. Two factors significantly affect the speed of the overall trust-region method: (i) the average amount of work needed to solve the trust-region subproblem, and (ii) the total number trust-region subproblems that must be solved, which in the algorithms to be considered is in direct relation to the number of evaluations of the objective function f and its first and second derivatives. Increasing the accuracy of the computed trust-region approximate solution usually increases the cost of solving each trust-region subproblem while decreasing the total number of function evaluations. Ultimately there is a point

where the total number of function evaluations reaches its minimum value; at this point, the extra work needed to solve the subproblems more accurately is wasted. What is relevant is the total work needed to find an unconstrained minimizer.

Broadly speaking there are two types of trust-region method based on the quality of the approximate trust-region solution. The first type proceeds to define an infinite sequence of "inner" iterates $\{s_j^i\}$ that converges to a solution of (1.1). An approximate solution is then defined by terminating this infinite sequence when s_j^i is judged to be accurate to within some preassigned tolerance. The method of Moré and Sorensen [32] is one of the principal methods of this type (see Section 2.3.5). In this method, each inner iterate s_j^i is found at the cost of computing the Cholesky factorization of a symmetric positive-definite matrix of the form $\nabla^2 f(x_j) + \sigma^i I$, where σ^i is a nonnegative scalar.

The second type of method does not even attempt to find a solution of (1.1). Instead, the step need only satisfy the trust-region constraint and force the overall convergence of the outer iterates $\{s_j\}$. The prototype for such methods was proposed by M. J. D. Powell in a seminal series of papers [41, 42, 43]. Powell proposed that the subproblem (1.1) be solved with the additional restriction that s be in a one-dimensional subspace spanned by the steepest-descent direction $-\nabla f(x_j)$. The optimal solution of this restricted problem is easily computed and is known as the *Cauchy step*. A fundamental result is that any method delivering an approximate solution of (1.1) with model value at least as good as the Cauchy step will generate a sequence $\{x_j\}$ such that $\nabla f(x_j) \to 0$ (see Section 2.3.3 for details).

Any computational method based on using the Cauchy step is guaranteed to converge, but may not converge quickly. Early methods, known as "dog-leg" methods, attempted to improve the convergence rate by combining the Cauchy step with a step based on the unconstrained minimizer of the quadratic model (see, e.g., Powell [42], Dennis and Mei [6], and Shultz, Schnabel and Byrd [47]). This idea has been refined by Byrd, Schnabel and Shultz [2], who propose solving the subproblem (1.1) subject to the restriction that s lies in a one- or two-dimensional subspace.

In this case, one of the basis vectors for the subspace is found by computing the Cholesky factors of a positive-definite matrix of the form $\nabla^2 f(x_j) + \sigma I$. The other basis vector, when needed, is a direction of negative curvature for $\nabla^2 f(x_j)$, i.e., a vector q such that $q^T \nabla^2 f(x_j) q < 0$.

Line-search methods constitute a different approach to minimizing f. Line-search methods are also descent methods, but define the step s_j by minimizing an *unconstrained* local quadratic model of f. The next iterate is then defined by taking a scalar step along s_j that gives a lower value of f. There are various alternative conditions on the step length that guarantee convergence, but a common requirement is that the improvement in f be comparable to the improvement predicted by some a simple linear or quadratic model of f. In general, line-search methods require more iterations than trust-region methods, but involve less work per iteration. More recent methods combine classical line-search and trust-region methods. For example, Gertz [11], Toint [40], and Nocedal and Yuan [36] add a line search to the standard trust-region method. All of the methods described above for solving the trust-region subproblem may be used within a combination line-search trust-region method. Throughout this thesis, a combined method similar to that proposed by Gertz [11] is used to define the outer iterations (see Section 2.3.2).

The combination line-search trust-region methods mentioned above require the computation of an explicit matrix factorization and are limited to problems with dense unstructured Hessian matrices of order $\lesssim 1000$, or medium-to-large sparse Hessian matrices of order $\lesssim 10,000$. Very large-scale problems or problems with large unstructured Hessians must use iterative methods to solve the constituent linear systems. The main focus of this thesis is the formulation and analysis of trust-region methods that employ the conjugate-gradient (CG) method as the linear solver. In their simplest form, CG methods require the storage of at most four or five one-dimensional vectors and have the potential of solving problems with up to 10^9 variables.

Two of the earliest trust-region methods based on the CG method were pro-

posed by Steihaug [48] and Toint [40]. These methods are similar, and are of the second type defined above—i.e., they do not attempt to find an accurate solution of the constrained subproblem. First, suppose that the Hessian $\nabla^2 f(x_j)$ is positive definite, which implies that the step to the unconstrained minimizer of the quadratic model satisfies the positive-definite system $\nabla^2 f(x_j)s = -\nabla f(x_j)$. It is well-known that the CG method for the linear system $\nabla^2 f(x_j)s = -\nabla f(x_j)$ may be interpreted as a method for a unconstrained minimizer of the quadratic model. Suppose that the CG method generates a sequence $\{s_j^{(k)}\}$ of approximate unconstrained minimizers. In the positive-definite case, two things can happen. First, $\{s_j^{(k)}\}$ may converge to a minimizer of the quadratic model lying inside the trust-region. In this case, the sequence $\{s_j^{(k)}\}$ is terminated at an approximate minimizer. Alternatively, the CG iterates will leave the trust region, in which case it can be shown that the solution of (1.1) must lie on the trust-region boundary. In this situation, both Steihaug and Toint terminate the CG sequence $\{s_j^{(k)}\}$ at the point where the iterates "cross" the trust-region boundary (see Section 4.2 for details). If $\nabla^2 f(x_j)$ is not positive definite, the CG method will eventually generate a direction q such that $q^T \nabla^2 f(x_j) q \leq 0$. In this situation, Toint's method reverts to the Cauchy step and Steihaug's method uses the direction q to define a point on the trust-region boundary.

Steihaug's method is generally preferred because it always gives a point on the boundary when the trust-region solution lies on the boundary. Trust-region methods based on using Steihaug's method to solve the subproblem provide a viable alternative to direct factorization methods in the large-scale case. However, in the indefinite case, the point at which the boundary is encountered depends only on the initial CG iterate, and so the final iterate is generally an inaccurate solution of the constrained subproblem. This may have an adverse affect on the efficiency and robustness of the overall trust-region method. This thesis concerns the formulation and analysis of CG-based methods that are designed to solve the trust-region subproblem to any prescribed accuracy.

One last remark concerning notation is in order. We have outlined iterative methods that defined sequences of iterates at three different levels: (i) the sequence $\{x_j\}$ is intended to converge to an unconstrained minimizer of f; (ii) the sequence $\{s_j^i\}$ converges to a solution of the trust-region subproblem at x_j, and (iii) $\{s_j^{(k)}\}$ is the sequence of CG iterates associated with the linear system $\nabla^2 f(x_j)s = -\nabla f(x_j)$. Fortunately, the exposition never requires the use of all three sequences simultaneously and so when the context is clear the elements of $\{s_j^{(k)}\}$ will be denoted s_k—i.e., a suffix k denotes an iterate associated with the CG method.

1.2 Contributions of this thesis

Chapter 2 presents a comprehensive review of existing line-search and trust-region methods for unconstrained optimization.

Chapter 3 considers the derivation of the CG method for solving linear equations and explores the equivalence between solving linear systems and minimizing strictly convex quadratic functions. The discussion highlights the equivalence of the CG method and the Lanczos-CG method. This equivalence allows the use of the Lanczos vectors as direction generators for the CG method and allows the simultaneous approximation of the subproblem solution and the leftmost eigenvector of the Hessian. The chapter concludes with a discussion of the role of preconditioning in the CG method and describes the preconditioned Lanczos-CG method.

Chapter 4 considers trust-region methods for large-scale optimization. Both Steihaug's method and the generalized Lanczos trust-region (GLTR) method of Gould, Lucidi, Roma, and Toint [18] are discussed in the context of controlling the accuracy of the trust-region solution on the boundary. Finally, a variant of the sequential subspace minimization (SSM) method of Hager [21] and Griffin [20] is applied to the trust-region subproblem.

In Chapter 5, a new two-phase method (the "phased-SSM method") is proposed for solving the trust-region subproblem to within a specified accuracy. The method

employs a PCG-based sequential subspace minimization method in each phase. The first phase can be viewed as an improved version of Steihaug's method in which two additional features are provided by the use of the Lanczos-CG method: (i) the availability of an estimate of the leftmost eigenvector at no extra cost; and (ii) the ability to compute a better exit point based on solving the trust-region subproblem on a low-dimensional subspace. Property (i) extends Steihaug's method to the case where the initial point is a stationary point but not a local minimizer. Property (ii) allows the calculation of a substantially better estimate of a trust-region solution on the boundary.

The second phase of the phased-SSM method is activated if the trust-region solution lies on the boundary. A PCG-based SSM method is used to solve the constrained problem over a sequence of evolving subspaces. The subspaces are spanned by three vectors: (i) the current best approximation to the solution; (ii) an estimate of the leftmost eigenvector; and (iii) an approximate solution of the Newton-SQP equations associated with the equality-constraint trust-region subproblem.

The Newton-SQP equations are not positive definite and so are not amenable to solution using the CG method. However, the equations may be *regularized* by applying a variant of the penalty function method of Forsgren and Gill [8]. The resulting regularized system is transformed into a symmetric positive-definite system that can be solved using the CG method. A second variant of phase two is based on minimizing an augmented primal-dual Lagrangian with a barrier function term. The barrier function allows the use of safeguarding and prevents this system from becoming indefinite at points that are far from the solution. With appropriate safeguarding, the associated Newton equations are also positive definite and may be solved using the PCG method.

In Chapter 6, block preconditioners are considered with the aim of removing ill-conditioning and accelerating convergence. Two general-purpose preconditioners are proposed. The first is based on an incomplete Cholesky factorization of $\nabla^2 f(x)$.

The second utilizes a limited-memory BFGS quasi-Newton approximation.

In Chapter 7, numerical results are presented based on a preliminary MATLAB implementation of the phased-SSM method. The results suggest that, in general, a trust-region method based on the proposed solver is more robust and requires fewer function evaluations than Steihaug's method.

1.3 Notation

Most of the notation in this thesis is standard in the optimization literature. Subscripts refer to both indices of vectors and iterates, with the precise meaning determined by the context. For example, x_j denotes the kth iterate in the sequence $\{x_j\}$, and f_j denotes $f(x_j)$. Throughout, e_i denotes the ith standard basis vector.

Lower-case Roman letters will be used to refer to vectors, and upper-case Roman letters will be reserved for matrices. Lower-case Greek letters will be used to refer to scalars, and upper-case Greek letters will be reserved for garden-variety functions. (An important exception to this rule is the letter "f", which typically is used for functions in the literature.) Well-known functions (i.e., functions with names) are generally written with calligraphic (script) Roman letters. Otherwise, calligraphic (script) Roman letters will refer to sets. These conventions will be departed from only when standard notation dictates otherwise.

Unless explicitly indicated otherwise, $\|\cdot\|$ denotes the vector two-norm or its subordinate matrix norm. The spectrum of a (possibly unsymmetric) matrix A will be denoted by $\mathrm{eig}(A)$. The inertia of a real symmetric matrix A, denoted by $\mathrm{In}(A)$, is the integer triple (a_+, a_-, a_0) giving the number of positive, negative and zero eigenvalues of A. The ith eigenvalue of a symmetric matrix A with eigenvalues ordered in decreasing order will be denoted by $\lambda_i(A)$, i.e., $\lambda_1(A) \geq \lambda_2(A) \geq \cdots \geq \lambda_n(A)$. Given vectors x_1 and x_2, the column vector consisting of the elements of x_1 augmented by the elements of x_2 is denoted by (x_1, x_2). The vector e denotes the vector of ones with dimension determined by the context.

Some sections provide algorithms written in a MATLAB-style pseudocode. In these algorithms, brackets will be used to differentiate between computed and stored quantities. For example, the expression $[Ax] \leftarrow Ax$ signifies that the matrix-vector product of A with x must be computed and assigned to a vector labeled as $[Ax]$.

2

Unconstrained Minimization

2.1 Overview

Finding a global minimizer of a function f without strong assumptions about its curvature is an NP-hard problem. Most deterministic optimization methods are designed to find the "next best thing", which is a local minimizer. In this thesis, "minimizing" f implicitly refers to finding a local minimizer of f, unless stated otherwise. Mathematically speaking, x^* is a local minimizer of f if and only if $f(x^*) \leq f(x)$ for all x in some small neighborhood of x^*. Any local minimizer x^* of a differentiable function f is also a stationary point of f, i.e., $\nabla f(x^*) = 0$. A point x^* that satisfies $\nabla f(x^*) = 0$ is called a *first-order point*. Methods that explicitly seek first-order points are called *first-order methods*. However, since maximizers and saddle points are also stationary points, simply finding a root of the gradient of f is not sufficient and additional care must be taken to ensure that the root is also a minimizer. *Second-order methods* use the curvature of f to distinguish between different types of stationary points. If $\nabla^2 f(x^*)$ is positive definite and $\nabla f(x^*) = 0$ then x^* is a local unconstrained minimizer of f. Moreover, if x^* is a local unconstrained minimizer of f then it is necessary that $\nabla^2 f(x^*)$ is positive semidefinite and $\nabla f(x^*) = 0$. Any point that satisfies these necessary

conditions is called a *second-order point*. Methods that seek second-order points are called *second-order methods*.

The necessary condition $\nabla f(x^*) = 0$ implies that x^* is a *zero* or *root* of the vector-valued function ∇f. Newton's method is one of the most commonly used iterative methods for finding the root of a vector-valued function. Let x_j be any point at which f is twice-differentiable. Suppose that x_j is the current best estimate of a stationary point of f. For all x sufficiently close to x_j the *linear model function* $\nabla f(x_j)+\nabla^2 f(x_j)(x-x_j)$ will be a good approximation to $\nabla f(x)$. Newton's method is based on approximating a zero of g by a zero of the linear model, i.e., the next iterate x_{j+1} is chosen such that $\nabla f(x_j) + \nabla^2 f(x_j)(x_{j+1} - x_j) = 0$, or, equivalently,

$$x_{j+1} = x_j + s_j, \quad \text{where } s_j \text{ satisfies} \quad \nabla^2 f(x_j) s_j = -\nabla f(x_j). \tag{2.1}$$

If the Hessian $\nabla^2 f(x_j)$ is nonsingular, then the Newton step s_j is unique and satisfies $s_j = -\nabla^2 f(x_j)^{-1} \nabla f(x_j)$.

Theorem 2.1.1 below is arguably the most famous and fundamental result in optimization—that the pure Newton iterates (2.1) converge *quadratically* to a stationary point of f.

Theorem 2.1.1 (Local convergence of Newton's method.)
Let $f\colon \mathcal{D} \subseteq \mathbb{R}^n \mapsto \mathbb{R}^1$ be twice-continuously differentiable on $\mathcal{D}$, where $\mathcal{D}$ is an open convex set, and assume that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is nonsingular for some $x^* \in \mathcal{D}$. Then

(a) *there exists a neighborhood $\mathcal{B}(x^*, \delta)$ such that, for any x_0 in $\mathcal{B}$, the Newton iterates*

$$x_{j+1} = x_j - \nabla^2 f(x_j)^{-1} \nabla f(x_j), \quad k = 0, \ldots, \tag{2.2}$$

are well defined, remain in $\mathcal{B}$ and converge to x^* at a Q-superlinear rate;

(b) *if, in addition, $\nabla^2 f$ satisfies a local Lipschitz condition at x^*,*

$$\|\nabla^2 f(x) - \nabla^2 f(x^*)\| \leq L \|x - x^*\| \quad \text{for all } x \in \mathcal{B}, \tag{2.3}$$

where L is a positive constant, the Newton iterates defined by (2.2) *converge Q-quadratically to x^*; and*

(c) *if, in addition, $\nabla^2 f$ satisfies a Lipschitz condition with constant L in a* neighborhood *of x^*, $\|\nabla f(x_j)\|$ converges Q-quadratically to zero.* ∎

Unfortunately, this strong local convergence property applies to all types of stationary points. Since Newton's method is unable to discriminate between maximizers and minimizers, additional care must be taken to ensure convergence to a local minimizer. Line-search methods and trust-region methods are designed force the global convergence of Newton's method to a local minimizer.

If the Hessian is positive definite at x_j, the Newton direction s_j defined in (2.1) may be interpreted as the minimizer of a *local quadratic model* of f. Given a point x, the truncated Taylor-series expansion of f gives the following local quadratic model:

$$f(x+s) \approx q(x+s) = f(x) + \nabla f(x)^T s + \tfrac{1}{2} s^T \nabla^2 f(x) s. \tag{2.4}$$

If $\nabla^2 f(x_j)$ is positive definite, then s_j is the unique minimizer of $q(x_j + s)$. A constant objective term does not affect the location of a minimizer and it follows that a minimizer of $q(x+s)$ also minimizes the quadratic function

$$\mathcal{Q}(s) \equiv \nabla f(x)^T s + \tfrac{1}{2} s^T \nabla^2 f(x) s, \tag{2.5}$$

which approximates $f(x+s) - f(x)$, the change in f.

If $\nabla^2 f(x)$ is not positive definite, $\mathcal{Q}(s)$ and $q(x+s)$ have unbounded minimizers and are not suitable quadratic models. Line-search methods solve an *unconstrained* model problem of the form

$$\underset{s \in \mathbb{R}^n}{\text{minimize}} \quad \mathcal{Q}_j^B(s) = \nabla f(x)^T s + \tfrac{1}{2} s^T B_j s, \tag{2.6}$$

where B_j is a positive-definite approximation of $\nabla^2 f(x_j)$. Solving (2.6) for s_j does not guarantee $f(x_j + s_j) < f(x_j)$, but it does guarantee that there is some $\hat{\alpha}_j > 0$

such $f(x_j + \alpha_j s) < f(x_j)$ for all $\alpha_j \in (0, \hat{\alpha}_j)$, provided $\nabla f(x_j)$ is not zero. Line-search methods choose a step length α_j that gives at least a sufficient relative decrease in the objective function. As we will see later, the hallmark of any line search algorithm is the criteria used to define a sufficient decrease of the objective.

Trust-region methods solve the constrained minimization problem:

$$\begin{array}{ll} \underset{s \in \mathbb{R}^n}{\text{minimize}} & \mathcal{Q}_j(s) = \nabla f(x_j)^T s + \frac{1}{2} s^T \nabla^2 f(x_j) s \\ \text{subject to} & \|N_j s\| \leq \delta_j, \end{array} \tag{2.7}$$

where N_j is a nonsingular matrix that scales the elements of s. The minimization problem (2.7) is called the *trust-region subproblem*. A solution to the trust-region subproblem always exists; that is, $\mathcal{Q}_j(s)$ always assumes its bounded minimum value on the compact set $\{s : \|N_j s\| \leq \delta_j\}$. Thus, in contrast to line-search methods, the Hessian of f need not be positive definite in a trust-region setting.

Basic trust-region methods evaluate the step $x_j + s$, where s is an approximate solution to (2.7). If the condition for a sufficient decrease is met, the step $x_j + s$ is accepted. The trust-region radius is then updated for the next iteration to reflect the degree to which the quadratic model predicts the actual decrease in f. If the objective is decreased by significantly more than the minimum acceptable decrease, the trust-region radius is increased with the aim of exploiting the accuracy of the model in a larger trust-region. If the condition for sufficient decrease is not met, the step is rejected and the subproblem is solved again with a smaller value of the trust-region radius.

In order to simplify the notation in subsequent sections, $g(x)$ will denote the gradient $\nabla f(x)$ and $H(x)$ will denote the Hessian $\nabla^2 f(x)$. Similarly, g_j and H_j denote the quantities $g(x_j)$ and $H(x_j)$.

2.2 Line-Search Methods

Almost all line-search methods solve the minimization problem:

$$\underset{s}{\text{minimize}} \quad \mathcal{Q}_j^B(s) = g_j^T s + \tfrac{1}{2} s^T B_j s, \tag{2.8}$$

where B_j is a sufficiently positive-definite approximation to the Hessian. Since B_j is positive definite, the stationary point of $\mathcal{Q}_j^B(s)$ is uniquely defined by the nonsingular linear system $B_j s_j = -g_j$ and is the unique minimizer of $\mathcal{Q}_j^B(s)$. In the simplest line-search methods, the next iterate is then defined as

$$x_{j+1} = x_j + \alpha_j s_j,$$

where $B_j s_j = -g_j$ and $\alpha_j > 0$ satisfies $f(x_j + \alpha_j s_j) < f(x_j)$. Since B_j is positive definite, $g_j^T s_j = -s_j^T B_j s_j < 0$; that is, the directional derivative of f along s_j is negative. Thus, the vector s_j is called a *descent direction* for f. The following lemma shows that a descent direction for f is always a *direction of decrease*, i.e., there exists a positive $\hat{\alpha}_j$ such that any $\alpha_j \in (0, \hat{\alpha}_j)$, $f(x_j + \alpha_j s_j) < f(x_j)$.

Lemma 2.2.1 (Existence of a direction of decrease.) *Given $f\colon \mathcal{D} \subseteq \mathbb{R}^n \mapsto \mathbb{R}^1$ on a convex set $\mathcal{D}$, assume that f is continuously differentiable on $\mathcal{D}$, and let x_j be an interior point of $\mathcal{D}$.*

(i) *If the vector s satisfies $g(x_j)^T s < 0$, then s is a direction of decrease for f at x_j.*

(ii) *If, in addition, f has a second derivative at x_j, then any vector $\hat{s}$ satisfying $g(x_j)^T \hat{s}_j = 0$ and $\hat{s}_j^T H(x_j) \hat{s}_j < 0$ is a direction of decrease for f at x_j.* ∎

Computing a step length α_j with minimal computational effort is the hallmark of any line-search method. The univariate minimizer α_j for the problem

$$\underset{\alpha_j > 0}{\text{minimize}} \quad f(x_j + \alpha_j s_j) \tag{2.9}$$

is not easily calculated when f is a general nonlinear function; in fact, it may require a substantial number of function evaluations of f. Moreover, the requirement $f(x_{j+1}) < f(x_j)$ and the condition $g_j^T s_j < 0$ are not sufficient to guarantee the sequence $\{x_j\}$ converges to a minimizer of f: It is possible that α_j may become so small that the decrease in f converges to zero before the gradient converges to zero. Two of the most commonly used line-search methods offer a computationally inexpensive way to overcome this difficulty by choosing a step length that produces a *sufficient decrease* in f.

2.2.1 Modified Newton methods

At the jth iterate x_j of a modified Newton method, the next iterate is defined as $x_{j+1} = x_j + p_j(\alpha_j)$, where $p_j(\alpha)$ is a continuous (possibly nonlinear) path starting at zero, parameterized by a nonnegative scalar step length α. In order to guarantee a decrease in f at each step, the path $p_j(\alpha)$ must satisfy general conditions defined in terms of the value and derivatives of the univariate function $\varphi_j(\alpha) = f\big(x_j + p_j(\alpha)\big)$. These conditions require that either $\varphi_j'(0) < 0$, or $\varphi_j'(0) \leq 0$ and $\varphi_j''(0) < 0$ (for more details, see Moré and Sorensen [31]).

The path $p_j(\alpha)$ may be defined in terms of two directions (s_j, d_j) known as a *descent pair*. A descent pair must satisfy the conditions:

$$g_j^T s_j < 0 \ \text{ and } \ d_j^T H_j d_j \leq 0, \ \text{ if } H_j \text{ is positive definite;}$$

$$g_j^T s_j \leq 0 \ \text{ and } \ d_j^T H_j d_j < 0, \ \text{ otherwise.}$$

The additional condition $g_j^T d_j \leq 0$ is enforced by changing the sign of d_j if necessary. If no nontrivial descent pair exists at x_j, i.e., both s_j and d_j are zero, then g_j must be zero and H_j must be positive semidefinite, which implies that x_j satisfies the second-order necessary conditions for optimality. At any other point, at least one of the vectors s_j and d_j must be nonzero.

Given a descent pair (s_j, d_j), the search path is defined as

$$p_j(0) = 0, \qquad p_j(\alpha) = \phi_1(\alpha) s_j + \phi_2(\alpha) d_j, \quad \alpha \in [0, \infty),$$

where $\phi_1(\alpha)$ and $\phi_2(\alpha)$ are nonnegative scalar functions of $\alpha \in [0, \infty)$ such that the required conditions on φ_j are satisfied (see [31]). The simplest choice is $\phi_1(\alpha) = \phi_2(\alpha) = \alpha$, giving $p_j(\alpha) = \alpha(s_j + d_j) = \alpha p_j$, where p_j is the vector $s_j + d_j$ (see, e.g., Forsgren, Gill and Murray [9]). Another choice is $\phi_1(\alpha) = \alpha^2$ and $\phi_2(\alpha) = \alpha$, which defines the curvilinear path $p_j(\alpha) = \alpha^2 s_j + \alpha d_j$ (see, e.g., [28, 15, 31]).

The precise definition of the descent pair depends on the algorithm being used. However, certain general rules apply. If H_j is positive semidefinite then $d_j = 0$. Otherwise, a common choice for d_j is $\pm|\tilde{u}^T H_j \tilde{u}|^{1/2}\tilde{u}$, where $\tilde{u}$ is a direction of negative curvature. For example, if $\tilde{u}$ is an estimate of u_n, the normalized eigenvector corresponding to the least eigenvalue of H_j, then $d_j^T H_j d_j \approx -\lambda_n^2$. The sign of d_j is chosen so that $g_j^T d_j \leq 0$.

The direction s_j is defined as the unconstrained minimizer of the quadratic model:

$$\mathcal{Q}_j^B(s) = g_j^T s + \tfrac{1}{2} s^T B_j s, \tag{2.10}$$

where B_j is a symmetric *positive-definite* matrix such that $B_j \approx H_j$. The restriction that B_j be positive definite ensures that the unconstrained subproblem has a bounded solution. Each of the many choices for B_j leads to a different method.

Once a suitable descent pair has been defined, the step length α_j is chosen to enforce a sufficient decrease in f. There are a number of alternative sets of conditions on α_j that guarantee a sufficient decrease, but a requirement common to all of these conditions is that the decrease in f be better than some preassigned factor η_1 of the decrease predicted by some simple *line-search model function*. The Armijo line search uses the affine line-search model function $\mathcal{L}_j(s) = g_j^T s$, which gives the sufficient decrease criterion:

$$\big(f(x_j) - f(x_j + \alpha_j s_j)\big)/\mathcal{L}_j(\alpha_j s_j) \geq \eta_1, \tag{2.11}$$

where η_1 is a preassigned scalar such that $0 < \eta_1 < \frac{1}{2}$. If s_j is a descent direction, then this *Armijo condition* may be stated as

$$f(x_j) - f(x_j + \alpha_j s_j) \geq -\eta_1 \alpha_j g_j^T s_j. \tag{2.12}$$

Examples of a line-search model function are the affine function $\mathcal{L}_j(s) = g_j^T s$ and the quadratic $\mathcal{Q}_j(s) = g_j^T s + \frac{1}{2} s^T H_j s$.

Consider the line-search model

$$\mathcal{Q}_j^-(p) = g_j^T p + \tfrac{1}{2}\big[p^T H_j p\big]_-, \tag{2.13}$$

where $[\,c\,]_-$ denotes the negative part of c, i.e., $[\,c\,]_- = \min\{\,0, c\,\}$. With this choice of model, the sufficient decrease condition on α_j is

$$\frac{f\big(x_j + p_j(\alpha_j)\big) - f(x_j)}{\mathcal{Q}_j^-\big(p_j(\alpha_j)\big)} > \eta_1, \tag{2.14}$$

where η_1 is a preassigned scalar such that $0 < \eta_1 < \frac{1}{2}$. It may be observed that $\mathcal{Q}_j^-(p) = \mathcal{Q}_j(p)$, whenever $p^T H_j p \leq 0$. If $p^T H_j p > 0$, then $\mathcal{Q}_j^-(p) = \mathcal{L}_j(p)$, which imposes the more stringent Armijo condition on α_j.

A Wolfe line search chooses a step length α_j that satisfies two sufficient reduction criteria. The first Wolfe condition is the Armijo condition (2.11). The second Wolfe condition is derived from the observation that if α^* is the solution to (2.9) then $g(x_j + \alpha^* s_j)$ must be orthogonal to the search direction s_j, i.e., α^* satisfies the first-order necessary conditions of a local minimizer. The second Wolfe condition is given by:

$$|g(x_j + \alpha_j s_j)^T s_j| \leq \omega\, |g_j^T s_j|. \tag{2.15}$$

A Wolfe line search is not backtracking procedure, but rather a minimization routine that typically uses interpolating polynomials to model the behavior of f and a safeguarding method to reduce an interval containing a satisfactory step. If f is bounded below and $0 < \eta_1 < \omega < 1$, there is always a nontrivial open interval of step lengths that satisfy (2.12) and (2.15) [35]. Unlike backtracking line searches, a Wolfe line search is able to determine step sizes that are greater than one.

The remainder of this section is concerned with the properties of the jth iteration and so the suffix j is omitted.

2.2.2 The modified eigensystem paradigm

Almost all algorithms for modifying the Hessian are modeled on a method originally proposed by Greenstadt [19]. This method uses the properties of the spectral decomposition of the Hessian. Any symmetric matrix H may be written in the form $H = U\Lambda U^T$, where $U = (u_1\ u_2 \cdots\ u_n)$ is the matrix of eigenvectors of H and $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is the corresponding matrix of eigenvalues. The order of the eigenvalues is arbitrary; without loss of generality, let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.

The *modified eigensystem paradigm* defines the positive-definite model Hessian B as $B = H + F$, where F is a positive semidefinite matrix computed as follows:

(1) Compute the spectral decomposition $H = U\Lambda U^T$.

(2) Define the positive scalar ϵ (the magnitude of the smallest eigenvalue of B).

(3) Define modified eigenvalues $\tilde{\Lambda} = \Lambda + E$, such that

$$\widetilde{\lambda}_i = \max\{\, |\lambda_i|, \epsilon \,\}, \quad \text{for } i = 1:n. \tag{2.16}$$

(4) Define the modified Hessian $H + F = U\tilde{\Lambda}U^T = U(\Lambda + E)U^T$.

If β is a preassigned upper bound on the spectral condition number of B, then the value $\epsilon = \|H\|/\beta$, ensures that $\mathrm{cond}(B) \leq \beta$.

An alternative modification F is defined by replacing all small or negative eigenvalues by ϵ, i.e.,

$$\widetilde{\lambda}_i = \max\{\, \lambda_i, \epsilon \,\}, \quad \text{for } i = 1:n. \tag{2.17}$$

This modification may be interpreted as giving the modified matrix that minimizes $\|H - B\|$ subject to the restriction that $\mathrm{cond}(B) \leq \beta$. This modification appears prominently in the literature, but for reasons discussed in Section 2.2.4, it is not to be recommended for modified Newton line-search methods.

Given the spectral decompositions of H and B, a descent pair (s, d) is easily computed. The vector s may be written as $s = -U^T\tilde{\Lambda}^{-1}U^T g$. If $\lambda_n \geq 0$ then $d = 0$.

Otherwise, $d = -\operatorname{sign}(g^T u_n)|\lambda_n|^{1/2} u_n$, where u_n is the normalized eigenvector corresponding to λ_n. With this choice, the vector d satisfies $d^T H d = -\lambda_n^2 < 0$.

2.2.3 Practical modified Newton methods

In practice, it is too expensive to compute the spectral decomposition every iteration. Instead, modified Newton methods implicitly define transformed variables y such that $y = Sx$, where S is a nonsingular matrix that depends on H. The transformation S is chosen so that the Hessian with respect to the y-variables $H_y = S^{-T} H S^{-1}$, has an inexpensive spectral decomposition $H_y = U \Lambda U^T$. If Λ is modified so that $\tilde{\Lambda} = \Lambda + E$, then a modified Hessian for the original variables is

$$B = S^T U(\Lambda + E)U^T S = H + F, \quad \text{where} \quad F = S^T U E U^T S.$$

Similarly, if d_y is a direction of negative curvature for H_y, then $d = S^{-1} d_y$ satisfies $d^T H d = d_y^T S^{-T} H S^{-1} d_y = d_y^T H_y d_y < 0$ and d is an appropriate direction of negative curvature for H.

In the context of guaranteeing the convergence of the sequence $\{x_j\}$, it is necessary to require that the sequence $\{\operatorname{cond}(S_j)\}$ be bounded independently of j. Note that, although H_y and H have the same inertia, the magnitudes of their eigenvalues are different unless S is orthogonal. This implies that the better the conditioning of S, the better the curvature of H_y reflects the true curvature of H.

The identity $H = S^T H_y S$ indicates that the matrices S and H_y may be computed efficiently in terms of a symmetric matrix factorization of H – for example, the symmetric indefinite factorization $H = LDL^T$, where L is a row-permuted unit lower-triangular matrix and D is block diagonal.

2.2.4 Properties of the modified Newton direction

This section focuses on properties of the vector s obtained using the modified eigensystem paradigm. Recall that the scalar ϵ determines the eigenvalue of small-

est magnitude of B. If $H = U\Lambda U^T$, then Λ and U may be partitioned conformally so that

$$\Lambda = \begin{pmatrix} \Lambda_+ & & \\ & \Lambda_\epsilon & \\ & & \Lambda_- \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} U_+ & U_\epsilon & U_- \end{pmatrix},$$

where Λ_+, Λ_ϵ and Λ_- are the diagonal matrices of eigenvalues satisfying, respectively, $\lambda_i \geq \epsilon$, $|\lambda_i| < \epsilon$, and $\lambda_i \leq -\epsilon$.

Lemma 2.2.2 *Consider the positive-definite matrix B computed from H using the modified eigensystem paradigm. If $Bs = -g$ with $g \neq 0$ then s may be written in the form $s = s_+ + s_\epsilon + s_-$, where*

(i) *s_+ is a direction of positive curvature that* minimizes *$\mathcal{Q}(s)$ in the space spanned by U_+;*

(ii) *$(-s_-)$ is a direction of negative curvature that* maximizes *$\mathcal{Q}(s)$ in the space spanned by U_-; and*

(iii) *s_ϵ is a multiple of the steepest-descent direction in the space spanned by U_ϵ, with $\|s_\epsilon\| = O(1/\epsilon)$.*

Proof. Using the orthogonality of the eigenvectors, we obtain

$$s = -U\tilde{\Lambda}^{-1}U^T g = -U_+\tilde{\Lambda}_+^{-1}U_+^T g - U_\epsilon\tilde{\Lambda}_\epsilon^{-1}U_\epsilon^T g - U_-\tilde{\Lambda}_-^{-1}U_-^T g.$$

And, the definitions $\tilde{\Lambda}_+ = \Lambda_+$, $\tilde{\Lambda}_\epsilon = \epsilon I$ and $\tilde{\Lambda}_- = -\Lambda_-$ give

$$s = -U_+\Lambda_+^{-1}U_+^T g - \frac{1}{\epsilon}U_\epsilon U_\epsilon^T g + U_-\Lambda_-^{-1}U_-^T g.$$

A simple calculation shows that $s_+ = -U_+\Lambda_+^{-1}U_+^T g$ is the unique minimizer of $\mathcal{Q}(s)$ over all $s \in \text{span}(U_+)$. Similarly, $-s_- = -U_-\tilde{\Lambda}_-^{-1}U_-^T g$ is the unique maximizer of $\mathcal{Q}(s)$ over all $s \in \text{span}(U_-)$. The vector $s_\epsilon = U_\epsilon U_\epsilon^T g$ has length $\|U_\epsilon^T g\|/\epsilon$ and is the unique portion of the gradient in the space spanned by the columns of U_ϵ.

Finally, from the definition of s_+, we have

$$s_+^T H s_+ = \sum_{\lambda_i \geq \epsilon} \lambda_i \big(u_i^T g/\lambda_i\big)^2 > 0,$$

where $\{ u_i \}$ are the columns of U. This implies that s_+ is a direction of positive curvature for H. A similar calculation for s_- gives $s_-^T H s_- < 0$ as required. ∎

These results imply that if $g \neq 0$, then s_+ and s_- are descent directions of positive and negative curvature for f. The decomposition of s into s_ϵ, s_+ and s_- provides a simple interpretation of the strategy (2.16) used to modify the Hessian. If H is not sufficiently positive definite, then the modification reverses the portion of s that steps to the maximizer of the quadratic model in the subspace U_-. In this subspace, the quadratic model provides an incorrect estimate of the step to a minimizer of f—in particular, the model is implying that an infinite step is appropriate, when this unlikely for any reasonable f. Reversing the step to the maximizer rejects this scaling and chooses instead a step that reflects scaling implied by the step to the maximizer of the model.

Unfortunately, Lemma 2.2.2 also reveals that if H is near-singular, then the modified Newton direction is almost parallel to the portion of the steepest-descent direction in the subspace spanned by the eigenvectors with small eigenvalues. This implies that near a singular point, a modified Newton method is condemned to use the method of steepest descent regardless of magnitudes of the larger eigenvalues of H. This behavior can adversely affect performance.

This feature of the method is even more marked if the alternative modification (2.17) is used. The next result is a direct consequence of Lemma 2.2.2.

Corollary 2.2.1 *Assume that H has at least one eigenvalue less than ϵ. If B is defined using the modification* (2.17)*, i.e., $\widetilde{\lambda}_i = \max\{ \lambda_i, \epsilon \}$, then the modified Newton direction s such that $Bs = -g$ is almost parallel to a large multiple of the steepest-descent direction in the subspace spanned by the eigenvectors associated with the eigenvalues λ_i such that $\lambda_i \leq \epsilon$.* ∎

2.3 Trust-Region Methods

2.3.1 Motivation

Trust-region methods define the step by solving the constrained minimization problem

$$\begin{aligned} \underset{s}{\text{minimize}} \quad & \mathcal{Q}_j(s) = g_j^T s + \tfrac{1}{2} s^T H_j s \\ \text{subject to} \quad & \|N_j s\| \leq \delta_j. \end{aligned} \tag{2.18}$$

The matrix N_j may be used to scale the components of s. If $\|\cdot\|$ is vector norm, then $\|N_j x\|$ is also a vector norm for all nonsingular N_j. In the simplest case, N_j is chosen as the identity matrix. There is no consensus on what choice of N_j is appropriate. Convergence theory allows the choice $N_j = I$ for all j, and this choice seems to be acceptable for well-scaled problems. Choosing N_j to be a constant matrix in order to improve the scaling of the problem is a trivial modification to a trust-region algorithm. However, some methods, particularly those considered in this thesis, must vary N_j at each iteration in order to compute the trial step efficiently (see Chapter 4).

The extreme-value theorem implies that the quadratic model $\mathcal{Q}_j(s)$ achieves its minimum value on the compact set $\mathcal{B}(x_j, \delta_j) = \{\, s : \|N_j s\| \leq \delta_j \,\}$, and hence the trust-region subproblem (2.18) is always well-posed. This means that, in contrast to the line-search problem (2.8), the quadratic model need not be convex.

The behavior of f near the point x_j determines the size of the region in which the quadratic model can be "trusted" to predict the change in f. Thus, trust-region methods systematically adjust the trust-region radius to reflect the accuracy of the previous quadratic model in a nearby region. Upon finding an approximate solution s_j to (2.18), trust-region methods evaluate f at the *trial point* $x_j + s_j$. If the actual reduction in f is within a given factor of the reduction predicted by the quadratic model, then f is assumed to have achieved a *sufficient decrease* and the trial point is accepted as the next iterate. Moreover, if the actual reduction in f is sufficiently larger than predicted, the trust-region radius is increased under

the assumption that a larger trust-region radius will also give an acceptable step in the next iteration. If the trial step does not give a sufficient decrease, the trial step is rejected and the subproblem is solved again with a smaller trust-region radius. This strategy is based on the assumption that the change in f will be better predicted by the quadratic model at points closer to x_j.

Algorithm 2.3.1 below is an example of a basic trust-region algorithm for minimizing f.

ALGORITHM 2.3.1. BASIC TRUST-REGION ALGORITHM

Specify constants $0 < \eta_1 < \eta_2 < 1$, $0 < \eta_1 < \frac{1}{2}$, $0 < \gamma_2 < 1 < \gamma_3$;
Choose x_0; $\quad j = 0$; $\quad \delta_j = 1$;
while not *converged* **do**
  Compute s_j, an approximate solution of $\min\{\mathcal{Q}_j(s) : \|N_j s\| \leq \delta_j\}$;
  $\rho_j = \big(f(x_j + s_j) - f(x_j)\big)/\mathcal{Q}_j(s_j)$;
  if $\rho_j \geq \eta_1$ **then**
    Successful iteration: $x_{j+1} = x_j + s_j$;
    if $\rho_j \geq \eta_2$ **then** $\delta_{j+1} = \max\{\delta_j, \gamma_3\|N_j s_j\|\}$ **else** $\delta_{j+1} = \delta_j$;
  else
    $x_{j+1} = x_j$; $\delta_{j+1} = \gamma_2\|N_j s_j\|$;
  end
  $j \leftarrow j + 1$;
end while

The shape of the trust region is determined by the choice of norm in (2.18). The two most popular choices for the norm are the Euclidean two-norm and the infinity-norm. In two-dimensions, the Euclidean norm corresponds to a circular region and the infinity-norm corresponds to a square region. Although, it is simple to test component-wise if a vector lies inside a trust region defined by the infinity norm, it is relatively easier to compute an approximation solution to a trust-region subproblem defined by the Euclidean norm. In particular, when H_j is indefinite, the infinity-norm trust-region subproblem is a nonconvex quadratic program. For

any nonconvex quadratic program there may exist certain *dead points* at which the second-order necessary conditions hold, but the second-order sufficient conditions do not hold. The verification of such a point as a local minimizer of the subproblem is an NP-hard problem (see Murty and Kabadi [33] and Pardalos and Schnitger [39]), and all quadratic programming methods cannot be expected to verify optimality in a reasonable amount of computational effort. There is still another compelling reason to prefer the Euclidean-norm problem: The *global* minimizer of the trust-region subproblem is completely characterized in the Euclidean norm.

Theorem 2.3.1 (Gay [10].) *Let δ be a given positive constant. A vector s is a global solution of the trust-region subproblem if and only if $\|Ns\| \leq \delta$ and there exists a unique $\sigma \geq 0$ such that*

$$(H + \sigma N^T N)s = -g, \qquad \sigma(\delta - \|Ns\|) = 0, \tag{2.19}$$

with $H + \sigma N^T N$ positive semidefinite. Moreover, if $H + \sigma N^T N$ is positive definite, then the global minimizer is unique. ∎

Not only is this theorem the basis for many trust-region algorithms, but it also provides a convenient measure for the quality of approximate solutions from any type of trust-region subproblem solver. For all of these reasons, we only consider methods that solve the trust-region subproblem defined by the Euclidean norm.

2.3.2 Trust-region methods with a line search

Modern trust-region algorithms avoid the need to re-solve the trust-region subproblem when the trial step does not give a sufficient decrease. These methods use a line search to ensure that f is sufficiently reduced after the solution of *every* trust-region subproblem. Toint [40], and Nocedal and Yuan [36] employ line searches to obtain a positive step length α_j such that $x_{j+1} = x_j + \alpha_j s_j$ gives $f(x_{j+1}) < f(x_j)$. Gertz [11] proposes two trust-region algorithms that use line searches to enforce

popular sufficient decrease criteria. The first algorithm ensures that iterates satisfy an Armijo-type condition.

ALGORITHM 2.3.2. ARMIJO TRUST-REGION ALGORITHM

Specify constants $0 < \eta_1 < \eta_2 < \frac{1}{2}$, $0 < \gamma_2 < 1 < \gamma_3$, $1 \leq \nu \leq 1/\gamma_2$;
Choose x_0; $\quad j = 0$; $\quad \delta_j = 1$;
while not *converged* **do**
  Compute s_j, an approximate solution of $\min \{\mathcal{Q}_j(s) : \|N_j s\| \leq \delta_j\}$;
  $\rho_j = \big(f(x_j + s_j) - f(x_j)\big)/\mathcal{Q}_j^-(s_j)$;
  if $\rho_j \geq \eta_1$ **then**
    Successful iteration: $x_{j+1} = x_j + s_j$;
    if $\rho_j \geq \eta_2$ **then** choose $\delta_{j+1} \in [\delta_j, \max\{\delta_j, \gamma_3 \|N_j s_j\|\}]$ **else** $\delta_{j+1} = \delta_j$;
  else
    Find smallest ℓ in $\{1, 2, \ldots\}$ such that $\alpha_j = \gamma_2^{-\ell}$ satisfies:
      $\big(f(x_j + \alpha_j s_j) - f(x_j)\big)/\mathcal{Q}_j^-(\alpha_j s_j) \geq \eta_1$;
    $x_{j+1} = x_j + \alpha_j s_j$;
    Choose $\delta_{j+1} \in [\alpha_j \|N_j s_j\|, \alpha_j \nu \|N_j s_j\|]$
  end
  $j \leftarrow j + 1$;
end while

If $s_j^T H_j s_j < 0$, the sufficient reduction in f is determined relative to a quadratic model. In this case, the sufficient reduction criterion in Algorithm 2.3.2 is the same as that used in vanilla trust-region methods. If $s_j^T H_j s_j \geq 0$, the sufficient reduction in f is determined relative to a linear model. In this case, the condition that is enforced is the Armijo condition (2.11). In this case, the sufficient reduction criterion is stronger than the criterion used in vanilla trust-region methods.

Gertz [11] also proposes the following Wolfe-type trust-region algorithm:

ALGORITHM 2.3.3. WOLFE TRUST-REGION ALGORITHM

Specify constants $0 < \eta_1 < \omega < 1$;

Choose x_0; $j = 0$; $\delta_j = 1$;
while not *converged* **do**
Compute s_j, an approximate solution of $\min\{\mathcal{Q}_j(s) : \|N_j s\| \leq \delta_j\}$;
Find α_j satisfying the conditions
$f(x_j + \alpha_j s_j) \leq f(x_j) + \eta_1 \mathcal{Q}_j^-(\alpha_j s_j)$ and $|g(x_j + \alpha_j s_j)^T s_j| \leq -\omega \mathcal{Q}_j^{-\prime}(\alpha_j s_j)$;
$x_{j+1} = x_j + \alpha_j s_j$;
Choose $\delta_{j+1} \in [\alpha_j \|N_j s_j\|, \alpha_j \nu \|N_j s_j\|]$
$j \leftarrow j + 1$;
end while

Algorithms 2.3.2 and 2.3.3 use the line search to control the size of the trust region. Intuitively speaking, a good trust-region radius for the subproblem would have been $\delta_j = \alpha_j \|N_j s_j\|$; that is, a basic trust-region method would have accepted the trial step $\alpha_j s_j$ if the trust-region radius been at least of size $\alpha_j \|N_j s_j\|$. It is reasonable to hope that $\mathcal{Q}_{j+1}$ will adequately model decreases in f in a similarly-sized trust region. Thus, the subsequent trust-region radius may be taken to be $\delta_{j+1} = \alpha_j \|N_j s_j\|$. Note that if the exact solution(s) to the trust-region subproblem lies on the boundary of the trust region, it is more accurate to set $\delta_{j+1} = \alpha_j \delta_j$, since s_j can only be computed within some tolerance to satisfy the trust-region constraint, i.e., $\|N_j s_j\| \approx \delta_j$. The scalar ν in both algorithms is used to compensate for this numerical inaccuracy.

Lemma 2.2.1 guarantees the existence of a step that satisfies the Armijo-style condition in Algorithm 2.3.2 whenever the approximate trust-region subproblem solution is a descent direction. Gertz proves the following lemma, which guarantees the existence of steps satisfying the Wolfe conditions used in Algorithm 2.3.3.

Lemma 2.3.1 *Given $f : \mathcal{D} \subseteq \mathbb{R}^n \mapsto \mathbb{R}^1$ be twice continuously differentiable and bounded below. If $g_j^T s_j < 0$ or $g_j^T s_j = 0$ and $s_j^T H_j s_j < 0$ then there are constants $0 < \alpha_l < \alpha_h$ such that α_j satisfies the conditions in Algorithm* 2.3.3 *whenever $\alpha_l \leq \alpha \leq \alpha_h$.* ∎

2.3.3 Convergence of trust-region methods

This section begins by considering *first-order* convergence behavior of trust-region methods. Broadly speaking, "first-order convergence" refers to characterizations of the limit of the gradient, i.e., $\lim_{j\to\infty} g(x_j)$. Typically, the strongest first-order convergence theory gives conditions such that $\lim_{k\to\infty} \|g_j\| \to 0$, regardless of the choice of starting point. However, without additional assumptions about the number of isolated stationary points or the proximity of an iterate to an isolated stationary point, it is not often possible to prove that the sequence of iterates $\{x_j\}$ converges to a stationary point.

Convergence theory for trust-region methods relies on the accuracy of the approximate solution to the trust-region subproblem (2.18). For efficiency, it is important that the trust-region subproblem not be solved exactly. Broadly speaking, it is desirable to compute a solution with as little effort as possible, subject to the requirement that the computed solution does not interfere with the overall convergence of the method.

In trust-region methods, the steepest-descent direction $p = -g(x)$ plays a vital role in defining viable approximate solutions of the subproblem. It will be shown that convergence will not be compromised as long as an approximate s_j gives a reduction in the objective function that is within a fixed factor of the reduction predicted by a constrained steepest-descent step.

We start by considering a simple trust-region algorithm that defines an approximate solution of the trust-region subproblem known as the *Cauchy step* s_j^c. This vector is a solution of the trust-region problem

$$\mathcal{Q}_j(s_j^c) = \min_{s,\alpha} \{\mathcal{Q}_j(s) : s = -\alpha g_j,\ \|s\| \le \delta_j\}. \tag{2.20}$$

Thus, the Cauchy step can be written as $s_j^c = -\alpha_j^c g_j$, where α_j^c is given by

$$\alpha_j^c = \begin{cases} g_j^T g_j / g_j^T H_j g_j & \text{if } g_j^T g_j / g_j^T H_j g_j \le \delta_j / \|g_j\| \text{ and } g_j^T H_j g_j > 0; \\ \delta_j / \|g_j\| & \text{otherwise.} \end{cases}$$

If $g_j \neq 0$, the Cauchy step can be interpreted as a constrained steepest-descent step. The choice of norm here is arbitrary, but whatever norm is chosen, it affects only the *scaling* of the Cauchy step, not its direction.

The next lemma shows that if the Cauchy step (as an approximate solution to the trust-region subproblem), is sufficiently accurate to ensure the convergence of the trust-region method.

Lemma 2.3.2 (Powell [41, 42, 43].) *Given any norm $\|\cdot\|$, let κ be a constant such that $\|s\|_2 \geq \kappa\|s\|$ for all s. If s_j^c is the Cauchy step, i.e., the solution of* (2.20)*, then*

$$\mathcal{Q}_j(s_j^c) \leq -\tfrac{1}{2}\kappa^2\|g_j\| \min\{\delta_j, \|g_j\|/\|B_j\|_2\}. \tag{2.21}$$

Proof. First we consider the case $g_j^T B_j g_j > 0$. Let p_j denote the steepest-descent direction $p_j = -g_j$. From the definition of $\mathcal{Q}_j$ we have $\mathcal{Q}_j(\alpha p_j) = -\alpha g_j^T g_j + \frac{1}{2}\alpha^2 g_j^T B_j g_j$. Since $g_j^T B_j g_j > 0$ by assumption, a unique unconstrained minimizer of $\mathcal{Q}_j(\alpha p_j)$ exists and is given by $\alpha^* = g_j^T g_j / g_j^T B_j g_j$. The change in the objective by this step is

$$\mathcal{Q}_j(\alpha^* p_j) = -\alpha^* g_j^T g_j + \tfrac{1}{2}(\alpha^*)^2 g_j^T B_j g_j = -\tfrac{1}{2} g_j^T g_j \left(\frac{g_j^T g_j}{g_j^T B_j g_j}\right) \leq -\tfrac{1}{2}\kappa^2 \|g_j\|^2/\|B_j\|_2.$$

If $g_j^T g_j / g_j^T B_j g_j \leq \delta_j/\|g_j\|$, then $\|\alpha^* g_j\| \leq \delta_j$ and the minimizer subject to the trust region constraint is $\alpha^c = \alpha^*$. If $g_j^T g_j / g_j^T B_j g_j > \delta_j/\|g_j\|$, then $\alpha_j^c = \delta_j/\|g_j\| < \alpha^*$ and the corresponding predicted change is

$$\begin{aligned}\mathcal{Q}_j(s_j^c) &= -g_j^T g_j \alpha^c + \tfrac{1}{2}(\alpha^c)^2 g_j^T B_j g_j \\ &\leq -g_j^T g_j \alpha^c + \tfrac{1}{2}\alpha^* \alpha^c g_j^T B_j g_j = -\tfrac{1}{2} g_j^T g_j \left(\frac{\delta_j}{\|g_j\|}\right) \leq -\tfrac{1}{2}\kappa^2 \|g_j\| \delta_j.\end{aligned}$$

If $g_j^T B_j g_j \leq 0$, no unconstrained minimizer exists and $\alpha^c = \delta_j/\|g_j\|$. In this case

$$\mathcal{Q}_j(s_j^c) = -\alpha^c g_j^T g_j + \tfrac{1}{2}(\alpha^c)^2 g_j^T B_j g_j \leq -\alpha^c g_j^T g_j = -g_j^T g_j \left(\frac{\delta_j}{\|g_j\|}\right) \leq -\tfrac{1}{2}\kappa^2 \|g_j\| \delta_j.$$

The result now follows by combining the inequalities from each of these cases. ∎

As long as $\|g_j\|$ is nonzero the trust-region algorithm defined in Algorithm 2.3.1 will eventually compute a trust-region radius δ_j such that the objective value $f(x_j+s_j)$ satisfies the sufficient decrease condition $f(x_j)-f(x_j+s_j) \geq -\eta_1 \mathcal{Q}_j(s_j)$. Moreover, for any approximate solution s_j such that $\mathcal{Q}_j(s_j) \leq \mathcal{Q}_j(s_j^c)$, Lemma 2.3.2 gives the following upper bound on $\mathcal{Q}_j(s_j)$:

$$\mathcal{Q}_j(s_j) \leq -\tfrac{1}{2}\eta_1\kappa^2\|g_j\| \min\{\delta_j, \|g_j\|/\|H_j\|\}.$$

Of all the results on first-order convergence, the next result requires the weakest conditions for convergence.

Theorem 2.3.2 (Powell [43].) *Let $f\colon \mathcal{D} \subseteq \mathbb{R}^n \mapsto \mathbb{R}$ be continuously differentiable on the open convex set $\mathcal{D}$. Let $\{x_j\} \subset \mathcal{D}$ be a sequence of iterates generated by the basic trust-region method of Algorithm* 2.3.1. *Assume that an approximate solution s_j of the trust-region subproblem* (2.18) *satisfies*

$$\mathcal{Q}_j(s_j) \leq -\tau\|g_j\| \min\{\delta_j, \|g_j\|/\|H_j\|\} \quad \textit{and} \quad \|s_j\| \leq \delta_j, \tag{2.22}$$

for some $\tau > 0$. Assume further that $\{\|H_j\|\}$ is bounded above. If f is bounded below in $\mathcal{D}$, then either $\liminf_{k\to\infty} \|g_j\| = 0$ or some x_j satisfies the algorithm's convergence criterion and the algorithm terminates. ∎

Lemma 2.3.2 provides a way to determine τ. (For example, if $\|\cdot\|$ is the Euclidean norm then $\tau < 1/2$.)

It is possible to derive stronger results when g is uniformly continuous in some region containing the iterates $\{x_j\}$. In particular, if the iterates lie in a compact region or $H(x)$ is bounded, then $\lim_{k\to\infty} \|g_j\| = 0$ (for more details, see Thomas [49]).

Theorem 2.3.3 *Let all the assumptions of Theorem* 2.3.2 *hold. Assume further that the sequence of iterates $\{x_j\}$ lies in a region in which $g(x)$ is uniformly continuous. Then Algorithm* 2.3.1 *will either terminate at a point that satisfies its convergence criterion, or $\lim_{k\to\infty} \|g_j\| = 0$.* ∎

Gertz [11] proves that line searches in Algorithms 2.3.2 and 2.3.3 do not interfere with the first-order convergence properties of basic trust-region method.

Theorem 2.3.4 *Let all the assumptions of Theorem* 2.3.3 *hold. If the solution to the trust-region subproblem* s_j *always satisfies* $g_j^T s_j \leq 0$*, then Algorithms* 2.3.2 *and* 2.3.3 *will either terminate at a point that satisfies their convergence criterion, or* $\lim_{k\to\infty} \|g_j\| = 0$. ∎

Obviously, no stronger results (such as convergence to a minimizer of f) can be proved without utilizing curvature information, since we know from Section 2.1 that a minimizer is distinguished from other stationary points by (at the least) a positive semidefinite Hessian. Nonetheless, in practice the strategy of reducing f at every iteration means that descent methods usually converge to minimizers.

Second-order methods seek limit points that satisfy *second-order* necessary conditions. Line-search methods based on a modified Newton method are unable to progress beyond a stationary point that is not a local minimizer; that is, the solution to the modified Newton equation at any stationary point will always be zero. Thus, a second-order line search must be able to use second-order information to move away from local maximizers and saddle points. It is relatively easy to implement a second-order line search that uses directions of negative curvature to generate a sequence $\{x_j\}$ such that

$$g_j \to 0 \quad \text{and} \quad \min\{\lambda_n(H_j), 0\} \to 0,$$

where $\lambda_n(H)$ denotes the smallest eigenvalue of H. Clearly, if $\{x_j\}$ converges, it will converge to a point that satisfies the second-order necessary conditions for a local minimizer of f.

Convergence results for trust-region methods depend on the quality of the approximate solution to the trust-region subproblem (2.18). The requirements for second-order convergence should be no weaker than the requirements for first-order

convergence, i.e.,

$$\mathcal{Q}_j(s_j) \leq \mathcal{Q}_j(s_j^c) = \min_{s,\alpha} \{\mathcal{Q}_j(s) : s = -\alpha g_j,\ \|s\| \leq \delta_j\}. \tag{2.23}$$

The following second-order convergence theorem and proof are adapted from Moré and Sorensen [32].

Theorem 2.3.5 *Let $f(x)$ be twice continuously differentiable in an open convex subset $\mathcal{D}$ of $\mathbb{R}^n$, and assume that $x_0 \in \mathcal{D}$ is chosen so that the level set $L(f(x_0))$ is compact. Suppose $\lambda_{\min}(N_j^T N_j) \geq m > 0$ for all k. And, suppose the sequence $\{x_j\}$ is defined by Algorithm* 2.3.1 *and s_j is an approximate solution to the subproblem* (2.18) *such that*

$$-\mathcal{Q}_j(s_j) \geq \beta_1 |\mathcal{Q}_j^*| \quad \textit{with} \quad \|N_j s_j\| \leq \beta_2 \delta_j, \tag{2.24}$$

where $\mathcal{Q}_j^*$ denotes the unique global minimum of (2.18) *and β_1 and β_2 are strictly positive constants. Then, either the algorithm terminates at $x_l \in L$ with $g(x_l) = 0$ and $H(x_l)$ positive semidefinite, or $\{x_j\}$ has a limit point $x^* \in L$ with $g(x^*) = 0$ and $H(x^*)$ positive semidefinite.*

Proof. If $g(x_l) = 0$ and $H(x_l)$ is positive semidefinite for some iterate $x_l \in L$, the algorithm terminates with $s_l = 0$. Otherwise, $\mathcal{Q}_j(s_j) < 0$ for all $k > 0$, and thus $\{x_j\}$ is well defined and lies in $L(f(x_0))$.

First, assume that a subsequence of $\{\sigma_j\}$ converges to zero. The compactness of $L(f(x_0))$ shows that the same subsequence of $\{x_j\}$ converges to $x^* \in L(f(x_0))$. By Theorem 2.3.1, there exists some σ_j that $H_j + \sigma_j N_j^T N_j$ is positive semidefinite. Moreover, $H(x^*)$ must be converging to a positive semidefinite matrix for the given subsequence. Let R_j be the Cholesky factor of $H_j + \sigma_j N_j^T N$, so that

$$R_j^T R_j s_j^* = (H_j + \sigma_j N_j^T N_j) s_j^* = -g_j. \tag{2.25}$$

To show that $g(x^*) = 0$, note that, using (2.25) and norm inequalities, we have

$$\|R_j s_j^*\|^2 \geq \frac{\|g_j\|^2}{\|H_j\| + \sigma_j \|N_j^T N_j\|}. \tag{2.26}$$

Finally, the descent condition in Algorithm 2.3.1 ensures that

$$\begin{aligned} f(x_k) - f(x_{j+1}) &\geq -\eta_1 \mathcal{Q}_j(s_j) \\ &\geq \eta_1 |\mathcal{Q}_j^*| \qquad (2.27) \\ &= \eta_1 \left| g_j^T s_j^* + \tfrac{1}{2} s_j^{*T} (H_j + \sigma_j N_j^T N_j) s_j^* - \tfrac{1}{2}\sigma_j \|(s_j^*)^T N_j^T N_j s_j^*\|^2 \right| \\ &= \eta_1 |\tfrac{1}{2} g_j^T s_j^* - \tfrac{1}{2}\sigma_j \delta_j^2| \\ &= \eta_1 \|R_j s_j^*\|^2 + \tfrac{1}{2}\sigma_j \delta_j^2. \qquad (2.28) \end{aligned}$$

Since $\{f(x_j)\}$ is a decreasing sequence that is bounded below, inequality (2.28) shows that $\|R_j s_j\|$ converges to zero. It follows from (2.26) that g_j converges to zero for the given subsequence.

It now remains to show that $\{\sigma_j\}$ cannot remain bounded away from zero. To do so, assume the contrary, i.e., that for some $\epsilon > 0$,

$$\sigma_j \geq \epsilon \qquad \text{for all } k. \qquad (2.29)$$

The string of inequalities from (2.27) to (2.28) show that

$$-\mathcal{Q}_j(s_j) \geq \beta_1 (\|R_j s_j\|^2 + \tfrac{1}{2}\sigma_j \delta_j^2),$$

and hence,

$$-\mathcal{Q}_j(s_j) \geq \tfrac{1}{2}\beta_1 \sigma_j \delta_j^2 \geq \tfrac{1}{2} \left(\frac{\beta_1}{\beta_2^2} \right) \epsilon \|N_j s_j\|^2. \qquad (2.30)$$

The second-order mean-value theorem gives the standard estimate

$$\|f(x_j + s_j) - f(x_j) - \mathcal{Q}_j(s_j)\| \leq \tfrac{1}{2}\|s_j\|^2 \max_{0 \leq \xi \leq 1} \|H(x_j + \xi s_j) - H_j\|.$$

Combining this equation and (2.30), we have

$$|\rho_j - 1| \leq \frac{1}{m^2} \left(\frac{\beta_2^2}{\beta_1 \epsilon} \right) \max_{0 \leq \xi \leq 1} \|H(x_j + \xi s_j) - H(x_j)\| \qquad (2.31)$$

where $\rho_j = \big(f(x_j + s_j) - f(x_j)\big) / \mathcal{Q}_j(s_j)$.

Since f is bounded below in $L(f(x_0))$, inequalities (2.28) and (2.29) imply that $\{\delta_j\}$ converges to zero, and hence $\{\|N_j s_j\|\}$ also converges to zero. Since $\eta_2 < 1$ in

Algorithm 2.3.1, and the constants β_1, β_2, ϵ, and $1/m^2$ are bounded, then (2.31) and the continuity of $H(x)$ on the compact region $L(f(x_0))$ imply that $\rho_j > \eta_2$ for all k sufficiently large. The updating rules for δ_j in Algorithm 2.3.1 then imply that $\{\delta_j\}$ is bounded away from zero. However, this contradicts the fact that $\{\delta_j\}$ converges to zero, and hence (2.29) must be false. ∎

Gertz [11] shows that the line search in Algorithms 2.3.2 and 2.3.3 does not interfere with the second-order convergence properties of basic trust-region method.

The remainder of this chapter is concerned with solving the trust-region subproblem and so the suffix j is omitted.

2.3.4 Solving the trust-region subproblem

There are two general approaches to finding an approximate solution of the trust-region subproblem (2.18). The first approach is to proceed with the solution of the unconstrained problem and consider the constraint only if the unconstrained solution appears to lie outside the trust-region. The class of dog-leg methods are of this type (see, e.g., Shultz, Schnabel and Byrd [47] and Byrd, Schnabel and Shultz [3]), as are the methods considered in this thesis. The second approach is to start with the equality-constraint problem

$$\underset{s\in\mathbb{R}^n}{\text{minimize}} \;\; g^T s + \tfrac{1}{2} s^T H s \quad \text{subject to} \;\; \|Ns\| = \delta. \tag{2.32}$$

and switch to the Newton step if it appears that the optimal σ (of Theorem 2.3.1) is zero. Methods of this type include those of Gay [10] and Moré and Sorensen [32].

Methods based on solving (2.32) attempt to find a root of the nonlinear equation $\|N(H+\sigma N^T N)^{-1} g\| = \delta$, such that σ is in the interval $(-\lambda_n, \infty)$, where λ_n denotes the left-most generalized eigenvalue for the matrix pair $(H, N^T N)$. Each step involves solving a positive-definite system $(H+\sigma N^T N)s = -g$ for a given σ. If the iterates appear to be converging to a negative root in $(-\lambda_n, \infty)$, the value $\sigma = 0$ is selected, implying that the Newton step lies inside the trust-region.

The trust-region problem is said to be *degenerate* if $\|Ns_L\| < \delta$, where s_L is the least-length solution of $(H - \lambda_n N^T N)s = -g$ (i.e., $s_L = -(H - \lambda_n N^T N)^\dagger g$). In the degenerate case, there are two situations to consider. If λ_n is positive, the quantities $\sigma = 0$ and $s = -H^{-1}g$ satisfy the optimality conditions (Theorem 2.3.1), since $\|Ns\| < \|Ns_L\| < \delta$. Alternatively, if λ_n is negative or zero, the system $(H + \sigma N^T N)s = -g$ cannot be used alone to determine the optimal s. In this case, the generalized eigenvector u_n is a null vector of $H - \lambda_n N^T N$, and there exists a scalar τ such that

$$(H - \lambda_n N^T N)(s_L + \tau u_n) = -g \quad \text{and} \quad \|N(s_L + \tau u_n)\| = \delta. \tag{2.33}$$

Taking $\sigma = -\lambda_n$ and $s = s_L + \tau u_n$ satisfies the optimality conditions (Theorem 2.3.1), and thus, constitutes a global solution of the trust-region subproblem.

2.3.5 The Moré-Sorensen algorithm

The Moré-Sorensen algorithm [32] is a *direct* method for solving the trust-region subproblem in the sense that each iteration involves an explicit matrix factorization. The algorithm approximates a global solution to the trust-region subproblem by finding a point s such that

$$\mathcal{Q}(s) \leq \tau \mathcal{Q}^*,$$

where $\mathcal{Q}^*$ is the optimal value of $\mathcal{Q}(s)$ and τ is a scalar such that $0 < \tau < 1$.

If s lies in the interior of the trust-region, then H is positive definite and $\sigma = 0$ satisfies the optimality conditions. On the other hand, if a solution s lies on the boundary of the trust-region, there are two important cases to consider. If g is not perpendicular to the eigenspace associated with λ_n, then there is an optimal solution pair (s, σ) with $\sigma > 0$ that satisfies

$$s = -(H + \sigma N^T N)^{-1} g \quad \text{and} \quad \sigma \in [-\lambda_n, \infty). \tag{2.34}$$

The so-called "hard case" occurs when g is perpendicular to the eigenspace associated with the generalized eigenvalue λ_n and there is no solution (s, σ) with

$\sigma > 0$ that satisfies (2.34). In other words, the so-called hard case occurs if $H+\sigma N^TN$ is positive definite and there no solution s on the boundary that satisfies $(H + \sigma N^TN)s = -g$. This corresponds to the degenerate case with $\lambda_n \leq 0$, and thus, a global solution to the trust-region subproblem is given by (2.33). In theory, this case should not occur often since it requires that both g is orthogonal to the eigenspace associated with the generalized eigenvalue λ_n and H must be indefinite.

One approach to approximating an optimal solution that lies on the boundary of the trust region is to find a zero of the equation $\psi(\sigma) = \|Ns_\sigma\| - \delta$, such that $s = -(H + \sigma N^TN)^{-1}g$. However, it turns out that ψ is nonlinear; in particular, ψ has poles. Instead, Moré and Sorensen focus on solving

$$\phi(\sigma) = \frac{1}{\delta} - \frac{1}{\|Np_\sigma\|}, \tag{2.35}$$

where p_σ satisfies $(H + \sigma N^TN)p_\sigma = -g$.

It is worth noting that Newton's method is particularly well-suited for finding a root of (2.35) due in part to the following properties of ϕ:

Lemma 2.3.3 *Suppose ϕ is defined as in* (2.35).

(i) *$\phi(\sigma)$ is twice-continuously differentiable on $(-\lambda_n, \infty)$;*

(ii) *$\phi(\sigma)$ is strictly decreasing on $(-\lambda_n, \infty)$;*

(iii) *$\phi(\sigma)$ is strictly convex on $(-\lambda_n, \infty)$;*

(iv) *if $\lim_{\sigma\to-\lambda_{n^+}} \phi(\sigma) > 0$ then $\phi(\sigma)$ has a unique zero in $(-\lambda_n, \infty)$;*

(v) *if $\lim_{\sigma\to-\lambda_{n^+}} \phi(\sigma) \leq 0$ if and only if the linear system $(H - \lambda_n N^TN)s = -g$ is compatible.*

Proof. See Gay [10] and Moré and Sorensen [32]. ∎

Upon finding a new approximation to σ^*, the Cholesky decomposition of $H + \sigma N^TN$ is used to obtain a solution to the system

$$(H + \sigma N^TN)p = -g, \tag{2.36}$$

and (if required) an approximate null vector z is computed. And, a safe-guarded Newton scheme is used to find a root of (2.35) and ensure that σ remains within $(-\lambda_n, \infty)$.

An optimal trust-region subproblem solution may be written as solutions $s = p+z$, where $p = s_L$ and $z = \tau u_n$ in the hard case (see (2.33)), and $z = 0$ otherwise. The following lemma shows that if the pair (s, σ) is sufficiently close to satisfying the optimality conditions, then the pair satisfies the inequalities in (2.24) required for second-order convergence.

Lemma 2.3.4 *Let ϵ be any scalar such that $0 < \epsilon < 1$. Let σ be a nonnegative scalar such that $H + \sigma N^T N$ is positive semidefinite. Suppose that $s = p + z$, where p satisfies*

$$(H + \sigma N^T N)p = -g, \tag{2.37}$$

and z is a (possibly zero) vector satisfying the conditions

$$z^T H z \leq \epsilon(2-\epsilon)(p^T H p + \sigma\delta^2), \quad \textit{and} \quad \|N(p+z)\| \leq (1+\epsilon)\delta.$$

Then s satisfies the inequality

$$\mathcal{Q}(s) \leq \tau \mathcal{Q}^* \quad \textit{and} \quad \|Ns\| \leq \bar{\delta}, \tag{2.38}$$

where $\tau = \left((1-\epsilon)/(1+\epsilon)\right)^2$, $\bar{\delta} = (1+\epsilon)\delta$, and $\mathcal{Q}^* = \min\{\mathcal{Q}(s) : \|Ns\| \leq \bar{\delta}\}$.

Proof. See Moré and Sorensen [32] and Gertz [11]. ∎

Given constants $c_1, c_2 \in (0, 1)$, Moré and Sorensen prove convergence to an approximate solution $\bar{s}$ satisfying

$$\mathcal{Q}(\bar{s}) - \mathcal{Q}^* \leq c_1(2-c_1)\max(|\mathcal{Q}^*|, c_2), \quad \text{and} \quad \|N\bar{s}\| \leq (1+c_1)\delta, \tag{2.39}$$

where $\mathcal{Q}^*$ denotes the unique global minimum of the trust-region subproblem. Moré and Sorensen also prove second-order convergence by meeting the conditions of Lemma 2.3.4, and thus, satisfying equation (2.38). In the context of unconstrained optimization, the Moré-Sorensen algorithm is guaranteed to converge to points that satisfy the first-order and second-order necessary conditions for optimality.

3

Iterative Methods for Minimizing a Quadratic Function

3.1 Motivation for Iterative Methods

As the number of variables increase, the time and storage associated with factoring large sparse matrices becomes prohibitive. Thus, direct methods such as Moré and Sorensen's method are only suitable for problems for which n is sufficiently small or H is sufficiently sparse.

In large-scale optimization, iterative methods are used to help speed up computations and economize storage requirements. Iterative trust-region methods are primarily concerned with *approximating* trust-region subproblem solutions quickly and efficiently. These methods avoid forming large dense matrices and must assume that sparse matrices are generally not stored as a full matrix. Computationally speaking, iterative trust-region methods must rely on the ability to access H only through matrix-vector products.

More economical iterative trust-region methods also try to minimize the number of (possibly very costly) function evaluations, as well as the required linear algebra, i.e., the number of matrix-vector products with H. However, there is a

well-known trade-off: Increasing the accuracy of the approximate trust-region subproblem solution usually decreases the number of required trust-region subproblem solves, but increases the cost of solving each subproblem. Practically speaking, the result is a method that requires fewer function evaluations but more matrix-vector products with H. Ultimately, there is a point at which increasing the accuracy of the subproblem solver is unproductive because the number of required subproblem solves is at a minimum.

If H is positive definite, then the unique unconstrained minimizer s^* of the quadratic function $\mathcal{Q}(s) = g^T s + \frac{1}{2} s^T H s$ satisfies the positive-definite linear system $Hs^* = -g$. This implies that any iterative method for minimizing a quadratic implicitly defines an associated method for solving a positive-definite linear system of equations $Ax = b$. The method of conjugate gradients (CG) is one of the most widely used iterative methods for solving positive-definite systems $Ax = b$. In order to emphasize the link between minimization and solving positive-definite linear equations, we will consider the following general quadratic function:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \;\; q(x), \quad \text{where} \quad q(x) = \tfrac{1}{2} x^T A x - b^T x. \tag{3.1}$$

The main intention of this chapter is to discuss methods that solve $Ax = b$ without directly accessing the elements of A or computing its factorization. In particular, the matrix A need be available only as an *operator*, i.e., it is used only to compute a matrix-vector product Av for a given v.

The methods to be discussed have the property that, in exact arithmetic, the function q is minimized in a finite number of steps. In particular, there is sequence of iterates x_0, x_1, $\ldots$, x_m with $m \leq n$, such that $q(x_{k+1}) < q(x_k)$ and $x_m = x^*$. We will see that the number m is independent of the initial point x_0, except in the special situation where b is an eigenvector of A (see Section 3.4.4). In practice, computer arithmetic is not exact and the methods to be discussed may need far more iterations, or far fewer if the eigenvalues of A are clustered into groups.

This section solely approximates the solution to minimization problems of the

form in (3.1); thus, in this section, $g(x)$ will denote the vector $\nabla q(x)$, and g_k will denote $g(x_k)$ for simplicity.

3.2 Minimization Over Expanding Subspaces

A fundamental idea in the development of optimization methods is that of minimizing a function over a *sequence of expanding subspaces*. More precisely, given the points x_0, x_1, x_2 ..., x_k, we seek the point x_{k+1} that minimizes the objective function over all vectors x such that

$$x \in x_k + \text{span}\{p_0, p_1, \ldots, p_k\}, \tag{3.2}$$

where p_0, p_1, ..., p_k form a set of linearly independent vectors to be determined. (Although it is commonly referred to as a sequence of expanding subspaces, the set $x_k + \text{span}\{p_0, p_1, \ldots, p_k\}$ is really a *manifold*, i.e., a subspace shifted by the vector x_0.). Let $\mathcal{P}_k$ denote $\text{span}\{p_0, p_1, \ldots, p_k\}$, the subspace of $\mathbb{R}^n$ consisting of all linear combinations of the vectors $\{p_i\}$. Similarly, let P_k be the matrix $P_k = (\, p_0 \;\; p_1 \;\; \cdots \;\; p_k \,)$ whose columns form a basis for $\mathcal{P}_k$. Then, every vector x on the manifold (3.2) may be written uniquely in the form $x_k + P_k y$ for some vector y. The point x_{k+1} that minimizes $q(x)$ on the manifold may be written in the form:

$$x_{k+1} = x_k + P_k y_k, \quad \text{where} \quad y_k = \underset{y \in \mathbb{R}^{k+1}}{\text{argmin}} \;\; q(x_k + P_k y). \tag{3.3}$$

If each iterate x_i is determined in this way, it follows immediately that this algorithm must minimize q in at most n iterations, since the linear independence assumption for the n vectors p_0, p_1, ..., p_{n-1} implies that x_n must minimize $q(x)$ over all $x \in \mathbb{R}^n$. Of course, as k increases, the subspace dimension increases, and eventually, the cost of solving the subproblem will be comparable to that of solving the original problem. The idea is to use information gained while minimizing over $\mathcal{P}_{k-1}$ to accelerate the minimization over $\mathcal{P}_k$.

3.2.1 Solving the reduced problem

Consider the following unconstrained minimization problem:

$$\underset{y\in\mathbb{R}^n}{\text{minimize}} \;\; q(x_k + P_k y) = q(x_k) + y^T P_k^T \nabla q(x_k) + \frac{1}{2} y^T P_k^T A P_k y. \tag{3.4}$$

First order optimality conditions imply that for fixed x_k, the optimal y must satisfy $\nabla_y q(x_k + P_k y_k) = 0$, i.e., y_k satisfies the system

$$P_k^T A P_k y_k = -P_k^T g_k. \tag{3.5}$$

This is an example of a *reduced system* that involves the solution of an optimization problem restricted to a subspace of $\mathbb{R}^n$. The (necessarily positive-definite) matrix $P_k^T A P_k$ is called the *reduced Hessian* and the vector $P_k^T g_k$ is the *reduced gradient*.

The iterates generated by this scheme have a number of interesting properties. First, g_{k+1}, the gradient at x_{k+1}, is orthogonal to all of the vectors $\{p_i\}$ that define P_k. To see this, using the definition of $g(x)$ and x_{k+1}, we obtain

$$P_k^T g_{k+1} = P_k^T (\nabla q(x_{k+1})) = P_k^T g_k - P_k^T A P_k (P_k^T A P_k)^{-1} P_k^T g_k = 0.$$

It follows that $g_{k+1}^T p_i = 0$ for $i = 0, \ldots, k$, and a simple inductive argument may be used to establish the identity:

$$g_j^T p_i = 0 \quad \text{for } i < j \text{ and } 0 \le j \le k. \tag{3.6}$$

These relations imply that the reduced gradient is simply a multiple of the unit vector e_{k+1}, with $P_k^T g_k = (p_k^T g_k) e_{k+1}$. Hence, the system (3.5) for y_k simplifies to

$$P_k^T A P_k y_k = -(p_k^T g_k) e_{k+1}. \tag{3.7}$$

One way of solving these equations is to choose the directions $\{p_k\}$ in such a way that x_{k+1} is easily computed from x_k. Suppose that the $k+1$ vectors $\{p_0, p_1, \ldots, p_k\}$ are mutually conjugate with respect to the matrix A, i.e., it holds that

$$p_i^T A p_j = 0 \quad \text{for} \quad i \neq j \quad \text{and} \quad 0 \le i, j \le k. \tag{3.8}$$

The vectors $\{p_i\}$ are called a set of *conjugate directions*. When A is positive definite, an important property of conjugate directions is that they are necessarily linearly independent.

If the vectors $\{p_i\}$ are conjugate, then the solution of the (diagonal) system (3.7) is the unit vector $y_k = -(g_k^T p_k / p_k^T A p_k) e_{k+1}$. If the last component of y_k is denoted by α_k, we obtain the simple recurrence:

$$x_{k+1} = x_k + \alpha_k p_k, \quad \text{where} \quad \alpha_k = -g_k^T p_k / p_k^T A p_k. \tag{3.9}$$

The value of α_k is the step to the minimizer of q along p_k. Thus (3.9) indicates that the $(k+1)$th vector p_k may be considered as a *search direction* in the usual model algorithm for unconstrained optimization. Note that this implies that

$$x_{k+1} = x_k + \alpha_k p_k = x_0 + \alpha_0 p_0 + \alpha_1 p_1 + \cdots + \alpha_k p_k.$$

so that x_{k+1} is the unique linear combination of x_0 and the $k+1$ vectors p_0, p_1, $\ldots$, p_k that minimizes q.

The preceeding discussion may be summarized the following theorem.

Theorem 3.2.1 *Assume that A is an $n \times n$ symmetric positive-definite matrix and b is an n-vector. Let $\{p_0, p_1, \ldots, p_m\}$ be a set of $m+1$ vectors such that $p_i^T A p_j = 0$ for $i \neq j$, $0 \leq i, j \leq m$. If $\{x_k\}$ is the sequence of minimizers of $q(x)$ on the sequence of expanding manifolds $x_k + \mathcal{P}_k$, the following holds for all $0 \leq k \leq m$:*

(i) *The vectors $\{p_j\}_{j=0}^m$ are linearly independent;*

(ii) *$x_{k+1} = x_k + \alpha_k p_k$, where $\alpha_k = -g_k^T p_k / p_k^T A p_k$;*

(iii) *$g_{k+1}^T p_i = 0$, $0 \leq i \leq k$.* ∎

3.3 Generating Conjugate Directions

Suppose that a set of k mutually conjugate vectors p_0, p_1, ..., p_{k-1} is known at the start of the kth step. We seek a new vector p_k such that

$$p_k^T A p_j = 0 \quad \text{for} \quad 0 \leq j \leq k-1.$$

An appropriate set of directions may be defined by taking $p_0 = -g_0$ and computing p_k as a linear combination of g_k and the previous k directions, i.e.,

$$p_k = -g_k + \sum_{j=0}^{k-1} \beta_{kj} p_j. \tag{3.10}$$

The definition p_k from (3.10) implies that g_k is a linear combination of p_0, p_1, ..., p_k, and thus for $1 \leq i \leq k$, it must hold that $g_k \in \mathcal{P}_k$. Since $p_0 = -g_0$, it also holds trivially that $g_0 \in \mathcal{P}_k$. Therefore, from (3.6)

$$g_k^T g_i = 0 \quad \text{for} \quad 0 \leq i \leq k-1. \tag{3.11}$$

The derivation of the required values of β_{kj} uses the important property of subspace minimization methods whereby the gradients are mutually orthogonal when p_k is defined by (3.10). This property is used to show that $\beta_{k,0}$, $\beta_{k,1}$, ..., $\beta_{k,k-1}$ of (3.10) can be chosen so that p_k is conjugate to p_0, p_1, ..., p_{k-1}. Premultiplying (3.10) by $p_i^T A$ and using the previous conjugacy conditions (3.8) yields

$$p_i^T A p_k = -p_i^T A g_k + \sum_{j=0}^{k-1} \beta_{kj}\, p_i^T A p_j = -p_i^T A g_k + \beta_{ki}\, p_i^T A p_i. \tag{3.12}$$

From the definition of the gradient of $q(x)$, we have

$$g_{i+1} - g_i = A(x_{i+1} - x_i) = \alpha_i A p_i,$$

and the conjugacy conditions (3.11) imply that $-p_i^T A g_k = (g_{i+1} - g_i)^T g_k / \alpha_i = 0$ for $i < k-1$. Thus, to make p_k conjugate to p_i for $i < k-1$, we can simply choose β_{ki} to be zero in (3.12). Since only $\beta_{k,k-1}$ is nonzero, the first subscript on β may

be dropped, so that $\beta_{k-1} \triangleq \beta_{k,k-1}$. To obtain the value of β_{k-1} that makes p_k conjugate to p_{k-1}, we premultiply (3.10) by $p_{k-1}^T A$ and impose the orthogonality condition $p_{k-1}^T A p_k = 0$; this gives $0 = -p_{k-1}^T A g_k + \beta_{k-1} p_{k-1}^T A p_{k-1}$, or, equivalently,

$$\beta_{k-1} = p_{k-1}^T A g_k / p_{k-1}^T A p_{k-1}.$$

Therefore p_k may be written as

$$p_k = -g_k + \beta_{k-1} p_{k-1},$$

where $\beta_{k-1} = p_{k-1}^T A g_k / p_{k-1}^T A p_{k-1}$. The orthogonality of the gradients and the definition of p_k implies the following (equivalent) definitions of β_{k-1}:

$$\beta_{k-1} = p_{k-1}^T A g_k / \|g_{k-1}\|_2^2, \quad \text{or} \quad \beta_{k-1} = \|g_k\|_2^2 / \|g_{k-1}\|_2^2. \tag{3.13}$$

The conjugate-direction method based on these recurrences is known as the *conjugate-gradient method*. A vanilla conjugate-gradient method for minimizing q is given in Algorithm 3.3.1.

Algorithm 3.3.1. Vanilla Conjugate-Gradient Method

Choose x_0; Set $g_0 = g(x_0)$; $k = 0$;
while $g_k \neq 0$ **do**
  if $k = 0$ **then**
    $p_k = -g_k$;
  else
    $\beta_{k-1} = g_k^T [A p_{k-1}] / p_{k-1}^T [A p_{k-1}]$;
    $p_k = -g_k + \beta_{k-1} p_{k-1}$;
  end if
  $[A p_k] = A p_k$; $\alpha_k = -g_k^T p_k / p_k^T [A p_k]$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $g_{k+1} = g_k + \alpha_k [A p_k]$;
  $k = k + 1$;
end do

Note that the relations

$$g_{k+1} = \nabla q(x_{k+1}) = \nabla q(x_k + \alpha_k p_k) = g_k + \alpha_k A p_k,$$

allow us to update the gradient rather than compute it from scratch. With this refinement, the vector Ap_k may be stored until p_k is updated, which allows the conjugate-gradient method to be implemented with one matrix-vector product each iteration.

The next result states that in exact arithmetic the conjugate-gradient method will terminate in a finite number of steps.

Theorem 3.3.1 *If A has r distinct eigenvalues, the conjugate-gradient method will compute the solution in at most r iterations.*

Proof. See Conn, Gould and Toint [4]. ∎

Unfortunately, Theorem 3.3.1 only holds in exact arithmetic; finite termination relies on the orthogonality of the conjugate directions. In finite-precision arithmetic the conjugate-gradient method may not converge in r iterations or even in a finite number of iterations. The following theorem is arguably the most well-known convergence result for the method of conjugates gradients:

Theorem 3.3.2 *Suppose A is a symmetric positive-definite matrix, and x^* is the unique minimizer of $q(x)$. Then, the sequence of iterates $\{x_k\}$ generated by Algorithm* 3.3.1 *satisfies the inequality*

$$\|x^* - x_k\|_A \leq 2\|x^* - x_0\|_A \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k$$

where $\kappa_2(A)$ denotes the condition number of A in the two-norm and $\|\cdot\|_A$ denotes the elliptic *norm, i.e., $\|x\|_A \equiv \sqrt{x^T A x}$ for all $x \in \mathbb{R}^n$.*

Proof. See Luenberger [27]. ∎

Notice that the conjugate-gradient method converges faster when $\kappa_2(A) \approx 1$. This observation is the basis for *preconditioned conjugate-gradient methods*, which apply the conjugate gradient method to a related linear system with the hope of obtaining faster convergence (see Section 3.5).

3.4 Conjugate Directions from Lanczos Vectors

An important tool in the formulation of methods for the solution of symmetric systems is the orthogonal reduction of a symmetric matrix to tridiagonal form. A *tridiagonal matrix* T has the property that $t_{ij} = 0$ for all $|i-j| > 2$. A tridiagonal form is the closest we can get to a diagonal matrix using an iterative procedure with a finite number of steps.

3.4.1 Direct orthogonal reduction to tridiagonal form

Theorem 3.4.1 *If A is an $n \times n$ symmetric matrix, there exists a tridiagonal T and an orthogonal V with unit first row and column such that $V^T A V = T$.*

Proof. The proof is by induction. The result is obviously true for matrices of order one and two because such matrices are trivially in tridiagonal form. Now assume that the result is true for matrices of order k and consider any symmetric matrix A of order $k+1$. Partition A so that

$$A = \begin{pmatrix} \alpha_0 & a_1^T \\ a_1 & A_2 \end{pmatrix},$$

where A_2 is $k \times k$. Define a Householder matrix H so that $Ha_1 = \|a_1\|_2 e_1$, with e_1 the first column of the identity matrix of order k. Then, if

$$V = \begin{pmatrix} 1 & 0 \\ 0 & H \end{pmatrix}, \quad \text{we have} \quad V^T A V = \begin{pmatrix} \alpha_0 & \beta_1 e_1^T \\ \beta_1 e_1 & \widehat{A} \end{pmatrix}, \quad \text{where} \quad \widehat{A} = H^T A_2 H.$$

The matrix $\widehat{A}$ is of order k, and the inductive hypothesis asserts the existence of an orthogonal $\widehat{V}$ and tridiagonal $\widehat{T}$ such that $\widehat{V}^T \widehat{A} \widehat{V} = \widehat{T}$ and $\widehat{V} e_1 = e_1$. Simple multiplication then gives

$$\begin{pmatrix} 1 & 0 \\ 0 & \widehat{V}^T \end{pmatrix} V^T A V \begin{pmatrix} 1 & 0 \\ 0 & \widehat{V} \end{pmatrix} = \begin{pmatrix} \alpha_0 & \beta_1 e_1^T \\ \beta_1 e_1 & \widehat{T} \end{pmatrix},$$

which is the required orthogonal reduction to tridiagonal form of the matrix A.

■

3.4.2 Lanczos reduction to tridiagonal form

Another technique for generating conjugate directions is based on the Lanczos reduction of A to tridiagonal form. This may be achieved directly by equating columns on the left-hand and right-hand sides of the identity $AV = VT$. Let the elements of the tridiagonal matrix T be written as

$$T = \begin{pmatrix} \gamma_0 & \beta_1 & & & \\ \beta_1 & \gamma_1 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \gamma_{n-1} \end{pmatrix}, \tag{3.14}$$

and the columns of V be denoted by v_0, v_1, ..., v_{n-1}. Equating columns in the identity $AV = VT$, gives

$$\beta_1 v_1 = Av_0 - \gamma_0 v_0, \qquad \beta_{k+1} v_{k+1} = Av_k - \gamma_k v_k - \beta_k v_{k-1}, \quad k = 1, 2, \ldots. \tag{3.15}$$

The orthonormality of the v_k (see Lemma 3.4.1) implies that $\gamma_k = v_k^T A v_k$. Moreover, if the vector $s_{k+1} = Av_k - \gamma_k v_k - \beta_k v_{k-1}$ is nonzero, then $v_{k+1} = (1/\beta_{k+1}) s_{k+1}$, where $\beta_{k+1} = \pm \|s_{k+1}\|_2$. If $\beta_{k+1} = 0$, the process terminates. Note that v_0 is arbitrary, and we can define $v_0 = v/\|v\|_2$ for any nonzero vector v.

The iteration described above for computing V and T is known as the *Lanczos process* and the vectors v_0, v_1, ..., v_k are known as the *Lanczos vectors*. An algorithm for computing the Lanczos vectors is given in Algorithm 3.4.1.

ALGORITHM 3.4.1. LANCZOS PROCESS

Set $s_0 = v$; $\beta_0 = -\|s_0\|_2$; $j = -1$;
while $\beta_{j+1} \neq 0$ **do**
    $v_{j+1} = s_{j+1}/\beta_{j+1}$; $j = j + 1$;
    $[Av_j] = Av_j$; $\gamma_j = v_j^T [Av_j]$;
    if $j = 0$ **then**
        $s_{j+1} = [Av_j] - \gamma_j v_j$;

else

$s_{j+1} = [Av_j] - \gamma_j v_j - \beta_j v_{j-1}$;

end if

$\beta_{j+1} = -\|s_{j+1}\|_2$;

end do

The Lanczos process will only break down if an off-diagonal element of T is zero, i.e., T is *reducible*. A breakdown indicates that the set of Lanczos vectors forms an invariant subspace. In order to continue the reduction to tridiagonal form, the Lanczos process must be restarted.

Note that T is called *irreducible* if it is not reducible.

3.4.3 Properties of the Lanczos vectors

The Lanczos iteration will terminate with $s_{m+1} = 0$ for some integer $m \leq n-1$. The following lemma establishes some basic properties of the Lanczos vectors.

Lemma 3.4.1 *If v_k, $0 \leq k \leq m$ are the Lanczos vectors generated by the Lanczos process, then*

(i) *The vectors $\{v_j\}_{j=0}^m$ are linearly independent;*

(ii) *The vectors $\{v_j\}_{j=0}^m$ are orthonormal;*

(iii) *For $0 \leq k \leq m$,* $\text{span}\{v_0, v_1, \ldots, v_k\} = \text{span}\{v_0, Av_0, \ldots, A^k v_0\}$,

Proof. Induction proofs for (ii) and (iii) may be found in Golub and Van Loan [16]. Note that (i) follows immediately from (ii). ∎

In the literature, $\text{span}\{v_0, Av_0, \ldots, A^{k-1}v_0\}$ is called the *k-th Krylov subspace* of A, and often denoted by $\mathcal{K}(A, v_0, k)$. The following theorem from Golub and Van Loan [16] summarizes the Lanczos process that reduces A to a tridiagonal matrix.

Theorem 3.4.2 *Let A be an $n \times n$ symmetric matrix. Then the Lanczos process terminates at $k = m$, where $m + 1 = \text{rank}(\mathcal{K}(A, v_0, n))$. Moreover, for $0 \leq k \leq m$, it holds that*

$$AV_k = V_k T_k + \beta_{k+1} v_{k+1} e_{k+1}^T = V_k T_k + s_{k+1} e_{k+1}^T, \tag{3.16}$$

where

$$T_k = \begin{pmatrix} \gamma_0 & \beta_1 & & & \\ \beta_1 & \gamma_1 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_k \\ & & & \beta_k & \gamma_k \end{pmatrix}. \tag{3.17}$$

∎

In exact arithmetic, Theorem (3.4.2) implies that $V_k^T A V_k = T_k$ for any $k \in \{0, \ldots, m\}$. However, in finite-precision arithmetic, roundoff error will lead to a loss of orthogonality of the Lanczos vectors; moreover, rounding errors will compound upon each other. Even for small values of n (e.g. $n = 15$) the roundoff error may be very considerable, and the Lanczos vectors may lose all semblance of orthogonality. It is worth noting that equation (3.16) is independent of the orthogonality of the Lanczos vectors (Paige [37], and Paige and Saunders [38]).

As it is, Algorithm 3.4.1 is unsuitable for finite-precision arithmetic; in particular, it is unlikely that $\beta_{k+1} = 0$ will ever be achieved. Moreover, a test comparing β_{k+1} to a small fixed constant will be a poor convergence test when the elements of T_{k+1} are either very large or very small. Intuitively, any test on β_{k+1} should be both relative and scale-independent so that the Lanczos process terminates if β_{k+1} is significantly smaller than the elements of T_{k+1}. Reasonable tests for convergence may look at the ratio of $|\beta_{k+1}|$ to $\|T_{k+1}\|_2$ or the ratio of $|\beta_{k+1}|$ to the largest (in modulus) element of T_{k+1}.

3.4.4 Conjugate directions from the Lanczos process

Given an arbitrary nonzero vector v with norm $\beta_0 = -\|v\|_2$, the Lanczos process generates scalars γ_0, γ_1, ..., γ_k, nonzero scalars β_1, β_2, ..., β_{k+1}, and orthonormal vectors v_0, v_1, ..., v_{k+1}, such that

$$T_k = \begin{pmatrix} \gamma_0 & \beta_1 & & & \\ \beta_1 & \gamma_1 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_k \\ & & & \beta_k & \gamma_k \end{pmatrix}, \tag{3.18}$$

and $AV_k = V_k T_k + \beta_{k+1} v_{k+1} e_{k+1}^T$, where V_k denotes the matrix with columns v_0, v_1, ..., v_k.

The matrix T_k is positive definite and its LDL^T factorization is given by $T_k = L_k D_k L^T$, where L_k unit-lower triangular and D_k a diagonal with positive diagonal entries. The columns of the matrix P_k such that $V_k = P_k L_k^T$ form a set of conjugate directions because

$$P_k^T A P_k = L_k^{-1} V_k^T A V_k L_k^{-T} = L_k^{-1} T_k L_k^{-T} = D_k. \tag{3.19}$$

The columns of P_k may be computed using a simple two-term recurrence involving the Lanczos vectors. To see this, notice that the lower-triangular factor L_k is unit lower-bidiagonal, with

$$L_k = \begin{pmatrix} 1 & & & \\ l_1 & 1 & & \\ & \ddots & \ddots & \\ & & l_k & 1 \end{pmatrix} \quad \text{and} \quad D_k = \begin{pmatrix} d_0 & & & \\ & d_1 & & \\ & & \ddots & \\ & & & d_k \end{pmatrix}.$$

As the calculation of the Lanczos vectors proceeds, the matrix T_k increases in dimension and a new row and column are added to its LDL^T factors. If we define $T_0 = \gamma_0$, $L_0 = 1$, and $D_0 = \gamma_0$, then the LDL^T factors of T_k are related to the

factors of T_{k-1} by the identity

$$\begin{pmatrix} T_{k-1} & \beta_k e_k \\ \beta_k e_k^T & \gamma_k \end{pmatrix} = \begin{pmatrix} L_{k-1} & 0 \\ l_k e_k^T & 1 \end{pmatrix} \begin{pmatrix} D_{k-1} & 0 \\ 0 & d_k \end{pmatrix} \begin{pmatrix} L_{k-1}^T & l_k e_k \\ 0 & 1 \end{pmatrix}. \tag{3.20}$$

At the end of the kth stage of the Lanczos process, the factors L_{k-1} and D_{k-1} of T_{k-1} are known, and it is necessary to find only the last row of L_k and last diagonal of D_k. Comparing both sides of (3.20) yields

$$l_k = \beta_k / d_{k-1} \quad \text{and} \quad d_k = \gamma_k - \beta_k l_k. \tag{3.21}$$

This convenient bidiagonal structure leads to a simple recurrence relation for the required columns of P_k. From the definition of P_k we have $V_k = P_k L_k^T$, which can be written column-wise as

$$\begin{pmatrix} v_0 & v_1 & \cdots & v_k \end{pmatrix} = \begin{pmatrix} p_0 & l_1 p_0 + p_1 & \cdots & l_k p_{k-1} + p_k \end{pmatrix}.$$

This identity allows us to solve for each column of P_k, giving the recurrence relations:

$$p_0 = v_0, \qquad p_i = v_i - l_i p_{i-1}, \ i = 1, \ldots, k. \tag{3.22}$$

Note that different sets of conjugate-directions are generated with each choice of initial vector v. In the next section, it will be shown that the choice $v = b$ gives directions that are rescaled versions of the directions generated by the conjugate-gradient method.

3.4.5 The Lanczos-CG method

A disadvantage of using Lanczos vectors to generate the conjugate directions (as in (3.22)) is that both the Lanczos process and the subspace minimization each require a product of A with a vector—thereby doubling the work per iteration compared to the vanilla conjugate-gradient method.

However, by rearranging the computation, it is possible to derive an efficient Lanczos-based conjugate-direction method that requires the same number

of matrix-vector products as the vanilla CG method. The derivation is based on the observation that the new iterate x_{k+1} is uniquely defined as a linear combination of x_0 and the $k+1$ columns of P_k, and that the columns of P_k may be written as linear combinations of the Lanczos vectors. This implies that x_{k+1} may be found by minimizing q over all points x such that

$$x \in x_0 + \text{span}\{v_0, v_1, \ldots, v_k\} = x_0 + V_k w,$$

for some vector w. If w_k is the unique optimal value of w, then the reduced equations analogous to (3.5) are

$$x_{k+1} = x_0 + V_k w_k, \quad \text{with} \quad V_k^T A V_k w_k = -V_k^T g_0,$$

where the right-hand side now involves the *initial* gradient $g_0 = \nabla q(x_0)$. If we choose $v_0 = -g_0/\|g_0\|_2$, the orthogonality of the Lanczos vectors gives $V_k^T g_0 = v_0^T g_0 e_1 = \beta_0 e_1$, and the reduced system is equivalent to

$$x_{k+1} = x_0 + V_k w_k, \quad \text{with} \quad T_k w_k = -\beta_0 e_1, \tag{3.23}$$

where T_k is the tridiagonal matrix such that $T_k = V_k^T A V_k$. These equations may be solved efficiently using the factorization $T_k = L_k D_k L_k^T$ and the auxiliary vector y_k such that $y_k = L_k^T w_k$. Substituting for T_k and w_k in (3.23) gives

$$x_{k+1} = x_0 + P_k y_k, \quad \text{with} \quad L_k D_k y_k = -\beta_0 e_1, \tag{3.24}$$

where P_k is the matrix such that $V_k = P_k L_k^T$ (see Section 3.4.4).

The crucial feature of the lower-bidiagonal system $L_k D_k y_k = -\beta_0 e_1$ is that the first k elements of y_k are just the elements of y_{k-1}. To show this, consider L_k and D_k in terms of the quantities L_{k-1} and D_{k-1} associated with the previous step. Then, the lower-triangular system $L_k D_k y_k = -\beta_0 e_1$ may be written as

$$\begin{pmatrix} L_{k-1} D_{k-1} & 0 \\ l_k d_{k-1} e_k^T & d_k \end{pmatrix} \begin{pmatrix} y_{k-1} \\ \alpha_k \end{pmatrix} = -\beta_0 e_1,$$

where $\alpha_0, \alpha_1, \ldots, \alpha_k$ denotes the elements of y_k. Hence, using the relation $l_k d_{k-1} = \beta_k$ (see 3.21), we have for all $k > 0$, $x_{k+1} = x_k + \alpha_k p_k$, with

$$\alpha_k = \begin{cases} \beta_0/d_0 & \text{if } k = 0; \\ -\beta_k \alpha_{k-1}/d_k & \text{otherwise.} \end{cases} \tag{3.25}$$

The next result shows that each Lanczos vector v_k is a (negative) scalar multiple of the gradient vector g_k.

Theorem 3.4.3 *If v_0 is chosen so that $v_0 = -g_0/\|g_0\|_2$, then*

$$g_k = \nabla q(x_k) = \alpha_{k-1}\beta_k v_k. \tag{3.26}$$

Proof. The Lanczos recurrence (3.15) implies that

$$AV_{k-1} = V_{k-1}T_{k-1} + \beta_k v_k e_k^T. \tag{3.27}$$

Post-multiplying this identity by w_{k-1} and substituting $x_k - x_0$ for $V_{k-1}w_{k-1}$ gives

$$A(x_k - x_0) = V_{k-1}T_{k-1}w_{k-1} + \beta_k v_k e_k^T w_{k-1}.$$

To simplify this expression, note that $T_{k-1}w_{k-1} = -\beta_0 e_1$, and the relation $y_{k-1} = L_{k-1}^T w_{k-1}$ implies that the kth (i.e., last) element of w_{k-1} is α_{k-1}, the last element of y_{k-1}. It follows that

$$A(x_k - x_0) = -\beta_0 V_{k-1}e_1 + \beta_k v_k e_k^T w_{k-1} = -\beta_0 v_0 + \alpha_{k-1}\beta_k v_k = -g_0 + \alpha_{k-1}\beta_k v_k.$$

The definition $g_0 = \nabla q(x_0)$ now implies that $Ax_k - b = \alpha_{k-1}\beta_k v_k$, as required. ∎

This theorem provides an alternative way of computing the two-norm of the gradient g_k:

$$\|g_k\|_2 = \|\beta_k v_k \alpha_{k-1}\|_2 = \beta_k |\alpha_{k-1}|. \tag{3.28}$$

In exact arithmetic, β_k will be zero in a finite number of steps, and hence the gradient will also be zero.

If $x_0 = 0$ (with a suitable shift of the constant vector b), then x_{k+1} lies in $\text{span}\{v_0, v_1, \ldots, v_k\}$ and we seek a solution of the form $x_{k+1} = V_k w_k$ for some unique vector w_k. In this case we have

$$Ax_k = V_{k-1}(\beta_0 e_1) + \beta_k v_k e_k^T w_{k-1} = b + \alpha_{k-1}\beta_k v_k, \tag{3.29}$$

Note that $V_{k-1}(\beta_0 e_1) = b$ *exactly* for all k because v_0 is a multiple of b. Also, the system $T_k w_k = -\beta_0 e_1$ can be solved accurately (and cheaply). Thus, in finite-precision arithmetic, (3.29) holds accurately for any w_k *even though the columns of* V_{k-1} *are not orthogonal.*

Algorithm 3.4.2. Lanczos-CG Method

Choose $\tau_{tol} > 0$;
$s_0 = -g(x_0)$; $\beta_0 = -\|s_0\|$; $\gamma_{-1} = 1$; $\tau = \beta_0$; $k = -1$;
while $\tau > \tau_{tol}$ **do**
  $v_{k+1} = s_{k+1}/\beta_{k+1}$; $k = k + 1$;
  $[Av_k] = Av_k$; $\gamma_k = v_k^T[Av_k]$;
  if $k = 0$ **then**
    $l_k = 0$; $p_k = v_k$;
    $s_{k+1} = [Av_k] - \gamma_k v_k$;
  else
    $l_k = \beta_k/d_{k-1}$; $p_k = v_k - l_k p_{k-1}$;
    $s_{k+1} = [Av_k] - \gamma_k v_k - \beta_k v_{k-1}$;
  end
  $d_k = \gamma_k - \beta_k l_k$; $\alpha_k = -\beta_k \alpha_{k-1}/d_k$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $\beta_{k+1} = -\|s_{k+1}\|$; $\tau = -\beta_{k+1}\alpha_k$;
end do

3.5 Preconditioned CG Methods

The preconditioned conjugate-gradient (PCG) method for solving $Ax = b$ is based on two fundamental properties of the CG method: (i) the rate of convergence is sensitive to the condition number of A (see, e.g., Theorem 3.3.2); and (ii) the CG method converges faster when A has few distinct eigenvalues (see, e.g., Theorem 3.3.1). The PCG method attempts to exploit these properties by replacing an original system $Ax = b$ with a related system of the form

$$M^{-1/2}AM^{-1/2}\hat{x} = M^{-1/2}b, \tag{3.30}$$

where M is a symmetric positive-definite matrix (the *preconditioner*) chosen so that $M^{-1/2}AM^{-1/2}$ has eigenvalues clustered around unity. Notice that solving $Ax = b$ for x is equivalent to solving system (3.30) for $\hat{x}$ and then forming $x = M^{-1/2}\hat{x}$. Since $M^{-1/2}AM^{-1/2}$ is similar to $M^{-1}A$, the problem reduces to finding M such that $M^{-1}A$ is well-conditioned and has many unit eigenvalues. (Ideally, $M^{-1}A$ will be approximately the identity matrix).

In practice, the computation is arranged so that only solves with M are required (i.e., $M^{1/2}$ never appears). To see this, consider one iteration of the CG method (Algorithm 3.3.1) applied to the system $\widehat{A}\hat{x} = \widehat{b}$, with vectors $\hat{x}_k$, $\hat{p}_k$ and $\hat{g}_k$, and scalars $\hat{\alpha}_k$ and $\hat{\beta}_k$. If x_k, p_k and g_k are defined so that $x_k = M^{-1/2}\hat{x}_k$, $p_k = M^{-1/2}\hat{p}_k$ and $g_k = M^{1/2}\hat{g}_k$, then $M^{1/2}$ no longer appears. The simplified recurrence relations are given in the following algorithm.

Algorithm 3.5.1. The PCG method

Choose x_0; Set $g_0 = g(x_0)$; $k = 0$;
while $g_k \neq 0$ **do**
  if $k = 0$ **then**
    $p_k = -M^{-1}g_k$;
  else
    $\beta_{k-1} = g_k^T M^{-1} A p_{k-1} / p_{k-1}^T A p_{k-1}$;

$p_k = -M^{-1}g_k + \beta_{k-1}p_{k-1};$

end if

$\alpha_k = -g_k^T p_k / p_k^T A p_k;$

$x_{k+1} = x_k + \alpha_k p_k;$

$g_{k+1} = g_k + \alpha_k A p_k;$

$k = k + 1;$

end do

The scalars α_k and β_k are the quantities $\hat{\alpha}_k$ and $\hat{\beta}_k$ associated with the CG method applied to $\widehat{A}\hat{x} = \widehat{b}$. It follows that both α_k and β_k are positive, as in the unpreconditioned case.

The orthogonality and conjugacy properties associated with the vectors defined by the preconditioned CG method are summarized in the following result.

Lemma 3.5.1 *Consider the PCG method with symmetric positive-definite preconditioner M applied to the symmetric system $Ax = b$. At the kth step, the directions $\{p_i\}_{i=0}^k$ and gradients $\{g_i\}_{i=0}^k$ satisfy*

$$p_i^T A p_j = 0, \quad \textit{and} \quad g_i^T M g_j = 0 \quad 0 \le i, j \le k, \quad i \ne j.$$

Moreover, $g_{k+1}^T p_i = 0$ for $0 \le i \le k$. ∎

The next result is crucial for the development of trust-region methods based on the conjugate-gradient method. It implies that for $i \ne j$, the directions satisfy $0 < p_j^T M p_i / \|p_i\|_M \|p_j\|_M < 1$, which means that the PCG directions form acute angles with each other.

Lemma 3.5.2 *The directions generated by the conjugate-gradient method satisfy*

$$p_j^T M p_i > 0 \quad \textit{for } i < j \textit{ and } 0 \le j \le k.$$

Proof. The proof is by induction. Observe that $p_0^T M p_0 = \|g_0\|_{M^{-1}}^2 > 0$, and so the result holds for $k = 0$. Assume that the result holds for $i < j$ and $0 \le j \le k - 1$.

The CG direction is given by $p_k = -M^{-1}g_k + \beta_{k-1}p_{k-1}$. It follows that, for all p_i such that $0 \leq i \leq k-1$,

$$p_k^T M p_i = -g_k^T p_i + \beta_{k-1} p_{k-1}^T M p_i.$$

The orthogonality property $g_{k+1}^T p_i = 0$ for $0 \leq i \leq k$ of Lemma 3.5.1, the second form of β_{k-1} given in (3.13), and the inductive hypothesis give $p_k^T M p_i = \beta_{k-1} p_{k-1}^T M p_i > 0$, as required. ∎

3.5.1 The preconditioned Lanczos-CG method

The Lanczos-CG method may also be defined with a preconditioner. In this case, the Lanczos process is applied to the matrix $M^{-1/2}AM^{-1/2}$, giving orthonormal directions w_k such that $M^{1/2}W_k$ transforms A to tridiagonal form. This is equivalent to applying the Lanczos process directly to A but with the Lanczos vectors v_k being M-orthogonalized. In this case, it is easy to see that v_k and w_k are related by the identity $Mv_k = w_k$). The following algorithm provides the details of this process. Note that, as in the PCG method, only *solves* with M are required to generate the directions $\{v_k\}$—the explicit elements of M are not required.

ALGORITHM 3.5.2. THE PRECONDITIONED LANCZOS PROCESS

Set $s_0 = v$; $\beta_0 = -\|s_0\|_{M^{-1}}$; $j = -1$;

while $\beta_{j+1} \neq 0$ **do**

$w_{j+1} = s_{j+1}/\beta_{j+1}$; Solve $Mv_{j+1} = w_{j+1}$; $j \leftarrow j+1$;

$[Av_j] = Av_j$; $\gamma_j = v_j^T[Av_j]$;

if $j = 0$ **then**

$s_{j+1} = [Av_j] - \gamma_j w_j$;

else

$s_{j+1} = [Av_j] - \gamma_j w_j - \beta_j w_{j-1}$;

end if

$\beta_{j+1} = -\|s_{j+1}\|_{M^{-1}}$;

end do

Using the vectors $\{v_k\}$ the preconditioned Lanczos process (Algorithm 3.5.2) may be viewed as a method for tridiagonalizing A; namely, for $k > 1$

$$AV_k = MV_kT_k + Ms_{k+1}e_k^T, \tag{3.31}$$

i.e., in exact arithmetic, $V_k^TAV_k = T_k$. Thus, Algorithm 3.4.2 may be rewritten as follows to accommodate a preconditioner.

Algorithm 3.5.3. The Lanczos-PCG Method

Choose $\tau_{tol} > 0$;

$x_0 = 0$; $\quad s_0 = g(x_0)$; $\quad \beta_0 = -\|s_0\|_{M^{-1}}$; $\quad \gamma_{-1} = 1$; $\quad \tau = \beta_0$; $\quad k = -1$;

while $\tau > \tau_{tol}$ **do**

$w_{k+1} = s_{k+1}/\beta_{k+1}$; $\quad$ Solve $Mv_{k+1} = w_{k+1}$; $\quad k \leftarrow k+1$;

$[Av_k] = Av_k$; $\quad \gamma_k = v_k^T[Av_k]$;

if $k = 0$ **then**

$l_k = 0$; $\quad p_k = v_k$;

$s_{k+1} = [Av_k] - \gamma_k w_k$;

else

$l_k = \beta_k/d_{k-1}$; $\quad p_k = v_k - l_k p_{k-1}$;

$s_{k+1} = [Av_k] - \gamma_k w_k - \beta_k w_{k-1}$;

end

$d_k = \gamma_k - \beta_k l_k$; $\quad \alpha_k = -\beta_k \alpha_{k-1}/d_k$;

$x_{k+1} = x_k + \alpha_k p_k$;

$\beta_{k+1} = -\|s_{k+1}\|_{M^{-1}}$; $\quad \tau = -\beta_{k+1}\alpha_k$;

end do

Note that the Lanczos-PCG method of Algorithm 3.5.3 reduces to the Lanczos-CG method of Algorithm 3.4.2 if $M = I$.

3.6 Economizing Matrix-Vector Products

The major computational cost involved with solving $Ax = b$ using an iterative method is associated with forming the matrix-vector products with A. In the context of large-scale optimization, it may be necessary to know the product of A with the final CG iterate x_k. Since this quantity is not a by-product of the standard CG method it must be computed directly using an extra matrix-vector product, or indirectly by means of a recurrence relation.

Here we consider forming this product indirectly while solving $Ax = b$. The algorithm presented below is specifically in the context of the Lanczos-PCG method. The details are similar for the other CG-type methods presented earlier.

ALGORITHM 3.6.1. THE EXTENDED LANCZOS-PCG METHOD

Choose $\tau_{tol} > 0$;

$x_0 = 0; \quad [Ax_0] = 0;$

$s_0 = g(x_0); \quad \beta_0 = -\|s_0\|_{M^{-1}}; \quad \gamma_{-1} = 1; \quad \tau = \beta_0; \quad k = -1;$

while $\tau > \tau_{tol}$ **do**

$w_{k+1} = s_{k+1}/\beta_{k+1}; \quad$ Solve $Mv_{k+1} = w_{k+1}; \quad k \leftarrow k + 1;$

$[Av_k] = Av_k; \quad \gamma_k = v_k^T[Av_k];$

if $k = 0$ **then**

$l_k = 0;$

$p_k = v_k; \quad [Ap_k] = [Av_k];$

$s_{k+1} = [Av_k] - \gamma_k w_k;$

else

$l_k = \beta_k/d_{k-1};$

$p_k = v_k - l_k p_{k-1}; \quad [Ap_k] = [Av_k] - l_k[Ap_{k-1}];$

$s_{k+1} = [Av_k] - \gamma_k w_k - \beta_k w_{k-1};$

end

$d_k = \gamma_k - \beta_k l_k; \quad \alpha_k = -\beta_k \alpha_{k-1}/d_k;$

$x_{k+1} = x_k + \alpha_k p_k; \quad [Ax_{k+1}] = [Ax_k] + \alpha_k[Ap_k];$

$\beta_{k+1} = -\|s_{k+1}\|_{M^{-1}}; \quad \tau = -\beta_{k+1}\alpha_k;$

end do

Only one matrix-vector product is required for each Lanczos-PCG iteration. In particular, the product Av_k must be calculated as an explicit matrix-vector product involving A and v_k. Once this vector is known, the calculation of Ax_{k+1} requires no additional work.

3.7 Computations with the Preconditioner

The preconditioned Lanczos-CG method for solving $Ax = b$ involves a positive-definite matrix M that approximates A and is such that systems of the form $My = x$ can be solved efficiently. In general, the *elements* of the matrices M and M^{-1} are not known explicitly and it is necessary to arrange the computation so that only solves with M are required.

Lanczos-PCG requires only products with M^{-1} so there are no computational difficulties in Algorithm 3.5.3 (nor in the extend algorithm (Algorithm 3.6.1)). However, in later sections, it will be desirable to compute $\|x_{k+1}\|_M$ and Mx_{k+1}. It turns out that these quantity cannot be computed exactly; they can only be estimated.

The following algorithm details how this estimation is performed in the context of preconditioned Lanczos-CG. Once again, the bracket notation (introduced in Section 1.3), will be used to emphasize stored quantities. The following algorithm is an extension of the basic preconditioned Lanczos-CG algorithm (Algorithm 3.5.3).

Algorithm 3.7.1. Lanczos-PCG with Elliptical Norm Estimates

Choose $\tau_{tol} > 0$;

$x_0 = 0; \quad [Mx_0] = 0;$

$s_0 = g(x_0); \ \beta_0 = -\|s_0\|_{M^{-1}}; \quad \gamma_{-1} = 1; \quad \tau = \beta_0; \ k = -1;$

while $\tau > \tau_{tol}$ **do**

$w_{k+1} = s_{k+1}/\beta_{k+1}; \quad$ Solve $Mv_{k+1} = w_{k+1}; \quad k \leftarrow k+1;$

$[Av_k] = Av_k; \quad \gamma_k = v_k^T[Av_k];$

if $k = 0$ **then**

$l_k = 0;$

$p_k = v_k; \quad [Mp_k] = [Mv_k];$

$s_{k+1} = [Av_k] - \gamma_k w_k;$

else

$l_k = \beta_k / d_{k-1};$

$p_k = v_k - l_k p_{k-1}; \quad [Mp_k] = [Mv_k] - l_k[Mp_{k-1}];$

$s_{k+1} = [Av_k] - \gamma_k w_k - \beta_k w_{k-1};$

end

$d_k = \gamma_k - \beta_k l_k; \;\; \alpha_k = -\beta_k \alpha_{k-1} / d_k;$

$x_{k+1} = x_k + \alpha_k p_k; \quad [Mx_{k+1}] = [Mx_k] + \alpha_k[Mp_k];$

$\|x_{k+1}\|_M^2 = x_{k+1}^T[Mx_{k+1}];$

$\beta_{k+1} = -\|s_{k+1}\|_{M^{-1}}; \quad \tau = -\beta_{k+1}\alpha_k;$

end do

Observe that the calculation of $\|x_k\|_M$ assumes that the Lanczos vectors are exactly M-orthogonal. Unfortunately, this property does not hold in finite-precision arithmetic and in general, the recurrence relations give only an *estimate* of $\|x_k\|_M$.

For completeness, the following algorithm implicitly forms *both* Ax_{k+1} (as in Algorithm 3.7.1) and Mx_{k+1} (as in Algorithm 3.6.1).

ALGORITHM 3.7.2. THE COMBINED LANCZOS-PCG METHOD

Choose $\tau_{tol} > 0;$

$x_0 = 0; \quad [Ax_0] = 0; \quad [Mx_0] = 0;$

$s_0 = g(x_0); \;\; \beta_0 = -\|s_0\|_{M^{-1}}; \quad \gamma_{-1} = 1; \quad \tau = \beta_0; \;\; k = -1;$

while $\tau > \tau_{tol}$ **do**

$w_{k+1} = s_{k+1}/\beta_{k+1}; \quad$ Solve $Mv_{k+1} = w_{k+1}; \quad k \leftarrow k+1;$

$[Av_k] = Av_k; \quad \gamma_k = v_k^T[Av_k];$

if $k = 0$ **then**

$l_k = 0;$

$p_k = v_k$; $[Ap_k] = [Av_k]$; $[Mp_k] = [Mv_k]$;

$s_{k+1} = [Av_k] - \gamma_k w_k$;

else

$l_k = \beta_k / d_{k-1}$;

$p_k = v_k - l_k p_{k-1}$; $[Ap_k] = [Av_k] - l_k[Ap_{k-1}]$; $[Mp_k] = [Mv_k] - l_k[Mp_{k-1}]$;

$s_{k+1} = [Av_k] - \gamma_k w_k - \beta_k w_{k-1}$;

end

$d_k = \gamma_k - \beta_k l_k$; $\alpha_k = -\beta_k \alpha_{k-1} / d_k$;

$x_{k+1} = x_k + \alpha_k p_k$; $[Ax_{k+1}] = [Ax_k] + \alpha_k[Ap_k]$; $[Mx_{k+1}] = [Mx_k] + \alpha_k[Mp_k]$;

$\|x_{k+1}\|_M^2 = x_{k+1}^T [Mx_{k+1}]$;

$\beta_{k+1} = -\|s_{k+1}\|_{M^{-1}}$; $\tau = -\beta_{k+1}\alpha_k$;

end do

4

Iterative Methods for the Trust-Region Subproblem

4.1 Introduction

This chapter concerns methods for solving the trust-region subproblem that are based on iterative methods for linear equations. The discussion will focus on the problem

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) \;\; \text{subject to} \;\; \|Ns\| \leq \delta, \tag{4.1}$$

where the outer iteration subscript j has been omitted to simplify notation.

In the methods to be discussed, each iterate solves the trust-region subproblem (4.1) subject to the additional restriction that the solution lies in a certain *subspace* of $\mathbb{R}^n$. One of the first methods of this type was proposed by Powell [41, 42, 43]. Powell proposed approximating the solution of (4.1) by solving a single subspace minimization of the form

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) \;\; \text{subject to} \;\; \|Ns\| \leq \delta, \quad s \in \text{span}\{-g\}. \tag{4.2}$$

This idea has been refined by Byrd, Schnabel and Shultz [2], who define the two-

dimensional subproblem

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) \;\; \text{subject to} \;\; \|Ns\| \leq \delta, \quad s \in \text{span}\{p, q\}, \tag{4.3}$$

where p is the solution of a positive-definite system of the form $(H+\sigma N^T N)p = -g$ and q is a direction of negative curvature for H.

The methods considered here generate a *sequence* of subspaces $\{\mathcal{S}_k\}$. Given the kth iterate s_k such that $s_k \in \mathcal{S}_k$, the next iterate s_{k+1} is the solution of

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) \;\; \text{subject to} \;\; \|Ns\| \leq \delta, \quad s \in \mathcal{S}_k. \tag{4.4}$$

The CG method for minimizing the unconstrained function $\mathcal{Q}(s)$ is a particular subspace minimization method that generates a sequence of *expanding* subspaces, i.e., $\{\mathcal{S}_k\}$ satisfies $\mathcal{S}_{k+1} \supset \mathcal{S}_k$ with $\dim(\mathcal{S}_{k+1}) > \dim(\mathcal{S}_k)$.

4.2 Steihaug's Method

There are two difficulties associated with using the CG method to solve the trust-region subproblem. First, the CG method is intended for unconstrained minimization and cannot be used directly to solve a constrained problem. Both Toint [40] and Steihaug [48] proposed using the CG method to minimize the *unconstrained* quadratic model until an iterate leaves the trust region. In this case, the approximate solution of the subproblem is defined as the point where the piecewise-linear path of CG iterates crosses the trust-region boundary.

It might appear that subsequent CG iterates could re-enter the trust region and converge to an unconstrained point. The next result, due to Steihaug [48], shows that, under certain conditions, if the CG method is preconditioned with $M = N^T N$, then the PCG iterates are increasing in the M-norm. This result is fundamental because it implies that once the PCG iterates leave the trust-region they cannot return.

Theorem 4.2.1 *Let $\{s_k\}$ denote the sequence of iterates defined by the PCG method with preconditioner $M = N^T N$ when applied to minimize a positive-definite quadratic function $\mathcal{Q}(s)$. If $s_0 = 0$ then the sequence $\{s_k\}$ satisfies*

$$\|s_{k+1}\|_M > \|s_k\|_M \quad \textit{for all } k \geq 0.$$

Proof. The PCG iterates satisfy $s_{k+1} = s_k + \alpha_k p_k$. Forming the square of the M-norm of s_{k+1} gives

$$\|s_{k+1}\|_M^2 = \|s_k + \alpha_k p_k\|_M^2 = \|s_k\|_M^2 + 2\alpha_k s_k^T M p_k + \alpha_k^2 \|p_k\|_M^2. \tag{4.5}$$

If $s_0 = 0$, then the kth iterate may be written in the form

$$s_k = \alpha_0 p_0 + \alpha_1 p_1 + \cdots + \alpha_{k-1} p_{k-1} = P_{k-1} y_{k-1},$$

where P_{k-1} is the matrix $P_{k-1} = (\, p_0 \;\; p_1 \;\; \cdots \;\; p_{k-1} \,)$ and y_{k-1} is the vector of PCG step lengths $(\alpha_0, \alpha_1, \ldots, \alpha_{k-1})$. The matrix H is positive definite, which implies that the scalars α_{k-1} are all positive. It follows from Lemma 3.5.2 that

$$s_k^T M p_k = y_k^T P_k^T M p_k = \alpha_0 p_k^T M p_0 + \alpha_1 p_k^T M p_1 + \cdots + \alpha_{k-1} p_k^T M p_{k-1} > 0.$$

The result now follows directly from (4.5). ∎

This result implies that PCG generates a sequence of iterates that are strictly increasing in the M-norm, i.e., $\|s_{k+1}\|_M > \|s_k\|_M$ for all $k \geq 0$. Thus, PCG iterates may be generated to minimize $\mathcal{Q}(s)$ (or, equivalently, solve $Hs = -g$) until an iterate does not lie inside the trust region, i.e., $\|s_k + \alpha_k p_k\|_M \geq \delta$. Theorem 4.2.1 implies that once a PCG iterate leaves the trust-region, then the solution of the constrained problem (4.1) must lie on the boundary.

A second difficulty that arises when applying PCG to minimize a general function $\mathcal{Q}(s)$ is that the iterates may not be well-defined when H is not positive definite. The PCG algorithm recognizes an indefinite H if it computes a negative entry in D while updating the LDL^T decomposition. In this case, the conjugate direction generated in this iteration is a direction of negative or zero curvature.

In Steihaug's method, the PCG iterations are terminated and the final iterate is a point on the direction of negative curvature that lies on the trust-region boundary.

The approach outlined above is formally known as Steihaug's method, and forms the basis of one of the most widely-used iterative trust-region methods. PCG iterates are computed until the quantities s_k, p_k and α_k associated with the kth PCG iteration satisfy one of the following three termination conditions hold:

($\mathbf{T}_1$) $s_k + \alpha_k p_k$ is an approximate minimizer of $\mathcal{Q}(s)$ such that $\|s_k + \alpha_k p_k\|_M < \delta$;

($\mathbf{T}_2$) $\|s_k + \alpha_k p_k\|_M \geq \delta$;

($\mathbf{T}_3$) $p_k^T H p_k \leq 0$.

If termination occurs because of condition ($\mathbf{T}_1$) then the subproblem is terminated with the approximate solution $s_k + \alpha_k p_k$. In this case, Steihaug's method and the PCG method are equivalent.

If termination occurs because of ($\mathbf{T}_2$) or ($\mathbf{T}_3$), the trust-region solution must lie on the boundary and the subproblem is terminated with the point $s_k + \alpha'_k p_k$, where α'_k is defined so that $\|s_k + \alpha'_k p_k\|_M = \delta$. If PCG terminates under condition ($\mathbf{T}_2$) or ($\mathbf{T}_3$), the final iterate is known as the *Steihaug point*.

The first PCG iterate s_1 is the Cauchy step for (4.1), and it follows that Steihaug's method generates an approximate solution that is at least as good as the Cauchy step because any subsequent iterates must give an additional reduction in $\mathcal{Q}(s)$. Hence, under minimal additional assumptions on the geometry of f, Steihaug's algorithm is globally convergent to first-order points (see Section 2.3.3).

It remains to determine the precise form of the termination condition ($\mathbf{T}_1$). For *inexact* Newton methods (i.e., Newton-based methods that *approximately* solve the Newton equations at each iteration), a commonly used termination condition requires that the norm of the residual of the Newton system should be less than a fixed multiple of the norm of the right-hand side (the residual for $s_0 = 0$). This implies that the final iterate s_k must satisfy

$$\|Hs_k + g\| \leq \eta \|g\|, \tag{4.6}$$

where $\eta \in (0, 1)$ and may be a member of a forcing sequence. The following theorem is adapted from Dembo, Eisenstat and Steihaug [5].

Theorem 4.2.2 *Suppose that a Newton-PCG method for minimizing f generates iterates $\{x_j\}$ that converge to x^*, where $x_{j+1} = x_j + s_j$ and s_j denotes an approximate solution of the Newton equations $H_j s = -g_j$. Then, $x_j \to x^*$ at a Q-superlinear rate if and only if $\|H_j s_j + g_j\| = o(\|g_j\|)$, as $j \to \infty$.* ∎

A desirable quality of any trust-region method is that the trust-region radius should not interfere with the natural rate of convergence of Newton's method. This implies that the outer iterates s_j must satisfy $H_j s_j = -g_j$ in the neighborhood of a local minimizer. If this is to happen, Steihaug's method must coincide with the PCG method, with each subproblem ending at a point inside the trust region. In this context, Theorem 4.2.2 implies that superlinear convergence of the outer iterates requires $H_j s_j = -g_j$ to be solved with increasing accuracy as $\{x_j\} \to x^*$. (For more details, see Dembo, Eisenstat and Steihaug [5], and Conn, Gould and Toint [4].)

4.3 The GLTR Method

An inherent weakness of Steihaug's method is its inability to control the accuracy of approximate subproblem solutions on the boundary. The Steihaug point is always at least as accurate as the Cauchy step, but the Cauchy step may be a poor approximate solution to the trust-region subproblem. In particular, the Steihaug point may be a poor approximate solution to the subproblem if the CG method terminates after a few iterations.

Gould, Lucidi, Roma, and Toint [18] propose the generalized Lanczos trust-region (GLTR) method, which may be viewed as a natural extension of the CG method to the constrained case. If the trust-region solution lies on the boundary, the GLTR method solves the *constrained* trust-region subproblem over an expand-

ing sequence of subspaces. As in Steihaug's method, the monotone norm property of the CG iterates is used first to find an interior solution or a Steihaug point.

If the PCG method terminates at a Steihaug point, the trust-region subproblem is solved on a sequence of expanding subspaces defined by the Lanczos vectors. Each iteration beyond the Steihaug point requires a matrix-vector product to form the new Lanczos vector and the solution of a reduced-space trust-region problem

$$\underset{h\in\mathbb{R}^{k+1}}{\text{minimize}} \;\; \beta_0 e_1^T h + \tfrac{1}{2} h^T T_k h \quad \text{subject to} \;\; \|h\| \leq \delta, \tag{4.7}$$

where T_k is the tridiagonal matrix obtained from the Lanczos process (see (3.17)). Gould et al. [18] adapt the Moré-Sorensen algorithm to find an approximate solution of (4.7) such that

$$\|(H + \sigma_k M)s_k + g\|_{M^{-1}} \leq \tau. \tag{4.8}$$

Their algorithm exploits the tridiagonal structure of T. (The method uses an equivalent form of (4.8) that does not involve σ_k, but relies on the M-orthogonality of the Lanczos vectors.) If the Lanczos process terminates (i.e, T_k is reducible—see Section 3.4.2), the Lanczos process is restarted, but the trust-region solver continues. Thus, even after Lanczos restarts, GLTR continues to solve the subproblem or reduced subproblem over a sequence of expanding subspaces.

The GLTR method may require two passes through the algorithm. If the quadratic function evaluated at the Steihaug point (or at a point within the trust region) is within a required percentage (say, 90%) of the smallest value of the quadratic function obtained by any iterate during first pass, the algorithm may terminate with either the Steihaug point or the last iterate inside the trust region. Otherwise, the reduced-space solution h_{k+1} is expanded into the full space to obtain the approximate solution $s_k = V_k h_k$ of the trust-region subproblem. Since the calculation of s_k requires V_k the Lanczos vectors must be regenerated using a second pass through the Lanczos process.

Although the GLTR method may be viewed as a natural extension of Steihaug's method, its susceptibility to rounding error limits its practical effectiveness. If a

substantial number of Steihaug or GLTR iterations are required, the columns of V_k lose their orthogonality and the matrices $V_k^T H V_k$ and T_k no longer correspond to each other. In this case, there is a significant difference between the solution of the original subproblem and the solution of the reduced subproblem. In practice, the expanded reduced-space solution h_{k+1} rapidly ceases to be effective.

If T_k is reducible, the Lanczos process must be restarted with a vector that is M-orthonormal with respect the previous Lanczos vectors. This reorthogonalization requires that the Lanczos vectors be formed one more time. Also, in this case, GLTR will possibly require more work than PCG to solve the *unconstrained* minimization problem

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) = g^T s + \tfrac{1}{2} s^T H s.$$

To see this, recall that the Lanczos process breaks down only if an off-diagonal element is zero, i.e., the residual to the linear system $Hs+g$ is small (Theorem 3.4.3). A break down of the Lanczos process signals that the PCG iterates (would have) converged to a solution outside the trust region; however, GLTR may continue to optimize beyond this point. Thus, the amount of work required by GLTR for this subproblem solve will be at least as much as the amount of work required to solve the unconstrained quadratic minimization problem using the PCG method.

4.4 Hager's Method

In contrast to Steihaug's method and GLTR, Hager's subproblem solver obtains theoretical global convergence without relying on a sequence of expanding subspaces. Global convergence relies on generating better quality subspaces rather than generating subspaces of increasing dimension. Hager [21] uses a sequential subspace minimization (SSM) method as the basis for a trust-region solver but restarts the Lanczos process at fixed intervals to help minimize the loss of orthogonality of the Lanczos vectors.

Hager's method uses two phases to approximate a solution for the *equality constrained* trust-region subproblem

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) = g^T s + \tfrac{1}{2} s^T H s \;\; \text{subject to} \;\; \|Ns\| = \delta. \tag{4.9}$$

The first phase generates a small number of Lanczos vectors (four or five, say), and then solves the subproblem over the subspace

$$\mathcal{S}_k = \text{span}\{s_k, \tilde{u}_n^k, v_1, \ldots, v_l\},$$

where s_k denotes the previous iterate, $\tilde{u}_n^k$ denotes the kth approximation to the leftmost generalized eigenvector of the matrix pair (H, M), and $\{v_i\}$ denotes the set of Lanczos vectors. Each subspace solve yields (s_{k+1}, σ_{k+1}), an approximation to a global solution pair in Theorem 2.3.1, and $(\tilde{\lambda}_n^{k+1}, \tilde{u}_n^{k+1})$, an approximation to the leftmost eigenpair of the generalized eigenvalue problem $Hu = \lambda Mu$.

Hager's first phase is similar to the GLTR method with Lanczos restarts at fixed intervals. After each iteration of the first phase, the previous Lanczos vectors are discarded and the Lanczos process is restarted with the residual of the current best approximation to a solution of the trust-region subproblem.

The second phase of Hager's method generates a sequence $\{(s_k, \sigma_k)\}$ that is intended to converge to the optimal (s, σ) associated with the solution of (4.1) (see Theorem 2.3.1, p. 24). At the start of the kth iteration, values (s_k, σ_k) are known such that $\|Ns_k\| = \delta$ and $\sigma_k \in (-\lambda_n(H), \infty)$, Hager defines the $(k+1)$th iterate (s_{k+1}, σ_{k+1}) as a solution of the subspace minimization problem

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) \;\; \text{subject to} \;\; \|Ns\| = \delta, \quad s \in \mathcal{S}_k, \tag{4.10}$$

where $\mathcal{S}_k = \text{span}\{s_k, \tilde{u}_n, \nabla\mathcal{Q}(s_k), s_{SQP}\}$. The vector $\tilde{u}_n$ is the best estimate of the leftmost eigenvector computed in phase one, and s_{SQP} is computed from one step of Newton's method applied to the constrained minimization problem

$$\underset{s \in \mathbb{R}^n}{\text{minimize}} \;\; g^T s + \tfrac{1}{2} s^T H s \;\; \text{subject to} \;\; \tfrac{1}{2}\delta^2 - \tfrac{1}{2} s^T N^T N s = 0. \tag{4.11}$$

(This problem is equivalent to (4.9). The "$\frac{1}{2}$" has been introduced to simplify the derivatives.)

Next, we focus on the definition of s_{SQP}. The Lagrangian associated with problem (4.11) is

$$L(s,\sigma) = g^T s + \tfrac{1}{2} s^T H s - \frac{\sigma}{2}(\delta^2 - s^T N^T N s) = \mathcal{Q}(s) + \sigma r(s),$$

where σ is a scalar Lagrange multiplier and $r(s) = \frac{1}{2}(s^T N^T N s - \delta^2)$. At the point (s,σ), the gradient and Hessian of the Lagrangian are given by

$$\nabla L(s,\sigma) = \begin{pmatrix} (H+\sigma M)s + g \\ r(s) \end{pmatrix} \quad \text{and} \quad \nabla^2 L(s,\sigma) = \begin{pmatrix} H+\sigma M & d \\ d^T & 0 \end{pmatrix},$$

where $M = N^T N$ and $d = \nabla r(s) = N^T N s = Ms$. To simplify the notation, $H(\sigma)$ will be used to denote the Hessian of the Lagrangian with respect to s, i.e., $H(\sigma) = H + \sigma N^T N$.

Since optimal values of s and σ define a stationary point of the Lagrangian, they may be found by applying Newton's method to find a zero of the function $F(s,\sigma) \triangleq \nabla L(s,\sigma)$. Given a solution estimate $w_k = (\bar{s}_k, \bar{\sigma}_k)$, the Newton update is defined as a solution $\Delta w_k = (\Delta s_k, \Delta\sigma_k)$ of the linear system $F'(w_k)\Delta w_k = -F(w_k)$. This gives the next Newton iterate as $w_k + \Delta w_k$, with

$$\begin{pmatrix} H(\bar{\sigma}_k) & d_k \\ d_k^T & 0 \end{pmatrix} \begin{pmatrix} \Delta\bar{s}_k \\ \Delta\bar{\sigma}_k \end{pmatrix} = - \begin{pmatrix} g + H(\bar{\sigma}_k)\bar{s}_k \\ r(\bar{s}_k) \end{pmatrix}. \tag{4.12}$$

The next result gives the conditions under which these Newton equations have a solution.

Lemma 4.4.1 *If $d \neq 0$, then the inertia of the Newton matrix is given by*

$$\mathrm{In}\begin{pmatrix} H(\sigma) & d \\ d^T & 0 \end{pmatrix} = \mathrm{In}\big(H(\sigma)\big) + (0,1,0).$$

In particular, if $\sigma \in (-\lambda_n, \infty)$, where λ_n is the leftmost eigenvalue of $Hu = \lambda Mu$, then the Newton matrix is nonsingular with n positive eigenvalues and one negative eigenvalue. ∎

This result implies that Newton's method is well defined only for values of $\bar{\sigma}_k$ in the interval $(-\lambda_n, \infty)$. Moreover, even with this restriction on $\bar{\sigma}_k$, the Newton equations are indefinite and the CG method cannot be used. Instead, Hager uses a variant of the CG method of Gould, Hribar and Nocedal [17], which is defined for general KKT systems. For this method to be applied, the equations must be modified so that the solution Δs lies in the null space of d^T. Then, under a suitable nonsingularity assumption, if all CG iterates are arranged to lie in null(d^T), then the CG method is well-defined.

In the case of the Newton equations (4.12), the conditions required for the Gould-Hribar-Nocedal method to be applied are simple to arrange. Suppose that the current Newton iterate $\bar{s}_k$ satisfies the trust-region constraint, i.e., $r(\bar{s}_k) = 0$. The Newton equations are then

$$\begin{pmatrix} H(\bar{\sigma}_k) & d_k \\ d_k^T & 0 \end{pmatrix} \begin{pmatrix} \Delta\bar{s}_k \\ \Delta\bar{\sigma}_k \end{pmatrix} = - \begin{pmatrix} g + H(\bar{\sigma}_k)\bar{s}_k \\ 0 \end{pmatrix}, \tag{4.13}$$

and the last equation implies that $d_k^T \Delta s_k = 0$. Consider the projection matrix

$$P = I - d_k d_k^T / \|d_k\|^2,$$

which projects vectors in $\mathbb{R}^n$ onto the set of vectors orthogonal to d_k (i.e., the null space of d_k^T). Applying this projection to (4.13) gives

$$\begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} H(\bar{\sigma}_k) & d_k \\ d_k^T & 0 \end{pmatrix} \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \Delta\bar{s}_k \\ \Delta\bar{\sigma}_k \end{pmatrix} = - \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} g + H(\bar{\sigma}_k)\bar{s}_k \\ 0 \end{pmatrix}, \tag{4.14}$$

which implies that $\Delta\sigma_k$ is arbitrary, and $\Delta\bar{s}_k$ satisfying the positive semidefinite equations

$$PH(\bar{\sigma}_k)P\Delta\bar{s}_k = -P\big(g + H(\bar{\sigma}_k)\bar{s}_k\big). \tag{4.15}$$

It can be shown that applying the CG method to these equations is equivalent to applying the PCG method (Algorithm 3.5.1, p. 54) to $H(\bar{\sigma}_k)\Delta\bar{s}_k = -(g + H(\bar{\sigma}_k)\bar{s}_k)$ with preconditioner $M^{-1} = P\,(= P^2)$. Moreover, the PCG iterates are well-defined, remain in the null-space of d_k^T and converge to the solution of (4.15).

Hager defines an approximate solution s_{SQP} of (4.9) by performing a *single* Newton step with $\bar{s}_k = s_k$, which satisfies $r(s_k) = 0$, as required. The single Newton system is solved for an approximate step $\Delta\bar{s}_k$ using the CG-based method MINRES (Paige and Saunders [38]). The approximate Newton direction s_{SQP} is then defined as $s_{SQP} = s_k + \Delta\bar{s}_k$.

Hager proves that any SSM method based on subspaces containing the vectors s_k, s_{SQP}, $\tilde{u}_n^k$, and $\nabla\mathcal{Q}(s_k)$ is locally, quadratically convergent to a solution of the trust-region subproblem. For more details, see Hager [21].

In a subsequent paper, Hager and Park [22] show that only *three* vectors are needed to define $\mathcal{S}_k$.

Theorem 4.4.1 *If each step of an SSM method uses an $\mathcal{S}_k$ spanned by the vectors s_k, $\nabla\mathcal{Q}(s_k)$ and the leftmost generalized eigenvector u_n, then the SSM method converges to a solution of the equality constrained trust-region subproblem* (4.9).

∎

If s_k is not a stationary point, then including s_k and $\nabla\mathcal{Q}(s_k)$ the subspace guarantees that the next iterate decreases $\mathcal{Q}$ (i.e., $\mathcal{Q}(s_k) < \mathcal{Q}(s_{k-1})$) and gives a point that is at least as good as the restricted steepest descent direction. Similarly, including s_{k-1} and u_n in the subspace ensures that the method will move away from nonoptimal stationary points.

As mentioned above, Hager recommends generating only four or five Lanczos vectors in the first phase to avoid a significant loss of orthogonality. Moreover, as only a few Lanczos vectors are required, they may be stored explicitly. A drawback to this phase is that each iteration has a fixed cost, regardless of how close the current approximate solution is to an acceptable solution.

Hager uses projection to change the indefinite system (4.12) into a positive semidefinite system (4.15). If $r(s) = 0$ and $d^T\Delta s = 0$ then $s_{SQP} = s + \Delta s$ does not satisfy the trust-region constraint. Hence, Hager is unable to repeatedly solve the Newton equations because the first iterate will not satisfy the trust-region

constraint. One advantage of applying the reduced subspace solve after each SQP iterate is that the subspace solve yields an iterate s_{k+1} that satisfies the trust-region constraint. Thus, a new SQP iterate can be defined once again.

Hager and Park point out that even though only three vectors (s_{k-1}, $\nabla\mathcal{Q}(s_{k-1})$, and u_n) are required for convergence, including the SQP vector in the subspace significantly improves the convergence rate. In fact, Hager proves quadratic convergence when the SQP vector is included in the subspace; however, Hager and Park are only able to prove linear convergence when the SQP vector is not included in the subspace.

The Hager-Park result is primarily of theoretical interest because it relies on having the exact leftmost generalized eigenpair of $Hu = \lambda Mu$. In practice, approximating the leftmost eigenpair is equivalent to solving the trust-region subproblem (4.4) with $g = 0$, i.e.,

$$\underset{z\in\mathbb{R}^n}{\text{minimize}} \;\; E(z) = \tfrac{1}{2}z^THz \quad \text{subject to} \;\; \|Nz\| \leq \delta.$$

Thus, although Hager and Park prove theoretical convergence to a solution of the constrained subproblem, it is difficult to utilize the convergence results in practice.

5

An Iterative Trust-Region Method

5.1 Introduction

There are two fundamental criteria used to evaluate iterative trust-region methods: (i) how does the method compares to methods based on direct factorization (e.g., the Moré-Sorensen algorithm); and (ii) whether the iterative method is efficient. The criteria used to compare iterative and direct trust-region methods is the number of function evaluations; the fewer the function evaluations on average, the better the method. Even if an iterative trust-region method is able to solve the subproblem as accurately as a direct method, the question remains whether the method is relatively efficient compared to other iterative trust-region methods. As the dimension of the problem increases, the time required to perform basic linear algebra tasks increases. The predominant linear algebra cost is in performing matrix-vector products; thus, efficiency is measured in terms of the number of matrix-vector products.

The proposed trust-region subproblem solver has two phases. The first phase is based on Steihaug's method; it applies Lanczos-CG to minimize $\mathcal{Q}(s)$. This phase

will terminate with either a point inside the trust-region boundary that satisfies the convergence criteria, or the phase will terminate when either a direction of negative curvature is encountered or a Lanczos-CG step leaves the trust region. In the latter two cases, rather than follow the direction of negative curvature or the last Lanczos-CG iterate to the boundary as in Steihaug's method, the first phase obtains an approximate solution on the boundary by minimizing $\mathcal{Q}(s)$ over a three-dimensional subspace. The cost for the first phase is one matrix-vector product with H for each Lanczos-CG iterate.

The second phase allow for accuracy control in the constrained case; it is entered only in the case when the solution(s) of the trust-region subproblem lies on the boundary. This phase uses two approaches to solve the constrained trust-region subproblem. First, it approximates the solution of an evolving related constrained trust-region subproblem. Second, it solves the original trust-region subproblem over a sequence of three-dimensional subspaces. These two approaches are combined in such a way that they are made dependent upon each other—accelerating the overall rate of convergence of both approaches. The cost for this phase is also one matrix-vector product with H each Lanczos-CG iterate.

The proposed method may terminate in either phase with an approximate solution of the trust-region subproblem.

5.2 Phase One

In the first phase, the proposed method follows the PCG iterates until an approximate interior solution is found or it becomes evident that the solution lies on the trust-region boundary. The Lanczos-PCG method is used to generate the PCG iterates, and the Lanczos vectors are also used as *direction generators* to obtain an approximation to the leftmost generalized eigenpair (λ_n, u_n) of the eigenvalue problem $Hu = \lambda Mu$ (see Section 5.2.1). The Lanczos-PCG iterates are computed until the quantities s_k, p_k and α_k associated with the kth PCG iteration satisfy

one of the four conditions:

$(\mathbf{T}_1')$ $s_k + \alpha_k p_k$ is an approximate minimizer of $\mathcal{Q}(s)$ such that $\|s_k + \alpha_k p_k\|_M < \delta$;

$(\mathbf{T}_2')$ $\|s_k + \alpha_k p_k\|_M \geq \delta$;

$(\mathbf{T}_3')$ $p_k^T H p_k \leq 0$;

$(\mathbf{T}_4')$ $\tilde{\lambda}_n^{k+1} \leq 0$.

If termination occurs because of condition $(\mathbf{T}_1')$ then the phased-SSM method is terminated with the approximate solution $s_k + \alpha_k p_k$. If termination occurs because of $(\mathbf{T}_2')$, $(\mathbf{T}_3')$ or $(\mathbf{T}_4')$, the trust-region solution must lie on the boundary and a single constrained subspace minimization problem is solved for an exit point s_{k+1}. This problem is given by

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) \quad \text{subject to} \;\; \|Ns\| = \delta, \;\; s \in \text{span}\{s_k, p_k, \tilde{u}_n^{k+1}\}, \tag{5.1}$$

where s_k is the last interior PCG iterate, p_k is the last PCG direction, and $\tilde{u}_n^{k+1}$ is the current approximation to the leftmost eigenvector of the eigenvalue problem $Hu = \lambda Mu$ (for details of the method applied to a similar problem, see Section 5.3.5). The inclusion of the last PCG iterate s_k in the subspace guarantees that the final phase-one iterate satisfies $\mathcal{Q}(s_{k+1}) < \mathcal{Q}(s_k)$.

The three-dimensional reduced problem (5.1) may be solved approximately using the Moré-Sorensen method. The solution of the reduced problem is then expanded to $\mathbb{R}^n$ to obtain s_{k+1}. The Moré-Sorensen algorithm also provides σ_{k+1}, the first estimate of σ for the second phase.

If one of the conditions $(\mathbf{T}_2')$ or $(\mathbf{T}_3')$ hold, the exit point is at least as good as the Steihaug point because the exit point minimizes $\mathcal{Q}(s)$ over a larger subspace. Note that this phase may terminate earlier than Steihaug's method because of the additional test on the approximate leftmost eigenvalue.

The proposed method retains many of the properties of Steihaug's method. First, if $\|NH^{-1}g\| < \delta$ and H is positive definite, then the algorithm is identical to

the Lanczos-PCG method. Termination with condition ($\mathbf{T}_1'$) is likely to occur with increasing frequency as the outer iterations converge. Second, the first Lanczos-CG step is the Cauchy point; thus, the underlying trust-region method is guaranteed to converge to a first-order point.

The cost associated with this phase is one matrix-vector product to form each Lanczos vector. The Lanczos-PCG routine requires the storage of two Lanczos vectors and their product with H (for details, see Sections 3.6 and 3.7).

5.2.1 Estimating the leftmost eigenpair

The leftmost eigenpair estimation routine approximates the leftmost generalized eigenpair of the matrix pair (H, M) by solving the eigenproblem over a reduced space. In particular, the generalized eigenvalue problem is solved over the subspace

$$Y_k = \text{span}\{v_{k+1}, v_k, \tilde{u}_n^{k+1}\},$$

where v_{k+1} and v_k are the two most current Lanczos vectors and $\tilde{u}_n^{k+1}$ is the previous estimate of the leftmost generalized eigenvector. The reduced problem is at most three dimensional, and thus, can be solved via a direct method.

By including the current approximation to the leftmost generalized eigenvector in the reduced space, the leftmost eigenvector routine is guaranteed to generate a sequence of monotonically decreasing estimates of the leftmost eigenvalue, i.e., $\tilde{\lambda}_n^{k+1} \leq \tilde{\lambda}_n^k$.

There is no additional cost for estimating leftmost eigenpair. In particular, if the columns of Q are formed from a subset of the vectors $\{v_k, v_{k-1}, \tilde{u}_n^{k-1}\}$, no additional matrix-vector products are required. To see this, note that the reduced generalized eigenvalue problem may be written as the generalized eigenvalue problem associated with the pencil $Q^THQ - \lambda Q^TMQ$, where Q is a matrix whose columns are a basis for $\text{span}\{v_k, v_{k-1}, \tilde{u}_n^{k-1}\}$. If the leftmost generalized eigenvector in the reduced space is $\bar{u}_k$, then $\tilde{u}_n^k = Q\bar{u}_k$ and $\tilde{u}_n^k$ is a linear combination of the vectors v_k, v_{k-1} and $\tilde{u}_n^{k-1}$.

Each approximate eigenpair iteration requires the matrices Q^THQ and Q^TMQ. In Section 3.6, we saw how to compute Hv_k, Hv_{k-1}, Mv_k, and Mv_{k-1}. If the columns of Q are made up of a subset of the vectors v_k, v_{k-1} and $\tilde{u}_n^{k-1}$, then with the vectors $H\tilde{u}_n^{k-1}$ and $M\tilde{u}_n^{k-1}$, it is trivial to explicitly form Q^THQ and Q^TMQ. However, notice that since $\tilde{u}_n^k$ is a linear combination of the vectors v_k, v_{k-1} and $\tilde{u}_n^{k-1}$, then $H\tilde{u}_n^k$ and $M\tilde{u}_n^k$ are also linear combinations of the products of H and M, respectively, with the vectors v_k, v_{k-1} and $\tilde{u}_n^{k-1}$.

The initial estimate $\tilde{u}_n^0$ can be taken to be the leftmost generalized eigenvector estimate obtained during the solution of the previous trust-region subproblem. Thus, the initial generalized eigenvalue problem is solved over the subspace $\text{span}\{v_0, \tilde{u}_n^0\}$. As the trust-region solver converges and the sequence $\{H_k\}$ converges, $\tilde{u}_n^0$ will be a good initial estimate for the leftmost eigenvector of the current Hessian.

In order to include $\tilde{u}_n^0$ in the subspace (and to allow the efficient formation of Q^THQ and Q^TMQ), it is necessary to know the vectors $H\tilde{u}_n^0$ and $M\tilde{u}_n^0$. The vector $H\tilde{u}_n^0$ is computed by direct multiplication. However, it is not possible to compute $M\tilde{u}_n^0$ without knowing M. Thus, for the $(j+1)$th subproblem, the initial estimate $\tilde{u}_n^0$ of the current generalized eigenvector is defined to be

$$\tilde{u}_n^0 \triangleq M_{j+1}^{-1}[M\tilde{u}]^{(j)}, \tag{5.2}$$

where $[M\tilde{u}]^{(j)}$ estimates the product of M_j and $\tilde{u}_n^{(j)}$, and $\tilde{u}_n^{(j)}$ denotes the final estimate of a leftmost eigenvector for H_j. The idea is that as the trust-region algorithm converges, not only does the sequence $\{H_j\}$ converge, but also the sequence $\{M_j\}$ converges; in this case, $\tilde{u}_n^0 = M_{j+1}^{-1}[M\tilde{u}]^{(j)} \approx \tilde{u}_n^{(j)}$. This choice of $\tilde{u}_n^0$ yields the following convenient definition:

$$[M\tilde{u}_n^0] = M_{j+1}M_{j+1}^{-1}[M\tilde{u}]^{(j)} = [M\tilde{u}]^{(j)}.$$

5.3 Phase Two

The kth iteration of the second phase generates a new point s_{k+1} by solving the subspace minimization problem

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) = g^T s + \tfrac{1}{2} s^T H s \;\; \text{subject to} \;\; \|s\|_M \leq \delta, \quad s \in \mathcal{S}_k, \tag{5.3}$$

where $\mathcal{S}_k = \text{span}\{s_k, \bar{s}_k, \tilde{u}_n^k\}$ and $\bar{s}_k$ is defined as *one step* of Newton's method applied to a related constrained problem.

Hager [21] includes the subspace basis vector s_{SQP} that is one step of Newton's method for the constrained problem

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) = g^T s + \tfrac{1}{2} s^T H s \;\; \text{subject to} \;\; \tfrac{1}{2}\delta^2 - \tfrac{1}{2} s^T M s = 0. \tag{5.4}$$

Newton's method is used to compute a stationary point of the Lagrangian

$$L(s, \sigma) = \mathcal{Q}(s) + \sigma r(s), \tag{5.5}$$

where $r(s) = \frac{1}{2}(s^T M s - \delta^2)$ and σ is a nonnegative Lagrange multiplier. Each step of Newton's method requires the solution of an *indefinite* system with matrix

$$\begin{pmatrix} H(\sigma) & d \\ d^T & 0 \end{pmatrix}.$$

Instead of solving an indefinite system, the second phase solves a related system that may be viewed as a *regularized* Newton system. The aim is to define a step satisfying a related positive-definite system that can be solved using the CG method.

5.3.1 The augmented penalty function

An alternative to using Newton's method to find a root of $\nabla L(s, \sigma)$ is to minimize the quadratic penalty function

$$P_2(s) = \mathcal{Q}(s) + \frac{1}{2\mu} r(s)^2$$

for a sequence of decreasing positive penalty parameters μ. A minimizer $s = s(\mu)$ must satisfy the first-order condition

$$\nabla P_2(s) = g + Hs + \frac{1}{\mu} r(s) N^T N s = 0.$$

If σ denotes the auxiliary quantity $\sigma = r(s)/\mu$, then the penalty minimizer $s(\mu)$ may be computed as part of the root $\big(s(\mu), \sigma(\mu)\big)$ of the nonlinear equations $G(s, \sigma) = 0$, where

$$G(s, \sigma) = \begin{pmatrix} g + (H + \sigma N^T N)s \\ r(s) - \mu\sigma \end{pmatrix}. \tag{5.6}$$

Forsgren and Gill [8] propose the augmented quadratic penalty function

$$P_\mu(s, \sigma) = \mathcal{Q}(s) + \frac{1}{2\mu} r(s)^2 + \frac{1}{2\mu}(\mu\sigma - r(s))^2, \tag{5.7}$$

which is the usual quadratic penalty function augmented with a term that penalizes deviations of $r(s)$ from $\mu\sigma$. The gradient and Hessian of $P_\mu(s, \sigma)$ may be written in the form

$$\nabla P_\mu(s, \sigma) = \begin{pmatrix} g + H(\hat{\sigma})s \\ \mu\sigma - r(s) \end{pmatrix}$$

and

$$\nabla^2 P_\mu(s, \sigma) = \begin{pmatrix} H(\hat{\sigma}) + \frac{2}{\mu} N^T N s s^T N^T N & -N^T N s \\ -s^T N^T N & \mu \end{pmatrix},$$

where $H(\sigma) = H + \sigma N^T N$ and $\hat{\sigma} = \sigma + 2\big(r(s) - \sigma\mu\big)/\mu$. If we define the intermediate vector $\pi = \pi(s) = r(s)/\mu$, we may write

$$\hat{\sigma} = \sigma + 2(\pi(s) - \sigma). \tag{5.8}$$

A key property is that a minimizer $(s(\mu), \sigma(\mu))$ of $P_\mu(\sigma, \mu)$ is a zero of $G(s, \sigma)$, and hence satisfies the first-order necessary conditions for a minimizer of the quadratic penalty function. Moreover, if $H + \sigma N^T N$ is positive definite then $(s(\mu), \sigma(\mu))$ is a minimizer of $P_\mu(s, \sigma)$ if and only if $s(\mu)$ is a minimizer of $P_2(s)$. (For further details on the augmented quadratic penalty function P_μ, see Forsgren and Gill [8] and Gertz and Gill [12].)

Given an approximate solution $w_k = (\bar{s}_k, \bar{\sigma}_k)$, the Newton step associated with minimizing $P_\mu(s, \sigma)$ is the solution $\Delta w_k = (\Delta s_k, \Delta\sigma_k)$ of the linear system

$$\begin{pmatrix} H(\hat{\sigma}_k) + \frac{2}{\mu} d_k d_k^T & d_k \\ d_k^T & \mu \end{pmatrix} \begin{pmatrix} \Delta\bar{s}_k \\ -\Delta\bar{\sigma}_k \end{pmatrix} = - \begin{pmatrix} g + H(\hat{\sigma}_k)\bar{s}_k \\ r(\bar{s}_k) - \mu\bar{\sigma}_k \end{pmatrix}, \tag{5.9}$$

where d_k denotes the vector $d_k = N^T N \bar{s}_k$.

As $(\bar{s}_k, \bar{\sigma}_k) \to \big(s(\mu), \sigma(\mu)\big)$, it holds that $r(\bar{s}_k)/\mu \to \sigma(\mu)$ and hence $\pi(\bar{s}_k) \to \sigma(\mu)$ and $\hat{\sigma}_k \to \sigma(\mu)$ (see, e.g., (5.8)). It follows that the matrix $H(\hat{\sigma})$ on the left-hand side of (5.9) may be approximated by $H(\sigma)$, giving the *approximate* Newton system

$$\begin{pmatrix} H(\bar{\sigma}_k) + \frac{2}{\mu} d_k d_k^T & d_k \\ d_k^T & \mu \end{pmatrix} \begin{pmatrix} \Delta\bar{s}_k \\ -\Delta\bar{\sigma}_k \end{pmatrix} = - \begin{pmatrix} g + H(\hat{\sigma})\bar{s}_k \\ r(\bar{s}_k) - \mu\bar{\sigma}_k \end{pmatrix}. \tag{5.10}$$

If both sides of this equation are multiplied by the nonsingular matrix

$$\begin{pmatrix} I & -\frac{2}{\mu} d_k \\ 0 & 1 \end{pmatrix},$$

and the last column is scaled by -1, we obtain

$$\begin{pmatrix} H(\bar{\sigma}_k) & d_k \\ d_k^T & -\mu \end{pmatrix} \begin{pmatrix} \Delta\bar{s}_k \\ \Delta\bar{\sigma}_k \end{pmatrix} = - \begin{pmatrix} g + H(\bar{\sigma}_k)\bar{s}_k \\ r(\bar{s}_k) - \mu\bar{\sigma}_k \end{pmatrix},$$

which is a perturbation of the SQP Newton equation (4.12) (p. 70). Equation (5.10) may be viewed as a regularization of the SQP Newton equation (4.12).

There are several advantages of solving (5.10) rather than the SQP Newton equations (4.12). First, provided $H(\sigma)$ is positive semidefinite with $\sigma > 0$ and $\mu > 0$, then (5.10) is positive definite. To see this, consider the following decomposition of the left-hand side:

$$\begin{pmatrix} H(\bar{\sigma}_k) + \frac{2}{\mu} d_k d_k^T & -d_k \\ -d_k^T & \mu \end{pmatrix} = \begin{pmatrix} I & -\frac{1}{\mu} d_k \\ 0 & 1 \end{pmatrix} \begin{pmatrix} H(\bar{\sigma}_k) + \frac{1}{\mu} d_k d_k^T & 0 \\ 0 & \mu \end{pmatrix} \begin{pmatrix} I & 0 \\ -\frac{1}{\mu} d_k^T & 1 \end{pmatrix}. \tag{5.11}$$

It follows that if $H(\bar{\sigma}_k)$ is positive definite, then (5.10) can be solved using a preconditioned CG method.

Theorem 5.3.1 *Suppose that $p^T H(\sigma) p > 0$ for all vectors $p \neq 0$ such that $d^T p = 0$. Then $H(\sigma) + \frac{2}{\mu} dd^T$ is positive definite for all $\mu < \bar{\mu}$, where*

$$\frac{1}{\bar{\mu}} = -\frac{1}{2} \min\{0, \lambda_n(H(\sigma))\} / \|d\|_2^2,$$

and $\lambda_n(H(\sigma))$ denotes the leftmost eigenvalue of $H(\sigma)$.

Proof. Let $H(\sigma) = Q \Lambda Q^T$ denote the spectral decomposition of $H(\sigma)$, where $Q = \begin{pmatrix} q_1 & q_2 & \cdots & q_n \end{pmatrix}$ and $\Lambda = \operatorname{diag}\big(\lambda_i(H(\sigma))\big)$ with $\lambda_1(H(\sigma)) \geq \cdots \geq \lambda_n(H(\sigma))$.

Let p be any vector of unit norm such that $d^T p \neq 0$ (otherwise, it holds that $p^T\big(H(\sigma) + \frac{2}{\mu} dd^T\big) p = p^T H(\sigma) p > 0$ and the result holds immediately). Then, there exist scalars β_i such that

$$p = \sum_{i=1}^{n} \beta_i q_i, \quad \text{with} \quad \sum_{i=1}^{n} \beta_i = 1,$$

and we may write

$$p^T H(\sigma) p = \Big(\sum_{i=1}^{n} \beta_i q_i^T \Big) H(\sigma) \Big(\sum_{i=1}^{n} \beta_i q_i \Big) = \sum_{i=1}^{n} \beta_i^2 \lambda_i(H(\sigma)) > 0.$$

Then, for all p such that $d^T p \neq 0$, it holds that

$$\begin{aligned} p^T \Big(H(\sigma) + \frac{2}{\mu} dd^T \Big) p &= \sum_{i=1}^{n} \beta_i^2 \lambda_i(H(\sigma)) + \frac{2}{\mu} (d^T p)^2 \\ &\geq \sum_{i=1}^{n} \beta_i^2 \lambda_i(H(\sigma)) + \frac{2}{\mu} \|d\|_2^2 \\ &> \sum_{i=1}^{n} \beta_i^2 \min\{0, \lambda_n(H(\sigma))\} + \frac{2}{\mu} \|d\|_2^2 \\ &= \min\{0, \lambda_n(H(\sigma))\} + \frac{2}{\mu} \|d\|_2^2. \end{aligned}$$

It follows that $p^T(H(\sigma) + \frac{2}{\mu} dd^T) p > 0$ for all $\mu < \bar{\mu}$, where

$$\frac{1}{\bar{\mu}} = \frac{-\min\{0, \lambda_n(H(\sigma))\}}{2\|d\|_2^2},$$

as required. ∎

The following result shows that the scalar μ serves as *regularization* parameter in the so-called "hard case" (see Section 2.3.5).

Theorem 5.3.2 (Regularization of the hard-case.) *Let (s^*, σ^*) denote a solution of the trust-region problem:*

$$\underset{s}{\text{minimize}} \;\; g^T s + \tfrac{1}{2} s^T H s \;\; \text{subject to} \;\; \|s\|_M \leq \delta,$$

in the situation where $H(\sigma^*) = H + \sigma^* M$ is positive semidefinite and singular, $g \in \text{null}(H(\sigma^*))^\perp$ and $\|(H + \sigma^* M)^\dagger g\| < \delta$. If the leftmost generalized eigenvalue of the matrix pair (H, M) is simple, then the augmented system matrix (with $d = Ms^*$)

$$\begin{pmatrix} H(\sigma^*) + \frac{2}{\mu} dd^T & -d \\ -d^T & \mu \end{pmatrix}$$

is positive definite.

Proof. This is the hard case; thus, $\sigma^* = -\lambda_n$ where λ_n is the leftmost generalized eigenvalue of the matrix pair (H, M). The solution s^* of the trust-region subproblem is given by

$$s^* = -(H - \lambda_n M)^\dagger g + \beta z, \tag{5.12}$$

where z is a unit vector such that $z \in \text{null}(H - \lambda_n M)$ and β is a nonzero scalar such that $\|s^*\|_M = \delta$.

Suppose that the generalized eigenvalues of the matrix pair (H, M) are such that

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > \lambda_{r+1} = \cdots = \lambda_n,$$

i.e., the left-most eigenvalue λ_n has multiplicity $n - r$. Let the generalized spectral decomposition of H be denoted by $H = U \Lambda U^T$, where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ and $U^T M^{-1} U = I$. Consider the matrix V such that $V = U^{-T}$ and partition V so that $V = (V_r \;\; V_{n-r})$, where V_r is $n \times r$ and V_{n-r} is $n \times (n - r)$. The columns of

V_r and V_{n-r} form bases for range$(H - \lambda_n M)$ and null$(H - \lambda_n M)$, respectively. We have

$$\begin{aligned} p &= -(H - \lambda_n M)^{\dagger} g \\ &= -U^{-T} \begin{pmatrix} (\Lambda_r - \lambda_n I_r)^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^{-1} g, \end{aligned}$$

and $z = V_{n-r} q$ for some nonzero $(n-r)$-vector q.

From (5.11) we have that the inertia of the augmented system is determined by the inertia of $H(\sigma^*) + \frac{1}{\mu} dd^T$, i.e.,

$$\begin{aligned} H(\sigma^*) + \frac{1}{\mu} dd^T &= U \begin{pmatrix} (\Lambda_r - \lambda_n I) & 0 \\ 0 & 0 \end{pmatrix} U^T + \frac{1}{\mu} dd^T \\ &= U \left(\begin{pmatrix} \Lambda_r - \lambda_n I_r & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{\mu} vv^T \right) U^T, \end{aligned}$$

where $v = U^{-1} d$. Using (5.12), we obtain

$$v = (U^T M U) U^T s^* = U^T M U \begin{pmatrix} -(\Lambda_r - \lambda_n I_r)^{-1} U_r^T g \\ \beta q \end{pmatrix}.$$

The eigenvalue interlacing theorem (see, e.g., Wilkinson [50, p.97]) and Sylvester's law of inertia now gives

$$\text{In}\left(H(\sigma^*) + \frac{1}{\mu} dd^T\right) = (r+1, 0, n-r-1).$$

If λ_n is a simple eigenvalue, then $r = n - 1$ and the result follows directly. ∎

These theorems imply that a CG-type algorithm will minimize $P_\mu(s, \sigma)$ until a conjugate direction (p_1, p_2) is obtained such that $p_1^T H(\sigma) p_1 \leq 0$. Intuitively this means that if μ is sufficiently small, the function P_μ does not introduce additional directions of negative or zero curvature into the subproblem.

A preconditioner that removes any ill-conditioning resulting from a small value of μ is given by

$$P = \begin{pmatrix} M + \frac{2}{\mu} dd^T & d \\ d^T & \mu \end{pmatrix},$$

where M is a preconditioner for $H(\sigma)$. Consider the case when M is a positive-definite diagonal approximation to $H(\sigma)$. The equations $Pv = r$ used to apply the preconditioner are solved by exploiting the equivalence of the systems:

$$\begin{pmatrix} M + \frac{2}{\mu} dd^T & d \\ d^T & \mu \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}, \tag{5.13a}$$

$$\text{and} \quad \begin{pmatrix} M & -d \\ -d^T & -\mu \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} r_1 - \frac{2r_2}{\mu} d \\ -r_2 \end{pmatrix}. \tag{5.13b}$$

As M is diagonal and $d = Ms$ is a column vector, this system may be solved analytically.

Unlike Hager's projected CG-type method to solve the SQP Newton equations, this approach does not require that the constraint is satisfied exactly, i.e., this approach does not require that $r(s) = 0$.

5.3.2 The primal-dual augmented Lagrangian

Given a positive scalar μ and a nonnegative scalar σ_e, consider the function of $n + 1$ variables:

$$L_{\mu,\sigma_e}(s, \sigma) = \mathcal{Q}(s) + \sigma_e r(s) + \frac{1}{2\mu} r(s)^2 + \frac{1}{2\mu} \big(\mu(\sigma - \sigma_e) - r(s)\big)^2.$$

The gradient and Hessian of $L_{\mu,\sigma_e}(s, \sigma)$ with respect to (s, σ) are

$$\nabla L_{\mu,\sigma_e}(s, \sigma) = \begin{pmatrix} g + H(\hat{\sigma})s \\ \mu(\sigma - \pi) \end{pmatrix},$$

and

$$\nabla^2 L_{\mu,\sigma_e}(s, \sigma) = \begin{pmatrix} H(\hat{\sigma}) + \frac{2}{\mu} dd^T & -d \\ -d^T & \mu \end{pmatrix},$$

where $\hat{\sigma} = \sigma + 2(\pi - \sigma)$ with $\pi = \pi(s) = \sigma_e + r(s)/\mu$.

A key property of $L_{\mu,\sigma_e}(s, \sigma)$ is that a solution of the trust-region subproblem is a minimizer of $L_{\mu,\sigma_e}(s, \sigma)$ with respect to both (s, σ).

Theorem 5.3.3 *Let (s^*, σ^*) be a solution of the trust-region subproblem (4.4) with $\|s^*\|_M = \delta$, then for all $\mu > 0$, the point (s^*, σ^*) minimizes the function*

$$L^*_{\mu,\sigma_e}(s,\sigma) = \mathcal{Q}(s) + \sigma^* r(s) + \frac{1}{2\mu} r(s)^2 + \frac{1}{2\mu}\big(\mu(\sigma - \sigma^*) - r(s)\big)^2.$$

Proof. We verify the first-order and second-order sufficient conditions for a minimizer. First, differentiating $L^*_{\mu,\sigma_e}(s,\sigma)$ with respect to s yields

$$\nabla_s L^*_{\mu,\sigma_e}(s,\sigma) = g + Hs + \sigma^* Ms + \frac{1}{\mu} r(s)d + \frac{1}{\mu}(\mu(\sigma^* - \sigma) + r(s))d.$$

Since $r(s^*) = 0$, then

$$\nabla_s L^*_{\mu,\sigma_e}(s^*,\sigma^*) = g + H(\sigma^*)s^* = 0.$$

Differentiating with respect to σ and evaluating at the point (s^*, σ^*) yields

$$\nabla_\sigma L^*_{\mu,\sigma_e}(s^*,\sigma^*) = \mu(\sigma^* - \sigma^*) - r(s^*) = 0.$$

Now, we verify the second-order conditions for an unconstrained minimizer. Observe that

$$\nabla^2_{s,\sigma} L^*_{\mu,\sigma_e}(s,\sigma) = -d, \quad \nabla^2_{\sigma,s} L^*_{\mu,\sigma_e}(s,\sigma) = -d^T, \quad \nabla^2_{\sigma,\sigma} L^*_{\mu,\sigma_e}(s,\sigma) = \mu.$$

And,

$$\begin{aligned}\nabla^2_{s,s} L^*_{\mu,\sigma_e}(s,\sigma) &= H + \sigma^* M + \frac{2}{\mu}(dd^T + r(s)M) + (\sigma^* - \sigma)M \\ &= H(\sigma) + 2(\sigma^* - \sigma)M + \frac{2}{\mu}(dd^T + r(s)M).\end{aligned}$$

Hence, $\nabla^2 L^*_{\mu,\sigma_e}(s,\sigma)$ evaluated at the point (s^*, σ^*) is given by

$$\begin{pmatrix} H(\sigma^*) + \frac{2}{\mu} dd^T & -d \\ -d^T & \mu \end{pmatrix}.$$

The inertia of this matrix is given by a factorization analogous to (5.11). Since $H(\sigma^*)$ is positive semidefinite, $H(\sigma^*) + \frac{1}{\mu} dd^T$ is positive semidefinite for all $\mu > 0$, implying $\nabla^2 L^*_{\mu,\sigma_e}(s^*,\sigma^*)$ is positive semidefinite.

Hence, the point (s^*, σ^*) satisfies the necessary conditions for an unconstrained minimizer of L^*_{μ,σ_e}. ∎

This result suggests that, given an approximate value σ_e, a good strategy to find an approximate solution to the trust-region subproblem is to minimize

$$L_{\mu,\sigma_e}(s,\sigma) = \mathcal{Q}(s) + \sigma_e r(s) + \frac{1}{2\mu} r(s)^2 + \frac{1}{2\mu}\big(\mu(\sigma - \sigma_e) - r(s)\big)^2$$

with respect to both (s, σ). The Newton equations for minimizing $L_{\mu,\sigma_e}(s,\sigma)$ are:

$$\begin{pmatrix} H(\hat{\sigma}) + \frac{2}{\mu} dd^T & -d \\ -d^T & \mu \end{pmatrix} \begin{pmatrix} \Delta s \\ \Delta\sigma \end{pmatrix} = - \begin{pmatrix} g + H(\hat{\sigma})s \\ \mu(\sigma - \pi) \end{pmatrix},$$

or, equivalently,

$$\begin{pmatrix} H(\hat{\sigma}) + \frac{2}{\mu} dd^T & d \\ d^T & \mu \end{pmatrix} \begin{pmatrix} \Delta s \\ -\Delta\sigma \end{pmatrix} = - \begin{pmatrix} g + H(\hat{\sigma})s \\ \mu(\pi - \sigma) \end{pmatrix}.$$

where, as before, $\hat{\sigma} = \sigma + 2(\pi - \sigma)$ with $\pi = \pi(s) = \sigma_e + r(s)/\mu$.

This regularization needs two parameters: μ and σ_e. The value of σ_e must be chosen so that if $H(\sigma)$ is positive definite, then the Newton system is positive definite. As the second phase converges, $\pi \approx \sigma$ and this system will be positive definite provided $H(\sigma)$ is positive definite; however, far from a solution to the trust-region subproblem, if σ_e is much smaller than σ, then $\pi = \sigma_e + r(s)/\mu \ll \sigma$ and this system can be indefinite. Nevertheless, additional care can be taken to guarantee that this system is positive definite whenever $H(\sigma)$ is positive definite (for more details, see Section 5.3.7).

As the second phase converges, $\sigma_e \approx \sigma^*$, and thus, μ need not approach zero. In particular, if σ_e is the optimal σ, then one minimization is required without μ needing to be small. This implies that it is unnecessary to precondition for μ, which is required for the penalty function regularization (see (5.13)). However, if σ_e does not converge to the optimal σ, then μ must go to zero.

There are two beneficial properties of the function $L_{\mu,\sigma_e}(s,\sigma)$. First, a good value for σ_e is available from the reduced trust-region subproblem solve (see Section 5.3.5). Second, an explicit lower bound σ_L may be imposed on σ by including

a barrier term in the definition of L_{μ,σ_e}. This is beneficial because the Newton system is positive definite if $\sigma > -\lambda_n$, where λ_n denotes the exact leftmost generalized eigenvalue of the matrix pair (H, M).

Consider the function

$$L^{\nu}_{\mu,\sigma_e}(s,\sigma) = \mathcal{Q}(s) + \sigma_e r(s) + \frac{1}{2\mu} r(s)^2 + \frac{1}{2\mu}\big(\mu(\sigma - \sigma_e) - r(s)\big)^2 - \nu \ln(\sigma - \sigma_L),$$

where ν is a small positive constant. The gradient and Hessian of $L^{\nu}_{\mu,\sigma_e}(s,\sigma)$ with respect to (s,σ) are

$$\nabla L^{\nu}_{\mu,\sigma_e}(s,\sigma) = \begin{pmatrix} g + H(\hat{\sigma})s \\ \mu(\sigma - \pi) - \nu/(\sigma - \sigma_L) \end{pmatrix},$$

and

$$\nabla^2 L^{\nu}_{\mu,\sigma_e}(s,\sigma) = \begin{pmatrix} H(\hat{\sigma}) + \frac{2}{\mu} dd^T & -d \\ -d^T & \mu + \nu/(\sigma - \sigma_L)^2 \end{pmatrix}.$$

This gives the Newton equations

$$\begin{pmatrix} H(\hat{\sigma}) + \frac{2}{\mu} dd^T & d \\ d^T & \mu + \nu/(\sigma - \sigma_L)^2 \end{pmatrix} \begin{pmatrix} \Delta s \\ -\Delta\sigma \end{pmatrix} = - \begin{pmatrix} g + H(\hat{\sigma})s \\ \mu(\pi - \sigma) + \nu/(\sigma - \sigma_L) \end{pmatrix},$$

with $\hat{\sigma} = \sigma + 2(\pi - \sigma)$.

The function L^{ν}_{μ,σ_e} shares many of the advantages of L_{μ,σ_e}. The Newton system arising from minimizing L^{ν}_{μ,σ_e} with respect to both (s,σ) can be made positive definite whenever $\sigma > -\lambda_n$, and thus, a CG-type method may be used to solve the Newton equations (see Section 5.3.7). Another advantage of both schemes is that the initial pair (s,σ) need not be feasible, i.e., $\|s\|_M \neq \delta$.

The optimal σ^* must lie in the interval $[-\lambda_n, \infty)$; thus, σ_L is initially set to be

$$\sigma_L = \max\{-\tilde{\lambda}_n, 0\},$$

and is then updated by a suitable safeguarding algorithm. There are two possible sources of updates for σ_L. First, if the Newton system is determined to be indefinite for a trial value of σ, then σ_L may be set to this value. Second, a new

estimate of the leftmost generalized eigenvalue may provide a better value of σ_L (see Section 5.3.3). In the hard case, the function L^{ν}_{μ,σ_e} may become undefined if $\sigma \approx \sigma_L$; thus, the lower bound σ_L may need to be reset to be made slightly smaller than the approximate leftmost generalized eigenvalue.

In practice, it is not efficient to solve the Newton equations very accurately. If the current approximate subproblem solution is close to an optimal solution, only a few iterations of a CG-type algorithm may be needed to sufficiently solve the subproblem.

5.3.3 Minimizing the primal-dual augmented Lagrangian

Each iteration of the second phase involved the computation of *one* Newton step for a minimizer of

$$L^{\nu}_{\mu,\sigma_e}(s,\sigma) = \mathcal{Q}(s) + \sigma_e r(s) + \frac{1}{2\mu} r(s)^2 + \frac{1}{2\mu}\big(\mu(\sigma - \sigma_e) - r(s)\big)^2 - \nu \ln(\sigma - \sigma_L).$$

In order to focus on this one step, we will use (s,σ) to denote the kth Newton iterate, and $(\Delta s, \Delta\sigma)$ to denote the kth Newton direction.

The Lanczos-PCG method is used to find an approximate solution of the Newton equations

$$\begin{pmatrix} H(\hat{\sigma}) + \frac{2}{\mu} dd^T & d \\ d^T & \mu + \nu/(\sigma - \sigma_L)^2 \end{pmatrix} \begin{pmatrix} \Delta s \\ -\Delta\sigma \end{pmatrix} = - \begin{pmatrix} g + H(\hat{\sigma}) s \\ \mu(\pi - \sigma) + \nu/(\sigma - \sigma_L) \end{pmatrix},$$

where $\hat{\sigma} = \sigma + 2(\pi - \sigma)$. A block preconditioner is used to exploit the structure of these equations. If M is a preconditioner for H, then a suitable preconditioner for the Newton equations is

$$P = \begin{pmatrix} (1+\hat{\sigma})M + \frac{2}{\mu} dd^T & d \\ d^T & \mu \end{pmatrix}. \tag{5.14}$$

Note that P can be written as the product of three matrices

$$P = \begin{pmatrix} 1 & \frac{2}{\mu} d \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{2}{\mu} d \\ 0 & 1 \end{pmatrix} \begin{pmatrix} (1+\hat{\sigma})M + \frac{2}{\mu} dd^T & d \\ d^T & \mu \end{pmatrix},$$

which may be simplified as

$$P = \begin{pmatrix} 1 & \frac{2}{\mu}d \\ 0 & 1 \end{pmatrix} \begin{pmatrix} (1+\hat{\sigma})M & -d \\ d^T & \mu \end{pmatrix},$$

This provides a convenient expression for P^{-1} as

$$P^{-1} = \begin{pmatrix} (1+\hat{\sigma})M & -d \\ d^T & \mu \end{pmatrix}^{-1} \begin{pmatrix} 1 & \frac{2}{\mu}d \\ 0 & 1 \end{pmatrix}^{-1}. \tag{5.15}$$

Equation (5.15) can be further simplified by using the following decomposition:

$$\begin{pmatrix} (1+\hat{\sigma})M & -d \\ d^T & \mu \end{pmatrix}^{-1} = WDW^{-T},$$

where

$$W = \begin{pmatrix} 1 & \sigma_t M^{-1}d \\ 0 & 1 \end{pmatrix} \quad D = \begin{pmatrix} \sigma_t M^{-1} & 0 \\ 0 & 1/(\mu + \sigma_t d^T M^{-1} d) \end{pmatrix},$$

and $\sigma_t \equiv 1/(1+\hat{\sigma})$. Using this decomposition, P^{-1} can be written as:

$$P^{-1} = \begin{pmatrix} \sigma_t M^{-1} & (1/(\mu(1+\hat{\sigma}) + d^T s))M^{-1}d \\ 0 & 1/(\mu + \sigma_t d^T s) \end{pmatrix} \begin{pmatrix} I & -\frac{2}{\mu}d \\ -\sigma_t s^T & \frac{2\sigma_t}{\mu}s^T d + 1 \end{pmatrix}, \tag{5.16}$$

Thus, matrix-vector products with P^{-1} may be performed as products with block matrices.

If $H(\sigma)$ is not positive definite then Lanczos-PCG may encounter a direction of negative curvature. If Lanczos-PCG computes a negative diagonal element of D in the LDL^T decomposition of the tridiagonal matrix, the algorithm terminates prematurely and returns the current Lanczos-PCG iterate. In all cases, the Newton update is a descent direction (of positive curvature) for L^{ν}_{μ,σ_e}.

The cost of a Lanczos-PCG solve is one matrix-vector product for each Lanczos vector. Even with the block preconditioner, Lanczos-PCG may converge very slowly or not at all. To guard against this situation, an upper limit is imposed on the number of Lanczos vectors computed for each Lanczos-PCG solve. In the numerical results of Chapter 7, this limit was $n/10$.

Each Lanczos vector is used to generate a new estimate of the leftmost generalized eigenpair. However, the Lanczos vectors have $n+1$ components, i.e., the Newton system is one dimension larger than the original subproblem. Nevertheless, the estimation routine is able to proceed as in the first phase since the Lanczos vectors only serve as direction generators (see Section 5.2.1). In particular, the estimation routine generates new subspaces using only (the first) n components of each Lanczos vector.

The Newton iterates in this section are distinct from the iterates used to approximate the solution to the trust-region subproblem. However, upon entering the second phase, the initial s is taken to be a the current approximation to the trust-region subproblem solution, and the initial value of σ is taken to be the approximation generated by the Moré-Sorensen solve at the end of the first phase. Thus, upon entering the second phase, the Newton iterate and the current approximation to the trust-region subproblem are equivalent.

5.3.4 Economizing matrix-vector products

The solution to the Newton equations in the second phase provides an update $(\Delta\bar{s}_k, \Delta\bar{\sigma}_k)$ to the Newton iterate. The Newton iterate is then given by $(\bar{s}_{k+1}, \bar{\sigma}_{k+1})$ where $\bar{s}_{k+1} = \bar{s}_k + \alpha_k \Delta\bar{s}_k$ and $\sigma_{k+1} = \bar{\sigma}_k + \alpha_k \Delta\bar{\sigma}_k$. After each Newton iteration, the vector $H\bar{s}_{k+1}$ must be updated. This vector is needed to form both the left-hand and right-hand sides of the *next* Newton system associated with minimizing L^{ν}_{μ,σ_e}.

This section presents details associated with forming $H\bar{s}_{k+1}$; however, since $H\bar{s}_{k+1} = H\bar{s}_k + \alpha_k H\Delta\bar{s}_k$, the task of forming $H\bar{s}_{k+1}$ reduces to the task of forming $H\Delta\bar{s}_k$. The method presented below is specifically in the context of the Lanczos-PCG method; however, the details are similar for other CG-type methods.

In Section 3.6, the Lanczos-CG algorithm was extended to solve the system $Ax = b$ and then *indirectly* form $A\Delta x_j$. Recall that forming $A\Delta x_j$ reduced to one matrix-vector product for each Lanczos-CG iteration; namely, the products Av_k were computed for all k. However, forming products of the form $H\Delta\bar{s}_k$ is

complicated by the fact that the update is the augmented vector $(\Delta\bar{s}_k, \Delta\bar{\sigma}_k)$, i.e., $\Delta\bar{s}_k$ is only part of the solution vector. Practically, this means that additional care must be taken because the Lanczos vectors are constructed to solve an augmented system of one dimension higher.

Consider the following version of Lanczos-PCG for the Newton system associated with minimizing L^{ν}_{μ,σ_e}. For simplicity, the Newton equations are denoted by $\bar{H}x = \bar{g}$, and the Lanczos-PCG iterates are denoted by $\{x_i\}$. For simplicity, $\hat{x}_i$ will denote the first n components of x_i. In this notation, the vector $H\Delta\bar{s}_k$ is denoted by $H\hat{x}_l$, where x_l is the final Lanczos-PCG iterate, i.e., $x_l = (\Delta\bar{s}_k, \Delta\bar{\sigma}_k)$. Finally, let P denote the preconditioner for the augmented system. Apart from these differences, the notation in the algorithm below is consistent with the notation in the original Lanczos-CG algorithm.

ALGORITHM 5.3.1. LANCZOS-PCG FOR THE AUGMENTED SYSTEM

Choose $\tau_{tol} > 0$;

$x_0 = 0; \quad [H\hat{x}_0] = 0;$

$s_0 = \bar{g}(x_0); \quad \beta_0 = -\|s_0\|_{P^{-1}}; \quad \gamma_{-1} = 1; \quad \tau = \beta_0; \quad i = -1;$

while $\tau > \tau_{tol}$ **do**

$w_{i+1} = s_{i+1}/\beta_{i+1}; \quad i \leftarrow i+1; \quad$ Solve $Pv_i = w_i; \quad [H\hat{v}_i] = H\hat{v}_i;$

$\gamma_i = v_i^T[\bar{H}v_i];$

if $k = 0$ **then**

$l_i = 0;$

$p_i = v_i; \quad [H\hat{p}_i] = [H\hat{v}_i];$

$s_{i+1} = [\bar{H}v_i] - \gamma_i w_i;$

else

$l_i = \beta_i/d_{i-1};$

$p_i = v_i - l_i p_{i-1}; \quad [H\hat{p}_i] = [H\hat{v}_i] - l_i[H\hat{p}_{i-1}];$

$s_{i+1} = [\bar{H}v_i] - \gamma_i w_i - \beta_i w_{i-1};$

end

$d_i = \gamma_i - \beta_i l_i; \quad \alpha_i = -\beta_i\alpha_{i-1}/d_i;$

$x_{i+1} = x_i + \alpha_i p_i; \quad [H\hat{x}_{i+1}] = [H\hat{x}_i] + \alpha_i [H\hat{p}_i];$

$\beta_{i+1} = -\|s_{i+1}\|_{P^{-1}}; \quad \tau = -\beta_{i+1}\alpha_i;$

end do

There is a subtle detail in this algorithm: only one matrix-vector product is needed each iteration. In particular, given $H\hat{v}_i$, the vector $\bar{H}v_i$ can be formed *implicitly* by utilizing the block structure of $\bar{H}$. An important implication of this is that the algorithm for the leftmost generalized eigenvalue requires no additional matrix-vector products with H. In the augmented Newton solve, the vectors $\{\hat{v}_i\}$ are used as direction generators (Section 5.3.3), and consequently, only $\{H\hat{v}_i\}$ and $\{M\hat{v}_i\}$ are required for the estimation (see Section 5.2.1). (For details on the *implicit* formation of the vectors $\{M\hat{v}_i\}$, see Section 5.3.6.)

5.3.5 Solution of the subproblem on a reduced subspace

The Newton iterate $\bar{s}_{k+1}$ is obtained using a Wolfe line search (see Section 5.3.7). With $\bar{s}_k$, the second phase solves the trust-region subproblem over a reduced space. In particular, each iteration of the second phase solves the reduced problem

$$\underset{s}{\text{minimize}} \;\; \mathcal{Q}(s) = g^T s + \tfrac{1}{2} s^T H s \;\; \text{subject to} \;\; s \in \mathcal{S}_k, \quad \|s\|_M \leq \delta, \tag{5.17}$$

where $\mathcal{S}_k = \text{span}\{s_k, \bar{s}_{k+1}, \tilde{u}_n^{k+1}\}$. The reduced problem is at most three dimensional, and can be solved efficiently using a direct method (e.g., the Moré-Sorensen algorithm).

No additional matrix-vector multiplications are required for the reduced subspace solve provided that the subspace is formed explicitly using a subset of the vectors $\{s_k, \bar{s}_{k+1}, \tilde{u}_n^{k+1}\}$. To see this, note that the reduced subproblem has the form

$$\underset{p}{\text{minimize}} \;\; \mathcal{Q}(p) = (Qg)^T p + \tfrac{1}{2} p^T (Q^T H Q) p, \quad \text{subject to} \;\; \|p\|_C \leq \delta, \tag{5.18}$$

where Q is a matrix whose columns span $\mathcal{S}_k$ and $C = Q^T M Q$. The QR decomposition can be used to fill the columns of Q with a maximally linearly independent

subset of the vectors $\{s_k, \bar{s}_{k+1}, \tilde{u}_n^{k+1}\}$. Since the quantities Hs_k, $H\bar{s}_{k+1}$, and $H\tilde{u}_n^{k+1}$ are already known (see Sections 3.6, 5.2.1, and 5.3.4, respectively), the matrix Q^THQ can be formed explicitly. Similarly, Q^TMQ can be calculated from the vectors Ms_k, $M\bar{s}_{k+1}$, and $M\tilde{u}_n^{k+1}$ (see Sections 3.6, 5.2.1, and 5.3.6, respectively). Once a solution p (say) of the reduced problem is known, it may be expanded into the full space as $s_{k+1} = Qp$, which implies that s_{k+1} is a linear combination of the columns of Q; thus, s_{k+1} is a linear combination of the vectors $\{s_k, \bar{s}_{k+1}, \tilde{u}_n^{k+1}\}$. Observe that Hs_{k+1} may also be formed as a linear combination of Hs_k, $H\bar{s}_{k+1}$, and $H\tilde{u}_n^{k+1}$. Similarly, Ms_{k+1} may be formed from Ms_k, $M\bar{s}_{k+1}$, and $M\tilde{u}_n^{k+1}$.

If the elements of M are available, the vector $\nabla\mathcal{Q}(s_k)$ can be included in the subspace, at the cost of one additional matrix-vector multiply with H, to obtain a subspace of at most four dimensions. (In order to form C, the product $M\nabla\mathcal{Q}(s_k)$ must be computed, which involves the product Mg; having no way to form Mg restricts our ability to include $\nabla\mathcal{Q}(s_k)$ in the reduced subspace.) In this case, the subspace solve is very similar to the subspace solve used by Hager (see equation (4.10), Section 4.4).

5.3.6 Estimating the elliptical norm

The quantity $M\bar{s}_{k+1}$ is needed for the reduced subspace solve and the leftmost generalized eigenvalue approximation routine for (H, M). Moreover, this quantity can be used to estimate the elliptical norm of $\bar{s}_{k+1}$, i.e., $\|\bar{s}_{k+1}\|_M$, which is used to determine whether an iterate satisfies the optimality conditions. Since $M\bar{s}_{k+1} = M\bar{s}_k + \alpha_k M\Delta\bar{s}_k$, the task of estimating $M\bar{s}_{k+1}$ reduces to the task of estimating $M\Delta\bar{s}_k$.

Algorithm 5.3.2. Lanczos-PCG for the augmented system (2)
Choose $\tau_{tol} > 0$;
$x_0 = 0$; $[M\hat{x}_0] = 0$;
$s_0 = \bar{g}(x_0)$; $\beta_0 = -\|s_0\|_{P^{-1}}$; $\gamma_{-1} = 1$; $\tau = \beta_0$; $i = -1$;

while $\tau > \tau_{tol}$ **do**

$w_{i+1} = s_{i+1}/\beta_{i+1}$; Solve $Pv_{i+1} = w_{i+1}$; $[M\hat{v}_{i+1}] = M\hat{v}_{i+1}$; $i = i+1$;

$[\bar{H}v_i] = \bar{H}v_i$; $\gamma_i = v_i^T[\bar{H}v_i]$;

if $k = 0$ **then**

$l_i = 0$;

$p_i = v_i$; $[M\hat{p}_i] = [M\hat{v}_i]$;

$s_{i+1} = [\bar{H}v_i] - \gamma_i w_i$;

else

$l_i = \beta_i/d_{i-1}$;

$p_i = v_i - l_i p_{i-1}$; $[M\hat{p}_i] = [M\hat{v}_i] - l_i[M\hat{p}_{i-1}]$;

$s_{i+1} = [\bar{H}v_i] - \gamma_i w_i - \beta_i w_{i-1}$;

end

$d_i = \gamma_i - \beta_i l_i$; $\alpha_i = -\beta_i\alpha_{i-1}/d_i$;

$x_{i+1} = x_i + \alpha_i p_i$; $[M\hat{x}_{i+1}] = [M\hat{x}_i] + \alpha_i[M\hat{p}_i]$;

$\|\hat{x}_{i+1}\|_M^2 = \hat{x}_{i+1}^T[M\hat{x}_{i+1}]$;

$\beta_{i+1} = -\|s_{i+1}\|_{P^{-1}}$; $\tau = -\beta_{i+1}\alpha_i$;

end do

By assumption, it is not possible to compute $M\hat{v}_{i+1}$ directly. Nevertheless, it may be *implicitly* formed by considering the block preconditioner P. To see this, consider the matrix $E \in \mathbb{R}^{n\times(n+1)}$ such that

$$E = \begin{pmatrix} I & 0 \end{pmatrix},$$

where I is the $n \times n$ identity matrix. Note that the matrix-vector product Ex is an n-vector containing the first n components of the vector x. Using this notation, $M\hat{v}_{i+1} = MEP^{-1}w_{i+1}$.

The block preconditioner for the Newton equations for minimizing L^{ν}_{μ,σ_e} was given by

$$P = \begin{pmatrix} (1+\bar{\sigma})M + \frac{2}{\mu}dd^T & d \\ d^T & \mu \end{pmatrix}.$$

From Section 5.3.3 (Equation (5.16)), P^{-1} can be written as:

$$P^{-1} = \begin{pmatrix} \sigma_t M^{-1} & (\sigma_t/(\mu + \sigma_t d^T s))M^{-1}d \\ 0 & 1/(\mu + \sigma_t d^T s) \end{pmatrix} \begin{pmatrix} I & -\frac{2}{\mu}d \\ -\sigma_t s^T & \frac{2\sigma_t}{\mu}s^T d + 1 \end{pmatrix},$$

where $\sigma_t \equiv 1/(1+\hat{\sigma})$.

Let $w_{i+1} = (w_1, w_2)$ where w_1 is an n-vector and w_2 is a scalar. Then, the first n components of $P^{-1}w_{i+1}$ are given by:

$$EP^{-1}w_{i+1} = \begin{pmatrix} M^{-1} & 0 \end{pmatrix} \begin{pmatrix} \sigma_t I & \sigma_t \sigma_\mu d \\ 0 & \sigma_\mu \end{pmatrix} \begin{pmatrix} w_1 - w_2 \frac{2}{\mu}d \\ -\sigma_t s^T w_1 + w_2(\frac{2\sigma_t}{\mu}s^T d + 1) \end{pmatrix},$$

where $\sigma_\mu = 1/(\mu + \sigma_t d^T s)$. And, finally,

$$MEP^{-1}w_{i+1} = \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} \sigma_t I & \sigma_t \sigma_\mu d \\ 0 & \sigma_\mu \end{pmatrix} \begin{pmatrix} w_1 - w_2 \frac{2}{\mu}d \\ -\sigma_t s^T w_1 + w_2(\frac{2\sigma_t}{\mu}s^T d + 1) \end{pmatrix}. \qquad (5.19)$$

Thus, $M\hat{v}_{i+1} = MEP^{-1}w_{i+1}$ can be computed via products with block matrices.

The quantity $M\hat{v}_{k+1}$ is crucial for continuing the leftmost generalized eigenvalue approximations inside the augmented Lanczos-CG solve. In particular, this quantity is required to form the reduced space for the generalized eigenproblem (see Sections 5.3.4 and 5.2.1).

5.3.7 The phase-two algorithm

The second phase allows for user-specified accuracy; thus, it can obtain accuracy comparable to the Moré-Sorensen algorithm in the constrained case. The cost of this phase depends on the desired accuracy of each subproblem solve: the more accurate, the more expensive. However, the cost associated with entire trust-region algorithm is only the cost in forming Lanczos vectors; there are no additional matrix-vector multiplications in the entire algorithm. The following algorithm gives an overview of the second phase.

ALGORITHM 5.3.3. OUTLINE OF THE SECOND PHASE

Choose $\eta_{tol} > 0$, $\nu > 0$;

$\eta_k = +\infty$;

$\bar{s}_k = s_k$; $[H\bar{s}_k] = [Hs_k]$; $[M\bar{s}_k] = [Ms_k]$;

$\sigma_e = \sigma_k$; $\sigma_L = \max\{-\tilde{\lambda}_n^k, 0\}$

Compute an acceptable μ; (Theorem 5.3.1, using $\lambda_n(H(\sigma)) \approx \tilde{\lambda}_n^k$)

while $\eta_k > \eta_{tol}$ **do**

Compute one step $(\Delta\bar{s}_k, \Delta\bar{\sigma}_k)$ of Newton's method for minimizing L^{ν}_{μ,σ_e};

(The quantities $\tilde{\lambda}_n^{k+1}$ and $\tilde{u}_n^{k+1}$ are updated simultaneously);

Set $\bar{s}_{k+1} = \bar{s}_k + \alpha_k\Delta\bar{s}_k$; $\bar{\sigma}_{k+1} = \bar{\sigma}_k + \alpha_k\Delta\bar{\sigma}_k$;

$[H\bar{s}_{k+1}] = [H\bar{s}_k] + \alpha_k[H\Delta\bar{s}_k]$; $[M\bar{s}_{k+1}] = [M\bar{s}_k] + \alpha_k[M\Delta\bar{s}_k]$;

Update σ_L, if necessary;

Solve the subproblem over a reduced subspace to obtain:

s_{k+1}; σ_{k+1}; $[Hs_{k+1}]$; $[Ms_{k+1}]$;

$\sigma_e = \sigma_{k+1}$;

Safeguard σ_{k+1} and $\bar{\sigma}_{k+1}$;

Update μ; (via Theorem 5.3.1)

$\eta_k = \|(H + \sigma_{k+1}M)p_{k+1} + g\|_{M^{-1}} + \frac{1}{2}|s_{k+1}^T M s_{k+1} - \delta^2|$;

$\sigma_L \leftarrow \max\{\tilde{\lambda}_n^{k+1}, \sigma_L\}$;

$k \leftarrow k + 1$;

end do

At the start of the second phase, $\bar{s}_k$ is initialized at s_k, the current approximation to the trust-region subproblem solution. (Thus, $H\bar{s}_k = Hs_k$ and $M\bar{s}_k = Ms_k$, which are computed in the first phase.) If the first Newton step for minimizing L^{ν}_{μ,σ_e} is denoted by $(\Delta\bar{s}_k, \Delta\bar{\sigma}_k)$, then the next Newton iterate is $(\bar{s}_{k+1}, \bar{\sigma}_{k+1})$, where

$$\bar{s}_{k+1} = \bar{s}_k + \beta_k\Delta\bar{s}_k \quad \text{and} \quad \bar{\sigma}_{k+1} = \bar{\sigma}_k + \beta_k\Delta\bar{\sigma}_k,$$

with β_k denoting the step length obtained via a Wolfe line search. The computation of $H\bar{s}_{k+1} = Hs_k + \beta_k H\Delta\bar{s}_k$ and $M\bar{s}_{k+1} = Ms_k + \beta_k M\Delta\bar{s}_k$ is discussed in

Sections 5.3.4 and 5.3.6, respectively.

The error in the optimality conditions for the subproblem iterate (s_{k+1}, σ_{k+1}) is defined as

$$\|(H + \sigma_{k+1}M)p_{k+1} + g\|_{M^{-1}} + \tfrac{1}{2}|s_{k+1}^T M s_{k+1} - \delta^2|, \tag{5.20}$$

where p_{k+1} approximately solves (2.36). This test takes into account that the Moré-Sorensen solve is only an approximation to the solution for the reduced subproblem.

There are several important issues associated with safeguarding this algorithm. In the hard case, care must be taken so that $\sigma_L \neq \sigma_k$ for any k; otherwise $\nabla L^{\nu}_{\mu,\sigma_e}$ and $\nabla^2 L^{\nu}_{\mu,\sigma_e}$ are undefined. The Lanczos-PCG algorithm terminates immediately it detects that $\nabla^2 L^{\nu}_{\mu,\sigma_e}$ is indefinite. This can occur only if $\sigma_k \notin [-\lambda_n, \infty)$, in which case σ_k is a new lower bound on σ, and σ_L is redefined as σ_k. The lower bound σ_L is also updated if a new, more negative, approximate leftmost generalized eigenvalue is computed during the Lanczos-PCG solve. These two possible updates to σ_L are separated in the code to emphasize that they are unrelated.

Safeguarding σ_{k+1} requires additional care. There are several situations in which the Newton system may be indefinite in the following iteration. First, the solution of the reduced problem may terminate with a point (s_{k+1}, σ_{k+1}) that is a less accurate solution of the subproblem than (s_k, σ_k), i.e., $\eta_{k+1} > \eta_k$. Second, the solution of the reduced subproblem may generate a value of σ_{k+1} such that $\sigma_{k+1} < \sigma_L$. In both these cases, the iterate from the reduced problem is rejected and (s_{k+1}, σ_{k+1}) is set to the previous value (s_k, σ_k). This forces a monotonic decrease in the error of the optimality conditions and guards against using a poor value of σ_e in the subsequent iteration.

Another possibility is that σ_e and $\bar{\sigma}_{k+1}$ are in the interval $[-\lambda_n, \infty)$, but the subsequent Newton system can be indefinite (for details, see Section 5.3.2). Assuming $\bar{\sigma}_{k+1} \in [-\lambda_n, \infty)$, the Newton system will be positive definite if $\pi - \bar{\sigma} > 0$, i.e., $\sigma_e + r(\bar{s}_{k+1})/\mu - \bar{\sigma}_{k+1} > 0$. However, it is acceptable for this inequality to not

hold as long as

$$\bar{\sigma}_{k+1} + 2(\sigma_e + r(s_{k+1})/\mu - \bar{\sigma}_{k+1}) > \sigma_L. \tag{5.21}$$

If this inequality does not hold, then the next Newton system is guaranteed to be indefinite; in this case, it is desirable to only make a change to $\bar{\sigma}_{k+1}$ in dire cases so as not to disrupt the convergence of the Newton iterates. The following algorithm details how this situation is handled; moreover, it guarantees that if $\bar{\sigma} \in [-\lambda_n, \infty)$ then the subsequent Newton system will be positive definite.

The following algorithm details the safeguarding procedure. (In this algorithm, ϵ_M denotes machine precision.)

Algorithm 5.3.4. Safeguarding for the Newton System

if $\eta_{k+1} > \eta_k$ **or** $\sigma_{k+1} < \sigma_L$

$s_{k+1} = s_k$; $\sigma_{k+1} = \sigma_k$; $\sigma_e = \sigma_k$; $[Hs_{k+1}] = [Hs_k]$; $[Ms_{k+1}] = [Ms_k]$;

end

if $\bar{\sigma}_{k+1} < \sigma_L < \sigma_{k+1}$ **then**

$\bar{\sigma}_{k+1} = \sigma_{k+1}$; $\bar{s}_{k+1} = s_{k+1}$; $[H\bar{s}_{k+1}] = [Hs_{k+1}]$; $[M\bar{s}_{k+1}] = [Ms_{k+1}]$;

end

if $\bar{\sigma}_{k+1} + 2(\sigma_e + r(\bar{s}_{k+1})/\mu - \bar{\sigma}_{k+1}) \leq \sigma_L$ **then**

if $\sigma_e < \sigma_L < \bar{\sigma}_{k+1}$ **then**

if $r(\bar{s}_{k+1}) > 0$ **then** $\sigma_e = \bar{\sigma}_{k+1}$; **else** $\sigma_e = \bar{\sigma}_{k+1} + |r(\bar{s}_{k+1})|/\mu$;

else if $\sigma_e > \sigma_L$ **and** $\sigma_{k+1} > \sigma_L$ **then**

if $\eta_{k+1} < 10^{-1}(\|(H + \bar{\sigma}_{k+1}M)\bar{s}_{k+1} + g\|_{M^{-1}} + \frac{1}{2}|\bar{s}_{k+1}^T M \bar{s}_{k+1} - \delta^2|)$ **then**

$\bar{\sigma}_{k+1} = \sigma_{k+1}$; $\bar{s}_{k+1} = s_{k+1}$;

$[H\bar{s}_{k+1}] = [Hs_{k+1}]$; $[M\bar{s}_{k+1}] = [Ms_{k+1}]$;

end

else if $\bar{\sigma}_{k+1} < \sigma_L$ **and** $\sigma_e < \sigma_L$

$\bar{\sigma}_{k+1} = \sigma_L + \sqrt{\epsilon_M}$;

if $\eta_{k+1} < (\|(H + \bar{\sigma}_{k+1}M)\bar{s}_{k+1} + g\|_{M^{-1}} + \frac{1}{2}|\bar{s}_{k+1}^T M \bar{s}_{k+1} - \delta^2|)$ **then**

$\bar{\sigma}_{k+1} = \sigma_{k+1}$; $\bar{s}_{k+1} = s_{k+1}$;

$[H\bar{s}_{k+1}] = [Hs_{k+1}]$; $[M\bar{s}_{k+1}] = [Ms_{k+1}]$;

end

end

if $\bar{\sigma}_{k+1} + 2(\sigma_e + r(\bar{s}_{k+1})/\mu - \bar{\sigma}_{k+1}) \leq \sigma_L$ **then** $\sigma_e = \bar{\sigma}_{k+1} + |r(\bar{s}_{k+1})|/\mu$;

end

Notice that the Newton iterates can only modified in the case when the following Newton system is guaranteed to be indefinite. Even then, the Newton iterates are only modified if either $\bar{\sigma}_{k+1} < \sigma_L$ or the error associated with the Newton iterates is significantly greater than the error associated with the subproblem iterate (s_{k+1}, σ_{k+1}). Subsequent Newton systems are most likely to be positive definite because of the modifications to σ_e. As the second phase converges, $\sigma_e \approx \bar{\sigma}_{k+1}$ and $r(\bar{s}_{k+1})/\mu$ should be very small; thus, modifications to σ_e will also be small.

6

Preconditioning

Preconditioning may not only accelerate the convergence of a CG-type method, but it may also prevent a form of *stalling* in which many CG iterates are computed with negligible progress towards a solution. Using preconditioned Lanczos-CG to solve the (linear) Newton equations in both phases of the proposed trust-region algorithm may increase efficiency by reducing the number of computed Lanczos vectors, and thus, requiring fewer matrix-vector multiplications.

This chapter presents two preconditioning methods. The first method uses incomplete Cholesky factorizations to generate preconditioners for $\nabla^2 f$. The second method is based on limited-memory quasi-Newton updating to improve the block preconditioner P for the augmented Newton system in the second phase. These methods are not mutually exclusive; they may be used in tandem to generate preconditioners for both phases.

It is not always desirable to use the preconditioning methods described in this chapter. In particular, when a good preconditioner is already available based on the problem formulation (e.g., a multigrid preconditioner for PDE constrained optimization), it may be unhelpful or counterproductive to try to improve upon the preconditioner. In this case, it is not recommended to use the additional preconditioning techniques of this chapter.

6.1 The Incomplete Cholesky Factorization

The preconditioner M should be a matrix that approximates H (i.e., $M^{-1}H \approx I$) such that it is possible to form matrix-vector products with M^{-1} by implicit or indirect means (i.e., without requiring explicit multiplies with M^{-1}).

Any symmetric positive-definite matrix H may be written uniquely as the product $H = LDL^T$ where D is a diagonal matrix with positive entries, and L is unit-lower triangular. The Cholesky factorization of H is then $H = R^T R$, where $R = LD^{1/2}$. The Cholesky factor R is also unique.

The *incomplete* Cholesky factorization is one of the most widely used methods to obtain a sparse, positive-definite approximation to H. If H is positive definite, incomplete Cholesky factors may be regarded as *sparse* approximations to the true Cholesky factors. In the case where H is not positive definite, it is possible to compute a *modified* incomplete Cholesky factorization. If the (modified) incomplete Cholesky factorization is taken to be M, then matrix-vector products with M^{-1} may be computed implicitly using triangular solves with R and R^T. For this reason, incomplete Cholesky factorizations make preconditioning practical for large problems.

6.1.1 The modified Cholesky factorization

If H is not positive definite, modified Cholesky algorithms define an upper-triangular matrix R such that $H + E = R^T R$ where E is positive semidefinite. If H is well-conditioned, then R should be well-conditioned. Furthermore, E should be as small as possible.

The first numerically stable modified Cholesky factorization was due to Gill and Murray [13], and was later refined to include symmetric interchanges in [14].

Given positive scalars β and δ, the modified Cholesky factorization of Gill and Murray [13] computes an upper-triangular matrix R and a *diagonal* E such that

$$H + E = R^T R, \tag{6.1}$$

where the elements of R satisfy

$$|r_{ki}| \leq \beta \quad \text{and} \quad |r_{ii}| > \delta, \quad \text{for} \quad i = 1\!:\!n, \;\; k > i. \tag{6.2}$$

The idea is to implicitly increase the diagonal elements of H until (6.2) holds. The value of e_{ii}, the ith diagonal of E, is then the amount that h_{ii} must be increased. Suppose that the smallest possible e_{11} that makes $h_{11} + e_{11}$ sufficiently positive leads to a large off-diagonal element in the first row of R, i.e., an element whose magnitude exceeds β. Let m denote the index of the off-diagonal element in the first row of H that is largest in magnitude; we then increase e_{11} further, until

$$\frac{|h_{1m}|}{\sqrt{h_{11} + e_{11}}} = \beta.$$

This fixes r_{11} to be $|h_{1m}|/\beta$, and the factorization proceeds to the next stage. Note that

$$|r_{1j}| = \frac{|h_{1j}|}{\sqrt{h_{11} + e_{11}}} \leq \frac{|h_{1m}|}{\sqrt{h_{11} + e_{11}}} = \beta, \qquad j = 1, 2, \ldots, n.$$

Hence, the elements of the first row of R are bounded in magnitude by β. This procedure is summarized by the following algorithm.

Algorithm 6.1.1. Modifed Cholesky Factorization

Let β ($\beta > 0$), δ ($\delta > 0$) be given;
$\widehat{H} = H$;
for $k = 1\!:\!n$ **do**
    $\nu_k = \max\{\, |\hat{h}_{kj}| : j = k+1\!:\!n \,\}$;
    $r_{kk} = \max\{\, \delta, \sqrt{|\hat{h}_{kk}|}, \nu_k/\beta \,\}$;
    $e_{kk} = r_{kk}^2 - \hat{h}_{kk}$;
    for $j = k+1\!:\!n$ **do**
        $r_{kj} = \hat{h}_{kj}/r_{kk}$;
        for $i = k+1\!:\!j$ **do**
            $\hat{h}_{ij} \leftarrow \hat{h}_{ij} - r_{kj}\hat{h}_{ki}$;
        end do;

end do;
end do.

The bound β should be large enough that H is not modified if it is sufficiently positive definite. The following lemma shows that if

$$\beta^2 > \gamma \equiv \max\{\, |h_{ii}| : i = 1:n\}, \tag{6.3}$$

then the modification $E = 0$ in (6.1) if H is "sufficiently" positive definite.

Lemma 6.1.1 *Let H be a symmetric matrix and assume that the modified Cholesky factorization of H has been obtained using* Algorithm 6.1.1 *with $\beta^2 > \gamma$. If q is the smallest integer that satisfies*

$$\hat{h}_{qq} \le \hat{h}_{jj}, \quad j = 1:n,$$

and d satisfies $Rd = r_{qq}e_q$, then $d^THd \le \hat{h}_{qq}$. Moreover, if $\delta = 0$ and H has at least one negative eigenvalue, then $\hat{h}_{qq} < 0$.

Proof. At the kth stage of the factorization, the elements $(\hat{h}_{kk}, \ldots, \hat{h}_{k,k+1}, \hat{h}_{kn})$ of the first row and column of the $k \times k$ unfactorized block can be written as

$$\hat{h}_{kj} = h_{kj} - \sum_{i=1}^{k-1} r_{ij}r_{ik}, \quad k \le j \le n, \tag{6.4}$$

$$\text{and} \qquad \hat{h}_{kk} = h_{kk} - \sum_{i=1}^{k-1} r_{ik}^2. \tag{6.5}$$

If $Rd = r_{qq}e_q$, then $d_q = 1$ and $d_i = 0$ for $i > q$. Moreover, since $H + E = R^TR$,

$$d^THd = d^T(R^TR - E)d = r_{qq}^2 - e_{qq} - \sum_{j=1}^{q-1} d_j^2 e_{jj}^2 = \hat{h}_{qq} - \sum_{j=1}^{q-1} d_j^2 e_{jj}^2,$$

and consequently, $d^THd \le \hat{h}_{qq}$.

To show the second result, assume that $\delta = 0$. If $\hat{h}_{qq} \ge 0$, (6.5) and the definition of q imply that

$$\beta^2 > h_{kk} \ge \sum_{i=1}^{k-1} r_{ik}^2, \qquad \text{for} \qquad 1 \le k \le n.$$

This inequality shows that $|r_{ik}| < \beta$ for $i < k$, so that e_{kk} does not need to be chosen to enforce a bound on the off-diagonal elements of R. This implies that if $r_{kk} \neq 0$, then $r_{kk} > \nu_k/\beta$, giving $r_{kk}^2 = |\hat{h}_{kk}|$ and $e_{kk} = 0$. If $r_{kk} = 0$, clearly $e_{kk} = 0$. It follows that $E = 0$ and $H = R^T R$. Thus we have shown that if $\hat{h}_{qq} \geq 0$, H must be positive semi-definite. ∎

In addition to satisfying the lower bound $\beta^2 > \gamma$, β should not be "too large", to prevent excessively large elements in R. For a given matrix H, the diagonal modification E depends on β. The following theorem derives a *bound* on $\|E\|_\infty$ as a function of β.

Theorem 6.1.1 (Gill and Murray [13]) *For a given $\beta > 0$,*

$$\|E(\beta)\|_\infty \leq \Big(\frac{\xi}{\beta} + (n-1)\beta\Big)^2 + 2\big(\gamma + (n-1)\beta^2\big) + \delta, \tag{6.6}$$

where ξ is the largest in modulus of the off-diagonal elements of H, and γ is defined by (6.3). ∎

If $n > 1$, the bound of Lemma 6.1.1 is minimized when $\beta^2 = \xi/\sqrt{n^2-1}$. Thus, a practical choice of β is

$$\beta^2 = \max\{\gamma, \xi/\sqrt{n^2-1}, \epsilon_M\}, \tag{6.7}$$

where ϵ_M is the relative machine precision, and is included to allow for the case where $\|H\|_2$ is small. Note that (6.7) always satisfies the lower bound (6.3).

The modified Cholesky factorization is numerically stable, and produces a positive-definite matrix differing from the original matrix only in its diagonal elements. The diagonal modification E with β defined by (6.7) is "optimal", in the sense that the *a priori* bound (6.6) on $\|E\|_\infty$ is minimized, subject to satisfying (6.3). In practice, the actual value of $\|E\|_\infty$ is almost always substantially less than the *a priori* bound.

An alternative modified Cholesky procedure was suggested by Schnabel and Eskow [46], which often gives a smaller bound on E. The algorithm uses the

Gerschgorin circle theorem to bound the elements of E and to prove the following theorem:

Theorem 6.1.2 (Schnabel and Eskow [46]) *Suppose that at each step of the modified Cholesky factorization* (Algorithm 6.1.1)*, the remaining matrix* $H_j \in \mathbb{R}^{(n+1-j)\times(n+1-j)}$ *is given by*

$$H_j = \begin{pmatrix} \alpha_j & h_j^T \\ h_j & \widehat{H}_j \end{pmatrix},$$

and $H_1 \triangleq H$*. Furthermore, suppose that* H_{j+1} *is defined as* $H_{j+1} = \widehat{H}_j - h_j h_j^T/(\alpha_j + \delta_j)$*, where*

$$\delta_j = \max\{0, \|h_j\|_1 - \alpha_j\}.$$

If $E \equiv \mathrm{diag}(\delta_1, \ldots, \delta_n)$*, then* $H + E$ *is positive semidefinite and*

$$\|E\|_\infty \leq \gamma + (n-1)\xi. \ \blacksquare$$

6.1.2 The incomplete Cholesky factorization

There are two general strategies used to define an incomplete Cholesky factorization

$$H = R^T R + E,$$

where R is a sparse, upper-triangular matrix. The first strategy is to use a drop tolerance to limit the number of nonzero elements in the upper-triangular Cholesky factor R. In this approach, an incomplete Cholesky factorization is computed and elements in R are set to zero if they are smaller in magnitude than the drop tolerance. A disadvantage to this approach is that the number of nonzero elements in R can be unpredictable; there is often no way to foretell the "magical" drop tolerance needed to meet a specific memory requirement. Another approach of many incomplete Cholesky methods is to use a fixed fill-in strategy. In this approach, the memory requirements are fixed; however, the sparsity pattern in R is explicitly tied to the *structure* of H rather than the entries.

There are several incomplete Cholesky methods that attempt to combine these two strategies. In [44] and [45], Saad proposes storing a maximum number of nonzero elements in each column of R^T and also implementing a drop tolerance. This approach uses an incomplete LU factorization of H; however, the method does not attempt to preserve the symmetry of H.

Jones and Plassmann [23] suggest an incomplete Cholesky factorization that sets the sparsity pattern of R dependent on the *entries* of H. This method limits the number of nonzero entries in each column of R^T to the number of nonzero entries in the strict lower-triangular portion of the original matrix H; moreover, the method determines which entries are maintained based on the magnitude of the entries rather than the sparsity structure of H. This approach enjoys the advantages of a drop tolerance-based factorization without actually fixing a drop tolerance.

In this same spirit, Lin and Moré [25] propose an incomplete Cholesky factorization that controls memory requirements and can be extended to indefinite matrices. Given a scalar $p \geq 0$, Lin and Moré count the number of nonzero entries in the jth column of the strict lower-triangular portion of H, denoted n_j, and compute the jth column of the incomplete Cholesky factor R^T but retain only the largest (in magnitude) $n_j + p$ entries. (Notice that this is equivalent to Jones and Plassmann's method with $p = 0$.) The following incomplete Cholesky algorithm details this process [25]. The algorithm overwrites the jth column of $\widehat{H}$ with the jth column of R^T; thus, the matrix R can be recovered by taking the transpose of the lower-triangular portion of $\widehat{H}$.

Algorithm 6.1.2. Incomplete Cholesky Factorization

Choose $p > 0$;
$\widehat{H} = H$;
for $j = 1 : n$ **do**
    $\hat{h}_{jj} = \sqrt{\hat{h}_{jj}}$;
    $n_j =$ number of nonzero entries $\hat{h}_{ij}$ such that $i > j$;

for $k = 1 : n$ **and** $\hat{h}_{jk} \neq 0$ **do**
  for $i = j + 1 : n$ **and** $\hat{h}_{ik} \neq 0$ **do**
    $\hat{h}_{ij} = \hat{h}_{ij} - \hat{h}_{ik}\hat{h}_{jk}$;
  end do
end do
for $i = j + 1 : n$ **and** $\hat{h}_{ik} \neq 0$ **do**
  $\hat{h}_{ij} \leftarrow \hat{h}_{ij}/\hat{h}_{jj}$;
  $\hat{h}_{ii} \leftarrow \hat{h}_{ii} - \hat{h}_{ij}^2$;
end do
Retain the largest $n_j + p$ elements of $\{\hat{h}_{j+1,j}, \ldots, \hat{h}_{nj}\}$;
end do

Incomplete Cholesky factorizations may fail on general positive-definite matrices; this occurs when a computed diagonal of element of R is not strictly positive (for an example, see [24]). One approach to extend the incomplete Cholesky factorization to general positive-definite matrices is to use a scaling and shifting technique. The following algorithm is similar to Jones and Plassman's method with $p = 0$ on positive-definite matrices; however, Lin and Moré have modified the scaling to extend the algorithm to general indefinite matrices.

Algorithm 6.1.3. ICFM for general matrices
Choose $\alpha_s > 0$; $p \geq 0$;
$D = \text{diag}(\|He_i\|_2)$; $\widehat{H} = D^{-1/2}HD^{-1/2}$;
if $\min(\widehat{H}_{ii}) > 0$ **then** $\alpha_0 = 0$ **else** $\alpha_0 = -\min(\widehat{H}_{ii}) + \alpha_s$;
for $k = 1 : \ldots$ **do**
  Apply Algorithm 6.1.2 to $\widehat{H}_k = \widehat{H} + \alpha_k I$;
  if successful **then**
    $\alpha_F = \alpha_k$; **break**;
  end
  $\alpha_{k+1} = \max(2\alpha_k, \alpha_s)$;
end do

Small values of α_F provide better preconditioners for H. Lin and Moré prove that this algorithm successfully computes an incomplete Cholesky factorization of H for values of α such that $\alpha > \beta^{-1/2}$, where β is the maximum number of nonzero entries in any column of H; thus, $\alpha_F \leq 2\beta^{-1/2}$. Lin and Moré find that increasing the additional memory allowance p usually decreases the value of α_F. This is consistent with the idea that allowing for more fill-in should increase the quality of the preconditioner.

6.2 Limited-Memory Quasi-Newton Updates

The fundamental idea of quasi-Newton updates is to use second-order information, in the form of finite-differences, to update an approximation to the Hessian. In the context of preconditioning, second-order information can be used to update a (possibly simple) initial preconditioner in order to obtain a better approximation of H. This strategy will be used to help construct a better preconditioner P for the augmented Newton system in the second phase.

Quasi-Newton preconditioning methods can be adapted to help precondition in the case when it is prohibitively expensive to store updated (dense) matrices. In this case, a limited number of pairs of vectors $\{(\eta_l, \gamma_l)\}$ can be saved and then *implicitly* used to form matrix-vector multiplies with the new, updated preconditioner. These types of methods are called *limited-memory* quasi-Newton preconditioning methods.

Quasi-Newton updates can be performed to either a preconditioner or its inverse. In the context of the proposed trust-region solver, it is numerically more stable to apply the quasi-Newton updates to P^{-1} (i.e., recall that PCG requires multiplies with P^{-1}). Thus, this section will focus on quasi-Newton updates to the inverse of the block preconditioner P.

6.2.1 Limited-memory BFGS updates

Quasi-Newton BFGS updates can be used to solve linear systems, and more generally, unconstrained optimization problems. In particular, the BFGS method with an exact line search to solve a positive-definite system $Ax = b$ is equivalent to applying CG to the linear system $Ax = b$ to solve the unconstrained quadratic minimization problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \;\; q(x) = \tfrac{1}{2} x^T A x - b^T x. \tag{6.8}$$

The BFGS method can also be used to generate a sequence of updates to P_0^{-1} to precondition a linear system $Ax = b$ by generating a sequence of iterates $\{x_l\}$ and then applying the formula

$$P_{l+1}^{-1} = G_l^T P_l^{-1} G_l + \rho_l \eta_l \eta_l^T, \tag{6.9}$$

where

$$\eta_l = x_{l+1} - x_l, \qquad \gamma_l = A(x_{l+1} - x_l),$$
$$\rho_l = 1/\gamma_l^T \eta_l, \qquad G_l = I - \rho_l \gamma_l \eta_l^T.$$

Provided the initial P_0^{-1} is positive definite and $\rho_l > 0$ for all l, this formula will produce a sequence of positive-definite matrices $\{P_l^{-1}\}$. (For more details, see Dennis and Schnabel [7] or Gill, Murray and Wright [14]).

The limited-memory BFGS method is designed to be an (efficient) alternative to the BFGS method for large problems. Instead of performing explicit updates to P^{-1}, the limited-memory BFGS algorithm *implicitly* updates P^{-1} by storing a small sequence of pairs $\{(\eta_l, \gamma_l)\}$. Specifically, limited-memory BFGS stores at most m pairs $\{(\eta_l, \gamma_l)\}$ at one time, and always maintains the most recently computed pairs. Typical values for m range between 2 and 20 (e.g., [30], [29], [26], or [34]).

The following algorithm outlines the limited-memory BFGS algorithm to update P^{-1} in the context of building a preconditioner for the (linear) Newton system associated with minimizing a general function f.

ALGORITHM 6.2.1. THE LIMITED-MEMORY BFGS TO UPDATE P^{-1}

Define constants m, ω $(0 < \eta_1 < \omega < 1)$, η_1 $(\eta_1 < 1/2)$;
Choose x_0 and symmetric positive-definite matrix P_0^{-1};
$k \leftarrow 0$; $\widehat{m} \leftarrow 0$;
while $\nabla f(x_k) \neq 0$ **do**
  $s_k = -P_k^{-1}\nabla f(x_k)$;
  $x_{k+1} \leftarrow x_k + \alpha_k s_k$, where α_k satisfies the Wolfe conditions:
    $f(x_k + \alpha_k s_k) \leq f(x_k) + \eta_1 \alpha_k \nabla f(x_k)^T s_k$,
    $\nabla f(x_k + \alpha_k s_k)^T s_k \geq \omega \nabla f(x_k)^T s_k$.
  Update P_k^{-1} using the pairs $\{(\eta_j, \gamma_j)\}_{j=k-\widehat{m}}^{k}$:
    for $j = k - \widehat{m}, \ldots, k$ **do**
      $P_{j+1}^{-1} = G_j^T P_j^{-1} G_j + \rho_j \eta_j \eta_j^T$
    end
  $k \leftarrow k + 1$;
  $\widehat{m} \leftarrow \min\{k, m-1\}$
end do

As written, this algorithm is misleading because the sequence $\{P_l^{-1}\}$ is not explicitly computed. Even if P_0^{-1} is sparse, matrices in the sequence $\{P_l^{-1}\}$ may be dense. Thus, updates to P_l^{-1} are never formed; instead, the following recursive formula using only the pairs $\{(\eta_l, \gamma_l)\}$ is used to compute products with P_l^{-1} ([34]):

ALGORITHM 6.2.2. RECURSIVE FORMULA TO COMPUTE $P^{-1}y = r$

Let m denote the number of stored pairs $\{(\eta_l, \gamma_l)\}$;
$q \leftarrow y$;
for $i = m - 1 : 0$
  $\alpha = \rho_j \eta_j^T q$;
  $q = q - \alpha \gamma_i$;
end
$r = P_0^{-1} q$;
for $i = 0 : m - 1$

$\beta = \rho\gamma_i^T r;$

$r = r + \eta_i(\alpha - \beta);$

end

Notice that the additional storage requirement for forming products with P^{-1} is negligible.

There are two additional subtleties in the limited-memory BFGS algorithm. First, the matrix P_0^{-1} must be positive definite; moreover, it should be sparse. A common choice initialization is the scaled identity matrix

$$P_0^{-1} = \frac{\eta_{\widehat{m}}^T \gamma_{\widehat{m}}}{\gamma_{\widehat{m}}^T \gamma_{\widehat{m}}} I,$$

where $(\eta_{\widehat{m}}, \gamma_{\widehat{m}})$ denotes the last computed pair. Second, the type of line search is not arbitrary—the Wolfe conditions are used to guarantee $\rho_l > 0$, and thus, ensures that the algorithm will generate a sequence of positive-definite matrices $\{P_l^{-1}\}$ (e.g., see [29]).

6.2.2 Implementing quasi-Newton updates in Phase Two

It is relatively simple to incorporate Algorithm 6.2.1 into the second phase to update the inverse of the block preconditioner P. By construction, the Lanczos-CG algorithm generates iterates $\{s_i\}$ to solve Newton system associated with minimizing L^{ν}_{μ,σ_e}. During each Lanczos-CG iteration the pairs (η_l, γ_l) may be defined as

$$\eta_i = s_{i+1} - s_i, \quad \gamma_i = \nabla^2_{ss} L^{\nu}_{\mu,\sigma_e}(s,\sigma)(s_{i+1} - s_i).$$

Recall, however, that only m pairs may be stored. Thus, as new Lanczos-CG iterates are computed only the last m iterates are maintained each iteration. Notice that these iterates are the same iterates generated by CG to minimize the quadratic

$$q(p) = L^{\nu}_{\mu,\sigma_e}(s,\sigma) + p^T \nabla_s L^{\nu}_{\mu,\sigma_e}(s,\sigma) + \tfrac{1}{2} p^T \nabla^2_{ss} L^{\nu}_{\mu,\sigma_e}(s,\sigma) p, \tag{6.10}$$

with the exception that only the last m iterates are stored for the limited-memory BFGS implementation. In other words, the Lanczos-CG iterates used to form finite differences to update the preconditioner are the same iterates that BFGS would use to form finite differences in order to minimize the quadratic approximation to $L^{\nu}_{\mu,\sigma_e}(s,\sigma)$. The implication is that, by construction, $\rho_l > 0$ for all l. Thus, the Wolfe line search used to ensure $\rho_l > 0$ for all l is not necessary in this application.

During each Lanczos-CG iteration information is gathered to form the pairs $\{(\eta_l, \gamma_l)\}$. This information is used to update the inverse of the block preconditioner for the *subsequent* Lanczos-CG solve. As the second phase converges, the hope is that $\nabla^2 L^{\nu}_{\mu,\sigma_e}(\bar{s}_k, \bar{\sigma}_k) \approx \nabla^2 L^{\nu}_{\mu,\sigma_e}(\bar{s}_{k+1}, \bar{\sigma}_{k+1})$; in this case, the limited-memory updates built into P^{-1} should improve the quality of the preconditioner. The following algorithm outlines this process:

ALGORITHM 6.2.3. THE SECOND PHASE WITH UPDATES

while *not* converged **do**

Newton-SQP solve to obtain $(\Delta\bar{s}_k, \Delta\bar{\sigma}_k)$:

while *not* converged **do**

one iteration of PCG with preconditioner P;

$(s_{i+1}, \sigma_{i+1}) \leftarrow (s_i, \sigma_i) + \beta_i p_i$; (PCG update)

Update the pairs (η_i, γ_i);

end do

$\bar{s}_{k+1} = \bar{s}_k + \alpha_k \Delta\bar{s}_k$;

$\bar{\sigma}_{k+1} = \bar{\sigma}_k + \alpha_k \Delta\bar{\sigma}_k$;

Solve the reduced trust-region subproblem;

$k \leftarrow k + 1$;

end do

As noted before, the inverse of the preconditioner is never explicity updated; instead, the pairs $\{\eta_l, \gamma_l\}$ are used to form multiplies with P^{-1}, as needed.

6.2.3 Estimating the elliptical norm

Recall that the quantity $M\bar{s}_{k+1}$ is required to form the reduced subspace used in the three-dimensional trust-region subproblem solve and to form the Newton equations associated with minimizing L^{ν}_{μ,σ_e} in the subsequent iteration. Moreover, this quantity can be used to estimate $\|\bar{s}_{k+1}\|_M$ to determine whether the Newton iterate satisfies optimality conditions. As in Section 5.3.6, the task of estimating $M\bar{s}_{k+1}$ reduces to the task of estimating $M\Delta\bar{s}_k$. However, the limited-memory quasi-Newton updating scheme introduces additional complications in estimating the quantity $M\Delta\bar{s}_k$. In this section there will be many references to Section 5.3.6, due to their similarity. To facilitate understanding, the notational devices used in Section 5.3.6 will be continued here without pause to restate definitions.

As in Section 5.3.6, Algorithm 5.3.2 will be used to update the quantity $M\bar{s}_{k+1}$; however, the expression for $M\hat{v}_{i+1}$ given by (5.19) is no longer valid because of the quasi-Newton updates. Fortunately, by storing an *additional* set of vectors there is an simple way to calculate $M\hat{v}_{i+1} \equiv MEP^{-1}w_{i+1}$ by modifying the recursion algorithm used to calculate matrix-vector products with P^{-1} (Algorithm 6.2.2). To see this, we begin by considering the first iteration in the second phase.

During the first iteration, P is the block preconditioner given by

$$P = \begin{pmatrix} (1+\tilde{\sigma})M + \frac{2}{\mu}dd^T & d \\ d^T & \mu \end{pmatrix}.$$

The formula given by (5.19) can be used to obtain the quantities $M\hat{\eta}_l = ME\eta_l$ for all l. For this quasi-Newton preconditioning scheme the pairs $\{(\eta_l, \gamma_l)\}$ and the vectors $\{M\hat{\eta}_l\}$ must be stored. With this, the following algorithm can be used to calculate $r \triangleq P^{-1}w_{i+1} = v_{i+1}$ as well as $Mr \triangleq M\hat{v}_{i+1} = MEP^{-1}w_{i+1}$.

ALGORITHM 6.2.4. RECURSIVE FORMULA TO COMPUTE $P^{-1}y$ AND $MEP_l^{-1}y$
Let m denote the number of stored pairs $\{(\eta_l, \gamma_l)\}$;
$q \leftarrow y$;
for $i = m-1:0$ **do**

$\alpha = \rho_j \eta_j^T q;$

$q = q - \alpha \gamma_i;$

end

$r = P_0^{-1} q;$

Compute Mr using (5.19);

for $i = 0 : m-1$ **do**

$\beta = \rho \gamma_i^T r;$

$r = r + \eta_i(\alpha - \beta);$

$[Mr] = [Mr] + [M\hat{\eta}_i](\alpha - \beta);$

end

Hence, Algorithm (5.3.2), together with this formula to obtain $M\hat{v}_{i+1}$, can be used to update $M\bar{s}_{k+1}$.

Being able to form $M\hat{v}_{i+1}$ also means that it is possible to continue the leftmost generalized eigenpair estimation routine inside the Lanczos-CG solve beyond the first iteration of the second phase (see Sections 5.3.4 and 5.2.1).

The following algorithm details the Lanczos-CG method to solve the augmented Newton systems with limited-memory quasi-Newton updating. The algorithm combines the two previous algorithms (Algorithm 5.3.1 and 5.3.2) used for the preconditioned Lanczos-CG algorithm for the augmented Newton solve. Moreover, it has features to accommodate the limited-memory quasi-Newton updating scheme. For further details on the notation, see Algorithm 5.3.1 (in Section 5.3.4) and Algorithm 5.3.2 (in Section 5.3.6).

Algorithm 6.2.5. Lanczos-PCG with QN preconditioning

Choose $\tau_{tol} > 0;$

$x_0 \leftarrow 0; \quad [M\hat{x}_0] = 0;$

$s_0 = \bar{g}(x_0); \ \beta_0 = -\|s_0\|_{P^{-1}}; \quad \gamma_{-1} = 1; \quad \tau = \beta_0; \ i = -1;$

while $\tau > \tau_{tol}$ **do**

$w_{i+1} = s_{i+1}/\beta_{i+1}; \quad$ Solve $Pv_{i+1} = w_{i+1};$

$[H\hat{v}_{i+1}] = H\hat{v}_{i+1}; \quad [M\hat{v}_{i+1}] = M\hat{v}_{i+1}; \quad i = i + 1;$

$\gamma_i = v_i^T[\bar{H}v_i]$;

if $k = 0$ **then**

$l_i = 0$;

$p_i = v_i$; $[\bar{H}p_i] = [\bar{H}v_i]$; $[H\hat{p}_i] = [H\hat{v}_i]$; $[M\hat{p}_i] = [M\hat{v}_i]$;

$s_{i+1} = [\bar{H}v_i] - \gamma_i w_i$;

else

$l_i = \beta_i / d_{i-1}$;

$p_i = v_i - l_i p_{i-1}$; $[\bar{H}p_i] = [\bar{H}v_i] - l_i[\bar{H}p_{i-1}]$;

$[H\hat{p}_i] = [H\hat{v}_i] - l_i[H\hat{p}_{i-1}]$; $[M\hat{p}_i] = [M\hat{v}_i] - l_i[M\hat{p}_{i-1}]$;

$s_{i+1} = [\bar{H}v_i] - \gamma_i w_i - \beta_i w_{i-1}$;

end

$d_i = \gamma_i - \beta_i l_i$; $\alpha_i = -\beta_i \alpha_{i-1}/d_i$;

$x_{i+1} = x_i + \alpha_i p_i$;

$[H\hat{x}_{i+1}] = [H\hat{x}_i] + \alpha_i[M\hat{p}_i]$; $[M\hat{x}_{i+1}] = [M\hat{x}_i] + \alpha_i[M\hat{p}_i]$;

$\eta_{i+1} = \alpha_i p_i$; $\gamma_{i+1} = \alpha[\bar{H}p_i]$;

$[M\hat{\eta}_{i+1}] = \alpha_i[M\hat{p}_i]$;

$\|\hat{x}_{i+1}\|_M^2 = \hat{x}_{i+1}^T[M\hat{x}_{i+1}]$;

$\beta_{i+1} = -\|s_{i+1}\|_{P^{-1}}$; $\tau = -\beta_{i+1}\alpha_i$;

end do

For notational simplicity, the above algorithm does not limit the number of limited-memory quasi-Newton updates. However, during each iteration the pairs $\{\eta_l, \gamma_l)\}$ and vectors $\{M\hat{\eta}_i\}$ can be updated so that only the m most recent pairs and corresponding vectors are stored.

Notice that this algorithm still requires only one matrix-vector multiply per iteration; specifically, a matrix-vector multiply is only required to compute $H\hat{v}_i$ for each i—everything else can be estimated from this one quantity.

7

Numerical Results

A combination line-search trust-region method based on the proposed phased-SSM method was tested on unconstrained problems from the CUTE test collection (see Bongartz et al. [1]). For comparison purposes, tests were also made using the same trust-region algorithm but with the subproblem solved using the Steihaug and Moré-Sorensen methods.

The first set of results compares unpreconditioned versions of the phased-SSM method and Steihaug's method. The second set of tables shows the results of implementing a quasi-Newton limited-memory updating scheme (as described in Section 6.2.2) to help precondition the second phase of the phased-SSM method. These tables focus on the effects of the quasi-Newton preconditioning scheme on the *overall* performance of the proposed method. The final set of tests compare the phased-SSM method and Steihaug's method when both are implemented with the incomplete Cholesky preconditioning scheme of Section 6.1.2.

7.1 Implementation Details

The trust-region method is considered to have solved a problem successfully when a trust-region iterate x_j satisfies

$$\|g(x_j)\| \leq 10^{-6} \max\{\|g(x_0)\|, |f(x_0)|\} \quad \text{and} \quad \tilde{\lambda}_n^j \geq 0,$$

where $\tilde{\lambda}_n^j$ is the best estimate of the leftmost eigenvalue of H_j. The alternative criterion

$$\|g(x_j)\| \leq 10^{-6} \quad \text{and} \quad \tilde{\lambda}_n^j \geq 0,$$

is used if $g(x_0) = 0$ and $f(x_0) = 0$. If x_0 is a maximizer, the presence of the term $f(x_0)$ prevents the trust-region algorithm from terminating at the initial non-optimal stationary point. However, additional measures would have to be taken if $f(x_0)$ was also small. If a solution is not found within n iterations, the algorithm terminates unsuccessfully, and the problem is considered unsolved.

In each iteration of the second phase, a suitable μ must satisfy Theorem 5.3.1. For the numerical results in this section, if $\tilde{\lambda}_n^{k+1} < 0$ then $\mu = \min\{0.1, \mu^*\}$, where

$$\frac{1}{\mu^*} = -\frac{\tilde{\lambda}_n^{k+1}}{s_{k+1}^T M s_{k+1}}.$$

Otherwise, if $\tilde{\lambda}_n^{k+1} \geq 0$, then $\mu = 0.1$. And, for each iteration of the second phase ν was taken to be the constant 10^{-4}.

7.1.1 Termination of phase one

In Steihaug's method and phase one of the phased-SSM method, the principal termination condition was based on the Dembo-Eisenstat-Steihaug criterion (Section 4.2). In particular, the Lanczos-PCG method terminated successfully with a point s_k inside the trust region if

$$\|H_j s_k + g_j\|_{M^{-1}} \leq \min\left\{10^{-1}, \|g_j\|_{M^{-1}}^{0.1}\right\} \|g_j\|_{M^{-1}}, \tag{7.1}$$

where $\|\cdot\|_{M^{-1}}$ denotes the elliptic norm associated with $M = M_j$. This condition forces a relative decrease in the residual and is equivalent to the termination criterion used by Gould et al. [18].

Two additional tests are used to determine if the Lanczos-PCG method has a sufficiently accurate solution of $H_j s = -g_j$. Theorem 3.4.3 (p. 52) indicates that a small off-diagonal element in the Lanczos tridiagonal T_k implies a small residual. In practice, it is reasonable to terminate Lanczos process if the ratio of an off-diagonal element to the maximum diagonal element is small. If the ratio drops below the square-root of the machine precision, the subproblem is terminated with an approximate unconstrained minimizer of the quadratic model $\mathcal{Q}_j(s)$.

The third and final phase-one test guards against over-solving the subproblem or attempting to solve the subproblem to an unattainable accuracy. This test terminates the Lanczos-PCG method if the residual norm satisfies

$$\|H_j s_k + g_j\| \leq \tau \|H_j\| \|x_j\|,$$

where τ is a small positive constant. To implement this test, the quantity $\|H_j\|$ is approximated by the largest diagonal element of the tridiagonal T_k.

7.1.2 Termination of phase two

The phase-two test for convergence occurs after computing an iterate from the reduced subspace solve. The test is based on the optimality conditions of Theorem 2.3.1 and involves the tolerance

$$\tau_j = \min\left\{10^{-1}, \|g_j\|_{M^{-1}}^{0.1}\right\} \|g_j\|_{M^{-1}}, \tag{7.2}$$

which is defined in terms of the M^{-1} norm to be consistent with the phase one condition. Given τ_j, the phase-two iterations are terminated when

$$\left\|\left(H_j + \sigma_k M_j\right) p_k + g_j\right\|_{M^{-1}} + \tfrac{1}{2}\left|s_k^T M_j s_k - \delta_j^2\right| \leq \tau_j, \tag{7.3}$$

where p_k an approximate solution of (2.36), p. 35. The vector p_k is given by $p_k = Q_k \bar{p}_k$, where $\bar{p}_k$ is the solution of the reduced system analogous to (2.36), i.e.,

$$(Q_k^T H_j Q_k + \sigma_k Q_k^T M_j Q_k)\bar{p}_k = Q_k^T g_j$$

(see Section 5.3.5). The condition (7.3) takes into account that the Moré-Sorensen algorithm gives only an *approximate* solution of the reduced subproblem (see Section 2.3.5).

It is also possible to test the Newton iterate $(\bar{s}_k, \bar{\sigma}_k)$, the approximate minimizer of L^{ν}_{μ,σ_e}, for convergence in the M^{-1}-norm. In this case, the error estimate can be computed as follows:

$$\left\| \left(H_j + \bar{\sigma}_k M_j\right)\bar{s}_k + g_j \right\|_{M^{-1}} + \tfrac{1}{2}\left|\bar{s}_k^T M_j \bar{s}_k - \delta_j^2\right|.$$

In practice, the error associated with Newton iterates $\{\bar{s}_k\}$ is usually larger than the error associated with the iterates obtained from the solution of the reduced problem. For the results in this chapter, only the iterates of the reduced problem were tested for convergence.

The reduced trust-region problem must be solved to an accuracy that is as least as good as that required for the full problem. Accordingly, the constants c_1 and c_2 of (2.39) were set at $c_1 = \min\{10^{-1}\tau_j, 10^{-6}\}$ and $c_2 = 0$, where τ_j is the tolerance defined in (7.2).

The phase-two method also guards against over-solving the subproblem using a two-norm residual test in the unconstrained case:

$$\|\eta_k\| \leq \tau \|H(x_j)\| \, \|x_j\|.$$

In all the runs, a limit of $n/10$ was imposed on the number Lanczos vectors computed in phase two. If the Lanczos-PCG method reaches the iteration limit, the Lanczos-PCG iterate associated with the smallest residual is returned as the approximate Newton step.

The second phase is also bound by a smaller iteration limit than the first phase. If the second phase is not converging well, this most likely indicates that the

estimate of the leftmost generalized eigenpair is poor. In this case, it is sensible to terminate the subproblem. In all the runs reported here, a maximum of 10 iterations was allowed in the second phase.

7.1.3 The line search

The approximate solution s_j of the jth trust-region subproblem is used to update the trust-region iterate as $x_{j+1} = x_j + \alpha_j s_j$, where α_j is obtained using a line search similar to Gertz's "biased" Wolfe line search (see Gertz [11]).

ALGORITHM 7.1.1. A BIASED WOLFE LINE SEARCH

Specify constants $0 < \eta_1 < \eta_2 < 1$, $0 < \eta_1 < \frac{1}{2}$, $0 < \eta_1 < \omega < 1$, $1 < \gamma_3$;

Find α_j satisfying the Wolfe conditions:

$f(x_j + \alpha_j s_j) \leq f(x_j) + \eta_1 \mathcal{Q}_j^-(\alpha_j s_j)$ and $|g(x_j + \alpha_j s_j)^T s_j| \leq -\omega \mathcal{Q}_j^{-\prime}(\alpha_j s_j)$;

$x_{j+1} = x_j + \alpha_j s_j$;

if $\big(f(x_{j+1} - f(x_j)\big)/\mathcal{Q}_j^-(s_j) \geq \eta_2$ **then**

  if $\|s_j\|_M = \delta_j$ **and** $\alpha_j = 1$ **then**

    $\delta_{j+1} = \gamma_3 \delta_j$;

  else if $\|s_j\|_M < \delta_j$ **and** $\alpha_j = 1$ **then**

    $\delta_{j+1} = \max\{\delta_j, \gamma_3 \|s_j\|_M\}$;

  else

    $\delta_{j+1} = \alpha_j \|s_j\|_M$;

  end if

else

  $\delta_{j+1} = \min\{\alpha_j \|s_j\|_M, \alpha_j \delta_j\}$;

end if

The line search parameters used for the experiments were $\eta_1 = 10^{-3}$, $\eta_2 = 0.25$, $\omega = 0.9$, and $\gamma_3 = 1.5$.

A key feature of the Wolfe line search proposed by Gertz is that the trust-region radius is updated as a function of α_j. The term "biased" is used by Gertz to refer

to a deliberate bias against reducing the trust-region radius when α_j is small. Algorithm 7.1.1 above differs from Gertz's line search in that it is possible for the trust-region radius to be reduced even when α_j is small. However, Algorithm 7.1.1 still retains a natural bias against decreasing the trust-region radius; in particular, the trust-region radius is not decreased if $\|s_j\|_M < \delta$ and $\alpha_j = 1$. Overall, this line search was more effective than both the traditional Wolfe line search and Gertz's biased Wolfe line search.

7.2 Problem selection

The test set was constructed using the CUTE interactive `select` tool, which allows the identification of groups of problems with certain features. In our case, the šelect tool was first used to locate the twice-continuously differentiable unconstrained problems for which the number of variables in the data file can be varied. Of these problems, the number of variables was set to a value in the range $100 \leq n \leq 1000$ according to criteria that we discuss below. The input for the šelect tool was as follows:

```
Objective function type         : *
Constraints type                : U  (No constraints)
Regularity                      : R  (twice-cont. differentiable)
Degree of available derivatives : *
Problem interest                : *
Explicit internal variables     : *
Number of variables             : v  (variable dimension)
Number of constraints           : 0.
```

A total of 87 problems were obtained via the CUTE `select` tool, and one problem (*eg2*) of fixed-dimension was added separately to the test set.

Several test problems were unsuitable for large values of n, i.e., $n \geq 1000$. In particular, the problems *chrosnb*, *errinros*, and *watson* had specific limits on the size of n, (i.e., less than 100); and, for three additional problems (*eigenals*, *eigenbls*, and *eigencls*) the recommended sizes of n were also well below 100. Other problems

could not be decoded using the CUTE tool `sifdec` for $n \geq 1000$. These problems included *arglina*, *arglinb*, *arglinc*, *brownal*, *hilberta*, *hilbertb*, *mancino*, *penalty3*, and *sensors*. Lastly, several problems were removed from the test set because convergence could not be obtained by either the phased-SSM method or Steihaug's method; these problems include *fletcbv3*, *fletchcr*, *indef*, *nonmsqrt*, *scosine*, and *sparsine*.

The final test set was thus composed of the following 67 problems suitable for minimization with $n \geq 1000$: *arwhead*, *bdqrtic*, *broydn7d*, *brybnd*, *chainwoo*, *cosine*, *cragglvy*, *curly10*, *curly20*, *curly30*, *dixmaana*, *dixmaanb*, *dixmaanc*, *dixmaand*, *dixmaane*, *dixmaanf*, *dixmaang*, *dixmaanh*, *dixmaani*, *dixmaanj*, *dixmaank*, *dixmaanl*, *dixon3dq*, *dqdrtic*, *dqrtic*, *edensch*, *eg2*, *engval1*, *extrosnb*, *fletcbv2*, *fletchbv*, *fminsrf2*, *fminsurf*, *freuroth*, *genhumps*, *genrose*, *liarwhd*, *morebv*, *msqrtals*, *msqrtbls*, *ncb20*, *ncb20b*, *noncvxu2*, *noncvxun*, *nondia*, *nondquar*, *penalty1*, *penalty2*, *powellsg*, *power*, *quartc*, *sbrybnd*, *schmvett*, *scurly10*, *scurly20*, *scurly30*, *sinquad*, *sparsqur*, *smpsrtls*, *srosenbr*, *testquad*, *tointgss*, *tquartic*, *tridia*, *vardim*, *vareigvl*, and *woods*. For all problems, the dimension was $n = 1000$ unless otherwise recommended by CUTE documentation; however, in all cases, $n \geq 1000$.

All methods were implemented and run in MATLAB.

7.3 Results with No Preconditioning

The first set of results compare the phased-SSM method with Steihaug's method when both methods are implemented without preconditioning. In order to obtain results that reflect real differences in the solvers, certain problems were excluded from the test set if the CG method appeared not to be converging in Steihaug's method and the first phase of the phased-SSM method. This behavior is caused by the lack of a good preconditioner and is independent of the method used to solve the subproblem. As a rule of thumb, a problem was excluded if more than $n/10$ Lanczos vectors (CG iterations) were required for the Lanczos-CG solve.

Based on this criterion, problems *curly10*, *curly20*, *curly30*, *dixmaani*, *dixon3dq*, *fletcbv2*, *fletchbv*, *genhumps*, *morebv*, *msqrtals*, *msqrtbls*, *sbrybnd*, *scurly10*, *scurly20*, *scurly30*, *sinquad*, and *tridia* were excluded, leaving a total of 50 problems in the test set.

Tables 7.1–7.2 give the results of applying Steihaug's method and the phased-SSM method on this reduced set of problems. For each solver the columns list the total number of function evaluation ("Fe"), the total number of matrix-vector products ("Prods"), the total cpu seconds required ("Cpu"), and the final value of f found by the solver. The final value of f is listed to help identify situations in which the solvers converged to a different local minimizer. In this case the computational effort cannot be compared directly because the number of iterations and function evaluations needed to find different local solutions may be completely different.

Tables 7.1–7.2 show that Steihaug's method and the phased-SSM method behave very similarly on many problems. If both methods require the same number of function evaluations, this implies that the approximate solutions of every subproblem is in the interior of the trust region. In these cases, the phased-SSM method never enters the second phase and is as efficient as Steihaug's method. (Note that the extra matrix-vector products are associated with the initial estimate of the leftmost generalized eigenvector for each subproblem (see Section 5.2.1).)

Tables 7.3–7.4 compare the number of function evaluations of Steihaug's method, phased-SSM, and the Moré-Sorensen method. The function evaluations reported for the Moré-Sorensen method may be interpreted as the number of evaluations required if the iterative methods were able to solve the subproblem to the accuracy of 10^{-7}, i.e., the same tolerance as that used for the Moré-Sorensen solve. (Recall that the accuracy of the subproblem solve for the iterative methods is a function of norm of the initial residual, i.e., see (7.1).)

On the few problems where phased-SSM required more function evaluations that Steihaug's method, the performance of phased-SSM was often similar to that of the Moré-Sorensen method. The superiority of Steihaug's method in these

cases is likely to be the effect of good fortune rather than a consistently better subproblem solution.

The results of Tables 7.1–7.2 are summarized in Table 7.7. The phased-SSM method was able to solve one more problem than Steihaug's method. On the 49 problems on which both methods converge, the phased-SSM method required 30.4% fewer function evaluations than Steihaug's method. In comparison, Gould et al. [18] reported that GLTR solved 16 out of 17 problems and only obtained a 12.5% improvement in the number of function evaluations relative to Steihaug's method.

7.4 Quasi-Newton Preconditioning

Tables 7.5–7.6 give the results of using quasi-Newton preconditioning for the Newton solve in the second phase. The results imply that quasi-Newton preconditioning has little or no effect on the efficiency of the method. Moreover, preconditioning does not lead to any additional problems being solved. The unpreconditioned results in Table 7.7 did not include results from problem *genrose*. If this problem is included, then the results suggest that quasi-Newton preconditioning had little impact on the total number of function evaluations, but had a negative impact on the number of matrix-vector products.

In theory, the limited-memory quasi-Newton preconditioning scheme should be helpful as the second phase converges. However, up until that time, it could be unhelpful by building irrelevant information into the preconditioner. To see this, consider the case when L^{ν}_{μ,σ_e} varies widely between iterations. In this case, second-order information from a very different function is built into the preconditioner for the Newton equations associated with minimizing the current L^{ν}_{μ,σ_e} function.

Based on the results in Tables 7.5–7.6, there seems little benefit in using limited-memory quasi-Newton preconditioning in the phased-SSM method.

7.5 Incomplete Cholesky Preconditioning

It is generally accepted by the optimization community that so-called "general-purpose" preconditioners are not always beneficial. For well-conditioned problems, such preconditioners can disturb an eigenvalue spectrum that is already clustered. However, for ill-conditioned problems, a preconditioner can be essential.

Incomplete Cholesky preconditioners, generally work best with problems with very sparse or highly structured Hessians. In this situation, given sufficient memory, the incomplete Cholesky factorization may be exact and yield an ideal preconditioner. However, on problems with dense or unstructured Hessians, the spectrum of a preconditioner based on an incomplete Cholesky factorization may look nothing like the spectrum of the Hessian.

The incomplete Cholesky preconditioners of Section 6.1.2 depend on the number of nonzeros in the matrix—in particular, it will build a dense preconditioner for a dense matrix. For large problems, the time required to factorize a dense matrix is prohibitive, and so the following problems with dense Hessians were excluded from the test set: *fminsurf*, *msqrtals*, *msqrtbls*, *penalty1*, *penalty2*, *power*, *vardim*, and *vareigvl*.

The results obtained using an incomplete Cholesky preconditioner are found in Tables 7.8–7.10. In particular, Table 7.10 concerns those CUTE problems that could not be solved using an unpreconditioned versions of both Steihaug's method and the phased-SSM method.

The results of Tables 7.8 and 7.9 provide a good indication of when preconditioning is beneficial. For example, for problem *brybnd*, Steihaug's method requires 33 function evaluations and 19 matrix-vector products, compared to 12 function evaluations and 46 matrix-vector products without preconditioning. The Hessian of *brybnd* is a band matrix and the preconditioner is exact. This implies that only one PCG iterate is required for each solve. (The trust-region iteration may require more than one function evaluation because of the Wolfe line search.) In these ta-

bles, this relationship between the number of function evaluations and the number of matrix-vector products is generally indicative of the quality of the preconditioner. On other banded problems such as *bdqrtic*, *broydn7d*, *brybnd*, *cragglvy*, *ncb20*, *ncb20b*, and *spmsrtls*, the number of function evaluations for Steihaug's method is at least as large as the number of matrix-vector products, indicating that a preconditioner was beneficial.

There are several peculiarities in the tables of the incomplete Cholesky preconditioning results. First, in several cases, a considerably better solution was found via the preconditioned algorithms, e.g., as in the case for *dqrtic* and *testquad*. Another peculiarity is that when the incomplete Cholesky preconditioner is used on a (frequently indefinite) banded Hessian, the proposed method may not be as effective as Steihaug's method, e.g., see *ncb20* and *spmsrtls*. One final peculiarity is that the proposed method terminated unsuccessfully on *ncb20b* because of a failure in the Wolfe line search, but was (unknowingly) close to a minimum. This run is marked with a superscript d.

Incomplete Cholesky preconditioning is not always beneficial. On problems such as *genrose* and *chainwoo*, the unpreconditioned algorithms appear to be much more efficient. However, care must be taken when drawing conclusions based only on differences in the number of function evaluations. In particular, the choice of initial trust-region radius is more complicated for preconditioned algorithms. Consider the initial radius $\delta_0 = 1$ used for the unpreconditioned algorithm. If the condition number of the preconditioner is large, then any PCG step s will be such that $s^T M s > 1$, i.e., the choice of $\delta_0 = 1$ causes the iterates to repeatedly hit the boundary until the trust region is expanded to an appropriate size. In practice, this means that the number of function evaluations for the preconditioned and unpreconditioned algorithms cannot be compared directly. (The initial value $\delta_0 = 100$ was used for the preconditioned runs.)

The preconditioned phased-SSM method also exhibits numerical instability on some CUTE problems. The table entries for these problems are marked with as-

terisks. In the second phase, the preconditioner plays a fundamental role in the proposed method. Implicit products with the preconditioner are required to form the Newton equations and to implement the reduced subspace solve. Unfortunately, the (implicitly-formed) matrix-vector products with the preconditioner are subject to rounding errors associated with a loss of orthogonality amongst the Lanczos vectors. The more Lanczos vectors needed to solve the linear system, the more likely the implicit preconditioner matrix-vector products will be contaminated by rounding error.

During the second phase, the loss of accuracy in the implicit matrix-vector products can be significant. In this case, the subspaces formed using H and M will differ significantly from their true values. For many of the CUTE problems, this inconsistency leads to the failure of the Moré-Sorensen algorithm for the reduced problem.

It is of interest to understand why these failures do not occur when preconditioning is not used. The condition number of the preconditioner is a key factor in the loss of accuracy in the implicit matrix-vector products with the preconditioner. Moreover, if one considers that the unpreconditioned case is effectively using a perfectly well-conditioned preconditioner (i.e., $M = I$), then it is not surprising that there is more numerical inaccuracy when M is an incomplete Cholesky preconditioner. This suggests that additional care must be taken to prevent artificial breakdowns when preconditioning is used.

This explanation of the cause of the failures may be verified experimentally. For the case of incomplete Cholesky preconditioning it is possible to form "exact" explicit products with $M = R^T R$. In an experiment, the phased-SSM method was modified so that products with M were formed explicitly. The justification for this is that the breakdown occurs during the reduced subspace solve; thus, instead of performing explicit matrix-vector products with M during the Newton solve, it is more efficient to wait until the Newton update is found before forming the explicit matrix-vector product with M. Consequently, before each reduced subspace solve,

three explicit products with M are computed, resulting in six additional matrix-vector products for each iteration of the second phase. Tables 7.11–7.13 give the results of this more stable implementation. There were no numerical breakdowns in the reduced subspace solve as a result of this implementation. As noted, this implementation is more expensive because it requires explicit matrix-vector products with M.

Preconditioning has two effects on CG-based trust-region methods. Most obviously, preconditioners are designed to accelerate the convergence of CG-type methods. A little less obviously, they impose a scaling on the trust-region subproblem. To understand the effect of the incomplete Cholesky preconditioner on the CUTE problems, it is helpful to consider the result of using a *direct* method (e.g., the Moré-Sorensen method) on subproblems scaled by the incomplete Cholesky preconditioner, i.e., solve

$$\underset{s\in\mathbb{R}^n}{\text{minimize}} \;\; g^T s + \tfrac{1}{2} s^T H s \quad \text{subject to} \;\; \|Ns\| = \delta,$$

where $M = N^T N$ and M is the incomplete Cholesky preconditioner for H.

A sparse implementation of the preconditioned Moré-Sorensen method was used in place of the iterative subproblem solvers in the generic trust-region algorithm. Tables 7.14–7.15 compare the function evaluations required by the direct Moré-Sorensen implementation to Steihaug's method and the proposed method. (The dense problems *fminsurf*, *penalty1*, *penalty2*, *power*, *vardim*, and *vareigvl* were removed from the test set because of excessive run times.) Surprisingly, the iterative trust-region methods often required fewer function evaluations than the direct method. This suggests that when the subproblem solution is found to high accuracy, the number of function evaluations may *increase*. One possible reason for this is that incomplete Cholesky preconditioning may induce a poor scaling of the subproblem variables. Moreover, if the preconditioners change dramatically between iterations, then updating the trust-region radius based on its value from the previous iteration may be problematic.

Based on their numerical results, Gould et al. [18] also conclude that, overall, incomplete Cholesky preconditioning is not beneficial. Results obtained using the GLTR method indicate that unpreconditioned iterative solvers perform better than their preconditioned counterparts—which include a banded preconditioner, an incomplete Cholesky preconditioner, and a modified Cholesky preconditioner. The results presented in this thesis and the observations of Gould et al. cast doubt on the effectiveness of "general-purpose" preconditioning methods on this test set.

7.6 Concluding Remarks

The proposed phased-SSM method can obtain highly accurate approximate solutions to the trust-region subproblem regardless of whether a solution lies in the interior or on the boundary of the trust region. The first phase of the phased-SSM method is almost identical to Steihaug's method. If the solution to the subproblem lies in the strict interior of the trust region, the phased-SSM method generates pure CG iterates to solve the Newton system. In the second phase, is a solution to the subproblem lies on the boundary, then the phased-SSM method can obtain solutions to any prescribed accuracy. Numerical results suggest that obtaining approximate subproblem solutions of higher accuracy reduces the required number of function evaluations: In the unpreconditioned case, the proposed trust-region method required 30% fewer function evaluations than Steihaug's method.

A trust-region method based on the phased-SSM method will generally be more efficient than Steihaug's method when the cost of performing function evaluations is relatively expensive. Note that this is not usually the case for the unconstrained problems in the CUTE set. On more realistic problems, the evaluation of the objective may require considerably more CPU time, and thus, it would be preferable to invest more matrix-vector products in order to reduce the number of function evaluations. In this case, the results in this thesis suggest that a trust-region method will be more efficient if it is based on the proposed method rather than on

Steihaug's method.

Table 7.1: Steihaug and phased-SSM. Problems A–F, $M = I$; $\delta_0 = 1$.

		Steihaug				phased-SSM			
Problem	n	Fe	Prods	Cpu	f	Fe	Prods	Cpu	f
arwhead	1000	6	6	1	1.7e-10	6	11	1	1.7e-10
bdqrtic	1000	14	41	4	4.0e+03	13	47	5	4.0e+03
broydn7d	1000	137	414	34	3.8e+02	77	1661	158	3.4e+02
brybnd	1000	12	46	4	6.7e-07	12	58	5	3.9e-07
chainwoo	1000	27	57	6	1.3e+01	23	73	9	3.9e+03
cosine	1000	12	10	2	-1.0e+03	12	18	2	-1.0e+03
cragglvy	1000	14	36	4	3.4e+02	14	49	5	3.4e+02
dixmaana	1500	13	11	4	1.0e+00	13	22	6	1.0e+00
dixmaanb	1500	13	11	5	1.0e+00	13	22	6	1.0e+00
dixmaanc	1500	13	11	4	1.0e+00	13	22	6	1.0e+00
dixmaand	1500	14	12	5	1.0e+00	14	24	6	1.0e+00
dixmaane	1500	14	93	14	1.0e+00	14	95	15	1.0e+00
dixmaanf	1500	15	30	7	1.0e+00	15	43	9	1.0e+00
dixmaang	1500	15	24	7	1.0e+00	15	37	8	1.0e+00
dixmaanh	1500	15	19	6	1.0e+00	15	32	7	1.0e+00
dixmaanj	1500	16	40	9	1.0e+00	16	54	11	1.0e+00
dixmaank	1500	16	30	8	1.0e+00	16	44	9	1.0e+00
dixmaanl	1500	16	25	7	1.0e+00	16	39	8	1.0e+00
dqdrtic	1000	14	11	2	1.9e-03	14	21	2	3.5e-05
dqrtic	1000	27	20	3	1.6e+11	28	40	5	5.5e+10
edensch	1000	19	19	3	6.0e+03	19	33	4	6.0e+03
eg2	1000	4	3	1	-1.0e+03	4	6	1	-1.0e+03
engval1	1000	14	17	2	1.1e+03	14	28	3	1.1e+03
extrosnb	1000	31	70	7	2.2e-02	29	75	7	4.4e-02
fminsrf2	1024	336	1072	99	1.0e+00	78	1702	169	1.0e+00
fminsurf	1024	318	672	343	1.0e+00	79	1430	206	1.0e+00
freuroth	1000	16	19	3	1.2e+05	16	30	3	1.2e+05

Table 7.2: Steihaug and phased-SSM. Problems G–Z, $M = I$, $\delta_0 = 1$.

		Steihaug				phased-SSM			
Problem	n	Fe	Prods	Cpu	f	Fe	Prods	Cpu	f
genrose	1000	$> n$	—	—	—	805	24529	2070	1.0e+00
liarwhd	1000	19	27	4	4.0e-07	19	42	5	1.5e-07
ncb20	1010	60	447	32	9.1e+02	114	2761	257	9.2e+02
ncb20b	1000	10	62	5	1.7e+03	9	68	6	1.7e+03
noncvxu2	1000	36	26	4	1.2e+06	45	242	31	1.6e+06
noncvxun	1000	37	28	4	6.9e+05	50	313	43	7.8e+05
nondia	1000	4	3	1	6.3e-03	4	6	1	6.3e-03
nondquar	1000	23	116	9	5.5e-04	23	194	21	6.4e-03
penalty1	1000	28	17	17	3.0e+13	28	34	18	3.0e+13
penalty2	1000	2	1	1	1.4e+83	2	2	1	1.4e+83
powellsg	1000	16	40	4	4.6e-03	15	54	5	6.8e-03
power	1000	15	33	15	3.6e+04	15	47	16	3.6e+04
quartc	1000	27	20	3	1.6e+11	28	40	5	5.5e+10
schmvett	1000	9	37	3	-3.0e+03	8	37	3	-3.0e+03
sparsqur	1000	14	23	3	4.2e-03	14	48	5	4.3e-03
spmsrtls	1000	18	126	8	2.8e-09	29	503	43	3.3e-09
srosenbr	1000	9	10	2	1.6e-09	9	18	2	2.3e-09
testquad	1000	12	103	7	7.1e+01	12	123	9	7.1e+01
tointgss	1000	15	13	2	1.0e+01	15	23	3	1.0e+01
tquartic	1000	17	23	3	5.7e-14	16	35	4	5.8e-14
vardim	1000	14	13	23	5.4e+14	14	26	24	5.4e+14
vareigvl	1000	14	26	13	3.5e-04	14	38	14	3.5e-04
woods	1000	12	14	2	2.0e+03	13	28	3	2.0e+03

Table 7.3: Steihaug, phased-SSM and Moré-Sorensen. Problems A–F, $M = I$, $\delta_0 = 1$.

		Steihaug		phased-SSM		Moré-Sorensen	
Problem	n	Fe	f	Fe	f	Fe	f
arwhead	1000	6	1.7e-10	6	1.7e-10	6	6.7e-13
bdqrtic	1000	14	4.0e+03	13	4.0e+03	11	3.9e+03
broydn7d	1000	137	3.8e+02	77	3.4e+02	39	3.8e+02
brybnd	1000	12	6.7e-07	12	3.9e-07	17	1.9e-07
chainwoo	1000	27	1.3e+01	23	3.9e+03	264	1.0e+02
cosine	1000	12	-1.0e+03	12	-1.0e+03	48	-7.1e+02
cragglvy	1000	14	3.4e+02	14	3.4e+02	14	3.6e+02
dixmaana	1500	13	1.0e+00	13	1.0e+00	12	1.0e+00
dixmaanb	1500	13	1.0e+00	13	1.0e+00	12	1.0e+00
dixmaanc	1500	13	1.0e+00	13	1.0e+00	22	1.0e+00
dixmaand	1500	14	1.0e+00	14	1.0e+00	13	1.0e+00
dixmaane	1500	14	1.0e+00	14	1.0e+00	13	1.0e+00
dixmaanf	1500	15	1.0e+00	15	1.0e+00	32	1.0e+00
dixmaang	1500	15	1.0e+00	15	1.0e+00	37	1.0e+00
dixmaanh	1500	15	1.0e+00	15	1.0e+00	33	1.0e+00
dixmaanj	1500	16	1.0e+00	16	1.0e+00	39	1.0e+00
dixmaank	1500	16	1.0e+00	16	1.0e+00	51	1.0e+00
dixmaanl	1500	16	1.0e+00	16	1.0e+00	47	1.0e+00
dqdrtic	1000	14	1.9e-03	14	3.5e-05	12	2.5e-27
dqrtic	1000	27	1.6e+11	28	5.5e+10	32	3.2e+10
edensch	1000	19	6.0e+03	19	6.0e+03	22	6.0e+03
eg2	1000	4	-1.0e+03	4	-1.0e+03	7	-1.0e+03
engval1	1000	14	1.1e+03	14	1.1e+03	13	1.1e+03
extrosnb	1000	31	2.2e-02	29	4.4e-02	27	1.7e-02
fminsrf2	1024	336	1.0e+00	78	1.0e+00	111	1.0e+00
freuroth	1000	16	1.2e+05	16	1.2e+05	13	1.2e+05

Table 7.4: Steihaug, phased-SSM and Moré-Sorensen. Problems G–Z, $M = I$, $\delta_0 = 1$

		Steihaug		phased-SSM		Moré-Sorensen	
Problem	n	Fe	f	Fe	f	Fe	f
genrose	1000	$> n$	—	805	1.0e+00	807	1.0e+00
liarwhd	1000	19	4.0e-07	19	1.5e-07	12	4.7e-08
ncb20	1010	60	9.1e+02	114	9.2e+02	71	9.3e+02
ncb20b	1000	10	1.7e+03	9	1.7e+03	11	1.7e+03
noncvxu2	1000	36	1.2e+06	45	1.6e+06	48	9.9e+05
noncvxun	1000	37	6.9e+05	50	7.8e+05	48	1.3e+06
nondia	1000	4	6.3e-03	4	6.3e-03	5	1.0e-07
nondquar	1000	23	5.5e-04	23	6.4e-03	15	6.7e-05
powellsg	1000	16	4.6e-03	15	6.8e-03	15	6.1e-03
quartc	1000	27	1.6e+11	28	5.5e+10	32	3.2e+10
schmvett	1000	9	-3.0e+03	8	-3.0e+03	6	-3.0e+03
sparsqur	1000	14	4.2e-03	14	4.3e-03	14	3.2e-03
spmsrtls	1000	18	2.8e-09	29	3.3e-09	29	1.0e-09
srosenbr	1000	9	1.6e-09	9	2.3e-09	9	1.5e-15
testquad	1000	12	7.1e+01	12	7.1e+01	9	2.3e+01
tointgss	1000	15	1.0e+01	15	1.0e+01	16	1.0e+01
tquartic	1000	17	5.7e-14	16	5.8e-14	11	1.0e-17
woods	1000	12	2.0e+03	13	2.0e+03	13	2.0e+03

Table 7.5: Quasi-Newton preconditioning. Problems A–F, $M = I$, $\delta_0 = 1$.

		phased-SSM				phased-SSM-QN			
Problem	n	Fe	Prods	Cpu	f	Fe	Prods	Cpu	f
arwhead	1000	6	11	1	1.7e-10	6	11	1	1.7e-10
bdqrtic	1000	13	47	5	4.0e+03	13	47	5	4.0e+03
broydn7d	1000	77	1661	158	3.4e+02	76	1986	204	3.4e+02
brybnd	1000	12	58	5	3.9e-07	12	58	5	3.9e-07
chainwoo	1000	23	73	9	3.9e+03	24	106	11	3.9e+03
cosine	1000	12	18	2	-1.0e+03	12	18	2	-1.0e+03
cragglvy	1000	14	49	5	3.4e+02	14	49	5	3.4e+02
dixmaana	1500	13	22	6	1.0e+00	13	22	6	1.0e+00
dixmaanb	1500	13	22	6	1.0e+00	13	22	6	1.0e+00
dixmaanc	1500	13	22	6	1.0e+00	13	22	6	1.0e+00
dixmaand	1500	14	24	6	1.0e+00	14	24	6	1.0e+00
dixmaane	1500	14	95	15	1.0e+00	14	95	14	1.0e+00
dixmaanf	1500	15	43	9	1.0e+00	15	43	9	1.0e+00
dixmaang	1500	15	37	8	1.0e+00	15	37	8	1.0e+00
dixmaanh	1500	15	32	7	1.0e+00	15	32	7	1.0e+00
dixmaanj	1500	16	54	11	1.0e+00	16	54	10	1.0e+00
dixmaank	1500	16	44	9	1.0e+00	16	44	9	1.0e+00
dixmaanl	1500	16	39	8	1.0e+00	16	39	8	1.0e+00
dqdrtic	1000	14	21	2	3.5e-05	14	21	2	3.5e-05
dqrtic	1000	28	40	5	5.5e+10	28	40	5	5.5e+10
edensch	1000	19	33	4	6.0e+03	19	33	4	6.0e+03
eg2	1000	4	6	1	-1.0e+03	4	6	1	-1.0e+03
engval1	1000	14	28	3	1.1e+03	14	28	3	1.1e+03
extrosnb	1000	29	75	7	4.4e-02	29	75	7	4.4e-02
fminsrf2	1024	78	1702	169	1.0e+00	68	1673	165	1.0e+00
fminsurf	1024	79	1430	206	1.0e+00	88	2021	280	1.0e+00
freuroth	1000	16	30	3	1.2e+05	16	30	3	1.2e+05

Table 7.6: Quasi-Newton preconditioning. Problems G–Z, $M = I$, $\delta_0 = 1$.

		phased-SSM				phased-SSM-QN			
Problem	n	Fe	Prods	Cpu	f	Fe	Prods	Cpu	f
genrose	1000	805	24529	2070	1.0e+00	813	25544	2292	1.0e+00
liarwhd	1000	19	42	5	1.5e-07	19	42	5	1.5e-07
ncb20	1010	114	2761	257	9.2e+02	104	3047	306	9.2e+02
ncb20b	1000	9	68	6	1.7e+03	9	68	6	1.7e+03
noncvxu2	1000	45	242	31	1.6e+06	45	255	33	1.6e+06
noncvxun	1000	50	313	43	7.8e+05	49	281	38	9.6e+05
nondia	1000	4	6	1	6.3e-03	4	6	1	6.3e-03
nondquar	1000	23	194	21	6.4e-03	23	194	21	6.4e-03
penalty1	1000	28	34	18	3.0e+13	28	34	18	3.0e+13
penalty2	1000	2	2	1	1.4e+83	2	2	1	1.4e+83
powellsg	1000	15	54	5	6.8e-03	15	54	5	6.8e-03
power	1000	15	47	16	3.6e+04	15	47	16	3.6e+04
quartc	1000	28	40	5	5.5e+10	28	40	5	5.5e+10
schmvett	1000	8	37	3	-3.0e+03	8	37	3	-3.0e+03
sparsqur	1000	14	48	5	4.3e-03	14	48	4	4.3e-03
spmsrtls	1000	29	503	43	3.3e-09	28	520	46	1.6e-08
srosenbr	1000	9	18	2	2.3e-09	9	18	2	2.3e-09
testquad	1000	12	123	9	7.1e+01	12	123	8	7.1e+01
tointgss	1000	15	23	3	1.0e+01	15	23	3	1.0e+01
tquartic	1000	16	35	4	5.8e-14	16	35	4	5.8e-14
vardim	1000	14	26	24	5.4e+14	14	25	24	5.4e+14
vareigvl	1000	14	38	14	3.5e-04	14	38	14	3.5e-04
woods	1000	13	28	3	2.0e+03	13	28	3	2.0e+03

Table 7.7: Comparison of unpreconditioned methods. $M = I$, $\delta_0 = 1$.

	Steihaug	phased-SSM	phased-SSM-QN
Number of problems solved	49	50	50
Total function evals. (excl. *genrose*)	1572	1094	1081
Total mat-vec mults. (excl. *genrose*)	4027	10398	11602
Percent improvement in function evals.	—	30.4%	31.2%

Table 7.8: Incomplete Cholesky preconditioning. Problems A–F, $\delta_0 = 100$.

		Steihaug				phased-SSM			
Problem	n	Fe	Prods	Cpu	f	Fe	Prods	Cpu	f
arwhead	1000	6	5	3	6.7e-13	6	10	4	6.7e-13
bdqrtic	1000	10	9	5	4.0e+03	10	18	7	4.0e+03
broydn7d	1000	87	44	23	4.0e+02	97	179	95	4.3e+02
brybnd	1000	33	19	10	3.6e-09	21	27	12	2.2e-11
chainwoo	1000	86	59	27	6.3e+01	**	**	**	**
cosine	1000	5	4	3	-1.0e+03	30	445	176	-9.6e+02
cragglvy	1000	14	13	7	3.4e+02	14	26	9	3.4e+02
dixmaana	1500	7	6	5	1.0e+00	10	13	9	1.0e+00
dixmaanb	1500	23	14	15	1.0e+00	45	60	57	1.0e+00
dixmaanc	1500	23	14	15	1.0e+00	55	138	136	1.0e+00
dixmaand	1500	25	16	17	1.0e+00	51	69	64	1.0e+00
dixmaane	1500	41	23	18	1.0e+00	63	138	100	1.0e+00
dixmaanf	1500	24	15	17	1.0e+00	**	**	**	**
dixmaang	1500	24	25	17	1.0e+00	**	**	**	**
dixmaanh	1500	30	18	19	1.0e+00	48	71	67	1.0e+00
dixmaani	1500	33	22	17	1.0e+00	**	**	**	**
dixmaanj	1500	31	18	20	1.0e+00	19	68	64	1.0e+00
dixmaank	1500	19	14	16	1.0e+00	**	**	**	**
dixmaanl	1500	31	20	22	1.0e+00	**	**	**	**
dqdrtic	1000	8	6	3	2.3e-24	8	12	4	2.9e-24
dqrtic	1000	37	22	9	3.9e+10	37	44	15	3.9e+10
edensch	1000	15	12	6	6.0e+03	15	24	8	6.0e+03
eg2	1000	4	3	2	-1.0e+03	4	6	2	-1.0e+03
engval1	1000	8	7	4	1.1e+03	8	14	5	1.1e+03
extrosnb	1000	22	19	9	3.0e-02	22	38	13	3.0e-02
fminsrf2	1024	21	12	7	1.0e+00	21	23	11	1.0e+00
freuroth	1000	9	7	3	1.2e+05	9	14	5	1.2e+05

Table 7.9: Incomplete Cholesky preconditioning. Problems G–Z, $\delta_0 = 100$

		Steihaug				phased-SSM			
Problem	n	Fe	Prods	Cpu	f	Fe	Prods	Cpu	f
genrose	1000	$> n$	—	—	—	$> n$	—	—	—
liarwhd	1000	15	23	28	3.1e-08	15	36	36	9.3e-10
ncb20	1010	76	49	46	9.0e+02	105	236	238	9.3e+02
ncb20b	1000	27	15	13	1.7e+03	13^d	17^d	20^d	$1.7e+03^d$
noncvxu2	1000	60	41	21	2.0e+06	178	468	271	2.6e+06
noncvxun	1000	52	35	18	1.5e+06	303	575	319	2.0e+06
nondia	1000	7	6	8	1.5e-03	7	12	13	2.3e-03
nondquar	1000	13	12	7	3.5e-06	13	24	9	3.5e-06
powellsg	1000	12	11	6	3.8e-03	12	22	8	3.8e-03
quartc	1000	37	22	9	3.9e+10	37	44	14	3.9e+10
schmvett	1000	4	3	2	-3.0e+03	4	6	2	-3.0e+03
sparsqur	1000	14	13	15	1.8e-03	14	26	21	1.8e-03
spmsrtls	1000	33	20	9	4.6e-09	76	166	70	3.7e-01
srosenbr	1000	10	8	4	3.5e-09	10	16	6	3.5e-09
testquad	1000	15	9	4	1.9e-25	15	18	6	4.5e-25
tointgss	1000	3	2	1	1.0e+01	3	4	2	1.0e+01
tquartic	1000	4	4	6	3.4e-13	4	7	7	3.4e-13
woods	1000	14	11	5	2.0e+03	14	22	8	2.0e+03

Table 7.10: Incomplete Cholesky preconditioning. Results for problems that could not be solved by unpreconditioned Steihaug or phased-SSM, $\delta_0 = 100$.

		Steihaug				phased-SSM			
Problem	n	Fe	Prods	Cpu	f	Fe	Prods	Cpu	f
curly10	1000	16	11	7	-1.0e+05	27	98	56	-1.0e+05
curly20	1000	16	12	11	-1.0e+05	15	65	57	-1.0e+05
curly30	1000	30	18	23	-1.0e+05	22	72	83	-1.0e+05
dixon3dq	1000	2	1	1	5.5e-28	2	2	2	5.5e-28
fletcbv2	1000	2	1	1	-5.0e-01	2	2	2	-5.0e-01
genhumps	1000	437	421	172	7.1e+01	**	**	**	**
morebv	1000	2	1	1	7.3e-13	2	2	1	7.3e-13
sbrybnd	1000	30	17	9	1.6e-09	63	65	27	2.0e-10
scurly10	1000	16	11	7	-1.0e+05	42	96	50	-1.0e+05
scurly20	1000	14	10	10	-1.0e+05	21	68	54	-1.0e+05
scurly30	1000	21	13	17	-1.0e+05	36	77	88	-1.0e+05
sinquad	1000	94	90	110	1.1e-06	97	199	278	1.3e-06
tridia	1000	6	4	2	1.5e-26	6	8	3	8.5e-27

Table 7.11: Incomplete Cholesky preconditioning using explicit matrix-vector products with M. Problems A–F, $\delta_0 = 100$.

		Steihaug				phased-SSM			
Problem	n	Fe	Prods	Cpu	f	Fe	Prods	Cpu	f
arwhead	1000	6	5	3	6.7e-13	6	10	4	6.7e-13
bdqrtic	1000	10	9	5	4.0e+03	10	30	12	4.0e+03
broydn7d	1000	87	44	23	4.0e+02	99	689	163	4.3e+02
brybnd	1000	33	19	10	3.6e-09	21	73	27	2.2e-06
chainwoo	1000	86	59	27	6.3e+01	123	612	163	1.0e+00
cosine	1000	5	4	3	-1.0e+03	30	981	133	-9.5e+02
cragglvy	1000	14	13	7	3.4e+02	14	42	15	3.4e+02
dixmaana	1500	7	6	5	1.0e+00	10	39	31	1.0e+00
dixmaanb	1500	23	14	15	1.0e+00	45	232	184	1.0e+00
dixmaanc	1500	23	14	15	1.0e+00	55	514	270	1.0e+00
dixmaand	1500	25	16	17	1.0e+00	51	245	196	1.0e+00
dixmaane	1500	41	23	18	1.0e+00	63	446	248	1.0e+00
dixmaanf	1500	24	15	17	1.0e+00	74	231	244	1.0e+00
dixmaang	1500	24	25	17	1.0e+00	52	276	173	1.0e+00
dixmaanh	1500	30	18	19	1.0e+00	47	161	139	1.0e+00
dixmaani	1500	33	22	17	1.0e+00	120	734	455	1.0e+00
dixmaanj	1500	31	18	20	1.0e+00	19	117	79	1.0e+00
dixmaank	1500	19	14	16	1.0e+00	19	156	92	1.0e+00
dixmaanl	1500	31	20	22	1.0e+00	27	122	89	1.0e+00
dqdrtic	1000	8	6	3	2.3e-24	8	32	12	2.9e-24
dqrtic	1000	37	22	9	3.9e+10	37	116	43	3.9e+10
edensch	1000	15	12	6	6.0e+03	15	44	16	6.0e+03
eg2	1000	4	3	2	-1.0e+03	4	6	2	-1.0e+03
engval1	1000	8	7	4	1.1e+03	8	22	8	1.1e+03
extrosnb	1000	22	19	9	3.0e-02	22	62	23	3.0e-02
fminsrf2	1024	21	12	7	1.0e+00	21	59	27	1.0e+00
freuroth	1000	9	7	3	1.2e+05	9	26	10	1.2e+05

Table 7.12: Incomplete Cholesky preconditioning using explicit matrix-vector products with M. Problems G–Z, $\delta_0 = 100$.

		Steihaug				phased-SSM			
Problem	n	Fe	Prods	Cpu	f	Fe	Prods	Cpu	f
genrose	1000	$> n$	—	—	—	$> n$	—	—	—
liarwhd	1000	15	23	28	3.1e-08	15	52	44	9.3e-10
ncb20	1010	76	49	46	9.0e+02	100	839	305	9.3e+02
ncb20b	1000	27	15	13	1.7e+03	13^d	121^d	22^d	$1.7e+03^d$
noncvxu2	1000	60	41	21	2.0e+06	278	2378	566	2.5e+06
noncvxun	1000	52	35	18	1.5e+06	285	2452	586	2.0e+06
nondia	1000	7	6	8	1.5e-03	7	28	20	2.3e-03
nondquar	1000	13	12	7	3.5e-06	13	24	9	3.5e-06
powellsg	1000	12	11	6	3.8e-03	12	30	11	3.8e-03
quartc	1000	37	22	9	3.9e+10	37	116	44	3.9e+10
schmvett	1000	4	3	2	-3.0e+03	4	6	2	-3.0e+03
sparsqur	1000	14	13	15	1.8e-03	14	38	26	1.8e-03
spmsrtls	1000	33	20	9	4.6e-09	60	523	112	3.7e-01
srosenbr	1000	10	8	4	3.5e-09	10	24	9	3.5e-09
testquad	1000	15	9	4	1.9e-25	15	50	19	4.5e-25
tointgss	1000	3	2	1	1.0e+01	3	8	3	1.0e+01
tquartic	1000	4	4	6	3.4e-13	4	7	7	3.4e-13
woods	1000	14	11	5	2.0e+03	14	42	16	2.0e+03

Table 7.13: Incomplete Cholesky preconditioning using explicit matrix-vector products with M. Problems that could not be solved without preconditioning, $\delta_0 = 100$.

		Steihaug				phased-SSM			
Problem	n	Fe	Prods	Cpu	f	Fe	Prods	Cpu	f
curly10	1000	16	11	7	-1.0e+05	27	344	81	-1.0e+05
curly20	1000	16	12	11	-1.0e+05	15	243	70	-1.0e+05
curly30	1000	30	18	23	-1.0e+05	22	196	101	-1.0e+05
dixon3dq	1000	2	1	1	5.5e-28	2	2	1	5.5e-28
fletcbv2	1000	2	1	1	-5.0e-01	2	2	2	-5.0e-01
genhumps	1000	437	421	172	7.1e+01	—	—	—	—
morebv	1000	2	1	1	7.3e-13	2	2	1	7.3e-13
sbrybnd	1000	30	17	9	1.6e-09	19	65	23	3.7e-11
scurly10	1000	16	11	7	-1.0e+05	30	238	71	-1.0e+05
scurly20	1000	14	10	10	-1.0e+05	21	199	72	-1.0e+05
scurly30	1000	21	13	17	-1.0e+05	20	170	87	-1.0e+05
sinquad	1000	94	90	110	1.1e-06	94	497	267	1.1e-06
tridia	1000	6	4	2	1.5e-26	6	20	8	8.5e-27

Table 7.14: Function evaluations of direct and indirect methods. Incomplete Cholesky preconditioning. Problems A–E, $\delta_0 = 100$.

Problem	n	Steihaug	phased-SSM	M-S
arwhead	1000	6	6	6
bdqrtic	1000	10	10	10
broydn7d	1000	87	99	116
brybnd	1000	33	21	70
chainwoo	1000	86	123	114
cosine	1000	5	30	49
cragglvy	1000	14	14	14
curly10	1000	16	27	29
curly20	1000	16	15	15
curly30	1000	30	22	24
dixmaana	1500	7	10	10
dixmaanb	1500	23	45	48
dixmaanc	1500	23	55	71
dixmaand	1500	25	51	108
dixmaane	1500	41	63	151
dixmaanf	1500	24	74	254
dixmaang	1500	24	52	306
dixmaanh	1500	30	47	193
dixmaani	1500	33	120	152
dixmaanj	1500	31	19	424
dixmaank	1500	19	19	304
dixmaanl	1500	31	27	145
dixon3dq	1000	2	2	4
dqdrtic	1000	8	8	8
dqrtic	1000	37	37	37
edensch	1000	15	15	15
eg2	1000	4	4	49
engval1	1000	8	8	8
extrosnb	1000	22	22	22

Table 7.15: Function evaluations for direct and indirect methods. Incomplete Cholesky preconditioning. Problems F–Z, $\delta_0 = 100$.

Problem	n	Steihaug	phased-SSM	M-S
fletcbv2	1000	2	2	3
fminsrf2	1024	21	21	26
freuroth	1000	9	9	9
liarwhd	1000	15	15	15
morebv	1000	2	2	2
ncb20	1010	76	100	**
ncb20b	1000	27	13[d]	28
noncvxu2	1000	60	278	213
noncvxun	1000	52	285	272
nondia	1000	7	7	8
nondquar	1000	13	13	13
powellsg	1000	12	12	12
quartc	1000	37	37	37
sbrybnd	1000	30	19	42
scurly10	1000	16	30	35
scurly20	1000	14	21	21
scurly30	1000	21	20	33
schmvett	1000	4	4	4
sinquad	1000	94	94	13
sparsqur	1000	14	14	14
spmsrtls	1000	33	60	114
srosenbr	1000	10	10	10
testquad	1000	15	15	15
tointgss	1000	3	3	3
tquartic	1000	4	4	2
tridia	1000	6	6	6
woods	1000	14	14	14

Bibliography

[1] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Trans. Math. Softw.*, 21(1):123–160, 1995.

[2] R. H. Byrd, R. B. Schnabel, and G. A. Shultz. A trust region algorithm for nonlinearly constrained optimization. *SIAM J. Numer. Anal.*, 24:1152–1170, 1987.

[3] R. H. Byrd, R. B. Schnabel, and G. A. Shultz. Approximate solution of the trust region problem by minimization over two-dimensional subspaces. *Math. Programming*, 40(3, (Ser. A)):247–263, 1988.

[4] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

[5] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19(2):400–408, 1982.

[6] J. E. Dennis Jr. and H. H. Mei. Two new unconstrained optimization algorithms which use function and gradient values. *J. Optim. Theory Appl.*, 28:453–482, 1979.

[7] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[8] A. Forsgren and P. E. Gill. Primal-dual interior methods for nonconvex nonlinear programming. *SIAM J. Optim.*, 8:1132–1152, 1998.

[9] A. Forsgren, P. E. Gill, and W. Murray. Computing modified Newton directions using a partial Cholesky factorization. *SIAM J. Sci. Comput.*, 16:139–150, 1995.

[10] D. M. Gay. Computing optimal locally constrained steps. *SIAM J. Sci. Statist. Comput.*, 2(2):186–197, 1981.

[11] E. M. Gertz. *Combination Trust-Region Line-Search Methods for Unconstrained Optimization.* PhD thesis, Department of Mathematics, University of California, San Diego, 1999.

[12] E. M. Gertz and P. E. Gill. A primal-dual trust-region algorithm for nonlinear programming. *Math. Program., Ser. B*, 100:49–94, 2004.

[13] P. E. Gill and W. Murray. Newton-type methods for unconstrained and linearly constrained optimization. *Math. Program.*, 7:311–350, 1974.

[14] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, London and New York, 1981.

[15] D. Goldfarb. Curvilinear path steplength algorithms for minimization which use directions of negative curvature. *Math. Program.*, 18:31–40, 1980.

[16] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, Maryland, third edition, 1996.

[17] N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic programming problems arising in optimization. *SIAM J. Sci. Comput.*, 23(4):1376–1395 (electronic), 2001.

[18] N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM J. Optim.*, 9(2):504–525, 1999.

[19] J. Greenstadt. On the relative efficiencies of gradient methods. *Math. Comp.*, 21:360–367, 1967.

[20] J. D. Griffin. *Interior-point methods for large-scale nonconvex optimization.* PhD thesis, Department of Mathematics, University of California, San Diego, March 2005.

[21] W. W. Hager. Minimizing a quadratic over a sphere. *SIAM J. Optim.*, 12(1):188–208 (electronic), 2001.

[22] W. W. Hager and S. Park. Global convergence of SSM for minimizing a quadratic over a sphere. *Math. Comp.*, 74(74):1413–1423, 2004.

[23] M. T. Jones and P. E. Plassmann. An improved incomplete Cholesky factorization. *ACM Trans. Math. Softw.*, 21:5–17, 1995.

[24] D. S. Kershaw. The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *J. Comput. Phys.*, 26:43–65, 1978.

[25] C.-J. Lin and J. J. Moré. Incomplete Cholesky factorizations with limited memory. *SIAM J. Sci. Comput.*, 21:24–45, 1999.

[26] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45:503–528, 1989.

[27] D. Luenberger. *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, New York, 1973.

[28] G. P. McCormick. A modification of Armijo's step-size rule for negative curvature. *Math. Program.*, 13:111–115, 1977.

[29] J. L. Morales. A numerical study of limited memory BFGS methods. *Appl. Math. Lett.*, 15(4):481–487, 2002.

[30] J. L. Morales and J. Nocedal. Automatic preconditioning by limited memory quasi-Newton updating. *SIAM J. Optim.*, 10(4):1079–1096 (electronic), 2000.

[31] J. J. Moré and D. C. Sorensen. On the use of directions of negative curvature in a modified Newton method. *Math. Program.*, 16:1–20, 1979.

[32] J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM J. Sci. and Statist. Comput.*, 4:553–572, 1983.

[33] K. G. Murty and S. N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Math. Program.*, 39:117–129, 1987.

[34] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Math. Comput.*, 35:773–782, 1980.

[35] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[36] J. Nocedal and Y.-x. Yuan. Combining trust region and line search techniques. In *Advances in Nonlinear Programming (Beijing, 1996)*, volume 14 of *Appl. Optim.*, pages 153–175. Kluwer Acad. Publ., Dordrecht, 1998.

[37] C. C. Paige. *The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices*. PhD thesis, University of London, 1971.

[38] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[39] P. M. Pardalos and G. Schnitger. Checking local optimality in constrained quadratic programming is NP-hard. *Operations Research Letters*, 7:33–35, 1988.

[40] Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. In I. S. Duff, editor, *Sparse Matrices and Their Uses*, pages 57–88, London and New York, 1981. Academic Press.

[41] M. J. D. Powell. A hybrid method for nonlinear equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*, pages 87–114. Gordon and Breach, 1970.

[42] M. J. D. Powell. A new algorithm for unconstrained optimization. In *Nonlinear Programming (Proc. Sympos., Univ. of Wisconsin, Madison, Wis., 1970)*, pages 31–65. Academic Press, New York, 1970.

[43] M. J. D. Powell. Convergence properties of a class of minimization algorithms. In *Nonlinear Programming, 2 (Proc. Sympos. Special Interest Group on Math. Programming, Univ. Wisconsin, Madison, Wis., 1974)*, pages 1–27. Academic Press, New York, 1974.

[44] Y. Saad. ILUT: A dual threshold incomplete LU factorization. *Numer. Linear Algebra Appl.*, 4:387–402, 1994.

[45] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, Boston, MA, 1996.

[46] R. B. Schnabel and E. Eskow. A new modified Cholesky factorization. *SIAM J. Sci. and Statist. Comput.*, 11:1136–1158, 1990.

[47] G. A. Shultz, R. B. Schnabel, and R. H. Byrd. A family of trust-region based algorithms for unconstrained minimization with strong global convergence properties. *SIAM J. Numer. Anal.*, 22:47–67, 1985.

[48] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.*, 20:626–637, 1983.

[49] S. W. Thomas. *Sequential estimation techniques for quasi-Newton algorithms*. PhD thesis, Cornell University, 1975.

[50] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, 1965.