UNIVERSITY OF CALIFORNIA, SAN DIEGO

SQP Methods for Large-Scale Optimization

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

in

Mathematics

by

Alexander Barclay

Committee in charge:

Professor Philip Gill, Chair
Professor Randolph Bank
Professor Richard Belew
Professor James Bunch
Professor J. Ben Rosen

1999

The dissertation of Alexander Barclay is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

1999

To my wife Claire, and my children Missy, Tyler and Trevor, whose love and patience make dreams come true.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my wife and children, to whom this work is dedicated. I owe a special debt to my wife Claire for enduring a struggle greater than either of us imagined at the outset. Her ability to find kind words at the most difficult times meant more that mere words can convey.

I would like to extend my deep thanks and gratitude to my advisor Philip Gill. His support, patience, insight and enthusiasm have fueled this process from beginning to end. Philip provided me with both financial assistance and friendly support. In addition, he showed great patience and understanding in dealing with an older graduate student with dependents. His contributions to one of the most significant chapters of my life will never be forgotten.

For their part in this chapter, I would like to thank the other members of my committee as well. I thank Ben Rosen for his friendship and research motivation. I came to admire Ben while working with him as a Research Assistant. I would also like to personally thank Jim Bunch for his friendship and occasional advice, both were welcome. For their time and effort on my behalf, I thank Randy Bank and Richard Belew.

I owe a special thanks to Michael Saunders of the Stanford Optimization Laboratory. I am grateful for his priceless insights on LSQR, MINOS and SNOPT, and for his helpful suggestions on the third chapter. I will forever be in awe of Michael's and Philip's numerical software skills.

I thank the National Science Foundation for providing the majority of my financial support during my time at UCSD.

I would like to thank Lois Stewart and Natalie Powell for helping to keep me afloat in times of trouble.

Finally, I would like to give thanks to my friends and extended family. There are far to many to mention, but their sincere acts of kindness and support, both large and small, are neither forgotten nor taken for granted.

VITA

September 5, 1962	Born, Folsom, California
1989	B. A., Computational Mathematics, University of California, Santa Cruz
1995-1999	Research Assistant, Department of Mathematics, University of California, San Diego
1996	M. A., Applied Mathematics, University of California, San Diego
1997–1998	Teaching assistant, Department of Mathematics, University of California, San Diego
1999	Ph. D., Mathematics, University of California, San Diego

PUBLICATIONS

"SQP Methods in Optimal Control". (With P. Gill and J. B. Rosen.) In R. Bulirsch, L. Bittner, W.H. Schmidt and K. Heier, editors, *Variational Calculus, Optimal Control and Applications*, volume 124 of *International Series of Numerical Mathematics*, pgs. 207-222, Birkhäuser, 1998.

ABSTRACT OF THE DISSERTATION

SQP Methods for Large-Scale Optimization

by

Alexander Barclay
Doctor of Philosophy in Mathematics

University of California San Diego, 1999

Professor Philip Gill, Chair

Sequential quadratic programming methods have proved highly effective for solving constrained optimization problems with smooth nonlinear functions in the objective and constraints. Sequential quadratic programming methods solve a sequence of quadratic programming (QP) subproblems, where each iteration of the QP subproblem requires the solution of a large linear system for the search direction. For some problems, the number of degrees of freedom, n_z, is of moderate size (say, up to 1000), and the large system can be solved using two smaller dense systems, one for the QP search direction and one for the QP multipliers. In this case, the system for the search direction is $Z^T HZp = -Z^T g$, where $Z^T HZ$ is an $n_z \times n_z$ reduced Hessian.

The thesis is concerned with the formulation and analysis of two sequential quadratic programming algorithms that are designed to be efficient when n_z is large ($n_z \gg 1000$). The first method uses the conjugate-gradient method to solve the linear systems. The reduced system is shown to be equivalent to a certain least-squares problem that can be solved using the conjugate-gradient algorithm LSQR. Two preconditioners are proposed to accelerate convergence. The conjugate-gradient algorithm is implemented in a large-scale sequential quadratic programming method based on the optimization code SNOPT. Numerical results

are presented to demonstrate the efficiency of the algorithm for problems with large degrees of freedom.

The second method uses a Schur-complement approach to solve a large sparse system derived directly from the optimality conditions. The search directions are calculated using a fixed large sparse system and a small dense system incorporating the changes to the active set at each QP iteration. A general algorithm for convex quadratic programming is discussed, and a Schur-complement method is formulated for use with the algorithm. In addition, a new Schur-complement method is derived for minimizing an ℓ_2 composite objective function when the initial point is not feasible. Modifications to the standard QP algorithm are presented that allow the use of a composite objective.

Chapter 1

Introduction

Large-scale sequential quadratic programming (SQP) methods have proved highly effective for solving constrained optimization problems of the form

$$\text{NP} \qquad \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad f(x) \qquad \text{subject to} \;\; l \le \begin{pmatrix} x \\ F(x) \\ Gx \end{pmatrix} \le u,$$

where l and u are constant upper and lower bounds, $f(x)$ is a smooth nonlinear function, G is a sparse matrix, $F(x)$ is a vector of smooth nonlinear constraint functions $\{F_i(x)\}$, and n is large ($n \gg 1000$). The upper and lower bounds l and u allow for the general description of various types of bounds and constraints. For instance, free rows and unbounded variables are represented by setting the appropriate $l_i = -\infty$ and $u_i = +\infty$. Equality constraints and fixed variables are represented by setting $l_i = u_i$. If $F(x)$ is not present, and if $f(x)$ is linear, then problem NP defines a linear program.

The success of SQP methods on practical problems of this form has inevitably lead to a substantial increase in expectation. As SQP methods have gained acceptance as a numerical tool, the size and difficulty of the problems attempted has inevitably increased. It is crucial that new SQP methods be developed for problems that cannot yet be solved efficiently by using existing technology.

1.1 Description of a SQP Method

A particular large-scale SQP method is implemented in the optimization code SNOPT [16]. The SQP method used in SNOPT first converts problem NP to an equality constrained problem with simple bounds. The upper and lower bounds on the m components of F and Gx are said to define the *general constraints* of the problem. SNOPT converts the general constraints to equalities by introducing a set of *slack variables* s, where $s = (s_1, s_2, \ldots, s_m)^T$. For example, the linear constraint $5 \leq 2x_1 + 3x_2 \leq +\infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$ together with the bounded slack $5 \leq s_1 \leq +\infty$. The problem NP can therefore be rewritten in the following equivalent form

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & f(x) \\ \text{subject to} & \begin{pmatrix} F(x) \\ Gx \end{pmatrix} - s = 0, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u. \end{array} \tag{1.1}$$

The linear and nonlinear general constraints become equalities of the form $F(x) - s_N = 0$ and $Gx - s_L = 0$, where s_L and s_N are known as the *linear* and *nonlinear* slacks.

The basic structure of a SQP method involves *major* and *minor* iterations. The major iterations generate a sequence of iterates (x_k) that satisfy the linear constraints. The iterates (x_k) converge to a point that satisfies the first-order conditions for optimality. At each iterate, a quadratic programming (QP) subproblem is used to generate a search direction towards the next iterate (x_{k+1}). Solving the QP subproblem is itself an iterative procedure, with the minor iterations of an SQP method being the iterations of the QP method.

1.1.1 Major Iterations

In SNOPT, the constraints of the subproblem are formed from the linear constraints $Gx - s_L = 0$ and the nonlinear constraint linearization

$$F(x_k) + F'(x_k)(x - x_k) - s_N = 0,$$

where $F'(x_k)$ denotes the *Jacobian matrix*, whose rows are the first derivatives of $F(x)$ evaluated at x_k. The QP constraints therefore comprise the m linear constraints

$$\begin{array}{lllcl} F'(x_k)x & -s_N & & = & -F(x_k) + F'(x_k)x_k, \\ Gx & & -s_L & = & 0, \end{array}$$

where x and s are bounded above and below by u and l as before. If the $m \times n$ matrix A and m-vector b are defined as

$$A = \begin{pmatrix} F'(x_k) \\ G \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -F(x_k) + F'(x_k)x_k \\ 0 \end{pmatrix},$$

then the QP subproblem can be written as

$$\underset{x,s}{\text{minimize}} \;\; \psi(x) \quad \text{subject to} \;\; Ax - s = b, \quad l \le \begin{pmatrix} x \\ s \end{pmatrix} \le u, \tag{1.2}$$

where $\psi(x)$ is a quadratic approximation to a modified Lagrangian function (see Section 2.1).

1.1.2 Minor Iterations

The QP method of SNOPT is an inertia-controlling active-set method [13, 21]. At each minor iteration of an active-set method, the constraints $Ax - s = b$ are (conceptually) partitioned into the form

$$Bx_B + Sx_S + Nx_N = b,$$

where the *basis matrix* B is square and nonsingular. The elements of x_B, x_S and x_N are called the *basic*, *superbasic* and *nonbasic* variables respectively; they are a permutation of the elements of x and s. At a QP solution, the basic and superbasic variables will lie somewhere between their bounds, while the nonbasic variables will be equal to one of their upper or lower bounds. At each iteration, x_S is regarded as a set of independent variables that are free to move in any desired direction, namely one that will improve the value of the QP objective (or the sum of infeasibilities). The basic variables are then adjusted in order to ensure that (x, s) continues to satisfy $Ax - s = b$. The number of superbasic variables (n_S say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied (or $n_z \equiv n_S$). In broad terms, n_S is a measure of *how nonlinear* the problem is. In particular, n_S will always be zero for LP problems.

If it appears that no improvement can be made with the current definition of B, S and N, a nonbasic variable is selected to be added to S, and the process is repeated with the value of n_S increased by one. At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of n_S is decreased by one.

Associated with each of the m equality constraints $Ax - s = b$ are the *dual variables* π. The dual variables are also known as the *Lagrange multipliers*. Similarly, each variable in (x, s) has an associated *reduced gradient* d_j. The reduced gradients for the variables x are the quantities $g - A^T\pi$, where g is the gradient of the QP objective, and the reduced gradients for the slacks are the dual variables π. The QP subproblem is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for other variables, including superbasics.

A more detailed description of the QP method used in SNOPT is given in *Chapter 2*.

1.1.3 The Merit Function

After a QP subproblem has been solved, new estimates of the NP solution are computed using a line search on the augmented Lagrangian merit function

$$\mathcal{M}(x, s, \pi) = f(x) - \pi^T(F(x) - s_N) + \tfrac{1}{2}(F(x) - s_N)^T D(F(x) - s_N), \tag{1.3}$$

where D is a diagonal matrix of penalty parameters. If (x_k, s_k, π_k) denotes the current solution estimate and $(\widehat{x}_k, \widehat{s}_k, \widehat{\pi}_k)$ denotes the optimal QP solution, the line search determines a step α_k $(0 < \alpha_k \leq 1)$ such that the new point

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \pi_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ \pi_k \end{pmatrix} + \alpha_k \begin{pmatrix} \widehat{x}_k - x_k \\ \widehat{s}_k - s_k \\ \widehat{\pi}_k - \pi_k \end{pmatrix}$$

gives a *sufficient decrease* in the merit function (1.3). When necessary, the penalties in D are increased by the minimum-norm perturbation that ensures descent for $\mathcal{M}$ [22]. In SNOPT, s_N is adjusted to minimize the merit function as a function of s prior to the solution of the QP subproblem. For more details, see [18, 9].

1.1.4 Treatment of Constraint Infeasibilities

SNOPT makes explicit allowance for infeasible constraints. Infeasible linear constraints are detected first by solving a problem of the form

$$\boxed{\begin{array}{lll} \text{FLP} & \underset{x,v,w}{\text{minimize}} & e^T(v+w) \\ & \text{subject to} & l \leq \begin{pmatrix} x \\ Gx - v + w \end{pmatrix} \leq u, \quad v \geq 0, \quad w \geq 0, \end{array}}$$

where e is a vector of ones. This is equivalent to minimizing the sum of the general linear constraint violations subject to the simple bounds. (In the linear programming literature, the approach is often called *elastic programming*.)

If the linear constraints are infeasible ($v \neq 0$ or $w \neq 0$), SNOPT terminates without computing the nonlinear functions.

If the linear constraints are feasible, all subsequent iterates satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which the functions can be safely evaluated.) SNOPT proceeds to solve NP as given, using search directions obtained from a sequence of quadratic programming subproblems (1.2).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variables π for the linearized constraints become large), SNOPT enters "elastic" mode and solves the problem

$$\begin{array}{|lll|}\hline \text{NP} & \underset{x,v,w}{\text{minimize}} & f(x) + \gamma e^T(v+w) \\ & \text{subject to} & l \le \begin{pmatrix} x \\ F(x) - v + w \\ Gx \end{pmatrix} \le u, \quad v \ge 0, \quad w \ge 0, \\ \hline \end{array}$$

where γ is a nonnegative parameter (the *elastic weight*), and $f(x) + \gamma e^T(v + w)$ is called a *composite objective*. If γ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds. A similar ℓ_1 formulation of NP is fundamental to the Sℓ_1QP algorithm of Fletcher [11]. See also Conn [8].

1.2 Contributions of this Thesis

The general focus of the thesis research is on using iterative methods to solve the QP subproblems within an SQP method. Currently, SNOPT solves the QP subproblems by solving a series of linear systems of the form

$$Z^T HZp_Z = -Z^T g, \tag{1.4}$$

where $Z^T HZ \in I\!R^{n_S \times n_S}$, with $n_S < n$. SNOPT is well suited for problems where the number of degrees of freedom, n_S, is of moderate size (say, up to 1000).

The primary objective of this dissertation is to develop QP algorithms that are efficient when n_S is large ($n_S \gg 1000$), for use within an SQP algorithm such as SNOPT. Two different large-scale QP algorithms are investigated.

1. A conjugate-gradient (CG) method is used to solve the linear system, $Z^T HZp_Z = -Z^T g$. This method includes the definition of two different preconditioners for accelerating convergence.

2. A Schur-complement (SC) algorithm is investigated for solving the

Karush-Kuhn-Tucker (KKT) systems associated with (1.2),

$$\begin{pmatrix} H & W^T \\ W & 0 \end{pmatrix} \begin{pmatrix} u \\ \lambda \end{pmatrix} = \begin{pmatrix} g \\ 0 \end{pmatrix}.$$

See Section 4.1.

The Conjugate-Gradient Method

The CG method solves the reduced-Hessian systems (1.4) by solving an associated least-squares problem. This allows the use of a least-squares CG algorithm. This approach benefits from the numerical advantages of using a least-squares CG algorithm instead of a general symmetric CG algorithm for solving linear systems. Critical to the successful implementation of the CG method is the calculation and maintenance of appropriate preconditioners for the associated least-squares problem. The thesis investigates the use of block diagonal preconditioners, with a dense upper-triangular block, and a larger diagonal extension.

The numerical results demonstrate the potential of the CG method on problems with as many as 5625 degrees of freedom. The performance is particularly good on unconstrained and nonlinearly constrained problems. The need for preconditioning is evident on some problems. However, when the number of CG iterations is small relative to the size of the linear system, the benefits of the preconditioner are overwhelmed by its cost.

The workspace size on the computer used for the test results limited the size of the problems attempted. In the future it should be possible to run problems with tens of thousands of degrees of freedom. It is assumed that this will show an even greater benefit from the CG method, and the preconditioners.

The use of a CG method to solve the QP subproblems within an SQP method has not been investigated elsewhere. However, a similar approach is used by Chen *et al.* [7] for linear programming, where the linear systems of a primal-dual log barrier algorithm are solved as a least-squares problem using a least-squares CG algorithm. One of the preconditioners investigated in the thesis uses a block diagonal quasi-Newton approximation. In related work, Morales and Nocedal [29] discuss using a CG algorithm with a limited-memory quasi-Newton preconditioner for solving a sequence of linear systems where the coefficient matrix and right-hand side are changing. However, their work is directed towards Hessian-free Newton methods and finite-element problems.

The Schur-Complement Method

A SC method is investigated for solving the QP subproblems. The method extends the work of Gill *et al.* [20, 21]. The general SC concept of using a fixed factorization of an initial KKT system, and an updated factorization of a smaller, dense Schur complement, is preserved. However, the thesis extends this to define the algorithm in terms of an extended KKT system, whose working set may consist of constraints whose *residual is not necessarily zero*. This allows a new Schur-complement QP method to be defined that only requires a positive-semidefinite Hessian. In addition, by proper formulation of the algorithm, some of the solves with the initial KKT system are avoided.

Particular attention is given to addressing an infeasible initial point. A new SC algorithm is derived using an ℓ_2 quadratic penalty function for the QP objective. The use of a composite objective function reduces the number of working set changes to the dense Schur complement when finding a feasible point. The choice of the ℓ_2 norm places fewer restrictions on the working set, by not requiring it to be linearly independent at each step. A linearly independent working set is only required when the penalty parameter becomes large.

The thesis is organized as follows. *Chapter 2* presents background on the SQP method of SNOPT, with particular attention on the QP algorithm used in SNOPT. The conjugate-gradient QP method is presented in *Chapter 3*, including some numerical results on a subset large problems from the CUTE test suite [4]. The Schur-complement QP method is presented in *Chapter 4*.

1.3 Notation

Unless otherwise indicated, the following notation is used throughout the dissertation.

Capital letters are used to represent matrices, lowercase letters represent vectors and Greek letters represent scalars.

For a matrix, the corresponding lowercase letter with a subscript represents a column of the matrix. For instance, h_i will be the i-th column of the matrix H. An element of a matrix is indicated by a double indexed subscript of the associated lowercase letter, i.e., h_{ij} will be the ij-th element of H.

Bracketed vectors with a subscript indicate an element of a vector. For example, $(v)_i$ would be the i-th element of the vector v.

The vector e_t is the t-th column of the identity. The lowercase e is reserved for a vector of all ones. In all cases, the dimension will depend on the context.

The norm $\|.\|$ is assumed to be the two-norm.

Chapter 2

Background

The QP solver used in SNOPT (that of SQOPT [15]) is based on an inertia-controlling method that maintains a Cholesky factorization of the reduced Hessian (see Section 2.2). The method follows Gill and Murray [13] and is described in [21]. This chapter briefly summarizes the main features of the method.

SQOPT solves the QP subproblem given in *Chapter 1* as

$$\underset{x,s}{\text{minimize}} \quad \psi(x) \qquad \text{subject to} \qquad Ax - s = b, \quad l \le \begin{pmatrix} x \\ s \end{pmatrix} \le u. \tag{2.1}$$

Since the slack variables are subject to the same upper and lower bounds as the components of Ax, they allow us to think of the bounds on Ax and x as simply bounds on the combined vector (x, s). (In order to indicate their special role in problem (2.1), the original variables x are sometimes known as "column variables", and the slack variables s are known as "row variables").

SQOPT's method has a *feasibility phase* (also known as *phase 1*), in which a feasible point is found by minimizing the sum of infeasibilities, and an *optimality phase* (or *phase 2*) in which the quadratic objective is minimized within the feasible region. The computations of both phases are performed by the same subroutines, with the change of phase being characterized by the objective changing from the sum of infeasibilities to the quadratic objective.

In general, an iterative process is required to solve a quadratic program. Given an iterate (x, s) in both the original variables x and the slack variables s, a new iterate $(\bar{x}, \bar{s})$ is defined by

$$\begin{pmatrix} \bar{x} \\ \bar{s} \end{pmatrix} = \begin{pmatrix} x \\ s \end{pmatrix} + \alpha p, \tag{2.2}$$

where the *step length* α is a non-negative scalar, and p is called the *search direction*. (For simplicity, the reference to the index of the iteration is dropped.) Once an iterate is feasible (i.e., satisfies the constraints), all subsequent iterates remain feasible.

2.1 Definition of the Objective Function

For this section, let x_k and π_k be estimates of the solution and the Lagrange multipliers of problem NP at the k-th major iteration. The SQP algorithm implemented in SNOPT is based on the *modified Lagrangian* associated with problem NP, namely

$$\mathcal{L}(x, x_k, \pi_k) = f(x) - \pi_k^T d_L(x, x_k), \tag{2.3}$$

which is defined in terms of the *constraint linearization* and the *departure from linearity*:

$$\begin{aligned} c_L(x, x_k) &= c(x_k) + J(x_k)(x - x_k), \\ d_L(x, x_k) &= c(x) - c_L(x, x_k); \end{aligned}$$

see Robinson [33] and Van der Hoek [34]. The first and second derivatives of the modified Lagrangian with respect to x are

$$\begin{aligned} \nabla\mathcal{L}(x, x_k, \pi_k) &= g(x) - (J(x) - J(x_k))^T \pi_k, \\ \nabla^2\mathcal{L}(x, x_k, \pi_k) &= \nabla^2 f(x) - \sum_i (\pi_k)_i \nabla^2 c_i(x). \end{aligned}$$

Observe that $\nabla^2\mathcal{L}$ is independent of x_k (and is the same as the Hessian of the conventional Lagrangian). At $x = x_k$, the modified Lagrangian has the same function and gradient values as the objective $f(x)$:

$$\mathcal{L}(x_k, x_k, \pi_k) = f(x_k), \qquad \nabla\mathcal{L}(x_k, x_k, \pi_k) = g(x_k).$$

The objective of the QP subproblem is a quadratic model of $\mathcal{L}$ (2.3), which (ignoring constant terms) is given by

$$\psi(x) = g^T x + \frac{1}{2}x^T Hx,$$

where g is the gradient of $f(x)$ at the current x_k, and H is a positive-definite approximation to Hessian of the modified Lagrangian function.

2.1.1 The Approximate Hessian

On completion of the line search, let the change in x and the gradient of the modified Lagrangian be

$$\delta_k = x_{k+1} - x_k \quad \text{and} \quad y_k = \nabla\mathcal{L}(x_{k+1}, x_k, \pi) - \nabla\mathcal{L}(x_k, x_k, \pi), \tag{2.4}$$

for some vector π. An estimate of the curvature of the modified Lagrangian along δ_k is incorporated using the BFGS quasi-Newton update,

$$H_{k+1} = H_k + \theta_k y_k y_k^T - \phi_k q_k q_k^T, \tag{2.5}$$

where $q_k = H_k\delta_k$, $\theta_k = 1/y_k^T\delta_k$ and $\phi_k = 1/q_k^T\delta_k$. When H_k is positive definite, H_{k+1} is positive definite if and only if the approximate curvature $y_k^T\delta_k$ is positive. Precautions are taken against a negative or small value of $y_k^T\delta_k$ [16].

The updated multipliers π_{k+1} from the line search are chosen for π, and the definition of $\mathcal{L}$ (2.3) yields

$$\begin{aligned} y_k &= \nabla\mathcal{L}(x_{k+1}, x_k, \pi_{k+1}) - \nabla\mathcal{L}(x_k, x_k, \pi_{k+1}) \\ &= g(x_{k+1}) - (J(x_{k+1}) - J(x_k))^T\pi_{k+1} - g(x_k). \end{aligned}$$

In the large-scale case, H_k cannot be treated as an $n \times n$ dense matrix. Section 2.1.3 discusses the limited-memory quasi-Newton scheme implemented in SNOPT.

2.1.2 Linear Variables

If only some of the variables occur nonlinearly in the objective and constraint functions, they are referred to as *linear variables.* If there are linear variables, the Hessian of the Lagrangian has structure that can be exploited during the optimization. Assume that the nonlinear variables are the first $\bar{n}$ components of x. By induction, if H_0 is zero in its last $n - \bar{n}$ rows and columns, the last $n - \bar{n}$ components of the BFGS update vectors y_k and $H_k \delta_k$ are zero for all k, and every H_k has the form

$$H_k = \begin{pmatrix} \bar{H}_k & 0 \\ 0 & 0 \end{pmatrix}, \tag{2.6}$$

where $\bar{H}_k$ is $\bar{n} \times \bar{n}$. A QP subproblem with Hessian of this form is either unbounded, or has at least $n - \bar{n}$ constraints in the final working set. This implies that the reduced Hessian need never have dimension greater than $\bar{n}$.

In order to treat semidefinite Hessians such as (2.6), SQOPT includes an *inertia controlling* working-set strategy, which ensures that the reduced Hessian has at most one zero eigenvalue [16].

2.1.3 Limited-memory Hessians

To treat problems where the number of nonlinear variables $\bar{n}$ is very large, a limited-memory procedure is used to update an initial Hessian approximation H_r a limited number of times. The present implementation is quite simple and has benefits in the SQP context when the constraints are linear.

Initially, suppose $\bar{n} = n$. Let ℓ be preassigned (say $\ell = 20$), and let r and k denote two major iterations such that $r \le k \le r + \ell$. Up to ℓ updates to a positive-definite H_r are accumulated to represent the Hessian as

$$H_k = H_r + \sum_{j=r}^{k-1} \theta_j y_j y_j^T - \phi_j q_j q_j^T, \tag{2.7}$$

where $q_j = H_j \delta_j$, $\theta_j = 1/y_j^T \delta_j$ and $\phi_j = 1/q_j^T \delta_j$. The quantities $(y_j, q_j, \theta_j, \phi_j)$

are stored for each j. During major iteration k, the QP solver accesses H_k by requesting products of the form $H_k v$. These are computed by

$$H_k v = H_r v + \sum_{j=r}^{k-1} \theta_j (y_j^T v) y_j - \phi_j (q_j^T v) q_j. \tag{2.8}$$

The work for forming vector products with H is $3n(k-r) - 2n$ flops, and $n+2$ additional multiplications. On completion of iteration $k = r + \ell$, the diagonals of H_k are computed from (2.7) and saved to form the next positive-definite H_r (with $r = k+1$). Storage is then "reset" by discarding the previous updates. (Similar schemes are suggested by Buckley and LeNir [5, 6] and Gilbert and Lemaréchal [12].)

If $\bar{n} < n$, H_k has the form (2.6) and the same procedure is applied to $\bar{H}_k$. Note that the vectors y_j and q_j have length $\bar{n}$—a benefit when $\bar{n} \ll n$.

2.2 Definition of the Working Set

At each minor iterate (x, s), a *working set* of constraints is defined to be a linearly independent subset of the constraints that are satisfied "exactly" (to within the value of a feasibility tolerance). For problem (2.1), the working set consists of the equality constraints $Ax - s = b$, and the bound constraints for those elements of (x, s) that are on one of their bounds. The working set is the current prediction of the constraints that hold with equality at a solution of the QP. Let m_w denote the number of constraints in the working set (including bounds), and let W denote the associated $m_w \times (n+m)$ *working-set matrix* consisting of the m_w gradients of the working-set constraints.

The search direction is defined so that constraints in the working set remain *unaltered* for any value of the step length. It follows that p must satisfy the identity $Wp = 0$. This characterization allows p to be computed using any $n \times n_Z$ full-rank matrix Z that spans the null space of W. (Thus, $n_Z = n - m_w$ and $WZ = 0$.) The null-space matrix Z is defined from a sparse LU factorization of part of W;

see (2.9)–(2.10) below. The direction p will satisfy $Wp = 0$ if $p = Zp_Z$ for any n_Z-vector p_Z.

The working set contains the constraints $Ax - s = 0$ and a subset of the upper and lower bounds on the variables (x, s). Since the gradient of a bound constraint $x_j \geq l_j$ or $x_j \leq u_j$ is a vector of all zeros except for ± 1 in position j, it follows that the working-set matrix contains the rows of $\begin{pmatrix} A & -I \end{pmatrix}$ and the unit rows associated with the upper and lower bounds in the working set.

The working-set matrix W can be represented in terms of a certain column partition of the matrix $\begin{pmatrix} A & -I \end{pmatrix}$. We partition the constraints $Ax - s = 0$ so that

$$Bx_B + Sx_S + Nx_N = 0,$$

where B is a square non-singular basis and x_B, x_S and x_N are the basic, superbasic and nonbasic variables respectively. The nonbasic variables are equal to their upper or lower bounds at (x, s), and the superbasic variables are independent variables that are chosen to improve the value of the current objective. The number of superbasic variables is n_s. Given values of x_N and x_S, the basic variables x_B are adjusted so that (x, s) satisfies $Bx_B + Sx_S + Nx_N = 0$.

If P is a permutation such that $\begin{pmatrix} A & -I \end{pmatrix} P = \begin{pmatrix} B & S & N \end{pmatrix}$, then the working-set matrix W satisfies

$$WP = \begin{pmatrix} B & S & N \\ 0 & 0 & I_N \end{pmatrix}, \tag{2.9}$$

where I_N is the identity matrix with the same number of columns as N.

The null-space matrix Z is defined from a sparse LU factorization of part of W. In particular, Z is maintained in "reduced-gradient" form, so that

$$Z = P \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}, \tag{2.10}$$

where P is a permutation. This choice of Z implies that n_Z, the number of degrees of freedom at (x, s), is the same as n_s, the number of superbasic variables.

The package LUSOL [19] is used to maintain sparse LU factors of the basis matrix B, which alters as the working set W changes. The matrix Z is used only as an operator, i.e., it is never computed explicitly. Products of the form Zv and Z^Tg are obtained by solving with B or B^T. Specifically, the calculation of $u = Zv$ is performed by first computing $w = Sv$, and then solving $By = w$ using the LU factors of B. The desired vector u is given by

$$u = \begin{pmatrix} y \\ v \\ 0 \end{pmatrix}.$$

If B and S were dense, the work of computing matrix-vector products with Z would be $m^2 + mn_s$ flops. When the sparse LU factors of B are used, one factor of m is reduced. Forming matrix-vector products with Z^T is a similar computation. The number of flops required is the same, but an additional n_s additions are needed to sum terms.

Let g_Z and H_Z denote the *reduced gradient* and *reduced Hessian*:

$$g_Z = Z^Tg \qquad \text{and} \qquad H_Z = Z^THZ, \tag{2.11}$$

where g is the objective gradient at (x, s). Roughly speaking, g_Z and H_Z describe the first and second derivatives of an n_Z-dimensional *unconstrained* problem for the calculation of p_Z.

At each iteration, an upper-triangular factor R is available such that $H_Z = R^TR$. Normally, R is computed from $R^TR = Z^THZ$ at the start of phase 2 and is then updated as the QP working set changes. For efficiency, the dimension of R should not be excessive (say, $n_Z \leq 1000$). This is guaranteed if the number of nonlinear variables is "moderate".

If all the constraints are linear, then the computation of R at the start of phase 2 can be avoided. For linear constraints, Z will not change between subproblems. In addition, since H is a limited-memory quasi-Newton approximation to Lagrangian Hessian, R can be modified for the updates to H at each major iteration. Therefore, the R at the start of each subproblem will be the correct R such that $R^TR = Z^THZ$, and the factorization of the reduced-Hessian is not necessary. This implies that SNOPT will be more efficient on problems where all the constraints are linear.

If the QP contains linear variables, H is positive semi-definite and R may be singular with at least one zero diagonal. In this case, an inertia-controlling active-set strategy is used to ensure that only the last diagonal of R can be zero. (See [21] for discussion of a similar strategy for indefinite quadratic programming.)

At the initial point, some variables are fixed at their current value so that enough temporary bound constraints are included in the working set to make R nonsingular. Thereafter, R can become singular only when a constraint is deleted from the working set (in which case no further constraints are deleted until R becomes nonsingular).

2.3 The Algorithm SQOPT

2.3.1 Computation of the Lagrange Multipliers

If the reduced gradient is zero, (x, s) is a subspace stationary point with respect to the current working set. During phase 1, the reduced gradient will usually be zero only at a vertex (although it may be zero elsewhere in the presence of constraint dependencies). During phase 2, a zero reduced gradient implies that x minimizes the quadratic objective when the constraints in the working set are treated as equalities. At a subspace stationary point, Lagrange multipliers λ are defined from the equations $W^T\lambda = g(x)$. A Lagrange multiplier λ_j corresponding to an inequality constraint in the working set is said to be *optimal* if $\lambda_j \leq \sigma$ when the associated constraint is at its *upper bound*, or if $\lambda_j \geq -\sigma$ when the associated constraint is at its *lower bound*, where σ depends on the optimality tolerance. If a multiplier is non-optimal, the objective function (either the true objective or the sum of infeasibilities) can be reduced by continuing the minimization with the corresponding constraint excluded from the working set (this step is sometimes referred to as "deleting" a constraint from the working set). If optimal multipliers occur during the feasibility phase but the sum of infeasibilities is not zero, there is no feasible point.

The special form (2.9) of the working set allows the multiplier vector, the solution of $W^T\lambda = g$, to be written in terms of the vector

$$d = \begin{pmatrix} g \\ 0 \end{pmatrix} - \begin{pmatrix} A & -I \end{pmatrix}^T \pi = \begin{pmatrix} g - A^T\pi \\ \pi \end{pmatrix}, \tag{2.12}$$

where π satisfies the equations $B^T\pi = g_B$, and g_B denotes the basic components of g. The components of π are the Lagrange multipliers λ_j associated with the equality constraints $Ax - s = b$. The vector d_N of nonbasic components of d consists of the Lagrange multipliers λ_j associated with the upper and lower bound constraints in the working set. The vector d_S of superbasic components of d is the reduced gradient g_Z (2.11). The vector d_B of basic components of d is zero, by construction.

2.3.2 Computing the Search Direction

If the reduced gradient is not zero, Lagrange multipliers need not be computed and the search direction is given by $p = Zp_Z$, where p_Z is defined below. The step length is chosen to maintain feasibility with respect to the satisfied constraints.

There are two possible choices for p_Z, depending on whether or not H_Z is singular. If H_Z is nonsingular, R is nonsingular and p_Z is computed from the equations,

$$R^TRp_Z = -g_Z, \tag{2.13}$$

where g_Z is the reduced gradient at x. In this case, $(x, s) + p$ is the minimizer of the objective function subject to the working-set constraints being treated as equalities. If $(x, s) + p$ is feasible, α is defined to be one. In this case, the reduced gradient at $(\bar{x}, \bar{s})$ will be zero, and Lagrange multipliers are computed at the next iteration. Otherwise, α is set to α_N, the step to the boundary of the "nearest" constraint along p. This constraint is added to the working set at the next iteration.

If H_Z is singular, then R must also be singular, and an inertia-controlling strategy is used to ensure that only the last diagonal element of R is zero. In this case, p_Z satisfies

$$p_Z^T H_Z p_Z = 0 \quad \text{and} \quad g_Z^T p_Z \leq 0, \tag{2.14}$$

which allows the objective function to be reduced by any step of the form $(x, s) + \alpha p$, $\alpha > 0$. The vector $p = Zp_Z$ is a direction of unbounded descent for the QP in the sense that the QP objective is linear and decreases without bound along p. If no finite step of the form $(x, s) + \alpha p$ $(\alpha > 0)$ reaches a constraint not in the working set, the QP is unbounded and SQOPT terminates at (x, s). Otherwise, α is defined as the maximum feasible step along p and a constraint active at $(x, s) + \alpha p$ is added to the working set for the next iteration.

As constraints are added to the working set, the working-set matrix W remains full rank.

2.3.3 The Active Set Strategy

When the working set changes, it is necessary to update the upper-triangular factor of $Z^T HZ$. There are two possible changes to the working set, a constraint is either added to, or deleted from, the working set. For problem (2.1), the working set always includes the linear equality constraints, therefore, changes in the working set only involve bound constraints. For this reason, it is common to think of *variables* being added to, or deleted from, the working set, since each bound constraint is associated with a single variable.

In an active-set method, these changes are defined in terms of the type of variable being added or deleted. When a variable, or constraint, is deleted from the working set, this is equivalent to adding a superbasic variable to the active set. If a constraint is added to the the working set, then either a superbasic or a basic variable is deleted from the active set. Thus, adding or deleting a constraint from the working set corresponds to the removal or addition of a column of S, and possibly the replacement of a column of the basis B. This changes the null-space matrix Z defined by (2.10) and results in a change to Z^THZ. There are three cases for updating R.

1. If a nonbasic variable moves off its bound, the associated column is moved from N to S. This is referred to as *adding a superbasic variable.*

2. If a superbasic variable moves to one of its bounds, the associated column is moved from S to N. This is referred to as *deleting a superbasic variable.*

3. If a basic variable moves to one of its bounds, the associated column is moved from B to N, and an appropriate column of S is moved to B. This is referred to as *deleting a basic variable.*

Adding a superbasic variable is equivalent to expanding R by a new row and column. The update to Z is given by $\bar{Z} = (\ Z \quad z\ )$, where

$$z = \begin{pmatrix} -B^{-1}s_t \\ e_t \\ 0 \end{pmatrix}, \tag{2.15}$$

with s_t the new column of S. If the update to R is defined as

$$\bar{R} = \begin{pmatrix} R & v \\ 0 & \rho \end{pmatrix},$$

then the identity $\bar{R}^T\bar{R} = \bar{Z}^TH\bar{Z}$, implies that v and ρ can be calculated by

$$R^Tv = Z^THz \qquad \text{and} \qquad \rho^2 = z^THz - v^Tv. \tag{2.16}$$

On the other hand, deleting a superbasic variable is equivalent to removing a row and column from R. The relevant column of R is removed and the remaining columns are permuted forward. A forward sweep of plane rotations is then used to restore R to upper-triangular form.

Deleting a basic variable is more complicated because it first requires swapping a column of B with a column of S. This process is sometimes referred to as a *BS swap*. If the p-th column of B is being moved to N, let the vector y satisfy $y = S^T w$ for $B^T w = e_p$. The elements $(y)_j$ can be thought of as the pivot elements that would arise if the j-th column of S were selected for the basis change. The q-th superbasic could be selected such that $|(y)_q| = y_{max} = \max_j |(y)_j|$. If y_{max} were at, or very near one of its bounds, another BS swap update might be required in only a few iterations. Since the BS swap requires extra work for an associated update to R (see Lemma 1), this condition is relaxed. Let $\mathcal{Q}$ be the set of superbasic variables associated with elements of y of sufficient magnitude, defined by

$$\mathcal{Q} = \{\, j \mid |(y)_j| \geq \theta |y_{max}| \,\},$$

where $0 < \theta < 1$. The p-th basic variable is swapped with the q-th superbasic, where q is chosen by

$$\{\; q \mid \nu_q = \max_{j \in \mathcal{Q}} \{\nu_j\} \;\}. \tag{2.17}$$

with $\nu_j = \min\{\, |(x,s)_j - l_j|,\ |(x,s)_j - u_j| \,\}$. This method of selecting the superbasic variable for the BS swap is the same as that used in MINOS [30].

The matrix R must be updated to reflect the swap. The update to R for the BS swap uses the following result, due to Murtaugh and Saunders [30]. This proof, which has not appeared elsewhere, is included for completeness.

Lemma 1 *Let Z be as in (2.10), with $B \in \mathbb{R}^{m\times m}$ nonsingular and $S \in \mathbb{R}^{m\times n_s}$; and R be upper-triangular with $R^T R = Z^T H Z$. If the p-th column of B and the q-th column of S are swapped, let $y = S^T w$, $B^T w = e_p$, assume $y \neq 0$. Then the update to R is given by*

$$\bar{R} = R - \frac{1}{(y)_q} r_q (y + e_q)^T, \tag{2.18}$$

where $(y)_q$ is the q-th element of y and r_q is the q-th column of R.

Proof. Let P be the permutation matrix used to define Z in (2.10). Define

$$Q = P \begin{pmatrix} -B^{-1}S & B^{-1} & 0 \\ I_S & 0 & 0 \\ 0 & 0 & I_N \end{pmatrix} = \begin{pmatrix} Z & Y & I_* \end{pmatrix},$$

so that

$$Q^{-1} = \begin{pmatrix} 0 & I_S & 0 \\ B & S & 0 \\ 0 & 0 & I_N \end{pmatrix} P^T.$$

Note that $\bar{Q}^{-1}$ is just a column permutation of Q^{-1} after its q-th row is replaced by e_p^T to preserve I_S. Therefore,

$$\bar{Q}^{-1} = \left(Q^{-1} + e_q(e_p - e_{m+q})^T\right) \widehat{P},$$

where post-multiplication by $\widehat{P}$ swaps columns p and $(m+q)$. Then

$$\bar{Q}^{-1} = \left(I + e_q(e_p - e_{m+q})^T Q\right) Q^{-1}\widehat{P}.$$

If y is the vector $y = S^T w$, where w the solution of $B^T w = e_p$, then

$$\bar{Q}^{-1} = (I + e_q v^T)Q^{-1}\widehat{P}, \quad \text{where} \quad v = \begin{pmatrix} -(y + e_q) \\ w \\ 0 \end{pmatrix}.$$

Since $y \neq 0$ by assumption, it follows that $(y)_q \neq 0$. Therefore, the Sherman-Morrison-Woodbury formula may be applied to give

$$\bar{Q} = \widehat{P}^T Q \left(I + \frac{1}{(y)_q} e_q v^T\right) \triangleq \widehat{P}^T Q M,$$

and it follows that

$$Q^T H Q = Q^T \widehat{P}\widehat{P}^T H \widehat{P}\widehat{P}^T Q = M^{-T}(\widehat{P}\bar{Q})^T H (\widehat{P}\bar{Q}) M^{-1}.$$

If $\bar{Q}$ is assumed to include the permutation $\widehat{P}$, so that $\bar{Q} = \widehat{P}\bar{Q}$, then $\bar{Q}^T H \bar{Q} = M^T Q^T H Q M$. Since $R^T R = Z^T H Z$ and

$$Q^T H Q = \begin{pmatrix} Z^T H Z & Z^T H Y & Z^T H I_* \\ Y^T H Z & Y^T H Y & Y^T H I_* \\ I_*^T H Z & I_*^T H Y & I_*^T H I_* \end{pmatrix},$$

if M_S denotes the leading principal submatrix of M, then

$$M_S = I_S - \frac{1}{(y)_q} e_q (y + e_q)^T,$$

and

$$\bar{R} = R M_S = R - \frac{1}{(y)_q} r_q (y + e_q)^T,$$

where $(y)_q$ is the q-th element of y and r_q is the q-th column of R. ∎

After R is updated via (2.18), $\bar{R}$ is restored to upper-triangular form using two sweeps of plane rotations. Once the BS swap is complete, the "new" p-th column of S is moved to N by updating $\bar{R}$ as described above for deleting a superbasic variable.

The implementation of SQOPT stores only the non-zero entries of R, thus the procedures for updating R are done in place, using the minimum extra work space necessary.

2.3.4 Algorithm SQOPT

The following pseudo-code description of Algorithm SQOPT is presented to help clarify the preceding discussion and to provide a reference for the algorithms presented in *Chapter 3* and *Chapter 4*. While this chapter provides the details needed in the development of these algorithms, it is not intended to be a complete description of SQOPT. For a more complete description of SQOPT, see [15].

Algorithm 2.3.1. *SQOPT*

Find a feasible point, (x, s)
Factorize the initial reduced Hessian $R^T R = Z^T H Z$
Calculate the initial g and ψ
while true
    Calculate multipliers π and $g_z = Z^T g$
    stationary-point $= \|g_z\| <$ `RGTOL`
    new-SB $=$ **false**
    if *stationary-point* **then**
        Calculate multipliers, λ
        Check (π, λ) for optimality and set *optimal*
        if *optimal* **then**
            stop
        else
            Select new superbasic
            new-SB $=$ **true**
        end if
    end if
    if *new-SB* **then**
        Update R and g_z
    end if
    Compute p_z to satisfy (2.13) or (2.14)
    $p = Zp_z$
    $\alpha_N =$ maximum feasible step along p
    if R is nonsingular **then**
        hit-constraint $= \alpha_N < 1$
    else
        hit-constraint $= \alpha_N < \infty$
        if not *hit-constraint* **then stop**
    end if
    if *hit-constraint* **then**
        $\alpha = \alpha_N$
    else
        $\alpha = 1$
    end if
    $(x, s) \leftarrow (x, s) + \alpha p$

$g \leftarrow g + \alpha Hp$

Calculate ψ

if *hit-constraint* **then**

Update R

end if

end while

When a superbasic is added, g_z is updated by adding the element corresponding to $z^T g$, where z is defined by (2.15). If λ_q is the multiplier for the new superbasic, then it follows from (2.12) that $z^T g = \lambda_q$ and

$$\bar{g}_Z = \begin{pmatrix} g_Z \\ \lambda_q \end{pmatrix}.$$

For details on calculating a descent direction with zero curvature to satisfy (2.14) when R is singular, see [15].

The reduced gradient tolerance, `RGTOL`, is used to determine convergence on the current subspace of active constraints. The value of `RGTOL` is typically 10^{-6}.

The reduced-Hessian QP algorithm presented in *Chapter 3* is derived directly from SQOPT. In contrast, the Schur-complement QP algorithm of *Chapter 4* is a distinctly different algorithm, however, it does generate the same search directions as SQOPT.

Chapter 3

The Conjugate-Gradient Method

A well known approach for solving a large symmetric positive-definite linear system $Cx = f$, without factoring C, is the conjugate-gradient method of Hestenes and Stiefel [26] (see also [10, 14, 25, 31]). The conjugate-gradient method is an iterative method based on a five-term recurrence relation (see below). The method proposed in this chapter is based on using a conjugate-gradient algorithm to solve the reduced-Hessian systems, $H_Z p_Z = -g_Z$.

A conjugate-gradient method generates a sequence of search directions that are mutually conjugate with respect to C. The conjugate-gradient method exhibits finite termination in exact arithmetic; in particular, if C has n_e distinct eigenvalues, then at most n_e iterations are required for convergence. Although, the CG method was originally conceived as an exact method, in practice, rounding errors degrade the numerical conjugacy of the search directions and convergence is characteristic of an iterative method.

The conjugate-gradient method works much better on systems that have only a few distinct eigenvalues, or clustered eigenvalues [27]. As a consequence, a common practice is to *precondition* the linear system to accelerate the convergence. Preconditioning involves applying CG to a transformed system that has a more favorable eigenvalue distribution (see below).

3.1 Linear Systems and Least-Squares Problems

Consider applying the conjugate-gradient (CG) method to the system

$$Cx = f, \tag{3.1}$$

where C is symmetric and positive definite. Given x_k, β_{k-1} and d_{k-1}, a CG iteration is defined by

$$\begin{aligned} d_k &= -r_k + \beta_{k-1} d_{k-1}, \\ \alpha_k &= \frac{\|r_k\|^2}{d_k^T C d_k}, \\ x_{k+1} &= x_k + \alpha_k d_k, \\ r_{k+1} &= r_k + \alpha_k C d_k, \\ \beta_k &= \frac{\|r_{k+1}\|^2}{\|r_k\|^2}. \end{aligned} \tag{3.2}$$

For the first iteration, $\beta_{-1} = 0$ and $d_{-1} = 0$. The iterates $\{x_k\}$ are approximations to the solution, and the search directions $\{d_k\}$ are such that $d_i C d_j = 0$ for $i \neq j$. The residual at each x_k is given by $r_k = Cx_k - f$. Note that the CG method only uses C as part of a matrix-vector product.

Suppose $C = A^T A$ for some $A \in \mathbb{R}^{m \times n}$, where $m \geq n$ and $\operatorname{rank}(A) = n$. Similarly, let $f = A^T b$ for some b. In this case, the original system $Cx = f$ is equivalent to

$$A^T A x = A^T b. \tag{3.3}$$

These are the normal equations for the least-squares (LS) problem

$$\min \|b - Ax\|^2. \tag{3.4}$$

It follows that the solution of (3.1) can be found by solving (3.4). In this case, the solution x can be found using the least-squares CG algorithm LSQR [32] on (3.4). LSQR requires matrix-vector products of the form Av and $A^T u$, instead of the $A^T Av$ form required by a general symmetric CG algorithm.

The sensitivity of least-squares solutions to perturbations in A and b are summarized in the following theorem. Define $\kappa(A)$ to be the spectral condition number of A given by $\kappa(A) = \|A\|\|A^+\|$, where A^+ is the pseudo-inverse of A. If A has full rank with $m \geq n$, $A^+ = (A^TA)^{-1}A^T$.

Theorem 1 *Let* $x = \arg\ \min\ \|Ax - b\|^2$ *and* $\widehat{x} = \arg\ \min\ \|(A + \delta A)\widehat{x} - (b + \delta b)\|$, *where* A *and* δA *are in* $I\!R^{m\times n}$ *with* $m \geq n$ *and* $0 \neq b$ *and* δb *are in* $I\!R^m$. *Furthermore, let* r *and* $\widehat{r}$ *be the residuals associated with* x *and* $\widehat{x}$ *respectively. If*

$$\epsilon = \max\left\{\frac{\|\delta A\|}{\|A\|}, \frac{\|\delta b\|}{\|b\|}\right\} < \frac{1}{\kappa(A)}$$

and $\sin\theta = \rho/\|b\| \neq 1$, *where* $\rho = \|Ax - b\|$, *then*

$$\frac{\|\widehat{x} - x\|}{\|x\|} \leq \epsilon\ \left\{\frac{2\kappa(A)}{\cos\theta} + \tan\theta\kappa(A)^2\right\} + \mathcal{O}(\epsilon^2), \tag{3.5}$$

$$\frac{\|\widehat{r} - r\|}{\|b\|} \leq \epsilon\ (1 + 2\kappa(A))\min(1, m - n) + \mathcal{O}(\epsilon^2). \tag{3.6}$$

Proof. See Golub and Van Loan [24]. ∎

Golub and Van Loan show that

$$\tan\theta = \rho/\sqrt{\|b\|^2 - \rho^2},$$

therefore, it follows from Theorem 1 that the sensitivity of the LS solution is proportional to $\kappa(A) + \rho\kappa(A)^2$. When $Ax = b$ is incompatible, the sensitivity is dominated by $\kappa(A)^2$ if ρ is not close to zero. The condition number of C is $\kappa(C) = \kappa(A^TA) = \kappa(A)^2$. Thus, it follows that the sensitivity of solutions of $Cx = d$, and the least-squares problem (3.4), are both proportional to $\kappa(C)$. Therefore, if $C = A^TA$ and $f = A^Tb$, the relative accuracy of the solutions from the CG algorithm for (3.1) and from LSQR for (3.4) will be comparable, but products of the form A^TAv are avoided.

Solving the reduced-Hessian system $H_Zp_Z = -g_Z$ using the CG method requires forming matrix-vector products of the form $H_Zu = Z^THZu$. Since both H and Z are available implicitly, a symmetric linear CG algorithm such as SYMMLQ [31] can be used. However, by using a CG algorithm based on a least-squares formulation we benefit from the numerical advantages discussed in [32].

3.2 Background on LSQR

Algorithm LSQR is equivalent to the CG method in exact arithmetic. However, in the least-squares context, LSQR has been shown to be more reliable than standard CG methods. In Paige and Saunders [32], the authors point out that the least-squares adaptation of symmetric CG is always more reliable than applying the symmetric CG algorithm to the normal equations. In addition, they show that when A is ill-conditioned, LSQR should be more reliable than other least-squares CG algorithms such as CGLS [3].

The algorithm LSQR is based on the bidiagonalization routine of Golub and Kahan [23]. This procedure generates a set of vectors $\{u_i, v_i\}$ using the following algorithm.

Algorithm 3.2.1. *BIDIAG*

$\beta_1 u_1 = b$

$\alpha_1 v_1 = A^T u_1$

for $i = 1, 2, \ldots$

  $\beta_{i+1} u_{i+1} = A v_i - \alpha_i u_i$

  $\alpha_{i+1} v_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$

end

The scalars $\alpha_i \geq 0$ and $\beta_i \geq 0$ are chosen so that $\|u_i\| = \|v_i\| = 1$. By defining

$$\begin{aligned} U_k &\equiv (u_1, u_2, \ldots, u_k), \\ V_k &\equiv (v_1, v_2, \ldots, v_k), \end{aligned} \qquad B_k \equiv \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_k \\ & & & \beta_{k+1} \end{pmatrix}, \tag{3.7}$$

the recurrence relations of *BIDIAG* may be written in matrix form as

$$U_{k+1}(\beta_1 e_1) = b, \tag{3.8}$$

$$A V_k = U_{k+1} B_k, \tag{3.9}$$

$$A^T U_{k+1} = V_k B_k^T + \alpha_{k+1} v_{k+1} e_{k+1}^T. \tag{3.10}$$

In exact arithmetic, $U_{k+1}^T U_{k+1} = I$ and $V_{k+1}^T V_{k+1} = I$.

LSQR uses the quantities generated by *BIDIAG* to solve the LS problem (3.4). Let x_k, r_k and t_{k+1} be defined by

$$x_k = V_k y_k, \tag{3.11}$$

$$r_k = b - A x_k, \tag{3.12}$$

$$t_{k+1} = \beta_1 e_1 - B_k y_k, \tag{3.13}$$

for some vector y_k. Using (3.8) and (3.9) it can be shown that

$$r_k = U_{k+1} t_{k+1}.$$

The quantity $\|r_k\|$ can be minimized by choosing y_k to minimize $\|t_k\|$, i.e., by defining y_k as the solution of

$$\min \|\beta_1 e_1 - B_k y_k\|. \tag{3.14}$$

The LSQR algorithm is derived by using the standard QR decomposition of B_k to solve (3.14).

3.3 Defining the Least-Squares Problem

In the SQP method considered here, the matrix H_k of the k-th QP subproblem (1.2) is a limited-memory quasi-Newton approximation to the Lagrangian Hessian. The updating scheme for H_k described in Section 2.1.3 is based on the BFGS update. The product form of the BFGS update for H_k is given by

$$H_{k+1} = (I + v_k u_k^T) H_k (I + u_k v_k^T), \tag{3.15}$$

where

$$u_k = \frac{1}{\|H_k s_k\|} s_k \quad \text{and} \quad v_k = \frac{1}{\sqrt{y_k^T s_k}} y_k - \frac{1}{\|H_k s_k\|} H_k s_k.$$

Here, s_k is the change in the position vector and y_k is the corresponding change in the gradient vector. By modifying the updating scheme used for H_k according to (3.15), a matrix G_k can be defined such that $H_k = G_k^T G_k$. This alternative limited-memory scheme stores the update vectors for G_k. The update vectors u_k and v_k are given by

$$u_k = \frac{1}{\|G_k \delta_k\|}\delta_k \qquad \text{and} \qquad v_k = \frac{1}{\sqrt{y_k^T \delta_k}} y_k - \frac{1}{\|G_k \delta_k\|} G_k^T G_k \delta_k,$$

where δ_k and y_k are defined by (2.4). Each update to G_k is defined by $G_{k+1} = G_k(I + u_k v_k^T)$, which implies

$$G_{k+1} = G_r(I + u_r v_r^T)(I + u_{r+1} v_{r+1}^T) \cdots (I + u_k v_k^T) = G_r \prod_{i=r}^{k} (I + u_i v_i^T),$$

where the nonsingular G_r is defined by $G_r^T G_r = H_r$, for the H_r described in Section 2.1.3.

This limited-memory scheme easily allows for the formation of matrix-vector products with the current G_k; in particular, a product of the form $w = G_k z$ can be computed using the following algorithm.

Algorithm 3.3.1. *(Computation of $w = G_k z$)*

$w = z$

for $i = k-1 : -1 : r$

$\quad w = w + (v_i^T w) u_i$

end

$w = G_r w$

A similar loop can be defined for computing $w = G_k^T z$. In addition, by proper application of the Sherman-Morrison-Woodbury formula, G_k^{-T} is given by

$$G_k^{-T} = G_r^{-1} \prod_{i=r}^{k-1} (I - \frac{1}{1 + u_i^T v_i} v_i u_i^T).$$

The solution of systems with G_k^T can be obtained by computing $w = G_k^{-T} z$ using the following algorithm.

Algorithm 3.3.2. *(Computation of $w = G_k^{-T} z$)*

$w = z$

for $i = k - 1 : -1 : r$

  $w = w - (u_i^T w)/(1 + u_i^T v_i) v_i$

end

$w = G_r^{-1} w$

The work required for computing $w = G_k z$ or $w = G_k^T z$ is $n(k - r + 1) + (k - r)$ flops, while solving $G_k^T w = z$ requires $(2n + 1)(k - r) + 2(k - r)$ flops and one division.

The subscript k is now dropped from the relevant quantities since only a single subproblem is being considered. The direction p_Z can be found by using LSQR to solve the LS problem (3.4) for the appropriate A and b. Let A and b be defined by

$$A = GZ \quad \text{and} \quad b = G^{-T} g, \tag{3.16}$$

where g is the current QP gradient. Then the modified reduced-Hessian system, $Z^T G^T G Z p_Z = Z^T g$, are the normal equations for the least-squares problem (3.4). If x is the solution to problem (3.4), then $p_Z = -x$.

For the definition given (3.16), $A \in I\!R^{n \times n_s}$, with $n > n_s$ and $\text{rank}(A) = n_s$. In addition, the system $Ax = b$ is usually incompatible, i.e., $\rho \neq 0$. Thus, the sensitivity analysis is similar to that discussed in Section 3.1. The following lemma corresponds to Theorem 1 with a perturbation to $A = GZ$ of the form $\delta A = G\delta Z$.

Lemma 2 *Let $x = \arg\ \min\ \|GZx - G^{-T}g\|^2$ and $\widehat{x} = \arg\ \min\ \|G(Z + \delta Z)\widehat{x} - G^{-T}(g + \delta g)\|$, where Z and δZ are in $I\!R^{n \times n_s}$ with $n \geq n_s$ and $0 \neq g$ and δg are in $I\!R^n$. Furthermore, let r and $\widehat{r}$ be the residuals associated with x and $\widehat{x}$ respectively. If*

$$\epsilon = \max \left\{ \frac{\|G\delta Z\|}{\|GZ\|}, \frac{\|G^{-T}\delta g\|}{\|G^{-T}g\|} \right\} < \frac{\sigma_n(GZ)}{\sigma_1(GZ)}$$

and, assuming $\|G^{-T}h\| \leq \|G^{-T}g\|$ and $\sin\theta = \rho/\|G^{-T}g\| \neq 1$, where $\rho = \|GZx -$

$G^{-T}g\|$, *then*

$$\frac{\|\widehat{x}-x\|}{\|x\|} \le \epsilon\left\{\frac{\kappa(GZ)}{\cos\theta}(1+\gamma\cos\theta)+\gamma\tan\theta\kappa(GZ)^2\right\}+\mathcal{O}(\epsilon^2), \tag{3.17}$$

$$\frac{\|\widehat{r}-r\|}{\|G^{-T}g\|} \le \epsilon(1+2\gamma\kappa(A))\min(1,m-n)+\mathcal{O}(\epsilon^2), \tag{3.18}$$

where

$$\gamma = \frac{\|G\|\|Z\|}{\|GZ\|}.$$

Proof. The proof follows that of Theorem 1 from Golub and Van Loan [24]. Let E and h be defined by $E = \delta Z/\epsilon$ and $h = \delta g/\epsilon$. Since $\|G\delta Z\| < \sigma_n(GZ)$ by assumption, it can be shown that $\text{rank}(GZ + GEt) = n_s$ for all $t \in [0,\epsilon]$. Thus, the solution $x(t)$ such that

$$(Z+Et)^TG^TG(Z+Et)x(t) = (Z+Et)^TG^TG^{-T}(g+ht) \tag{3.19}$$

is continuously differentiable for all $t \in [0,\epsilon]$. Since $x = x(0)$ and $\widehat{x} = x(\epsilon)$, it follows that

$$\widehat{x} = x + \epsilon\dot{x}(0) + \mathcal{O}(\epsilon^2).$$

The assumptions that $g \ne 0$ and $\sin\theta \ne 1$ ensure that x is nonzero, and hence

$$\frac{\|\widehat{x}-x\|}{\|x\|} = \epsilon\frac{\|\dot{x}(0)\|}{\|x\|} + \mathcal{O}(\epsilon^2). \tag{3.20}$$

To obtain a bound for $\|\dot{x}(0)\|$, (3.19) can be differentiated, t set to zero and the equation solved for $\dot{x}(0)$. This gives

$$\dot{x}(0) = H_Z^{-1}\{Z^TG^T(G^{-T}h - GEx) + E^TG^Tr\}. \tag{3.21}$$

It can be shown that $\|E\| \le \|Z\|$ and $\|h\| \le \|g\|$. Therefore, since $\|G^{-T}h\| \le \|G^{-T}g\|$, substituting (3.21) into (3.20) and taking norms yields

$$\begin{aligned}\frac{\|\widehat{x}-x\|}{\|x\|} \;\le\; & \epsilon\left\{\|GZ\|\|H_Z^{-1}(GZ)^T\|\left(\frac{\|G^{-T}g\|}{\|GZ\|\|x\|}+\gamma\right)\right.\\ & \left. +\;\gamma\rho\frac{\|H_Z^{-1}\|\|GZ\|}{\|x\|}\right\}+\mathcal{O}(\epsilon^2).\end{aligned}$$

The following relations are easily established using the definition of the pseudo-inverse, and the singular values of GZ:

$$\begin{aligned} \|GZ\|\|H_Z^{-1}(GZ)^T\| &= \kappa(GZ), \\ \|GZ\|^2\|H_Z^{-1}\| &= \kappa(GZ)^2. \end{aligned} \tag{3.22}$$

In addition, since $(GZ)^T(GZx - G^{-T}g) = 0$, it follows that GZx and $GZx - G^{-T}g$ are orthogonal with

$$\|GZ\|^2\|x\|^2 \geq \|G^{-T}g\|^2 - \rho^2.$$

Finally, using (3.22)

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \epsilon \left\{ \frac{\kappa(GZ)}{\cos\theta}(1 + \gamma\cos\theta) + \gamma\tan\theta\kappa(GZ)^2 \right\} + \mathcal{O}(\epsilon^2),$$

which proves (3.17).

To establish (3.18), define $r(t)$ by the differentiable function

$$r(t) = G^{-T}(g + ht) - G(Z + Et)x(t).$$

Using (3.21), it can be shown that

$$\dot{r}(0) = (I - GZ(H_Z^{-1})(GZ)^T)(G^{-T}h - GEx) - GZ(H_Z^{-1})E^TG^Tr.$$

Noting that $r = r(0)$ and $\hat{r} = r(\epsilon)$, and using $\|\hat{r} - r\| = \epsilon\|\dot{r}(0)\| + \mathcal{O}(\epsilon^2)$, yields

$$\begin{aligned} \frac{\|\hat{r} - r\|}{\|G^{-T}g\|} &= \epsilon\frac{\|\dot{r}(0)\|}{\|G^{-T}g\|} + \mathcal{O}(\epsilon^2) \\ &\leq \epsilon \left\{ \|I - GZ(H_Z^{-1})(GZ)^T\| \left(1 + \frac{\|G\|\|Z\|\|x\|}{\|G^{-T}g\|}\right) \right. \\ &\qquad \left. + \|GZ(H_Z^{-1})\|\|G\|\|Z\|\frac{\rho}{\|G^{-T}g\|} \right\} + \mathcal{O}(\epsilon^2). \end{aligned}$$

The inequality (3.18) follows from

$$\begin{gathered} \|GZ\|\|x\| = \|GZ\|\|(GZ)^+G^{-T}g\| \leq \kappa(GZ)\|G^{-T}g\|, \\ \rho = \|I - GZ(H_Z^{-1})(GZ)^TG^{-T}g\| \leq \|I - GZ(H_Z^{-1})(GZ)^T\|\|G^{-T}g\|, \end{gathered}$$

and

$$\|I - GZ(H_Z^{-1})(GZ)^T\| = \min(m - n, 1). \quad \blacksquare$$

The implication of this result is that the sensitivity of the LS problem with $A = GZ$ is proportional to $\gamma(\kappa(GZ) + \kappa(GZ)^2)$. Thus, the appearance of G has the effect of increasing the sensitivity by the factor $\gamma \geq 1$.

Each iteration of LSQR for solving (3.4) will require matrix-vector products involving GZ and $(GZ)^T$. In addition, forming the right-hand side requires the solution of a system with G^T. The modified updating scheme for H allows for the formation of matrix-vector products with G, G^T and G^{-T}. In addition, calculating matrix-vector products with Z and Z^T can be done using the implicit form given in Section 2.2.

If there are linear variables, then H is positive semidefinite and has the form (2.6) (see Section 2.1.2). This type of *structural* singularity can be handled by solving the reduced-Hessian system as a damped least-squares problem. In the linear variable case, H is given by

$$H = \begin{pmatrix} G^T G & \\ & 0 \end{pmatrix},$$

where G is defined by the modified updating scheme for H. Let H be approximated by

$$H \approx \begin{pmatrix} G^T G & \\ & \sigma^2 I \end{pmatrix},$$

for some small positive σ. The direction p_Z can be found by using LSQR to solve the damped LS problem of the form (3.4), where

$$A = \begin{pmatrix} G & \\ & \sigma I \end{pmatrix} Z \quad \text{and} \quad b = \begin{pmatrix} G^{-T} & \\ & \sigma^{-1} I \end{pmatrix} g.$$

3.4 Preconditioning the Least-Squares System

The convergence of the CG algorithm resulting from the recurrence relations (3.2) can be characterized by the following theorems.

Theorem 2 *If $C = I + B$ is an $n \times n$ symmetric positive-definite matrix and* $\mathrm{rank}(B) = r$, *then the CG algorithm applied to $Cx = f$ converges in at most $r+1$ steps.*

Proof. See Golub and Van Loan [24]. ∎

This result can be used to show that if C has n_e distinct eigenvalues, then the CG algorithm will require at most n_e steps.

The second theorem shows that the CG algorithm will converge more rapidly when C is well conditioned, or has clustered eigenvalues. The theorem is stated in terms of the elliptic norm

$$\|w\|_C = \sqrt{w^T C w}.$$

Theorem 3 *Suppose $C \in \mathbb{R}^{n\times n}$ is symmetric positive-definite and $d \in \mathbb{R}^n$. If the CG algorithm applied to $Cx = f$ produces iterates $\{x_k\}$ and $\kappa(C) = \|C\|\|C^{-1}\|$, then*

$$\|x - x_k\|_C \le 2\|x - x_0\|_C \left(\frac{\sqrt{\kappa(C)} - 1}{\sqrt{\kappa(C)} + 1}\right).$$

Proof. See Luenberger [28]. ∎

The intent of preconditioning is to transform the original system to one that exploits the convergence properties of the CG algorithm. Let M be a nonsingular preconditioner for the system $Cx = f$. The preconditioned matrix $M^{-T}CM^{-1}$ will have fewer distinct eigenvalues than C, or its eigenvalues will exhibit better clustering than those of C.

The term *exact preconditioner* is often used in the context of CG methods. Loosely speaking, an exact preconditioner is one that gives a preconditioned system with a block of unit eigenvalues. For example, $M = C^{-\frac{1}{2}}$ is an exact preconditioner since the preconditioned system is the identity.

In either case, applying CG to the preconditioned system $M^{-T}CM^{-1}y = M^{-T}f$ may be preferable to solving $Cx = f$ directly. The solution x can be obtained from y by solving $Mx = y$. For $y_0 = Mx_0$ and $r_0 = M^{-T}CM^{-1}y_0 - M^{-T}f$ (and again $\beta_{-1} = 0$ and $d_{-1} = 0$), the preconditioned CG method can be defined for $k = 0, 1, 2, \ldots$ as

$$\begin{aligned} d_k &= -r_k + \beta_{k-1}d_{k-1}; \\ \alpha_k &= \frac{r_k^T r_k}{d_k^T(M^{-T}CM^{-1})d_k}; \\ y_{k+1} &= y_k + \alpha_k d_k; \\ r_{k+1} &= r_k + \alpha_k(M^{-T}CM^{-1})d_k; \\ \beta_k &= \frac{\|r_{k+1}\|_2^2}{\|r_k\|_2^2}, \end{aligned}$$

and x_k is the solution of $Mx_k = y_k$. Note that the preconditioned iterations require additional solves with M. Thus, M must be such that systems of the form $Mx = y$ can be solved efficiently. For example, M could be a diagonal or triangular matrix.

If $C = A^T A$ and $d = A^T b$, then x can be obtained by using LSQR to solve the preconditioned least-squares problem

$$\underset{y}{\text{minimize}}\ \|b - AM^{-1}y\|. \tag{3.23}$$

The basis for the CG method is to use LSQR to solve (3.23) when A and b are defined as in (3.16), and M is one of two different preconditioners investigated in Sections 3.5 and 3.6.

The first preconditioner involves calculating the Cholesky factor R_r of the $n_r \times n_r$ leading principal submatrix of the reduced Hessian, denoted by $Z_r^T H Z_r$. This preconditioner is of the form

$$R_C = \begin{pmatrix} R_r & 0 \\ 0 & D \end{pmatrix},$$

where D is a positive-definite diagonal extension. The second preconditioner involves using R_C with quasi-Newton approximations to R_r and D.

Although $R_r^T R_r = Z_r^T H Z_r$, neither preconditioner is exact. Let the R such that $R^T R = Z^T H Z$ be partitioned by

$$R = \begin{pmatrix} R_r & R_{rs} \\ & R_s \end{pmatrix}, \tag{3.24}$$

where, $R_r \in I\!R^{n_r \times n_r}$, $R_{rs} \in I\!R^{n_r \times (n_s - n_r)}$ and $R_s \in I\!R^{(n_s-n_r)\times(n_s-n_r)}$. The following result shows that both R_r and R_{rs} are required for a block preconditioner based on R to be exact.

Theorem 4 *Let R be as in (3.24) and let A be defined as in (3.16). If*

$$M = \begin{pmatrix} R_r & R_{rs} \\ & D \end{pmatrix},$$

where D is positive-definite, then the preconditioned matrix $M^{-T} A^T A M^{-1}$ has at least n_r unit eigenvalues.

Proof. Let Z by partitioned as $Z = (\ Z_r \quad Z_s\ )$, where Z_r has n_r columns and Z_s has $n_s - n_r$ columns. This defines a corresponding block partition of the reduced Hessian, given by

$$Z^T H Z = \begin{pmatrix} Z_r^T H Z_r & Z_r^T H Z_s \\ Z_s^T H Z_r & Z_s^T H Z_s \end{pmatrix}. \tag{3.25}$$

From $R^T R = Z^T H Z$, and substituting from (3.25), it follows that

$$R^{-T} \begin{pmatrix} Z_r^T H Z_r & Z_r^T H Z_s \\ Z_s^T H Z_r & Z_s^T H Z_s \end{pmatrix} R^{-1} = \begin{pmatrix} I_{n_r} & 0 \\ 0 & I_{(n_s-n_r)} \end{pmatrix}, \tag{3.26}$$

for

$$R^{-1} = \begin{pmatrix} R_r^{-1} & -R_r^{-1} R_{rs} R_s^{-1} \\ & R_s^{-1} \end{pmatrix}.$$

Computing both sides of (3.26) and comparing the one-one block, and the off-diagonal blocks, yields

$$R_r^{-T} Z_r^T H Z_r R_r^{-1} = I, \tag{3.27}$$

$$R_{rs} = R_r^{-T} (Z_r^T H Z_s). \tag{3.28}$$

It follows from (3.16) that $A^T A = Z^T H Z$. Thus, with

$$M^{-1} = \begin{pmatrix} R_r^{-1} & -R_r^{-1} R_{rs} D^{-1} \\ 0 & D^{-1} \end{pmatrix},$$

$M^{-T} A^T A M^{-1}$ reduces to

$$\begin{pmatrix} R_r^{-T}(Z_r^T H Z_r) R_r^{-1} & -R_{rs} D^{-1} + R_r^{-T}(Z_r^T H Z_s) D^{-1} \\ -D^{-T} R_{rs} + D^{-T}(Z_s^T H Z_r) R_r^{-1} & D^{-T}(Z_s^T H Z_s - R_{rs}^T R_{rs}) D^{-1} \end{pmatrix}. \tag{3.29}$$

Using (3.27) and (3.28), it follows that

$$M^{-T} A^T A M^{-1} = \begin{pmatrix} I_{n_r} & 0 \\ 0 & B \end{pmatrix},$$

where B is the (2,2) block of (3.29). ∎

While the efficiency of an exact preconditioner would be beneficial, R_{rs} is a $n_r \times (n_s - n_r)$ dense matrix. Since the dimension n_r is of moderate size, for problems where n_s is large, the column dimension $n_s - n_r$ will be large as well. Thus, the expense of computing and maintaining R_{rs} is prohibitive. Therefore, using R_C as a preconditioner will not yield a system with n_r unit eigenvalues, but it is expected that the eigenvalues of the preconditioned system will be more clustered than those of H_Z.

3.5 The Partial Reduced-Hessian Preconditioner

The first preconditioner to be investigated has block diagonal structure

$$R_C = \begin{pmatrix} R_r & 0 \\ 0 & D \end{pmatrix},$$

with R_r the Cholesky factor of $Z_r^T H Z_r$, the $n_r \times n_r$ leading principal submatrix of the reduced Hessian. The matrix D is a positive diagonal matrix whose elements approximate the associated diagonal elements of the Cholesky factor of the full reduced Hessian.

After each QP search direction is calculated and a step taken, the preconditioner R_C must be updated to reflect changes to the working set. When the working set changes, the strategy for updating R_C is to maintain R_r as the Cholesky factor of the *current* $Z_r^T H Z_r$. The two possible changes to the working set correspond to the three updates discussed in Section 2.3.3; adding a superbasic variable, deleting a superbasic variable or deleting a basic variable.

3.5.1 Adding a Superbasic Variable

When a superbasic variable is added, the row and column associated with the new superbasic are added to R_r. Let g_Z be partitioned as $g_Z = \begin{pmatrix} g_1 & g_2 \end{pmatrix}^T$, where $g_1 \in I\!R^{n_r}$ corresponds to the variables that define R_r. The expectation is that as the QP algorithm reduces the norm of the reduced gradient, the reduction in the magnitude of the elements of g_1 associated with R_r will be greater than the corresponding reduction in the elements of g_2 associated with D. By including the new superbasic with the variables defining the dense upper-triangular block, the aim is to emphasize the reduction of the component of the reduced gradient associated with the new variable. Since the dimension of R_r is fixed, one of the variables already associated with R_r must be moved to D. The q-th superbasic is moved, where q is chosen to correspond to the smallest element in magnitude of g_1; i.e.,

$$\{q \mid |(Z^T g)_q| \leq |(Z^T g)_j|, 1 \leq j \leq n_r\}. \tag{3.30}$$

The q-th diagonal of R_r is added to the end of the diagonal matrix D, so that

$$\bar{D} = \begin{pmatrix} D & 0 \\ 0 & r_{qq} \end{pmatrix}.$$

The q-th column of R_r is then deleted using the algorithm for deleting a superbasic from the Cholesky factor of the full reduced Hessian. Columns $q+1$ through n_r are then cyclicly permuted forward, and a sweep of plane rotations are applied to restore R_r to upper-triangular form. After the plane rotations have been applied, the last row and column of R_r are zero. Let $\widehat{R}_r$ be the $(n_r-1)\times(n_r-1)$ upper-triangular submatrix of R_r without the zero row and column.

The calculation of the new column of R_r requires first partitioning S as $S = (\, S_1 \quad S_2 \,)$, where S_1 is the first n_r columns of S and S_2 the remaining $n_s - n_r$ columns. After updating D and deleting the q-th column from R_r, the updated S is given by

$$\bar{S} = (\, \bar{S}_1 \quad s_{n_r} \quad \bar{S}_2 \,),$$

where s_{n_r} is the new column of S. The matrix $\bar{S}_1$ results from deleting the q-th column of S_1, and the updated $\bar{S}_2$ is the result of adding this column to the end of S_2. The new Z is given by

$$\bar{Z} = \begin{pmatrix} -B^{-1}\bar{S} \\ I \\ 0 \end{pmatrix} = \begin{pmatrix} -B^{-1}\bar{S}_1 & -B^{-1}s_{n_r} & -B^{-1}\bar{S}_2 \\ \tilde{I} & e_{n_r} & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} = (\, \bar{Z}_1 \quad w \quad \bar{Z}_2 \,).$$

where $\tilde{I}$ is an identity matrix of dimension $n_r - 1$ augmented by a row of zeros. The update to R_r is given by

$$\bar{R}_r = \begin{pmatrix} \widehat{R}_r & v \\ 0 & \rho \end{pmatrix},$$

where the new column of R_r is calculated from

$$\widehat{R}_r^T v = \bar{Z}_1^T H w \qquad \text{and} \qquad \rho^2 = w^T H w - v^T v. \tag{3.31}$$

The work for calculating the new column is dominated by the cost of forming the right-hand side, $\bar{Z}_1^T H w$. From the discussions in Section 2.2 on Z, and Section 3.3 on $H = G^T G$, it easily follows that the work required to form the right-hand side is at most $m^2 + mn_r + 2nN$ flops. Here, N is the maximum number of limited-memory quasi-Newton updates before a reset. The work estimates are based on using an LU factorization of the matrix B to form products with Z. When a sparse LU factorization is used, there is a corresponding reduction in the work.

Finally, the update to the full R_C is

$$\bar{R}_C = \begin{pmatrix} \bar{R}_r & 0 \\ 0 & \bar{D} \end{pmatrix}. \tag{3.32}$$

3.5.2 Deleting a Superbasic Variable

The method for deleting a superbasic variable depends on whether or not the variable is associated with R_r or D. If the q-th superbasic variable is to be deleted and $q \le n_r$, then the q-th column of R_r is deleted in the same manner a column is deleted to make room for a new superbasic. The submatrix $\widehat{R}_r$ of the modified R_r is defined as above.

To maintain R_r as a $n_r \times n_r$ matrix, a variable currently associated with D must be moved to R_r. The index of a superbasic variable associated with D is chosen by

$$\{t \mid |(Z^Tg)_t| \ge |(Z^Tg)_j|, n_r + 1 \le j \le n_s\}. \tag{3.33}$$

This corresponds to selecting the variable associated with the element of largest magnitude in g_2. The update to S is given by

$$\bar{S} = (\, \bar{S}_1 \quad s_t \quad \bar{S}_2 \,),$$

where s_t was the t-th column of S. The updated $\bar{S}_1$ results from deleting the q-th column of S_1, and the matrix $\bar{S}_2$ is the result of deleting s_t from S_2. Then,

$$\bar{Z}_1 = \begin{pmatrix} -B^{-1}\bar{S}_1 \\ \tilde{I} \\ 0 \end{pmatrix} \qquad \text{and} \qquad w = \begin{pmatrix} -B^{-1}s_t \\ e_{n_r} \\ 0 \end{pmatrix}$$

yield

$$\bar{R}_r = \begin{pmatrix} \widehat{R}_r & v \\ 0 & \rho \end{pmatrix},$$

where v and ρ are calculated as in (3.31). For $D = \text{diag}(d_1, .., d_{n_s - n_r})$, the updated $\bar{D} = \text{diag}(d_1, ..., d_{t-n_r-1}, d_{t-n_r+1}, ..., d_{n_s-n_r})$ and $\bar{R}_C$ is given by (3.32). In this case, the work for deleting a superbasic is the same as for adding a superbasic, since in both cases a new column has to be computed for R_r.

When $q > n_r$, no update to R_r is needed since $\bar{Z}_1^T H \bar{Z}_1 = Z_1^T H Z_1$. Therefore, $\bar{R}_r = R_r$ and $\bar{D} = \text{diag}(d_1, .., d_{q-n_r-1}, d_{q-n_r+1}, .., d_{n_s-n_r})$. Again, $\bar{R}_C$ is given by (3.32).

3.5.3 Deleting a Basic Variable

Recall from Section 2.3.3 that deleting a basic variable corresponds to deleting a column of the basis matrix B. To maintain a nonsingular basis, an appropriate column of S is chosen to swap into B. The BS swap for R_C will depend on whether or not the superbasic to be swapped is associated with R_r or D. In contrast to the case where a superbasic is deleted, if the variable to be swapped is associated with D, there is a change to R_r. This is because the change to B affects both Z_1 and Z_2, and $\bar{Z}_1^T H \bar{Z}_1 \neq Z_1^T H Z_1$.

Assume that the p-th basic variable is being deleted. Let $y = S^T w$, where $B^T w = e_p$. The q-th superbasic variable is selected for the BS swap by (2.17).

The BS swap update to R, the Cholesky factor of the reduced Hessian, is given by (2.18). The update can be stated in terms of y and R as

$$\bar{R} = R - U \qquad \text{where} \qquad U_{i,j} = \begin{cases} ((y)_j/(y)_q)R_{i,q} & i \le q, j \ne q \\ (((y)_q + 1)/(y)_q)R_{i,q} & i \le q, j = q \\ 0 & i > q \end{cases}$$

This update requires knowledge of the nonzero entries of the q-th column of R, indicated by the elements $R_{i,q}$ for $i \le q$. If $q \le n_r$, this column is well defined and the update to R_r is given by

$$\bar{R}_r = R_r - \frac{1}{(y)_q} r_1 (y + e_q)^T, \tag{3.34}$$

where r_1 is the q-th column of R_r. The nonzero entries of r_1 are identical to the nonzero entries of r_q, the q-th column of R. In this case, two sweeps of plane rotations restore $\bar{R}_r$ to triangular form.

If $q > n_r$, however, some nonzero entries of r_q are not explicitly available. In this case, all that is required is to update R_r by

$$\bar{R}_r = R_r - \widetilde{U}, \qquad \text{where} \qquad \widetilde{U}_{i,j} = \begin{cases} ((y)_j/(y)_q) R_{i,q}, & i \le n_r; \\ 0, & i > n_r. \end{cases}$$

Thus, only the first n_r elements of r_q are needed, which can be obtained from

$$R_r^T r_1 = Z_1^T H w \qquad \text{where} \qquad w = \begin{pmatrix} -B^{-1} s_q \\ e_q \\ 0 \end{pmatrix},$$

and s_q is the q-th column of S.

Once r_1 has been obtained, R_r can be updated using (3.34), and $\bar{R}_r$ can be restored to triangular form using two sweeps of plane rotations. Let r_q be partitioned as $r_q = (\, r_1 \quad r_2 \,)^T$, where r_1 is defined as above, and the nonzero entries of the $(n_s - n_r)$-vector r_2 are unknown. Applying the plane rotations to $\bar{R}_r$ requires the two-norm of r_2. Since $r_q^T r_q = r_1^T r_1 + r_2^T r_2$, and $r_q^T r_q$ is the q-th diagonal of the reduced Hessian $Z^T H Z$, the desired norm can be obtained from

$$\|r_2\|^2 = (Z^T H w)_q - \|r_1\|^2.$$

Computing $\|r_2\|$ requires forming $Z^T H w$ (instead of $Z_1^T H w$). This dominates the work of the BS swap, and requires approximately $m^2 + m n_s + 2nN$ flops. As before, N is the maximum number of stored limited-memory quasi-Newton updates before a reset. and when a sparse LU factorization of B is used, there is a corresponding reduction in the work.

The update to Z is given by

$$Z = \begin{pmatrix} -\bar{B}^{-1}\bar{S} \\ I \\ 0 \end{pmatrix}, \tag{3.35}$$

where $\bar{B}$ and $\bar{S}$ are the result of swapping the p-th column of B with the q-th column of S.

The diagonal matrix D is not updated after the BS Swap, and the "new" q-th superbasic variable is deleted as in Section 3.5.2.

3.6 A Quasi-Newton Preconditioner

The second preconditioner is a quasi-Newton approximation of the form

$$R_Q = \begin{pmatrix} T & 0 \\ 0 & D \end{pmatrix},$$

where T is a dense $n_r \times n_r$ upper-triangular matrix and D is a positive diagonal. The matrix T is a quasi-Newton approximation to R_r, and the elements of D approximate the corresponding diagonal elements of a quasi-Newton approximation to the exact R.

As in Section 3.5 for R_C, the preconditioner R_Q will need to be updated for changes in the working set. In addition, a quasi-Newton update for R_Q must be defined.

3.6.1 Updating the Preconditioner

The quasi-Newton update for R_Q is derived from the BFGS formula. To define a BFGS update for R_Q, consider the product form of the BFGS update given by (3.15). Let the vectors u and v be given by

$$u = \frac{1}{\|R_Q q\|}q, \qquad v = \frac{1}{\sqrt{y^T q}}y - \frac{1}{\|R_Q q\|}R_Q^T R_Q q,$$

where $q = \alpha p_Z$ and y is the change in g_Z. From (3.15), the standard BFGS update to R_Q is

$$\bar{R}_Q = R_Q(I + uv^T) = \begin{pmatrix} T & 0 \\ 0 & D \end{pmatrix} (I + uv^T). \tag{3.36}$$

For the BFGS update to R_Q, only elements of T are updated.

Thus, the new T is given by

$$\bar{T} = T(I + u_r v_r^T), \tag{3.37}$$

where u_r and v_r are the first n_r elements of u and v. After the update to T, two sweeps of plane rotations are used to restore $\bar{T}$ to upper-triangular form. The work for updating T is $\mathcal{O}(n_r^2)$ flops.

Let R_q be a full dense upper-triangular quasi-Newton approximation to the exact R. For the appropriate u and v, the update to R_q from (3.15) would be

$$\bar{R}_q = R_q + R_q uv^T.$$

Therefore, the update to the diagonals of R_q given by

$$(\bar{R}_q)_{ii} = (R_q)_{ii} + (R_q u)_i v_i^T$$

depends on the vector $R_q u$. For the approximation R_Q, the (1,2) block is zero, and the (2,2) block is diagonal. Thus, all the information needed for $R_q u$ is not available. From (3.36), the update to the diagonal elements of D can be defined by

$$(\bar{d})_i = (d)_i[1 + (u)_{n_r+i}(v)_{n_r+i}] \qquad \text{for} \quad i = 1, \ldots, (n_s - n_r). \tag{3.38}$$

This method is similar to the one used in MINOS for updating a quasi-Newton diagonal extension. However, these are not the diagonals of the quasi-Newton approximation R_q. For this reason, when R_Q is used as the preconditioner, the algorithm must include safeguards to ensure that the elements of D remain positive.

The efficiency of any quasi-Newton method depends significantly on the initial R_Q. When no other information is available, common practice is to initialize R_Q

to a multiple of the identity. This convention is used for the first QP subproblem. For subsequent subproblems, the final R_Q of the previous subproblem is used. This greatly enhances the potential accuracy of the initial estimate, especially for the latter subproblems, when the working set has settled down.

3.6.2 The Active Set Strategy

The quasi-Newton approximation R_Q must be updated to reflect changes to the working set. This requires updates for the three cases defined in Section 2.3.3; adding a superbasic variable, deleting a superbasic variable and deleting a basic variable. Changes to R_Q for changes in the working set will occur after the quasi-Newton update to R_Q.

Adding a Superbasic Variable

When adding a new superbasic, the new row and column are added to T. Therefore, the quasi-Newton update of (3.37) will be applied to the new column to allow the approximation to incorporate more of the new information. As in Section 3.5.1, one of the superbasics associated with T must be moved to D. The q-th superbasic is chosen by (3.30) and the q-th diagonal of T is moved to D, giving

$$\bar{D} = \begin{pmatrix} D & \\ & t_{qq} \end{pmatrix}.$$

The q-th column of T is deleted and a forward sweep of plane rotations restores T to triangular form. As in Section 3.5.1 for R_r, after the plane rotations are applied, the last row and column of T are zero. Let $\widehat{T}$ be the $(n_r - 1) \times (n_r - 1)$ upper-triangular submatrix of T with the zero row and column omitted. A new column for T may be calculated using equations (3.31), with $\widehat{T}$ in place of $\widehat{R}_r$. This new column will be an approximation to the associated column of R_r. However, to avoid the work of calculating this column, the new column of T is approximated by a column of the identity. The update to T is given by

$$\bar{T} = \begin{pmatrix} \widehat{T} & 0 \\ 0 & 1 \end{pmatrix},$$

and the update to R_Q is

$$\bar{R}_Q = \begin{pmatrix} \bar{T} & \\ & \bar{D} \end{pmatrix}. \tag{3.39}$$

Deleting a Superbasic Variable

As in Section 3.5.2, the method of deleting a superbasic variable depends on if the superbasic is associated with T or D. Suppose that the q-th superbasic is to be deleted. If $q \leq n_r$, then the q-th column of T is deleted in the same manner as a column is deleted for adding a superbasic. In this case, $\widehat{T}$ is defined as above, i.e., a superbasic associated with D is chosen by (3.33), and the updated T is given by

$$\bar{T} = \begin{pmatrix} \widehat{T} & 0 \\ 0 & d_t \end{pmatrix}.$$

The update to D is $\bar{D} = \mathrm{diag}(d_1, ..., d_{t-n_r-1}, d_{t-n_r+1}, ..., d_{n_s-n_r})$ and $\bar{R}_Q$ is given by (3.39).

If $q > n_r$, then $\bar{T} = T$ and $\bar{D} = \mathrm{diag}(d_1, .., d_{q-n_r-1}, d_{q-n_r+1}, .., d_{n_s-n_r})$, and $\bar{R}_Q$ is given by (3.39).

Deleting a Basic Variable

If the p-th basic variable is deleted, the q-th superbasic is chosen to swap into the basis by (2.17). For the quasi-Newton preconditioner, the BS swap update is only performed if $q \leq n_r$. The update to T is given by

$$\bar{T} = T - \frac{1}{(y)_q} t_q (y + e_q)^T,$$

where $(y)_q$ is the q-th element of the vector y such that, $y = S^T w$ with $B^T w = e_p$. The vector t_q is the q-th column of T. The diagonal extension D is not updated for the BS swap, and two sweeps of plane rotations restore $\bar{T}$ to triangular form. If $q > n_r$, neither T nor D are updated for the BS swap. While R_Q may not be updated for the BS swap, Z must always be updated according to (3.35).

Again, once the BS swap is performed, the "new" q-th superbasic variable can be deleted.

3.7 Using Relaxed Tolerances

The efficiency of the QP method can be improved by appropriate relaxation of `RGTOL`, the tolerance on the reduced gradient used to indicate convergence on the subspace defined by the current working-set constraints. During the early iterations of each subproblem many of the multipliers will have a non-optimal sign. The work of minimizing accurately on a subspace can be avoided for these early iterations by removing constraints from the working set before a subspace stationary point is reached. When `RGTOL` is small, constraints are only deleted at subspace stationary points. Thus, the values of (π, λ) represent the Lagrange multipliers at those points. If `RGTOL` is relaxed, then constraints are deleted at non-stationary points. The values of (π, λ) at these points are no longer Lagrange multipliers, but are approximations that converge to the multipliers as the iterates approach a subspace stationary point. Relaxing `RGTOL` exploits the fact that these approximation will often have the correct sign, even though the numerical value is not accurate.

The conjugate-gradient tolerance, `CGTOL`, is used to terminate the iterates of the CG algorithm. If `CGTOL` is small, say $\mathcal{O}(10^{-17})$, then the reduced-Hessian systems are solved accurately, and the QP search directions generated by the CG method are the same as those generated by SQOPT. During the early iterations when the working set is settling down, a relaxed `CGTOL` can be used to decrease the work of getting a precise CG solution. The following theorem shows that termination at any iterate of LSQR will yield a descent direction.

Theorem 5 *Assume that H_Z is positive definite, and that LSQR is applied to (3.4) with A and b defined by (3.16). Then, if the algorithm is terminated after t iterations, the truncated solution d_t is a descent direction, in the sense that $g_Z^T d_t < 0$ for $0 \le t \le n$.*

Proof. The algorithm LSQR is based on the bidiagonalization routine described by *BIDIAG*. The proof makes use of the description of LSQR presented in Section 3.2. The matrices U_{t+1}, V_t and B_t are defined by (3.7).

From (3.11), the current iterate is defined as

$$d_t = V_t y_t, \tag{3.40}$$

where y_t is the solution to the least-squares problem

$$\min \|\beta_1 e_1 - B_t y_t\|.$$

Therefore, y_t is given by

$$y_t = \beta_1 (B_t^T B_t)^{-1} B_t^T e_1. \tag{3.41}$$

Furthermore, from (3.4) and (3.16) for the reduced-Hessian system, it follows that

$$g_Z = -A^T b. \tag{3.42}$$

Substituting (3.40) and (3.42) into $g_Z^T d_t$ yields

$$g_Z^T d_t = -b^T A V_t y_t.$$

By using equations (3.8) and (3.9) this reduces to

$$g_Z^T d_t = -\beta_1 e_1^T B_t y_t = -\beta_1 \alpha_1 (e_1^T y_t).$$

Noting that $B_t^T e_1 = \alpha_1 e_1$, and substituting from (3.41) yields

$$g_Z^T d_t = -\alpha_1^2 \beta_1^2 (e_1^T (B_t^T B_t)^{-1} e_1).$$

By definition, $U_{t+1}^T U_{t+1} = I$, and it follows from (3.9) that

$$B_t^T B_t = B_t^T U_{t+1}^T U_{t+1} B_t = V_t^T A^T A V_t.$$

For the definition of A given in (3.16), $A^T A = H_Z$, thus $B_t^T B_t = V_t^T H_Z V_t$. Since H_Z is positive definite by assumption, and V_t is full rank, it follows that $B_t^T B_t$ is positive definite. Therefore, $e_1^T (B_t^T B_t)^{-1} e_1 > 0$, which yields $g_Z^T d_t < 0$. ∎

This result allows the possibility of using `CGTOL` to terminate LSQR after only a few LSQR iterations (or possibly none). The initial search direction is the steepest-descent direction, $-g_Z$. If the system is solved accurately the search direction is equivalent to the SQOPT search direction (up to rounding errors). Thus, the "coarse" descent directions interpolate between these two directions. These coarse directions are often sufficient while the working set is settling down. However, once the working set is optimal, the reduced-Hessian system must be solved accurately to yield the SQOPT direction. Linking the magnitude of `CGTOL` to the magnitude of the "most" non-optimal multiplier, ensures that the value will be small when the working set is optimal. Let λ_q be the largest magnitude non-optimal multiplier ($\lambda_q = 0$ if all the multipliers have the correct sign) . Each time the multipliers are calculated, `CGTOL` is updated by

$$\mathtt{CGTOL} = \max\{\xi|\lambda_q|, \mathtt{CGTOL}_0\}, \tag{3.43}$$

where ξ is a constant parameter such that $0 < \xi < 1$. The fixed small tolerance $\mathtt{CGTOL}_0$ must be chosen sufficiently small to ensure that when all the multipliers have optimal sign the algorithm will generate the SQOPT direction.

One possible strategy for implementing the relaxed tolerances would be to use one LSQR iteration, take a step and immediately change the working set if some of the multipliers are non-optimal. This procedure would be repeated until the working set is optimal, at which time the tolerances would be tightened and the problem solved accurately. A flaw of this strategy is that deleting constraints without exact Lagrange multipliers can lead to a bound repeatedly moving in and out of the working set. When the exact Lagrange multipliers are available, and the t-th constraint is deleted from the working set, the search direction satisfies the relations

$$g^T p < 0 \quad \text{and} \quad a_t^T p > 0,$$

where a_t^T is the normal of the t-th constraint. However, when the exact multipliers are not available, it is possible for $a_t^T p \leq 0$. This results in the t-th constraint immediately being added back into the working set. Thus, precautions have to be taken to safeguard against cycling.

An alternative strategy is to not delete a constraint until the reduced gradient has been significantly reduced, in the sense that the norm of g_Z should be less than the magnitude of the "most" non-optimal multiplier, λ_q.

Assume that $\|g_Z\|$ is less than the current relaxed `RGTOL`. If some of the multipliers have a non-optimal sign, then a constraint is deleted only if

$$\|g_Z\|_\infty \leq \theta |\lambda_q|, \tag{3.44}$$

where $0 < \theta < 1$. The smaller the value of θ, the greater the reduction required in g_Z. When a constraint is successfully deleted, `RGTOL` is reset for the new subspace to be

$$\mathtt{RGTOL} = \max\{\eta \|\bar{g}_Z\|_\infty, \mathtt{RGTOL}_0\},$$

where η is a constant parameter such that $0 < \eta < 1$, $\bar{g}_Z = (\ g_Z \quad \lambda_q\ )^T$, and $\mathtt{RGTOL}_0$ is a fixed small tolerance. The smaller the value of η the more accurate the optimization on each subspace. If all the multipliers have their optimal sign, a test is made to see if a solution has been found by checking $\|g_Z\| \leq \mathtt{RGTOL}_0$. If the current point is not a solution, or if (3.44) is not satisfied, then `RGTOL` is updated by

$$\mathtt{RGTOL} = \max\{0.9\|g_Z\|_\infty, \mathtt{RGTOL}_0\}, \tag{3.45}$$

and optimization continues on the current subspace.

While this strategy is admittedly heuristic, a similar approach has been implemented in MINOS with practical success.

3.8 Algorithm SQOPT-CG

With the modifications necessary for calculating the search direction and the preconditioner, the algorithm SQOPT-CG can be defined from algorithm SQOPT (see Section 2.3.4). The algorithm is the same as SQOPT except in the definition of the search direction and the choice of tolerances. If $n_s \le n_r$ the standard method of SQOPT is used to calculate the search directions. If $n_s > n_r$ the least-squares CG algorithm LSQR is used. If a small `CGTOL` is used in LSQR, the search directions of the two algorithms are identical except for rounding error.

The efficiency of Algorithm SQOPT-CG on a given problem is dependent on three modifiable settings.

1. *The choice of preconditioner.* The work required to calculate and update the preconditioner is significantly greater for the partial reduced-Hessian preconditioner, R_C, than for the quasi-Newton preconditioner R_Q. However, the eigenvalue clustering of the preconditioned system should be better with R_C than with R_Q. As mentioned, the degree of clustering has a significant impact on the number of CG iterations needed to solve the system. The work needed to maintain a particular preconditioner has to be balanced against the savings in CG iterations.

   Obviously, it would be advantageous to avoid the work associated with calculating and updating the preconditioner. For well-conditioned problems, preconditioning is unnecessary and the use of a preconditioner can be avoided.

2. *The choice of n_r.* The value of n_r represents the size of the dense upper-triangular block of the preconditioners (R_r or T). The larger the value of n_r, the more beneficial the effect of R_r and T on the preconditioned system.

   Since the direct method is used for $n_s \le n_r$, it would be preferable to use $n_r \approx 1000$. However, for problems with $n_s \gg 1000$, the work associated with using a matrix R_r of this size is prohibitive. The work for calculating the initial R_r is of $\mathcal{O}(n_r^3)$, and each solve with R_r requires $\mathcal{O}(n_r^2)$ operations. A more reasonable value of n_r, say $200 \leq n_r \leq 500$, results in approximately 8-125 times less work in calculating the initial R_r, and about 4-25 times less work in performing the solves with R_r.

For the quasi-Newton preconditioner, the work for performing the solves with T remains $\mathcal{O}(n_r^2)$. However, the work for calculating the initial R_r is avoided. This means that larger values of n_r can exploit the full efficiency of the direct method.

3. *The values of η, θ and ξ.* Section 3.7 contains a discussion on the benefits of relaxing `RGTOL` and `CGTOL`. When `RGTOL` is relaxed, not minimizing on the subspace saves minor iterations. However, for every constraint deleted in error, the search direction must be recomputed, requiring an additional call to the CG algorithm. The more accurate the optimization on each subspace, the less likely this is to occur. As mentioned is Section 3.7, the accuracy of the optimization on each subspace is controlled by the value of η. A smaller value of η will require more minor iterations, but will result in fewer erroneous modifications to the working set. In addition, constraints are only deleted if (3.44) is satisfied. The smaller the value of θ, the greater the required reduction in the reduced gradient before a constraint can be deleted. A similar scheme implemented in MINOS uses $\theta = 0.9$ and $\eta = 0.2$.

   A relaxed `CGTOL` gives a direction that interpolates between the steepest-descent direction and the direction generated by SQOPT. The larger the value of `CGTOL`, the fewer CG iterations required, and the closer the resulting direction is to the steepest-descent direction. However, the steepest-descent method is notoriously inefficient and a larger than necessary value for `CGTOL` can yield search directions that are inefficient. The smaller the value of $|\lambda_q|$, the more accurately the system is solved. A typical value for ξ is $\xi = 0.9$.

The algorithm SQOPT-CG is now given as follows. The value of θ, η and ξ are given, and the initial `RGTOL` and `CGTOL` are set prior to entering the QP subproblem.

Algorithm SQOPT-CG

Find an initial feasible point, (x, s).

Set n_r and choose preconditioner R_C or R_Q

Set *QN-precon* = **true** or **false**

if $n_s > n_r$ **then**

  Initialize the preconditioner:

  $R_Q = R_Q$ of previous subproblem, or $T = I$ and $D = I$

  OR, calculate the initial $R_r^T R_r = Z_r^T H Z_r$ and set $D = I$

else

  Calculate the initial $R^T R = Z^T HZ$

end if

Calculate the initial g and ψ

while true

  Calculate g_z and π

  stationary-point = $\|g_z\| <$ `RGTOL`

  new-SB = **false**

  if *stationary-point* **then**

    Calculate multipliers, λ

    Set λ_q to largest magnitude non-optimal multiplier

    `CGTOL` $= \max\{\xi|\lambda_q|, \mathtt{CGTOL}_0\}$

    if $\lambda_q = 0$ **then**

      if $\|g_Z\| < \mathtt{RGTOL}_0$ **then stop**

      `RGTOL` $= \max\{0.9\|g_Z\|_\infty, \mathtt{RGTOL}_0\}$

    else

      if $\|g_Z\|_\infty \le \theta|\lambda_q|$ **then**

        Add superbasic corresponding to λ_q

        $\bar{g}_Z = (\ g_Z \quad \lambda_q\ )^T$

        new-SB = **true**

        `RGTOL` $= \max\{\eta\|\bar{g}_Z\|_\infty, \mathtt{RGTOL}_0\}$

      else

        `RGTOL` $= \max\{0.9\|g_Z\|_\infty, \mathtt{RGTOL}_0\}$

      end if

    end if

  end if

if *new-SB* **then**
  if $n_s > n_r$ **then**
    Update the preconditioner
  else
    Update R
  end if
  Update g_z
end if
if $n_s > n_r$ **then**
  Solve LS problem (3.23) for p_z
else
  Solve for p_z as in SQOPT
end if
Set $p = Zp_z$
α_N = maximum feasible step along p
hit-constraint $= \alpha_N < 1$
if *hit-constraint* **then**
  $\alpha = \alpha_N$
else
  $\alpha = 1$
end if
$(x, s) \leftarrow (x, s) + \alpha p$
$g \leftarrow g + \alpha Hp$
Calculate ψ
if $n_s > n_r$ **and** *QN-precon* **then**
  Perform BFGS update to R_Q
end if
if *hit-constraint* **then**
  if $n_s > n_r$ **then**
    Update the preconditioner
  else
    Update R
  end if
end if
end while

Typical values for $\mathtt{CGTOL}_0$ and $\mathtt{RGTOL}_0$ are 10^{-17} and 10^{-6} respectively. The same storage space is used for R and the preconditioner (R_C or R_Q).

3.9 Numerical Results

This section gives numerical results for the CG method on a subset of problems from the CUTE test suite [4]. The CUTE test suite is a diverse set of test problems of varying sizes. The CUTE package provides an interface to run the test problems using preset starting points. The problems chosen have a large number of superbasics, with most problems having a maximum value of n_s that exceeds 1000. Over half of the problems have a value of n_s of approximately 3000 or more. The largest value of n_s was 5625. Two problems had a maximum value of $n_s \approx 600$.

The test set consists of 74 problems. Of these, 35 are unconstrained problems (UC), 22 are problems with simple bounds only (BO), 10 are nonlinearly constrained problems (NC) and 7 are linearly constrained problems (LC). Table 3.1 lists the problems in the test set, along with the number of variables, n, the maximum value of n_s attained during the runs, and the problem type.

For all runs, the CG tolerance was set to $\mathtt{CGTOL} = 10^{-17}$. The value of reduced-gradient tolerance was fixed at $\mathtt{RGTOL} = 10^{-6}$. The relaxed tolerance strategy was not used for these runs, therefore, the results compare the methods when generating numerically equivalent search directions. For the SNOPT-CG runs, the value of n_r was set to 500, and the partial reduced-Hessian preconditioner, R_C, was used.

The algorithm was implemented in FORTRAN 77 with SNOPT. The runs were made on an SGI R10000 (180MhZ). The performance varied by problem type, thus the results are presented by problem type, with a general summary at the end.

Table 3.1: CUTE test problems with n_s large

Problem Name	n	n_s	Type	Problem Name	n	n_s	Type
ARWHEAD	5000	5000	UC	NONDQUAR	5000	5000	UC
BDEXP	5000	5000	BO	NONSCOMP	5000	5000	BO
BLOWEYA	2002	1000	LC	OBSTCLAE	5625	5329	BO
BLOWEYB	2002	1000	LC	OBSTCLAL	5625	2917	BO
BLOWEYC	2002	1000	LC	OBSTCLBL	5625	4120	BO
BRYBND	5000	5000	UC	OBSTCLBM	5625	4916	BO
CRAGGLVY	5000	5000	UC	OBSTCLBU	5625	4042	BO
DIXMAANA	3000	3000	UC	ORTHRDM2	4003	2001	NC
DIXMAANB	3000	3000	UC	ORTHREGC	4005	2005	NC
DIXMAANC	3000	3000	UC	ORTHREGD	4003	2002	NC
DIXMAAND	3000	3000	UC	ORTHREGE	3006	1006	NC
DIXMAANE	3000	3000	UC	ORTHREGF	1880	1255	NC
DIXMAANF	3000	3000	UC	ORTHRGDM	4003	2003	NC
DIXMAANG	3000	3000	UC	POWELLSG	5000	5000	UC
DIXMAANH	3000	3000	UC	QUARTC	5000	4999	UC
DIXMAANI	3000	3000	UC	READING9	2002	680	NC
DIXMAANJ	3000	3000	UC	SCHMVETT	5000	5000	UC
DIXMAANK	3000	3000	UC	SINROSNB	2000	2000	NC
DIXMAANL	3000	3000	UC	SROSENBR	5000	5000	UC
DQDRTIC	5000	5000	UC	STCQP1	4097	2923	LC
DQRTIC	5000	4999	UC	STCQP2	4097	2023	LC
DTOC6	10001	5000	NC	STNQP1	4097	2560	LC
EDENSCH	2000	2000	UC	TESTQUAD	1000	1000	UC
ENGVAL1	5000	5000	UC	TOINTGSS	5000	4998	UC
FMINSRF2	5625	5625	UC	TORSION1	5476	3696	BO
FMINSURF	5625	5625	UC	TORSION2	5476	5184	BO
FREUROTH	5000	5000	UC	TORSION3	5476	1912	BO
JNLBRNG2	5625	3198	BO	TORSION4	5476	5184	BO
JNLBRNGA	5625	3529	BO	TORSION6	5476	5184	BO
JNLBRNGB	5625	2989	BO	TORSIONA	5476	3664	BO
LCH	600	599	NC	TORSIONB	5476	5184	BO
LIARWHD	5000	5000	UC	TORSIONC	5476	1952	BO
LINVERSE	1999	1988	BO	TORSIOND	5476	5184	BO
MOSARQP1	2500	1073	LC	TORSIONF	5476	5184	BO
NCB20	2010	2009	UC	TQUARTIC	5000	5000	UC
NCB20B	2000	2000	UC	TRIDIA	5000	5000	UC
NOBNDTOR	5476	4366	BO	VAREIGVL	5000	5000	UC

3.9.1 Unconstrained Problems

Table 3.2 gives the run times on the unconstrained problems for SNOPT 5.3 and SNOPT-CG, with and without preconditioning. The times are in seconds. The values of n and n_s are also included.

The performance of SNOPT-CG on the unconstrained problems is significantly better than SNOPT 5.3. Most of the problems were solved 8-16 times faster using SNOPT-CG with the preconditioner. There were three problems, NCB20, NCB20B and TESTQUAD, where the preconditioned SNOPT-CG runs did not show an improvement, or the performance was worse. In contrast, the SNOPT-CG runs without preconditioning showed improvement for all of the unconstrained problems. The run times for most problems improved by 20-40 times.

Table 3.3, lists the number of major (Maj) iterations, minor (Min) iterations and LSQR iterations for each of the runs. The table shows the number of major and minor iterations for the preconditioned and non-preconditioned runs is the same for most problems. In general, the number of minor iterations for SNOPT-CG was greater than for SNOPT 5.3. The effect of the preconditioner can be examined by comparing the number of LSQR iterations. For most problems, the number of LSQR iterations were approximately the same for both the preconditioned and non-preconditioned runs. In this case, the significant difference in run times reflects the extra work of applying the preconditioner.

The cumulative summary of the results on the unconstrained problems is presented in Table 3.4. The run times are, on average, about 8 times faster using SNOPT-CG with preconditioning, and 20 times faster without preconditioning. Note that the average number of LSQR iterations per minor iterations is very low when compared to the maximum size of n_s for the test problems.

3.9.2 Nonlinearly Constrained Problems

The run times in Table 3.5 indicate the potential of SNOPT-CG on nonlinearly constrained problems. In general, performance was 3-5 times faster, and

Table 3.2: Unconstrained problems - SNOPT-CG vs. SNOPT 5.3 Run Times

			SNOPT-CG		
Problem Name	n	n_s	w/o precnd	with precnd	SNOPT 5.3
ARWHEAD	5000	5000	58.47	153.54	2727.43
BRYBND	5000	5000	79.11	195.55	3430.44
CRAGGLVY	5000	5000	121.61	319.17	4601.94
DIXMAANA	3000	3000	20.79	68.59	615.44
DIXMAANB	3000	3000	21.46	75.34	711.44
DIXMAANC	3000	3000	22.03	71.94	660.99
DIXMAAND	3000	3000	22.88	73.86	669.82
DIXMAANE	3000	3000	66.04	166.31	1435.93
DIXMAANF	3000	3000	52.17	137.78	1294.44
DIXMAANG	3000	3000	49.77	132.65	1248.14
DIXMAANH	3000	3000	40.39	117.49	1104.58
DIXMAANI	3000	3000	649.52[1]	1402.89[1]	5201.64[1]
DIXMAANJ	3000	3000	133.95	322.74	2415.27
DIXMAANK	3000	3000	79.12	191.23	1729.05
DIXMAANL	3000	3000	101.22	244.77	1938.36
DQDRTIC	5000	5000	69.98	180.63	2803.18
DQRTIC	5000	4999	76.95[2]	208.10[2]	2779.21[2]
EDENSCH	2000	2000	17.30	64.84	254.28
ENGVAL1	5000	5000	65.35	166.76	3081.08
FMINSRF2	5625	5625	1266.28	3343.23	15486.19
FMINSURF	5625	5625	671.99	1147.27	23558.69
FREUROTH	5000	5000	180.03	439.58	4699.69
LIARWHD	5000	5000	73.58	185.78	3314.88
NCB20	2010	2009	506.49	1776.28	1628.29
NCB20B	2000	2000	453.28[1]	1844.97[1]	1872.85[1]
NONDQUAR	5000	5000	377.39	653.42	11944.72
POWELLSG	5000	5000	73.87	190.13	3671.06
QUARTC	5000	4999	77.87[2]	223.43[2]	2780.66[2]
SCHMVETT	5000	5000	87.30	218.26	3567.18
SROSENBR	5000	5000	62.56	161.97	2914.90
TESTQUAD	1000	1000	243.08	4450.28	296.63
TOINTGSS	5000	4998	57.14	152.86	2631.76
TQUARTIC	5000	5000	97.37	362.84	3476.58
TRIDIA	5000	5000	1155.25	3920.86	19213.10
VAREIGVL	5000	5000	96.85	251.36	4642.57

[1] major iteration limit exceeded. [2] problem is unbounded.

Table 3.3: Number of iterations for the unconstrained problems

	SNOPT-CG						SNOPT 5.3	
	w/o precnd			with precnd				
Problem Name	Min	Maj	LSQR	Min	Maj	LSQR	Min	Maj
ARWHEAD	5012	9	8696	5014	9	8711	5014	9
BRYBND	5186	40	9965	5169	40	10264	5062	40
CRAGGLVY	7878	99	13523	7881	99	15364	7637	99
DIXMAANA	3015	14	4736	3015	14	4768	3016	14
DIXMAANB	3019	18	4791	3020	18	4840	3021	18
DIXMAANC	3050	23	4839	3035	23	4890	3026	23
DIXMAAND	3127	25	4968	3100	25	5028	3030	25
DIXMAANE	3190	190	8331	3191	190	8643	3191	190
DIXMAANF	3161	160	6628	3161	160	6967	3161	160
DIXMAANG	3180	149	6337	3163	149	6648	3151	149
DIXMAANH	3530	119	5377	3407	119	5872	3122	119
DIXMAANI	4000	1000	91941	4001	1000	86833	4001	1000
DIXMAANJ	3402	401	17357	3402	401	17542	3404	401
DIXMAANK	3277	253	9769	3261	253	9778	3254	253
DIXMAANL	3737	296	13239	3596	296	13093	3304	296
DQDRTIC	6265	13	9887	6147	13	9867	5022	13
DQRTIC	9000	0	8037	9000	0	8430	5623	0
EDENSCH	3448	52	4624	3111	52	4951	2072	52
ENGVAL1	5157	29	9138	5139	29	9133	5033	29
FMINSRF2	6053	428	75107	6053	428	126797	6053	428
FMINSURF	6697	1072	43548	6529	904	39819	6449	820
FREUROTH	5781	106	17679	5737	106	19273	5160	106
LIARWHD	5959	39	9695	5853	39	9840	5074	39
NCB20	2806	785	131571	2826	806	131702	2830	809
NCB20B	3006	1000	111441	3006	1000	135360	3006	1000
NONDQUAR	5525	490	31457	5531	491	28051	5597	523
POWELLSG	5271	60	9200	5234	61	9759	5081	61
QUARTC	9000	0	8037	9000	0	8430	5623	0
SCHMVETT	5064	56	10642	5062	56	11306	5062	56
SROSENBR	5158	19	8774	5139	19	8846	5029	19
TESTQUAD	2704	797	147833	2739	912	476261	2400	883
TOINTGSS	5002	4	8676	5002	4	8671	5003	4
TQUARTIC	5024	15	13602	5023	15	18159	5025	15
TRIDIA	10495	905	90768	10386	897	164201	6708	899
VAREIGVL	6408	117	7705	6200	117	9378	5118	117

Table 3.4: Summary of unconstrained problems

Summary	SNOPT-CG w/o precnd	SNOPT-CG with precnd	SNOPT 5.3
No. problems attempted :	35	35	35
No. optimal :	31	31	31
No. iterations limit :	2	2	2
No. cannot be improved :	0	0	0
No. Unbounded :	2	2	2
No. Major iterations :	8783	8745	8669
No. Minor iterations :	171587	170133	153362
Avg. LSQR itrs / minor :	5.6	8.5	
No. Function evals. :	9669	9625	9515
Total Time :	7228.44	23616.70	144402.41

in the case of DTOC6 and SINROSNB, over 50 times faster. The improvement is most significant for problems where many iterations are required with a large number of degrees of freedom. The combined effect of not computing the full R, and only computing new columns of dimension n_r (vs. n_s) is magnified for the nonlinear problems. This benefit is even reflected in the smaller problems LCH and READING9, where n_s is only 600, but the run continues for 1000 iterations.

The preconditioner has the greatest success in reducing the number of LSQR iterations on the nonlinearly constrained problems. In Table 3.6, most problems show a reduction in the number of LSQR iterations with the preconditioner. The average reduction for these problems was 23%. In addition, without preconditioning the problem SINROSNB terminates at a search direction that is not a descent direction. However, the execution times for the runs without preconditioning are still better than those with preconditioning. Thus, the benefit of the preconditioner is not sufficient to overcome its inherent additional cost.

The number of minor iterations for SNOPT-CG is higher than for SNOPT 5.3. However, the use of the preconditioner required fewer minor iterations than the runs without preconditioning.

Table 3.7 summarizes the totals for the nonlinearly constrained problems. The results for SINROSNB are not included. The numbers are dominated by the results for DTOC6. The average number of LSQR iterations per minor iteration is again very small.

Table 3.5: Nonlinearly constrained problems - SNOPT-CG vs. SNOPT 5.3 Run Times

			SNOPT-CG		
Problem Name	n	n_s	w/o precnd	with precnd	SNOPT 5.3
DTOC6	10001	5000	1567.54	3196.50	169057.88
LCH	600	599	58.85[1]	1825.92[1]	2265.99[1]
ORTHRDM2	4003	2001	252.27	396.56	2170.32
ORTHREGC	4005	2005	419.30	607.27	9477.25
ORTHREGD	4003	2002	259.34	412.69	2415.71
ORTHREGE	3006	1006	370.18	2288.41	4576.41
ORTHREGF	1880	1255	83.84	256.59	1328.20
ORTHRGDM	4003	2003	257.04	422.00	2902.12
READING9	2002	680	55.81[1]	1475.72[1]	4040.47[1]
SINROSNB	2000	2000	1109.62[2]	5134.35[1]	235767.25[1]

[1] major iteration limit exceeded. [2] final point cannot be improved on.

3.9.3 Linearly Constrained Problems

In contrast to the unconstrained and nonlinearly constrained problems, where SNOPT-CG exhibits a significant improvement, the performance on linearly constrained problems is generally poorer than for SNOPT 5.3. Table 3.8 gives the run times for SNOPT-CG on the linearly constrained problems. As mentioned in Section 2.2, SNOPT 5.3 is more efficient on problems where all the constraints are linear. This is a result of R not needing to be refactorized after each major iteration. The performance of SNOPT-CG on the problems where $n_s \approx 1000$ is worse than SNOPT 5.3. However, on problems where $n_s > 2000$, SNOPT-CG starts to show modest improvement, with the best results being for the problem with the largest maximum value of n_s.

Table 3.6: Number of iterations for the nonlinearly constrained problems

	SNOPT-CG						SNOPT 5.3	
	w/o precnd			with precnd				
Problem Name	Min	Maj	LSQR	Min	Maj	LSQR	Min	Maj
DTOC6	14510	22	180379	14278	22	170307	10030	22
LCH	1761	1000	40896	1616	1000	76013	1605	1000
ORTHRDM2	5499	8	39238	4208	8	24773	4018	8
ORTHREGC	5319	94	54480	4064	33	29055	4081	38
ORTHREGD	5699	9	40065	4508	9	25301	4204	9
ORTHREGE	3860	280	41487	3514	333	55388	3223	204
ORTHREGF	2686	32	26556	1915	32	18366	1914	32
ORTHRGDM	5521	11	39576	4398	11	25284	4117	11
READING9	2679	1000	3540	2679	1000	4536	2679	1000
SINROSNB	21391	13	83571	36028	1000	148420	65599	1000

Table 3.7: Summary of nonlinearly constrained problems

	SNOPT-CG		SNOPT 5.3
Summary	w/o precnd	with precnd	
No. problems attempted :	10	10	10
No. optimal :	7	7	7
No. iterations limit :	2	3	3
No. cannot be improved :	1	0	0
No. Unbounded :	0	0	0
No. Major iterations :	2456	2448	2324
No. Minor iterations :	47534	41180	35871
Avg. LSQR itrs / minor :	9.8	10.4	
No. Function evals. :	4146	3631	3775
Total Time :	3322.17	10881.66	198234.35

Table 3.9 shows the number of major and minor iterations is essentially the same for all the runs. In general, the number of LSQR iterations is significantly higher for the preconditioned runs. A notable exception was MOSARQP1, where the preconditioner reduced the number of LSQR iterations by almost 80%. However, despite the significant reduction in the LSQR iterations, there was only a modest improvement in the run time.

The cumulative totals for the linearly constrained problems are given in Table 3.10. The average number of LSQR iterations per minor iteration is larger than for the unconstrained and nonlinearly constrained problems. However, the values are still small relative to the size of n_s.

Table 3.8: Linearly constrained problems - SNOPT-CG vs. SNOPT 5.3 Run Times

			SNOPT-CG		
Problem Name	n	n_s	w/o precnd	with precnd	SNOPT 5.3
BLOWEYA	2002	1000	701.10	2996.99	399.83
BLOWEYB	2002	1000	485.96	2161.20	336.64
BLOWEYC	2002	1000	817.72	2469.84	294.72
MOSARQP1	2500	1073	273.42	248.66	151.44
STCQP1	4097	2923	111.21	500.29	1028.94
STCQP2	4097	2023	95.20	580.14	390.74
STNQP1	4097	2560	59.41	212.58	483.65

Table 3.9: Number of iterations for the linearly constrained problems

	SNOPT-CG						SNOPT 5.3	
	w/o precnd			with precnd				
Problem Name	Min	Maj	LSQR	Min	Maj	LSQR	Min	Maj
BLOWEYA	1536	390	170842	1565	436	241274	1483	390
BLOWEYB	1453	310	109735	1372	310	175303	1329	310
BLOWEYC	1410	388	196966	1325	311	185921	1325	311
MOSARQP1	4199	13	123874	4155	13	25444	3823	13
STCQP1	6963	14	11221	6982	14	28520	6963	14
STCQP2	5856	60	9277	5900	60	28998	5856	60
STNQP1	6270	3	4123	6270	3	10161	6270	3

3.9.4 Problems with Only Simple Bounds

The results for problems with only simple bounds are mixed. In general, performance depends on the number of active bounds. Times on problems where $n - n_s$ stays small (say ≤ 1200) more closely resemble those for an unconstrained problem. In contrast, when $n - n_s$ is large, which implies that many bounds are active, the run-times resemble those for the simple linearly constrained problems, and the performance on these problems is more modest.

A notable exception to this trend are the TORSION problems. When preconditioning was used, the performance on these problems was generally worse than that of SNOPT 5.3. Without preconditioning, SNOPT-CG showed only a modest improvement

Comparing the LSQR iterations in Table 3.12 shows that the preconditioner performed very poorly on problems with only simple bounds.

The cumulative totals for the problems with only simple bounds are given in Table 3.13. The average number of LSQR iterations per minor iteration is similar to the numbers for the linearly constrained problems.

3.9.5 Summary

The overall performance of the CG method is typical of that of other iterative methods, in the sense that performance is extremely problem dependent. In general, the method performed well on unconstrained problems, nonlinearly constrained problems, and problems with simple bounds with relatively few active bounds (i.e., $n_s \approx n$).

The preconditioner performed best on problems with nonlinear constraints. However, the improvement in iterations tended to be overwhelmed by the cost of applying the preconditioner. Performance was best for SNOPT-CG without preconditioning. This may be caused by the low average number of LSQR iterations per minor iteration. The overall average for the runs without preconditioning was 14.7 LSQR iterations per minor iteration. This value is small relative to the typical size of n_s, thus it may have been difficult for the preconditioner to significantly reduce the number of LSQR iterations. However, there is evidence that preconditioning is necessary for some problems. The quasi-Newton preconditioner avoids much of the work associated with preconditioning and will likely provide a better alternative to the partial reduced-Hessian preconditioner.

Table 3.10: Summary of linearly constrained problems

	SNOPT-CG		SNOPT 5.3
Summary	w/o precnd	with precnd	
No. problems attempted :	7	7	7
No. optimal :	7	7	7
No. iterations limit :	0	0	0
No. cannot be improved :	0	0	0
No. Unbounded :	0	0	0
No. Major iterations :	1178	1147	1101
No. Minor iterations :	27687	27569	27049
Avg. LSQR itrs / minor :	22.6	25.2	
No. Function evals. :	1339	1308	1254
Total Time :	2544.02	9169.70	3085.96

Table 3.11: Problems with only simple bounds - SNOPT-CG vs. SNOPT 5.3 Run Times

			SNOPT-CG		
Problem Name	n	n_s	w/o precnd	with precnd	SNOPT 5.3
BDEXP	5000	5000	58.82	154.51	2909.09
JNLBRNG2	5625	3198	936.56	1803.77	3974.54
JNLBRNGA	5625	3529	2247.08	16877.73	4758.04
JNLBRNGB	5625	2989	3994.34	13889.41	5754.79
LINVERSE	1999	1988	2426.85	9524.32	4555.55
NOBNDTOR	5476	4366	591.31	1440.70	5526.85
NONSCOMP	5000	5000	1000.98	1794.35	10944.32
OBSTCLAE	5625	5329	2191.45	7382.87	16112.66
OBSTCLAL	5625	2917	580.42	1819.50	1725.03
OBSTCLBL	5625	4120	3743.52	12765.81	11381.93
OBSTCLBM	5625	4916	1484.87	5168.41	15895.72
OBSTCLBU	5625	4042	4846.31	15187.39	8381.12
TORSION1	5476	3696	1075.65	3370.38	4709.72
TORSION2	5476	5184	15836.76	48779.39	36296.82
TORSION3	5476	1912	350.27	1000.93	450.09
TORSION4	5476	5184	13635.31	48756.90	21950.53
TORSION6	5476	5184	10543.89	34678.18	17692.28
TORSIONA	5476	3664	1102.13	3527.48	5289.43
TORSIONB	5476	5184	13313.99	40639.84	43210.66
TORSIONC	5476	1952	370.10	1089.07	534.54
TORSIOND	5476	5184	12783.09	39232.64	25700.64
TORSIONF	5476	5184	9106.59	23918.17	16373.41

Table 3.12: Number of iterations for the problems with only simple bounds

	SNOPT-CG						SNOPT 5.3	
	w/o precnd			with precnd				
Problem Name	Min	Maj	LSQR	Min	Maj	LSQR	Min	Maj
BDEXP	5015	15	8674	5015	15	8699	5015	15
JNLBRNG2	4764	437	63808	4762	433	60427	4752	426
JNLBRNGA	5929	264	153284	5928	264	561229	5928	264
JNLBRNGB	9917	633	321862	9816	594	521080	9159	548
LINVERSE	20968	198	655965	20968	198	711705	20968	198
NOBNDTOR	5426	184	39881	5424	184	58556	5424	184
NONSCOMP	17534	44	61336	17361	44	63235	16798	44
OBSTCLAE	11805	168	130137	11806	168	263701	11805	168
OBSTCLAL	3637	112	40608	3637	112	65944	3637	112
OBSTCLBL	22178	92	301590	22178	92	513859	22178	92
OBSTCLBM	11969	85	99634	11969	85	198653	11969	85
OBSTCLBU	22970	97	370134	22970	97	583751	22970	97
TORSION1	5895	174	76193	5890	174	139079	5893	174
TORSION2	46104	263	1147404	46017	220	1855646	45373	228
TORSION3	2584	74	26105	2586	74	43170	2586	74
TORSION4	44718	122	988519	44733	122	2050675	44706	122
TORSION6	42754	82	633782	42761	82	1727239	42792	81
TORSIONA	6001	181	73936	6000	181	137304	6005	181
TORSIONB	41553	190	922034	41558	191	1573115	41510	191
TORSIONC	2859	76	28075	2860	76	46474	2858	76
TORSIOND	48014	127	989224	48026	127	1778269	42379	65
TORSIONF	42399	65	706682	42397	65	1228068	47974	127

Table 3.13: Summary of problems with only simple bounds

Summary	SNOPT-CG w/o precnd	SNOPT-CG with precnd	SNOPT 5.3
No. problems attempted :	22	22	22
No. optimal :	22	22	22
No. iterations limit :	0	0	0
No. cannot be improved :	0	0	0
No. Unbounded :	0	0	0
No. Major iterations :	3683	3598	3552
No. Minor iterations :	424993	424662	422679
Avg. LSQR itrs / minor :	18.4	33.4	
No. Function evals. :	4243	4141	4105
Total Time :	102220.49	332801.75	264127.76

Chapter 4

The Schur-Complement Method

The Schur-complement QP algorithm of Gill *et al.* [20], maintains a fixed factorization of the initial KKT system, and updates a smaller dense factorization of a Schur complement. These factorizations are used to compute the search directions and multipliers for the quadratic program. The method proposed in this chapter uses a Schur complement-based algorithm to solve the QP subproblems.

The method uses an extended KKT system associated with a specially defined working set (see below). This modification improves the efficiency of existing methods by defining simpler linear systems at each iteration. In addition, the efficiency is improved by limiting the number of updates to the dense Schur complement.

In the final section of this chapter, a new Schur-complement method is derived based on minimizing an ℓ_2 composite objective function. This single phase algorithm does not require the two-phase approach of SQOPT for finding a feasible point.

Sections 4.1 and 4.2 review the theory and development of a general algorithm for convex quadratic programming used for the Schur-complement method. The proposed Schur-complement method is presented in Sections 4.3–4.4. Finally, Section 4.5 discusses the use of a composite objective function, with an ℓ_2 penalty term, when the initial point is not feasible.

4.1 Convex Quadratic Programming

Consider a QP of the form:

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & f(x) \equiv c^T x + \frac{1}{2} x^T H x \\ \text{subject to} & Ax \geq b. \end{array}$$

Assume that H is positive semidefinite and x_0 is feasible. Define the constraint residual $r(x) = Ax - b$ and objective gradient $g(x) = c + Hx$.

Active-set quadratic programming (QP) methods use a *working set* of the constraint gradients to define the search direction and multiplier estimates. In the methods proposed here, the working set is chosen to control the inertia of the reduced Hessian, which is never permitted to have more than one zero eigenvalue. Such methods are called *inertia-controlling*.

Given a matrix W of full row rank, a point x such that $g(x) = W^T \lambda$ for some vector λ is known as a *subspace stationary point* (with respect to W). Given a null-space basis Z for W, a subspace stationary point such that $Z^T H Z$ is positive definite is known as a *subspace minimizer* (with respect to W). A feasible point is *optimal* if the vector λ is nonnegative at a subspace minimizer.

At a point x_k, the matrix W will be defined as a linearly independent subset W_k of the rows of A determined by an index set $\mathcal{W}_k$ known as a *working set*. Assume for the moment that x_k and $\mathcal{W}_k$ are defined such that every constraint in $\mathcal{W}_k$ is active, i.e., $\mathcal{W}_k = \{i \mid a_i^T x_k = \beta_k\}$. Moreover, assume that x_k is a non-optimal subspace minimizer with respect to W_k, in which case there exists an index t in $\mathcal{W}_k$ such that

$$g_k = W_k^T \lambda_k, \qquad \text{and} \qquad (\lambda_k)_t < 0. \tag{4.1}$$

At this point it is useful to partition W_k so that the constraint $a_t^T x \geq \beta_t$ is separate from the other constraints in the working set. In particular, it is assumed for notational purposes that the working-set matrix can be arranged in the form

$$W_k = \begin{pmatrix} A_k \\ a_t^T \end{pmatrix}, \tag{4.2}$$

where A_k is the matrix with rows $\{a_i^T\}$ for $i \in \mathcal{W}_k$, $i \neq t$.

As $(\lambda_k)_t$ is negative, Farkas' Lemma implies that there must exist a direction p that reduces f while satisfying the feasibility conditions

$$A_k p = 0 \qquad \text{and} \qquad a_t^T p > 0.$$

These conditions ensure that the constraints with gradients in A_k will remain active at any step of the form $x_k + \alpha p_k$. (This motivates the use of the notation "A" for "active" working-set constraints.) An obvious choice for p is the vector that both satisfies these conditions and minimizes $f(x_k + p)$. This vector, denoted by p_k, is defined by the equality-constraint subproblem

$$\underset{p \in \mathbb{R}^n}{\text{minimize}} \;\; g_k^T p + \tfrac{1}{2} p^T H p, \qquad \text{subject to} \;\; A_k p = 0,$$

and satisfies the system

$$\begin{pmatrix} H & A_k^T \\ A_k & 0 \end{pmatrix} \begin{pmatrix} p_k \\ -\pi_k \end{pmatrix} = \begin{pmatrix} -g_k \\ 0 \end{pmatrix}, \tag{4.3}$$

where the vector π_k is the vector of multipliers at the subspace stationary point $x_k + p_k$ (which may or may not be feasible with respect to constraints not in the working set). Once p_k is defined, the step α_k is required to retain feasibility with respect to constraints not in the working set. If α_B denotes the step to the blocking constraint, the step α_k is given by $\alpha_k = \min\{1, \alpha_B\}$. If $\alpha_B \geq 1$, then $x_k + p_k$ is feasible, and $x_k + p_k$ is a subspace stationary point with respect to A_k. Otherwise, $\alpha_k = \alpha_B < 1$ and the working set is modified to include the blocking constraint that is satisfied exactly at the new point. (If α_B is the maximum feasible step for more than one constraint, so that there is a "tie" in the nearest constraint, an active-set method typically adds only one constraint to the working set at each iteration.)

In the strictly convex case, H is positive definite and the system (4.3) for p_k is nonsingular. However, in the general convex case, the reduced Hessian $Z_k^T H Z_k$ (and hence also the system (4.3)) may be singular. The strategy used here is to choose the working set to control the inertia of the reduced Hessian, which is never permitted to have more than one zero eigenvalue. Such methods are called *inertia-controlling*. At any initial iterate x_0, it is always possible to find enough real or temporary constraints to define a working set W_0 with a nonsingular $Z_0^T H Z_0$. Thereafter, it can be shown that the reduced Hessian can become singular only when a constraint is deleted from the working set. When $Z_k^T H Z_k$ is singular at a non-optimal point, it is used to define a direction p_Z such that

$$Z_k^T H Z_k p_Z = 0 \quad \text{and} \quad g_k^T Z_k p_Z < 0.$$

The vector $p_k = Z_k p_Z$ is a direction of unbounded descent in the sense that the objective is linear in that direction and decreases without bound along p_k. Normally, a step along p_k reaches a new constraint, which is then added to the working set for the next iteration. (If no constraint can be added to the working set, the problem must be unbounded and the algorithm terminates.) In an inertia-controlling strategy, no further constraints are deleted until the reduced Hessian becomes nonsingular.

The calculation of a direction of unbounded descent is relatively straightforward if $Z_k^T H Z_k$ is computed explicitly. In this case, p_k can be computed from a triangular system involving the (singular) Cholesky factor of $Z_k^T H Z_k$ (see, e.g., [21]). If the reduced Hessian cannot be computed (because of sparsity considerations for example) a vector p_k can be computed from a *nonsingular* system with matrix

$$K = \begin{pmatrix} H & W_k^T \\ W_k & 0 \end{pmatrix}, \tag{4.4}$$

where W_k includes the gradient a_t of the most recently deleted constraint. The theory of this type of method is based on the next result, given, e.g., by [21].

Lemma 3 *Let W_k be an $m \times n$ matrix of full row rank, and let A denote W with its t-th row omitted, so that A_k also has full row rank. The matrices Z and Z_W denote null-space bases for A_k and W_k, and $Z_k^T H Z_k$ and $Z_W^T H Z_W$ denote the associated* *reduced Hessian matrices. Define K and K_W as*

$$K = \begin{pmatrix} H & A_k^T \\ A_k & 0 \end{pmatrix} \quad and \quad K_W = \begin{pmatrix} H & W_k^T \\ W_k & 0 \end{pmatrix},$$

where H is an $n \times n$ symmetric matrix. Consider the nonsingular linear system

$$\begin{pmatrix} H & W_k^T \\ W_k & 0 \end{pmatrix} \begin{pmatrix} u_k \\ \mu_k \end{pmatrix} = K_W \begin{pmatrix} u_k \\ \mu_k \end{pmatrix} = \begin{pmatrix} 0 \\ e_t \end{pmatrix}, \tag{4.5}$$

where u_k has n components. Let $(\mu_k)_t$ denote the t-th component of μ_k. If $Z_W^T H Z_W$ is positive definite, then if $(\mu_k)_t < 0$ then $Z_k^T H Z_k$ is positive definite; if $(\mu_k)_t = 0$ then $Z_k^T H Z_k$ is singular. ∎

In the singular case, $(\mu_k)_t = 0$, and the first n equations of (4.5) imply that

$$0 = u_k^T H u_k + u_k^T A_k^T \mu_E = u_k^T H u_k.$$

It follows that the direction $p_k = u_k$ satisfies both the descent condition $p_k^T H p_k = 0$ and $g_k^T p_k < 0$, as well as the feasibility conditions $A_k p_k = 0$ and $a_t^T p_k > 0$.

It remains to be shown how the direction p_k of (4.3) may be computed from a system with matrix (4.4). The following result shows how this can be done.

Lemma 4 *Let x_k be a non-optimal subspace minimizer with respect to W_k. Let t be the index of a column of W_k such that $g_k = W_k^T \lambda_k$ and $(\lambda_k)_t < 0$. Let u_k denote the solution of the system*

$$\begin{pmatrix} H & W_k^T \\ W_k & 0 \end{pmatrix} \begin{pmatrix} u_k \\ -\nu_k \end{pmatrix} = \begin{pmatrix} -g_k \\ e_t \end{pmatrix}. \tag{4.6}$$

If p_k is the direction $p_k = \sigma_k u_k$, with $\sigma_k = (\lambda_k)_t/(\mu_k)_t$, then $x_k + p_k$ minimizes f on the manifold $\{x \mid A_k x = b_k\}$.

Proof. The vector p_k defined by the system (4.3) minimizes f on the manifold $\{x \mid a_i^T x = \beta_i, i \in \mathcal{W}_k,\ i \neq t\}$. Let $\bar{\pi}_k$ denote the m_k-vector obtained from π_k by inserting a zero component in the t-th position, i.e.,

$$(\bar{\pi}_k)_i = \begin{cases} (\pi_k)_i & \text{if } i < t; \\ 0 & \text{if } i = t; \\ (\pi_k)_{i-1} & \text{if } i > t. \end{cases}$$

Then

$$\begin{pmatrix} H & W_k^T \\ W_k & 0 \end{pmatrix} \begin{pmatrix} p_k \\ -\bar{\pi}_k \end{pmatrix} = \begin{pmatrix} -g_k \\ \sigma e_t \end{pmatrix}, \tag{4.7}$$

where $\sigma = a_t^T p_k$ and a_t is the t-th row of W_k.

Since x_k is a subspace minimizer with respect to $\mathcal{W}_k$, the optimality conditions $g_k = W_k^T \lambda_k$ must hold, and the multipliers λ_k may be interpreted as satisfying the expanded system

$$\begin{pmatrix} H & W_k^T \\ W_k & 0 \end{pmatrix} \begin{pmatrix} 0 \\ -\lambda_k \end{pmatrix} = \begin{pmatrix} -g_k \\ 0 \end{pmatrix}. \tag{4.8}$$

Subtracting this equation from (4.7) yields

$$\begin{pmatrix} H & W_k^T \\ W_k & 0 \end{pmatrix} \begin{pmatrix} p_k \\ \lambda_k - \bar{\pi}_k \end{pmatrix} = \begin{pmatrix} 0 \\ \sigma e_t \end{pmatrix}.$$

Similarly, subtracting (4.8) from (4.6) gives

$$\begin{pmatrix} H & W_k^T \\ W_k & 0 \end{pmatrix} \begin{pmatrix} u_k \\ \mu_k \end{pmatrix} = \begin{pmatrix} 0 \\ e_t \end{pmatrix}, \qquad \text{with } \mu_k = \lambda_k - \nu_k. \tag{4.9}$$

(*cf*. (4.5) These identities imply that $p_k = \sigma u_k$ and $x_k + p_k = x_k + \sigma u_k$.

It remains to find a form of σ that is independent of p_k. From the definition of p_k, the point $\alpha = 1$ minimizes $f(x_k + \alpha p_k)$ as a function of α. If σ_k minimizes $f(x_k + \alpha u_k)$, the collinearity of p_k and u_k implies that $x_k + p_k$ and $x_k + \sigma_k u_k$ are the same point. The univariate minimizer σ_k may be computed using the identity

$$\frac{d}{d\alpha}f(x_k+\alpha u_k)\bigg|_{\alpha=\sigma_k} = g_k^T u_k + \sigma_k u_k^T H u_k = 0,$$

which implies $\sigma_k = -g_k^T u_k / u_k^T H u_k$. The required form of σ_k now follows from (4.9) and (4.1) using the relations $-u_k^T H u_k = u_k^T W_k^T \mu_k = e_t^T \mu_k = (\mu_k)_t$, and $g_k^T u_k = \lambda_k^T W_k u_k = \lambda_k^T e_t = (\lambda_k)_t$. ∎

Once p_k is defined, the step α_k is required to retain feasibility with respect to constraints not in the working set. If α_B denotes the step to the blocking constraint, the step α_k is given by $\alpha_k = \min\{1, \alpha_B\}$. If $\alpha_B \geq 1$, then $x_k + p_k$ is feasible, and $x_k + p_k$ is a subspace stationary point with respect to A_k. Otherwise, if $\alpha_B < 1$, then $\alpha_k = \alpha_B$ and the working set is modified to include the blocking constraint that is satisfied exactly at the new point. (If α_B is the maximum feasible step for more than one constraint, so that there is a "tie" in the nearest constraint, an active-set method typically adds only one constraint to the working set at each iteration.)

The intent is to be able to use a system of the form (4.6) to define the search direction at every iteration. However, the properties of the linear system (4.6) of Lemma 4 depend upon x_k being a *subspace minimizer* with respect to W_k. This is certainly the case if the step $x_{k+1} = x_k + p_k$ is taken, since x_{k+1} is then a subspace minimizer with respect to the constraints in A_k. (In this case the new working set W_{k+1} is just A_k, which is W_k with row a_t^T deleted.)

Unfortunately, stationarity with respect to A_k does not usually hold when $\alpha_k < 1$ (i.e., when progress along p_k is impeded by a blocking constraint). This difficulty is treated by redefining what is meant by a "working set". In particular, the most recently deleted constraint $a_t^T x \geq \beta_t$ may be *retained* in the working set even though the constraint may be inactive at the current point. Suppose that p_k is computed as in Lemma 4, and a new constraint $a_r^T x \geq \beta_r$ is added to the working set at $x_{k+1} = x_k + \alpha_k p_k$. Define the new working-set matrix as

$$W_{k+1} = \begin{pmatrix} A_{k+1} \\ a_t^T \end{pmatrix}, \quad \text{where} \quad A_{k+1} = \begin{pmatrix} A_k \\ a_r^T \end{pmatrix}.$$

Consider a non-optimal subspace minimizer x_k at which all the working-set constraints *are* active. If W_k is partitioned in terms of A_k and the constraint to be deleted (see (4.2)), it follows that

$$g_k = A_k^T \lambda_E^k + \lambda_t^k a_t, \quad \text{with} \quad \lambda_t^k < 0, \tag{4.10}$$

where λ_E^k denotes the multiplier vector corresponding to A_k and λ_t^k denotes the (necessarily negative) multiplier $(\lambda_k)_t$.

The next result shows that x_{k+1} is itself a subspace minimizer with respect to a particular set of *shifted* constraints with working-set matrix W_{k+1}.

Lemma 5 *Let g_k and W_k denote the gradient and working set at a non-optimal subspace minimizer x_k. Assume that W_k is partitioned as in (4.2), with a_t defined so that*

$$g_k = A_k^T \lambda_E^k + \lambda_t^k a_t, \quad \textit{with} \quad \lambda_t^k < 0, \tag{4.11}$$

for some vector λ_E^k and scalar λ_t^k. Let $x_{k+1} = x_k + \alpha_k p_k$, and assume that constraint a_r is added to W_k at x_{k+1} to give the working set W_{k+1}. Then

(a) g_{k+1}, the gradient at x_{k+1}, is also a linear combination of A_k^T and a_t;

(b) There exist a vector λ_E^{k+1} and scalar λ_t^{k+1} such that

$$g_{k+1} = A_{k+1}^T \lambda_E^{k+1} + \lambda_t^{k+1} a_t, \quad \textit{with} \quad \lambda_t^{k+1} < 0. \tag{4.12}$$

Proof. Because f is quadratic, the gradients at x_k and x_{k+1} are related by the identity

$$g_{k+1} = g(x_k + \alpha_k p_k) = g_k + \alpha_k H p_k. \tag{4.13}$$

The form of g_{k+1} for the two definitions of p_k is now considered.

When the reduced Hessian is positive definite, p satisfies (4.3), and it follows that $Hp_k = -g_k + A_k^T\pi_k$. Substituting from this expression and (4.10) in (4.13), yields

$$g_{k+1} = g_k + \alpha_k Hp_k = (1-\alpha_k)g_k + \alpha_k A_k^T\pi_k = A_k^T\nu_E + \lambda_t^{k+1}a_t,$$

where $\nu_E = (1-\alpha_k)\lambda_k + \alpha_k\pi_k$ and $\lambda_t^{k+1} = (1-\alpha_k)\lambda_t^k$. Because a constraint was added to the working set at x_{k+1}, it must hold that $\alpha_k < 1$ and hence $\lambda_t^{k+1} < 0$. The expression (b) is obtained by forming λ_{k+1} from ν_E and a zero component corresponding to row a_r^T in A_{k+1}.

When the reduced Hessian is singular, p_k is defined as u_k, where u_k satisfies (4.9), so that $Hp_k = -A_k^T\mu_E$. Substituting from this relation and (4.10) in (4.13) gives

$$g_{k+1} = g_k + \alpha_k Hp_k = g_k - \alpha_k A_k^T\mu_E = A_k^T(\lambda_E^k - \alpha_k\mu_E) + \lambda_t^k a_t,$$

and (4.12) holds with $\lambda_t^{k+1} = \lambda_t^k$ and λ_E^{k+1} formed by augmenting $\nu_E = \lambda_E^k - \alpha_k\mu_E$ with a zero component as above. ∎

This lemma implies that x_{k+1} can be regarded as the solution of a problem in which the most recently deleted constraint is *shifted* to pass through the point x_{k+1}; i.e., x_{k+1} solves the equality-constraint problem

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & c^Tx + \tfrac{1}{2}x^THx \\ \text{subject to} & A_kx = b_k, \quad a_t^Tx = a_t^Tx_{k+1}. \end{array}$$

Note that this problem can be solved without needing to know x_{k+1} in advance. The optimal multipliers are the quantities $(\lambda_E^{k+1}, \lambda_t^{k+1})$ of Lemma 5. More specifically, the scalar λ_t^k is the multiplier corresponding to the shifted constraint $a_t^Tx \geq a_t^Tx_{k+1}$.

If the step $\alpha_k = 1$ is taken to reach the point x_{k+1}, this multiplier becomes *zero*, which implies that x_{k+1} is a subspace minimizer with respect to both A_k and W_k.

A simple inductive argument shows that in a sequence of consecutive steps at which a constraint is added to the working set, each iterate is a subspace minimizer of an appropriately shifted problem. To simplify the notation, consider an initial feasible point x_0, and an initial working set A_0 such that x_0 is not a minimizer with respect to A_0. It follows from the above discussion that a sequence of $N+1$ ($N \geq 0$) consecutive iterates is defined such that x_{N+1} is the first minimizer—i.e., x_{N+1} is a minimizer with respect to A_{N+1}, but none of the x_k is a minimizer with respect to A_k for $0 \leq k \leq N$. At each of the first N steps, $\alpha_k < 1$ and a constraint is added to the working set. At the Nth iteration one of two things can happen. If $A_{N+1} = A_N$ and an "unconstrained" step $\alpha_N = 1$ is taken, the sequence of iterates may be represented schematically as

$$A_0 \quad \xrightarrow{\textit{move \& add}} \quad A_1 \quad \cdots \quad \xrightarrow{\textit{move \& add}} \quad A_N \quad \xrightarrow{\textit{move}} \quad A_{N+1}\,(\mathrm{min})$$

Alternatively, x_{N+1} can be a minimizer after a constrained step $\alpha_N < 1$. This will happen, for example, if x_{N+1} is a vertex, since a vertex is trivially a subspace minimizer with respect to A_k. In this case,

$$A_0 \quad \xrightarrow{\textit{move \& add}} \quad A_1 \quad \cdots \quad \xrightarrow{\textit{move \& add}} \quad A_N \quad \xrightarrow{\textit{move \& add}} \quad A_{N+1}\,(\mathrm{min})$$

The structure above suggests the name of *intermediate* iterates for the points $\{x_k\}$, $k = 0, 1, \ldots, N$. Essentially, a point x_k is an intermediate iterate if it is not a subspace stationary point with respect to A_k.

If a constraint is now deleted at x_{N+1}, a new sequence of intermediate iterates is started, beginning with the point x_{N+1} (suitably relabeled to reflect the fact that a constraint has been deleted from A_{N+1}). Note that for any sequence of consecutive intermediate iterates, N can be zero (when the first step locates a minimizer), but no greater than n (the number of steps from an unconstrained point to a vertex minimizer).

The properties of a group of *consecutive* intermediate iterates that occur *after* a constraint is deleted at a subspace minimizer, but *before* the next minimizer is reached, are now examined. Each such iterate is associated with a unique *most recently deleted constraint*. Consider a sequence of consecutive intermediate iterates

$\{x_k\}$, $k = 0, \ldots, N$, such that x_k is intermediate with respect to A_k. Assume that the first working set has the form

$$W_0 = \begin{pmatrix} A_0 \\ a_t^T \end{pmatrix}, \tag{4.14}$$

and that x_0 is a subspace minimizer with respect to W_0.

Let Z_W denote a basis for the null space of W. Because of the inertia-controlling strategy, the reduced Hessian $Z_W^T H Z_W$ must be positive definite. Relation (4.14) implies that

$$p^T H p > 0 \text{ for any nonzero } p \text{ such that } A_0 p = 0 \text{ and } a_t^T p = 0. \tag{4.15}$$

If the iterate following x_k is intermediate and the algorithm continues, α_k is the step to the nearest constraint, and *a constraint is added to the working set* at each x_k, $k \geq 1$. If a constraint is added, *it must hold that* $\alpha_k < 1$. (Otherwise, if $\alpha_k = 1$, $x_k + p_k$ is a subspace minimizer with respect to A_k, and the sequence of intermediate iterates ends.) Let a_k denote the normal of the constraint added to A_k at x_{k+1} to produce A_{k+1}, so that the form of W_{k+1} and A_{k+1} is

$$W_{k+1} = \begin{pmatrix} A_{k+1} \\ a_t^T \end{pmatrix}, \quad \text{and} \quad A_{k+1} = \begin{pmatrix} A_k \\ a_k^T \end{pmatrix} = \begin{pmatrix} A_0 \\ a_0^T \\ \vdots \\ a_k^T \end{pmatrix}. \tag{4.16}$$

Some useful results are now derived concerning the sequence of intermediate iterates.

Lemma 6 *Given a sequence of consecutive intermediate iterates $\{x_k\}$, the gradient g_k satisfies (4.11) for $k \geq 0$.*

Proof. To begin the induction, note that if the multiplier associated with a_t at x_0 is *negative*, then, from (4.14),

$$g_0 = A_0^T \lambda_E^0 + \lambda_t^0 a_t,$$

where $\lambda_t^0 < 0$. Lemma 5 therefore applies at all subsequent intermediate iterates, with the convention that λ increases in dimension by one at each step to reflect the fact that A_k has one more row than A_{k-1}. ∎

Lemma 7 *Let $\{x_k\}$ be a sequence of consecutive intermediate iterates. Each search direction satisfies $g_k^T p_k < 0$ and $a_t^T p_k > 0$.*

Proof. From part (b) of Lemma 6, for the stated values of k, there exist a vector λ_E^k and positive scalar λ_t^k such that

$$g_k = A_k^T \lambda_E^k + \lambda_t^k a_t.$$

Therefore, $g_k^T p_k = \lambda_t^k a_t^T p_k$ and the desired results are immediate. ∎

Lemma 8 *Let $\{x_k\}$, $k = 0, \ldots, N$, denote a sequence of consecutive intermediate iterates. Assume further that α_N is the step to the constraint with normal a_N, which is added to A_N to form the working set A_{N+1}. Let $x_{N+1} = x_N + \alpha_N p_N$. Then*

- *(a) If x_{N+1} is a stationary point with respect to A_{N+1}, then a_N is linearly dependent on A_N^T and a_t, and $Z_{N+1}^T H Z_{N+1}$ is positive definite;*
- *(b) If a_N is linearly dependent on A_N^T and a_t, then x_{N+1} is a minimizer with respect to A_{N+1}.*

Proof. By construction, the working set (4.14) has full row rank, so that a_t is linearly independent of the rows of A_0. From part (b) of Lemma 6

$$g_k = A_k^T \lambda_E^k + \lambda_t^k a_t, \qquad k = 0, \ldots, N, \tag{4.17}$$

where $\lambda_t^k < 0$. Since it is assumed that x_k is not a subspace stationary point with respect to A_k for any $1 \le k \le N$, (4.17) shows that a_t^T is linearly independent of A_k. Furthermore, part (a) of Lemma 5 implies that there exists a vector ν_E such that

$$g_{N+1} = A_N^T \nu_E + \lambda_t^{N+1} a_t, \tag{4.18}$$

where $\lambda_t^{N+1} < 0$. It follows from the linear independence of a_t^T and A_N that x_{N+1} cannot be a minimizer with respect to A_N.

To show part (a), assume that x_{N+1} is a stationary point with respect to A_{N+1} (which includes a_N), i.e.,

$$g_{N+1} = A_N^T \pi + \pi_N a_N, \tag{4.19}$$

where π_N (the multiplier associated with a_N) must be nonzero. Equating the right-hand sides of (4.18) and (4.19), yields

$$A_N^T \nu + \lambda_t^{N+1} a_t = A_N^T \pi + \pi_N a_N. \tag{4.20}$$

Since $\lambda_t^{N+1} \neq 0$ and $\pi_N \neq 0$, this expression implies that a_t may be expressed as a linear combination of A_N^T and a_N, where the coefficient of a_N is *nonzero*:

$$a_t = A_N^T \xi + \gamma a_N, \quad \text{with } \gamma = \frac{\pi_N}{\lambda_t^{N+1}} \neq 0 \tag{4.21}$$

and $\xi = (1/\lambda_t^{N+1})(\pi - \nu)$.

Stationarity of x_{N+1} with respect to A_{N+1} thus implies a *special relationship* among the most recently deleted constraint, the working set at x_N and the newly encountered constraint. Any nonzero vector p in the null space of A_{N+1} satisfies

$$A_{N+1} p = \begin{pmatrix} A_N \\ a_N^T \end{pmatrix} p = 0. \tag{4.22}$$

For any such p, it follows from the structure of A_{N+1} (see (4.16)) that $A_0 p = 0$, and from (4.21) that $a_t^T p = 0$; hence, p lies in the null space of W. Since $Z_W^T H Z_W$ is positive definite (i.e., (4.15) holds), it follows that $p^T H p > 0$ for p satisfying (4.22). Thus, the reduced Hessian at x_{N+1} with respect to A_{N+1} is *positive definite*, and x_{N+1} is a minimizer with respect to A_{N+1}.

To verify part (b), assume that a_N is linearly dependent on A_N and a_t, i.e., that $a_N = A_N^T \beta + a_t \beta_t$, where $\beta_t \neq 0$. Simple rearrangement then gives $a_t = (1/\beta_t) a_N - (1/\beta_t) A_N^T \beta$. Substituting in (4.18), yields

$$g_{N+1} = A_N^T \nu + \frac{\lambda_t^{N+1}}{\beta_t} a_N + \frac{\lambda_t^{N+1}}{\beta_t} A_N^T \beta,$$

which shows that x_{N+1} must be a subspace stationary point with respect to A_{N+1}. Positive definiteness of the reduced Hessian follows as before, and hence x_{N+1} is a minimizer with respect to A_{N+1}. ▮

Lemma 8 is crucial in ensuring that *adding* a constraint in an inertia-controlling algorithm cannot produce a stationary point where the reduced Hessian is not positive definite.

4.2 Implementation

In addition to the search direction and work vectors, the proposed algorithm requires the storage of the five vectors: u, μ, z, η, and λ, where λ is the multiplier vector satisfying $W^T\lambda = g$. These vectors are needed to solve the following systems:

$$\text{System 0 (initialization)} \quad \begin{pmatrix} H & W_0^T \\ W_0 & 0 \end{pmatrix} \begin{pmatrix} p_0 \\ -\lambda_0 \end{pmatrix} = - \begin{pmatrix} g_0 \\ r_0 \end{pmatrix}, \tag{4.23}$$

$$\text{System 1 (constraint deletion)} \quad \begin{pmatrix} H & W^T \\ W & 0 \end{pmatrix} \begin{pmatrix} u \\ \mu \end{pmatrix} = \begin{pmatrix} 0 \\ e_t \end{pmatrix}, \tag{4.24}$$

$$\text{System 2 (constraint addition)} \quad \begin{pmatrix} H & W^T \\ W & 0 \end{pmatrix} \begin{pmatrix} z \\ \eta \end{pmatrix} = \begin{pmatrix} a_r \\ 0 \end{pmatrix}. \tag{4.25}$$

System 0 is solved only once, for the initial values of x and λ. At a subspace minimizer, System 1 is solved for u and μ. The search direction p is defined to be either σu or u depending on whether or not the reduced Hessian is singular after a_t is deleted from the working set. (This information is provided by the sign of $(\mu)_t$). Once the step α has been determined, the multipliers λ are updated to reflect the change in g resulting from the move from x to $x + \alpha p$. (The specific updates require the current values of the vectors u and μ.) If a constraint is added to the working set, the vectors u, μ and λ must be updated to reflect the addition of the new row to W. These vectors are updated using the quantities z and η calculated from System 2. Each time a constraint is added, u and μ are updated, requiring an additional solve with System 2.

This discussion implies that a sequence of N intermediate iterations involves the solution of $N+1$ systems, comprising one solve with System 1 and N solves with System 2.

4.2.1 Updating the Required Vectors

This section concerns the details of how u, μ and λ are updated. To simplify the discussion, the subscript k is suppressed whenever possible. With this notation, unbarred quantities belong to iteration k. Similarly, a bar over a quantity will indicate its updated value.

Lemma 9 *(Move to a new iterate.) Suppose that x is an iterate of an inertia-controlling method. Let $\bar{x} = x + \alpha p$. The solution of $W^T\bar{\lambda} = \bar{g}$, where $\bar{g} = g(\bar{x}) = g + \alpha Hp$, is related to the vector λ via the equations*

$$\bar{\lambda}_E = \lambda_E - \alpha(a_t^T p)\mu_E \qquad \textit{and} \qquad \bar{\lambda}_t = \begin{cases} (1-\alpha)\lambda_t & \textit{if } p = \sigma u; \\ \lambda_t - \alpha\mu_t & \textit{if } p = u. \end{cases}$$

Proof. In this lemma, the move from x to $\bar{x}$ changes only the gradient (not the working set). The desired result can be obtained by substituting p and the relation $Hu = -W^T\mu$ (from (4.24)) in the expression $\bar{g} = g + \alpha Hp$. ∎

Updates following the addition of a constraint (say, a_r) to the working set use vectors z and η defined by the system

$$\begin{pmatrix} H & W^T \\ W & 0 \end{pmatrix} \begin{pmatrix} z \\ \eta \end{pmatrix} = \begin{pmatrix} a_r \\ 0 \end{pmatrix}, \tag{4.26}$$

i.e., z and η satisfy

$$Hz + A^T\eta_E + \eta_t a_t = a_r, \quad Az = 0 \quad \text{and} \quad a_t^T z = 0. \tag{4.27}$$

The case when a_r can be added directly to W is considered first. Following the updates given in the next lemma, m increases by one and the "new" λ and μ have one additional component.

Lemma 10 *(Constraint addition; independent case.) Let x denote an iterate of an inertia-controlling method. Assume that constraint a_r is to be added to the working set at x, where W^T and a_r are linearly independent. Let $\rho = a_r^T u / a_r^T z$. Then the vectors $\bar{u}$ and $\bar{\mu}$ defined by*

$$\bar{u} = u - \rho z, \qquad \bar{\mu} = \begin{pmatrix} \mu - \rho\eta \\ \rho \end{pmatrix} \tag{4.28}$$

satisfy

$$\begin{pmatrix} H & \bar{W}^T \\ \bar{W} & 0 \end{pmatrix} \begin{pmatrix} \bar{u} \\ \bar{\mu} \end{pmatrix} = \begin{pmatrix} 0 \\ e_t \end{pmatrix}. \tag{4.29}$$

Proof. When a_r and W^T are linearly independent, (4.27) shows that z must be nonzero. Since $Wz = 0$ and $Z_W^T H Z_W$ is positive definite, $a_r^T z = z^T H z > 0$, so that ρ is well defined.

For any scalar ρ, (4.24) and (4.26) imply that

$$\begin{pmatrix} H & W^T & a_r \\ W & 0 & 0 \\ a_r^T & 0 & 0 \end{pmatrix} \begin{pmatrix} u - \rho z \\ \mu - \rho\eta \\ \rho \end{pmatrix} = \begin{pmatrix} 0 \\ e_t \\ a_r^T u - \rho a_r^T z \end{pmatrix}. \tag{4.30}$$

The linear independence of a_r and W^T means that the solution vectors of (4.29) are unique. If ρ is chosen so that the last component of the right-hand side of (4.30) vanishes, then the $\bar{u}$ and $\bar{\mu}$ of (4.28) satisfy (4.29). ∎

If $Z^T H Z$ is positive definite and $W \neq A$, a_t can be deleted from W, and W then becomes A itself. The following lemma may be applied in two situations: when a constraint is deleted from the working set at a minimizer and the reduced Hessian remains positive definite after deletion; and at an intermediate iterate after a constraint has been added that makes $Z^T H Z$ positive definite.

Lemma 11 *(Deleting a_t from W.) Suppose that x is an iterate of an inertia-controlling method, $W \neq A$, and $Z^T HZ$ is positive definite. Then the vector $\bar{\lambda}$ is defined by*

$$\bar{\lambda} = \lambda_E + \rho\mu_E, \quad \textit{where} \quad \rho = -\frac{\lambda_t}{\mu_t}, \tag{4.31}$$

satisfies $A^T \bar{\lambda} = g$.

Proof. Let $\lambda' = \lambda + \rho\mu$ for some scalar ρ. Substituting these values in $W^T \lambda = g$, yields

$$A^T(\lambda_E + \rho\mu_E) + a_t(\lambda_t + \rho\mu_t) = g.$$

It follows that $A^T \bar{\lambda} = g$ will be satisfied by λ'_E if $\lambda_t + \rho\mu_t = 0$. It is permissible to delete a_t from W only if $Z^T HZ$ is positive definite, which means that $\mu_t < 0$, and hence ρ is well defined. ∎

Note that u and μ are no longer needed to define the search direction after a_t has been removed.

The only remaining possibility occurs when a_r, the constraint to be added, is linearly dependent on W^T; in this case, $z = 0$ in (4.26). It follows from Lemma 8 that the iterate just reached must be a *minimizer with respect to the working set composed of A^T and a_r*, which means that a_t is no longer necessary. However, it is not possible to apply Lemma 11 because μ_t may be zero. The following lemma gives an update that simultaneously removes a_t from W and adds a_r to the working set. After application of these updates, $\bar{A}$ is the "real" working set at $\bar{x}$, and the algorithm either terminates or deletes a constraint (which cannot be a_r).

Lemma 12 *(Constraint addition; dependent case.) Suppose that x is an intermediate iterate. Assume that a_r is to be added to the working set at x, and that a_r and W^T are linearly dependent. Let $\bar{A}$ denote A with a_r^T as an additional row, and define $\omega = \lambda_t/\eta_t$. The vector $\bar{\lambda}$ specified by*

$$\bar{\lambda}_E = \lambda_E - \omega\eta_E, \quad \bar{\lambda}_a = \omega, \tag{4.32}$$

where $\bar{\lambda}_a$ denotes the component of $\bar{\lambda}$ corresponding to a_r, satisfies $\bar{W}^T \bar{\lambda} = g$.

Proof. First, observe that linear dependence of W^T and a_r means that $z = 0$. As the initial working set has full row rank, for an active-set QP algorithm of the form described, any constraint added to the working set must be linearly independent of the constraints in the working set. Hence a_r cannot be linearly dependent on A^T, which implies that $\eta_t \neq 0$. Lemma 8 tells us that x must be a subspace minimizer with respect to a working set. The desired results follow from substitution. ∎

Algorithm 4.2.1. *Convex Quadratic Programming.*

```
Solve System 0 for p_0 and λ_0;
x ← x_0 + p_0;
subspace_minimizer ← true;  λ ← λ_0;
repeat
   if subspace_minimizer then
      λ_t ← minimum λ ;
      optimal ← (λ_t ≥ 0);
      if optimal then stop;
      [Solve System 1 for u and μ;]
      singular_H ← (μ_t = 0);
   end
   p ← if singular_H then u else (λ_t/μ_t)u end;
   α_B ← maximum feasible step along p;
   α ← if singular_H then α_B else min{1, α_B} end;
   x ← x + αp;
   λ ← if singular_H then λ − αμ else λ − α(λ_t/μ_t)μ end;
   subspace_minimizer ← λ_t = 0;
   hit_constraint ← (α = α_B);
   if hit_constraint then
      [Solve System 2 for z and η;]
      singular_W ← (z = 0);
      if singular_W then
         Swap a_r and a_t;
         ω ← λ_t/η_t;  λ_E ← λ_E − ωη_E;  λ_t ← ω;
         subspace_minimizer ← true;
```

else

Add constraint with normal a_r;

Define $\rho \leftarrow a_r^T u / a_r^T z$;

$u \leftarrow u - \rho z; \;\; v \leftarrow \begin{pmatrix} \mu - \rho\eta \\ \rho \end{pmatrix}; \;\; \lambda \leftarrow \begin{pmatrix} \lambda \\ 0 \end{pmatrix};$

end;

else

Delete t-th row of W;

end;

until *optimal*;

4.3 Special Properties of the Standard Form

For the development of the Schur-complement (SC) method, the QP is assumed to be of the form

$$\begin{array}{llcl} \underset{x}{\text{minimize}} & \psi(x) & = & c^T x + \frac{1}{2} x^T H x \\ \text{subject to} & Ax & \geq & b, \quad x \geq 0, \end{array}$$

where H is positive semi-definite. The corresponding slack variable form is given by

$$\begin{array}{llcl} \underset{x,s}{\text{minimize}} & \psi(x) & = & c^T x + \frac{1}{2} x^T H x, \\ \text{subject to} & Ax - s & = & b, \quad x \geq 0, \quad s \geq 0, \end{array} \tag{4.33}$$

where s are the slack variables. This form simplifies the notation used in the following sections, however, the results of these sections can easily be extended to the general form given in (2.1).

So far, we have discussed the role of the working set matrix W without particular attention to the computational advantages that arise when the constraints are in the standard form (4.33). Standard form allows the nonzero ("free") components of the search direction to be computed using a matrix whose column dimension is equal to the number of free variables (rather than the total number of variables). To formalize this idea, assume that W is associated with a working set containing only active constraints, and let n_{FR} denote the number of free variables (i.e., corresponding to bounds not in W), and let the subscript "$_{FR}$" denote the corresponding components of a vector or matrix. For example, A_{FR} denotes the $m \times n_{FR}$ submatrix of columns of A corresponding to free variables. Similarly, the subscript "$_{FX}$" means the components corresponding to *fixed* variables (i.e., those whose bounds are in the working set). (We shall henceforth switch freely between the terminologies of "working sets" and "free/fixed variables".)

Since the rows of $\begin{pmatrix} A & -I \end{pmatrix}$ are linearly independent, a working set will include all the constraints $Ax - s = b$ together with the gradients of a subset of the nonnegativity constraints. Hence, we can always find a permutation P such that

$$WP = \begin{pmatrix} A_{FR} & A_{FX} \\ 0 & I_{FX} \end{pmatrix}, \tag{4.34}$$

where A_{FR} and A_{FX} contain the columns of $\begin{pmatrix} A & -I \end{pmatrix}$ corresponding to the free and fixed variables respectively. In the notation of Chapter 2, $A_{FX} = N$ and $A_{FR} = \begin{pmatrix} B & S \end{pmatrix}$.

Given this form of the working set, the symmetrically permuted version of System 1 can be written as

$$\begin{pmatrix} H_{FR} & H_{OD} & A_{FR}^T & 0 \\ H_{OD}^T & H_{FX} & A_{FX}^T & I_{FX} \\ A_{FR} & A_{FX} & 0 & 0 \\ 0 & I_{FX} & 0 & 0 \end{pmatrix} \begin{pmatrix} u_{FR} \\ u_{FX} \\ \mu_E \\ \mu_X \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ e_t \end{pmatrix},$$

where H_{OD} is the matrix of off-diagonal mixed terms of the Hessian, and μ_E and μ_B are elements of μ corresponding to the general and bound constraints respectively. Close inspection of these partitioned equations reveals that $u_{FX} = e_t$ and $\mu_B = -A_{FX}^T \mu_E - H_{OD}^T u_{FR} - h_t$, where u_{FR} and μ_E satisfy

$$\begin{pmatrix} H_{FR} & A_{FR}^T \\ A_{FR} & 0 \end{pmatrix} \begin{pmatrix} u_{FR} \\ \mu_E \end{pmatrix} = - \begin{pmatrix} h_t \\ a_t \end{pmatrix}. \tag{4.35}$$

This system, which is analogous to System 1, preserves the two-by-two block structure of the KKT matrix, but only involves the free variables.

Now we turn to System 2. The symmetrically permuted system is

$$\begin{pmatrix} H_{FR} & H_{OD} & A_{FR}^T & 0 \\ H_{OD}^T & H_{FX} & A_{FX}^T & I_{FX} \\ A_{FR} & A_{FX} & 0 & 0 \\ 0 & I_{FX} & 0 & 0 \end{pmatrix} \begin{pmatrix} z_{FR} \\ z_{FX} \\ \eta_E \\ \eta_X \end{pmatrix} = \begin{pmatrix} e_r \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

where the elements of η_E and η_B match the general and bound constraints as above. These equations give $z_{FX} = 0$ and $\eta_B = -H_{OD}^T u_{FR} - A_{FX}^T \eta_E$, where u_{FR} and η_E satisfy

$$\begin{pmatrix} H_{FR} & A_{FR}^T \\ A_{FR} & 0 \end{pmatrix} \begin{pmatrix} z_{FR} \\ \eta_E \end{pmatrix} = \begin{pmatrix} e_r \\ 0 \end{pmatrix}. \tag{4.36}$$

This is the "free-variable form" of System 2.

4.4 Schur Complement QP

Schur-complement QP methods are based on the properties of certain *block-bordered* linear systems. Consider the nonsingular block two-by-two system

$$\begin{pmatrix} K_0 & V \\ V^T & D \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}, \tag{4.37}$$

where K_0 and D are symmetric. This system can be solved using factorizations of K_0 and C, the *Schur complement of* K_0:

$$C \equiv D - V^T K_0^{-1} V, \tag{4.38}$$

(see, e.g., Bisschop and Meerhaus, [1, 2]; Gill *et al.*, [17]). To show this, we write

$$\begin{pmatrix} K_0 & V \\ V^T & D \end{pmatrix} = \begin{pmatrix} K_0 & 0 \\ V^T & I \end{pmatrix} \begin{pmatrix} I & W \\ 0 & C \end{pmatrix},$$

where W satisfies $K_0 W = V$. A simple calculation gives

$$\begin{pmatrix} K_0 & 0 \\ V^T & I \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}, \qquad \begin{pmatrix} I & W \\ 0 & C \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix},$$

and the following equations are solved in turn:

$$\begin{aligned} K_0 z_1 &= f_1 \\ z_2 &= f_2 - V^T z_1 \\ C y_2 &= z_2 \\ K_0 y_1 &= f_1 - V y_2 \quad \text{or} \quad y_1 = z_1 - W y_2 \end{aligned} \tag{4.39}$$

It will be shown in the next section that solutions of System 1 (4.35) and System 2 (4.36) can be defined in terms of a block two-by-two matrix of the form (4.37). Moreover, at each iteration, a column is added to V and a row and column are added to D. This results in the addition of a single row and column to the Schur complement C. To show this, we write the associated KKT system as

$$\begin{pmatrix} K_0 & \bar{V} \\ \bar{V}^T & \bar{D} \end{pmatrix}, \quad \text{where} \quad \bar{V} = \begin{pmatrix} V & v \end{pmatrix} \quad \text{and} \quad \bar{D} = \begin{pmatrix} D & d \\ d^T & \delta \end{pmatrix}. \tag{4.40}$$

(The definitions of v, d and δ depend on the nature of the change in the working set.) The new Schur complement $\bar{C}$ for (4.40) is given by

$$\bar{C} = \bar{D} - \bar{V}^T K_0^{-1} \bar{V} = \begin{pmatrix} D & d \\ d^T & \delta \end{pmatrix} - \begin{pmatrix} V^T \\ v^T \end{pmatrix} K_0^{-1} \begin{pmatrix} V & v \end{pmatrix}. \tag{4.41}$$

Comparison of this identity with (4.38) reveals that the Schur complement is bordered by a single row and column;

$$\bar{C} = \begin{pmatrix} C & c \\ c^T & \tau \end{pmatrix}, \tag{4.42}$$

where

$$K_0 q = v, \qquad c = d - V^T q \quad \text{and} \quad \tau = \delta - v^T q. \tag{4.43}$$

Note that a solve with K_0 is needed to update C.

4.4.1 The Schur Complement Update

To illustrate the application of the Schur-complement method to convex QP, we consider a sequence of intermediate iterates, starting at a non-optimal subspace minimizer (x_0, s_0). Let K_0 denote the KKT matrix of free variables at (x_0, s_0), with

$$K_0 = \begin{pmatrix} H_0 & A_0^T \\ A_0 & 0 \end{pmatrix},$$

where A_0 and H_0 denote the free rows and columns of $(\; A \quad -I \;)$ and H. If variable t_0 is moved away from its bound at (x_0, s_0), it is necessary to solve System 1 (4.35) at (x_0, s_0), i.e.,

$$\begin{pmatrix} H_0 & A_0^T \\ A_0 & 0 \end{pmatrix} \begin{pmatrix} u_0 \\ \mu_0 \end{pmatrix} = - \begin{pmatrix} h_{t_0} \\ a_{t_0} \end{pmatrix},$$

where h_{t_0} and a_{t_0} denote the t_0-th columns of H and $(\; A \quad -I \;)$. The vector u_0 is used to define p_0.

Assume that the next iterate (x_1, s_1) is intermediate, with variable r_1 blocking at $(x_1, s_1) = (x_0, s_0) + \alpha_0 p_0$. In this case, u_1 and μ_1 are found by updating the vectors u_0 and μ_0 using z_1 and η_1 defined by System 2 (4.36), i.e.,

$$\begin{pmatrix} H_0 & A_0^T \\ A_0 & 0 \end{pmatrix} \begin{pmatrix} z_1 \\ \eta_1 \end{pmatrix} = \begin{pmatrix} e_{r_1} \\ 0 \end{pmatrix}.$$

Once u_1 is known, the new iterate $(x_2, s_2) = (x_1, s_1) + \alpha_1 p_1$ is calculated using the direction p_1.

So far, all the systems to be solved have been in terms of K_0. At (x_1, s_1), however, the set of free variables will either increase or decrease, thereby increasing or decreasing the the dimension of the underlying KKT system changes. The two alternative cases will be considered separately.

First, assume that (x_2, s_2) is also intermediate, with variable r_2 blocking. One obvious way to proceed would be to redefine the set of fixed and free variables and solve System 2 (4.36) with right-hand-side e_{r_2} for z_2 and η_2. This method would need to efficiently solve a KKT system that varies in composition and size as the set of fixed and free variables changes. In contrast, we now show that the special nature of these changes allows us to define an algorithm in which the solution of (4.35) and (4.36) may be obtained during k successive iterations using a *fixed* sparse factorization of K_0, and a dense factorization of the Schur complement of (at most) order k.

We return to the calculation of z_2 and η_2 from System 2 (4.36). Consider the system

$$\begin{pmatrix} H_0 & A_0^T & e_{r_1} \\ A_0 & 0 & 0 \\ e_{r_1}^T & 0 & 0 \end{pmatrix} \begin{pmatrix} z_2' \\ \eta_2' \\ \eta_{r_1}' \end{pmatrix} = \begin{pmatrix} e_{r_2} \\ 0 \\ 0 \end{pmatrix}.$$

Every component of z_2' is a component of z_2 except the r_2th, which has been forced to be zero. The variable η_{r_1}' appears only in the r_1th equation, which is redundant. Observe that the matrix of this system has the form

$$K_1 \triangleq \begin{pmatrix} K_0 & e_{r_1} \\ e_{r_1}^T & 0 \end{pmatrix}, \tag{4.44}$$

which has the bordered structure (4.37). Solutions with K_1 can be determined using the Schur complement of K_0 in K_1.

Once z_2 and η_2 are computed, they are used to define new vectors u_2 and μ_2 and to define the Schur complement update for

$$K_2 = \begin{pmatrix} K_1 & e_{r_2} \\ e_{r_2}^T & 0 \end{pmatrix}$$

(*cf*. (4.41)). Subsequent intermediate iterates are treated the same way, i.e., after each step to an intermediate iterate, the KKT matrix is bordered by a row and column of the identity matrix.

Now suppose that $(x_2, s_2) = (x_1, s_1) + \alpha_1 p_1$ is a subspace minimizer instead of an intermediate iterate. In this case, variable t_0 is added to the list of free variables, which implies that the Schur complement for the bordered matrix

$$\begin{pmatrix} H_0 & A_0^T & e_{r_1} & h_{t_0} \\ A_0 & 0 & 0 & a_{t_0} \\ e_{r_1}^T & 0 & 0 & 0 \\ h_{t_0}^T & a_{t_0}^T & 0 & h_{t_0,t_0} \end{pmatrix}$$

is computed from the Schur complement of (4.44). Given a new variable t_2 to be moved from its bound, the new values of u and μ are determined from the system

$$\begin{pmatrix} H_0 & A_0^T & e_{r_1} & h_{t_0} \\ A_0 & 0 & 0 & a_{t_0} \\ e_{r_1}^T & 0 & 0 & 0 \\ h_{t_0}^T & a_{t_0}^T & 0 & h_{t_0,t_0} \end{pmatrix} \begin{pmatrix} u_2' \\ \mu_2' \\ \mu_{r_1}' \\ u_{t_0}' \end{pmatrix} = - \begin{pmatrix} (h_{t_2})_0 \\ a_{t_2} \\ 0 \\ h_{t_0,t_2} \end{pmatrix},$$

where $(h_{t_2})_0$ denotes the elements of the t_2th column of H that correspond to the list of free variables at (x_0, s_0).

We have shown that at each iteration, the free-variable KKT system is bordered by a single row and column. When a variable is fixed on its bound, this column is a column of the identity. When a variable is added to the free list, the column is formed from the free elements of a row and column of H and a column of $\begin{pmatrix} A & -I \end{pmatrix}$.

A typical iteration requires three solves with K_0: two solves to obtain the solution of System 1 or 2, and one solve to update the Schur complement. However, if the vectors of the form

$$K_0^{-1} \begin{pmatrix} h_t \\ a_t \end{pmatrix} \quad \text{and} \quad K_0^{-1} \begin{pmatrix} e_r \\ 0 \end{pmatrix}$$

are stored as they are computed, only one solve with K_0 is required at each step.

4.4.2 Algorithmic Details

When a variable $(x, s)_j$ changes state, there are only three possible changes:

$$\begin{array}{ccc} \text{free} & \rightarrow & \text{fixed}, \\ \text{fixed} & \rightarrow & t, \\ t & \rightarrow & \text{free}. \end{array}$$

The notation $\rightarrow t$ implies that $(x, s)_j$ becomes the t-th variable, and $t \rightarrow$ implies that the t-th variable is changing state.

Initially $t = 0$, so all the variables are either free or fixed. For all possible relative states to K_0, Table 4.1 indicates the effect on the size of C, and the value of t, when each of these states is encountered.

Table 4.1: The change in value of t and size of C for active-set updates

	For $(x, s)_j$ initially free	size of C	value of t
1.	free $\rightarrow$ fixed	increases by 1	no change
2.	free $\rightarrow$ fixed $\rightarrow t$	no change	$t \leftarrow j$
3.	free $\rightarrow$ fixed $\rightarrow t \rightarrow$ free	decreases by 1	$t \leftarrow 0$
	For $(x, s)_j$ initially fixed		
4.	fixed $\rightarrow t$	no change	$t \leftarrow j$
5.	fixed $\rightarrow t \rightarrow$ free	increases by 1	$t \leftarrow 0$
6.	fixed $\rightarrow t \rightarrow$ free $\rightarrow$ fixed	decreases by 1	no change

If $(x, s)_j$ hits its bound, and $j \neq t$, then a free variable is becoming fixed. If the variable was originally fixed (6), then the row and column associated with freeing the j-th variable can be deleted from C. If the j-th variable was originally free (1), then a row and column must be added to C. This corresponds to expanding V and D as given in (4.40), where, if the r-th free variable is being fixed, $v = e_r$, $d = 0$ and $\delta = 0$. In either case, there is no change to t.

At a stationary point, if $(x, s)_j$ is to be freed, it is first held as the next variable to be deleted (*fixed* $\rightarrow t$, (2, 4)). If the current $t = 0$, as it is initially and after it has been reset, there is no change to C. Otherwise, there must be a corresponding $t \rightarrow$ *free* change to first delete the current t-th variable. Depending on the status of the t-th variable at (x_0, s_0), the size of C either increases (5) or decreases (3) by an associated row and column. For an increase in the size of C, the extensions to V and D are given by by,

$$v = \begin{pmatrix} (h_t)_{FR} \\ a_t \end{pmatrix}, \tag{4.45}$$

where a_t is the t-th column of $(\begin{array}{cc} A & -I \end{array})$, and $(h_t)_{FR}$ is defined by the free variables at the initial point (x_0, s_0). The vector d is given by

$$d = (h_t)_{FR}, \tag{4.46}$$

where $(h_t)_{FR}$ is defined by the variables freed during iterations $1, \ldots, j-1$. The scalar $\delta = h_{tt}$. In all cases, the value of t is updated to j, $t \leftarrow j$.

The update to C for the addition of a row and column is given by (4.42) and (4.43). The updated LU factors of C will be of the form

$$\bar{L} = \begin{pmatrix} L & 0 \\ l^T & 1 \end{pmatrix} \quad \text{and} \quad \bar{U} = \begin{pmatrix} U & u \\ 0 & \rho \end{pmatrix}, \tag{4.47}$$

where here l and u are the new row of L and column of U respectively. Setting $\bar{L}\bar{U} = \bar{C}$, substituting from (4.47) and comparing sides, indicates that l, u and ρ can be calculated by

$$\begin{aligned} Lu &= c, \\ Ul &= c, \\ \rho &= \tau - l^T u. \end{aligned} \tag{4.48}$$

Now consider the update to C that deletes a row and column. Let C be defined by

$$\begin{pmatrix} C_{11} & c_1 & C_{12} \\ c_1^T & \gamma & c_2^T \\ C_{12}^T & c_2 & C_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & & \\ l_1^T & 1 & \\ L_{21} & l_2 & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & u_1 & U_{12} \\ & \rho & u_2^T \\ & & U_{22} \end{pmatrix}, \tag{4.49}$$

and let the row and column to be deleted be associated with c_1, c_2 and γ. The updated LU factors of $\bar{C}$ are given by

$$\bar{C} = \begin{pmatrix} C_{11} & C_{12} \\ C_{12}^T & C_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & \bar{U}_{22} \end{pmatrix}, \tag{4.50}$$

where $\bar{U}_{22}$ satisfies

$$L_{22}\bar{U}_{22} = L_{22}U_{22} + l_2 u_2^T. \tag{4.51}$$

The implementation requires two extra vectors, $i_0 \in I\!R^{n+m}$ defined by

$$(i_0)_j = \begin{cases} +1 & \text{if} \quad (x,s)_j \quad \text{is free at} \quad (x_0, s_0), \\ -1 & \text{if} \quad (x,s)_j \quad \text{is fixed at} \quad (x_0, s_0), \end{cases}$$

and i_{sc} with

$$(i_{sc})_j = \begin{cases} +r & \text{if} \quad (x,s)_r \quad \text{was freed for column} \quad j \quad \text{of} \quad C, \\ -r & \text{if} \quad (x,s)_r \quad \text{was fixed for column} \quad j \quad \text{of} \quad C, \end{cases}$$

for $j = 1, \ldots, k$, where the dimension of C is $k \times k$. If n_c is the maximum dimension of C allowed, then $i_{sc} \in I\!R^{n_c}$.

When $(x,s)_r$ hits a bound, and C must be updated, a check can be done to see if $r = |(i_{sc})_j|$ for some j. If not, then a new row and column is added to C, and

$$(i_{sc})_{k+1} = -r.$$

Otherwise, the j-th row and column of C are deleted, and the j-th element of i_{sc} is deleted as well. The value of t remains unchanged.

If the current point is a subspace minimizer (i.e., $p = 0$), then a check is made to see if $t \neq 0$. If t is nonzero, then a check is first made to see if $t = |(i_{sc})_j|$ for some j. If $j \neq 0$ then the j-th row and column of C are deleted, as is the j-th element of i_{sc}. If t is not part of the Schur complement (i.e., $j = 0$), then a row and column are added to C, and

$$(i_{sc})_{k+1} = +t.$$

In both cases, t is temporarily reset to 0.

At a subspace minimizer, if $t = 0$ (including after the update to free the t-th variable), the value of t needs to be updated. If the r-th variable is chosen to be the new t-th variable, then $t \leftarrow r$. A check can be made to see if $r = |(i_{sc})_j|$ for some j. If $j \neq 0$, then the j-th element of i_{sc} can be ignored in extracting the solution of the linear systems. If $j = 0$, then i_{sc} is unaffected. There is no effect on C for setting the new t.

Then, i_0 and i_{sc} can be used to reconstruct the solution of

$$\begin{pmatrix} H & W^T \\ W & 0 \end{pmatrix} \begin{pmatrix} u \\ \mu \end{pmatrix} = \begin{pmatrix} 0 \\ e_t \end{pmatrix}$$

from (4.37). The initial values of u and μ can be set using i_0 and y_1, then while stepping through i_{sc} the appropriate changes can be made using elements of y_2 to overwrite elements of u and μ depending on if i_{sc} is positive or negative. If $(i_{sc})_j = t$, then the j-th element of q is ignored. A similar approach can be used to unscramble the solution of *System 2*.

4.5 Using an ℓ_2 Penalty Objective Function

For the SC method described in the previous sections, it was assumed that an initial subspace minimizer was available. Thus, a constraint was deleted at (x_0, s_0). Of course, in general this will not be true. A system of the form (4.23) can be solved for a given (x_0, s_0), and the solution will yield a search direction p_0 such that $(x_0, s_0) + p_0$ is a subspace minimizer. However, for problem (4.33) the point $(x_0, s_0) + p_0$ may not be feasible. In this section, a method using a composite objective function is derived to address the case when $(x_0, s_0) + p_0$ is not feasible.

A two-phase approach such as that used in SQOPT could be used, where the first phase minimizes the sum of infeasibilities. Since the first phase is a linear program, the solution is generally a vertex solution. For problems where the number of degrees of freedom is large, this can imply a large number of changes to the active set during the first subproblem. For an SC method this would be disastrous since the size of the dense LU factors of C usually increases for each change to the active set.

A method that avoids this disadvantage uses a composite objective function in which the objective function $\psi(x)$ is augmented by a penalty term involving the constraint violations. This composite objective is of the form

$$\underset{x}{\text{minimize}}\ \psi(x) + \rho\|v\|, \tag{4.52}$$

where $\|.\|$ is an appropriate norm, ρ is a large positive constant and v represents the constraint violations of the linear constraints and variable bounds at x. The approach developed in this section is to use the SC method to minimize the composite objective function subject to the original constraints. If $v = 0$ at some point, then the penalty term can be dropped. If the composite objective is minimized at a point where $v \neq 0$, then the original QP is infeasible or ρ must be increased.

4.5.1 Selecting the Penalty Norm

As mentioned in Section 1.1.4, the method of SNOPT uses a composite objective function when the QP subproblem is infeasible or unbounded. The algorithm enters *elastic* mode, where the objective is modified as in (4.52), using the ℓ_1 norm for the constraint violations.

The general form for the KKT system of the modified subproblem is given by

$$\begin{pmatrix} H & 0 & W^T \\ 0 & 0 & I \\ W & I & 0 \end{pmatrix} \begin{pmatrix} p \\ p_v \\ -\lambda \end{pmatrix} = - \begin{pmatrix} g \\ \rho e \\ 0 \end{pmatrix}, \tag{4.53}$$

where p_v is the search direction for the constraint violations. The form of the KKT matrix in (4.53) implies that the working set matrix W is still linearly independent. In the case of SNOPT, where the composite objective is used when the algorithm enters elastic mode, the working set is already linearly independent.

As a step is taken along the QP search directions derived from (4.53), constraints change from being violated to being on one of their bounds. As the step increases, a constraint that moved onto one of its bounds will become satisfied. Thus, it is not always necessary to compute a new search direction. Instead, the search can continue along the same search direction, with the previously violated constraint now satisfied. This is repeated until a satisfied constraint is about to go infeasible, or until the minimizer of the composite objective is found along the search direction. A change to the working set only occurs for the variable fixed at the last step. The other variables go from being violated to satisfied, but remain free.

For illustration, consider the simple example of

$$\underset{x}{\text{minimize}}\; c^T x + \frac{1}{2} x^T H x, \quad \text{subject to } x \geq b_l,$$

where $c^T = (1,1,1,1)$, $b_l = (0.8, 1.6, 2.4, 3.2)^T$, $\rho = 10$ and

$$H = 10^{-1} \times \begin{pmatrix} 4 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}.$$

At $x_0 = 0$, all the bounds are violated. Let the search direction $p = e$, where e is a vector of all ones. The plot of the original objective and the ℓ_1 penalty objective along p are shown in Figure 4.1. The *break points*, i.e., the points where the derivative is discontinuous, of the ℓ_1 penalty function along p are evident as each bound becomes satisfied. When the final bound on x_4 becomes satisfied, at $\alpha = 3.2$, the composite and original objective functions are the same. The ℓ_1 objective is minimized at the third break point, with $\alpha = 2.4$. The algorithm would continue the search along p up to this point, where the bound on x_3 would be added to the working set. Both x_1 and x_2 would remain free, however, their associated bound constraints would no longer be violated. The remaining variable, x_4, remains free with its bound still violated.

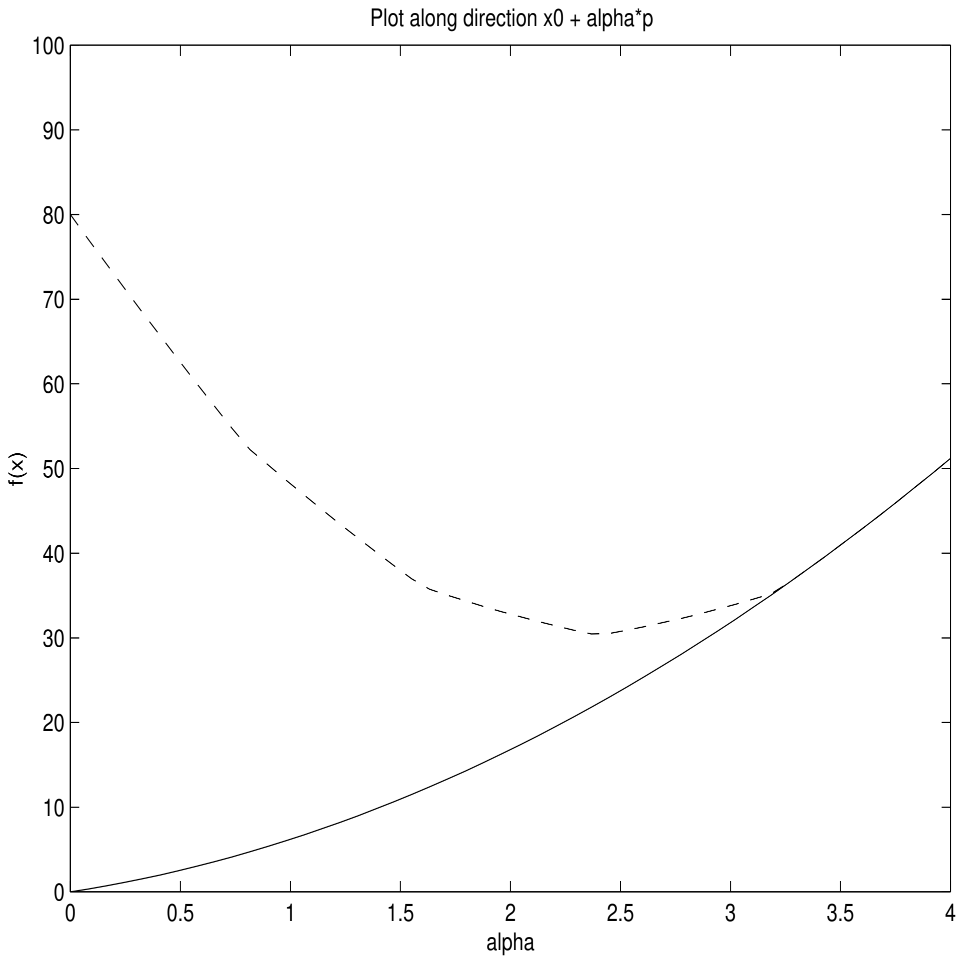

Figure 4.1: A plot of $q(x)$ (solid line), and $q(x) + \|v\|_1$ (dashed line) along the search direction p. Note that once $\alpha \geq 3.2$ all the bounds are satisfied and the two functions are the same. The penalty function is minimized where $\alpha = 2.4$, thus only the bound on $(x)_3$ is added to the working set.

For the SC method, a phase 1 approach can be used to find an initial feasible point, and a composite objective used for infeasible or unbounded problems. This two-phase approach can be defined with the phase 1 Hessian consisting of an identity matrix associated with the sum of infeasibilities, and the phase 2 Hessian being H. However, this strategy has two major disadvantages for a SC method. First, the transition from phase 1 to phase 2 requires the initial K_0 to be refactorized. Moreover, the phase 1 solution is often a vertex solution. Therefore, for problems with a large number of degrees of freedom, many working set changes are required to free variables from their bounds. This can cause the size of the Schur complement to grow very rapidly, forcing K_0 to be factorized many times.

To avoid the disadvantages of a two-phase approach on an SC method, a composite objective can be used initially. Thus, the algorithm starts in elastic mode, without the benefit of a previously determined linearly independent working set. The modified subproblem, using the ℓ_2 penalty term, is given by

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x + \frac{\rho}{2} \|v\|_2^2, \\ \text{subject to} & Ax - s = b, \\ & x \geq 0, \ s \geq 0, \ v \geq 0, \end{array} \tag{4.54}$$

where v represents the violations of the bounds on (x, s). The general form of the KKT system for problem (4.54) is given by

$$\begin{pmatrix} H & W^T \\ W & -\frac{1}{\rho} I \end{pmatrix} \begin{pmatrix} p \\ -\lambda \end{pmatrix} = - \begin{pmatrix} g \\ -v \end{pmatrix}. \tag{4.55}$$

Note that the search direction for the constraint violations is dependent on both v and λ (see Section 4.5.2). In addition, the constraints in the working set do not have to be linearly independent.

It can be shown for the ℓ_1 penalty function, that there exists a $\bar{\rho}$, such that for $\rho > \bar{\rho}$, the solution of the composite objective is a solution of the original problem. Thus, the ℓ_1 penalty function is often referred to as the "exact" penalty function. A disadvantage of the ℓ_2 penalty function is that the solutions with the penalty and original objectives are only the same as $\rho \to \infty$. Therefore, the penalty parameter must be made very large to get an accurate solution of the original problem. If the working set is not full rank, the KKT matrix becomes more ill-conditioned as $\rho \to \infty$.

Another disadvantage of using the ℓ_2 penalty function is that minimizing the composite objective function for a given search direction requires updates to the Schur complement. In SNOPT, when using the ℓ_1 penalty function, no working set updates are required at the intermediate break-points. For the SC method, with the ℓ_2 penalty function, even though the search direction remains the same, updates to the Schur complement are required as each break-point is encountered (see Section 4.5.6). On the other hand, the ℓ_2 penalty function is a piecewise quadratic, with continuous first derivatives. Thus, the minimizer along a given search direction will usually occur between break points.

A picture of the different objective functions for the previous simple example is given in Figure 4.2. The continuity of the first derivatives of the ℓ_2 penalty function along p are evident. The ℓ_2 objective is minimized between the second and third break points. The algorithm continues along p until this point point is reached, at which no change is required to the working set. Both x_1 and x_2 would remain free, however, their associated bound constraints would no longer be violated. An update to the Schur complement is required as each variable changes from being violated to being satisfied. The remaining variables, x_3 and x_4, remain free with their bounds violated.

4.5.2 Deriving the KKT System

The $(n+m)$-vector v of constraint violations is defined by

$$v = \begin{pmatrix} -I^x x \\ -I^s s \end{pmatrix}, \tag{4.56}$$

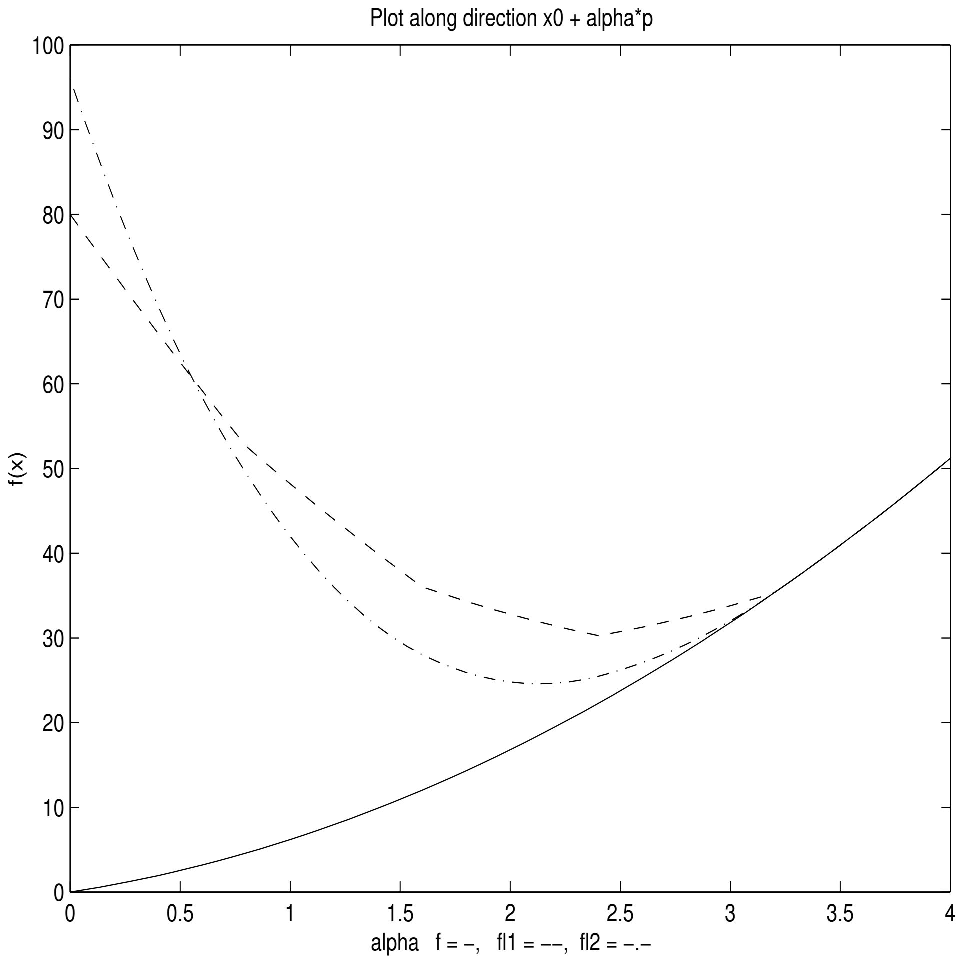

Figure 4.2: A plot of $q(x)$ (solid line), $q(x) + \|v\|_1$ (dashed line) and $q(x) + \|v\|_2^2$ (dash-dot line) along the search direction p. Note that once $\alpha \geq 3.2$ all the bounds are satisfied and all the functions are the same. The ℓ_2 penalty function is minimized between the second and third break points.

where I^x is a diagonal matrix whose entries are

$$I^x_{ii} = \begin{cases} 0 & \text{if} \quad (x)_i \geq 0, \\ 1 & \text{if} \quad (x)_i < 0. \end{cases}$$

The matrix I^s is similarly defined for the slack variables. Both I^x and I^s are symmetric and idempotent, thus the problem (4.54) can be stated more precisely as

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x + \frac{\rho}{2} x^T I^x x + \frac{\rho}{2} s^T I^s s \\ \text{subject to} & Ax - s = b, \; x \geq 0, \; s \geq 0. \end{array} \tag{4.57}$$

Let $\begin{pmatrix} A & -I \end{pmatrix} P = \begin{pmatrix} A_{FR} & A_{FX} & I^s_v \end{pmatrix}$, where P is a permutation matrix. The columns of A_{FR} correspond to variables such that $x_i \neq 0$ and $s_i > 0$. If $(x, s)_j = 0$ then the j-th column of $\begin{pmatrix} A & -I \end{pmatrix}$ is in A_{FX}. The matrix I^s_v consists of the nonzero columns of I^s corresponding to the violated slacks ($s_i < 0$). The working-set matrix W satisfies

$$WP = \begin{pmatrix} A_{FR} & A_{FX} & -I^s_v \\ 0 & I_{FX} & 0 \end{pmatrix},$$

where I_{FX} is an identity matrix with the same number of columns as A_{FX}. By an appropriate permutation, the KKT system for (4.57) is equivalent to

$$\begin{pmatrix} H_{FR} + \rho I^x_s & H_{OD} & 0 & A^T_{FR} & 0 \\ H^T_{OD} & H_{FX} & 0 & A^T_{FX} & I_{FX} \\ 0 & 0 & \rho I & (-I^s_v)^T & 0 \\ A_{FR} & A_{FX} & -I^s_v & 0 & 0 \\ 0 & I_{FX} & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} p_{FR} \\ p_{FX} \\ p_v \\ -\lambda_E \\ -\lambda_X \end{pmatrix} = - \begin{pmatrix} g_{FR} \\ g_{FX} \\ g_v \\ 0 \\ 0 \end{pmatrix}. \tag{4.58}$$

The vectors p_v and g_v are the search direction and objective gradient corresponding to the violations of the slack variables. The matrix I^x_s is defined similar to A_{FR} with columns from I^x and I^s. If $(x)_i \neq 0$ then the i-th column of I^x is in I^x_s, and if $(s)_j > 0$ then the j-th column of I^x_s is a zero column of appropriate dimension.

If the SC method is applied to (4.57), then the solution of *Systems 0*, *1* and *2*, and the updates to the Schur complement, must be defined for the KKT system (4.58).

4.5.3 Solving *System 0*

The system (4.58) can be reduced to a KKT system in the free variables given by

$$\begin{pmatrix} H_{FR} + \rho I_s^x & A_{FR}^T \\ A_{FR} & -\frac{1}{\rho} I^s \end{pmatrix} \begin{pmatrix} p_{FR} \\ -\lambda_E \end{pmatrix} = - \begin{pmatrix} g_{FR} \\ I^s s \end{pmatrix}. \tag{4.59}$$

When the system is solved at the initial point for *System 0*, the matrix in (4.59) is the initial K_0 for the SC method. If p_{FR} and λ_E are the solution of (4.59) at the initial point, then the required λ_0 can be recovered from λ_E and

$$\lambda_X = g_{FX} + H_{OD}^T p_{FR} - A_{FX}^T \lambda_E.$$

In addition, p_0 can be obtained by permuting p_{FR}, p_{FX} and p_v, where $p_{FX} = 0$ and

$$p_v = -(I_v^s)^T (s + \frac{1}{\rho} \lambda_E).$$

4.5.4 Solving *System 1*

The solution of *System 1* is derived from the KKT based system (4.24). From the KKT system (4.58), the solution of *System 1* for problem (4.57) can be calculated by first solving

$$\begin{pmatrix} H_{FR} + \rho I_s^x & A_{FR}^T \\ A_{FR} & -\frac{1}{\rho} I^s \end{pmatrix} \begin{pmatrix} u_{FR} \\ \mu_E \end{pmatrix} = - \begin{pmatrix} (h_t)_{FR} \\ a_t \end{pmatrix}, \tag{4.60}$$

for u_{FR} and μ_E. Then u_v, u_{FX}, $(u)_t$, μ_X and $(\mu)_t$ are given by

$$
\begin{aligned}
u_v &= -\tfrac{1}{\rho}(I_v^s)^T \mu_E, \\
\begin{pmatrix} u_{FX} \\ (u)_t \end{pmatrix} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \\
\begin{pmatrix} \mu_X \\ (\mu)_t \end{pmatrix} &= -\begin{pmatrix} H_{OD}^T \\ (h_t)_{FR}^T \end{pmatrix} u_{FR} - \begin{pmatrix} (h_t)_{FX} \\ \widehat{h}_{tt} \end{pmatrix} - \begin{pmatrix} A_{FX}^T \\ a_t^T \end{pmatrix} \mu_E.
\end{aligned}
\tag{4.61}
$$

where the definition of $\widehat{h}_{tt}$ depends on if the t-th variable is a slack variable ($\widehat{h}_{tt} = \rho$ or 0), or an x variable ($\widehat{h}_{tt} = h_{tt} + \rho I_{tt}^x$).

The reduced KKT system is not reformulated for each iteration. A system of the form (4.37) is solved at each iteration, where K_0 is the matrix in (4.60) at the initial point (x_0, s_0). For the SC method there were only two cases for updates to the Schur complement. Either a fixed variable was being freed, or a free variable was being fixed. The same basic two cases exist for the modified problem, however, the updates are dependent on whether the affected variable is an x variable or a slack variable. In the case of the slack variables, it also depends on if the free variable is violated or satisfied, or if the fixed variable is becoming violated or satisfied. The three distinct cases resulting from changes to the active set can be explicitly defined by

CASE 1 – A fixed $(x)_j$ is freed, becoming satisfied or violated; or a fixed $(s)_j$ is freed, becoming satisfied.

CASE 2 – A free, satisfied or violated, $(x)_j$ is fixed; or a free satisfied $(s)_j$ is fixed.

CASE 3 – A fixed $(s)_j$ is freed, becoming violated; or a free violated $(s)_j$ is fixed.

The updates to Schur complement for each of the three cases are given in Section 4.5.6.

The vectors y_1 and f_1 in (4.37) for *System 1* are defined by $y_1 = [u_{FR} \quad \mu_E]^T$ and $f_1 = [(h_t)_{FR} \quad a_t]^T$, where u_{FR}, μ_E and $(h_t)_{FR}$ are all defined at the initial point. The elements of y_2 and f_2 are defined for the k-th iteration by

$$(y_2)_k = \begin{cases} (u)_j & \text{for CASE 1,} \\ (\mu)_j & \text{for CASE 2,} \\ (y_2)_j & \text{for CASE 3,} \end{cases}$$

and

$$(f_2)_k = \begin{cases} h_{jt} & \text{for CASE 1,} \\ 0 & \text{for CASE 2 and CASE 3.} \end{cases}$$

The element $(y_2)_j$ is not part of the current u_{FR} or μ_E, but is a working variable needed to define a rank-one update to I^s in SC form (see Section 4.5.6).

4.5.5 Solving *System 2*

The solution of *System 2* for problem (4.57) requires solving (4.25) with the matrix in (4.58). By appropriate permutation, the system is equivalent to

$$\begin{pmatrix} H_{FR} + \rho I_s^x & H_{OD} & (h_t)_{FR} & 0 & A_{FR}^T & 0 & 0 \\ H_{OD}^T & H_{FX} & (h_t)_{FX} & 0 & A_{FX}^T & I_{FX} & 0 \\ (h_t)_{FR}^T & (h_t)_{FX}^T & \widehat{h}_{tt} & 0 & a_t^T & 0 & 1 \\ 0 & 0 & 0 & \rho I & (-I_v^s)^T & 0 & 0 \\ A_{FR} & A_{FX} & a_t & -I_v^s & 0 & 0 & 0 \\ 0 & I_{FX} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} z_{FR} \\ z_{FX} \\ (z)_t \\ z_v \\ \eta_E \\ \eta_X \\ (\eta)_t \end{pmatrix} = \begin{pmatrix} e_{FR} \\ 0 \\ 0 \\ e_v \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

The right-hand side of this system is always a unit vector corresponding to the bound constraint of the free variable that has stepped onto its bound. Thus, if a violated slack variable has stepped onto its bound, e_v is a unit vector and $e_{FR} = 0$. Otherwise, $e_v = 0$ and e_{FR} is a unit vector.

The associated reduced system for z_{FR} and η_E is

$$\begin{pmatrix} H_{FR} + \rho I_s^x & A_{FR}^T \\ A_{FR} & -\frac{1}{\rho} I^s \end{pmatrix} \begin{pmatrix} z_{FR} \\ \eta_E \end{pmatrix} = \begin{pmatrix} e_{FR} \\ \frac{1}{\rho} e_v \end{pmatrix}, \tag{4.62}$$

and z_v, z_{FX}, $(z)_t$, η_X and $(\eta)_t$ are computed via

$$\begin{aligned} z_v &= \tfrac{1}{\rho}(\ (I_v^s)^T \eta_E - e_v\ ) \\ \begin{pmatrix} z_{FX} \\ (z)_t \end{pmatrix} &= \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \\ \begin{pmatrix} \eta_X \\ (\eta)_t \end{pmatrix} &= -\begin{pmatrix} H_{OD}^T \\ (h_t)_{FR}^T \end{pmatrix} z_{FR} - \begin{pmatrix} A_{FX}^T \\ a_t^T \end{pmatrix} \eta_E. \end{aligned} \tag{4.63}$$

The algorithm starts with (4.62) at the initial point, and a bordered system of the form (4.37) is used to solve subsequent systems. The vector $y_1 = [z_{FR} \quad \eta_E]^T$ at (x_0, s_0), and the elements of y_2 for the k-th iteration defined by

$$(y_2)_k = \begin{cases} (z)_j & \text{for CASE 1,} \\ (\eta)_j & \text{for CASE 2,} \\ (y_2)_j & \text{for CASE 3.} \end{cases}$$

The right-hand side $[f_1 \quad f_2]^T$ is always a unit vector, (or a multiple of a unit vector, i.e. $\frac{1}{\rho} e_v$), corresponding to the variable that has hit its bound.

4.5.6 Updating the Schur Complement

When a variable changes state there are now five possible changes:

$$\begin{aligned} \text{violated} &\rightarrow \text{fixed,} \\ \text{satisfied} &\rightarrow \text{fixed,} \\ \text{fixed} &\rightarrow t, \\ t &\rightarrow \text{violated,} \\ t &\rightarrow \text{satisfied.} \end{aligned}$$

The notation $\rightarrow t$ implies that $(x,s)_j$ becomes the t-th variable, and $t \rightarrow$ implies that the t-th variable is changing state. The notation *violated* represents a free variable that is outside its bounds, and similarly *satisfied* represents a free variable within its bounds.

Table 4.2 corresponds to the effect on the dimension of C, and value of t, for each of the relative states to K_0. Initially $t = 0$, so all variables are either violated, satisfied or fixed.

Table 4.2: The change in value of t and size of C for active-set updates

	For $(x,s)_j$ initially violated	size of C	value of t
1.	violated $\rightarrow$ fixed	increases by 1	no change
2.	violated $\rightarrow$ fixed $\rightarrow t$	no change	$t \leftarrow j$
3.	violated $\rightarrow$ fixed $\rightarrow t \rightarrow$ violated	decreases by 1	$t \leftarrow 0$
4.	violated $\rightarrow$ fixed $\rightarrow t \rightarrow$ satisfied	increases by 1	$t \leftarrow 0$
5.	violated $\rightarrow$ fixed $\rightarrow t \rightarrow$ satisfied $\rightarrow$ fixed	decreases by 1	no change
	For $(x,s)_j$ initially satisfied		
6.	satisfied $\rightarrow$ fixed	increases by 1	no change
7.	satisfied $\rightarrow$ fixed $\rightarrow t$	no change	$t \leftarrow j$
8.	satisfied $\rightarrow$ fixed $\rightarrow t \rightarrow$ satisfied	decreases by 1	$t \leftarrow 0$
9.	satisfied $\rightarrow$ fixed $\rightarrow t \rightarrow$ violated	increases by 1	$t \leftarrow 0$
10.	satisfied $\rightarrow$ fixed $\rightarrow t \rightarrow$ violated $\rightarrow$ fixed	decreases by 1	no change
	For $(x,s)_j$ initially fixed		
11.	fixed $\rightarrow t$	no change	$t \leftarrow j$
12.	fixed $\rightarrow t \rightarrow$ violated	increases by 1	$t \leftarrow 0$
13.	fixed $\rightarrow t \rightarrow$ satisfied	increases by 1	$t \leftarrow 0$
14.	fixed $\rightarrow t \rightarrow$ violated $\rightarrow$ fixed	decreases by 1	no change
15.	fixed $\rightarrow t \rightarrow$ satisfied $\rightarrow$ fixed	decreases by 1	no change

If $(x,s)_j$ hits its bound, and $j \neq t$, then either a *violated* or *satisfied* free variable is becoming fixed. If the variable was previously fixed (5, 10, 14, 15), then the row and column associated with freeing the j-th variable can be deleted from C. If the j-th variable was originally free, then a row and column must be added to C. For a *satisfied* variable (6), or a *violated* $(x)_j$ (1, CASE 2), the corresponding expansion to V and D given by (4.40) is defined by $v = e_r$, $d = 0$ and $\delta = 0$, where the r-th variable is being freed. If $(x, s)_j$ corresponds to a *violated* slack that is becoming fixed (1, CASE 3), then the update to V and D is given by

$$v = \frac{1}{\sqrt{\rho}} e_r, \tag{4.64}$$

with $d = 0$ and $\delta = -1$. For all cases of a free variable becoming fixed, there is no change to t.

As before, if $(x, s)_j$ is to be freed at a stationary point, it is first held as the next variable to be deleted (*fixed* $\rightarrow t$, (2, 7, 11)). If the current $t = 0$, there is again no change to C. Otherwise, there must be a corresponding $t \rightarrow$ *violated* (3, 9, 12), or $t \rightarrow$ *satisfied* (4, 8, 13), to first delete the current t-th variable. If the t-th variable is returning to the state it had at (x_0, s_0), then C decreases in size by a row and column (3, 8). However, if the t-th variable was originally fixed (12, 13), or if the new state differs from the state at (x_0, s_0) (4, 9), then C increases by a row and column. The definition of v, d and δ for the update depend on if the t-th variable is in x or s, and if it is becoming *satisfied* or *violated*.

If the t-variable is becoming satisfied, then v and d are given by (4.45) and (4.46) respectively, and the scalar is defined by $\delta = h_{tt}$ as before. If $(x, s)_t = (x)_t$, and $(x)_t$ is becoming violated, then v and d are also defined by (4.45) and (4.46), but $\delta = h_{tt} + \rho$. Finally, if the t-th variable is a slack that is becoming violated, then $d = 0$, v is given by (4.64) and $\delta = 1$. The value of t is updated to j, $t \leftarrow j$, for all cases.

4.5.7 Algorithmic Details

The updates for adding and deleting rows and columns of C are the same as those given in Sections 4.4.1–4.4.2.

For the ℓ_2 formulation, the implementation requires three extra vectors instead of the two needed in Section 4.4.2. These are defined as $i_0 \in I\!R^{n+m}$ with

$$(i_0)_j = \begin{cases} +1 & \text{if} \quad (x,s)_j \quad \text{is satisfied at} \quad (x_0, s_0), \\ 0 & \text{if} \quad (x,s)_j \quad \text{is fixed at} \quad (x_0, s_0), \\ -1 & \text{if} \quad (x,s)_j \quad \text{is violated at} \quad (x_0, s_0), \end{cases}$$

and i^{sc}_{FR} with

$$(i^{sc}_{FR})_j = \begin{cases} +r & \text{if} \quad (x,s)_r \quad \text{was becoming satisfied for column} \quad j \quad \text{of} \quad C, \\ -r & \text{if} \quad (x,s)_r \quad \text{was becoming violated for column} \quad j \quad \text{of} \quad C, \\ 0 & \text{otherwise,} \end{cases}$$

and i^{sc}_{FX} with

$$(i^{sc}_{FX})_j = \begin{cases} +r & \text{if} \quad (x,s)_r \quad \text{was fixed from being satisfied for} \\ & \text{column} \quad j \quad \text{of} \quad C, \\ -r & \text{if} \quad (x,s)_r \quad \text{was fixed from being violated for} \\ & \text{column} \quad j \quad \text{of} \quad C, \\ 0 & \text{otherwise,} \end{cases}$$

for $j = 1, \ldots, k$, where the dimension of C is $k \times k$. If n_c is the maximum dimension of C allowed, then $i^{sc}_{FR} \in I\!R^{n_c}$ and $i^{sc}_{FX} \in I\!R^{n_c}$. Here, the subscripts $_{FR}$ and $_{FX}$ do not refer to a partition of the same vector, instead i^{sc}_{FR} and i^{sc}_{FX} are distinct vectors.

To update C when $(x,s)_r$ hits a bound ($r \neq t$), a check is made to see if $r = (i^{sc}_{FR})_j$ for some j. If not, then a new row and column is added to C, and

$$\begin{aligned} (i^{sc}_{FR})_{k+1} &= 0, \\ (i^{sc}_{FX})_{k+1} &= \begin{cases} +r & \text{if} \quad (x,s)_r \quad \text{was satisfied,} \\ -r & \text{if} \quad (x,s)_r \quad \text{was violated.} \end{cases} \end{aligned}$$

As mentioned above, the row and column added depend on if $(x,s)_r$ was a violated slack ($r > n$ and $(x,s)_r < 0$). This case is part of CASE 3 defined in Section 4.5.4, therefore q_{k+1} is a dummy variable and will be ignored in extracting the solution of the linear system. If $j \neq 0$, the j-th row and column of C are deleted, and the j-th element of i^{sc}_{FR} and i^{sc}_{FX} are deleted as well. The value of t remains unchanged.

If the current point is a subspace minimizer, a check is made to see if $t \neq 0$. If t is nonzero, a check is made to see if $t = |(i_{FX}^{sc})_j|$ for some j. If $j = 0$, or if $j \neq 0$ and $(i_{FX}^{sc})_j$ and $(i_0)_t$ are of opposite sign, then a row and column is added to C, and

$$\begin{aligned}(i_{FR}^{sc})_{k+1} &= \begin{cases} +t & \text{if} \quad (x,s)_t \quad \text{is becoming satisfied,} \\ -t & \text{if} \quad (x,s)_t \quad \text{is becoming violated.} \end{cases} \\ (i_{FR}^{sc})_{k+1} &= 0, \end{aligned}$$

The row and column added to C will depend on if t is a slack ($t > n$) that is becoming violated. This case also corresponds to CASE 3 defined in Section 4.5.4, thus q_{k+1} is a dummy variable and will be ignored. If $j \neq 0$, and $(i_{FX}^{sc})_j$ and $(i_0)_t$ have the same sign, the j-th row and column of C can be deleted, as well as the j-th elements of i_{FX}^{sc} and i_{FR}^{sc}.

When $t = 0$ at a subspace minimizer (including after the update to free the t-th variable), the value of t is reset to r ($t \leftarrow r$). A check is made to see if $r = |(i_{FX}^{sc})_j|$ for some j. If $j \neq 0$ then $(i_{FX}^{sc})_j$ is ignored in extracting the solution of the linear systems. Otherwise, i_{FX}^{sc} is unaffected. There is no update to C for reseting the value of t.

To extract the correct u and μ, the initial values can be set using i_0 and y_1. By stepping through i_{FX}^{sc}, the appropriate changes can be made to μ. If $i_{FX}^{sc} = 0$, or $(i_{FX}^{sc})_k < 0$ and $|(i_{FX}^{sc})_k| > n$, then $(y_2)_k$ is ignored in updating μ. Similarly, u can be updated by stepping through i_{FR}^{sc}. If $(i_{FR}^{sc})_k = 0$, or $(i_{FR}^{sc})_k < 0$ and $|(i_{FR}^{sc})_k| > n$, then $(y_2)_k$ is ignored. Additionally, if $|(i_{FR}^{sc})_k| = t$, then the k-th element of q is ignored. A similar approach can used to unscramble the solution of *System 2*.

Bibliography

[1] J. Bisschop and A. Meeraus. Matrix augmentation and partitioning in the updating of the basis inverse. *Math. Prog.*, 13:241–254, 1977.

[2] J. Bisschop and A. Meeraus. Matrix augmentation and structure preservation in linearly constrained control problems. *Math. Prog.*, 18:7–15, 1980.

[3] Å. Björck and T. Elfving. Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations. *Nordisk Tidskr. Informationsbehandling (BIT)*, 19:145–163, 1979.

[4] I. Bongartz, A. R. Conn, N. I. M. Gould, and P. L. Toint. CUTE: Constrained and unconstrained testing environment. Report 93/10, Département de Mathématique, Facultés Universitaires de Namur, 1993.

[5] A. Buckley and A. LeNir. QN-like variable storage conjugate gradients. *Math. Prog.*, 27:155–175, 1983.

[6] A. Buckley and A. LeNir. BBVSCG–a variable storage algorithm for function minimization. *ACM Trans. Math. Software*, 11:103–119, 1985.

[7] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM J. Sci. Comput.*, 20:33–61, 1998.

[8] A. R. Conn. Constrained optimization using a nondifferentiable penalty function. *SIAM J. Numer. Anal.*, 10:760–779, 1973.

[9] S. K. Eldersveld. *Large-scale sequential quadratic programming algorithms.* PhD thesis, Department of Operations Research, Stanford University, Stanford, CA, 1991.

[10] R. Fletcher. *Practical Methods of Optimization.* Volume 1: Unconstrained Optimization. John Wiley and Sons, New York and Toronto, 1980.

[11] R. Fletcher. An ℓ_1 penalty method for nonlinear constraints. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel, editors, *Numerical Optimization 1984*, pages 26–40, Philadelphia, 1985. SIAM.

[12] J. C. Gilbert and C. Lemaréchal. Some numerical experiments with variable-storage quasi-Newton algorithms. *Math. Prog.*, pages 407–435, 1989.

[13] P. E. Gill and W. Murray. Numerically stable methods for quadratic programming. *Math. Prog.*, 14:349–372, 1978.

[14] P. E. Gill and W. Murray. Conjugate-gradient methods for large-scale nonlinear optimization. Report SOL 79-15, Department of Operations Research, Stanford University, Stanford, CA, 1979.

[15] P. E. Gill, W. Murray, and M. A. Saunders. SQOPT: An algorithm for large-scale quadratic programming. To appear.

[16] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. Numerical Analysis Report 97-2, Department of Mathematics, University of California, San Diego, La Jolla, CA, 1997.

[17] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Sparse matrix methods in optimization. *SIAM J. Sci. and Statist. Comput.*, 5:562–589, 1984.

[18] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. 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, CA, 1986.

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

[20] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A Schur-complement method for sparse quadratic programming. In M. G. Cox and S. J. Hammarling, editors, *Reliable Numerical Computation*, pages 113–138. Oxford University Press, 1990.

[21] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Inertia-controlling methods for general quadratic programming. *SIAM Review*, 33:1–36, 1991.

[22] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Some theoretical properties of an augmented Lagrangian merit function. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 101–128. North Holland, North Holland, 1992.

[23] G. H. Golub and W. Kahan. Calculating the singular values and pseudoinverse of a matrix. *SIAM J. Numer. Anal.*, 2:205–224, 1965.

[24] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, second edition, 1989. ISBN 0-8018-5414-8.

[25] M. R. Hestenes. *Conjugate-Direction Methods in Optimization*. Springer-Verlag, Berlin, Heidelberg and New York, 1980.

[26] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952.

[27] A. Jennings. Influence of the eigenvalue spectrum on the convergence rate of the conjugate gradient method. *J. Inst. Math. Applics.*, 20:61–72, 1993.

[28] D. G. Luenberger. *Introduction to Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, New York, 1973.

[29] J. L. Morales and J. Nocedal. Automatic preconditioning by limited memory quasi-newton updating. To appear.

[30] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Math. Prog.*, 14:41–72, 1978.

[31] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[32] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software*, 8(1):43–71, 1982.

[33] S. M. Robinson. A quadratically-convergent algorithm for general nonlinear programming problems. *Math. Prog.*, 3:145–156, 1972.

[34] G. Van der Hoek. Asymptotic properties of reduction methods applying linearly equality constrained reduced problems. *Math. Prog. Study*, 16:162–189, 1982.