SOL--92-4

DE93 002528

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-4022

Large-Scale Sequential Quadratic Programming Algorithms

by
Samuel K. Eldersveld

TECHNICAL REPORT SOL 92-4

September 1992

DISCLAIMER

Research and reproduction of this report were partially supported by the National Science Foundation Grant DDM-9204208, the Department of Energy Grant DE-FG03-92ER25117, Office of Naval Research Grant N00014-90-J-1242, and an IBM Graduate Technical Fellowship.

Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do NOT necessarily reflect the views of the above sponsors.

Also listed as Operations Research Department Technical Report 92-11.

LARGE-SCALE SEQUENTIAL QUADRATIC PROGRAMMING ALGORITHMS

Samuel Keith Eldersveld, Ph.D.
Stanford University, 1992

Abstract

The problem addressed is the general nonlinear programming problem: finding a local minimizer for a nonlinear function subject to a mixture of nonlinear equality and inequality constraints. The methods studied are in the class of sequential quadratic programming (SQP) algorithms, which have previously proved successful for problems of moderate size. Our goal is to devise an SQP algorithm that is applicable to large-scale optimization problems, using sparse data structures and storing less curvature information but maintaining the property of superlinear convergence. The main features are:

1. *The use of a quasi-Newton approximation to the reduced Hessian of the Lagrangian function.* Only an estimate of the reduced Hessian matrix is required by our algorithm. The impact of not having available the full Hessian approximation is studied and alternative estimates are constructed.

2. *The use of a transformation matrix Q.* This allows the QP gradient to be computed easily when only the reduced Hessian approximation is maintained.

3. *The use of a reduced-gradient form of the basis for the null space of the working set.* This choice of basis is more practical than an orthogonal null-space basis for large-scale problems. The continuity condition for this choice is proven.

4. *The use of incomplete solutions of quadratic programming subproblems.* Certain iterates generated by an active-set method for the QP subproblem are used in place of the QP minimizer to define the search direction for the nonlinear problem.

An implementation of the new algorithm has been obtained by modifying the code MINOS. Results and comparisons with MINOS and NPSOL are given for the new algorithm on a set of 92 test problems.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The problem addressed in this report is that of finding a local minimizer for a general nonlinear function $F(x)$ subject to a set of nonlinear constraints $c(x) \geq 0$. This is the general *nonlinear programming problem* (NLP):

$$\boxed{\begin{array}{ll} \underset{x \in \Re^n}{\text{minimize}} & F(x) \\ \text{s.t.} & c(x) \geq 0, \end{array}} \qquad \text{NLP}$$

where $F : \Re^n \rightarrow \Re$ and $c : \Re^n \rightarrow \Re^m$.

There are a number of mathematically equivalent forms of NLP. The relevance of the precise form of the problem to the efficiency of specific algorithms is discussed later. We assume that the objective function $F(x)$ and the nonlinear constraint functions $c_i(x)$, $i = 1, \ldots, m$, are twice continuously differentiable.

A wide variety of algorithms exist for solving NLP, none of which can be considered preferable for all problems. For a general discussion of NLP the reader is referred to Fletcher [Fle87] and Gill *et al.* [GMW81]. For a recent survey of methods see [GMSW89].

The particular focus of this report is the case of large, sparse NLP. By *large and sparse* we mean that we are concerned with the instances of problem NLP in which n is large, the $m \times n$ Jacobian of the nonlinear constraints is sparse, and usually $n - m \ll n$. Although many algorithms have been proposed to solve NLP, few have been adapted for the large sparse case. A notable exception is the Lagrangian method of Murtagh and Saunders [MurS82]. This algorithm has been implemented as the mathematical programming system MINOS [MurS87].

There is a concensus that the best methods for solving NLP when n is small are so-called *sequential quadratic programming* (SQP) methods. Such methods make use of local curvature information to construct a quadratic programming (QP) model or *subproblem* of NLP. A local minimizer is found by solving a sequence of these QP subproblems. The rate of convergence of SQP methods is usually superlinear under certain assumptions on the closeness of the quadratic approximation. We are concerned with developing large-scale SQP algorithms that are globally convergent to a local minimizer of NLP, and have a fast rate of convergence.

All methods for solving NLP are iterative. In the case when n is small the efficiency of an algorithm is usually measured in terms of the number of iterations, or possibly the

number of function evaluations, required to attain some specified approximation to the solution. In the large sparse case we also need to be concerned with the effort required to compute the iterates. It is sometimes worthwhile to modify the definition of the iterative sequence in order to compute the iterates more efficiently. We may then take more iterations, but the savings in effort to compute the iterates is sufficient compensation.

1.1. Notation and definitions

Our notation in this report follows that used in [GMSW86b] and [Pri89]. In addition to F, x and c defined above we shall use the following definitions and conventions:

- Subscripts on a function denote the value of the function evaluated at the variable with the same subscript (for example, $F_k = F(x_k)$).
- Bars on functions or variables or data will often be used to denote updated quantities (for example, when x_{k+1} corresponds to the new iterate, the new value of the constraints is denoted $\bar{c} = c(x_{k+1})$).
- λ is the vector of Lagrange multiplier estimates for c.
- $L(x, \lambda) = F(x) - \lambda^T c(x)$ is the Lagrangian function.
- $g(x) = \nabla F(x)$ is the $n \times 1$ gradient vector for F.
- $J(x)$ is the $m \times n$ matrix of gradients for the constraint functions (the Jacobian). Then $J_{ij} = \partial c_i / \partial x_j$.
- $A(x) = \begin{pmatrix} J(x) & -I \end{pmatrix}$ is the constraint matrix for QP subproblems. Often we will refer to a partition of A as in $A = \begin{pmatrix} B & S & N \end{pmatrix}$, where B is nonsingular.
- We will often refer to a partitioned vector as in

$$p = (p_B, \; p_S, \; p_N) \equiv \begin{pmatrix} p_B \\ p_S \\ p_N \end{pmatrix}.$$

  In this notation the commas denote that the partitioned vector has a column dimension of one.
- Z denotes a basis for the null space of a matrix of the form $\hat{A} = \begin{pmatrix} B & S & N \\ & & I \end{pmatrix}$.
- Y denotes a matrix such that $(Z \;\; Y)$ is nonsingular.
- Q is a transformation matrix of the form $Q(x) = \begin{pmatrix} Z(x) & Y(x) \end{pmatrix}$.
- $G(x)$ denotes the Hessian of $F(x)$. Then $G_{ij} = \partial^2 F / \partial x_i \partial x_j$.
- $G_i(x)$ denotes the Hessian of $c_i(x)$.

- $W(x,\lambda) \equiv G(x) - \sum \lambda_i G_i(x)$ is the Hessian of the Lagrangian function.
- H is an $n \times n$ approximation to $W(x,\lambda)$.
- s is the m-vector of slack variables for constraints c such that $c(x) - s = 0$.
- μ is the m-vector of QP multipliers (for A).
- $\xi = \mu - \lambda$ is the search direction for Lagrange multiplier estimates.
- p is the search direction for x.
- $q = (c - s) + Ap$ is the search direction for slack variables.
- x^* is a solution of the constrained optimization problem NLP.
- λ^* is the vector of Lagrange multipliers at x^*.

The above notation will be defined again when the terms involved are first introduced in the text. This list is intended to be a convenient reference to save searching for definitions in the text.

1.2. Optimality conditions for NLP

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

$$\|x - x^*\| \leq \delta \quad \text{and} \quad c(x) \geq 0. \tag{1.2.1}$$

If $F(x) > F(x^*)$ for all $x \neq x^*$ satisfying (1.2.1), x^* is defined as a *strong local minimizer.*

The above definition of a local minimizer is of little aid in determining x^* or verifying that some given point is indeed a minimizer. Most optimization methods (and all of the algorithms described in this report) determine x^* by seeking points that satisfy verifiable optimality conditions on x^*. These conditions are characterized by the first and second derivatives of the Lagrangian function (see for example [FiaM68]). Assuming that the Jacobian of *active* constraints at a solution to NLP has full rank, we now give optimality conditions for NLP.

Necessary conditions for x^* to be a local minimizer are that there exist multipliers λ^* such that (x^*, λ^*) satisfy the *Karush-Kuhn-Tucker (KKT) second-order conditions:*

$$c(x^*) \geq 0; \tag{1.2.2}$$

$$\nabla F(x^*) = A^{*T}\lambda^*; \tag{1.2.3}$$

$$\lambda^* \geq 0; \tag{1.2.4}$$

$$Z^{*T}W^*Z^* \quad \text{is positive semi-definite,} \tag{1.2.5}$$

where $W^* \equiv W(x^*, \lambda^*)$ is the Hessian of the Lagrangian, A^* is the Jacobian of the active constraints at x^* and Z^* is a basis for the nullspace of A^*. *Sufficient conditions* for (x^*, λ^*)

to be a local minimizer are that there exist multipliers λ^* such that (x^*, λ^*) satisfy the KKT conditions (1.2.2)–(1.2.3) and

$$\lambda^* > 0; \tag{1.2.6}$$

$$Z^{*T} W^* Z^* \quad \text{is positive definite.} \tag{1.2.7}$$

A point x^* with multipliers λ^* satisfying only KKT conditions (1.2.2)–(1.2.4) will be referred to as a *first-order KKT point*, while a point satisfying only KKT conditions (1.2.2)–(1.2.3) will be referred to as a constrained *stationary point*. It should be noted that the algorithms developed for NLP in this report do not require the provision of analytical second derivatives. As a result, the algorithms presented in Section 2 only guarantee that a computed solution x^* is a first-order KKT point. Despite this theoretical restriction on *all* algorithms that do not evaluate second derivatives, it is important that such algorithms still attempt to seek a minimizer and not simply a first-order KKT point.

Consider the case of unconstrained minimization. We could simply generate iterates that reduce $g^T g$. In so doing we would converge to a first-order KKT point that could be either a maximizer, a saddle-point or a minimizer. If on the other hand we generate iterates that reduce $F(x)$, we could still only be assured of finding a stationary point (i.e. $g(x) = 0$) but it is more likely to be a minimizer. The generalization of this idea for a constrained problem involves choosing a suitable merit function (see Section 1.3).

It is also important to note that without a strong assumption on the form of the problem it is not possible to distinguish between local and global solutions.

1.3. SQP algorithms

It is not possible in general to determine an optimal solution to NLP in a finite number of iterations, except in special cases such as linear and quadratic programs. SQP algorithms construct a sequence $\{x_k\}_{k=0}^{\infty}$ whose limit points are KKT points. Given a point x_k, we may obtain a new point x_{k+1} by solving a mathematical programming problem whose solution x_{k+1} approximates x^*. One method of approximating the optimal step $(x^* - x_k)$ is to find a minimizer of a local approximation to the problem. For SQP methods the model problem takes the form of a QP subproblem in which a quadratic approximation is made to the Lagrangian and linear approximations are made to the nonlinear constraints. For each x_k in the sequence, the k-th QP subproblem may be stated as follows:

$$\boxed{\begin{array}{ll} \underset{p \in \Re^n}{\text{minimize}} & \frac{1}{2} p^T H_k p + g_k^T p \\ \text{s.t.} & A_k p \geq -c_k, \end{array}} \qquad \text{QP}$$

where $g_k = \nabla F(x_k)$ and H_k is an approximation to $W(x_k, \lambda_k)$, the Hessian of the Lagrangian. The solution and Lagrange multipliers are denoted by the pair (p_k, μ_k).

Ideally, we hope to accept $x_k + p_k$ as the next iterate, especially near the solution. However, the QP subproblem is defined only by *local* information (i.e. at the current iterate); the solution of the model problem may be a poor approximation of the solution to NLP when the current iterate is not close to x^*. Hence, we regard the solution to QP as a *search direction* p_k that will be a descent direction for some merit function, as discussed next.

1.3.1. Merit functions

In contrast to algorithms for unconstrained and linearly constrained problems, it is not practical in general to generate a sequence $\{x_k\}_{k=0}^{\infty}$ such that all points x_k are feasible. Given two feasible points we can determine which is best by comparing $F(x)$ evaluated at the two points. Once the points are allowed to be infeasible it becomes problematical to determine which is best. To illustrate this difficulty, consider Figure 1. Some of the five points $x_1, \ldots, x_5$ are infeasible with respect to the single inequality constraint $c(x) \geq 0$ and it is not clear which offers the best approximation to x^*. Because the sequence $\{x_k\}_{k=0}^{\infty}$

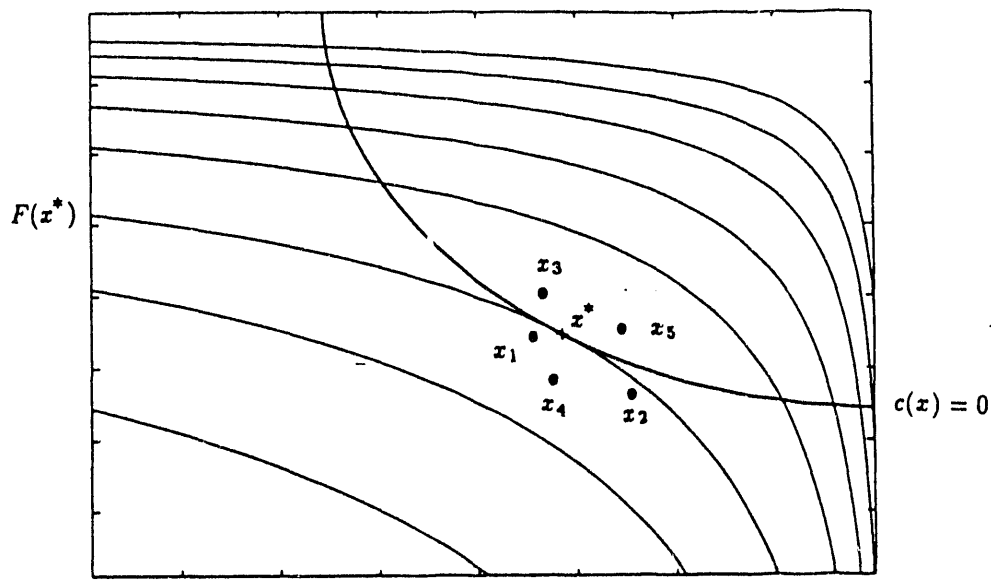

Figure 1. Which is the best point?

in our algorithms may contain infeasible points it is necessary to order the iterates in some way other than simply noting which iterate has the lowest function value. Many algorithms for NLP do this by means of a *merit function*, which is used to determine a step α_k along a search direction p_k. The requirement is that the new point $x_k + \alpha_k p_k$ reduce the merit function by a sufficient amount. The points x_k therefore form an "improving" sequence.

One possible choice for a merit function is the *augmented Lagrangian merit function* due to Rockafellar [Roc73]:

$$M(x, \lambda, \rho) = F(x) - \lambda^T \hat{c} + \tfrac{1}{2}\rho \|\hat{c}\|^2, \tag{1.3.1}$$

where $\rho > 0$ is a penalty parameter and the vector $\hat{c}$ is defined as

$$\hat{c}_i(x) = \begin{cases} c_i(x) & \text{if } c_i(x) - \lambda_i/\rho \leq 0, \\ \lambda_i/\rho & \text{otherwise.} \end{cases} \tag{1.3.2}$$

Note that this merit function assigns a positive penalty for increasing constraint violations. To illustrate the use of (1.3.1), consider the following example:

$$\begin{array}{ll} \underset{x \in \Re^2}{\text{minimize}} & F(x) = x_1 x_2^2 \\ \text{s.t.} & c(x) = 2 - x_1^2 - x_2^2 \geq 0. \end{array}$$

The optimal solution is $x^* = (-0.81650, -1.1547)$ with optimal Lagrange multiplier $\lambda^* = 0.81650$. Figure 1 depicts the contours of $F(x)$ with $c(x) = 0$ superimposed for this example.

Figure 2 depicts the contours of the augmented Lagrangian merit function $M(x, \lambda^*, \rho)$ for the same problem as in Figure 1, using $\lambda^* = 0.81650$ and $\rho = 0.1$ and $\rho = 1$. Figure 3

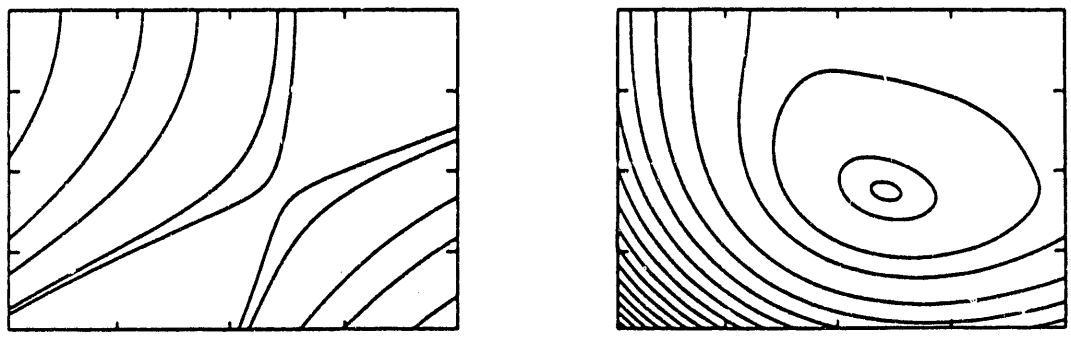

Figure 2. Contours of $M(x, \lambda^*, \rho)$ for $\rho = 0.1$ and $\rho = 1$

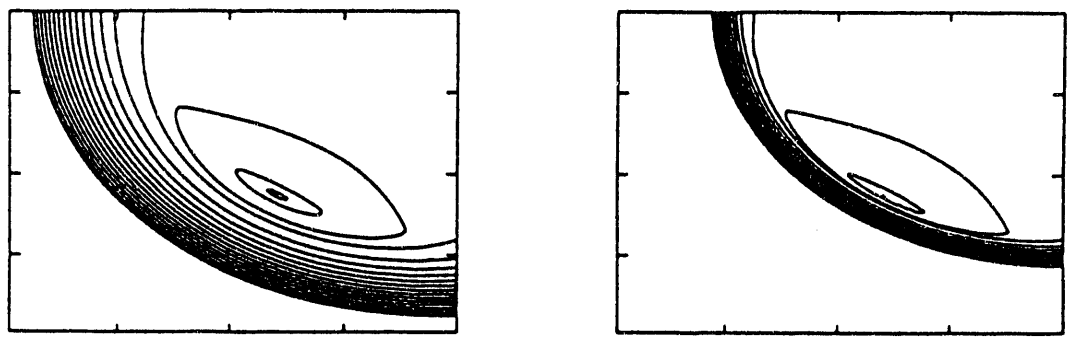

Figure 3. Contours of $M(x, \lambda^*, \rho)$ for $\rho = 10$ and $\rho = 100$

shows the contours of the augmented Lagrangian function with $\rho = 10$ and $\rho = 100$. Figures 2 and 3 demonstrate the complications that may arise in choosing the penalty parameter ρ. If ρ is set too small as in the left part of Figure 2, the merit function may become unbounded below. If ρ is set too large as in the right part in Figure 3, the merit function may become ill-conditioned.

The steps required by an SQP method are summarized in Algorithm 1.3.1 below. Each step will be discussed in detail in Section 2.

Algorithm 1.3.1. (Model SQP algorithm)

Start with estimates x_0 and H_0 of a solution and the Hessian of the Lagrangian at x_0.

while not *converged* **do**

Set up and solve a QP subproblem to obtain a search direction p_k and Lagrange multipliers μ_k.

Compute a steplength α to reduce some merit function.

Update x according to $x_{k+1} \leftarrow x_k + \alpha p_k$.

Evaluate constraints c and gradients g and J at x_{k+1}.

Update (or form) H_{k+1}, the QP Hessian to be used in the next subproblem.

end do

Figure 4. Model SQP algorithm

1.4. Historical background

SQP methods for NLP were first introduced in 1963. Here we outline the development of SQP methods since then and emphasize some of the key ideas. For a more detailed discussion of the history of SQP methods, see [GMW81, Pow83, GMSW88].

1.4.1. Early SQP methods

Wilson [Wil63] is believed to have been the first to propose an SQP algorithm. In his doctoral dissertation he proposed solving *convex* nonlinear programming problems using a sequence of inequality constrained QP's in which the QP objective was defined using the exact Hessian of the Lagrangian. Successive NLP iterates were obtained as $\bar{x} = x + p$ (i.e. without the benefit of a merit function and linesearch).

In 1969, Murray [Mur69] proposed an SQP algorithm employing a quasi-Newton approximation to the Hessian of the Lagrangian. He also introduced the important concept of using the QP solution to define a search direction and choosing the next iterate by taking a step to reduce a merit function. The use of a quasi-Newton approximation and a linesearch enabled Wilson's convexity assumption to be relaxed.

Notable developments in SQP algorithms occurred throughout the 1970's. Biggs [Big72] proposed an algorithm using an equality-constrained subproblem, and a term for the multiplier estimate was added to the constraints. Han [Han76] established sufficient conditions for local and superlinear convergence of an SQP algorithm under the assumption that the Hessian of the Lagrangian is positive definite on the whole space. Powell [Pow78a] used the framework of Han to provide a proof of superlinear convergence under additional assumptions on how well the Hessian of the Lagrangian is approximated.

1.4.2. Merit functions

Much research has been done on the choice of merit function for SQP iterates. Murray's pioneering approach used an ℓ_2 merit function [Mur69]. Since then the focus has been not on the use of the merit function but on its form. Han [Han76] and Powell [Pow78b] in their SQP algorithms proposed the use of the ℓ_1 merit function (also known as an exact penalty function),

$$M(x,\rho) = F(x) + \rho\|\hat{c}\|_1, \tag{1.4.1}$$

where ρ is a nonnegative penalty parameter and $\hat{c}$ contains only the values of constraints $c(x)$ considered to be violated at x. A virtue of the ℓ_1 merit function over the ℓ_2 merit function is that there exists a bounded value of ρ for which x^* is a minimizer of $M(x,\rho)$. This latter property makes convergence proofs relatively simple. However, the ℓ_1 merit function is nonsmooth across constraint violations. Maratos [Mar78] in his doctoral dissertation demonstrated that imposing linesearch conditions using this merit function could impede the superlinear rate of convergence. To overcome this deficiency SQP methods based on the ℓ_1 merit function must depart from a pure SQP strategy.

As an alternative, consider the augmented Lagrangian merit function

$$M(x,\lambda,\rho) = F(x) - \lambda^T\hat{c} + \tfrac{1}{2}\rho\hat{c}^T\hat{c}, \tag{1.4.2}$$

where $\hat{c}$ is defined in (1.3.2). Fletcher [Fle70] first proposed the use of this merit function (but not in the context of an SQP algorithm). In contrast to (1.4.1), (1.4.2) is a smooth function. However, it requires estimates of the Lagrange multipliers λ. In general, x^* is a minimizer of $M(x,\lambda,\rho)$ only if $\lambda = \lambda^*$. This requirement makes convergence proofs for SQP methods using (1.4.2) somewhat more difficult than proofs using (1.4.1). Both Wright [Wri76] and Schittkowski [Sch82] proposed SQP algorithms based on this merit function.

Consider next an augmented Lagrangian merit function defined in terms of slack variables s as well as multipliers λ and variables x:

$$M(x,s,\lambda,\rho) = F(x) - \lambda^T(c-s) + \tfrac{1}{2}\rho(c-s)^T(c-s). \tag{1.4.3}$$

It is no longer necessary to restrict the terms involving $c(x)$ in (1.4.3) to some subset of the constraints. The merit function has the same continuity properties as $F(x)$ and $c(x)$. Gill *et al.* [GMSW86b] proposed this merit function in the context of an SQP method. They showed under certain assumptions that $x_k \to x^*$ and $\lambda_k \to \lambda^*$. In his doctoral dissertation, Prieto [Pri89] showed that a finite value of ρ suffices. The steplength α is determined by performing a search in the space of x, s and λ. This merit function has been implemented in the nonlinear programming code NPSOL [GMSW86a].

1.4.3. Use of the reduced Hessian

For moderate-sized problems, the most successful SQP algorithms to date have used dense approximations to W, the Hessian of the Lagrangian. A key concept for large-scale optimization is the use of an approximation to the *reduced* Hessian Z^TWZ. This is of prime computational importance for the following reasons:

- A dense approximation to all of W may require excessive storage.

- Computation of exact second derivatives may not be possible or may be too expensive.
- Even if W can be evaluated cheaply, computation of the matrix product Z^TWZ from Z and W may be too expensive (unless Z has very few columns or some special structure).

Gill and Murray [GilM73, GilM74, GilM77] are credited with the first use of reduced Hessian approximations for linearly constrained problems. Murtagh and Saunders in [MurS78, MurS87] showed how to apply this approach to the large-scale case. Wright [Wri76] and Murray and Wright [MurW78] proposed the use of a quasi-Newton approximation to the reduced Hessian for nonlinearly constrained optimization. Coleman and Conn [ColC84] analyzed an SQP method that approximated the reduced Hessian and showed that the method when applied to *equality-constrained* problems converges 2-step superlinearly. Nocedal and Overton [NocO85], Coleman and Fenyes [ColF88], and Gurwitz and Overton [GurO89] have all proposed algorithms in which approximations are made to either Z^TWZ or Z^TW.

1.4.4. Active-set methods

Although interior-point/barrier methods could be used within an SQP method, we shall restrict our interest to the solution of the QP subproblems using active-set methods. This does not preclude the use of barrier methods at the outer level of the SQP algorithm. That is, inequality constraints could be removed by a barrier transformation and the algorithm we propose used to solve the resulting barrier subproblem.

For an overview of active-set methods, see Gill *et al.* [GMW81] or Fletcher [Fle87].

1.4.5. Early termination of subproblems

As problem size grows, the number of iterations required by an active-set method to solve a QP subproblem to optimality may become large. In a large-scale SQP implementation it is therefore desirable to impose a limit on the number of QP iterations allowed to solve the subproblem.

Murray [Mur69] is credited as the first to suggest this *early-termination* approach. Dembo and Tulowitzki [DemT85] defined an early-termination rule for the QP subproblem, based on the norm of the reduced gradient Z^Tg in the subproblem. Gurwitz and Overton [GurO89] presented an implementation of an early-termination algorithm in which a subproblem is terminated at the first stationary point (i.e. $Z^T(g + Hp) = 0$). In the work of Prieto [Pri89], global convergence was proved for an SQP algorithm in which the search direction is defined from information available at any stationary point encountered in the solution of the QP subproblem. Prieto also proved that use of a reduced Hessian approximation in this context provides a 2-step superlinear rate of convergence.

1.4.6. Large-scale SQP

Nickle and Tolle [NicT89] have described a sparse SQP approach in which they maintain an approximation to W, the full Hessian of the Lagrangian. They sacrifice satisfying the

quasi-Newton condition (see Section 4.2.2) in order to define an H with the same sparsity pattern as W.

1.5. Contents and subsequent Chapters

In Section 2 a prototype SQP algorithm for solving NLP is presented. Each subproblem uses an approximation to the reduced Hessian of the Lagrangian. A discussion of active-set methods for QP subproblems is also given. In Section 3, important computational building blocks are developed in order that the large-scale QP subproblems arising in the SQP method may be solved efficiently. Section 4 discusses quasi-Newton updates for the reduced Hessian of the Lagrangian and gives computational details. Section 5 presents computational results for both small and large (i.e. dense and sparse) test problems from a variety of applications.

Chapter 2

A Prototype SQP Algorithm

In this chapter we define the main theoretical tools for solving large-scale nonlinear programs and present an algorithm for solving NLP. The algorithm allows the use of incomplete solutions from QP subproblems.

2.1. Large-scale NLP

Many SQP methods have been proposed for solving NLP. Most of them perform algebraic operations that are appropriate for dense problems, but are not practical for large and sparse ones. For example, the storage required by dense methods may become excessive when there are many variables. This section gives a standard form for large-scale NLP and the optimality conditions modified for this form.

2.1.1. The form of the nonlinear problem

In methods for small or dense nonlinear programming problems both nonlinear equality and inequality constraints (i.e. $c_E(x) = 0$ and $c_I(x) \geq 0$) are usually allowed. In large-scale optimization the precise form of the problem is crucial, and it is often computationally convenient to assume that the problem is in the so-called *nonlinear programming standard form* (NLPSF):

$$\begin{array}{lll} \underset{x,s}{\text{minimize}} & F(x) & \\ \text{s.t.} & c(x) - s = 0 & \\ \text{and} & l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u. & \end{array} \qquad \text{NLPSF}$$

Each constraint $c_i(x)$ is associated with a slack variable s_i with upper and lower bounds on its value as shown. The bounds on each slack determine whether the associated constraint is an equality or inequality. For example, if $c_i(x)$ corresponds to an equality constraint, the bounds for slack s_i are zero.

The definition of the set of constraints considered to be binding at a point must be modified for the standard form. At any point x, the set of active constraints will consist of all functional constraints $c(x) - s = 0$, as well as the set of active bounds at (x, s).

It may appear that increasing the number of variables for the problem in this way is computationally disadvantageous. For some methods of optimization this is true. However, it will be seen that this is not the case for the methods presented here.

It has been demonstrated since the earliest simplex codes that there are advantages in using a standard form involving slacks. In particular, the standard form only requires access to columns of the Jacobian (as opposed to columns and rows). Also, when the working-set basis matrix B is factorized (see Section 2.4), the columns associated with slacks introduce no extra nonzero elements (i.e. fill-in) in the LU factors of B. These same advantages were retained for sparse NLP by Murtagh and Saunders [MurS78, MurS82].

2.1.2. Optimality conditions for large-scale NLP

In the discussion of optimality conditions it will be convenient to assume that the slack variables s are included in the definition of the variables x. That is we augment the variables x to include the slacks s:

$$x \equiv \begin{pmatrix} x \\ s \end{pmatrix},$$

so that for the NLP in standard form, $x \in \Re^{n+m}$. We also modify the nonlinear constraints to include the slack variables. The NLPSF may then be written:

$\underset{x \in \Re^{n+m}}{\text{minimize}}$	$F(x)$	NLPSF′
s.t.	$c(x) = 0$	
and	$l \leq x \leq u.$	

As discussed in Section 1, the *Karush-Kuhn-Tucker* (KKT) conditions describe local solutions to NLP. These are characterized by conditions on the first and second derivatives of the Lagrangian. The NLPSF′ now has bound constraints on x (including slacks). This will slightly change the conditions while making the first-order conditions easier to identify. Define $g^* \equiv \nabla F(x^*)$ and A^* to be the Jacobian of $c(x^*)$ (that is, J is a matrix of row vectors each corresponding to $c_i(x^*)^T$ for $i = 1, \ldots, t$, where t is the number of active functional constraints at x^*). Let Z^* be a basis for the null space of A^* (so that $A^*Z^* = 0$). The KKT *necessary* conditions for (x^*, λ^*) to be a *first-order KKT pair* for NLP are

$$\begin{aligned}
c(x^*) &= 0, && (2.1.1)\\
x^* &\geq l, && (2.1.2)\\
x^* &\leq u, && (2.1.3)\\
A^{*T}\lambda^* - \eta + \sigma &= g^*, && (2.1.4)\\
\eta^T(x^* - l) &= 0, && (2.1.5)\\
\sigma^T(u - x^*) &= 0, && (2.1.6)\\
\eta &\geq 0, && (2.1.7)\\
\sigma &\geq 0. && (2.1.8)
\end{aligned}$$

Let a_j^* be the j-th column of A^*. Since the complementarity conditions (2.1.5) and (2.1.6) enforce $\eta_j = \sigma_j = 0$ when x_j^* is not equal to either of its bounds, we can write the

optimality conditions (2.1.4)–(2.1.8) in what is for us a computationally more useful form involving Z^* and the explicit values of σ and η:

$$Z^{*T} g^* = 0, \tag{2.1.9}$$

$$(u_j - x_j^*)(g_j^* - a_j^{*T}\lambda^*) \geq 0 \text{ for } j = 1, \ldots, n, \tag{2.1.10}$$

$$(x_j^* - l_j)(g_j^* - a_j^{*T}\lambda^*) \leq 0 \text{ for } j = 1, \ldots, n. \tag{2.1.11}$$

Optimal variables x_j^* lying between their bounds will have the corresponding Lagrange multipliers or *reduced costs* equal to zero:

$$\sigma_j = \eta_j = g_j^* - a_j^{*T}\lambda^* = 0. \tag{2.1.12}$$

Optimality of the Lagrange multipliers requires nonnegative (nonpositive) reduced costs for variables on lower (upper) bounds.

2.2. Expansion of the model algorithm

Section 1 presented a model algorithm. The optimality procedures within Algorithm 1.3.1 consisted of (1) solving a QP subproblem, (2) performing a linesearch in conjunction with a merit function, and (3) updating nonlinear quantities including the Hessian approximation to be used in the next QP subproblem. A discussion of each of these steps follows.

2.2.1. Subproblem definition

A number of methods have been proposed for solving large-scale NLP. Two mentioned in this section are SQP methods and a Lagrangian method [MurS82]. Although not an SQP method, the Lagrangian method is mentioned here because it offers one of the few efficient methods currently available for large-scale problems. Also, the numerical results of Section 5 compare the SQP algorithms presented in this report with the Lagrangian method implemented in the form of MINOS. Both approaches use linearly constrained subproblems. The Lagrangian method uses an augmented Lagrangian as the subproblem objective, while SQP methods use a quadratic approximation to the Lagrangian. Both methods involve the use of *major* and *minor* iterations. A major iteration is defined to be the set of steps required to form and solve a single subproblem, while a minor iteration constitutes a single iteration within a subproblem.

2.2.2. Linearly constrained subproblems

We may obtain a linearly constrained subproblem from NLP by replacing $c(x)$ with a linear approximation from its Taylor series expansion:

$$c(x_k + p) \approx c(x_k) + J(x_k)p, \tag{2.2.1}$$

where $J(x_k)$. is the Jacobian of c evaluated at x_k. Define $J_E(x_k)$ and $J_I(x_k)$ to be the Jacobian of the nonlinear equality and inequality constraints respectively. The linearized constraints are then

$$J_E p = -c_E(x_k) \quad \text{and} \tag{2.2.2}$$

$$J_I p \geq -c_I(x_k). \tag{2.2.3}$$

As described earlier, it is computationally convenient to convert the linearly constrained subproblem so that all general constraints are equalities:

$$\begin{pmatrix} J_k & -I \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} = -(c - s). \tag{2.2.4}$$

The only inequalities are then simple bounds on the variables:

$$l - x_k \leq \begin{pmatrix} p \\ q \end{pmatrix} \leq u - x_k. \tag{2.2.5}$$

The linearly constrained subproblem (LCS) can now be written

$$\boxed{\begin{array}{ll} \underset{p,q}{\text{minimize}} & \tilde{F}(x_k + p) \\ \text{s.t.} & J_k p - q = -(c_k - s_k), \\ & l_k \leq \begin{pmatrix} p \\ q \end{pmatrix} \leq u_k, \end{array}} \quad \text{LCS}$$

where $l_k = l - x_k$ and $u_k = u - x_k$.

The subproblem objective in LCS may have many forms. For example in the Lagrangian algorithm of Murtagh and Saunders [MurS82], $\tilde{F}$ takes the form of an augmented Lagrangian:

$$\tilde{F}(x, \lambda, \rho) = F(x) - \lambda^T(\tilde{c} - A_k p) + \tfrac{1}{2}\rho\|\tilde{c} - A_k p\|^2, \tag{2.2.6}$$

where $\tilde{c} \equiv c(x) - c_k$ and $p = x - x_k$. Note that this subproblem objective requires evaluations of both $F(x)$ and $c(x)$ and possibly their derivatives during each minor iteration of the subproblem.

2.2.3. QP subproblems

SQP methods use the same linearized constraints, but the subproblem objective $\tilde{F}$ is a local quadratic model of the Lagrangian

$$L(x, \lambda) = F(x) - \lambda^T c(x) - \eta^T(x - l) - \sigma^T(u - x). \tag{2.2.7}$$

The first two terms of the Taylor expansion of (2.2.7) define the quadratic model:

$$\tilde{F} = Q_k(p) \equiv \tfrac{1}{2} p^T H_k p + g_k^T p, \tag{2.2.8}$$

where H_k is an approximation to the Hessian of the Lagrangian ($H_k \approx W(x_k, \lambda_k)$) and $g_k = \nabla F(x_k)$. One resulting QP subproblem at the point x_k is

$$\boxed{\begin{array}{ll} \underset{p,q}{\text{minimize}} & Q_k(p) = \tfrac{1}{2} p^T H_k p + g_k^T p \\ \text{s.t.} & J_k p - q = -(c_k - s_k), \\ & l_k \leq \begin{pmatrix} p \\ q \end{pmatrix} \leq u_k, \end{array}} \quad QP(x_k)$$

whose solution we denote by (p_k, q_k) with Lagrange multipliers μ_k.

Note that the linear term in $Q_k(p)$ is defined using $g_k = \nabla F(x_k)$ and not $\nabla_x L$. In general, replacing $\nabla_x L$ by g_k in (2.2.8) may affect the sequence $\{x_k\}$. However, if $\eta = \eta_{QP}$ and $\sigma = \sigma_{QP}$ (the optimal QP multipliers from the solution of $QP(x_k)$), then examination of the KKT conditions for the QP subproblem shows that $p_k(g_k) = p_k(\nabla_x L)$. To see why, note that the KKT necessary and sufficient conditions require only that p_k satisfy

$$Z^T \nabla Q_k(p_k) = 0. \tag{2.2.9}$$

Define $A = (J \quad -I)$. Recall from (2.1.10) and (2.1.11) that the terms η and σ correspond to the reduced costs $(g - A^T\lambda)$; hence, the only difference between the QP gradient in (2.2.8) and that of the model function defined in terms of $\nabla_x L$ is the addition of linear terms involving A. These terms are annihilated by premultiplication by Z^T in (2.2.9); hence the solution p_k of the subproblem $QP(x_k)$ is unaltered.

A side-effect is that, when (2.2.8) is used as the objective in $QP(x_k)$, the optimal Lagrange multipliers μ^* for the constraints $J_k p = -(c_k - s_k)$ of $QP(x_k)$ are used as estimates of λ^* (rather than as a search direction for λ_k).

To obtain fast convergence it is necessary to include approximations to the second-order constraint terms, which are part of W, the Hessian of the Lagrangian:

$$W(x_k, \lambda_k) \equiv G(x_k) - \sum_{i=1}^{m} (\lambda_k)_i G_i(x_k).$$

Also note that errors in λ_k and μ^* will affect only the second-order terms in the model function. This gives some insight into why these methods are superlinearly convergent.

Using the subproblem defined by $QP(x_k)$ does not require evaluations of the true objective $F(x)$ or constraints $c(x)$ during minor iterations of the subproblem. This can be a great advantage in some applications.

2.2.4. The merit function

As discussed in Section 1, much research has been undertaken on the form of the merit function for measuring the progress of an SQP method. As in [GMSW86b] and [Pri89] we use an augmented Lagrangian merit function and include slack variables for the nonlinear constraints in the merit function:

$$M(x, s, \lambda, \rho) = F(x) - \lambda^T(c(x) - s) + \tfrac{1}{2}\rho\|(c(x) - s)\|^2. \tag{2.2.10}$$

As in NPSOL [GMSW86b], the slacks $s = (s_1, \ldots, s_m)$ are specially constructed for the linesearch function:

$$s_i = \begin{cases} 0 & \text{if } i \in \mathcal{E}, \\ \max(0, c_i(x)) & \text{if } i \in \mathcal{I} \text{ and } \rho = 0, \\ \max(0, c_i(x) - \lambda_i/\rho) & \text{otherwise,} \end{cases} \tag{2.2.11}$$

where $\mathcal{E}$ and $\mathcal{I}$ denote the sets of indices for the nonlinear equality and inequality constraints respectively. Choosing s_k in this way is equivalent to setting the slacks at their optimal feasible values if the merit function were being minimized only with respect to

s. The merit function is thereby reduced. Define $\xi_k \equiv \mu_k - \lambda_k$ as the search direction for the Lagrange multiplier estimates λ_k and de ne $q_k \equiv A_k p_k + c_k - s_k$ as the search direction for the slacks s_k just defined. Finally, define p_k from the solution of $QP(x_k)$ as the search direction for the variables x_k. To obtain the next iterate the linesearch for the merit function is then performed along the triple search direction:

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \lambda_{k+1} \end{pmatrix} \leftarrow \begin{pmatrix} x_k \\ s_k \\ \lambda_k \end{pmatrix} + \alpha \begin{pmatrix} p_k \\ q_k \\ \xi_k \end{pmatrix}. \tag{2.2.12}$$

The requirement is that at the new iterate the merit function be reduced by a "sufficient" amount. There are a various ways to define "sufficient". Let $\phi(\alpha, \rho)$ (or sometimes $\phi(\alpha)$) denote the (linesearch) merit function:

$$\phi(\alpha, \rho) \equiv M(x_k + \alpha p_k, s_k + \alpha q_k, \lambda_k + \alpha \xi_k, \rho). \tag{2.2.13}$$

In our algorithm we shall choose α to ensure the following conditions:

$$\phi(\alpha) - \phi(0) \leq \sigma \alpha \phi'(0), \tag{2.2.14}$$

$$|\phi'(\alpha)| \leq -\eta \phi'(0), \tag{2.2.15}$$

where $0 < \sigma \leq \eta \leq \frac{1}{2}$. For a proof that a point satisfying these conditions exists, see [GMSW86b] and [MorS84].

2.2.5. Choice of the penalty parameter

The *performance* of the SQP algorithm depends on the choice of the penalty parameter ρ. In practice it is worthwhile having a parameter for each nonlinear constraint even though for theoretical purposes a single parameter ρ would suffice. We take ρ to be an m-vector of penalty parameters and we define $D \equiv \text{diag}(\rho_i)$, where ρ_i is the penalty parameter for the i-th constraint. With this definition the merit function becomes

$$M(x, s, \lambda, \rho) = F(x) - \lambda^T(c(x) - s) + \tfrac{1}{2}(c(x) - s)^T D(c(x) - s). \tag{2.2.16}$$

At each iteration the vector ρ may be modified to ensure that the merit function is reduced by a sufficient amount.

We now omit the subscripts k. Let ϕ' denote the derivative of ϕ with respect to α. We can show that

$$\phi'(0) = g^T p + (2\lambda - \mu)^T(c - s) - (c - s)^T D(c - s). \tag{2.2.17}$$

To achieve reduction of the merit function in the linesearch we require

$$\phi'(0) \leq -\tfrac{1}{2} p^T H p \tag{2.2.18}$$

(see [GMSW86b]). When the nonlinear constraints $c(x)$ are violated, $\phi'(0)$ may not satisfy (2.2.18) for the current value of the vector ρ, which then must be modified.

To obtain a sufficient reduction in the merit function when $\rho_i = \rho$, it can be shown by rearranging terms in (2.2.17) that the *minimum* value of ρ that ensures (2.2.18) is given by

$$\bar{\rho} = \frac{g^T p + (2\lambda - \mu)^T(c - s) + \frac{1}{2} p^T H p}{\|c - s\|^2}. \tag{2.2.19}$$

Clearly, other choices of ρ_i will also ensure condition (2.2.18) is satisfied. Two natural questions are: *What is a "good" choice for ρ_i that also ensures (2.2.18)?* and: *What is an adequate measure of goodness?*

As in the code NPSOL [GMSW86a], one possibility is to minimize the two-norm of ρ. Define $r \equiv (r_1, \ldots, r_m)$ where $r_i = (c_i - s_i)^2$ and $\theta \equiv g^T p + (2\lambda - \mu)^T(c - s) + \frac{1}{2}p^T H p$. The choice of parameters under this condition can be expressed as the solution to the following problem:

$$\begin{aligned} \underset{\rho \in \mathbb{R}^m}{\text{minimize}} \quad & \tfrac{1}{2}\rho^T\rho \\ \text{s.t.} \quad & r^T\rho \geq \theta, \\ & \rho \geq 0. \end{aligned} \tag{2.2.20}$$

The solution ρ^* of this optimization problem is easily found, as shown by the following lemma.

Lemma 2.2.1. *For $\theta > 0$ the minimum-euclidean-norm choice of the m-vector of penalty parameters for the augmented Lagrangian merit function (2.2.16) is given by $\rho^* = \lambda r$, where $\lambda = \theta / r^T r$.*

Proof. Let λ and μ be Lagrange multipliers for the inequality constraints. The KKT conditions for (2.2.20) are

$$\begin{aligned} r^T\rho^* &\geq \theta, \\ \rho^* &= \lambda r + \mu, \\ \lambda(r^T\rho^* - \theta) &= 0, \\ \mu_i\rho_i^* &= 0, \\ \lambda &\geq 0, \\ \mu_i &\geq 0, \\ \rho^* &\geq 0. \end{aligned}$$

Since the objective function is strictly convex, (2.2.20) has a unique solution. It may be verified that the above equations are satisfied by $\rho^* = \lambda r$ and $\mu = 0$, where $\lambda = \theta / r^T r$. ∎

In our implementation of the prototype algorithm (Algorithm 2.3.1), we increase ρ_i to the value ρ_i^* when (2.2.18) is violated.

2.2.6. Decreasing ρ

We use the result of Lemma 2.2.1 to increase the value of each ρ_i when it is smaller than its optimal value. As mentioned in Section 1, it is also necessary to ensure that ρ is not too large. When ρ_i is larger than ρ_i^* it is possible to reduce ρ_i and still satisfy condition (2.2.18). In these instances we compute a trial value $\hat{\rho}_i$ that is equal to the geometric mean of the previous ρ and a damped value of ρ^*. The trial value is

$$\hat{\rho}_i = \sqrt{\rho_i(\delta_k + \rho_i^*)}, \tag{2.2.21}$$

where $\delta_k \geq 1$ is a damping parameter, and the new ρ_i is defined as

$$\rho_i = \begin{cases} \hat{\rho}_i & \text{if } \hat{\rho}_i \leq \frac{1}{2}\rho_i, \\ \rho_i & \text{otherwise.} \end{cases} \tag{2.2.22}$$

To avoid too many modifications of ρ, each time any element of ρ changes, the damping parameter δ_k is increased by a factor of two. This ensures that ρ_i will oscillate only a finite number of times.

2.2.7. Updates to the QP Hessian

Upon completion of the linesearch procedure it is necessary to set up the next subproblem. This consists of evaluating the gradient $\bar{g}$, the nonlinear constraints $\bar{c}$ and the Jacobian $\bar{J}$ at the new iterate. The new QP subproblem also requires $\bar{H}$, an approximation to $\bar{W}$. In dense SQP algorithms, $\bar{H}$ is usually taken as a quasi-Newton approximation to $\bar{W}$. As mentioned in Section 1, for the large-scale case this can be computationally prohibitive. Methods for obtaining $\bar{H}$ in which only an approximation to the reduced Hessian $\bar{Z}^T\bar{H}\bar{Z}$ is maintained are discussed in detail in Section 3.

2.3. The prototype algorithm

Taking into account the descriptions of Section 2.2 we can embellish the model algorithm of Section 1. The new algorithm builds on the framework of the model algorithm by requiring

- use of the augmented Lagrangian merit function (2.2.16);
- estimation of Lagrange multipliers;
- use of the reduced Hessian.

We now present a *prototype SQP algorithm* for NLP. The main steps are summarized in Algorithm 2.3.1.

2.3.1. Convergence of the prototype algorithm

The prototype algorithm draws on the work of Prieto [Pri89] and Gurwitz [Gur87]. It solves a sequence of problems of the form $QP(x_k)$, giving a sequence of solutions $\{x_k\}$. We make the following assumptions:

A_1. The SQP iterates $\{x_k\}$ all lie in a closed, bounded region $\Omega \subset \Re^n$.

A_2. The objective $F(x)$ and the constraints $c_i(x)$ and their first and second derivatives are continuous and uniformly bounded in norm on Ω.

A_3. The Jacobian of active constraints at any limit point of $\{x_k\}_{k=0}^{\infty}$ has full row rank.

A_4. There exists a feasible point for each QP subproblem.

Algorithm 2.3.1. (Prototype SQP algorithm)

Start with estimates of the solution x_0, multipliers λ_0, and reduced Hessian of the Lagrangian H_0.

while not *converged* **do**

Evaluate the Jacobian $J(x)$ and set up the subproblem $QP(x)$.

Find a constrained stationary point p of $QP(x)$ with associated multipliers μ.

if $p = 0$ *and convergence criteria are satisfied* **then**

converged = true

else

Compute slacks s for the merit function and search directions for multipliers and slacks: $\xi = \mu - \lambda$, and $q = Ap + (c - s)$.

Update the diagonal penalty matrix $D = diag(\rho_i)$ if necessary.

Compute the steplength α satisfying steplength criteria for the merit function $M(x, s, \lambda, \rho)$. The linesearch is performed on the variables x, s, and λ along corresponding search directions p, q, and ξ.

Update $\bar{x} \leftarrow x + \alpha p$ and $\bar{\lambda} \leftarrow \lambda + \alpha\xi$.

Update the reduced Hessian approximation $Z^T H Z$.

end if

end do

Figure 1. Prototype SQP algorithm

A_5. Strict complementarity holds for each constrained stationary point of NLP in Ω.

A_6. The reduced Hessian of the Lagrangian is nonsingular at all KKT points of NLP.

2.3.2. Global convergence

Prieto [Pri89] in his doctoral dissertation analyzed the convergence properties of a reduced-Hessian algorithm based on the use of the Lagrangian to define the QP subproblem. He proved under assumptions A_1–A_6 and certain conditions on the multiplier estimates that the algorithm is globally convergent. (These conditions are satisfied by the estimates in our prototype SQP algorithm.) The main theorem and an important corollary from this work are repeated here. Theorem 2.3.1 and Corollary 2.3.1 are proved in [Pri89] as Corollary 5.2.1 and Corollary 5.2.2 respectively.

Theorem 2.3.1. *Under assumptions A_1-A_6 and the additional assumption that* $\mu_k = \lambda^* + \mathcal{O}(\|x_k - x^*\|)$,

$$\lim_{k\to\infty} \|x_k - x^*\| = 0.$$

Corollary 2.3.1. *Under assumptions A_1-A_6 and the additional assumption that* $\mu_k = \lambda^* + \mathcal{O}(\|x_k - x^*\|)$,

$$\lim_{k\to\infty} \|\lambda_k - \lambda^*\| = 0.$$

2.3.3. Rate of convergence

In addition to global convergence, we are naturally interested in the rate of convergence. We have assumed that our approximation to the Hessian is only accurate on the null space of the active constraints. A consequence of the use of less precise information is a degradation in the rate of convergence for the algorithm, relative to one in which the full Hessian is available or approximated. It is shown in [Pri89] that provided the penalty parameter is chosen to be sufficiently large and H_z is a sufficiently good approximation to the reduced Hessian of the Lagrangian, the algorithm converges two-step superlinearly. That is, the iterative sequence $\{x_k\}$ satisfies

$$\lim_{k\to\infty} \frac{\|x_{k+2} - x^*\|}{\|x_k - x^*\|} = 0. \tag{2.3.1}$$

The precise conditions are:

- A_7. When the iterates are close to the solution, the penalty parameter is chosen to be "large enough".
- A_8. $\|Z_k^T(H_k - W_k)Z_k p_{z_k}\| = o(\|p_k\|)$, where W_k denotes the Hessian of the Lagrangian at x_k.

The theorem giving the required rate of convergence (Theorem 5.3.1 in [Pri89]) is stated here without proof.

Theorem 2.3.2. *There exists a value $\bar{\rho}$, such that if ρ_k is selected satisfying $\rho_k \geq \bar{\rho}$, and if assumptions A_1-A_8 and the additional assumption that $\mu_k = \lambda^* + \mathcal{O}(\|x_k - x^*\|)$ are satisfied, then the algorithm converges two-step superlinearly.*

2.4. Solution of the subproblem

The method used to solve the QP subproblems is an *active-set method.* It is related to the reduced-gradient method as implemented in MINOS [MurS78].

As in MINOS, it is computationally convenient to convert all general constraints to equalities, with the only inequalities being simple bounds on the variables. For notational convenience, the search direction for the subproblem $QP(x_k)$ is augmented to include the slacks q. Define $A_k \equiv \begin{pmatrix} J_k & -I \end{pmatrix}$ and define $g_k \in \Re^{n+m}$ to be the original gradient vector

augmented by m zeros. Likewise, the QP Hessian is augmented with zeros so that it has dimension $n + m$:

$$\bar{H}_k \equiv \begin{pmatrix} H_k & 0 \\ 0 & 0 \end{pmatrix}. \tag{2.4.1}$$

We may write the new QP subproblem in standard form as

$$\boxed{\begin{array}{ll} \underset{p\in\Re^{n+m}}{\text{minimize}} & Q_k(p) = \frac{1}{2}p^T\bar{H}_k p + g_k^T p \\ \text{s.t.} & A_k p = -(c_k - s_k), \\ & l_k \le p \le u_k, \end{array}} \qquad QPSF(x_k)$$

with optimal solution p^* and optimal Lagrange multipliers μ^*.

At a local minimizer of $QPSF(x_k)$, the point p^* satisfies $A_k p^* = -(c_k - s_k)$. In addition, many variables p_j^* (usually) attain the value of one of their bounds. It is of interest to consider the set of indices corresponding to the bounds on p^* that are exactly satisfied (i.e. $p_j^* = (l_k)_j$ or $(u_k)_j$). We call this index set of constraints that are "tight" or "active" at the solution the *active set.* The active set for the subproblem can be represented by the *active-set matrix,*

$$\bar{A} = \begin{pmatrix} \tilde{A} & \tilde{N} \\ & I \end{pmatrix}, \tag{2.4.2}$$

where $\tilde{N}$ consists of the columns of the linearized constraints corresponding to variables exactly equal to one of their bounds at the optimal solution of the subproblem. The columns of $\tilde{A}$ correspond to the remaining variables.

If the active set were known *a priori*, the solution to the subproblem could be solved in a single iteration. In general we do not know the active set at the start of the solution process for a subproblem. Active-set methods employ what is called a *working set,* which attempts to predict the active set. The working-set matrix $\hat{A}$ has the form

$$\hat{A} = \begin{pmatrix} B & S & N \\ & & I \end{pmatrix}, \tag{2.4.3}$$

where N consists of the columns of A corresponding to variables temporarily held at their current values (typically on a bound), and B and S are the remaining columns of A partitioned so that B is nonsingular. We refer to the variables corresponding to the columns in B, S, and N as *basic, superbasic,* and *nonbasic* variables respectively.

Active-set methods employ a procedure to check whether a feasible stationary point is optimal (i.e. when the working set has identified the active set). Let $g_{QP}^* \equiv \bar{H}_k p^* + g_k$ and omit the subscript k. Consider the first-order KKT conditions for (p^*, μ^*) to be a local minimum of the QP subproblem:

$$Ap^* = -(c - s), \tag{2.4.4}$$

$$p^* \ge l, \tag{2.4.5}$$

$$p^* \le u, \tag{2.4.6}$$

$$Z^{*T} g_{QP}^* = 0, \tag{2.4.7}$$

$$(u_j - p_j^*)((g_{QP}^*)_j - a_j^{*T}\mu^*) \ge 0 \text{ for } j = 1, \ldots, n, \tag{2.4.8}$$

$$(p_j^* - l_j)((g_{QP}^*)_j - a_j^{*T}\mu^*) \quad \leq \quad 0 \text{ for } j = 1, \ldots, n. \tag{2.4.9}$$

This means that p^* must be feasible and the reduced gradients $(g_{QP}^*)_j - a_j^{*T}\mu$ must be zero for any variable not on a bound (including for example a free variable that is nonbasic). Let p be a feasible constrained stationary point for the subproblem. If KKT conditions (2.4.8)–(2.4.9) hold then the active set has been identified and p is an optimal solution to the subproblem. Verification of (2.4.8) and (2.4.9) for nonbasic variables is carried out in the process of solving the QP subproblem. This procedure is known as *pricing* and is used to modify the working set. When a nonbasic variable fails the pricing test, the QP objective can be reduce by deleting the variable from the working set and moving the variable off its bound. These ideas give rise to a model algorithm for QP subproblems.

Algorithm 2.4.1. (Model QP algorithm)

Find a feasible point for the QP subproblem. This defines a working set.

while not *converged* **do**

While remaining feasible, find a constrained stationary point for the QP subproblem. This process may increase the size of the working set as one or more of the basic or superbasic variables encounter a bound.

Price to determine if the working-set size should be reduced.

Modify the working set by allowing one or more nonbasic variables to move off a bound.

end do

Figure 2. Model QP algorithm

Chapter 3

Large-scale Quadratic Programs

In this chapter, important computational constructs are developed to assist solution of the large-scale QP subproblems arising in the SQP method.

3.1. The null-space basis Z

Recall from Section 2 that during the solution of a QP subproblem it is necessary to maintain Z, a null-space basis of the working-set matrix $\hat{A}$. This is because the optimality conditions for the QP subproblem depend in part on Z and, as demonstrated in Section 3.2, the form of the QP Hessian also depends on the form of Z.

For the dense case of NLP, Gill *et al.* [GMSW84, GMSSW85] have used an orthonormal basis Z obtained by *updating* the rows and columns of the TQ factorization

$$\hat{A} = \begin{pmatrix} 0 & T \end{pmatrix} Q, \tag{3.1.1}$$

where Q is an $n \times n$ orthogonal matrix, and T is a triangular matrix with varying dimension t. In this case Q can be partitioned as

$$Q = (\overbrace{Z}^{n-t} \quad \overbrace{Y}^{t}). \tag{3.1.2}$$

Forming an orthogonal Q in the large-scale case is prohibitively expensive in general. A practical method when n is large is to represent Z as the *reduced-gradient* null-space basis of the working-set matrix $\hat{A}$ (2.4.3). This has been used with success in the mathematical programming system MINOS [MurS78, MurS87] and has the form

$$Z = \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}. \tag{3.1.3}$$

As long as B in (2.4.3) is nonsingular, Z in (3.1.3) is a basis for the null space of $\hat{A}$. In contrast to the dense case, this matrix Z is not computed or stored explicitly. Instead, a sparse factorization $B = LU$ is maintained along with an index set for the columns of B, S, and N. Products involving Z and Z^T can then be performed easily by solving with

B or B^T and using the nonzero elements of the columns in S. For example, the reduced gradient

$$\eta \equiv Z^T g = g_S - S^T B^{-T} g_B \tag{3.1.4}$$

may be obtained from the operations

$$\text{solve} \quad U^T v = g_B; \tag{3.1.5}$$

$$\text{solve} \quad L^T \pi = v; \tag{3.1.6}$$

$$\text{form} \quad \eta = g_S - S^T \pi. \tag{3.1.7}$$

3.2. Solution of subproblems

SQP methods make use of local curvature information to construct QP subproblems. Recall from Section 2 that the search direction p for each SQP iterate is constructed from a constrained stationary point of a large-scale QP subproblem (LSQP) defined at the current NLP iterate x:

$$\boxed{\begin{array}{ll} \underset{p \in \Re^{m+n}}{\text{minimize}} & \frac{1}{2} p^T H p + g^T p \\ \text{s.t.} & Ap = -c, \\ & l \leq p \leq u. \end{array}} \qquad \text{LSQP}$$

This form of LSQP is slightly different from the subproblem QP(x_k) defined in Section 2. Here $p \in \Re^{m+n}$ is a search direction for the m slack variables s as well as the variables x. Accordingly, we define $A \equiv (\; J(x) \quad -I \;)$, where $J(x)$ is the Jacobian of $c(x)$. For the sake of brevity, the right-hand-side vector for LSQP has been redefined to be $c \equiv c(x) - s$.

Clearly, the ability to solve large-scale QP subproblems efficiently is crucial to our SQP algorithm. Because we have some freedom in determining the form of the QP Hessian H, we construct H to be positive definite. The minimizer of the subproblem p^* is then unique (i.e. a *global* minimizer). In our algorithm, H will not be explicitly available. A key concept is to work with a nonsingular matrix Q such that Q^{-1} and $Q^T H Q$ have a reasonably simple form. The forms of Q, Q^{-1} and $Q^T H Q$ are discussed in Section 3.5.

At each iteration of most active-set methods (and all of the methods we consider) a KKT system is solved:

$$\begin{pmatrix} H & \hat{A}^T \\ \hat{A} & \end{pmatrix} \begin{pmatrix} p \\ -\mu \end{pmatrix} = - \begin{pmatrix} g \\ \hat{c} \end{pmatrix}, \tag{3.2.1}$$

where $\hat{A}$ is the working-set matrix and $\hat{c}$ is the QP right-hand-side vector padded with zeros to make it compatible with $\hat{A}$. Nearly all active-set methods for solving LSQP generate the same sequence of iterates (see [CotD79], for example). The methods differ in *how* the iterates are computed, and their efficiency depends on the problem type. QP methods may be categorized according to the approach used to solve (3.2.1). If the system is solved directly we say the QP method is a *Lagrangian method.* If (3.2.1) is reduced to solving two smaller systems we refer to the method as a *projection method.*

Let Z be a basis for the null space of $\hat{A}$ and define Y so that $(\; Z \quad Y \;)$ and $\hat{A}Y$ are nonsingular. Projection methods come in two different flavors: *range-space* methods and *null-space* methods (see [GMW81], for example). This terminology arises from the fact

that $\hat{A}$ defines two complementary subspaces spanned by Z and Y. The work required to solve for the optimality conditions is directly related to the dimension of either the null space (the dimension of $Z^T H Z$) or the range space (the dimension of $Y^T H^{-1} Y$). Because we use a null-space method for solving (3.2.1), the method is most efficient when the dimension of the null space of $\hat{A}$ is small. It is this class of problems that we are most interested in.

Each iteration of an active-set method for solving LSQP is called a *minor* iteration. Solving the associated KKT system (3.2.1) is equivalent to solving a single *equality-constrained* quadratic program (EQP).

The following is a model active-set algorithm for solving LSQP. It assumes that a working set of the form (2.4.3) is available and that the associated point p is feasible, i.e. satisfies the constraints in LSQP.

Algorithm 3.2.1. (Active-set algorithm)
while not *converged* **do**
  if *the minimizer has been found on the current subspace* **then**
    Price nonbasic variables.
    if *no new superbasic candidates exist* **then**
      converged = *true*
    else
      Delete a bound from the working set.
    end if
  end if
  if not *converged* **then**
    Solve the EQP defined by the working set.
    if *a bound was encountered in the solution of the EQP* **then**
      Add the bound encountered to the working set.
    end if
  end if
end do

Figure 1. Active-set algorithm

Our definition of *convergence* in Algorithm 3.2.1 depends on finding a minimizer of LSQP. The algorithm is easily modified to halt upon finding the first constrained stationary point (see Section 3.2.3). In addition, Algorithm 3.2.1 requires finding a feasible point, the computation of which is itself an optimization problem. The main parts of Algorithm 3.2.1 (signified by the boxed text) will be discussed in the following sections.

3.2.1. Finding an optimal point

Let $\hat{A}$ denote the working set for a feasible point p for LSQP. Since H is chosen to be positive definite, the reduced Hessian $Z^T H Z$ is known *a priori* to be positive definite for every minor iteration. Define $g_{QP} \equiv g + Hp$, the QP gradient at the point p, and $H_Z \equiv Z^T H Z$, the reduced Hessian.

The *optimality-phase algorithm* starts with a feasible point p, tolerances δ_{rg} and δ_{dj}, and a partition of A into (B S N). Define n_S to be the number of columns in S. Assume that the QP gradient g_{QP} is available, along with factorizations for the basis $B = LU$ and the reduced Hessian $H_Z = R^T R$.

The first step in finding an optimal point is to determine whether the current point p is a constrained stationary point, by checking whether the current point is a minimizer on the current subspace or equivalently, the norm of the reduced gradient is zero (or smaller than a specified tolerance). If so, it is then necessary to check whether p is the *minimizer of the subproblem*, by checking the signs of the multipliers (also called reduced costs) for the bound constraints. The multipliers are calculated by *pricing*, as summarized in Procedure 3.2.1.

Procedure 3.2.1. (Price nonbasic variables)

Form $g_{QP} = g + Hp$.

Solve $U^T L^T \mu = (g_{QP})_B$.

Calculate $\eta = (g_{QP})_N - N^T \mu$.

Select η_l (η_u), *the most negative (positive) element of* η *corresponding to variables at their lower (upper) bounds.*

If $\eta_l \geq -\delta_{dj}$ and $\eta_u \leq \delta_{dj}$, we conclude that p is a minimizer of the subproblem. Otherwise, the QP objective may be reduced by deleting a bound from the working set and adding a variable to the superbasic set. Procedure 3.2.2 summarizes the steps required to modify the working set when a bound is deleted.

Procedure 3.2.2. (Delete a bound from the working set)

Choose $q = argmax(|\eta_l|, \eta_u)$.

Add q *to the superbasic index set.*

Add η_q *as a new element of* $Z^T g_{QP}$.

Add a new column to R *and increase* n_S *by 1.*

The search direction for LSQP at p is defined by the Newton equations,

$$H_Z p_Z = -Z^T g_{QP}, \tag{3.2.2}$$

$$\hat{p} = Z p_Z, \tag{3.2.3}$$

which solve the EQP defined by the current working set. The next iterate $\bar{p}$ for the subproblem is $\bar{p} = p + \alpha\hat{p}$. Due to the quadratic nature of the objective function along

$\hat{p}$, when $p + \hat{p}$ is feasible then $\alpha = 1$ and $\bar{p}$ is a constrained stationary point for LSQP. If $p + \hat{p}$ is not feasible then $\alpha < 1$ is the step to the nearest bound constraint along $\hat{p}$. The precise steps for solving the EQP are summarized in Procedure 3.2.3.

Procedure 3.2.3. (Solve an EQP)

Solve $R^T R p_S = -Z^T g_{QP}$.
Solve $LU p_B = -S p_S$.
Calculate α_c, *the step to the nearest bound along* $\hat{p}$, *where*

$$\hat{p} = (\ p_B, \ \ p_S, \ \ 0 \ ).$$

Compute $\alpha = min(1, \alpha_c)$.
Update $p = p + \alpha \hat{p}$.

When the unit step is not feasible the nearest bound is added to the working set. If the variable corresponds to a column in B, a column from S must be chosen to replace it in B (see Section 3.3, no. 3). The steps required when a bound is encountered in the solution of the EQP are summarized in Procedure 3.2.4.

Procedure 3.2.4. (Add a bound to the working set)

if $\alpha = \alpha_c$ **then**
    Add the new bound to the working set.
    Decrease n_S *by 1.*
    Update R.
    Update the LU *factors if necessary.*
end if

Two tests for convergence are required: one to check for convergence in the current subspace and one to detect convergence to the QP minimizer. As discussed in Section 2, it is not always necessary to obtain the minimizer of the QP subproblem in order to obtain a search direction for NLP. The emphasis of Section 3.2.3 will be to develop a strategy in which we terminate the solution of the subproblem upon finding a QP stationary point. The complete set of steps for the optimality-phase algorithm for finding a minimizer of the QP is given in Algorithm 3.2.2.

3.2.2. Obtaining a feasible point

Algorithm 3.2.2 requires a feasible point for LSQP. At the start of a QP subproblem, the basic variables p_B are defined by

$$B p_B = -c.$$

If p_B is feasible we may commence with the *optimality phase*. A subproblem is infeasible only if the bounds on the variables ($l_B \leq p_B \leq u_B$) are violated. During the feasibility

Algorithm 3.2.2. (Optimality phase for LSQP)

while not *converged* **do**

if $\|Z^T g_{QP}\| \leq \delta_{rg}$ **then** {*Price nonbasic variables*}

Form $g_{QP} = g + Hp$.
Solve $U^T L^T \mu = (g_{QP})_B$.
Calculate $\eta = (g_{QP})_N - N^T \mu$.
Select η_l (η_u), *the most negative (positive) element of* η *corresponding to variables at their lower (upper) bounds.*

if $\eta_l \geq -\delta_{dj}$ **and** $\eta_u \leq \delta_{dj}$ **then**

converged = *true*

else {*Delete a bound from the working set*}

Choose $q = argmax(|\eta_l|, \eta_u)$.
Add q *to the superbasic index set.*
Add η_q *as a new element of* $Z^T g_{QP}$.
Add a new column to R *and increase* n_S *by 1.*

end if

end if

if not *converged* **then** {*Solve an EQP*}

Solve $R^T R p_S = -Z^T g_{QP}$.
Solve $LU p_B = -S p_S$.
Calculate α_c, *the step to the nearest bound along* $\hat{p} = (\ p_B,\ \ p_S,\ \ 0\ )$.
Compute $\alpha = min(1, \alpha_c)$.
Update $p = p + \alpha \hat{p}$.

if $\alpha = \alpha_c$ **then** {*Add a bound to the working set*}

Add the new bound to the working set and decrease n_S *by 1.*
Update R *and update the* LU *factors if necessary.*

end if

Update the reduced gradient.

end if

end do

Figure 2. Optimality phase for LSQP

phase of the subproblem, the objective is the sum of infeasibilities for the bounds. This *Phase 1* problem may be written

$$\begin{array}{ll} \underset{p\in\Re^n}{\text{minimize}} & \sum_{j=1}^{n}((l_k)_j - p_j)^+ + (p_j - (u_k)_j)^+ \\ \text{s.t.} & Ap = -c, \end{array}$$

where $\beta^+ = \max(0, \beta)$. This procedure for finding a feasible point is similar to the *Phase 1* method for finding a feasible point for a linear program, extended to work with nonbasic points (i.e. with superbasic variables). The difference between the feasibility phase and the optimality phase is that the gradient of the sum of infeasibilities must be formed in place of g_{QP}, and the steepest descent search direction is used in place of a search direction defined in terms of the reduced Hessian H_Z.

It is not enough just to find a feasible point. If the working set is changed in the feasibility phase it is necessary to modify H_Z, since this matrix is required for the optimality phase.

When the working set is modified to include additional bounds, the reduced Hessian is modified *within* the feasibility phase. When bounds are deleted from the working set there are two options worth considering.

Because only the steepest descent direction is used during the feasibility phase, one strategy is to wait until a feasible point has been found before expanding the reduced Hessian, should that be necessary. When the working set has been modified and has fewer bounds than at the start of the feasibility phase, the new reduced Hessian may be modified by appending rows and columns of the identity:

$$\bar{H}_Z = \begin{pmatrix} H_Z & \\ & I \end{pmatrix}. \tag{3.2.4}$$

Another strategy is to update $H_Z = R^T R$ to take into account deletions from the working set as they occur in the feasibility phase. For this strategy, the updates are the same as those for the optimality phase and may require multiplications with the full H when the working-set size is reduced. The modifications to H_Z are discussed in detail in Section 3.7.

The *feasibility-phase algorithm* starts with tolerances δ_{rg} and δ_{dj} and a partition of A into $(B \;\; S \;\; N)$. Define n_S to be the number of columns in S. Assume that the basis factorization $B = LU$ is available, set $p_S = 0$ and $p_N = 0$, and solve $LUp_B = -c$ to determine the initial elements of the search direction.

3.2.3. Early termination of subproblems

The prototype SQP algorithm for NLP (Algorithm 2.3.1) allows the use of only stationary points rather than minimizers to construct a search direction for the NLP merit function. As mentioned in Section 1, there are two reasons for early termination of the active-set method. First, it is desirable to place a limit on the computational effort made. Second, when x_k is a poor approximation to x^* (and this is the circumstance when many minor iterations may be required), the effort to find a QP minimizer seems unwarranted in light

Algorithm 3.2.3. (Feasibility phase for LSQP)
while not *(converged* **or** *infeasible)* **do**
Form $\hat{g}$, the gradient of the sum of infeasibilities.
if $\|\hat{g}\| = 0$ **then**
converged = true
else
if $\|Z^T\hat{g}\| < \delta_{rg}$ **then** {*Price nonbasic variables*}
Calculate $\eta = \hat{g}_N - N^T\mu$.
Select η_l (η_u), the most negative (positive) element of η corresponding to variables at their lower (upper) bounds.
if $\eta_l \geq -\delta_{dj}$ **and** $\eta_u \leq \delta_{dj}$ **then**
infeasible = true
else {*Delete a bound from the working set*}
Choose $q = argmax(|\eta_l|, \eta_u)$.
Add q to the superbasic index set.
Add η_q as a new element of $Z^T\hat{g}$ and increase n_S by 1.
end if
end if
if not *(converged* **or** *infeasible)* **then** {*Solve an EQP*}
Solve $p_S = -\hat{g}_S$.
Solve $LUp_B = -Sp_S$.
Calculate α_c, the step to the nearest bound along $\hat{p} = (\ p_B,\ p_S,\ 0\ )$.
Update $p = p + \alpha\hat{p}$.
Add the new bound to the working set and decrease n_S by 1.
Update R and the LU factors if necessary.
end if
end if
end do

Figure 3. Feasibility phase algorithm

of the fact that the subproblem may be a poor model (even locally). Thus, an early-termination strategy may reduce the total number of QP iterations required to find a minimizer for NLP.

There may be a further benefit associated with early termination. The pricing step requires much computational effort (Procedure 3.2.1) and often constitutes a large percentage of overall processing time when the subproblem is solved to optimality. Pricing involves solving with B^T (to obtain μ) and forming $N^T\mu$ (to obtain reduced costs η). The early-termination strategy only requires solves with B (to obtain the search direction) and a linesearch. If a unit step is taken we terminate the solution of the subproblem, since the resulting point is a constrained stationary point. Otherwise, a bound is encountered during the linesearch, the reduced gradient is updated cheaply, and the search direction is calculated anew in the smaller subspace. *This approach requires neither Lagrange multiplier estimates nor reduced costs!* Hence, an important advantage of early termination of subproblems is the elimination of pricing *during* subproblem solution (although it is necessary to price outside the subproblem to determine if the working set should be modified).

Algorithm 3.2.4. (First-stationary-point algorithm for LSQP)
while not *converged* **do**
  if $\|Z^T g_{QP}\| \leq \delta_{rg}$ **then**
    converged = *true*
  else {*Solve an EQP*}
    Solve $R^T R p_S = -Z^T g_{QP}$.
    Solve $LU p_B = -S p_S$.
    Calculate α_c, *the step to the nearest bound along* $\hat{p} = (\ p_B,\ p_S,\ 0\ )^T$.
    Compute $\alpha = min(1, \alpha_c)$.
    Update $p = p + \alpha\hat{p}$.
    if $\alpha = \alpha_c$ **then** {*Add a bound to the working set*}
      Add the new bound to the working set, decrease n_S *by 1, and update R and the LU factors if necessary.*
    end if
    Update the reduced gradient.
  end if
end do

Figure 4. First stationary point algorithm

With the early-termination strategy, we have two options for modifying H_Z (as in the feasibility phase). Upon completion of a major iteration we may decide to price *inside* or *outside* the subproblem. After completion of the linesearch to reduce the merit function, the next QP is set up, Lagrange multiplier estimates are then calculated and nonbasic columns are priced. If the current point is not the minimizer, a decision is made to move off one or more of the bounds in the working set.

Pricing outside the subproblem allows the reduced Hessian to be updated to reflect the new superbasic components in a computationally convenient way. The new H_Z is obtained by appending a row and column of the identity for each new superbasic variable as in (3.2.4). This avoids products of the form Hz, which can be computationally expensive. Note that a major difference relative to the algorithm of Prieto [Pri89] is that the prototype algorithm does not calculate an auxiliary search direction once a stationary point has been identified.

Algorithm 3.2.4 presents a stationary-point algorithm for LSQP that terminates upon finding the *first* constrained stationary point encountered during subproblem solution. The *First-stationary-point algorithm* starts with a feasible point p, a tolerance δ_{rg}, and a partition of A into (B S N). Define n_S to be the number of columns in S. Assume that the QP gradient $g_{QP} = g + Hp$ is available and factorizations are available for the basis $B = LU$ and the reduced Hessian $H_Z = R^T R$.

3.3. Updating Z

When $\hat{A}$ is updated during the solution of the subproblem it is necessary to update both Z and H_Z. The updates to the working set come in three forms:

1. *A bound is deleted and the corresponding column is added to S.*
   When we decide to drop one or more of the bound constraints from the working set this has the effect of adding one or more columns to the matrix S. The addition of a superbasic also increases the number of columns of Z, and the dimension of H_Z.

2. *A variable corresponding to a column in S hits a bound.*
   When a variable corresponding to the q-th column in S encounters a bound, the variable is deleted from S and added to N. Both Z and H_Z must be updated to reflect the modification to S. The q-th column of Z is implicitly deleted, and R is modified to reflect deletion of the q-th row and column of H_Z.

3. *A variable corresponding to a column in B hits a bound.*
   When a variable corresponding to a column in B encounters a bound the updates to H_Z are more complicated. It is necessary to replace the column from B with a suitable column from S (one that maintains the nonsingularity of B). Updates to the LU factors of B are carried out using a method standard to the simplex method for linear programming (see [GMSW87]).

The updates to H_Z for each of these cases are discussed in Section 3.7.

3.4. Continuity of Z

In order to prove that Algorithm 2.3.1 has a superlinear rate of convergence, it is necessary to assume that H_z is an adequate approximation to Z^TWZ. In Section 2 we assumed that the gradients and Hessians of $F(x)$ and $c(x)$ exist and are continuous and uniformly bounded in norm on Ω. The quasi-Newton scheme for approximating Z^TWZ is based on *inherited* information. If Z^TZ is not continuously differentiable in the neighborhood of x^* then the assumption that H_z is a good approximation to Z^TWZ is not reasonable.

Discussion of the continuity of Z was initiated by Coleman and Sorensen [ColS84], who showed that a standard method for computing an orthogonal factorization of $\hat{A}$ may not provide a continuously differentiable $Z(x)$. Gill *et al.* [GMSSW85] showed how to compute a continuously differentiable Z using regularized Householder transformations, and they proved the convergence of both Q and Z under appropriate assumptions.

The difficulties associated with the continuity of $Z(x)$ that arise using orthogonal transformations do not arise for the *reduced-gradient* form of Z, as the following lemma demonstrates.

Lemma 3.4.1. *Let the sequence $\{x_k\}$ be defined by the prototype SQP algorithm with limit point x^*. Let $\mathcal{B}$ be a ball around a point x^* and suppose that the correct active set has been identified and*

$$A(x_k) = \begin{pmatrix} B_k & S_k & N_k \end{pmatrix}$$

is a continuously differentiable function of x_k in $\mathcal{B}$. Further, suppose that B_k has rank m for all x_k in $\mathcal{B}$ and the indices defining the columns in B_k are identical for all x_k in $\mathcal{B}$. Then

$$\begin{array}{ccc} B_k & S_k & N_k \\ \downarrow & \downarrow & \downarrow \\ B^* & S^* & N^* \end{array}$$

and the null-space basis $Z(x_k)$ obtained as the reduced gradient matrix from (3.1.3) has elements that are continuously differentiable for all x_k in $\mathcal{B}$.

Proof. Since B_k has the same indices and the active set is fixed inside $\mathcal{B}$, by definition of the active set the result $A(x_k) - A^* = \begin{pmatrix} B^* & S^* & N^* \end{pmatrix}$ must follow. In addition, since $A(x_k) = \begin{pmatrix} B_k & S_k & N_k \end{pmatrix}$ is continuously differentiable for x_k in $\mathcal{B}$ then B_k also has these properties. Moreover, since B_k is continuously differentiable and has full rank, B_k^{-1} exists and is also continuously differentiable. Finally, the continuity of $Z(x_k)$ follows from the fact that since S_k has elements that are continuously differentiable for all x_k in $\mathcal{B}$, the linear transformation $B_k^{-1}S_k$ has continuously differentiable elements for all x_k in $\mathcal{B}$. ∎

The fact that B_k^{-1} is not explicitly computed, but operations with the matrix are done using the LU factors of B_k, does not impact the continuity of Z. Note that in general the LU factors are not continuously differentiable but B_k^{-1} is.

3.5. The QP Hessian and the transformation matrix Q

In our prototype algorithm we recur H_z, an approximation to the reduced Hessian of the Lagrangian. Condition A_8 of Section 2.3.1 requires that H, the approximation to W, be

accurate only in the null space of the rows of $\hat{A}$. We are free to define H in any way provided

$$Z^T H Z = H_Z. \tag{3.5.1}$$

It is important to note that although (3.5.1) must hold, the matrix product is never actually formed.

When only the reduced Hessian is recurred it is not obvious how the QP gradient can be formed without great expense, since the QP gradient depends on H. To form the QP gradient in the dense case (when H is known explicitly) we would simply form

$$g_{QP} = g + Hp. \tag{3.5.2}$$

In the large-scale case, H is not stored and so direct multiplication is not possible. Fortunately, we have considerable freedom in the definition of H while still being able to satisfy (3.5.1). We shall use this freedom to make the computation of g_{QP} easy.

In the dense SQP method of NPSOL [GMSW86a], an important computational device has been to work with a transformed Hessian approximation $Q^T H Q = R^T R$, where Q is the nonsingular matrix that triangularizes the working-set matrix. An analogous device is essential to the success of our large-scale algorithm. We define the *transformation matrix* Q to be

$$Q \equiv \begin{pmatrix} Z(x) & Y(x) \end{pmatrix}, \tag{3.5.3}$$

where Z and Y satisfy the following requirements:

- Z is a basis for the null space of the active constraints at the current point; i.e. $\hat{A}(x)Z(x) = 0$;
- Q is nonsingular.

We also define

$$Q^T H Q \equiv \begin{pmatrix} D_B & 0 & 0 \\ 0 & H_Z & 0 \\ 0 & 0 & D_N \end{pmatrix},$$

where D_B and D_N are nonnegative diagonal matrices each having zero elements on its diagonals corresponding to linear or slack variables.

Thus, $Z^T H Y = 0$ and $Y^T H Y = D$, and the gradient for the QP is given by

$$g_{QP} = g + Q^{-T}(Q^T H Q)Q^{-1}p. \tag{3.5.4}$$

Note that the transformed Hessian approximation $Q^T H Q$ is known and is simple, but operations with Q^{-1} (rather than Q) are required to compute g_{QP}. We choose the null-space basis Z to have the reduced-gradient form (3.1.3) but we have some freedom in choosing Y. In the next section we discuss various choices for Y. Each choice gives rise to a different Q^{-1}, which in turn affects the effort required to compute the QP gradient. The merits of the various choices are then compared.

3.6. The matrix Y

For each QP subproblem the QP Hessian depends only on $\hat{A}_0$, the working-set matrix *at the start of the QP subproblem.* Define

$$\hat{A}_0 \equiv \begin{pmatrix} B_0 & S_0 & N_0 \\ & & I \end{pmatrix}. \tag{3.6.1}$$

The null-space basis for $\hat{A}_0$ will be denoted by Z_0 and Y for $\hat{A}_0$ will be denoted by Y_0 with $Q_0 \equiv (Y_0 \;\; Z_0)$. The definition of $\hat{A}$ (i.e. B, S, and N), Z, and Y may change during the solution of the QP subproblem, but the matrix H remains constant. It is therefore necessary to remember the column indices in $\hat{A}_0$.

In this section we define three possible choices for Y_0. In addition to requiring Q_0 to be nonsingular we would like Q_0 to be well-conditioned and operations with Q_0^{-1} to be cheap. In the following sections the subscripts on the matrices Y_0, Z_0, etc. will be dropped (e.g. $Q = Q_0$); when a matrix corresponds to a QP iteration other than the initial one it will be denoted by a subscript for the current iteration count (e.g. $B \equiv B(p_0)$ and $B_k \equiv B(p_k)$).

3.6.1. Y defined using a partition of I

Lemma 3.6.1. *When Y is of the form*

$$Y = \begin{pmatrix} I & 0 \\ 0 & 0 \\ 0 & I \end{pmatrix}, \tag{3.6.2}$$

then Q is nonsingular and

$$Q^{-1} = \begin{pmatrix} 0 & I & 0 \\ I & B^{-1}S & 0 \\ 0 & 0 & I \end{pmatrix}. \tag{3.6.3}$$

Proof. Permutation of the first two blocks of columns from Q gives

$$\begin{pmatrix} -B^{-1}S & I & \\ I & & \\ & & I \end{pmatrix} \begin{pmatrix} & I & \\ I & & \\ & & I \end{pmatrix} = \begin{pmatrix} I & -B^{-1}S & \\ & I & \\ & & I \end{pmatrix}, \tag{3.6.4}$$

which shows that Q is nonsingular. The result for Q^{-1} may be shown by block premultiplication of (3.6.3) by Q. ∎

With this choice of Y the QP gradient becomes

$$g_{QP} = g + \begin{pmatrix} (p_B + B^{-1}Sp_S) \\ S^TB^{-T}(p_B + B^{-1}Sp_S) + H_Zp_S \\ p_N \end{pmatrix}. \tag{3.6.5}$$

The work required to compute g_{QP} consists of matrix-vector products with S and S^T, a matrix-vector product with H_Z, and two solves with the basis (one with B and one with

B^T). The QP gradient can be obtained as follows:

$$\begin{aligned}
&\text{form} && w = Sp_S; && (3.6.6)\\
&\text{solve} && Bv = w; && (3.6.7)\\
&\text{solve} && B^T u = p_B + v; && (3.6.8)\\
&\text{form} && w = S^T u; && (3.6.9)\\
&\text{form} && q = H_Z p_S; && (3.6.10)\\
&\text{form} && g_{QP} = \begin{pmatrix} g_B \\ g_S \\ g_N \end{pmatrix} + \begin{pmatrix} p_B + v \\ q + w \\ p_N \end{pmatrix}. && (3.6.11)
\end{aligned}$$

We may simplify the computations (3.6.6)–(3.6.10) when the feasibility phase is terminated without changing the working set. When the initial p is feasible with respect to its bounds, equations (3.6.6)–(3.6.7) can be omitted from the computation.

During the solution of a subproblem the factors $L_k U_k$ of the basis B_k change as the working set changes (refer to Procedure 3.2.4, for example). When Y is of the form (3.6.2) we must maintain a *factorization* of both B and B_k since the use of Q requires the use of B_0^{-1}. To avoid having to store two separate LU factorizations, one for B and one for B_k, we could obtain B_k from B_{k-1} using a classical *product-form* update [Orc68] (or alternatively a *block-LU* update as described in Eldersveld and Saunders [EldS90]). This would allow us to perform operations with both B and B_k with no increase in storage over that for just B_k. However, difficulty arises if the number of updates to B becomes large. Normally, B_k is refactorized every 50–150 minor iterations. Thus, if the QP solution process caused B to undergo many updates it would still be necessary to store two factorizations.

3.6.2. An orthogonal Y

Following the lead established by using the orthogonal TQ factorization for the dense case, we can choose Y to satisfy $Z^T Y = 0$. Unlike the Y obtained from the TQ factorization we shall not require $Y^T Y = I$. This choice ensures that Q is nonsingular if Y has full column rank. We would also expect Q to be similarly conditioned to Z provided Y is also similarly conditioned to Z.

There are many possible choices for an orthogonal Y. An obvious choice is A^T, but such a choice does not give a computationally convenient form for Q^{-1}. A convenient choice is presented in the following lemma.

Lemma 3.6.2. *When Y is of the form*

$$Y = \begin{pmatrix} I & 0 \\ S^T B^{-T} & 0 \\ 0 & I \end{pmatrix}, \qquad (3.6.12)$$

then Y is orthogonal to Z and Q is nonsingular, with

$$Q^{-1} = \begin{pmatrix} -S^T B^{-T} D & C & 0 \\ D & B^{-1} S C & 0 \\ 0 & 0 & I \end{pmatrix}, \qquad (3.6.13)$$

where $C = (I + S^T B^{-T} B^{-1} S)^{-1} = (Z^T Z)^{-1}$ and $D = (I + B^{-1} S S^T B^{-T})^{-1} = (Y^T Y)^{-1}$.

Proof. The result that $Z^TY = 0$ follows easily from the definition of Y and Z. Permutation of Q as in (3.6.4) yields

$$\begin{pmatrix} -B^{-1}S & I & \\ I & S^TB^{-T} & \\ & & I \end{pmatrix} \begin{pmatrix} & I & \\ I & & \\ & & I \end{pmatrix} = \begin{pmatrix} I & -B^{-1}S & \\ S^TB^{-T} & I & \\ & & I \end{pmatrix}. \tag{3.6.14}$$

Performing block elimination on the right-hand side of (3.6.14) reduces the permuted Q to

$$\begin{pmatrix} I & -B^{-1}S & \\ & S^TB^{-T}B^{-1}S & \\ & & I \end{pmatrix}, \tag{3.6.15}$$

which is nonsingular since $S^TB^{-T}B^{-1}S$ is positive definite. Hence Q is nonsingular. The result for Q^{-1} may be shown by block premultiplication of (3.6.13) by Q. ∎

Clearly, using Q^{-1} directly to obtain the QP gradient does not look promising. Using (3.6.13) to form matrix-vector products with Q^{-1} would require separate Cholesky factorizations of both Z^TZ and Y^TY. While we would expect Z^TZ to be small for the problems addressed in this report, the same cannot be said for Y^TY. The following lemma shows the precise form of $(Y^TY)^{-1}$.

Lemma 3.6.3. *When Y is of the form (3.6.12), $(Y^TY)^{-1}$ has the form*

$$(Y^TY)^{-1} = (I - B^{-1}SCS^TB^{-T}), \tag{3.6.16}$$

where $C = (I + S^TB^{-T}B^{-1}S)^{-1} = (Z^TZ)^{-1}$.

Proof. The result may be verified by forming Y^TY from (3.6.12) and multiplying the result into the right-hand side of (3.6.16). ∎

Use of (3.6.16) allows us to provide a more convenient form of Q^{-1} that does not require a factorization of Y^TY.

Lemma 3.6.4. *When Y is of the form (3.6.12), Q^{-1} has the form*

$$Q^{-1} = \begin{pmatrix} -CS^TB^{-T} & C & 0 \\ I - B^{-1}SCS^TB^{-T} & B^{-1}SC & 0 \\ 0 & 0 & I \end{pmatrix}, \tag{3.6.17}$$

where $C = (I + S^TB^{-T}B^{-1}S)^{-1} = (Z^TZ)^{-1}$.

Proof. The result for Q^{-1} arises by the substitution of (3.6.16) into (3.6.13). ∎

The result of Lemma 3.6.4 allows us to compute the subproblem gradient without maintaining a Cholesky factorization of Y^TY. To compute g_{QP} requires four matrix-vector products with S or S^T, four solves with B or B^T, two solves with $C^{-1} \equiv Z^TZ$ and one product with H_Z:

$$\text{solve} \quad B^Tw = p_B; \tag{3.6.18}$$

$$\text{form} \quad u = S^Tw; \tag{3.6.19}$$

$$\begin{aligned}
&\text{solve} && (Z^TZ)y &&= p_S - u; && (3.6.20)\\
&\text{form} && w &&= Sy; && (3.6.21)\\
&\text{solve} && Bv &&= w; && (3.6.22)\\
&\text{solve} && B^Tw &&= p_B + v; && (3.6.23)\\
&\text{form} && u &&= S^Tw; && (3.6.24)\\
&\text{form} && t &&= H_Zy; && (3.6.25)\\
&\text{solve} && (Z^TZ)r &&= t + u; && (3.6.26)\\
&\text{form} && w &&= Sr; && (3.6.27)\\
&\text{solve} && Bq &&= w. && (3.6.28)
\end{aligned}$$

The subproblem gradient may then be written

$$g_{QP} = \begin{pmatrix} g_B \\ g_S \\ g_N \end{pmatrix} + \begin{pmatrix} p_B + v - q \\ r \\ p_N \end{pmatrix}. \tag{3.6.29}$$

As when Y has the form (3.6.2), Q^{-1} is defined in terms of B_0^{-1}. Hence, this form of Y requires two separate factorizations of the basis matrices within the subproblem, one for the initial basis B_0 and one for the current basis B_k.

3.6.3. Y defined using B^{-1}

A choice for Y that leads to a very simple and computationally efficient form for Q^{-1} is given in the following lemma.

Lemma 3.6.5. *When Y is of the form*

$$Y = \begin{pmatrix} B^{-1} & 0 \\ 0 & 0 \\ 0 & I \end{pmatrix}, \tag{3.6.30}$$

Q is nonsingular, and

$$Q^{-1} = \begin{pmatrix} 0 & I & 0 \\ B & S & 0 \\ 0 & 0 & I \end{pmatrix}. \tag{3.6.31}$$

Proof. Permutation of Q as in (3.6.4) yields

$$\begin{pmatrix} -B^{-1}S & B^{-1} & \\ I & & \\ & & I \end{pmatrix} \begin{pmatrix} & I & \\ I & & \\ & & I \end{pmatrix} = \begin{pmatrix} B^{-1} & -B^{-1}S & \\ & I & \\ & & I \end{pmatrix}, \tag{3.6.32}$$

which is nonsingular since B is. Hence Q is nonsingular. The result for Q^{-1} may be shown by block premultiplication of (3.6.31) by Q. ∎

The form of Q^{-1} allows us to compute the QP gradient easily. By definition,

$$g_{QP} = g + Q^{-T} \begin{pmatrix} H_Z & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix} Q^{-1} p \tag{3.6.33}$$

$$= g + \begin{pmatrix} B^T(Bp_B + Sp_S) \\ S^T(Bp_B + Sp_S) + H_Z p_S \\ p_N \end{pmatrix}. \tag{3.6.34}$$

To compute g_{QP} requires three matrix-vector products, one with H_Z, one with $(B \quad S)$, and one with its transpose:

$$w = \begin{pmatrix} B & S \end{pmatrix} \begin{pmatrix} p_B \\ p_S \end{pmatrix}, \tag{3.6.35}$$

$$u = H_Z p_S, \tag{3.6.36}$$

$$v = \begin{pmatrix} B^T \\ S^T \end{pmatrix} w. \tag{3.6.37}$$

Since H_Z is small by comparison to $(B \quad S)$, the main effort is in computing w and v. We then have

$$g_{QP} = \begin{pmatrix} g_B \\ g_S \\ g_N \end{pmatrix} + \begin{pmatrix} v_B \\ v_S + u \\ p_N \end{pmatrix}.$$

It is important to note that Q^{-1} is defined in terms of B_0 not B_0^{-1}. Operations with Q^{-1} require multiplications with B_0. This is easily accomplished by keeping track of the indices of the columns from A that make up B_0. Hence, a benefit of this choice of Y is that operations with Q^{-1} do not require two separate factorizations of the basis matrices, in contrast to the previous two choices of Y.

From a computational standpoint, if we restrict ourselves to pricing only after the feasibility phase of the QP subproblem, the QP gradient can often be computed with about half the effort of the above method. This is because the matrix-vector product w in (3.6.35) can be rewritten as $w = -(c_k - s_k)$ when the initial p is feasible with respect to the bounds (as is usually the case in the later stages as we near optimality). In the event that the initial p is not feasible, w may be calculated as $w = -(c_k - s_k) - Np_N$. This is often cheaper than (3.6.35) because p_N is expected to contain few nonzero elements.

3.6.4. A comparison of the three choices of Y

We summarize the presentation of the three choices of Y by highlighting the computational effort required to compute the QP gradient. The first column of Table 1 gives the specific choice of Y. The remaining columns represent the number of operations (solves or products) with the matrices B, S, H_Z, and Z^TZ respectively. It may be seen that the first and third choices are similar in their computational cost. The orthogonal choice of Y requires about twice the effort and a second Cholesky factorization. Its main benefit is that the resulting Q may be better conditioned than with the other two choices. A

Form of Y	B mult.	B solve	S mult.	H_z mult.	Z^TZ solve
Identity		2	2	1	
Orthogonal		4	4	1	2
Inverse B	2		2	1	

Table 1. Matrix products and solves required to compute g_{QP} for various choices of Y.

virtue of the last choice of Y is that no additional factorization is required. Moreover, multiplications with B are likely to require less effort thatn solves with B, since the LU factors of B contain at least as many nonzero elements as in B itself. It is this definition of Y that is used for the computational tests of Section 5.

3.7. Updating the reduced Hessian

Changes in the working set cause changes to Z in the following situations:

1. *A bound is deleted from the working set and added to S.*
2. *A variable corresponding to a column in S hits a bound.*
3. *A variable corresponding to a column in B hits a bound.*

These changes to Z in turn cause changes to the reduced Hessian H_z. To ease computation only the Cholesky factor of the reduced Hessian is recurred (instead of H_z). This section describes how the Cholesky factorization $H_z = R^TR$ may be maintained. The updates to R are the same as in MINOS [MurS78] except in the first case, where the new column of R is not open to choice.

3.7.1. Updates to H_z arising from deletion of a bound

When a new variable is added to the superbasic set, the reduced-gradient form of Z is unaltered except for gaining a new column: $\bar{Z} = \begin{pmatrix} Z & z \end{pmatrix}$. The new reduced Hessian is given by

$$\bar{H}_z = \bar{Z}^T H \bar{Z} = \begin{pmatrix} Z^THZ & Z^THz \\ z^THZ & z^THz \end{pmatrix}. \tag{3.7.1}$$

Let $\bar{R}$ denote the new Cholesky factor. It follows from (3.7.1) that

$$\bar{R} = \begin{pmatrix} R & r \\ & \delta \end{pmatrix}. \tag{3.7.2}$$

Hence, $\bar{R}$ can be obtained by the following operations:

$$\text{form} \quad w = Hz; \tag{3.7.3}$$

$$\text{form} \quad v = Z^Tw; \tag{3.7.4}$$

$$\text{solve} \quad R^Tr = v; \tag{3.7.5}$$

$$\text{form} \quad \delta = \sqrt{z^Tw - \|r\|^2}. \tag{3.7.6}$$

Since $\bar{H}_Z$ is positive definite, δ is well defined.

3.7.2. Consequences of an early change of subspace

In our discussion of Section 3.2 we assumed that the minimizer would be found on the current subspace before pricing. This is not always practical (for some of the same reasons that lead us to consider an early-termination strategy for subproblems). Optimization algorithms such as MINOS may terminate the minimization on a subspace when the norm of the reduced gradient is smaller than a dynamic tolerance. As in the case of early termination, this may help to avoid unnecessary computation when far from the optimal active set. Unfortunately, if H_Z is expanded as in (3.7.1) there is no guarantee that the search direction resulting from the solution of the Newton equations will provide a feasible descent direction for the QP iteration. This assertion is reflected in Lemma 3.7.1.

Lemma 3.7.1. *If a Lagrange multiplier estimate arising during the QP subproblem is used to delete a constraint from the working set, the search direction arising from the solution of the Newton equations*

$$\bar{R}^T\bar{R}p_Z = -\bar{Z}^T g_{QP} \tag{3.7.7}$$

may not be a feasible descent direction for the QP subproblem, unless $Z^T g_{QP} = 0$ before the constraint is deleted.

Proof. The proof is given in [GilM79]. ∎

Despite this result it may still be worth pricing before finding a minimizer. In the event that the search direction is not a feasible direction of descent we can simply revert to minimizing on the current (smaller) subspace.

The use of *multiple pricing*, i.e. choosing more than one variable to become superbasic, causes greater difficulty, since it is a combinatorial problem to identify the subset of variables that prevent the search direction from being feasible. This problem does not arise in MINOS since the expanded Cholesky factor of the reduced Hessian becomes

$$\bar{R} = \begin{pmatrix} R & \\ & I \end{pmatrix}. \tag{3.7.8}$$

Clearly, the new search direction is always feasible with respect to the bounds on the new superbasic variables. A strategy of expanding the reduced Hessian using (3.7.8) is acceptable for the optimality-phase algorithm when the update is performed outside the subproblem. The strategy is unacceptable inside the subproblem since the relation

$$\bar{R}^T\bar{R} \equiv \bar{Z}^T H \bar{Z} \tag{3.7.9}$$

would no longer hold. The result of this discrepancy is that if a unit step is taken in the QP then in general $\bar{Z}^T g_{QP} \neq 0$ (as would be the case if equation (3.7.9) held). Since the reduced Hessian approximation defines the full QP Hessian, when (3.7.9) is violated we are no longer solving the "correct" subproblem. When this occurs it is necessary to either modify the definition of the QP Hessian to satisfy (3.7.9) or introduce new linesearch criteria and termination conditions for the subproblem to ensure that the resulting search

direction is a descent direction for the merit function. In the computational tests of Section 5, a conservative strategy was adopted and the subspace was not changed until the norm of the reduced gradient (for the subproblem) was quite small. To be precise, minimization on the subspace was continued in the subproblem until

$$\|Z^T g_{QP}\|_\infty \leq \sqrt{\epsilon}\|\mu\|_\infty,$$

where ϵ is machine precision.

3.7.3. Updates to H_z arising from changes to S

If the q-th superbasic variable hits a bound, the new null-space matrix is obtained by removing the the q-th column from S. The new Cholesky factor is updated by applying plane rotations to R followed by the removal of its q-th row and column (see [GGMS74]).

3.7.4. Updates to H_z arising from changes to B

If a variable corresponding to the p-th column in B hits a bound, the updating is performed in two stages. First, the basis is updated by *replacing* the p-th column in B with the q-th column from S, where q is chosen to keep B well-conditioned. (The procedure requires a solve with B^T.) Finally, we delete the chosen column from S as described in Section 3.7.3.

Let $\bar{B}$ denote the new basis and $\bar{Z}$ the corresponding null-space matrix. We shall show that $\bar{Z}$ has the form

$$P\bar{Z} = ZM, \tag{3.7.10}$$

where P is a permutation matrix and M is a rank-one modification of the identity. This expression for $\bar{Z}$ enables the Cholesky factor of $\bar{Z}^T H \bar{Z}$ to be determined easily from that of $Z^T H Z$.

Let a denote the p-th column of B, and suppose it is exchanged with s, the q-th column of S. Also, let $B^T r = e_p$ and $\theta = r^T s$. As in [MurS78], q is chosen by first forming $y = S^T r$ and to some extent maximizing the "pivot element" $y_q = \theta = r^T s$. We have $a = Be_p$, and the updated basis $\bar{B}$ is given by

$$\bar{B} = B + (s - a)e_p^T = (I + (s - a)r^T)B, \tag{3.7.11}$$

so that

$$\bar{B}^{-1} = B^{-1}(I - \phi(s - a)r^T), \tag{3.7.12}$$

where $\phi = 1/\theta$. The change in S (before deleting the q-th column) is

$$\bar{S} = S - (s - a)e_q^T. \tag{3.7.13}$$

Lemma 3.7.2. *When the basis has been updated as in (3.7.11) and (3.7.13), then $\bar{Z}$ is of the form $P\bar{Z} = ZM$, where*

$$M = I + e_q v^T \qquad \text{and} \qquad v = -\phi(S^T r + e_q). \tag{3.7.14}$$

Proof. Define $\tilde{Z} \equiv ZM$. The top $m \times n_S$ submatrix of $\tilde{Z}$ after swapping the q-th column of S with the p-th column of B is

$$\begin{aligned}
\bar{B}^{-1}\tilde{S} &= B^{-1}(I - \phi(s-a)r^T)(S - (s-a)e_q^T) \\
&= (B^{-1} - \phi B^{-1}(s-a)r^T)(S - se_q^T + ae_q^T) \\
&= B^{-1}S - \phi B^{-1}sr^TS + \phi e_p r^T S - \phi B^{-1}se_q^T + \phi e_p e_q^T \\
&= B^{-1}S\Big(I - \phi e_q(r^TS + e_q^T)\Big) + \phi e_p(r^TS + e_q^T) \\
&= B^{-1}SM - e_p v^T.
\end{aligned}$$

Clearly, the result is satisfied for all rows except the p-th:

$$e_p^T\bar{B}^{-1}\tilde{S} = -e_q^T - v^T,$$

which should be e_q^T after the update. To make the update complete we swap the p-th row with the $(m+q)$-th row (which is e_q^T), using the permutation matrix

$$P_{p,m+q} = (e_1 \ \ldots \ e_{p-1} \ e_{m+q} \ e_{p+1} \ \ldots \ e_{m+q-1} \ e_p \ e_{m+q+1} \ \ldots \ e_{m+n_s}). \tag{3.7.15}$$

Premultiplication of $\tilde{Z}$ by $P_{p,m+q}$ gives

$$P_{p,m+q}\tilde{Z} = P_{p,m+q}\begin{pmatrix} -\bar{B}^{-1}\tilde{S} \\ I \\ 0 \end{pmatrix} = Z\Big(I - \phi e_q(r^TS + e_q^T)\Big). \tag{3.7.16}$$

The permuted update satisfies $e_p^T\bar{Z} = e_q^T$. The lower $n_S \times n_S$ portion of $\bar{Z}$ is

$$I - e_q\Big(\frac{r^TS + e_q^T}{\theta}\Big), \tag{3.7.17}$$

which is the identity matrix except that the q-th row which is the p-th row of $-\bar{B}^{-1}\tilde{S}$. Finally, $\bar{Z}$ is constructed to have one less column than Z by having its q-th column removed and its $(m+q)$-th row zeroed out and permuted to the bottom. ∎

3.7.5. Updating the Cholesky factor of H_z

The changes to Z must be reflected in updates to the Cholesky factor of H_z. We have

$$\bar{H}_z = M^TH_zM = (R^T + vu^T)(R + uv^T), \tag{3.7.18}$$

where $u = Re_q$ is the q-th column of R and $v = -\phi(S^Tr + e_q)$. The updated factor $\bar{R}$ is obtained by reducing $R + uv^T$ to upper triangular form with two partial sweeps of plane rotations as in [GGMS74, MurS78]. Finally, because a superbasic column was used to replace a column from B, the number of superbasic variables is reduced, and a row and column are removed from $\bar{H}_z$ (and its Cholesky factor) as described in Section 3.7.3.

Chapter 4

The Quasi-Newton Update to the Reduced Hessian

In the prototype algorithm, we assume that the QP Hessian H approximates W, the Hessian of the Lagrangian. The relationship between W and H and computational details of updating approximations to the *reduced* Hessian of the Lagrangian are the concerns of this section.

4.1. Introduction

Quasi-Newton methods are "Newton-like" algorithms in which the Hessian (or classically, its inverse) is replaced by an approximation. The approximation is obtained by using the known curvature along the directions of search. Most quasi-Newton methods are based on a formula from the one-parameter family of updates introduced by Broyden [Bro67], for unconstrained optimization. Define $\phi \in [0,1]$, $s \equiv \bar{x} - x$ and $y \equiv \bar{g} - g$. The one-parameter family of Broyden updates is then

$$\bar{H} = H - \frac{Hss^TH}{s^THs} + \frac{yy^T}{y^Ts} + \phi ww^T, \tag{4.1.1}$$

where

$$w \equiv (s^THs)^{\frac{1}{2}} \left(\frac{y}{y^Ts} - \frac{Hs}{s^THs} \right). \tag{4.1.2}$$

The choice of the factor $\phi = 1$ gives the classical DFP formula, while the choice of $\phi = 0$ gives the BFGS formula. At least in one sense, the choice of ϕ is not critical. Dixon [Dix72a, Dix72b] showed that when the iterates are chosen to satisfy

$$Hp = -g, \tag{4.1.3}$$

$$\bar{x} = x + \alpha p, \tag{4.1.4}$$

where α is the step that minimizes $F(\alpha) \equiv F(x + \alpha p)$, the Broyden updates generate identical iterates *independent of the choice of* ϕ.

Despite this result, the performance of quasi-Newton methods does depend significantly on the choice of ϕ because it is inefficient in practice to perform accurate line-

searches. Moreover, even the effort to perform the linesearch depends on ϕ. For unconstrained problems, $\phi = 0$ (the BFGS update) has been shown to be a good choice [GMP72].

Recently there has been considerable interest in the rank-one update (see [KBS90] and [CCT91]). Because our interest is in exploring the differences between our large-scale SQP approach (using only an reduced Hessian approximation) and NPSOL, we did not wish to make use of a different update. We have opted to use the BFGS update since this is the one used in NPSOL.

4.1.1. The BFGS update

The pedagogical form of the BFGS update for unconstrained optimization is given by

$$\bar{H} = H - \frac{Hss^TH}{s^THs} + \frac{yy^T}{y^Ts}. \tag{4.1.5}$$

If H is positive definite, the new approximation $\bar{H}$ is positive definite if and only if $y^Ts > 0$. For unconstrained optimization, we can always ensure $y^Ts > 0$ by choosing appropriate termination conditions for the linesearch (see [GMSW79]). For nonlinearly constrained optimization, however, $y^Ts > 0$ is no longer assured.

4.2. Quasi-Newton updates for NLP

A number of authors have proposed SQP algorithms for NLP using H as a quasi-Newton approximation to W. This idea is credited to Murray who proposed it in [Mur69]. It has been used with success by others (see [Han76], [Pow83], or [GMSW86b] for example). Complications arise because of the constraints; for example:

1. The BFGS update as well as many quasi-Newton approximation methods are dependent on the approximation H being positive definite, yet it is not necessary (nor is it expected) that the exact Hessian W be positive definite, even at the solution.

2. Both W and H require estimates of the Lagrange multipliers, which themselves are nonlinear functions of x.

3. We can no longer ensure that $y^Ts > 0$.

In addition to these difficulties, large-scale SQP methods that approximate the full Hessian of the Lagrangian may be computationally intractable since the Broyden updates do not preserve the sparsity of the true Hessian. While the true Hessian may be quite sparse, its approximation using (4.1.1) is almost always completely dense. Thus, as problem size grows, the storage and effort required to perform the update may become enormous.

In the case of unconstrained optimization, attempts have been made to exploit sparsity in quasi-Newton methods for problems whose Hessian has a known sparsity pattern. Unfortunately, results for these methods have not been promising (see [Tha83] and [Sha80] for example).

To preserve sparsity and still satisfy the quasi-Newton condition (see Section 4.2.2) a significant amount of time may be required to perform the linear algebra defining a suitable

sparse update. In addition, a positive-definite $\bar{H}$ cannot be guaranteed and superlinear convergence is not in general achieved.

To preserve sparsity and still satisfy positive definiteness in $\bar{H}$, it is possible to construct an update in which the fill-in (according to the sparsity pattern of the true Hessian) is ignored. Unfortunately, satisfaction of the quasi-Newton condition cannot be guaranteed and superlinear convergence is not in gene achieved.

Recently, Nickle and Tolle [NicT89] have attempted to use sparsity-exploiting quasi-Newton updates within an SQP method for constrained problems. They use the BFGS update for the approximation to the full Hessian of the Lagrangian and maintain a Cholesky factorization of the approximation that ignores the fill-in associated with the standard update. In their implementation, they do not enforce quasi-Newton condition. The success of this approach for large-scale NLP has not been verified as the method has only been tested on small problems (the largest problem in their test set having 60 variables, 40 linear constraints and 10 nonlinear constraints).

4.2.1. An approximation to the reduced Hessian

The poor performance of late of sparsity-exploiting methods for approximating the full Hessian leads us to explore alternatives for large-scale NLP.

Consider the use of a quasi-Newton approximation to the reduced Hessian Z^TWZ. This approach for NLP, proposed by Murray and Wright [MurW78], takes advantage of the property that $Z^{*T}W^*Z^*$ is positive semidefinite. This result combined with the computational expense of approximating all of W makes it unreasonable to use a positive-definite approximation to W for large-scale problems.

As discussed in Section 2, to ensure two-step superlinear convergence of the prototype algorithm we have assumed that our approximation to the Hessian is accurate on the null space of the active constraints. Specifically, we assume the approximation to the reduced Hessian satisfies

$$\|Z_k^T(H_k - W_k)Z_k p_{z_k}\| = o(\|p_k\|), \tag{4.2.1}$$

where W_k denotes the Hessian of the Lagrangian at x_k. Although (4.2.1) cannot be verified computationally, our goal is to approximate this condition by using a positive-definite approximation to the reduced Hessian at the initial point, followed by a quasi-Newton update to the approximation at the end of each major iteration. The update must satisfy the following minimal requirements:

- The new reduced Hessian approximation, $\bar{H}_z = \bar{Z}^T\bar{H}\bar{Z}$, must be positive definite.
- The "reduced" quasi-Newton condition must be satisfied (see next section).

In addition, there may be special cases to be considered. For example, when all of the NLP constraints are linear, Lagrange multiplier estimates are not required to update the Hessian approximation. This is not generally the case for nonlinearly constrained problems but curiously, one form of the BFGS update for the reduced Hessian does not require Lagrange multiplier estimates, which clearly circumvents any difficulties arising from such estimates being poor. The precise form of the updates are discussed in the following sections.

4.2.2. The quasi-Newton condition

In the unconstrained case, the quasi-Newton condition may be written as

$$Hs = y, \tag{4.2.2}$$

where $s \equiv \bar{x} - x$ and $y \equiv \bar{g} - g$. Likewise, for the linearly constrained case, the quasi-Newton condition for the reduced Hessian is

$$H_Z s_Z = y_Z, \tag{4.2.3}$$

where $s_Z = Z^T(\bar{x} - x)$ and $y_Z = Z^T(\bar{g} - g)$. Note that no additional gradients are necessary to compute y_Z.

It can be shown that the reduced Hessian obtained in the linearly constrained case is identical to the matrix obtained by updating the full matrix and then forming the reduction. Unfortunately, this property is no longer true for the case of nonlinear constraints. The difference is due to the fact that the step s taken is not of the form $s = Zs_Z$. It is no longer clear what the definition of s_Z and y_Z should be. An approach adopted by Coleman and Conn [ColC84] is to solve $H_Z s_Z = -Z^T g$, evaluate $g(x + Zs_Z)$ and then form the "correct" y corresponding to this intermediate step. This requires an additional gradient evaluation per iteration. Other candidates for s and y include

$$s = Z^T(\bar{x} - x), \quad \text{or} \tag{4.2.4}$$
$$s = \bar{Z}^T(\bar{x} - x), \quad \text{or} \tag{4.2.5}$$
$$s = \bar{Z}^T\bar{x} - Z^T x, \quad \text{and} \tag{4.2.6}$$
$$y = Z^T(\bar{g} - \bar{A}^T\lambda - g), \quad \text{or} \tag{4.2.7}$$
$$y = \bar{Z}^T(\bar{g} - g + A^T\lambda), \quad \text{or} \tag{4.2.8}$$
$$y = \bar{Z}^T\bar{g} - Z^T g. \tag{4.2.9}$$

The motivation for these choices for s are straightforward. For y, note that (4.2.7)–(4.2.8) are transformations of $y_L = \nabla_x L(\bar{x}, \lambda) - \nabla_x L(x, \lambda)$ under either Z or $\bar{Z}$. Equation (4.2.9) presents a practical choice of y that does not need multiplier estimates. An additional candidate for s is analogous to the one in MINOS [MurS87], namely

$$s = \alpha p_S, \tag{4.2.10}$$

where α is the steplength from the linesearch (which is used each *minor* iteration to reduce the augmented Lagrangian objective). For the prototype SQP algorithm, α is taken as the steplength from the merit function linesearch (performed each *major* iteration). The motivation for this choice of s arises from the fact that when the correct active set has been identified, each QP subproblem is solved in a single iteration. The superbasic component of the search direction is found by solving

$$H_Z p_S = -Z^T g_{QP}, \tag{4.2.11}$$

where $g_{QP} \to g$ as $p \to 0$ near the solution. Prior to the computational tests described in Section 5 each of these cases for y and s was tested. A decision based on the tests was then made to use (4.2.10) for s and (4.2.9) for y.

Because NLP is an inequality constrained problem, the number of superbasic variables usually changes between major iterations. Thus, Z_k and Z_{k+1} (and hence H_z and $\bar{H}_z$) may not be the same size. It would seem that s and y may not be well defined. Fortunately, this is not a difficulty. During the solution of the QP subproblem the basis B_k, the reduced Hessian H_z and the index set of superbasic variables are updated each minor iteration. As a result the set of superbasics and the size of the reduced Hessian are the same at the *completion* of the last major iteration as at the *start* of the next major iteration. The quasi-Newton update takes place at the start of the new major iteration and can be completed without excessive expense. The key idea is to ensure that the reduced gradient $Z^T g$ used in the update corresponds to the Z at the *end* of the last major iteration.

4.3. Modifications to the BFGS update

At times it may not be possible to use the standard BFGS update for the reduced Hessian without encountering numerical difficulties. Subsequent sections discuss modifications to the BFGS update for the following cases:

- When $y^T s \leq 0$.
- When $\|s\|$ is "small".
- When $y^T s > 0$ but $y^T s$ is "small".

4.3.1. When $y^T s \leq 0$

Normally, when $y^T s \leq 0$ it is necessary to skip the BFGS update to avoid indefiniteness or singularity in the new approximation. One modification of the BFGS update for the full Hessian is due to Powell [Pow78a] and attempts to perform the update under conditions when $y^T s \leq 0$. Let $\eta \in [0, 1]$. The Powell modification defines the new approximation as

$$\bar{H} = H - \frac{H s s^T H}{s^T H s} + \frac{d d^T}{s^T d}, \tag{4.3.1}$$

where $d \equiv \theta y + (1 - \theta) H s$ and

$$\theta \equiv \begin{cases} 1 & \text{if } y^T s \geq \eta s^T H s, \\ (1 - \eta) s^T H s / (s^T H s - y^T s) & \text{otherwise.} \end{cases} \tag{4.3.2}$$

This modification ensures that $\bar{H}$ is positive definite and the determinant of $\bar{H}$ is no less than η times the determinant of H. For computational purposes, Powell sets $\eta = 0.2$.

The priorities are different when updating an approximation to the reduced Hessian. It may be that the search direction lies (almost) entirely in the range space of $\hat{A}$. In such circumstances there is no point in updating H_z (since there is no new information). This idea is reinforced by the results of Coleman and Fenyes ([ColF88], page 11, Corollary 3.6), who give updates for approximations to $Z^T W Z$ and $Z^T W Y$ and show there instances when $y^T s \leq 0$ leads to updating only the approximation to $Y^T H Y$ (while skipping the update of $Z^T H Z$). For this reason the Powell update was not adopted and updates were skipped when $y^T s \leq 0$.

4.3.2. When $\|s\|$ is small

A further problem with the standard BFGS update arises when $\|s\| \ll \|y\|$. If this occurs, the modification of Powell (4.3.1)–(4.3.2) may not help. To see why, note that it is possible to construct examples where $\|s\| \ll \|y\|$ and $y^T s \gg s^T H s$. Hence, the update using (4.3.2) may make the new $\bar{H}$ nearly singular.

As discussed in [Gur87], the method of Coleman and Conn [ColC84] may be interpreted as a method that prevents these difficulties when $\|s\|$ is small. They define an intermediate point $\hat{x} \equiv x + Zp_z$, obtained by first solving $H_z p_z = -Z^T g$. They use the standard BFGS update (4.1.5) with an orthogonal Z and with s and y defined as

$$\begin{aligned} s &= Z^T(\hat{x} - x) && (4.3.3)\\ &= -{H_z}^{-1} Z^T g, && (4.3.4)\\ y &= Z^T(\nabla_x L(\hat{x}, \lambda) - \nabla_x L(x, \lambda)) && (4.3.5)\\ &= Z^T(\hat{g} - \hat{A}^T\lambda) - Z^T g, && (4.3.6) \end{aligned}$$

which ensures that $\|s\|$ is not too much smaller than $\|y\|$. Using this method they were able to prove two-step superlinear convergence of an SQP algorithm. As pointed out in Section 4.2.2, this approach requires an additional set of gradient evaluations.

Nocedal and Overton [NocO85] have used an alternative strategy that does not require extra gradients. Their method is based on skipping the update when $\|s\|$ is small relative to a range-space projection of $(\bar{x} - x)$. They define $v \equiv Y^T(\bar{x} - x)$ and perform the update (4.1.5) only if

$$\|v\| < (\omega/(k+1)^{1+\nu})\|s\|, \qquad (4.3.7)$$

where ω and ν are positive constants and k corresponds to the current iteration. The authors show two-step superlinear convergence of an SQP algorithm under various conditions. For their computational tests they used $\omega = 1$ and $\nu = 0.01$. This strategy was not adopted for the implementation of the prototype SQP algorithm. Instead, we use a *self-scaled* update, as described next.

4.3.3. The self-scaled BFGS update

When $y^T s$ is "small" we could skip the update to prevent the updated approximation from being close to singularity. This is not an ideal strategy since some change in the curvature of the Lagrangian generally takes place. To avoid skipping the update we may employ the *self-scaled quasi-Newton update* [Ore74]. This is exactly the same as the standard BFGS update except that the current approximation is scaled by a dynamic factor γ:

$$\bar{H} = \gamma H - \gamma \frac{H s s^T H}{s^T H s} + \frac{y y^T}{y^T s}, \qquad (4.3.8)$$

As described in Brodlie [Bro77], the self-scaled update exhibits the property of a monotonically decreasing condition number for $\bar{H}$ provided the scaling factor γ is chosen to satisfy

$$\gamma = \beta \frac{y^T s}{s^T H s} + (1 - \beta)\frac{y^T H^{-1} y}{y^T s}, \qquad (4.3.9)$$

for $\beta \in [0,1]$. If $\beta = 1$ then $\gamma = y^T s / s^T H s$. With this choice of γ, (4.3.8) has the property of correcting the curvature along s *before performing the update* as well as after the update:

$$s^T(\gamma H)s = (y^T s / s^T H s) s^T H s = y^T s \quad \text{and} \tag{4.3.10}$$

$$s^T \bar{H} s = y^T s. \tag{4.3.11}$$

Other choices for ϕ and γ have been studied by Brodlie [Bro77] to maintain well-conditioned approximations to the Hessian. Oren and Spedicato [OreS76] study different choices for ϕ and γ and give optimal choices that minimize a sharp bound on the condition number of the inverse Hessian approximation at each iteration. For the computational tests described in Section 5 we use the standard BFGS method (4.1.5) for the reduced Hessian when

$$y^T p \geq (1 - \eta) p^T H p, \tag{4.3.12}$$

where y and s are defined by (4.2.9) and (4.2.10) respectively, and η is a linesearch parameter (typically $\eta = 0.9$). When (4.3.12) is not satisfied but $y^T s$ is positive, the self-scaled BFGS update (4.3.8) is performed with $\gamma = y^T s / s^T H s$.

4.3.4. Updating the Cholesky factors of the reduced Hessian

In classical implementations of quasi-Newton methods, the *inverse* approximation H^{-1} was updated at each iteration. Although convenient, this technique may create serious numerical difficulties. For example, due to rounding error in just one iteration, all subsequent approximations to the inverse Hessian may be indefinite. Unfortunately, it is not generally possible to determine if the approximation is singular or indefinite by a simple examination of the matrix itself.

In the prototype algorithm the system of equations $H_z p_z = -Z^T g$ must be solved to determine the superbasic search direction. A reliable and convenient approach is based on using the Cholesky factors of H_z, as developed by Gill and Murray [GilM72] for unconstrained optimization. The method has the advantage of being able to detect easily a singular or indefinite approximation to the Hessian. In addition, there is no penalty for maintaining an approximation to the Hessian rather than its inverse. Let $H_z = R^T R$. To obtain the search direction we solve $R^T w = -Z^T g$ and $R p_z = w$ using forward and back substitution.

For the computational tests of Section 5 we have implemented the prototype algorithm using updates to the Cholesky factors of the reduced Hessian. As described in Dennis and Schnabel [DenS83] for unconstrained optimization, the update to H_z (4.1.5) is a rank-two modification to $R^T R$, reflected in a rank-one update to R itself:

$$\bar{R} = R + \frac{(\alpha R s)(y - \alpha H_z s)^T}{y^T s}, \tag{4.3.13}$$

where

$$\alpha \equiv \sqrt{\frac{y^T s}{s^T H_z s}}. \tag{4.3.14}$$

The factor $\bar{R}$ is returned to upper triangular form using plane rotations as described in [GGMS74].

4.3.5. The update for self-scaled factors

When the self-scaled BFGS update (4.3.8) is used, the update to the Cholesky factors has the form

$$\bar{R} = \eta R + \frac{(\alpha Rs)(y - \alpha\eta H_z s)^T}{y^T s}, \tag{4.3.15}$$

where $\eta = \alpha$ and α is given by (4.3.14).

Chapter 5

Computational Results

In this chapter numerical results obtained from a sparsity-exploiting implementation of the prototype SQP algorithm (Algorithm 2.3.1) are given. The tests consist of solving two sets of test problems using the new algorithm and comparing the results with those of MINOS and NPSOL. The first set of problems are from the literature and are all dense and relatively small. The second set of problems are sparse optimal control problems. The purpose of the testing is to demonstrate the large-scale SQP algorithm's strengths and weaknesses.

5.1. Implementation

The implementation, hereafter referred to as LSSQP, has been written as a major modification of the mathematical programming system MINOS. For a description of MINOS see [MurS78, MurS82] and [MurS87].

Many features and modules of the Fortran code in MINOS were used in LSSQP, including:

- MPS data handling and hashing routines. These routines are used to read in the data corresponding to constraint rows and columns, coefficients for linear constraints (if any), right-hand side values for constraints, and bounds and initial values for the variables.
- Basis-handling routines for factorizing and updating the LU factors of the basis B (routines from the LUSOL package as described in [GMSW87]).
- The pricing routine used to calculate reduced costs for nonbasic variables.
- The linear search routine used to calculate the step to the nearest bound in the QP subproblem.
- The nonlinear search routine used to find a steplength α that sufficiently reduces the augmented Lagrangian merit function.
- Routines for updating the Cholesky factors of the reduced Hessian following changes in the working set.

Because LSSQP and MINOS share many of the same routines, the differences in the computational performance of the two methods are due to the basic approach. The main features of LSSQP and some of the differences between SQP methods (LSSQP and NPSOL) and MINOS are discussed in the following sections.

5.1.1. Form of the subproblem

For problems with nonlinear constraints, MINOS uses a sequential linearly constrained (SLC) method and evaluates the functions and gradients during each minor iteration. NPSOL and LSSQP solve quadratic programming subproblems (which have the same linearized constraints) but evaluate nonlinear functions and gradients only after termination of a subproblem (as part of each major iteration).

SLC and SQP methods deal with the constraint linearizations the same way. The subproblems differ in their objective functions. In general, evaluating a QP objective should be less expensive than evaluating a general nonlinear objective. This should give an advantage to an SQP method over a Lagrangian method. However, for some simple problems this advantage may be negated by the expense of evaluating the QP gradient. For SQP methods, most of the expense associated with the QP gradient comes from the matrix-vector multiplication with the QP Hessian (i.e. forming Hp). While NPSOL recurs an approximation to the full Hessian of the Lagrangian (so the matrix-vector product is straightforward), LSSQP maintains an approximation on a subspace and therefore requires a more complex approach to obtain the full QP gradient (see Section 3.5).

Since the subproblem solved in MINOS has a general nonlinear objective, one hypothesis is that the number of iterations required to solve a subproblem is likely to be greater than the number required by NPSOL or LSSQP.

5.1.2. Quasi-Newton updates

In MINOS, the gradient of the objective is evaluated each minor iteration. With this information available, a quasi-Newton update to the reduced Hessian is performed each minor iteration. In SQP methods, the gradient of the objective is evaluated before the beginning of a *major* iteration. As a result, a quasi-Newton update to the reduced Hessian is performed only once per major iteration. With fewer opportunities to update the reduced Hessian, a hypothesis is that LSSQP and NPSOL may require more major iterations than MINOS. As already noted, we may expect SQP methods to require fewer minor iterations *per major iteration* than methods such as MINOS, but it is not clear which approach is likely to require fewer *total* minor iterations.

5.1.3. Basis refactorization

The *initialization* of a major iteration consists of setting up the subproblem to be solved. Part of this process involves linearization of the nonlinear constraints (if any). A basis B is then formed and factorized. The cumulative effort to factorize the basis at the start of each major iteration may constitute a large percentage of the overall expense of solving NLP. Because of this, methods requiring few total minor iterations may not be efficient if they require many major iterations.

SQP methods such as LSSQP should prove to be more efficient on large sparse problems if the average effort to factorize the basis (for each subproblem) is small compared to the average effort to evaluate functions and gradients *during* solution of the subproblem.

5.1.4. Termination conditions

Let δ_{opt} be an optimality tolerance and δ_{fea} be the feasibility tolerance for nonlinear constraints (typically, $\delta_{\text{opt}} = 10^{-6}$ and $\delta_{\text{fea}} = 10^{-6}$). Let p be the search direction obtained from the last QP subproblem. In all, five criteria must be satisfied if a point x is to be considered optimal for NLP:

$$\alpha\|p\| \leq \delta_{\text{opt}}\,(1 + \|x\|), \tag{5.1.1}$$

$$\|(c - s)\|_\infty \leq \delta_{\text{fea}}, \tag{5.1.2}$$

$$\|Z^T g\|_\infty \leq \delta_{\text{opt}}\,(1 + |F|), \tag{5.1.3}$$

$$\text{sign}(u_j - x_j)\eta_j \geq -\delta_{\text{opt}}\,(1 + \max(|F|, \|g_N\|_\infty), \quad j \text{ nonbasic}, \tag{5.1.4}$$

$$\text{sign}(x_j - l_j)\eta_j \leq +\delta_{\text{opt}}\,(1 + \max(|F|, \|g_N\|_\infty), \quad j \text{ nonbasic}, \tag{5.1.5}$$

where g is the gradient of F, g_N is the *subproblem* gradient of the nonbasic variables and $\eta_j \equiv (g_{QP})_j - \pi^T a_j$.

The termination conditions (5.1.1)–(5.1.3) are similar to those in NPSOL [GMSW86a]. Conditions (5.1.4)–(5.1.5) are evaluated only at feasible points for the subproblem and correspond to having no nonbasic variable with a nonoptimal Lagrange multiplier (of a significant magnitude). This condition is tested as a by-product of pricing (Procedure 3.2.1). If a minimizer has been reached on the current subspace and the pricing procedure finds no nonbasic variable with a suitable reduced cost, the current p is a solution for the subproblem. Note that when a subproblem is terminated early, these conditions will not be satisfied. Also, a dynamic tolerance (which is gradually reduced to δ_{opt}) is optionally used to check for optimal multipliers. Early subproblems may therefore be solved to only moderate accuracy. Termination condition (5.1.3) differs from (5.1.4)–(5.1.5) in that it checks whether the norm of the reduced gradient *for NLP* is small enough.

For the experimental results we defined the optimality tolerance $\delta_{\text{opt}} = 10^{-6}$ for all problems unless specified in the *"Comment"* section of Tables 7 and 8. The termination conditions for MINOS are described in [MurS87]. In general, all methods (NPSOL, MINOS, and LSSQP) obtained solutions to similar accuracy on all test problems solved. On many problems the convergence is fast, so that small differences in the accuracy of the methods are not significant.

5.1.5. Testing Environments

Section 5.3 gives numerical results for 92 test problems described in Section 5.2. These problems are sorted into two test sets. The first set (of smaller problems) was tested using a Digital Equipment Corporation VAXstation II with 9 megabytes of main memory. The operating system was VAX/VMS version 5.0. All Fortran files were compiled under VAX FORTRAN version 5.2-015 using the default options, including code optimization.

The second set (of larger problems) was tested using a Digital Equipment Corporation DECstation 3100 with 24 megabytes of main memory. The operating system was Ultrix version 4.1. Fortran files were compiled under full optimization for these tests.

5.2. Test problems

The test problems used in this section come from a variety of applications. There are two test problem sets described next.

5.2.1. The small test problems

The first set of test problems consists of 80 small problems ($n \leq 100$), whose names and statistics are given in Tables 7 and 8.

The first three columns give the problem number, problem name and *comments* such as the type of application, author, or alternative name of the test problem. Comments may also allude to a different starting point from one given in the literature or special features of the MINOS options file. Columns 4–7 give the number of variables for the problem, the number of linear constraints, and the number of nonlinear constraints. The final column gives the optimal (published) objective value.

Unless noted in the *Comment* section, the Jacobian for these problems is treated in a dense manner by MINOS and LSSQP. (NPSOL treats all problems as dense.) Likewise, unless noted, the initial starting points for the problems are the ones given in the published references.

These problems have been used to test the mathematical programming code NPSOL [GMSW86a] and many are considered to be difficult to solve. Several of the problems do not satisfy the assumptions in Section 2 that were used in the proof of convergence for the prototype algorithm. For example, in some cases the Jacobian at the solution has less than full row rank. Other problems do not satisfy the strict complementarity conditions or have infeasible subproblems. The test problems from the small set are taken from the following sources:

- Problem 1 is the sample test problem distributed with NPSOL. It is described in [GMSW86a].
- Problems 2, 6–9, 39–41 and 45–79 have the prefix *HS*. They correspond to the same numbered problems from Hock and Schittkowski [HocS81]. The upper bound for x_3 in problem number 52 (HS 70) was changed from 1.0 to 0.9999, as otherwise MINOS would evaluate the objective at a singularity. The modification does not change the optimal solution.
- Problems 3 and 10 have the prefix *KS*. They correspond to the same numbered problems from Schittkowski [Sch87].
- Problems 4–5 and 16 are described in [MurS82]. Problems 4–5 correspond to the problems *Wright No. 4* and *Wright No. 9*. The starting points for these two problems are from point (d) of the reference.
- Problems 11–15, 17, 32–38, 42–44 and 80 are from Prieto [Pri89]. Problem 36 is solved again as problem 37 with an alternate starting point of $x_0 = (0.097, 0.063)$.
- Problems 18–21 are from Fraley [Fra88].
- Problems 22–31 are from Boggs and Tolle [BogT84].

It should be noted that the functions and gradients for the small problems are usually very cheap to evaluate. Many of the problems have been chosen to be test problems for precisely this reason, but historically it has been assumed that the efficiency of an algorithm is measured by the number of function and gradient evaluations required to find a minimizer. We will be concerned with these measures of efficiency as well as others such as total CPU time required to solve the test set.

Many of the small test problems have multiple local minimizers. As shown in Tables 9–16, different methods (NPSOL, MINOS or LSSQP) may converge to different minimizers.

Fourteen of the small test problems contain only linear constraints: problems 10, 12–14, 24, 43–44, 47–49, 61, 76–77 and 80. On such problems MINOS requires only a single major iteration. This is not true for the SQP methods. The differences between the approaches arise from the fact that the reduced Hessian approximation is updated each *minor* iteration for MINOS and each *major* iteration for SQP methods. Thus, MINOS generally has more opportunities to perform the update. As a result, we would expect the performance of NPSOL and LSSQP to be similar on these linearly constrained problems (but to differ from that of MINOS).

5.2.2. Run-time parameters: Small test set

In MINOS, NPSOL and LSSQP the SPECS or options file sets various run-time parameters that describe the nature of the problem to be solved and the quality of the solution to be obtained. The options file must begin with the *keyword* "`BEGIN`". Each subsequent line of the options file contains one or more keywords and an associated value. For example, the line

```
Nonlinear Constraints    14
```

specifies that the problem has 14 nonlinear constraints. The last line of the options file is signified with the keyword "`END`". A full description of the MINOS and NPSOL options can be found in [MurS87] and [GMSW86a] respectively. The LSSQP solver maintains the use of all MINOS options as well as a few others, such as whether or not to use self-scaling for the BFGS update to the reduced Hessian. Except where noted in the *Comment* section of Tables 7–8, a uniform set of options was used for all runs. An example of the MINOS options file (for problem number 1) is given in Figure 1.

Problems 1–3 used the "`Jacobian Sparse`" option. Problems 4–80 used the "`Jacobian Dense`" option. Problems with more than 10 variables used a SPECS file that increased the major iterations limit to 300 and the total minor iterations limit to 1000. All other parameters were set to their default options. For the runs with NPSOL the default options were used (see [Pri89]).

5.2.3. The large test problems

The large test problems come from a class of applications known as *trajectory optimization.* All were generated using the system OTIS [HarP87], and their specifications are given in Table 1. A description of these sparse optimal control problems and their mathematical programming formulation is given in the Appendix. It is important to note that the OTIS function routines compute first derivatives by finite differences. Advantage is taken of the

```
BEGIN Hexagon (Sparse version)

   Problem Number            1
   Nonlinear Constraints    14
   Nonlinear Variables       9

   Iterations              300
   Major Iterations         50
   Print Level               0

   Jacobian             Sparse

END Hexagon
```

Figure 1. Sample SPECS file for small test problems

sparsity pattern of the Jacobian, but this is partly why the function/gradient evaluations are expensive. Also, the truncation error in the gradients makes it difficult to confirm "optimality" unless δ_{opt} is raised to about 10^{-4}.

No.	*Problem*	*Comment*	*n*	*Lcon.*	*Ncon.*	$F(x^*)$
1	F4 Min-time climb	6 Nodes	112	8	126	1.0280052864e+00
2	F4 Min-time climb	7 Nodes	130	8	150	1.0030551461e+00
3	F4 Min-time climb	8 Nodes	148	8	174	9.9471482986e−01
4	F4 Min-time climb	9 Nodes	166	8	198	9.9011108551e−01
5	F4 Min-time climb	10 Nodes	184	8	222	9.8758647160e−01
6	F4 Min-time climb	11 Nodes	202	8	246	9.8531969486e−01
7	F4 Min-time climb	12 Nodes	220	8	270	9.8519559525e−01
8	F4 Min-time climb	14 Nodes	256	8	318	9.8490191159e−01
9	F4 Min-time climb	15 Nodes	274	8	342	9.8347245192e−01
10	F4 Min-time climb NP	6 Nodes	112	8	104	1.0018209554e+00
11	F4 Min-time climb NP	15 Nodes	274	8	284	9.8165983233e−01
12	VTOL Descent		435	24	483	−1.6825846298e+00

Table 1. Large problem statistics.

5.2.4. Minimum time-to-climb problems

The first 9 large test problems are for a *minimum time-to-climb problem* [Bry69] for an F4 Phantom II supersonic interceptor. The aim is to find the pitch function to take the aircraft from sea level and Mach 0.34 to an altitude of 20 km ($\approx$ 65,617 ft) and Mach 1.0 in minimum time. The problems differ in the number of distinct time segments or *nodes* used to define the problem. The number of nodes varies from 6 to 15. In general, as the number of nodes increases, the model becomes more accurate and the optimal objective decreases but the problem becomes more difficult to solve.

Problems 10–11 correspond to the F4 minimum time-to-climb problem in which the pressure constraint has been omitted from the problem formulation. Although the number of constraints is fewer, the problem appears to be more difficult to solve.

Problem 12 is the the *VTOL Descent problem*, which finds the optimal descent trajectory for a vertical take-off and landing aircraft.

5.2.5. Run-time parameters: Large test set

Most of the standard MINOS run-time options were used. Special options for the large problems were contained in a single options file used for all large runs using either MINOS, NPSOL or LSSQP as the solver. The MINOS options file for these runs is given in Figure 2.

```
BEGIN OTIS  (Trajectory Problems)

   Major Iterations           500
   Minor Iterations           200
   Iterations               10000

   Linesearch Tolerance       0.5

   Row Tolerance          1.0E-05
   Function Precision     1.0E-10
   Optimality Tolerance   1.0E-04
   Feasibility Tolerance  1.0E-05

   Verify Level                -1
   Print Level                  0
   Hessian Dimension           50

   Partial Price                1
   Crash Option                 1

   Jacobian                Sparse
   Solution                    No

END OTIS
```

Figure 2. SPECS file for large test problems

When LSSQP was run with early termination of the subproblems, the parameters

```
Optlevel  Partial
Multiple  Price 5
```

were added to the MINOS options file. These options were *not* included in the MINOS runs

on the small test set or used in the NPSOL runs. The statement "`Optlevel Partial`" invokes the early-termination strategy and halts the solution of the subproblem after finding the first stationary point (i.e. when $\|Z^T g_{QP}\| \leq \delta_{\text{opt}}$). Invoking "`Multiple Price 5`" allows more than one nonbasic variable to be deleted from the working set during a single minor iteration. It is hoped that this will prevent the prototype algorithm from expending too many minor iterations on a nonoptimal working set.

5.3. Numerical results

In this section we compare the results from LSSQP with NPSOL [GMSW86a] (version 4.05) and MINOS [MurS87] (version 5.3). The purpose of the tests is to demonstrate the efficiency of the new algorithm for large sparse nonlinear optimization and show the following:

- The method is a practical alternative to the dense SQP method of NPSOL for large sparse problems.
- The method is a practical alternative to the Lagrangian method of the MINOS system for problems with functions and gradients that are expensive to compute.

Numerical result for the small test set are summarized in Table 2. Complete results for all test problems are given in Tables 3–16. The descriptive headings for the columns in the tables correspond to problem number and name, major iteration count, minor iteration count, total function evaluations, final objective value, maximum constraint violation (MINOS and LSSQP only), and solution time in CPU seconds. The final status of the solver is given in the last column of the tables. The notation is as follows:

Opt: An optimal solution was found (MINOS, NPSOL, LSSQP).

Fail: The algorithm failed to find an optimal solution (NPSOL).

Itr: The solver reached the limit on total major or minor iterations for the problem (MINOS, LSSQP).

Cbi: The final point did not satisfy the termination conditions but could not be improved upon (MINOS, LSSQP).

Note that "function evaluation" means computation of both the objective and constraint functions and their gradients.

5.3.1. Results for the small test set

Four runs were made on the small test set. One run each was made using MINOS (Tables 9 and 10) and NPSOL (Tables 11 and 12), and two runs were performed using the prototype algorithm LSSQP. The LSSQP runs differed only in whether the subproblems were solved to completion (Tables 13 and 14) or the early-termination strategy was used (Tables 15 and 16).

A comparison of the runs using LSSQP, NPSOL and MINOS on the small test set is summarized in Table 2. The column headings entitled *MINOS* and *NPSOL* are self-explanatory. *LSSQP-O* gives results for LSSQP in which QP subproblems were solved to optimality and *LSSQP-E* gives results for LSSQP when subproblems were terminated early.

Each of the three methods were similarly robust. The first three terminated successfully on 76 of the 80 problems. For all *solved* problems, NPSOL required the fewest function evaluations (1642 and 797 CPU seconds). MINOS required significantly more function evaluations (5876) but the least time (560 CPU seconds). LSSQP-O required 2658 function

Algorithm	*MINOS*	*NPSOL*	*LSSQP-O*	*LSSQP-E*
Problems attempted	80	80	80	80
No. Optimal	76	76	76	67
Total major iterations	646	1142	1493	1339
Total minor iterations	2477	2067	2802	2086
Total function evals.	5876	1642	2658	2489
Total Time	559.96	797.12	753.97	944.42

Table 2. Summary of small test problem results.

evaluations and 754 CPU seconds, while LSSQP-E (with the early-termination strategy) solved only 67 of the test problems and required 2489 function evaluations and 944 CPU seconds.

Note that two of our hypotheses of Section 5.1 are borne out by the results on the small test problems. MINOS required the fewest major iterations by about 2:1 over NPSOL and LSSQP. In addition, MINOS required more minor iterations *per major iteration*, while the total number of minor iterations was similar for all methods.

While the number of function evaluations is a salient measure of the efficiency of the methods tested, computational efficiency may also be measured by total solution time. NPSOL provided the fastest solution time for 48 out of the 80 problems, while MINOS proved to be fastest for 21 of the 80. The two LSSQP tests were fastest on only 11 problems. Note that the overall time for the 80 problems is somewhat misleading since two of the problems, namely numbers 11 and 80 (*OPF 30 BUS* and *Weapon*), required a disproportionate amount of solution time for all four methods. Deleting these problems gives the overall timing results as: NPSOL: 208 seconds, MINOS: 336 seconds, LSSQP-O: 399 seconds. LSSQP-E required 365 seconds to solve 65 of the 80 problems to optimality.

On the 14 problems with only linear constraints, MINOS required 1334 function evaluations and 133 CPU seconds, while NPSOL required 248 functions and 171 CPU seconds. LSSQP-O (LSSQP-E) required 392 (411) functions and 233 (223) seconds. MINOS required the fewest CPU seconds to find a minimizer for 8, NPSOL for 4 and LSSQP for 2 of the 14 problems.

As yet, the early-termination option is not as robust as solving the subproblems to optimality. This could be due to the fact that only the stationary point from the QP subproblem is used for the merit function (i.e. the method does not use an auxiliary search direction as does Prieto [Pri89], who provided more encouraging results using this strategy in a modification of NPSOL).

5.3.2. Results for the large test set

The large set (problems 81–92) were solved using MINOS, NZSOL and LSSQP. NZSOL is a version of NPSOL in which the QP subproblems are solved using QPSOL [GMSW83] instead of LSSOL [GHMSW86]. Like NPSOL, NZSOL is a dense SQP method, but has been modified to maintain a factorization of the reduced Hessian Z^THZ instead of the full (and dense) transformation Q^THQ. Hence, NZSOL is expected to outperform NPSOL on large

No.	Problem	Itns.	LC It.	Funs.	$F(x^*)$	Time	Stat.
81	F4 6 Node	12	307	540	1.0280054062e+00	644.98	*Opt*
82	F4 7 Node	17	637	1245	1.0099320805e+00	1769.14	*Opt*
83	F4 8 Node	21	792	1511	9.9609431692e−01	2490.31	*Opt*
84	F4 9 Node	20	1200	3046	9.9011118247e−01	5660.29	*Opt*

Table 3. Large Problem Results: MINOS version 5.3.

No.	Problem	Itns.	QP It.	Funs.	$F(x^*)$	Time	Stat.
81	F4 6 Node	18	64	31	1.029382e+00	175.11	*Opt*
82	F4 7 Node	19	58	29	1.003055e+00	263.37	*Opt*
83	F4 8 Node	21	94	39	9.960943e−01	431.41	*Opt*
84	F4 9 Node	19	95	32	9.901111e−01	530.23	*Opt*
85	F4 10 Node	21	127	37	9.875875e−01	844.71	*Opt*
86	F4 11 Node	27	80	33	9.863111e−01	1264.22	*Opt*
87	F4 12 Node	28	85	34	9.851976e−01	1748.94	*Opt*
88	F4 14 Node	25	78	30	9.849011e−01	2383.98	*Opt*
89	F4 15 Node	26	108	31	9.834741e−01	3154.55	*Opt*
90	F4-NP 6 Node	22	40	30	1.001819e+00	181.78	*Opt*
91	F4-NP 15 Node	33	65	38	9.695897e−01	3858.89	*Opt*
92	VTOL Descent	20	159	25	−1.683044e−00	10351.63	*Opt*

Table 4. Large Problem Results: NZSOL

problems with few degrees of freedom.

Results for MINOS and NZSOL are given in Tables 3 and 4. Results for LSSQP-O and LSSQP-E are given in Tables 5 and 6.

The results of the tests on these larger problems show that LSSQP-O is very competitive with MINOS and NZSOL. NZSOL required the fewest function evaluations and major and minor iterations than either MINOS or LSSQP-O on all the test problems. LSSQP-O required many fewer function evaluations and exhibited faster solution times than MINOS. In addition, LSSQP-O was very competitive with NZSOL with respect to solution time.

One of the major differences between LSSQP-O and NZSOL is in the number of function evaluations and major and minor iterations required to solve the problems to the specified accuracy. For example, LSSQP required three to thirty times as many functions as NZSOL. One reason for this could be the differences in the form of the quasi-Newton update. Another reason could be the form of the null-space basis Z.

It should be noted that extensive tests using LSSQP on the large test problems indicate that it does not have the same level of robustness offered by NZSOL. Modification of one or more of the run-time parameters may lead to significantly slower solution times. One method for increasing the robustness of LSSQP would be to modify the linesearch routines to account for the lack of precision in the gradients of these problems. The tests reported in this section used a 'function plus gradient' linesearch even though the gradients for these large problems are obtained by differencing. Preliminary tests have shown that a 'function only' linesearch may produce a more robust (and efficient) version of LSSQP. More tests are required to determine the exact cause of the large differences in performance between

No.	*Problem*	*Itns.*	*QP It.*	*Funs.*	$F(x^*)$	*Time*	*Stat.*
81	F4 6 Node	40	373	82	1.0280052864e+00	104.27	*Opt*
82	F4 7 Node	42	455	97	1.0030551461e+00	151.83	*Opt*
83	F4 8 Node	36	382	70	9.9471482986e−01	126.03	*Opt*
84	F4 9 Node	40	394	91	9.9011108551e−01	160.97	*Opt*
85	F4 10 Node	76	1139	221	9.8758697042e−01	506.04	*Opt*
86	F4 11 Node	85	1249	254	9.8531899149e−01	645.75	*Opt*
87	F4 12 Node	150	2770	532	9.8519559525e−01	1536.22	*Opt*
88	F4 14 Node	182	2831	794	9.8490191159e−01	2563.11	*Opt*
89	F4 15 Node	246	5067	996	9.8347245192e−01	3901.23	*Opt*
90	F4-NP 6 Node	63	255	212	1.0018209554e+00	232.15	*Opt*
91	F4-NP 15 Node	281	4906	1196	9.8165983233e−01	4793.97	*Opt*
92	VTOL Descent	105	1857	320	−1.6825846298e−00	2363.40	*Opt*

Table 5. Large Problem Results: (LSSQP-O) Full completion.

No.	*Problem*	*Itns.*	*QP It.*	*Funs.*	$F(x^*)$	*Time*	*Stat.*
81	F4 6 Node	64	279	144	1.0280054023e+00	187.85	*Opt*
82	F4 7 Node	44	204	94	1.0030551798e+00	143.22	*Opt*
83	F4 8 Node	143	452	143	9.9471483247e−01	544.59	*Opt*
84	F4 9 Node	85	393	235	9.9011120191e−01	455.78	*Opt*
85	F4 10 Node	143	888	494	9.8758760121e−01	1063.13	*Opt*
86	F4 11 Node	102	625	325	9.8531969486e−01	786.71	*Opt*
87	F4 12 Node	263	1418	804	9.8519638914e−01	2166.77	*Opt*

Table 6. Large Problem Results: (LSSQP-E) Early termination.

NZSOL and LSSQP.

For the largest problem in the test set, the *VTOL Descent problem,* the value of sparse-matrix operations becomes very clear. Even though NZSOL requires a twelfth of the (very expensive) function evalutions, the total CPU time is more than four times that of LSSQP.

The results from MINOS show that the method requires many function evaluations, which results in a substantial increase in the solution times compared to NZSOL and LSSQP-O. As the problem size increases, the solution times grow rapidly. For this reason MINOS was tested on only the four smallest F4 Minimum time-to-climb problems (problems 81–84).

The performance of MINOS on these problems is somewhat anomalous. We see that the hypothesis that MINOS takes fewer major iterations still holds, but the number of minor iterations per major iteration relative to those required by NZSOL and LSSQP is significantly more than for the small test problems. The computation of the constraints and gradients for these problems are very expensive (see the Appendix). Since MINOS must evaluate the constraints and gradients many times to solve each subproblem it cannot match the SQP methods, which only evaluate functions and gradients *after* each subproblem (requiring fewer total function evaluations).

As with the small test set, the early-termination strategy did not perform as well as

the full-optimization strategy. LSSQP-E was tested on the first 6 problems in the large test set and required 60% more CPU than LSSQP-O. It also required more major iterations and function evaluations but fewer minor iterations than LSSQP-O. LSSQP-E required more solution time on all but the 7-node problem over that required by LSSQP-O. It is important to note that the QP iterations on these optimal control problems are relatively cheap compared to function and gradient evalutions. Since the early-termination strategy is designed to reduce the number of QP iterations required, it is not expected to impact the solution of problems such as these. Still, it is encouraging that the early-termination algorithm (even with its acknowledged deficiencies) proved to be robust on the set of large test problems. Moreover, it did achieve a reduction in the number of QP iterations required to find a minimizer. More testing is needed on large-scale problems for which the QP iterations are similar in cost to function evaluations.

5.3.3. A final note on computational results

There are several criteria that should be used to measure the efficiency of an algorithm. Two important measures are speed (measured in CPU time required for solution) and the storage required by all data structures used by the algorithm. LSSQP is similar to MINOS in this last respect, its data structures being almost identical. Because of their sparse-matrix technology, both of these methods have an advantage over NPSOL (or NZSOL) on large problems whose Jacobian is sparse.

Sparse-matrix technology does not give an advantage to either MINOS or LSSQP over NPSOL for problems in the small test set. However, because the function and gradient evaluations are relatively cheap for these problems, the CPU time is highly correlated with the number of basis factorizations required to find a minimizer (at least for the larger of the small test problems).

For the trajectory optimization problems, if sparse-matrix methods are employed the computational cost of solving the problem is dominated by the cost of function and gradient evaluations. For these problems the salient measure of efficiency is time. For MINOS and the two variants of LSSQP, the solution time is highly correlated with the number of function evalutions required. As the problems grow in size, the cost of the dense TQ factorization required within NZSOL increasingly impacts the solution time. As a result, LSSQP exhibits a growing advantage in time over NZSOL. This is clearly shown by the results. The worst relative performance for NZSOL is for the largest problem.

5.4. Conclusions

We have proposed a new algorithm for the solution of large-scale nonlinear programming problems. Our approach differs substantially from previous methods because the QP subproblems are solved using sparse techniques and we approximate only the reduced Hessian of the Lagrangian. The theoretical convergence properties of the new method are the same as for dense implementations that approximate only the reduced Hessian of the Lagrangian. Based on the preliminary numerical test results for the algorithm, there is every reason to expect that the algorithm will prove useful in practice for many large-scale problems in which the nonlinear function and constraints are computationally expensive to evaluate.

5.4.1. Future work

Many modifications could be made to the prototype SQP algorithm. Some possibilities follow:

1. *Use of a "limited size" quasi-Newton approximation to the reduced Hessian.* In this strategy, an approximation to a pseudo reduced Hessian of a fixed size (e.g. of dimension 200) would be updated at each iteration. This would include the reduced Hessian for the current superbasic variables as well as for other *pseudo superbasic variables.* The latter would be a subset of the remaining nonbasic variables and could be identified as those variables that would be expected to be superbasic or had "recently" been superbasic (but were not currently). This would allow curvature information associated with these variables to be maintained throughout the solution process and may lead to faster convergence. An observed feature of LSSQP and NZSOL on large problems is that the number of QP iterations does not always decrease to 1 in a neighborhood of the solution but is usually some small number. Variables on a bound with small reduced costs may enter and leave the superbasic set. The proposed modification would prevent the curvature for these variable from being lost.

2. *Use of second derivatives in the solution of subproblems.* In general, we would expect to obtain faster rates of convergence (i.e. quadratic versus two-step superlinear) at the expense of a more complicated algorithm. With exact second derivatives it would be necessary to have a more complex linesearch as well as a method for maintaining a positive-definite reduced Hessian and routines for obtaining directions of negative curvature (see [Pri89] for example).

3. *Use of single-phase subproblems.* Such a method would incorporate a merit function within the subproblem itself, and should perform well on large-scale problems that require a large number of minor iterations in order to obtain a feasible point for the subproblem. Such a modification would also allow for infeasible subproblems.

4. *Use of the present algorithm at a lower level.* Part of the large-scale SQP algorithm in this report is an algorithm to solve a QP based on the provision of a reduced Hessian. Such an algorithm could be used to solve the subproblems in MINOS. It would have three levels of iteration. At the lowest level, function and gradients would not be required. At the intermediate level, subproblems would be solved with the use of an augmented Lagrangian objective function (as is done now with MINOS). The top level corresponds to a major iteration and makes use of a merit function and a linesearch. We expect that such an algorithm would improve upon the performance of MINOS on problems for which the functions were expensive. It may be expected to do better than the algorithm described here on problems for which the function evaluations, although expensive, did not overwhelm the total computational effort.

5.4.2. Acknowledgments

I am indebted to my coadvisor Professor Michael Saunders for allowing me to use MINOS, many parts of which have been used in the development of the code described in this

report. In addition, his careful checking of the written report has helped to improve both its style and content.

I would like to thank my coadvisor Professor Walter Murray who suggested the problem addressed in this dissertation. Professor Murray has been a driving force in motivating this work.

Special mention must be made of Professor Philip Gill, who has provided invaluable help in the research presented here. Professor Gill has always made his presence felt when it was most needed. Many of the improvements in the report as well as to the implementation of LSSQP were encouraged by Professor Gill, who generously opened his home to me during a visit to La Jolla and to UCSD during the winter of 1991.

No.	*Problem*	*Comment*	*n*	*Lcon.*	*Ncon.*	$F(x^*)$
1	Hexagon	Sparse version	9	4	14	−1.349963e+00
2	HS 108	Sparse version	9	0	13	−8.660254e−01
3	KS 372	Sparse version	9	0	12	1.339009e+04
4	MHW 4	Margaret Wright	5	0	3	2.787187e+01
5	MHW 9	Margaret Wright	5	0	3	−4.247468e+01
6	HS 14		2	1	1	1.393465e+00
7	HS 26	$\delta_{opt} = 10^{-5}$	3	0	1	0.000000e+00
8	HS 43	Rosen-Suzuki	4	0	3	−4.400000e+01
9	HS 65		3	0	1	9.535289e−01
10	KS 231		2	2	0	0.000000e+00
11	OPF 30 BUS		67	0	60	9.720420e−01
12	QP problem		7	7	0	−1.847785e+06
13	LC7		7	7	0	9.295973e+05
14	Norway		7	6	0	−2.402344e+01
15	Singular		2	0	2	0.000000e+00
16	Alan Manne	Economic growth	30	10	10	−2.670099e+00
17	Steinke2	$\delta_{opt} = 10^{-5}$	6	0	4	4.000131e−04
18	Square root 1		9	0	9	2.500000e+03
19	Square root 2		9	0	9	2.999795e+00
20	Square root 3		9	0	9	2.000000e+00
21	Square root 4		4	0	4	2.500000e+01
22	Boggs-Tolle 1		2	0	1	−1.000000e+00
23	Boggs-Tolle 2		3	0	1	3.256820e−02
24	Boggs-Tolle 3		5	3	0	4.093023e+00
25	Boggs-Tolle 4		3	1	1	−4.551055e−03
26	Boggs-Tolle 5	HS 63	3	1	1	9.577426e+02
27	Boggs-Tolle 6	HS 77	5	0	2	2.415051e−01
28	Boggs-Tolle 7		5	0	3	3.065000e+02
29	Boggs-Tolle 8		5	0	2	1.000000e+00
30	Boggs-Tolle 9	HS 39	4	0	2	−1.000000e+00
31	Boggs-Tolle 10		2	0	2	−1.000000e+00
32	Boggs-Tolle 11	HS 79	5	0	3	9.171343e−02
33	Boggs-Tolle 12		5	0	3	6.188119e+00
34	Powell triangles		7	0	5	2.331371e+01
35	Powell badly scaled		2	0	1	3.586574e−03
36	Powell wriggle		2	0	2	−1.911618e−16
37	Powell wriggle	$x_0 = (0.097, 0.063)$	2	0	2	−1.911618e−16
38	Powell-Maratos		2	0	1	−1.000000e+00
39	HS 72	$\delta_{opt} = 10^{-4}$	4	0	2	7.266794e+02
40	HS 73	Cattle feed	4	2	1	2.989438e+01

Table 7. Small problem statistics (1–40).

No.	*Problem*	*Comment*	*n*	*Lcon.*	*Ncon.*	$F(x^*)$
41	HS 107		9	0	6	5.055011e+03
42	Mukai-Polak		6	0	2	5.000000e+00
43	Penalty1	a	50	1	0	4.313635e−02
44	Penalty1	c	50	1	0	4.313635e−02
45	HS 32		3	1	1	1.000000e+00
46	HS 46		5	0	2	0.000000e+00
47	HS 51		5	3	0	0.000000e+00
48	HS 52		5	3	0	5.326648e+00
49	HS 53		5	3	0	4.093023e+00
50	HS 13		2	0	1	1.000000e+00
51	HS 64		3	0	1	6.299842e+03
52	HS 70		4	0	1	7.498464e−03
53	HS 71		4	0	2	1.701402e+01
54	HS 74		4	2	3	5.126498e+03
55	HS 75		4	2	3	5.174413e+03
56	HS 78		5	0	3	−2.919700e+00
57	HS 80		5	0	3	5.394985e−02
58	HS 81		5	0	3	5.394985e−02
59	HS 84		5	0	3	−5.329025e+06
60	HS 85	$\delta_{opt} = 10^{-8}$	5	0	38	−1.905134e+00
61	HS 86	Colville No. 1	5	10	0	−3.234868e+01
62	HS 93	Transformer design	6	0	2	1.350760e+02
63	HS 95		6	0	4	1.561953e−02
64	HS 96		6	0	4	1.561953e−02
65	HS 97		6	0	4	3.135809e+00
66	HS 98		6	0	4	3.135809e+00
67	HS 99		7	0	2	−8.310799e+08
68	HS 100		7	0	4	6.806301e+02
69	HS 104	Reactor design	8	0	5	3.951163e+00
70	HS 109		9	1	8	5.362069e+03
71	HS 111		10	0	3	−4.776109e+01
72	HS 112	Chemical equilibrium	10	3	0	−4.776109e+01
73	HS 113	Wong No. 2	10	3	5	2.430621e+01
74	HS 114	Alkylation process	10	5	6	−1.768807e+03
75	HS 117	Colville No. 2, Shell dual	15	0	5	3.234867e+03
76	HS 118	LC problem	15	17	0	6.648204e+02
77	HS 119	Colville No. 7	16	8	0	2.448997e+02
78	HS 83	Dembo No. 2	5	0	6	1.012243e+04
79	HS 106	Dembo No. 5	8	3	3	7.049331e+04
80	Weapon assignment		100	12	0	−1.735019e+03

Table 8. Small problem statistics (41–80).

No.	*Problem*	*Itns.*	*LC It.*	*Funs.*	$F(x^*)$	*viol.*	*Time*	*Stat.*
1	Hexagon	9	59	126	−1.349963e+00	1.03e−13	6.87	*Opt*
2	HS 108	9	50	104	−8.660254e−01	2.94e−14	4.90	*Opt*
3	KS 372	12	40	84	1.339009e+04	1.73e−10	5.13	*Opt*
4	MHW 4	6	13	31	2.787187e+01	7.93e−12	1.14	*Opt*
5	MHW 9	19	37	130	−4.247468e+01	2.47e−13	3.95	*Opt*
6	HS 14	7	3	17	1.393465e+00	0.00e+00	.55	*Opt*
7	HS 26	16	76	194	1.808355e−12	3.96e−06	4.13	*Opt*
8	HS 43	15	56	117	−4.400000e+01	2.22e−16	3.67	*Opt*
9	HS 65	7	14	39	9.535289e−01	2.67e−15	.97	*Opt*
10	KS 231	1	5	13	1.449526e−25	0.00e+00	.27	*Opt*
11	OPF 30 BUS	8	127	197	9.720420e−01	5.33e−15	185.17	*Opt*
12	QP problem	1	10	18	−1.847785e+06	0.00e+00	.57	*Opt*
13	LC7	1	9	12	9.295973e+05	0.00e+00	.47	*Opt*
14	Norway	1	6	8	−4.086420e+00	0.00e+00	.39	*Opt*[1]
15	Singular	18	1	20	0.000000e+00	1.16e−10	1.13	*Opt*
16	Alan Manne	4	20	36	−2.670099e+00	2.31e−10	4.10	*Opt*
17	Steinke2	—	—	—	—	—	—	*Itr*
18	Square root 1	5	2	9	2.500000e+03	9.71e−16	.75	*Opt*
19	Square root 2	27	0	29	3.000000e+00	1.42e−14	3.54	*Opt*
20	Square root 3	5	4	14	2.000000e+00	5.43e−10	.85	*Opt*
21	Square root 4	21	0	23	2.500000e+03	2.78e−17	1.67	*Opt*
22	Boggs-Tolle 1	17	21	67	−1.000000e+00	4.12e−12	1.72	*Opt*
23	Boggs-Tolle 2	8	20	66	3.256820e−02	4.00e−15	1.19	*Opt*
24	Boggs-Tolle 3	1	3	10	4.093023e+00	0.00e+00	.24	*Opt*
25	Boggs-Tolle 4	7	6	18	−7.317428e+01	1.78e−15	.71	*Opt*
26	Boggs-Tolle 5	—	—	—	—	—	—	*Itr*
27	Boggs-Tolle 6	15	49	121	2.415051e−01	4.44e−16	3.19	*Opt*
28	Boggs-Tolle 7	4	3	12	3.603798e+02	2.00e−12	.43	*Opt*
29	Boggs-Tolle 8	12	2	16	1.000000e+00	2.38e−07	.87	*Opt*
30	Boggs-Tolle 9	11	21	54	−1.000000e+00	1.39e−13	1.65	*Opt*
31	Boggs-Tolle 10	8	1	12	−1.000000e+00	2.78e−17	.59	*Opt*
32	Boggs-Tolle 11	9	19	52	9.171343e−02	2.03e−11	1.67	*Opt*
33	Boggs-Tolle 12	19	75	191	6.188119e+00	1.42e−14	4.92	*Opt*
34	Powell triangles	11	34	82	2.331371e+01	2.22e−16	3.37	*Opt*
35	Powell bad scale	6	4	19	1.146177e−12	1.17e−12	.54	*Opt*
36	Powell wriggle S1	5	7	25	1.061979e+00	0.00e+00	.67	*Opt*
37	Powell wriggle S2	—	—	—	—	—	—	*Itr*
38	Powell-Maratos	9	10	36	−1.000000e+00	0.00e+00	.96	*Opt*
39	HS 72	4	1	7	7.266819e+02	1.16e−16	.33	*Opt*
40	HS 73	5	7	18	2.989438e+01	4.44e−16	.71	*Opt*

[1] Converged to a different minimizer.

Table 9. Small problems: MINOS (1–40).

No.	Problem	Itns.	LC It.	Funs.	$F(x^*)$	viol.	Time	Stat.
41	HS 107	6	12	28	5.055012e+03	4.51e−17	1.53	*Opt*
42	Mukai-Polak	14	113	238	5.000000e+00	0.00e+00	6.44	*Opt*
43	Penalty1 a	1	161	384	4.313635e−02	0.00e+00	43.92	*Opt*
44	Penalty1 c	1	161	384	4.313635e−02	0.00e+00	43.64	*Opt*
45	HS 32	7	7	19	1.000000e+00	0.00e+00	.69	*Opt*
46	HS 46	11	52	120	9.862650e−15	7.57e−07	3.11	*Opt*
47	HS 51	1	3	10	9.629650e−34	0.00e+00	.21	*Opt*
48	HS 52	1	3	6	6.000000e+00	0.00e+00	.25	*Opt*[1]
49	HS 53	1	3	10	4.093023e+00	0.00e+00	.16	*Opt*
50	HS 13	28	6	42	1.000069e+00	0.00e+00	1.88	*Opt*
51	HS 64	13	30	106	6.299842e+03	2.54e−08	2.21	*Opt*
52	HS 70	7	34	92	7.498464e−03	0.00e+00	5.40	*Opt*
53	HS 71	10	21	51	1.701402e+01	8.88e−16	1.65	*Opt*
54	HS 74	16	25	55	5.126498e+03	1.42e−14	2.65	*Opt*
55	HS 75	13	13	33	5.174413e+03	4.83e−13	1.80	*Opt*
56	HS 78	7	10	29	−2.919700e+00	2.08e−14	1.07	*Opt*
57	HS 80	9	18	49	5.394985e−02	2.35e−12	1.66	*Opt*
58	HS 81	9	18	51	5.394985e−02	3.19e−11	1.73	*Opt*
59	HS 84	7	27	78	−5.191258e+06	0.00e+00	2.12	*Opt*[1]
60	HS 85	6	7	32	−1.905155e+00	6.91e−11	4.85	*Opt*
61	HS 86	1	11	19	−3.234868e+01	0.00e+00	.74	*Opt*
62	HS 93	9	35	76	1.350760e+02	2.07e−14	2.44	*Opt*
63	HS 95	3	1	5	1.561953e−02	0.00e+00	.32	*Opt*
64	HS 96	3	1	5	1.561953e−02	0.00e+00	.24	*Opt*
65	HS 97	4	14	20	4.071246e+00	0.00e+00	.92	*Opt*[1]
66	HS 98	4	5	12	4.071246e+00	0.00e+00	.56	*Opt*
67	HS 99	11	60	145	−8.310799e+08	1.02e−10	8.78	*Opt*
68	HS 100	10	56	120	6.839810e+02	1.13e−11	4.27	*Opt*
69	HS 104	—	—	—	—	—	—	*Itr*
70	HS 109	14	64	109	5.362069e+03	1.39e−13	7.18	*Opt*
71	HS 111	19	149	363	−4.776109e+01	9.00e−14	14.31	*Opt*
72	HS 112	1	38	110	−4.776109e+01	0.00e+00	2.26	*Opt*
73	HS 113	18	103	212	2.430621e+01	8.88e−16	10.64	*Opt*
74	HS 114	18	42	120	−1.768807e+03	5.16e−09	6.04	*Opt*
75	HS 117	8	71	140	3.234868e+01	0.00e+00	7.11	*Opt*
76	HS 118	1	17	22	6.648204e+02	0.00e+00	1.20	*Opt*
77	HS 119	1	22	28	2.448997e+02	0.00e+00	2.46	*Opt*
78	HS 83	4	4	9	1.012243e+04	6.89e−13	.67	*Opt*
79	HS 106	5	42	107	2.100000e+03	0.00e+00	4.58	*Opt*[1]
80	Weapon	1	203	410	−1.735019e+03	0.00e+00	38.33	*Opt*

[1] Converged to a different minimizer.

Table 10. Small problems: MINOS (41–80).

No.	*Problem*	*Itns.*	*QP It.*	*Funs.*	$F(x^*)$	*Time*	*Stat.*
1	Hexagon	12	45	16	−.1349963e+01	3.69	*Opt*
2	HS 108	15	32	21	−.8660254e+00	4.41	*Opt*
3	KS 372	14	48	23	.1339009e+04	10.36	*Opt*
4	MHW 4	10	14	18	.2787187e+02	1.31	*Opt*
5	MHW 9	30	42	56	−.3618808e+02	3.71	*Opt*[1]
6	HS 14	6	1	8	.1393465e+00	.49	*Opt*
7	HS 26	47	48	64	.1969433e−20	3.39	*Opt*
8	HS 43	8	9	11	−.4400000e+02	.81	*Opt*
9	HS 65	8	16	10	.9535289e+00	.70	*Opt*
10	KS 231	20	25	24	.1339909e−20	1.41	*Opt*
11	OPF 30 BUS	18	53	19	.9927005e+00	468.12	*Opt*[1]
12	QP problem	8	23	9	−.1847785e+07	1.10	*Opt*
13	LC7	7	13	9	.9295973e+06	.76	*Opt*
14	Norway	4	34	5	−.2402344e+02	1.23	*Opt*
15	Singular	15	4	16	.0000000e+00	1.03	*Opt*
16	Alan Manne	17	40	18	−.2670099e+01	21.13	*Opt*
17	Steinke2	—	—	—	—	—	*Fail*
18	Square root 1	—	—	—	—	—	*Fail*
19	Square root 2	23	0	36	.2999795e+01	5.01	*Opt*
20	Square root 3	6	7	9	.2000000e+01	.95	*Opt*
21	Square root 4	—	—	—	—	—	*Fail*
22	Boggs-Tolle 1	11	11	19	−.1000000e+01	.81	*Opt*
23	Boggs-Tolle 2	9	9	14	.3256820e−01	.71	*Opt*
24	Boggs-Tolle 3	2	2	5	.4093023e+01	.19	*Opt*
25	Boggs-Tolle 4	12	13	18	−.4551055e−03	.92	*Opt*
26	Boggs-Tolle 5	6	8	9	.9577426e+03	.58	*Opt*
27	Boggs-Tolle 6	15	16	21	.2415051e+00	1.52	*Opt*
28	Boggs-Tolle 7	31	32	56	.3065000e+03	3.36	*Opt*
29	Boggs-Tolle 8	17	17	19	.1000000e+01	1.25	*Opt*
30	Boggs-Tolle 9	13	14	16	−.1000000e+01	.95	*Opt*
31	Boggs-Tolle 10	8	0	11	−.1000000e+01	.48	*Opt*
32	Boggs-Tolle 11	9	10	12	.9171343e−01	1.05	*Opt*
33	Boggs-Tolle 12	27	28	57	.6188119e+01	3.04	*Opt*
34	Powell triangles	23	36	37	.2331371e+02	3.27	*Opt*
35	Powell bad scale	12	13	15	.1305195e−23	.85	*Opt*
36	Powell wriggle S1	34	60	69	−.1911618e−15	2.77	*Opt*
37	Powell wriggle S2	8	11	11	−.2530612e−10	.81	*Opt*
38	Powell-Maratos	6	6	7	−.1000000e+01	.44	*Opt*
39	HS 72	7	8	8	.7266794e+03	.69	*Opt*
40	HS 73	4	4	5	.2989438e+02	.38	*Opt*

[1] Converged to a different minimizer.

Table 11. Small problems: NPSOL (1–40).

No.	*Problem*	*Itns.*	*QP It.*	*Funs.*	$F(x^*)$	*Time*	*Stat.*
41	HS 107	11	27	18	.5055012e+04	2.77	*Opt*
42	Mukai-Polak	10	13	16	.5000000e+01	1.08	*Opt*
43	Penalty1 a	16	77	18	.4313635e−01	20.01	*Opt*
44	Penalty1 c	29	152	85	.4313635e−01	24.35	*Opt*
45	HS 32	2	3	3	.1000000e+01	.25	*Opt*
46	HS 46	55	56	58	.1936782e−22	5.26	*Opt*
47	HS 51	2	2	5	.3851860e−32	.18	*Opt*
48	HS 52	2	2	5	.5326648e+01	.19	*Opt*
49	HS 53	2	2	5	.4093023e+01	.19	*Opt*
50	HS 13	22	13	23	.1002181e+01	1.29	*Opt*
51	HS 64	29	47	39	.6299842e+04	2.34	*Opt*
52	HS 70	36	39	39	.7498464e−02	3.33	*Opt*
53	HS 71	5	9	6	.1701402e+02	.53	*Opt*
54	HS 74	10	14	15	.5126498e+04	1.17	*Opt*
55	HS 75	6	7	10	.5174413e+04	.72	*Opt*
56	HS 78	10	11	14	−.2919700e+01	1.15	*Opt*
57	HS 80	8	8	10	.5394985e−01	.92	*Opt*
58	HS 81	14	15	20	.5394985e−01	1.57	*Opt*
59	HS 84	—	—	—	—	—	*Fail*
60	HS 85	17	33	18	−.1905155e+01	4.00	*Opt*
61	HS 86	6	11	8	−.3234868e+02	.62	*Opt*
62	HS 93	12	14	15	.1350760e+03	1.36	*Opt*
63	HS 95	1	1	2	.1561953e−01	.15	*Opt*
64	HS 96	1	1	2	.1561953e−01	.17	*Opt*
65	HS 97	3	3	6	.3135809e+01	.40	*Opt*
66	HS 98	3	8	6	.3135809e+01	.43	*Opt*
67	HS 99	23	74	44	−.8290102e+09	3.99	*Opt*
68	HS 100	14	18	29	.6806301e+03	2.07	*Opt*
69	HS 104	18	23	20	.3951163e+01	3.36	*Opt*
70	HS 109	11	25	13	.5362069e+04	3.23	*Opt*
71	HS 111	41	44	64	−.4773239e+02	8.08	*Opt*
72	HS 112	19	54	39	−.4776109e+02	2.78	*Opt*
73	HS 113	14	38	19	.2430621e+02	3.12	*Opt*
74	HS 114	18	36	19	−.1768807e+04	3.81	*Opt*
75	HS 117	17	96	21	.3234868e+02	6.75	*Opt*
76	HS 118	4	20	6	.6648204e+03	1.35	*Opt*
77	HS 119	12	41	16	.2448997e+03	4.25	*Opt*
78	HS 83	4	4	6	.1012243e+05	.54	*Opt*
79	HS 106	17	30	21	.7049248e+04	2.90	*Opt*
80	Weapon	96	244	98	−.1735019e+04	120.78	*Opt*

Table 12. Small problems: NPSOL (41–80).

No.	*Problem*	*Itns.*	*QP It.*	*Funs.*	$F(x^*)$	*viol.*	*Time*	*Stat.*
1	Hexagon	23	56	36	−1.349963e+00	5.13e−13	11.53	*Opt*
2	HS 108	7	7	10	−8.660254e−01	4.85e−14	1.88	*Opt*
3	KS 372	35	205	87	1.339009e+04	5.93e−12	26.26	*Opt*
4	MHW 4	15	19	28	2.787187e+01	2.84e−11	2.74	*Opt*
5	MHW 9	14	19	27	−4.247468e+01	5.42e−12	2.42	*Opt*
6	HS 14	8	2	14	1.393465e+00	3.21e−12	.81	*Opt*
7	HS 26	37	36	47	7.028190e−12	5.55e−17	4.59	*Opt*
8	HS 43	13	17	17	−4.400000e+01	7.11e−11	2.16	*Opt*
9	HS 65	17	25	22	9.535289e−01	2.06e−13	2.38	*Opt*
10	KS 231	15	16	44	6.321700e−21	0.00e+00	1.51	*Opt*
11	OPF 30 Bus	30	112	40	9.720420e−01	5.61e−15	178.95	*Opt*
12	QP problem	11	26	16	−1.847785e+06	0.00e+00	2.51	*Opt*
13	LC7	10	24	14	9.295973e+05	0.00e+00	2.04	*Opt*
14	Norway	6	20	9	−2.402344e+01	0.00e+00	1.66	*Opt*
15	Singular	20	1	23	0.000000e+00	7.28e−12	1.75	*Opt*
16	Alan Manne	14	35	22	−2.670099e+00	8.98e−14	9.64	*Opt*
17	Steinke2	2	14	5	4.142865e−04	1.02e−07	.77	*Opt*
18	Square root 1	17	2	47	2.499997e+03	2.50e−09	4.19	*Opt*
19	Square root 2	18	0	25	2.999939e+00	6.43e−10	3.49	*Opt*
20	Square root 3	13	12	18	2.000000e+00	2.86e−12	3.37	*Opt*
21	Square root 4	—	—	—	—	—	—	*Itr*
22	Boggs-Tolle 1	11	10	18	−1.000000e+00	0.00e+00	1.26	*Opt*
23	Boggs-Tolle 2	12	12	20	3.256820e−02	5.80e−14	1.53	*Opt*
24	Boggs-Tolle 3	5	7	10	7.957949e−01	0.00e+00	.70	*Opt*[1]
25	Boggs-Tolle 4	8	7	16	−7.317428e+01	1.83e−13	1.05	*Opt*[1]
26	Boggs-Tolle 5	10	10	15	9.617152e+02	1.77e−11	1.41	*Opt*[1]
27	Boggs-Tolle 6	25	27	34	2.415051e−01	2.23e−14	3.75	*Opt*
28	Boggs-Tolle 7	—	—	—	—	—	—	*Itr*
29	Boggs-Tolle 8	21	3	24	1.000000e+00	9.10e−13	2.41	*Opt*
30	Boggs-Tolle 9	45	46	87	−1.000001e+00	2.77e−09	6.84	*Opt*
31	Boggs-Tolle 10	8	1	10	−1.000000e+00	2.97e−09	.78	*Opt*
32	Boggs-Tolle 11	14	16	19	9.171343e−02	2.78e−12	2.33	*Opt*
33	Boggs-Tolle 12	56	62	162	6.188119e+00	7.11e−13	9.69	*Opt*
34	Powell triangles	15	34	20	2.331371e+01	4.56e−11	4.02	*Opt*
35	Powell bad scale	15	12	54	0.000000e+00	0.00e+00	1.85	*Opt*
36	Powell wriggle S1	139	178	283	−3.171038e−13	6.78e−08	20.05	*Opt*
37	Powell wriggle S2	47	14	99	−3.677403e−07	6.44e−10	5.12	*Opt*
38	Powell-Maratos	7	5	10	−1.000000e+00	5.55e−17	.83	*Opt*
39	HS 72	4	1	7	7.266819e+02	1.18e−16	.48	*Opt*
40	HS 73	5	7	9	2.989438e+01	4.44e−16	.79	*Opt*

[1] Converged to a different minimizer.

Table 13. Small problems: (LSSQP-O) Full completion (1-40).

No.	Problem	Itns.	QP It.	Funs.	$F(x^*)$	viol.	Time	Stat.
41	HS 107	—	—	—	—	—	—	*Itr*
42	Mukai-Polak	39	51	99	5.000000e+00	2.32e−13	6.65	*Opt*
43	Penalty1 a	5	100	14	4.961360e−02	0.00e+00	14.57	*Opt*[1]
44	Penalty1 c	5	100	14	4.961360e−02	0.00e+00	14.20	*Opt*[1]
45	HS 32	4	6	6	1.000000e+00	0.00e+00	.61	*Opt*
46	HS 46	24	26	30	6.666278e−13	2.16e−15	3.57	*Opt*
47	HS 51	5	8	10	6.499273e−02	0.00e+00	.75	*Opt*
48	HS 52	7	13	14	1.372951e+00	0.00e+00	1.08	*Opt*[1]
49	HS 53	5	7	10	7.957949e−01	0.00e+00	.75	*Opt*[1]
50	HS 13	24	22	30	9.995875e−01	8.78e−12	2.52	*Opt*[1]
51	HS 64	15	21	21	6.299842e+03	9.43e−10	2.04	*Opt*
52	HS 70	36	39	41	7.498464e−03	0.00e+00	6.01	*Opt*
53	HS 71	9	13	13	1.701402e+01	1.78e−14	1.43	*Opt*
54	HS 74	12	13	15	5.126498e+03	2.27e−13	2.15	*Opt*
55	HS 75	8	7	10	5.174413e+03	2.30e−10	1.30	*Opt*
56	HS 78	10	11	16	−2.919700e+00	3.08e−13	1.67	*Opt*
57	HS 80	11	11	16	5.394985e−02	0.00e+00	1.78	*Opt*
58	HS 81	12	12	17	5.394985e−02	2.22e−16	2.05	*Opt*
59	HS 84	4	6	6	−5.329025e+06	0.00e+00	.59	*Opt*
60	HS 85	17	40	46	−1.905155e+00	3.55e−15	17.33	*Opt*
61	HS 86	15	38	25	−3.234868e+01	0.00e+00	3.41	*Opt*
62	HS 93	26	58	35	1.350760e+02	3.40e−11	5.77	*Opt*
63	HS 95	3	1	5	1.561953e−02	0.00e+00	.36	*Opt*
64	HS 96	3	1	5	1.561953e−02	0.00e+00	.38	*Opt*
65	HS 97	8	23	11	3.135809e+00	0.00e+00	1.66	*Opt*
66	HS 98	8	20	11	3.135809e+00	0.00e+00	1.55	*Opt*
67	HS 99	28	49	55	−8.310799e+08	2.11e−07	7.28	*Opt*
68	HS 100	20	28	31	6.839810e+02	1.12e−09	4.28	*Opt*
69	HS 104	22	49	34	3.951163e+00	6.88e−15	6.53	*Opt*
70	HS 109	18	56	24	5.362069e+03	4.87e−13	7.13	*Opt*
71	HS 111	54	65	68	−4.776109e+01	2.78e−17	11.99	*Opt*
72	HS 112	19	62	54	−4.776109e+01	0.00e+00	5.13	*Opt*
73	HS 113	60	148	107	2.430621e+01	1.23e−08	23.85	*Opt*
74	HS 114	30	63	91	−8.825283e+02	5.68e−14	11.51	*Opt*[1]
75	HS 117	—	—	—	—	—	—	*Itr*
76	HS 118	4	30	7	6.648204e+02	0.00e+00	3.42	*Opt*
77	HS 119	18	95	40	2.448997e+02	0.00e+00	11.44	*Opt*
78	HS 83	4	7	7	1.012243e+04	6.89e−13	1.08	*Opt*
79	HS 106	5	10	7	2.100000e+03	0.00e+00	1.27	*Opt*[1]
80	Weapon	2124	393	165	−1.735019e+03	0.00e+00	176.05	*Opt*

[1] Converged to a different minimizer.

Table 14. Small problems: (LSSQP-O) Full completion (41–80).

No.	*Problem*	*Itns.*	*QP It.*	*Funs.*	$F(x^*)$	*viol.*	*Time*	*Stat.*
1	Hexagon	33	63	43	−1.349963e+00	6.73e−12	14.87	*Opt*
2	HS 108	15	24	22	−8.660254e−01	2.78e−17	5.18	*Opt*
3	KS 372	23	67	38	1.339009e+04	9.98e−09	11.24	*Opt*
4	MHW 4	42	18	84	2.787187e+01	1.24e−12	6.00	*Opt*
5	MHW 9	20	21	37	−4.247468e+01	1.31e−13	3.32	*Opt*
6	HS 14	7	2	11	1.393465e+00	8.60e−12	.75	*Opt*
7	HS 26	31	29	39	7.717254e−10	2.22e−16	3.49	*Opt*
8	HS 43	18	21	28	−4.400000e+01	1.23e−10	3.02	*Opt*
9	HS 65	21	22	33	9.535288e−01	0.00e+00	2.55	*Opt*
10	KS 231	15	14	44	6.321700e−21	0.00e+00	1.18	*Opt*
11	OPF 30 BUS	81	144	129	9.720420e−01	1.33e−08	384.83	*Opt*
12	QP problem	5	9	9	−7.750787e+05	0.00e+00	.91	*Opt*[1]
13	LC7	12	19	15	7.790963e+05	0.00e+00	1.69	*Opt*[1]
14	Norway	6	13	8	−4.086420e+00	0.00e+00	1.10	*Opt*[1]
15	Singular	20	1	23	0.000000e+00	7.28e−12	1.75	*Opt*
16	Alan Manne	24	46	29	−2.670099e+00	5.62e−14	16.72	*Opt*
17	Steinke2	7	14	14	4.000131e−04	8.01e−16	1.56	*Opt*
18	Square root 1	15	2	28	2.500000e+03	4.90e−13	3.38	*Opt*
19	Square root 2	17	0	29	2.999878e+00	2.27e−10	3.22	*Opt*
20	Square root 3	3	0	5	2.000000e+00	0.00e+00	.48	*Opt*
21	Square root 4	—	—	—	—	—	—	*Itr*
22	Boggs-Tolle 1	15	8	27	−1.000000e+00	2.87e+00	1.60	*Opt*
23	Boggs-Tolle 2	14	12	23	3.256820e−02	1.31e−14	1.63	*Opt*
24	Boggs-Tolle 3	6	8	10	8.117684e−01	0.00e+00	.63	*Opt*[1]
25	Boggs-Tolle 4	9	6	13	−7.317428e+01	1.78e−15	1.11	*Opt*[1]
26	Boggs-Tolle 5	11	8	16	9.617152e+02	0.00e+00	1.44	*Opt*[1]
27	Boggs-Tolle 6	29	28	50	2.415051e−01	1.94e−12	4.09	*Opt*
28	Boggs-Tolle 7	—	—	—	—	—	—	*Itr*
29	Boggs-Tolle 8	21	3	24	1.000000e+00	9.10e−13	2.30	*Opt*
30	Boggs-Tolle 9	—	—	—	—	—	—	*Itr*
31	Boggs-Tolle 10	8	1	12	−1.000000e+00	2.97e−09	.79	*Opt*
32	Boggs-Tolle 11	15	16	22	9.171343e−02	1.11e−16	2.34	*Opt*
33	Boggs-Tolle 12	70	56	275	6.188119e+00	2.64e−11	11.08	*Opt*
34	Powell triangles	—	—	—	—	—	—	*Itr*
35	Powell bad scale	—	—	—	—	—	—	*Itr*
36	Powell wriggle S1	—	—	—	—	—	—	*Itr*
37	Powell wriggle S2	—	—	—	—	—	—	*Itr*
38	Powell-Maratos	7	4	10	−1.000000e+00	1.57e−14	.74	*Opt*
39	HS 72	4	1	7	7.266819e+02	1.18e−16	.47	*Opt*
40	HS 73	5	7	8	2.989438e+01	0.00e+00	.75	*Opt*

[1] Converged to a different minimizer.

Table 15. Small problems: (LSSQP-E) Early termination (1–40).

No.	*Problem*	*Itns.*	*QP It.*	*Funs.*	$F(x^*)$	*viol.*	*Time*	*Stat.*
41	HS 107	—	—	—	—	—	—	*Itr*
42	Mukai-Polak	17	20	49	5.000000e+00	7.79e−07	2.63	*Opt*
43	Penalty1 a	4	50	15	4.961363e−02	0.00e+00	4.93	*Opt*[1]
44	Penalty1 c	4	50	15	4.961363e−02	0.00e+00	4.64	*Opt*[1]
45	HS 32	4	6	6	1.000000e+00	0.00e+00	.66	*Opt*
46	HS 46	40	41	49	3.370541e−11	4.72e−14	5.75	*Opt*
47	HS 51	5	8	10	6.499273e−02	0.00e+00	.72	*Opt*[1]
48	HS 52	14	21	50	7.222497e−01	0.00e+00	1.96	*Opt*[1]
49	HS 53	5	7	10	8.117684e−01	0.00e+00	.67	*Opt*[1]
50	HS 13	24	22	30	9.995875e−01	8.78e−12	2.30	*Opt*
51	HS 64	18	23	33	6.299842e+03	4.00e−08	2.32	*Opt*
52	HS 70	37	35	41	7.498464e−03	0.00e+00	5.82	*Opt*
53	HS 71	15	17	19	1.701402e+01	1.65e−12	2.08	*Opt*
54	HS 74	12	12	15	5.126498e+03	2.27e−13	1.98	*Opt*
55	HS 75	8	7	11	5.174413e+03	2.30e−10	1.28	*Opt*
56	HS 78	—	—	—	—	—	—	*Itr*
57	HS 80	12	10	18	5.394985e−02	0.00e+00	1.84	*Opt*
58	HS 81	11	9	15	5.394985e−02	1.11e−16	1.63	*Opt*
59	HS 84	6	5	8	−5.329025e+06	1.46e−11	.75	*Opt*
60	HS 85	—	—	—	—	—	—	*Itr*
61	HS 86	5	9	9	−3.234868e+01	0.00e+00	.90	*Opt*
62	HS 93	18	26	25	1.350760e+02	8.67e−14	3.10	*Opt*
63	HS 95	5	5	7	1.561953e−02	0.00e+00	.70	*Opt*
64	HS 96	5	5	7	1.561953e−02	0.00e+00	.73	*Opt*
65	HS 97	4	15	7	3.135809e+00	0.00e+00	1.08	*Opt*
66	HS 98	—	—	—	—	—	—	*Itr*
67	HS 99	—	—	—	—	—	—	*Itr*
68	HS 100	23	25	33	6.839810e+02	3.20e−11	4.17	*Opt*
69	HS 104	26	48	56	3.951163e+00	1.21e−17	6.92	*Opt*
70	HS 109	13	44	18	5.362287e+03	7.35e−13	5.07	*Opt*
71	HS 111	68	55	88	−4.776109e+01	6.39e−08	12.58	*Opt*
72	HS 112	24	60	74	−4.776109e+01	0.00e+00	5.06	*Opt*
73	HS 113	53	64	65	2.430621e+01	8.40e−09	14.63	*Opt*
74	HS 114	—	—	—	—	—	—	*Cbi*
75	HS 117	31	193	304	1.325514e+03	0.00e+00	27.38	*Opt*[1]
76	HS 118	10	37	16	6.648204e+02	0.00e+00	4.36	*Opt*
77	HS 119	13	44	25	2.448997e+02	0.00e+00	5.48	*Opt*
78	HS 83	9	9	12	1.012243e+04	3.36e−08	1.95	*Opt*
79	HS 106	7	7	9	2.100000e+03	0.00e+00	1.32	*Opt*[1]
80	Weapon	129	410	175	−1.735019e+03	0.00e+00	194.84	*Opt*

[1] Converged to a different minimizer.

Table 16. Small problems: (LSSQP-E) Early termination (41–80).

Appendix A

Nonlinear Programming for Trajectory Optimization

A.1. Trajectory optimization

Despite the empirical sucess of optimization implementations such as MINOS and NPSOL, we can identify problems for which improved performance is desirable. A class of problems that we feel will benefit from large-scale SQP methods is in the area of *trajectory optimization*. In general these mathematical programming problems are characterized by matrices that are large and sparse and have functions that are expensive to evaluate. An example of a trajectory optimization problem is the *Supersonic Interceptor Minimum-Time Climb* (SIMTC) problem [Bry69]. The problem statement is:

> *Find the path taking a supersonic interceptor from sea level and Mach 0.34 to an altitude of 20km and Mach 1.0 in minimum time.*

Sample graphs of the optimal altitude and thrust profiles (plotted against elapsed time of flight) for a Phantom F4 are given in Figure 1. Note the non-intuitive shape of the optimal trajectory.

A.2. Problem statement

Trajectory optimization leads to problems in optimal control. The goal is to minimize a specified performance index F. For the SIMTC problem in Section A.1 the performance index is the *time* required to reach a specific altitude and speed. Other possibilities for F are the amount of fuel burned or the time to reach a specific destination. Trajectory optimization problems are described in terms of a sequence of N time *stages* with time points E_i (called *events*) delimiting the stages. For the general formulation we write F as

$$F(x(E), u(E), \omega, E). \tag{A.2.1}$$

The performance index F is a function of

- A vector of states x, governed by first-order differential equations (see below),
- Control functions $u(t)$ (e.g. pitch angle),

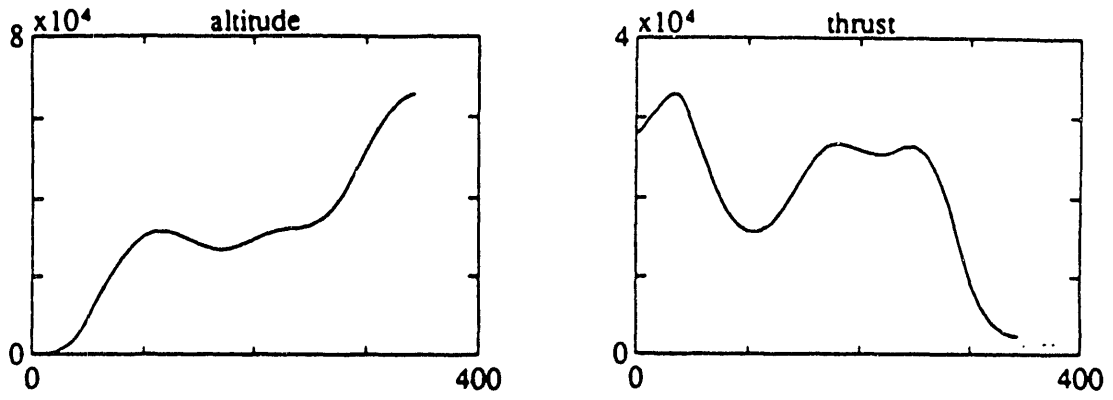

Figure 1. Altitude and thrust profiles for the Phantom-F4 SIMTC problem

- Vehicle design parameters ω (e.g. rocket nozzle diameter),
- Time points E_i, $i = 1, \ldots, N+1$, delimiting the stages.

The i-th stage is a dynamical system restricted by differential constraints (called state equations) of the form

$$x_i' \equiv \frac{dx}{dt} = f_i(x, u, \omega, t) \quad t \in [E_i, E_{i+1}], \tag{A.2.2}$$

for $i = 1, \ldots, N$. These constraints correspond to differential equations of motion. The variables must also satisfy nonlinear initial and terminal conditions a_i at each stage:

$$l_i^A \leq a_i(x(E_i), (E_i), \omega) \leq u_i^A. \tag{A.2.3}$$

In addition, path constraints h_i may be imposed on the system at each stage E_i:

$$l_i^H \leq h_i(x, u, \omega, t) \leq u_i^H. \tag{A.2.4}$$

The functions f_i, a_i and h_k are assumed to be twice continuously differentiable *within* each stage. However, the functions are allowed to be discontinuous *between* events. That is, at event boundaries, discontinuities of the form

$$x(E_{i+1}) = x(E_i) + \sigma_i \tag{A.2.5}$$

are allowed. These allow the modelling of characteristics such as the jettison of a payload or a modification of velocity. The σ_i's may be fixed or included in the design parameter set ω.

Hargraves and Paris [HarP87] presented a direct trajectory optimization method of the form (A.2.1)-(A.2.5) that represents state and control variables by piecewise polynomials. This method has been developed into OTIS, a system for trajectory optimization [HarP88]. Specifically, Hargraves and Paris transformed the optimal control problem into a mathematical programming problem by using an implicit integration scheme known

as *collocation* (see [Enr91]) to satisfy equations (A.2.2). This method of transforming the optimal control problem into a mathematical programming problem is called *direct transcription.*

A complete description of the transcription method used to approximate (A.2.2) can be found in [Enr91] or [HarP87]. The method is summarized below:

1. Each stage $[E_i, E_{i+1}]$ is partitioned into a set of M smaller segments.
2. A cubic spline is fitted for each segment. The data for the fit is taken from the values of $\frac{dx}{dt}$ at the mesh points of each segment.
3. A numerical integration scheme is used to approximate $x(t)$ at the midpoint of each segment.
4. Variables δ_i (called *defects*) are defined as the difference between the approximation and the true value at the midpoint.

If the defects can be driven to zero, the cubic spline will provide an accurate approximation to (A.2.2). As a result of this transcription process, the optimal control constraints (A.2.2) can be replaced in the formulation by equality constraints of the form $\delta_j = 0$ for $i = 1, \ldots, M$, for each of the N events for the problem.

A.2.1. Problem formulation

The mathematical programming problem now includes terms for the defects of the interpolation method in place of equations (A.2.2). In addition, the boundary conditions (A.2.3) are enforced and the nonlinear path constraints (A.2.4) are enforced at the grid points. The transcribed trajectory optimization problem may be written as the following mathematical program:

$$
\begin{array}{lccccl}
\text{minimize} & & F(x,u,E,\omega) & & & \\
\text{s.t.} & & d_i & = & 0, & i = 1,\ldots,N, \\
& l_i^A \leq & a_i(x(E_1),(E_1),\omega) & \leq & u_i^A, & i = 1,\ldots,N, \\
& l_i^H \leq & h_i(x,u,\omega,t) & \leq & u_i^H, & i = 1,\ldots,N, \\
& & x(E_{i+1}) - x(E_i) - \sigma_i & = & 0, & i = 1,\ldots,N, \\
& l^B \leq & (x,u,E,\omega) & \leq & u^B, &
\end{array}
$$

where d_i is a vector of center defects for stage i ($d_i = [\delta_{i1}, \ldots, \delta_{iM}]$).

References

[Big72] M.C. Biggs (1972). Constrained minimization using recursive equality constrained quadratic programming, in *Numerical Methods for Nonlinear Optimization* (Academic Press, London/New York).

[BogT84] P.T. Boggs and J.W. Tolle (1984). A family of descent functions for constrained optimization, *SIAM J. Numerical Analysis* **21**, 4, 1146–1161.

[Bro77] K.W. Brodlie (1977). An assessment of two approaches to variable metric methods, *Mathematical Programming* **12**, 344–355.

[Bro67] C.G. Broyden (1967). Quasi-Newton methods and their application to function minimization, *Mathematics of Computation* **21**, 368–381.

[Bry69] A.E. Bryson, M.N. Desai and W.C. Hoffman (1969). Energy-state approximations in performance optimization of supersonic aircraft, *J. of Aircraft* **6**, 6.

[CCT91] A.R. Conn, N.I.M. Gould and Ph. Toint (1991). Convergence of quasi-Newton matrices generated by the symmetric rank one update, *Mathematical Programming* (to appear).

[ColC84] T.F. Coleman and A.R. Conn (1984). On the local convergence of a quasi-Newton method for the nonlinear programming problem, *SIAM J. Numerical Analysis* **21**, 4, 755–769.

[ColF88] T.F. Coleman and P.A. Fenyes (1988). Partitioned quasi-Newton methods for nonlinear equality constrained optimization, Report TR 88-931, Department of Computer Science, Cornell University, Ithaca, New York.

[ColS84] T.F. Coleman and D.C. Sorensen (1984). A note on the computation of an orthonormal basis for the null space of a matrix, *Mathematical Programming* **29**, 234–242.

[Cot74] R.W. Cottle (1974). Manifestations of the Schur-complement, *J. Linear Algebra and its Applications* **8**, 189–211.

[CotD79] R.W. Cottle and A. Djang (1979). Algorithmic equivalence in quadratic programming, I: A least distance programming problem. *J. of Optimization Theory and Applications*, **28**, 275–301.

[DemT85] R.S. Dembo and U. Tulowitzki (1985). Sequential truncated quadratic programming methods, in **Numerical Optimization 1984.** Proceedings of the SIAM conference on Numerical Optimization, Boulder, Colorado. P.T. Boggs, R.H. Byrd, and R.B. Schnabel, eds. (SIAM, Philadelphia), 83–101.

[DenM77] J.E. Dennis Jr. and J.J. Moré (1977). Quasi-Newton methods, motivation and theory, *SIAM Review* **19**, 1, 47–89.

[DenS83] J.E. Dennis Jr. and R.B. Schnabel (1983). *Numerical Methods for Unconstrained Optimization* (Prentice-Hall, Englewood Cliffs, New Jersey).

[Dix72a] L.C.W. Dixon (1972). Quasi-Newton algorithms generate identical points, *Mathematical Programming* **2**, 383–387.

[Dix72b] L.C.W. Dixon (1972). Quasi-Newton algorithms generate identical points II, the proof of four new theorems, *Mathematical Programming* **3**, 345–358.

[EldS92] S.K. Eldersveld and M.A. Saunders (1990). A block-LU update for large-scale linear programming, *SIAM J. Matrix Anal. Appl.* **13**, 191–201.

[Enr91] P.J. Enright (1991). Optimal finite-thrust spacecraft trajectories using direct transcription and nonlinear programming, Ph.D. Thesis, University of Illinois.

[FiaM68] A.V. Fiacco, G.P. McCormick (1968). *Nonlinear Programming: Sequential Unconstrained Minimization Techniques* (John Wiley and Sons, New York/London/Sydney/Toronto).

[Fle70] R. Fletcher (1970). A class of methods for nonlinear programming with termination and convergence properties, in *Integer and Nonlinear Programming*, J. Abadie, ed. (North Holland, Amsterdam).

[Fle87] R. Fletcher (1987). *Practical methods of Optimization* (John Wiley and Sons, Chichester/New York/Brisbane/Toronto/Singapore).

[Fra88] C. Fraley (1988). Software performance on nonlinear least-squares problems. Report SOL 88-17, Department of Operations Research, Stanford University, Stanford, California.

[GHMSW86] P.E. Gill, S.J. Hammarling, W. Murray, M.A. Saunders and M.H. Wright (1986). User's guide for LSSOL (version 1.0): A Fortran package for constrained linear least squares and convex quadratic programming, Report SOL 86-1, Department of Operations Research, Stanford University, Stanford, California.

[GGMS74] P.E. Gill, G.H. Golub, W. Murray and M.A. Saunders (1974). Methods for modifying matrix factorizations, *Mathematics of Computation* **28**, 4, 505–535.

[GilM72] P.E. Gill and W. Murray (1972). Quasi-Newton methods for unconstrained optimization, *J. Inst. Maths. Applics.* **9**, 91–108.

[GilM73] P.E. Gill and W. Murray (1973). Quasi-Newton methods for linearly constrained optimization, *Nat. Phys. Lab. Rept.* NAC **32**.

[GilM74] P.E. Gill and W. Murray (1974). Newton-type methods for linearly constrained optimization, in **Numerical Methods for Constrained Optimization.** P.E. Gill and W. Murray, eds. (Academic Press, New York), 29–66.

[GilM77] P.E. Gill and W. Murray (1977). Linearly constrained problems including linear and quadratic programming, in **The State of the Art in Numerical Analysis,** D.A.H. Jacobs, ed. (Academic Press, London/New York/San Francisco), 313–363.

[GilM79] P.E. Gill and W. Murray (1979). The computation of Lagrange-multiplier estimates for constrained minimization, *Mathematical Programming* **17**, 32–60.

[GMP72] P.E. Gill, W. Murray and R.A. Pitfield (1972). The implementation of two revised quasi-Newton algorithms for unconstrained minimization, *Nat. Phys. Lab. Rept.* NAC **11**.

[GMSSW85] P.E. Gill, W. Murray, M.A. Saunders, G.W. Stewart, and M.H. Wright (1985). Properties of a representation of a basis for the null space, *Mathematical Programming* **33**, 172–186.

[GMSW79] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright (1979). Two steplength algorithms for numerical optimization, Report SOL 79-25, Department of Operations Research, Stanford University, Stanford, California.

[GMSW83] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright (1983). User's guide for SOL/QPSOL:A Fortran package for quadratic programming, Report SOL 83-7, Department of Operations Research, Stanford University, Stanford, California.

[GMSW84] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright (1984). User's guide for NPSOL (version 2.1): A Fortran package for nonlinear programming, Report SOL 84-7, Department of Operations Research, Stanford University, Stanford, California.

[GMSW86a] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright (1986). User's guide for NPSOL (version 4.0): A Fortran package for nonlinear programming. Report SOL 86-2, Department of Operations Research, Stanford University, Stanford, California.

[GMSW86b] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright (1986). Some theoretical properties of an augmented Lagrangian merit function, Report SOL 86-6R, Department of Operations Research, Stanford University, Stanford, California.

[GMSW87] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright (1987). Maintaining *LU* factors of a general sparse matrix, *Linear Algebra and its Applications*, 88/89 239–270.

[GMSW88] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright (1989). Recent developments in constrained optimization, *J. Computational and Applied Mathematics* **22**, 257–270.

[GMSW89] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright (1989). Constrained nonlinear programming, in **Handbooks in OR & MS, Vol. 1,** G.L. Nemhauser *et al.* eds. (Elsevier Science, North Holland), 171–210.

[GMW81] P.E. Gill, W. Murray and M.H. Wright (1981). *Practical Optimization*, (Academic Press, London/New York).

[Gur87] C.B. Gerwitz (1986). Sequential quadratic programming methods based on approximating a projected Hessian matrix, Report 219, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York.

[GurO89] C.B. Gurwitz and M.L. Overton (1989). Sequential Quadratic Programming Methods based on approxomating a projected Hessian matrix, *SIAM J. Scientific and Statistical Computing* **10**, 4, 631–653.

[Han76] S. Han (1976). Superlinearly convergent variable metric algorithms for general nonlinear programming problems, *Mathematical Programming* **11**, 263–282.

[HarP87] C.R. Hargraves and S.W. Paris (1987). Direct trajectory optimization using nonlinear programming and collocation, *J. of Guidance, Control, and Dynamics* **10**, 4, 338–348.

[HarP88] C.R. Hargraves and S.W. Paris (1988). Optimal trajectories by implicit simulation: Volume I - Formulation manual, Report AFWAL-TR-88-3057, Flight dynamics laboratory, Air Force Wright Aeronautical Laboratories.

[HocS81] W. Hock and K. Schittkowski (1981). *Test Examples for Nonlinear Programming Codes*, in **Lecture Notes in Economics and Mathematical Systems 187** (Springer-Verlag,Berlin/Heidelberg/New York).

[KBS90] H. Khalfan, R.H. Byrd, and R.B. Schnabel (1990). A theoretical and experimental study of the summetric rank one update, Report CU-CS-489-90, Department of Computer Science, University of Colorado at Boulder, Boulder, Colorado.

[Mar78] N. Maratos (1978). Exact penalty function algorithms for finite dimensional and control optimization problems, Ph.D. Thesis, Imperial College, London.

[MorS84] J.J. Moré and D.C. Sorensen (1984). Newton's method, in *Studies in Numerical Analysis*, G.H. Golub, ed. (Mathematical Association of America) 29–82.

[Mur69] W. Murray (1969). An algorithm for constrained minimization, in *Optimization*, R. Fletcher, ed. (Academic Press, London/New York).

[MurW78] W. Murray and M.H. Wright (1978). Projected Langrangian methods based on the trajectories of penalty and barrier functions, Report SOL 78-23, Department of Operations Research, Stanford University, Stanford, California.

[MurS78] B.A. Murtagh and M.A. Saunders (1978). Large-scale linearly constrained optimization, *Mathematical Programming* **14**, 41–72.

[MurS82] B.A. Murtagh and M.A. Saunders (1982). A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints, *Mathematical Programming* **16**, 84–117.

[MurS87] B.A. Murtagh and M.A. Saunders (1987). MINOS 5.1 User's Guide, Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, California.

[NicT89] R.H. Nickel and J.H. Tolle (1989). A sparse sequential quadratic programming algorithm *J. of Optimization Theory and Applications* **60**,3, 453–472.

[NocO85] J. Nocedal and M.L. Overton (1985). Projected Hessian updating algorithms for nonlinearly constrained optimization, *SIAM J. Numerical Analysis* **22**, 5, 822–850.

[Orc68] W. Orchard-Hays (1968). *Advanced Linear-Programming Computing Techniques* (McGraw-Hill, New York).

[Ore74] S.S. Oren (1974). On the selection of parameters in self-scaling variable metric algorithms, *Mathematical Programming* **7** 351–367.

[OreS76] S.S. Oren and E. Spedicato (1976). Optimal conditioning of self-scaling variable metric algorithms, *Mathematical Programming* **10**, 70–90.

[Pow78a] M.J.D. Powell (1978). The convergence of variable metric methods for nonlinearly constrained optimization calculations, in **Nonlinear programming 3**, O.L Mangasarian, R. Meyer and S.M. Robinson, eds. (Academic Press, New York), 27–63.

[Pow78b] M.J.D. Powell (1978). A fast algorithm for nonlinearly constrained optimization calculations, in **Numerical Analysis, Dundee, 1977, Lecture Notes in Mathematics 630**, G.A. Watson, ed. (Springer-Verlag, Berlin/Heidelberg/New York). 144–157.

[Pow83] M.J.D. Powell (1983). Variable metric methods for constrained optimization, in **Mathematical Programming, The State of the Art, Bonn 1982**, A. Bachem, M. Grötschel and B. Korte, eds. (Springer-Verlag, Berlin/Heidelberg/New York), 288–311.

[Pri89] F.J. Prieto (1989). Sequential quadratic programming algorithms for optimization, Report SOL 89-7, Department of Operations Research, Stanford University, Stanford, California.

[Roc73] R.T. Rockafellar (1973). The multiplier method of Hestenes and Powell applied to convex programming, *J. Optimization Theory and Applications*, **12**, 555–562.

[Sch82] K. Schittkowski (1982). On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search function, Report SOL 82-4, Department of Operations Research, Stanford University, Stanford California.

[Sch87] K. Schittkowski (1987). *More Test Examples for Nonlinear Programming Codes*, in **Lecture Notes in Economics and Mathematical Systems 282** (Springer-Verlag, Berlin/Heidelberg/New York/London/Paris/Tokyo).

[Sha80] D.F. Shanno (1980). On variable metric methods for sparse Hessians. *Mathematics of Computation* **34**, 499–514.

[Tha83] M.T. Thapa (1983). Optimization of unconstrained functions with sparse Hessian matrices:quasi-Newton methods. *Mathematical Programming* **25**, 158–182.

[Wil63] R.B. Wilson (1963). A simplicial algorithm for concave programming, Ph.D. Thesis, Harvard University.

[Wri76] M.H. Wright (1976). Numerical methods for nonlinearly constrained optimization, Ph.D. Thesis, Department of Computer Science, Stanford University, Stanford, California.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1992	

4. TITLE AND SUBTITLE
Large-Scale Sequential Quadratic Programming Algorithms

5. FUNDING NUMBERS
N00014-90-J-1242
DE-FG03-92ER25117

6. AUTHOR(S)
Samuel K. Eldersveld

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Operations Research - SOL
Stanford University
Stanford, CA 94305-4022

8. PERFORMING ORGANIZATION REPORT NUMBER
1111MA

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research - Department of the Navy
800 W. Quincy Street
Arlington, VA 22217

Office of Energy Research
U.S. Department of Energy
Washington, DC 20585

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
SOL 92-4

12a. DISTRIBUTION / AVAILABILITY STATEMENT
UNLIMITED

12b. DISTRIBUTION CODE
UL

13. ABSTRACT (Maximum 200 words)

(see other side)

14. SUBJECT TERMS
Quasi-Newton; Large-Scale Optimization; Nonlinear Constrained Optimization; Progressive Programming.

15. NUMBER OF PAGES
82 pages

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED			SAR

Abstract

The problem addressed is the general nonlinear programming problem: finding a local minimizer for a nonlinear function subject to a mixture of nonlinear equality and inequality constraints. The methods studied are in the class of sequential quadratic programming (SQP) algorithms, which have previously proved successful for problems of moderate size. Our goal is to devise an SQP algorithm that is applicable to large-scale optimization problems, using sparse data structures and storing less curvature information but maintaining the property of superlinear convergence. The main features are:

1. *The use of a quasi-Newton approximation to the reduced Hessian of the Lagrangian function.* Only an estimate of the reduced Hessian matrix is required by our algorithm. The impact of not having available the full Hessian approximation is studied and alternative estimates are constructed.

2. *The use of a transformation matrix Q.* This allows the QP gradient to be computed easily when only the reduced Hessian approximation is maintained.

3. *The use of a reduced-gradient form of the basis for the null space of the working set.* This choice of basis is more practical than an orthogonal null-space basis for large-scale problems. The continuity condition for this choice is proven.

4. *The use of incomplete solutions of quadratic programming subproblems.* Certain iterates generated by an active-set method for the QP subproblem are used in place of the QP minimizer to define the search direction for the nonlinear problem.

An implementation of the new algorithm has been obtained by modifying the code MINOS. Results and comparisons with MINOS and NPSOL are given for the new algorithm on a set of 92 test problems.

END

DATE FILMED

1/20/93