A REGULARIZED ACTIVE-SET METHOD FOR SPARSE CONVEX QUADRATIC PROGRAMMING

A DISSERTATION
SUBMITTED TO THE INSTITUTE FOR
COMPUTATIONAL AND MATHEMATICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Christopher M. Maes
November 2010

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

(Michael Saunders) Principal Adviser

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

(Walter Murray)

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

(Philip Gill)

Approved for the University Committee on Graduate Studies

Abstract

An active-set algorithm is developed for solving convex quadratic programs (QPs). The algorithm employs primal regularization within a bound-constrained augmented Lagrangian method. This leads to a sequence of QP subproblems that are feasible and strictly convex, and whose KKT systems are guaranteed to be nonsingular for any active set.

A simplified, single-phase algorithm becomes possible for each QP subproblem. There is no need to control the inertia of the KKT system defining each search direction, and a simple step-length procedure may be used without risk of cycling in the presence of degeneracy. Since all KKT systems are nonsingular, they can be factored with a variety of sparse direct linear solvers. Block-LU updates of the KKT factors allow for active-set changes.

The principal benefit of primal and dual regularization is that warm starts are possible from any given active set. This is vital inside sequential quadratic programming (SQP) methods for nonlinear optimization, such as the SNOPT solver.

The method provides a reliable approach to solving sparse generalized least-squares problems. Ordinary least-squares problems with Tikhonov regularization and bounds can be solved as a single QP subproblem.

The algorithm is implemented as the QPBLUR solver (MATLAB and Fortran 95 versions) and the Fortran version has been integrated into SNOPT. The performance of QPBLUR is evaluated on a test set of large convex QPs, and on the sequences of QPs arising from SNOPT's SQP method.

To my grandparents Donald and Terrill Miller

Acknowledgements

Michael Saunders has been an extraordinary thesis advisor and a great friend. His kindness and generosity are truly unmatched. I am thankful for the tremendous amount of time he devoted to me and this research. I will always have fond memories of late nights together spent debugging QPBLUR.

I thank the members of my defense committee: Stephen Boyd, Yinyu Ye, Philip Gill and Walter Murray. I thank Stephen Boyd for serving as chair. He is an excellent teacher and a great evangelist for convex optimization. Thanks to Yinyu Ye for being a valuable mentor. I had the pleasure of working with Yinyu on several research projects during my time at Stanford. His enthusiasm for optimization is infectious. Thanks to Philip Gill for traveling from San Diego for my defense, and for his valuable comments on my thesis. He is the definitive expert on QP. I thank Walter Murray for sparking my interest in optimization. Walter was a careful reader of this thesis. His comments and suggestions made it much better. I also thank Walter for the valuable advice he gave me (and countless other ICME students) during his years as ICME's director of student affairs.

This thesis could not have begun without Hanh Huynh's work on QPBLU. Many thanks to Hanh for the software that performs block-LU updates and interfaces with the sparse direct solvers. I thank Elizabeth Wong for her help with the CUTEr interface.

I am grateful for the support of several other mentors. Richard Cottle and Robert Entriken mentored me on a summer research project for EPRI and encouraged me for the rest of my graduate career. Gene Golub welcomed me to the linear algebra community. I enjoyed being a part of his Monday lunches and many Chinese-food dinners. His passing left a void at Stanford and beyond.

I am thankful for the friendships of fellow students. I thank Les Fletcher for being my student mentor, David Gleich for being a wonderful roommate and a close friend, Laura Bofferding for her friendship and amazing popcorn, Michael Lesnick and Thomas Callaghan for being my quals companions, and John Bodley, Michael Atkinson, and Sean Kamkar for being great hiking buddies. Thanks to other ICME friends: Andrew Bradley, Christine Desmarais, Nick Henderson, Arik Motskin, Nick West, and many more.

Kindest thanks to Nicole, Kiana, Jane and Reza Taheri, who became a second family to me.

I thank all the members of Fort Awesome. Special thanks to Shervin Fatehi, May Lim, and George Waksman, who brought a part of Fort Awesome from MIT to the Bay Area. I thank Jen Mitchel and Jim Paris for their friendship, especially during the last few months of my thesis work.

Finally, I thank my grandparents Donald and Terrill Miller for all their love and support through the years.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Convex quadratic programming formulation

A quadratic program (QP) is a special type of optimization problem in which a quadratic objective function is minimized or maximized subject to linear inequality constraints. QPs may be stated in many equivalent forms and we consider several different formulations in this thesis. We define a QP in the most general form to be

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x + \tfrac{1}{2} x^T H x \\ \text{subject to} \quad & \ell \leq \begin{pmatrix} x \\ Ax \end{pmatrix} \leq u, \end{aligned} \tag{1.1}$$

where $x \in \mathbb{R}^n$ is the vector of decision variables, $H \in \mathbb{R}^{n \times n}$ is the symmetric Hessian matrix, $A \in \mathbb{R}^{m \times n}$ is the linear constraint matrix, $c \in \mathbb{R}^n$ is the linear objective vector, $\ell \in \mathbb{R}^{m+n}$ is a vector of lower bounds, and $u \in \mathbb{R}^{m+n}$ is a vector of upper bounds. We require $\ell \leq u$, but ℓ_i and u_i may be taken as $-\infty$ or ∞ if no bounds are present, and we can have $\ell_i = u_i$ for equality constraints. This thesis concerns a method for solving QPs when H and A are sparse. That is, most of their entries are zero.

The convexity of the quadratic objective function plays a defining role in the computational complexity of quadratic programming methods. We say that a QP is *convex* if the objective $\phi(x) = c^T x + \frac{1}{2} x^T H x$ is a convex function. The objective is convex when H is positive semidefinite (with a zero eigenvalue) and *strictly convex* when H is positive definite. When $\phi(x)$ is convex, a local optimum of the QP is also a global optimum. When H is indefinite or negative definite, $\phi(x)$ is nonconvex and locally optimal points may not be globally optimal solutions. Polynomial-time algorithms exist for solving convex QPs [64, 114]. Finding the global optimum of a nonconvex QP is NP-hard [93, 107]. Furthermore, finding a local minimizer of a nonconvex QP is NP-hard [81]. Under certain circumstances, checking local optimality for a feasible point of a nonconvex QP is NP-hard [86]. In this thesis we consider only convex QPs. Our motivating application is sequential quadratic programming (SQP) methods for large-scale nonlinear optimization. These methods obtain search

directions by solving a sequence of convex QPs that become increasingly similar.

1.2 Optimality conditions

With the introduction of slack variables $s \in \mathbb{R}^m$, problem (1.1) becomes

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x \\ \text{subject to} & Ax - s = 0 \\ & \ell \leq (x, s) \leq u. \end{array} \tag{1.2}$$

Let $\phi(x, s)$ denote the objective of problem (1.2).

The Karush-Kuhn-Tucker (KKT) optimality conditions define the necessary and sufficient conditions for a point (x, s) to be an optimal solution to (1.2) (assuming H is positive semidefinite):

$$\begin{pmatrix} A & -I \end{pmatrix} \begin{pmatrix} x \\ s \end{pmatrix} = 0, \quad \ell \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u \tag{1.3}$$

$$\nabla \phi(x, s) = \begin{pmatrix} A^T \\ -I \end{pmatrix} y + z_l - z_u \tag{1.4}$$

$$\left(\begin{pmatrix} x \\ s \end{pmatrix} - \ell \right)^T z_l = 0, \quad \left(u - \begin{pmatrix} x \\ s \end{pmatrix} \right)^T z_u = 0, \quad z_l, z_u \geq 0. \tag{1.5}$$

Here $y \in \mathbb{R}^m$ are the Lagrange multipliers associated with the equality constraints and $z_l, z_u \in \mathbb{R}^{n+m}$ are the multipliers associated with the inequality constraints.

1.3 Applications of quadratic programming

QPs arise naturally in a variety of applications, including portfolio analysis [75], support vector machines [61], structural analysis [67], VLSI design [62], optimal control [68], finite-impulse-response filter design [66], optimal power flow [101], and economic dispatch [100]. The online bibliography of Gould and Toint [54, 55] contains references to over 400 applications of quadratic programming. In many applications, such as model predictive control and classification using support vector machines, an approximate solution may be known in advance.

Quadratic programming also plays an important role in methods for solving nonlinear optimization problems of the form

$$\begin{array}{ll} \text{(locally) } \underset{x}{\text{minimize}} & \varphi(x) \\ \text{subject to} & \ell \leq \begin{pmatrix} x \\ c(x) \\ Ax \end{pmatrix} \leq u, \end{array} \tag{1.6}$$

where the objective $\varphi(x)$ and the constraints $c(x)$ are smooth, but otherwise general, nonlinear functions. SQP methods solve a sequence of QPs to obtain a solution to problem (1.6). Each QP in the sequence is related to the previous QP. Thus, it is important that a quadratic programming method used inside an SQP method be able to take advantage of information obtained from the solution of the current QP to solve the next QP. Methods that are able to take advantage of such information are called *warm starting*.

1.4 Quadratic programming methods

Quadratic programming methods can be roughly divided into two categories: interior-point and active-set. On large sparse QPs the majority of work for both methods occurs in solving large sparse systems of linear equations to compute search directions. These systems are called saddle-point or KKT systems, because they correspond to the KKT optimality conditions (1.3)–(1.4). KKT matrices take the form

$$\begin{pmatrix} H_k & A_k^T \\ A_k & \end{pmatrix},$$

where H_k is a symmetric submatrix of H and A_k is a submatrix of A.

Primal-dual interior-point methods maintain an approximate solution that lies strictly inside the feasible region defined by the inequality constraints. These methods attempt to follow a central path from the current point to the optimal solution (which may lie on the boundary of the feasible region). At each iteration a KKT system involving *all* of the decision variables is solved ($H_k = H + D_x, A_k = A$, where D_x is a diagonal matrix that changes from iteration to iteration). The sparsity structure of the KKT matrix is fixed (although the numerical values vary) from iteration to iteration. Each iteration requires a new factorization ($O((n+m)^3)$ in the dense case) for the linear solve, but interior-point methods generally perform very few iterations regardless of problem size.

Active-set methods distinguish between inequality constraints that are satisfied exactly and those that are not. A constraint of the form $a^T x \geq \beta$ is said to be *active* if $a^T x = \beta$, *inactive* if $a^T x > \beta$, and *violated* if $a^T x < \beta$. Active-set methods are motivated by the fact that if the set of active and inactive inequality constraints at an optimal solution were known *a priori*, a solution could be found by solving a simpler equality constrained QP. Active-set methods make a prediction of the set of active and inactive constraints at an optimal solution. This prediction is usually wrong and thus these methods contain procedures for testing and altering the current prediction. At each iteration an active-set method solves a KKT system defined by the active constraints. Consequently, when a significant number of constraints are active, the system is much smaller than those in an interior-point method. The sparsity structure of the KKT matrices varies from iteration to iteration along with the prediction of the active set, but usually a factorization of the KKT matrix can be updated. Each iteration therefore involves only a small amount of work ($O((n+m)^2)$ in the dense case), but active-set methods typically perform many iterations compared to interior-point methods.

Historically, active-set methods for quadratic programming were developed first. These methods

were based on extending the simplex algorithm for linear programming. Interior-point methods for quadratic programming became popular later and offer several advantages. Interior-point methods can provably solve quadratic programs in weakly polynomial time [98, 64]. Active-set methods can display exponential-time behavior on certain contrived problems [63]. However, in practice they are efficient and the number of iterations required is typically a small polynomial function of the dimension [113]. Average-case complexity analysis has shown the discrepancy between practice and the worst-case behavior of active-set methods [9].

Interior-point methods have a theoretical polynomial iteration bound, and in practice the number of iterations these methods perform is essentially independent of m and n. Although they perform a large amount of work per iteration, the ratio of work to communication is large. Thus, they are able to exploit the memory and cache hierarchy of modern computer architectures, as well as advances in sparse linear system solvers; including parallel methods to solve linear systems. Active-set methods, which must update matrix factorizations from iteration to iteration, perform a small amount of work per iteration but the ratio of work to communication is small. Historically, they have also not been able to exploit advances in sparse linear solvers because they have required specialized solvers in order to maintain the inertia of KKT matrices.

Interior-point methods have an important flaw—they cannot be warm started efficiently [82, 113].

1.5 Overview

In this thesis we develop a method for sparse convex quadratic programming. The motivating application is SQP methods for large-scale nonlinear optimization. An active-set method is chosen because the ability to warm start is crucial for this application. To exploit advances in modern sparse linear solvers, we utilize recent research [59] in updating sparse block-LU factorizations of KKT matrices.

Active-set methods for quadratic programming have difficulty when the KKT matrices become ill-conditioned, and they fail if these matrices become singular. The central theme of this thesis is to avoid singular KKT matrices by perturbing the QP being solved. A sequence of perturbed QP *subproblems* is solved to obtain an accurate approximation to the solution of the original problem.

We consider several different perturbations of the original QP: primal regularization, dual regularization, proximal point, and augmented Lagrangian methods. We develop a method based on primal regularization within a bound-constrained augmented Lagrangian framework. In this method, the QP subproblems are strictly convex and the KKT-type systems solved are guaranteed to be nonsingular for any active set of constraints.

Quadratic programming methods often divide their work into two phases. In the first phase a feasible point for the constraints is found, and in the second an optimal solution is computed. The method in this thesis uses a single-phase algorithm because each subproblem is feasible.

Inertia-controlling methods for quadratic programming are an alternative for avoiding singular KKT systems. These methods must solve one of several different systems to control the number of negative, zero, and positive eigenvalues of the KKT matrices. There is no need to control the inertia

of the KKT matrices arising in the method presented here, because the subproblems involved are strictly convex. Because all KKT matrices are nonsingular, they can be factored with a variety of sparse direct linear solvers. Thus, the method allows us to exploit advances in modern sparse direct solvers, including parallel algorithms. Block-LU updates of the KKT factors allow for active set changes without refactorization.

The primal and dual regularization allows for warm starting from any given active set. This is in contrast to traditional active-set methods, which must start from an active set that defines a nonsingular basis. This ability is beneficial inside SQP methods and eliminates the need for basis repair or KKT repair procedures.

The method is implemented in the QPBLUR (Quadratic Programming with Block-LU updates and Regularization) solver. MATLAB and Fortran 95 versions of QPBLUR have been developed, and the Fortran 95 version has been integrated into SNOPT, a sparse SQP method for nonlinear optimization. The performance of QPBLUR is evaluated on a test set of large convex QPs, and the sequences of QPs arising from SNOPT on a test set of nonlinear programs.

Chapter 2 contains an introduction to current active-set methods for quadratic programming. Chapter 3 describes recent research on block-LU updates for KKT systems arising in active-set methods. Chapter 4 introduces primal and dual regularization, as a perturbation to the original QP, and defines a quadratic programming method based on solving a sequence of regularized QP subproblems. Chapter 5 discusses the relationship between dual regularization and the quadratic penalty method. Chapter 6 presents the bound-constrained Lagrangian method as an alternative to the quadratic penalty method. Chapter 7 defines an algorithm for solving the regularized QP subproblem. Chapter 8 considers how the method may be applied to several different least-squares problems. Chapter 9 discusses the implementation of QPBLUR. Chapter 10 discusses quadratic programming within SQP methods and integration of QPBLUR with the SNOPT solver. Chapter 11 presents computational results. Finally, Chapter 12 presents conclusions, contributions, and further work.

Chapter 2

Active-set methods for quadratic programming

2.1 Background

Active-set methods for quadratic programming are iterative algorithms that generate a sequence of approximate solutions and maintain and update a prediction of the optimal sets of active and inactive constraints. As discussed in Section 1.4, an inequality constraint $a^T x \geq \beta$ is *active* at the current point x if the constraint holds with equality: $a^T x = \beta$. An inequality constraint is *inactive* if it is strictly satisfied: $a^T x > \beta$. Equality constraints of the form $a^T x = \beta$ are always considered active.

For ease of notation, in this chapter we consider QPs of the form

$$\begin{array}{ll} \underset{x}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x \\ \text{subject to} & Ax = b \\ & \ell \leq x \leq u. \end{array} \tag{2.1}$$

In problem (2.1) the inequality constraints occur as simple bounds on the variables x. When a simple-bound constraint in the form $x_j \geq \ell_j$ is active, we have $x_j = \ell_j$. Thus, at each iteration of the algorithm, some variables will be free to move and others will be fixed on their bounds. Let $\mathcal{M}$ and $\mathcal{N}$ denote the set of indices of moving variables and nonmoving variables respectively. Note that $\mathcal{N}$ is often partitioned into two sets $\mathcal{L}$ and $\mathcal{U}$, with the property that $j \in \mathcal{L}$ implies $x_j = \ell_j$ and $j \in \mathcal{U}$ implies $x_j = u_j$. Although $\mathcal{M}$ and $\mathcal{N}$ are sets of indices, we refer to variables "in $\mathcal{M}$" or "in $\mathcal{N}$" if the indices of these variables are contained in these sets.

Active-set algorithms generate a sequence $\{x_k\}$ of approximate solutions to (2.1). Throughout this discussion the subscript k is used to denote quantities associated with the kth iteration of the

algorithm. Each (stationary) iterate x_k is a solution to the following problem:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x \\ \text{subject to} & Ax = b \\ & x_j = \ell_j, \ j \in \mathcal{L}_k \\ & x_j = u_j, \ j \in \mathcal{U}_k \\ & \ell_j \leq x_j \leq u_j, \ j \in \mathcal{M}_k. \end{array} \tag{2.2}$$

(We discuss what it means for an iterate to be a subspace stationary point in Section 2.7.) If $\mathcal{M}_k, \mathcal{L}_k$ and $\mathcal{U}_k$ correspond to optimal sets $\mathcal{M}_\star, \mathcal{L}_\star$ and $\mathcal{U}_\star$ at a solution to (2.1) then x_k is optimal for (2.1). Unfortunately, the optimal sets $\mathcal{M}_\star, \mathcal{L}_\star$ and $\mathcal{U}_\star$ are not known in advance. Thus, active-set algorithms adjust $\mathcal{M}_k, \mathcal{L}_k$ and $\mathcal{U}_k$ from iteration to iteration using information obtained from Lagrange multipliers associated with the constraints $x_j = \ell_j$ and $x_j = u_j$. The name $\mathcal{M}_k$ indicates the set of "moving" variables that are optimized while the variables in $\mathcal{L}_k$ and $\mathcal{U}_k$ are temporarily frozen on their bounds. (More generally, in some active-set methods the variables in $\mathcal{N}_k$ may be frozen anywhere *between* their bounds.)

2.2 The feasible phase and the optimal phase

Primal feasible active-set methods divide their work into two different phases. In Phase 1, or the feasibility phase, such methods try to compute a point that is primal feasible ($Ax = b$ and $\ell \leq x \leq u$). Feasibility can be established by solving a linear program (LP) that minimizes the sum of violations of the constraints. In Phase 2, the optimality phase, the methods maintain feasibility and try to compute an optimal point. Reversion to Phase 1 may be necessary if numerical issues cause a loss of feasibility. Once feasibility is regained Phase 2 may begin again.

2.3 Definition of an iteration

Active-set algorithms evolve iterates x_k via

$$x_{k+1} = x_k + \alpha_k p_k, \tag{2.3}$$

where the *search direction* p_k is a vector that is a direction of descent for the objective function, and the scalar *step length* α_k is nonnegative. In the following discussion we consider a typical Phase 2 iteration (the kth) and use unscripted symbols to denote quantities associated with iteration k when the meaning is clear.

2.4 The search direction

The search direction p is computed by solving the equality constrained problem

$$\underset{p}{\text{minimize}} \quad \phi(x+p) \quad \text{subject to} \quad Ap = 0, \quad p_{\mathcal{N}} = 0, \tag{2.4}$$

where $\phi(x)$ is the quadratic objective $c^T x + \frac{1}{2} x^T H x$. The constraint $Ap = 0$ is chosen so that $A(x + \alpha p) = b$ for any $\alpha \geq 0$. The constraint $p_{\mathcal{N}} = 0$ specifies that variables currently fixed on their bounds do not move. If we expand the objective and drop the constant terms $c^T x$ and $x^T H x$, this problem is

$$\begin{array}{ll} \underset{p}{\text{minimize}} & g^T p + \frac{1}{2} p^T H p \\ \text{subject to} & Ap = 0, \quad p_{\mathcal{N}} = 0, \end{array} \tag{2.5}$$

where $g = c + Hx$ is the gradient of $\phi(x)$. We can eliminate $p_{\mathcal{N}}$ from the above problem to yield

$$\begin{array}{ll} \underset{p_{\mathcal{M}}}{\text{minimize}} & g_{\mathcal{M}}^T p_{\mathcal{M}} + \frac{1}{2} p_{\mathcal{M}}^T H_{\mathcal{M}} p_{\mathcal{M}} \\ \text{subject to} & A_{\mathcal{M}} p_{\mathcal{M}} = 0, \end{array} \tag{2.6}$$

where $H_{\mathcal{M}}$ is a submatrix of the Hessian containing rows and columns of H corresponding to the variables in $\mathcal{M}$, and $A_{\mathcal{M}}$ contains a subset of the columns of A corresponding to variables in $\mathcal{M}$.

If $H_{\mathcal{M}}$ is positive definite, the necessary and sufficient condition for $p_{\mathcal{M}}$ to be a solution to (2.6) is that there exists a vector Lagrange multipliers y such that

$$K_{\mathcal{M}} \begin{pmatrix} p_{\mathcal{M}} \\ -y \end{pmatrix} = \begin{pmatrix} -g_{\mathcal{M}} \\ 0 \end{pmatrix}, \quad \text{with} \quad K_{\mathcal{M}} \equiv \begin{pmatrix} H_{\mathcal{M}} & A_{\mathcal{M}}^T \\ A_{\mathcal{M}} & \end{pmatrix}. \tag{2.7}$$

The matrix $K_{\mathcal{M}}$ in (2.7) is known as a saddle-point matrix or a KKT matrix (because it arises from the KKT optimality conditions of problem (2.6)).

If $H_{\mathcal{M}}$ is singular, problem (2.6) may have a unique or non-unique solution, or it may be unbounded, depending on the inertia (the number of positive, zero, and negative eigenvalues) of $K_{\mathcal{M}}$ and the compatibility of system (2.7).

2.5 Methods for solving KKT systems

There are many different implementations of active-set algorithms. Often, what distinguishes one implementation from another is the linear algebra that is performed. Benzi, Golub, and Liesen [5] discuss many ways to solve KKT systems. We consider only two methods here: nullspace and fullspace.

2.5.1 Nullspace methods

Nullspace methods compute a basis for the nullspace of $A_{\mathcal{M}}$ and use it to solve (2.7). For large sparse $A_{\mathcal{M}}$ an orthogonal basis for the nullspace is likely to be dense and therefore prohibitively expensive to compute and store. Thus, nullspace methods for sparse quadratic programming select a basis for the nullspace that can be computed efficiently (and stored implicitly), but is not necessarily orthogonal. The reduced-gradient method is a nullspace method that constructs a particular basis for the nullspace from a selection of m linear independent columns of $A_{\mathcal{M}}$. To have m linear independent columns, $A_{\mathcal{M}}$ must have full row-rank. This rank requirement places a restriction on the sets $\mathcal{M}$ and $\mathcal{N}$. See Section 2.8 for further details.

The reduced-gradient method partitions the variables in $\mathcal{M}$ into two sets: basic and superbasic. These are carefully chosen and maintained so that at each iteration the matrix $A_{\mathcal{M}}$ can be written as

$$A_{\mathcal{M}}P = \begin{pmatrix} B & S \end{pmatrix}, \tag{2.8}$$

where P is a column permutation matrix, B is an $m \times m$ nonsingular matrix containing linearly independent columns of $A_{\mathcal{M}}$ corresponding to basic variables, and S is an $m \times n_S$ matrix of columns corresponding to superbasic variables. The number of superbasic variables, n_S, is called the *number of degrees of freedom*. With $A_{\mathcal{M}}$ in this form, it is possible to define a matrix $Z_{\mathcal{M}}$ whose columns form a basis for the nullspace of $A_{\mathcal{M}}$ (*i.e.* $A_{\mathcal{M}}Z_{\mathcal{M}} = 0$) as

$$Z_{\mathcal{M}} = P \begin{pmatrix} -B^{-1}S \\ I \end{pmatrix}. \tag{2.9}$$

It should be noted that $Z_{\mathcal{M}}$ is not formed and stored explicitly. Instead, matrix-vector and matrix-matrix products with $Z_{\mathcal{M}}$ and $Z_{\mathcal{M}}^T$ are computed as needed by solves with B and B^T.

If $p_{\mathcal{M}}$ is defined by $p_{\mathcal{M}} = Z_{\mathcal{M}}p_z$, then $A_{\mathcal{M}}p_{\mathcal{M}} = 0$ and system (2.7) can be solved by first solving

$$(Z_{\mathcal{M}}^T H_{\mathcal{M}} Z_{\mathcal{M}})p_z = -Z_{\mathcal{M}}^T g_{\mathcal{M}}. \tag{2.10}$$

The $n_S \times n_S$ matrix $Z_{\mathcal{M}}^T H_{\mathcal{M}} Z_{\mathcal{M}}$ is called the *reduced Hessian* and the vector $Z_{\mathcal{M}}^T g_{\mathcal{M}}$ is called the *reduced gradient*.

We may view the reduced-gradient method as a Schur-complement method in which a specific ordering of the KKT matrix has been chosen. With $A_{\mathcal{M}}$ partitioned as in (2.8), system (2.7) becomes

$$\begin{pmatrix} H_{BB} & H_{BS} & B^T \\ H_{SB} & H_{SS} & S^T \\ B & S & \end{pmatrix} \begin{pmatrix} \Delta x_B \\ \Delta x_S \\ -y \end{pmatrix} = \begin{pmatrix} -g_B \\ -g_S \\ 0 \end{pmatrix}.$$

If we reorder this to be

$$\left(\begin{array}{cc|c} H_{BB} & B^T & H_{BS} \\ B & & S \\ \hline H_{SB} & S^T & H_{SS} \end{array}\right) \begin{pmatrix} \Delta x_B \\ -y \\ \Delta s \end{pmatrix} = \begin{pmatrix} -g_B \\ 0 \\ -g_S \end{pmatrix},$$

then the reduced Hessian is the Schur complement of the top left block in the KKT matrix:

$$Z_{\mathcal{M}}^T H_{\mathcal{M}} Z_{\mathcal{M}} = H_{SS} - \begin{pmatrix} H_{SB} & S^T \end{pmatrix} \begin{pmatrix} H_{BB} & B^T \\ B & \end{pmatrix}^{-1} \begin{pmatrix} H_{BS} \\ S \end{pmatrix}.$$

This definition of the reduced Hessian as a Schur complement reveals that regardless of the sparsity of $H_{\mathcal{M}}$ and $A_{\mathcal{M}}$ the reduced Hessian will tend to be a dense matrix. Thus, nullspace methods work well when n_S, the number of degrees of freedom, is fairly small. The reduced Hessian may be formed (using solves with B and B^T) without forming $Z_{\mathcal{M}}$ explicitly, and a dense Cholesky factorization is appropriate. When n_S becomes large, forming and storing the reduced Hessian can be prohibitively expensive. Thus, some implementations switch to iterative methods (*e.g.* conjugate-gradient) that only require matrix-vector products with the reduced Hessian and hence do not require forming and storing the large dense matrix $Z_{\mathcal{M}}^T H_{\mathcal{M}} Z_{\mathcal{M}}$. These matrix-vector products can be computed (again via solves with B and B^T) without forming the reduced Hessian at all. However, iterative methods may require a large number of iterations to converge to a solution of (2.10) and may be sensitive to the scaling and condition of the reduced Hessian.

The solver SQOPT [41] is an implementation of a reduced-gradient method for sparse convex quadratic programming. SQOPT is discussed in further detail in Section 10.3.

2.5.2 Fullspace methods

Fullspace methods work directly with the KKT matrix

$$K_{\mathcal{M}} = \begin{pmatrix} H_{\mathcal{M}} & A_{\mathcal{M}}^T \\ A_{\mathcal{M}} & \end{pmatrix}$$

using, for instance, a triangular factorization. Because the KKT matrix is symmetric and indefinite, the factorization $P^T K_{\mathcal{M}} P = LBL^T$ is often used, where L is a lower triangular matrix and B is a block-diagonal matrix that contains 1×1 and 2×2 blocks [12, 13]. Gaussian elimination with pivoting can also be used to compute $PK_{\mathcal{M}}Q = LU$, although this approach does not take advantage of the symmetry of the KKT matrix. Iterative methods for symmetric indefinite systems, such as SYMMLQ [85], are an alternative to direct factorization. However, these methods are not often used inside active-set algorithms, as $K_{\mathcal{M}}$ may be ill-conditioned, and it may be difficult to construct an effective preconditioner for KKT matrices arising from an arbitrary active set.

Because $\mathcal{M}$ varies from iteration to iteration it is not practical for a fullspace method to factor $K_{\mathcal{M}}$ at each iteration. Thus, these methods employ routines to update the factorization of $K_{\mathcal{M}_0}$ to

solve later systems involving $K_{\mathcal{M}_k}$ [52, 59]. Chapter 3 discusses one such update method that uses an LU factorization of $K_{\mathcal{M}_0}$ to form a block-LU factorization of an augmented system. The solution of a system involving $K_{\mathcal{M}_k}$ may be extracted from the solution of this augmented system.

Fullspace methods must solve larger systems than nullspace methods. However, when $H_{\mathcal{M}}$ and $A_{\mathcal{M}}$ are sparse, so is $K_{\mathcal{M}}$. Therefore, fullspace methods are competitive with reduced-gradient methods when the reduced Hessian is large and dense.

QPBLU [59] is an implementation of a fullspace method for sparse convex quadratic programming that employs block-LU updates. QPBLU is discussed in further detail in Sections 9.2 and 11.2.

2.6 The step length

After a search direction has been computed (by any method), a step length α is chosen to minimize the objective $\phi(x + \alpha p)$ while not violating any constraints. This is most clearly stated as the one-dimensional optimization problem

$$\begin{array}{ll} \underset{\alpha}{\text{minimize}} & \phi(x + \alpha p) \\ \text{subject to} & \ell \leq x + \alpha p \leq u, \quad \alpha \geq 0, \end{array} \tag{2.11}$$

where α is a scalar and x, p, ℓ and u are problem data. The objective function in (2.11) has the form

$$\phi(x + \alpha p) = \phi(x) + \alpha g^T p + \tfrac{1}{2}\alpha^2 p^T H p,$$

with first and second derivatives given by

$$\frac{d}{d\alpha}\phi(x + \alpha p) = g^T p + \alpha p^T H p, \tag{2.12}$$

$$\frac{d^2}{d\alpha^2}\phi(x + \alpha p) = p^T H p. \tag{2.13}$$

When p is a descent direction (*i.e.* $g^T p < 0$) and $p^T H p > 0$, the unconstrained minimizer of $\phi(x+\alpha p)$ is unique and given by $\alpha_M = -g^T p / p^T H p > 0$. From (2.12) we have that $\phi(x + \alpha p)$ is decreasing on the interval $[0, \alpha_M)$.

When $p = (p_{\mathcal{M}}, p_{\mathcal{N}})$ with $p_{\mathcal{N}} = 0$ and $p_{\mathcal{M}}$ defined by the solution to (2.6), we have $p^T H p = p_{\mathcal{M}}^T H_{\mathcal{M}} p_{\mathcal{M}}$ and $g^T p = g_{\mathcal{M}}^T p_{\mathcal{M}}$. If $p_{\mathcal{M}}$ satisfies system (2.7), $p_{\mathcal{M}}^T H_{\mathcal{M}} p_{\mathcal{M}} = -g_{\mathcal{M}}^T p_{\mathcal{M}}$. Thus, $\alpha_M = -g^T p / p^T H p = 1$ and so the optimal unconstrained step length is the *unit step*.

The constraints $\ell \leq x + \alpha p \leq u$ may prevent $\alpha^\star$, the optimal solution to (2.11), from being the unit step. Because $\phi(x + \alpha p)$ is decreasing on the interval $[0, 1)$ the constrained minimizer will be the maximum feasible step. The maximum feasible step length, α_F, is defined by

$$\alpha_F = \max\left\{ \max_{j \in \mathcal{M}: p_j < 0} \left(\frac{\ell_j - x_j}{p_j} \right), \quad \max_{j \in \mathcal{M}: p_j > 0} \left(\frac{u_j - x_j}{p_j} \right) \right\}.$$

Thus, when $\alpha_F < 1$ the optimal solution is $\alpha^\star = \alpha_F$. If p is a descent direction but $p^T H p = 0$,

the unconstrained minimizer of $\phi(x + \alpha p)$ is unbounded. In this case $\alpha^\star$ will also be the maximum feasible step length α_F.

2.7 Adding and deleting constraints

If $\alpha^\star$ is not the unit step, at least one variable x_j will be placed on its bound when the step $x_{k+1} = x_k + \alpha_k p_k$ is taken. In this case, active-set methods update $\mathcal{M}$ and $\mathcal{N}$ by moving j from $\mathcal{M}$ into $\mathcal{N}$. Thus, a constraint is added to the active set.

If $\alpha^\star$ is the unit step, the iterate x_{k+1} satisfies

$$Ax_{k+1} = b \tag{2.14}$$
$$[x_{k+1}]_j = \ell_j,\ j \in \mathcal{L} \tag{2.15}$$
$$[x_{k+1}]_j = u_j,\ j \in \mathcal{U} \tag{2.16}$$
$$\ell_j \leq [x_{k+1}]_j \leq u_j,\ j \in \mathcal{M}. \tag{2.17}$$

Observe that x_{k+1} is therefore feasible for problem (2.2). In addition, the Lagrange multipliers y at x_{k+1} satisfy

$$g_{\mathcal{M}} = A_{\mathcal{M}}^T y. \tag{2.18}$$

Lagrange multipliers $z_{\mathcal{L}}$ and $z_{\mathcal{U}}$ associated with the equality constraints $x_j = \ell_j$ and $x_j = u_j$ in (2.2) may be defined via

$$z_{\mathcal{L}} = g_{\mathcal{L}} - A_{\mathcal{L}}^T y$$
$$z_{\mathcal{U}} = -g_{\mathcal{U}} + A_{\mathcal{U}}^T y,$$

where $g_{\mathcal{L}}$ and $g_{\mathcal{U}}$ contain the components of $\nabla\phi(x_{k+1})$ belonging to $\mathcal{L}$ and $\mathcal{U}$ respectively. Lagrange multipliers for the inequality constraints involving moving variables may be defined to be zero. With these Lagrange multipliers we see that x_{k+1} is a solution to (2.2). Thus, x_{k+1} minimizes the quadratic objective in a subspace defined by $\mathcal{M}$, $\mathcal{L}$ and $\mathcal{U}$. Therefore, x_{k+1} is called a *subspace minimizer*. In nonconvex quadratic programming, x_{k+1} is a subspace stationary point but not necessarily a subspace minimizer. Because we deal with convex QP we use the terms subspace minimizer and subspace stationary point interchangeably.

If $z_{\mathcal{L}}$ and $z_{\mathcal{U}}$ are nonnegative, x_{k+1} is an optimal solution to the QP (2.1).

If a component of $z_{\mathcal{L}}$ or $z_{\mathcal{U}}$ is negative, an improvement of the objective can be made by moving the corresponding variable x_j off its bound. In this case j is moved from $\mathcal{N}$ into $\mathcal{M}$. Active-set methods perform a routine called *pricing* to determine the variable x_j to free.

Most active-set methods work as described above, with at most one change made to $\mathcal{M}$ and $\mathcal{N}$ per iteration. A class of methods known as gradient-projection methods [82] can make multiple changes to $\mathcal{M}$ and $\mathcal{N}$ in a single iteration. Gradient-projection methods work best on bound-constrained

problems

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \psi(x) \\ \text{subject to} & \ell \leq x \leq u, \end{array}$$

where a projection onto the feasible set $\{x : \ell \leq x \leq u\}$ is easily calculated.

The ability to make multiple changes to $\mathcal{M}$ and $\mathcal{N}$ is beneficial because otherwise the number of iterations required by an active-set method is bounded below by

$$\max(|\mathcal{M}_0 \backslash \mathcal{M}_\star|, |\mathcal{N}_0 \backslash \mathcal{N}_\star|),$$

where $\mathcal{M}_0$ and $\mathcal{N}_0$ are the initial set of moving and nonmoving variables, and $\mathcal{M}_\star$ and $\mathcal{N}_\star$ are the set of moving and nonmoving variables at an optimal solution. For example, consider a problem with 500 variables for which the initial active set has 200 variables fixed on their bounds, and the optimal active set has all variables in between their bounds. If only one change is made to the active set per iteration, at least 200 iterations are required. This lower-bound on the number of iterations may be unacceptable if the problem contains a large number of variables.

2.8 Basis repair and KKT repair

Until now we have not discussed an important property required by nearly all active-set algorithms: that $A_{\mathcal{M}}$ have full row-rank. The reduced-gradient method (a nullspace method) requires a nonsingular basis, the matrix B in (2.8), to define $Z_{\mathcal{M}}$. Fullspace methods require $A_{\mathcal{M}}$ to have full rank so that $K_{\mathcal{M}}$ is nonsingular. Active-set methods have the crucial property that if the initial $A_{\mathcal{M}}$ has full row-rank, then *in exact arithmetic* all subsequent $A_{\mathcal{M}}$ also have full row-rank.

Certain active-set algorithms, called *inertia-controlling* methods, choose $\mathcal{M}$ in order to control the inertia of the reduced Hessian [44]. For convex QP, inertia-controlling methods prevent the reduced Hessian from having more than one zero eigenvalue. When the initial $K_{\mathcal{M}}$ is nonsingular and the initial iterate is a subspace minimizer, this provides a theoretical guarantee that $K_{\mathcal{M}}$ remains nonsingular.

In practice, active-set methods depend critically on B or $K_{\mathcal{M}}$ being reasonably well-conditioned relative to the floating-point precision (typically 15 digits). If these matrices become nearly singular, a procedure called *basis repair* for algorithms that work with B, or *KKT repair* for algorithms that work with $K_{\mathcal{M}}$, is required to determine which variables in $\mathcal{M}$ should be replaced by variables in $\mathcal{N}$ before iterations can proceed reliably.

While basis repair is well understood and efficient, KKT repair is difficult to implement and computationally expensive. It requires sparse LU or LBL^{T} factorizations that are rank-revealing to identify singularity or near-singularity. Unfortunately, there are only a few sparse factorizations that are rank-revealing. All sparse direct factorizations have a parameter τ called the *factor tolerance* that must lie in a specific range ($1 \leq \tau \leq 10$ say). (Typically, one of the triangular factors L has unit diagonals and its off-diagonal elements are bounded by τ.) The value of τ influences the balance between sparsity and stability, with values nearer one giving improved stability. To achieve reliable

rank-revealing properties, τ must be near one. The resulting sparse matrix factors are thus generally more dense and expensive to compute.

Repair procedures are called infrequently in practice. However, a robust solver must include such procedures to deal with ill-conditioned QPs. An elaborate step-length procedure is also required to try and avoid small pivots p_j and prevent B or $K_{\mathcal{M}}$ from becoming unnecessarily ill-conditioned.

Requiring $A_{\mathcal{M}}$ to have full row-rank may also force variables x_j without bounds (those with $-\infty < x_j < \infty$) that should naturally be in $\mathcal{M}$ to be placed in $\mathcal{N}$. When this happens, x_j is said to lie on a *temporary* or *artificial bound*. The algorithm must proceed until it can free x_j from this temporary bound.

2.9 Degeneracy, cycling, and stalling

An active-set algorithm is said to be at a *degenerate point* if a moving variable is on its bound. If this variable forces the maximum feasible step α_F to be zero, the algorithm will remain at the same point and thus not decrease the objective function. Proofs of finite termination for active-set algorithms for quadratic programming rely on a strict decrease in the objective function at each iteration [82] and the fact that the number of possible active sets is finite.

At a degenerate point there is the possibility of a sequence of iterations occurring with no change in the objective function and with the eventual return to the same active set. This behavior is called *cycling*. Procedures such as EXPAND [43] have been developed to try to avoid cycling in active-set methods. However, examples can be constructed where cycling occurs (even with EXPAND) [56].

Cycling can occur in primal active-set methods. However, most dual active-set methods for strictly convex quadratic programming cannot cycle. For more details on nondegeneracy in dual active-set methods see Gill and Wong [46, 47].

One of the causes of cycling is the need to maintain the rank of the matrix $A_{\mathcal{M}}$. Suppose $|\mathcal{M}_k| > m$, the iterate x_k is a degenerate point, and a sequence of degenerate steps $\alpha_k = \alpha_{k+1} = \cdots = \alpha_{k+r} = 0$ are taken until $|\mathcal{M}_{k+r}| = m$. At this point x_{k+r} is a stationary iterate. Thus, the algorithm must look for a variable that can be freed from its bound. Suppose it chooses $j \in \mathcal{N}_{k+r}$ to move into $\mathcal{M}_{k+r+1}$. Let Δx_j denote the jth component of the search direction p_{k+r+1}. The sign of Δx_j is guaranteed to be such that there exists a nonzero α_j for which $x_j + \alpha_j \Delta x_j$ is feasible. Unfortunately, there might be another variable x_i on its bound with $i \in \mathcal{M}_{k+r+1}$ and $i \neq j$ that would force $\alpha_{k+r+1} = 0$. At this point the algorithm must again choose a constraint to delete. Thus, a cycle may occur with $\mathcal{N}_k = \mathcal{N}_{k+r+l}$ and $x_k = x_{k+1} = \cdots = x_{k+r+l}$. In practice cycling occurs rarely; more important is *stalling*, where a long sequence of degenerate points are encountered [8].

Chapter 3

A block-LU method for KKT updates

During each iteration of an active-set method for quadratic programming a search direction p must be computed. As discussed in the previous chapter, p is defined in terms of $p_{\mathcal{M}}$ and $p_{\mathcal{N}}$, as the solution of the equality-constrained QP (2.5). Given an initial set $\mathcal{M}_0$ of moving variables, $p_{\mathcal{M}}$ is defined as the solution of the system

$$K_0 \begin{pmatrix} p_{\mathcal{M}} \\ -y \end{pmatrix} = \begin{pmatrix} -g_{\mathcal{M}} \\ 0 \end{pmatrix}, \quad \text{where} \quad K_0 = \begin{pmatrix} H_0 & A_0^T \\ A_0 & 0 \end{pmatrix}. \tag{3.1}$$

A fullspace method might solve the above system by factoring K_0. However, on the next iteration if a moving variable is fixed, or a nonmoving variable is freed, the set $\mathcal{M}_1$ will differ from $\mathcal{M}_0$ and thus a similar system with a different matrix K_1 will need to be solved. When only a single variable is fixed or freed, K_1 differs from K_0 by one row and column. Thus, it would be inefficient to factor the new matrix K_1 (and given the number of iterations performed by active-set methods, computationally prohibitive). Therefore, fullspace methods use the factors of K_0 in some way to solve with K_1. This chapter describes an update method based on maintaining a block-LU factorization of an augmented matrix. The block-LU update was motivated by the work of Bisschop and Meeraus [7]. It was developed by Gill *et al.* [42] and implemented and tested on a set of LP problems by Eldersveld and Saunders [29]. The presentation in this chapter follows recent work by Huynh [59].

3.1 The augmented KKT matrix

Suppose the initial KKT system is defined by the sets $\mathcal{M}_0$ and $\mathcal{N}_0$ and takes the form of (3.1). The augmented matrix and associated system takes the form

$$\left(\begin{array}{c|c} K_0 & V \\ \hline V^T & D \end{array}\right)\begin{pmatrix} u \\ w \end{pmatrix} = \begin{pmatrix} f \\ h \end{pmatrix}, \tag{3.2}$$

where V and D come from changes to $\mathcal{M}$. The nature of these matrices can be described by considering the different cases that arise as variables are fixed and freed, *i.e.*, as changes are made to $\mathcal{M}$ and $\mathcal{N}$.

Case 1. A variable $r \notin \mathcal{M}_0$ is added to $\mathcal{M}$. The augmented system takes the form

$$\left(\begin{array}{cc|c} H_0 & A_0^T & h_r \\ A_0 & & a_r \\ \hline h_r^T & a_r^T & h_{rr} \end{array}\right)\begin{pmatrix} p_0 \\ -y \\ p_r \end{pmatrix} = \begin{pmatrix} -g_0 \\ 0 \\ -g_r \end{pmatrix},$$

where $h_r \in \mathbb{R}^{|\mathcal{M}_0|}$ is the rth column of H restricted to the variables in $\mathcal{M}_0$, $a_r \in \mathbb{R}^m$ is the rth column of A, $h_{rr} \in \mathbb{R}$ is the rth diagonal of H, $p_r \in \mathbb{R}$ is the rth component of the search direction, and $g_r \in \mathbb{R}$ is the rth component of the gradient. Observe that if we reorder this matrix we recover K_1:

$$\begin{pmatrix} H_0 & h_r & A_0^T \\ h_r^T & h_{rr} & a_r^T \\ A_0 & a_r & \end{pmatrix}.$$

Case 2. A variable $s \in \mathcal{M}_0$ is removed from $\mathcal{M}$. This is the interesting case, because it is not immediately obvious that a variable can be removed from $\mathcal{M}$ by expanding the augmented system. In this case the augmented system is bordered with a row and column of the identity matrix:

$$\left(\begin{array}{cc|cc} H_0 & A_0^T & h_r & e_s \\ A_0 & & a_r & \\ \hline h_r^T & a_r^T & h_{rr} & \\ e_s^T & & & \end{array}\right)\begin{pmatrix} p_0 \\ -y \\ p_r \\ z_s \end{pmatrix} = \begin{pmatrix} -g_0 \\ 0 \\ -g_r \\ 0 \end{pmatrix},$$

where e_s is the vector of all zeros with a one in the sth position. The last row in the above system forces $e_s^T p_0 = p_s = 0$. This implies $s \in \mathcal{N}$.

Case 3. A variable $r \notin \mathcal{M}_0$ is removed from $\mathcal{M}$. Because r was not among the initial moving variables in $\mathcal{M}_0$, it must have been added to $\mathcal{M}$ as in *Case 1.* To place r into $\mathcal{N}$ the corresponding column of V and row and column of D are deleted from the augmented system:

$$\left(\begin{array}{cc|c} H_0 & A_0^T & e_s \\ A_0 & & \\ \hline e_s^T & & \end{array}\right)\begin{pmatrix} p_0 \\ -y \\ z_s \end{pmatrix} = \begin{pmatrix} -g_0 \\ 0 \\ 0 \end{pmatrix}.$$

Case 4. A variable $s \in \mathcal{M}_0$ that was placed in $\mathcal{N}$ is freed. To fix the variable s on its bound the KKT system must have been updated as in *Case 2.* To place s into $\mathcal{M}$, the corresponding column of V and row and column of D are deleted from the augmented system. This yields the original system (3.1).

3.2 Incorporating Hessian updates

Quasi-Newton methods are used in algorithms for nonlinear programming to approximate second derivatives. The popular Broyden-Fletcher-Goldfarb-Shanno (BFGS) update is a rank-two update that maintains symmetry and positive-definiteness of the approximate Hessian. This update is commonly expressed in one of two forms. In this section we discuss how the augmented system (3.2) may be used to incorporate limited-memory BFGS updates.

The BFGS approximate Hessian may be expressed in terms of updates to an initial positive definite matrix H_0:

$$H_k = H_0 + \sum_{j=1}^{k} u_j u_j^T - q_j q_j^T. \tag{3.3}$$

In the limited memory setting H_0 is diagonal and the number of updates k is kept small (say 5–25) to avoid storing many dense vector pairs $\{(u_j, q_j)\}_{j=1}^k$. Gill, Murray and Saunders [39] and Gill and Wong [47] describe how the limited-memory BFGS update may be used inside an SQP method for nonlinear programming.

Equation (3.3) is called the *summation form* of the BFGS update. To ensure positive definiteness of H_k in the presence of numerical errors, the update may be rewritten in *product form* as $H_k = G_k^T G_k$, where G_k is the product of elementary matrices

$$G_k = H_0^{1/2}(I + s_1 v_1^T) \cdots (I + s_k v_k^T), \tag{3.4}$$

and the pairs $\{(s_j, v_j)\}_{j=1}^k$ are related to $\{(u_j, q_j)\}_{j=1}^k$ [11, 39].

If H_k is formed explicitly, it would, in general, be dense. It could therefore be expensive to factor and solve the KKT system

$$K_0 z \equiv \begin{pmatrix} H_k & A_0^T \\ A_0 & \end{pmatrix} \begin{pmatrix} p \\ -y \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}. \tag{3.5}$$

The augmented system (3.2) may be used to accommodate these Hessian updates while preserving the sparsity. Although it is possible to interleave the updates described below with those described in the previous section for handling changes to the active set, in practice, there is no reason not to incorporate Hessian updates first (because they are known in advance).

3.2.1 The summation form of the Hessian update

An updated Hessian expressed in summation form (3.3) may be written as

$$H_k = H_0 + UU^T - QQ^T, \tag{3.6}$$

where $U, Q \in \mathbb{R}^{n\times k}$ are matrices whose columns are the update vectors. These matrices are dense but they have few columns. KKT systems of the form (3.5) may be solved via the augmented system

$$\left(\begin{array}{cc|cc} H_0 & A_0^T & U & Q \\ A_0 & & & \\ \hline U^T & & -I & \\ Q^T & & & I \end{array}\right) \begin{pmatrix} p \\ -y \\ r \\ s \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ 0 \\ 0 \end{pmatrix}. \tag{3.7}$$

Eliminating $r = U^T p$ and $s = -Q^T p$ from the above system yields the equations

$$\begin{aligned} (H_0 + UU^T - QQ^T)p - A_0^T y &= d_1 \\ A_0 p &= d_2. \end{aligned}$$

Thus, we see that when H_k is expressed in summation form we can compute a solution to (3.5) by solving a larger (but sparse) augmented system (3.7).

3.2.2 The product form of the Hessian update

Consider a Hessian after a single product-form update (3.4):

$$H_1 = G_1^T G_1 = (I + vs^T)H_0(I + sv^T). \tag{3.8}$$

The solution to the KKT system

$$\begin{pmatrix} H_1 & A_0^T \\ A_0 & \end{pmatrix} \begin{pmatrix} p \\ -y \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \tag{3.9}$$

may be obtained by solving the augmented system

$$\left(\begin{array}{cc|cc} H_0 & A_0^T & \bar{s} & v \\ A_0 & & & \\ \hline \bar{s}^T & & \gamma & -1 \\ v^T & & -1 & \end{array}\right) \begin{pmatrix} p \\ -y \\ \rho \\ \sigma \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ 0 \\ 0 \end{pmatrix}, \tag{3.10}$$

where $\bar{s} = H_0 s$ and $\gamma = s^T H_0 s$. To see this, use the last two rows to eliminate $\rho = v^T p$ and $\sigma = \bar{s}^T p + \gamma\rho$, leaving

$$\begin{aligned}(H_0 + \bar{s}v^T + v\bar{s}^T + \gamma vv^T)p - A_0^T y &= d_1 \\ A_0 p &= d_2.\end{aligned}$$

Expanding the updated Hessian

$$\begin{aligned}H_1 &= (I + vs^T)H_0(I + sv^T) \\ &= H_0 + H_0 sv^T + vs^T H_0 + v(s^T H_0 s)v^T \\ &= H_0 + \bar{s}v^T + v\bar{s}^T + \gamma vv^T,\end{aligned}$$

we find that (3.10) is equivalent to

$$\begin{aligned}H_1 p - A_0^T y &= d_1 \\ A_0 p &= d_2,\end{aligned}$$

and thus solves system (3.9). After k updates to H_0 of the form

$$\begin{aligned}H_k &= (I + v_k s_k^T)H_{k-1}(I + s_k v_k^T) \\ &= (I + v_k s_k^T)\cdots(I + v_1 s_1^T)H_0(I + s_1 v_1^T)\cdots(I + s_k v_k^T),\end{aligned}$$

KKT systems of the form (3.5) may be solved via an augmented system involving H_0:

$$\left(\begin{array}{cc|cccccc} H_0 & A_0^T & \bar{s}_1 & v_1 & \cdots & \bar{s}_k & v_k \\ A_0 & & & & & & \\ \hline \bar{s}_1^T & & \gamma_1 & -1 & & & \\ v_1^T & & -1 & & & & \\ \vdots & & & & \ddots & & \\ \bar{s}_k^T & & & & & \gamma_k & -1 \\ v_k^T & & & & & -1 & \end{array}\right)\begin{pmatrix} p \\ -y \\ \rho_1 \\ \sigma_1 \\ \vdots \\ \rho_k \\ \sigma_k \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix},$$

where $\bar{s}_i = H_{i-1}s_i$ and $\gamma_i = s_i^T \bar{s}_i = s_i^T H_{i-1} s_i$ for $i = 1, \ldots, k$.

3.3 A block-LU factorization of the augmented system

The block-LU method maintains a block factorization of the augmented matrix in (3.2):

$$\begin{pmatrix} K_0 & V \\ V^T & D \end{pmatrix} = \begin{pmatrix} L_0 & \\ Z^T & I \end{pmatrix}\begin{pmatrix} U_0 & Y \\ & C \end{pmatrix}. \tag{3.11}$$

The matrices L_0, U_0, Y, Z and C in the block factors are defined by

$$\begin{aligned} L_0 U_0 &= K_0 \\ L_0 Y &= V \\ U_0^T Z &= V \\ C &= D - Z^T Y, \end{aligned}$$

where $C = D - V^T K_0^{-1} V$ is the Schur complement of K_0 in the augmented matrix. The solution to augmented system (3.2) may then be found by solving

$$\begin{aligned} L_0 t &= f \\ C w &= h - Z^T t \\ U_0 u &= t - Y w. \end{aligned}$$

To compute the search direction p and Lagrange multipliers y this method requires only one solve with K_0 (through the separate L_0 and U_0 solves) and one with C at each iteration.

3.4 Updating the block-LU factorization

Consider the block-LU factorization of the augmented matrix defined in (3.11). The matrices C, Y and Z are updated as rows of $\begin{pmatrix} V^T & D \end{pmatrix}$ or columns of $\begin{pmatrix} V \\ D \end{pmatrix}$ are added and removed from the augmented matrix. If a column $\begin{pmatrix} v \\ d \end{pmatrix}$ is added, Y and C are updated according to

$$Y = \begin{pmatrix} Y & y \end{pmatrix} \text{ and } C = \begin{pmatrix} C & c \end{pmatrix},$$

where $L_0 y = v$ and $c = d - Z^T y$. If a row $\begin{pmatrix} v^T & d^T \end{pmatrix}$ is appended, Z and C are updated according to

$$Z = \begin{pmatrix} Z & z \end{pmatrix} \text{ and } C = \begin{pmatrix} C \\ c^T \end{pmatrix},$$

where $U_0^T z = v$ and $c^T = d^T - z^T Y$. If a column of $\begin{pmatrix} V \\ D \end{pmatrix}$ is deleted, the corresponding columns of Y and C are deleted. If a row of $\begin{pmatrix} V^T & D \end{pmatrix}$ is deleted, the corresponding column of Z and row of C are deleted.

The matrices Y and Z are often sparse (and are stored as such). To see this, note that when L_0 and U_0 come from a sparse factorization of K_0, and v is a sparse right-hand side, the solutions y and z to $L_0 y = v$ and $U_0^T z = v$ are often sparse. (For more details on sparse forward and backward solves with sparse right-hand sides see Davis [24].)

The matrix C is likely to be dense and is treated as a dense matrix. The size of C is bounded

by the number of block-LU updates, and in practice is often significantly smaller.

Instead of explicitly updating the matrix C or its inverse we update a dense LU-type factorization of C [15, 69]. This factorization takes the form $EC = R$, where E is a square (but not lower triangular) well-conditioned product of elementary transformations [110], and R is upper triangular.

3.5 A block-LBL$^{\mathrm{T}}$ update

The block-LU update does not make use of the symmetry of K_0 and the augmented system. To take advantage of the symmetry of K_0, we may use the symmetric indefinite factorization $K_0 = L_0 B_0 L_0^T$, where B_0 is symmetric indefinite and block-diagonal with 1×1 and 2×2 blocks. This leads to the following symmetric block-LBL$^{\mathrm{T}}$ factorization of the augmented matrix:

$$\begin{pmatrix} K_0 & V \\ V^T & D \end{pmatrix} = \begin{pmatrix} L_0 & \\ Y^T & I \end{pmatrix} \begin{pmatrix} B_0 & \\ & C \end{pmatrix} \begin{pmatrix} L_0^T & Y \\ & I \end{pmatrix}. \tag{3.12}$$

The block factors L_0, B_0, Y and C are defined by

$$\begin{aligned} L_0 B_0 L_0^T &= K_0 \\ L_0 W &= V \\ B_0 Y &= W \\ C &= D - Y^T W. \end{aligned}$$

While the block-LU method requires storage for both Y and Z, the block-LBL$^{\mathrm{T}}$ method only requires storage for Y. The solution of system (3.2) may then be found by solving

$$\begin{aligned} L_0 t &= f \\ B_0 v &= t \\ C w &= h - Y^T t \\ L_0^T u &= v - Y w. \end{aligned}$$

This method requires the equivalent of one solve with K_0 and one solve with C at each iteration.

Chapter 4

Primal and dual regularization

A common approach to solving an ill-posed problem—one whose solution is not unique or is extremely sensitive to small perturbations in data—is to construct a related problem whose solution is unique and differs only slightly from a solution of the original problem. Regularization is a method of constructing such a nearby problem. In this chapter we apply regularization to QPs in order to ensure that their solution is unique and bounded, and that all search directions in the solution algorithm are uniquely defined. We investigate the effect that regularization has on the KKT systems solved during the course of an active-set algorithm for quadratic programming. Then we propose an algorithm for convex quadratic programming based on solving a sequence of regularized QP subproblems.

4.1 Primal regularization

Regularization can take many forms. A common form, and the one we consider in this thesis, is Tikhonov regularization. This approach is named for the Russian mathematician Andrey Tikhonov, who used regularization to solve ill-posed integral equations [103]. Tikhonov regularization adds a penalty term to the objective function based on the 2-norm of the current point. Thus, if $\phi(x)$ is the original convex objective, the regularized objective function will be $\phi_\delta(x) = \phi(x) + \frac{1}{2}\delta \|D_x x\|_2^2$, where D_x is some chosen full-rank matrix and $\delta > 0$ is a regularization parameter.

We refer to Tikhonov regularization as *primal regularization* because it is applied to the variables x of the primal problem. Primal regularization alters the curvature of the objective function. The Hessian of the regularized objective ϕ_δ is $H + \delta D_x^T D_x$, which is positive definite for all $\delta > 0$. Thus, ϕ_δ is strictly convex even if ϕ is merely convex.

If we apply primal regularization to the QP (1.2) with $D_x = D_s = I$ we get the primal-regularized QP:

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & \phi_\delta(x,s) \equiv c^T x + \frac{1}{2} x^T H x + \frac{1}{2}\delta \|x\|_2^2 + \frac{1}{2}\delta \|s\|_2^2 \\ \text{subject to} & Ax - s = 0 \\ & \ell \leq (x,s) \leq u. \end{array} \qquad (\text{QP}(\delta))$$

The Hessian of the regularized objective

$$\nabla^2 \phi_\delta(x,s) = \begin{pmatrix} H + \delta I & \\ & \delta I \end{pmatrix}$$

is positive definite. Thus, (QP(δ)) has an unique solution.

Consider an active-set algorithm applied to (QP(δ)) and suppose that the algorithm is at a feasible point (x,s) with sets $\mathcal{M}$ and $\mathcal{N}$ of moving variables and nonmoving variables. A search direction $p = (\Delta x, \Delta s)$ will be computed by solving an equality constrained QP of the form (2.6), whose solution is defined by the KKT system

$$\begin{pmatrix} H_{\mathcal{M}} + \delta I & & A_{\mathcal{M}}^T \\ & \delta I & -I_{\mathcal{M}}^T \\ A_{\mathcal{M}} & -I_{\mathcal{M}} & \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ \Delta s_{\mathcal{M}} \\ -y \end{pmatrix} = \begin{pmatrix} -g_x \\ -g_s \\ 0 \end{pmatrix},$$

where $I_{\mathcal{M}}$ is a matrix containing columns of the identity corresponding to moving slack variables, and g_x and g_s are components of $\nabla\phi(x,s)$ corresponding to moving variables. If the matrix $\begin{pmatrix} A_{\mathcal{M}} & -I_{\mathcal{M}} \end{pmatrix}$ does not have full row-rank, the above KKT matrix is singular. Thus, an algorithm to solve (QP(δ)) must contain a method for choosing variables to enter and leave $\mathcal{M}$ so that this matrix retains full row-rank. We consider other means of avoiding singular KKT matrices later in this chapter.

4.1.1 Exact regularization

In general the solution of (QP(δ)) need not be a solution of the original problem. We say that regularization is *exact* if the solution of (QP(δ)) is also a solution of the original QP for all positive values of δ below some threshold value $\bar{\delta}$. We now present a theorem that states the conditions for a QP to admit exact regularization.

Theorem 1. *Let $\phi^\star$ denote the optimal objective value of a QP in the form of problem* (1.2)*. The solution of* (QP(δ)) *is a solution of problem* (1.2) *for all $0 < \delta < \bar{\delta}$ when the following selection problem has a Lagrange multiplier λ:*

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & \frac{1}{2}\|x\|_2^2 + \frac{1}{2}\|s\|_2^2 \\ \text{subject to} & Ax - s = 0 \\ & \ell \le (x,s) \le u \\ & \phi(x) \le \phi^\star. \end{array} \tag{4.1}$$

If the optimal Lagrange multiplier $\lambda^\star$ for the constraint $\phi(x) \le \phi^\star$ exists, the threshold regularization value $\bar{\delta}$ is given by $\bar{\delta} = 1/\lambda^\star$.

Proof. See the proof of Theorem 2.1, on the exact regularization of convex programs, in Friedlander and Tseng [35]. □

Note that the selection problem (4.1) is feasible whenever the original QP (1.2) is feasible and has a bounded solution. However, the Lagrange multiplier λ need not exist (be finite). For example, the one-dimensional QP

$$\text{minimize} \quad (x-1)^2$$

admits no exact regularization. The solution of this problem is $x^\star = 1$, but the solution to the regularized problem

$$\text{minimize} \quad (x-1)^2 + \delta x^2$$

is given by $x^\star = 1/(1+\delta) \neq 1$ for all $\delta > 0$. In terms of Theorem 1, the selection problem is

$$\text{minimize} \quad x^2 \quad \text{subject to} \quad (x-1)^2 \leq 0,$$

whose dual is

$$\text{maximize} \quad \frac{\lambda}{1+\lambda} \quad \text{subject to} \quad \lambda \geq 0.$$

The dual objective never achieves the optimal primal objective of 1 and the dual problem has no bounded solution $\lambda^\star$. Thus, in general, a QP may not admit exact regularization. In contrast, for LPs there always exists a threshold regularization parameter, and all LPs admit exact regularization [71, 35].

Even if a QP admits exact regularization, finding the threshold regularization value $\bar{\delta}$ would involve solving the original QP to obtain the optimal value $\phi^\star$, solving the selection problem (4.1) to obtain $\lambda^\star$, and taking $\bar{\delta} = 1/\lambda^\star$. In this thesis we are interested in using regularization to enable the original QP to be solved more efficiently. Performing the process described above would defeat this purpose.

4.1.2 The primal regularized algorithm

A point $(x_k^\star, s_k^\star)$ is a solution of the regularized problem (QP(δ_k)) if and only if it satisfies the following optimality conditions:

$$\begin{pmatrix} A & -I \end{pmatrix} \begin{pmatrix} x_k^\star \\ s_k^\star \end{pmatrix} = 0, \quad \ell \leq \begin{pmatrix} x_k^\star \\ s_k^\star \end{pmatrix} \leq u, \tag{4.2}$$

$$\nabla \phi_{\delta_k}(x_k^\star, s_k^\star) = \begin{pmatrix} A^T \\ -I \end{pmatrix} y_k^\star + z_{l\,k}^\star - z_{u\,k}^\star, \tag{4.3}$$

$$\left(\begin{pmatrix} x_k^\star \\ s_k^\star \end{pmatrix} - \ell \right)^T z_{l\,k}^\star = 0, \quad \left(u - \begin{pmatrix} x_k^\star \\ s_k^\star \end{pmatrix} \right)^T z_{u\,k}^\star = 0, \quad z_{l\,k}^\star, z_{u\,k}^\star \geq 0, \tag{4.4}$$

where $y_k^\star$ are optimal Lagrange multipliers for the equality constraints, and $z_{l\,k}^\star$ and $z_{u\,k}^\star$ are Lagrange multipliers for the inequality constraints. Conditions (4.2) and (4.4) are identical to the optimality conditions (1.3) and (1.5) of the original QP (1.2). The dual optimality condition (4.3) differs from (1.4) because the regularized objective $\phi_{\delta_k}(x, s)$ differs from $\phi(x, s)$.

Algorithm 1 The primal regularized algorithm

Require: Optimality tolerance ω, initial regularization δ_0, reduction factor $\tau > 1$
  `converged` $\leftarrow$ `false`
  $k \leftarrow 0$
  repeat
    Solve (QP(δ_k)) to obtain $(x_k^\star, s_k^\star, y_k^\star, z_{l_k}^\star, z_{u_k}^\star)$
    `converged` $\leftarrow \left\| \nabla\phi(x_k^\star, s_k^\star) - \begin{pmatrix} A^T \\ -I \end{pmatrix} y_k^\star - z_{l_k}^\star + z_{u_k}^\star \right\|_\infty \leq \omega$
    $\delta_{k+1} \leftarrow \delta_k/\tau$ [Decrease primal regularization parameter]
    $k \leftarrow k + 1$
  until `converged`
  return $(x_k^\star, s_k^\star, y_k^\star, z_{l_k}^\star, z_{u_k}^\star)$

A strategy for obtaining a solution of the original QP is to solve a sequence of problems (QP(δ_k)) with a sequence of decreasing primal regularization parameters $\{\delta_k\}$ with the property that $\delta_k \to 0$. The algorithm may stop when the regularized solution satisfies the optimality condition (1.4) to within a tolerance ω. This strategy is formalized in Algorithm 1.

Computational experiments by Friedlander [33] show that modest values such as $\delta = 10^{-4}$ are sufficiently small for the above strategy to converge on a wide range of LPs. In Chapter 11 we present numerical results on a range of QPs when primal and dual regularization has been applied. Dual regularization is discussed in Section 4.3.

4.2 The primal proximal point method

A method closely related to the primal regularization algorithm is the method of proximal points. We consider the proximal point method for solving the convex program

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \psi(x) \\ \text{subject to} & x \in \mathcal{C}, \end{array} \tag{4.5}$$

where $\psi(x)$ is a smooth convex function and $\mathcal{C}$ is a closed convex set. The method generates a sequence of iterates $\{x_k\}$ that are approximate solutions to (4.5) by solving a sequence of subproblems of the form

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \psi(x) + \frac{1}{2}\delta_k \|x - x_k\|_2^2 \\ \text{subject to} & x \in \mathcal{C}, \end{array} \tag{PPt$_k$}$$

where $\{\delta_k\}$ is a sequence of decreasing positive parameters such that $\delta_k \to \delta^\star$ with $\delta^\star \geq 0$. The solution of (PPt$_k$) yields the next iterate x_{k+1}. A *proximal point* term has been added to the original objective, which penalizes solutions far from the previous iterate.

Let $\psi_{\delta_k}(x)$ denote the objective in problem (PPt$_k$). The Hessian of $\psi_{\delta_k}(x)$ is $\nabla^2\psi(x) + \delta_k I$, which is positive definite even if $\nabla^2\psi(x)$ is positive semidefinite. Thus, the solution x_{k+1} to (PPt$_k$)

Algorithm 2 The primal proximal point algorithm

Require: Optimality tolerance ω, initial parameter δ_0, initial point (x_0, s_0), reduction factor $\tau > 1$

```
converged ← false
k ← 0
repeat
  Solve (QP(δ_k, x_k, s_k)) to obtain (x_{k+1}, s_{k+1}, y_{k+1}, z_{l k+1}, z_{u k+1})
  converged ← ‖∇φ(x_{k+1}, s_{k+1}) − (A^T; −I) y_{k+1} − z_{l k+1} + z_{u k+1}‖_∞ ≤ ω
  δ_{k+1} ← δ_k/τ                    [Decrease parameter]
  k ← k + 1
until converged
return (x_{k+1}, s_{k+1}, y_{k+1}, z_{l k+1}, z_{u k+1})
```

is unique. Note that the Hessian of $\psi_{\delta_k}(x)$ has the same form as the Hessian of the primal regularized objective $\phi_{\delta_k}(x)$.

The theory of the proximal point algorithm for convex optimization was developed by Rockafellar [91, 92]. Ferris [30] gives conditions under which the proximal point algorithm will terminate in a finite number of iterations, and when it will terminate in a single iteration.

The proximal point method applied to the QP (1.2) yields subproblems of the form

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x + \frac{1}{2}\delta_k \left\|x - x_k\right\|_2^2 + \frac{1}{2}\delta_k \left\|s - s_k\right\|_2^2 \\ \text{subject to} & Ax - s = 0 \\ & \ell \leq (x, s) \leq u. \end{array} \qquad (\text{QP}(\delta_k, x_k, s_k))$$

The proximal point algorithm for convex quadratic programming is formalized in Algorithm 2.

We return to the proximal point method (applied to the dual QP) in Section 6.2.

4.3 Dual regularization

Regularization may also be applied to dual variables (Lagrange multipliers associated with the equality constraints). Consider the problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \phi(x) \\ \text{subject to} & Ax = b \\ & x \geq 0, \end{array} \qquad (4.6)$$

where $\phi(x)$ is a convex quadratic objective. Suppose the constraints and the objective of this problem are perturbed to yield the problem

$$\begin{array}{ll} \underset{x,y}{\text{minimize}} & \phi(x) + \frac{1}{2}\mu \|y\|_2^2 \\ \text{subject to} & Ax + \mu y = b \\ & x \geq 0, \end{array} \qquad (\text{QP}(\mu))$$

where μ is a positive parameter called the *dual regularization parameter*. Let w denote the dual variables associated with the constraint $Ax + \mu y = b$. From the dual optimality condition

$$\begin{pmatrix} \nabla\phi(x^\star) \\ \mu y^\star \end{pmatrix} = \begin{pmatrix} A^T \\ \mu I \end{pmatrix} w^\star + \begin{pmatrix} z^\star \\ 0 \end{pmatrix}$$

we conclude that $y^\star = w^\star$. Thus, we may consider y to be the dual variables in (QP(μ)).

A linearly constrained problem containing dual regularization has an important property: the perturbed constraints $\{Ax + \mu y = b, x \geq 0\}$ are always feasible. This is because there are no inequality constraints on y, and so for any $x \geq 0$ we may choose $\mu y = b - Ax$ to satisfy the equality constraints.

Dual regularization leads to modified KKT systems. Consider an active-set method applied to (QP(μ)), and suppose the algorithm is at a point (x, y) with sets $\mathcal{M}$ and $\mathcal{N}$ of moving and nonmoving variables. A search direction $\Delta x_{\mathcal{M}}$ and a new dual iterate y^+ can be computed by solving the modified KKT system

$$\begin{pmatrix} H_{\mathcal{M}} & A_{\mathcal{M}}^T \\ A_{\mathcal{M}} & -\mu I \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ -y^+ \end{pmatrix} = \begin{pmatrix} -g_{\mathcal{M}} \\ r \end{pmatrix},$$

where $r = b - Ax$. Note the presence of the μI term in the $(2, 2)$ block of this KKT-type system. The above matrix is singular if and only if there is a nonzero vector that lies in both the nullspace of $H_{\mathcal{M}}$ and the nullspace of $A_{\mathcal{M}}$. We consider methods for avoiding singular KKT systems later in this chapter.

We now present a dual regularized algorithm for solving problem (4.6). Let $(x_k^\star, y_k^\star, z_k^\star)$ denote an optimal solution to (QP(μ_k)). Except for primal feasibility, $(x_k^\star, y_k^\star, z_k^\star)$ satisfies the optimality conditions of the original problem (4.6). The dual regularized algorithm attempts to find a feasible point by solving a sequence of problems (QP(μ_k)) with a decreasing sequence of dual regularized parameters $\{\mu_k\}$ where $\mu_k \to 0$. The algorithm stops when a solution $(x_k^\star, y_k^\star, z_k^\star)$ satisfies $\|Ax_k - b\|_\infty \leq \epsilon$ for some specified feasibility tolerance ϵ. This procedure is defined formally in Algorithm 3.

In Chapter 5 we discuss the equivalence between the dual regularization algorithm and the quadratic penalty method.

Algorithm 3 The dual regularized algorithm

Require: Feasibility tolerance ϵ, initial dual regularization parameter μ_0, reduction parameter $\tau > 1$

```
converged ← false
k ← 0
repeat
  Solve (QP(μ_k)) to obtain (x*_k, y*_k, z*_k)
  converged ← ‖Ax*_k − b‖_∞ ≤ ε
  μ_{k+1} ← μ_k/τ                [Decrease dual regularization parameter]
  k ← k + 1
until converged
return (x*_k, y*_k, z*_k)
```

4.4 Primal-dual regularization

If we apply both primal and dual regularization to problem (1.2) we arrive at the primal-dual regularized QP:

$$\begin{array}{ll} \underset{x,s,y}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x + \frac{1}{2}\delta \|x\|_2^2 + \frac{1}{2}\delta \|s\|_2^2 + \frac{1}{2}\mu \|y\|_2^2 \\ \text{subject to} & Ax - s + \mu y = 0 \\ & \ell \leq (x, s) \leq u. \end{array} \qquad (\text{QP}(\delta, \mu))$$

Provided $\ell \leq u$ a solution to (QP(δ, μ)) exists and is unique.

Consider an active-set algorithm applied to (QP(δ, μ)) and suppose we are at a feasible iterate (x, s, y) with sets $\mathcal{M}$ and $\mathcal{N}$ of moving and nonmoving variables. A search direction $p = (\Delta x_{\mathcal{M}}, \Delta s_{\mathcal{M}})$ and an updated iterate y^+ can be found by solving the KKT system

$$\begin{pmatrix} H_{\mathcal{M}} + \delta I & & A_{\mathcal{M}}^T \\ & \delta I & -I_{\mathcal{M}}^T \\ A_{\mathcal{M}} & -I_{\mathcal{M}} & -\mu I \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ \Delta s_{\mathcal{M}} \\ -y^+ \end{pmatrix} = \begin{pmatrix} -g_x \\ -g_s \\ \mu y \end{pmatrix}. \qquad (4.7)$$

Regardless of the choice of $\mathcal{M}$ and $\mathcal{N}$ the above KKT matrix is nonsingular. We prove this in Theorem 2 by considering a matrix in the same form as the matrix in (4.7).

Theorem 2. *Let H be an $n \times n$ positive semidefinite matrix and A be an $m \times n$ matrix of rank $r \leq \min(m, n)$. Regardless of the relationship between m and n (i.e. $m < n$, $m > n$ or $m = n$) the matrix*

$$\begin{pmatrix} H + \delta I & A^T \\ A & -\mu I \end{pmatrix}$$

is nonsingular for all $\delta, \mu > 0$.

Proof. The proof is by contradiction. Suppose there exists some $(x, y) \neq 0$ such that

$$\begin{pmatrix} H + \delta I & A^T \\ A & -\mu I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

From $Ax = \mu y$ we know that if $x = 0$ then $y = 0$. Therefore we must have $x \neq 0$. If we eliminate y from the above system we have

$$\mu H x + \mu \delta x + A^T A x = 0,$$

and therefore

$$\mu x^T H x + \mu \delta x^T x + x^T A^T A x = 0.$$

The first and last terms in this sum are nonnegative because H and $A^T A$ are positive semidefinite. The middle term $\mu \delta x^T x$ is strictly positive. Thus, the sum is strictly positive. We have reached a contradiction and so must conclude the matrix is nonsingular. □

The nonsingularity of the KKT matrices is a crucial property. It means an algorithm may move variables into and out of $\mathcal{M}$ without ensuring that $\begin{pmatrix} A_{\mathcal{M}} & -I_{\mathcal{M}} \end{pmatrix}$ retains full row-rank. In fact, A need not have full row-rank. Thus, a very simple algorithm may be used to solve (QP(δ, μ)). The algorithm need not control the inertia of the KKT matrices, or employ a Phase 1 method. There is also no need for KKT repair. Chapter 7 discusses in detail methods for solving (QP(δ, μ)).

4.5 A primal-dual regularized algorithm

Unfortunately, (QP(δ, μ)) is a perturbed problem. To recover the solution of the original QP (assuming one exists) we need to remove the regularization. To do this we solve a sequence of regularized subproblems (QP(δ, μ)) and let $\delta, \mu \to 0$. This procedure is developed in Algorithm 4. Algorithm 4 does not try to compute an exact solution (if the QP does not admit exact regularization we can only approach the solution in the limit as $\delta \to 0$) or become exactly feasible (again we only expect feasibility in the limit as $\mu \to 0$). Instead, Algorithm 4 tries to make δ and μ small enough that the optimality conditions are satisfied to within a tolerance ω and the constraints are satisfied to within a tolerance ϵ.

Algorithm 4 The primal-dual regularized quadratic programming algorithm

Require: Feasibility tolerance ϵ, optimality tolerance ω, initial regularization parameters (δ_0, μ_0), reduction parameter $\tau > 1$

`converged` $\leftarrow$ `false`, `feasible` $\leftarrow$ `false`

repeat

Solve (QP(δ_k, μ_k)) to obtain $(x_k^\star, s_k^\star, y_k^\star, z_{l_k}^\star, z_{u_k}^\star)$

`feasible` $\leftarrow \|Ax_k^\star - s_k^\star\|_\infty \leq \epsilon$

`converged` $\leftarrow \left\| \nabla\phi(x_k^\star, s_k^\star) - \begin{pmatrix} A^T \\ -I \end{pmatrix} y_k^\star - z_{l_k}^\star + z_{u_k}^\star \right\|_\infty \leq \omega$ and `feasible`

$\delta_{k+1} \leftarrow \delta_k/\tau$ [Decrease primal regularization parameter]

$\mu_{k+1} \leftarrow \mu_k/\tau$ [Decrease dual regularization parameter]

until `converged`

return $(x_k^\star, s_k^\star, y_k^\star, z_{l_k}^\star, z_{u_k}^\star)$

4.6 Symmetric Quasidefinite matrices

The KKT matrices that appear in (4.7) have another important property: for certain values of δ and μ these matrices are symmetric quasidefinite. Vanderbei [106] introduced *symmetric quasidefinite* (SQD) matrices of the form

$$K = \begin{pmatrix} H & A^T \\ A & -G \end{pmatrix},$$

where H and G are symmetric positive definite matrices, and advocated use of sparse Cholesky-type factors $PKP^T = LDL^T$, where D is diagonal but indefinite. Note the difference between this factorization and the symmetric indefinite factorization $PKP^T = LBL^T$, where B is block diagonal with 1×1 and 2×2 blocks. Vanderbei [105] discusses the use of these matrices and Cholesky-type factorizations within an early version of the LOQO QP solver.

Because the indefinite Cholesky factors exist for all permutations, P may be chosen to maintain sparsity (as in the positive definite case). However, with K indefinite, choosing P to maintain sparsity does not always lead to a stable factorization. If a stable method were used to factorize K and solve $Kz = d$, the relative error in $\hat{z}$ (the computed z) would be bounded by an expression of the form

$$\frac{\|\hat{z} - z\|}{\|z\|} \leq \epsilon\eta \,\mathrm{Cond}(K),$$

where ϵ is the floating-point precision, η is a slowly-growing function of the dimension of K, and $\mathrm{Cond}(K)$ is the condition number of K. A stability analysis by Gill *et al.* [45] concluded that if a Cholesky-type method is used to factor K in order to solve $Kz = d$, then $\mathrm{Cond}(K)$ in the above expression can be replaced by an *effective condition number* $\mathrm{Econd}(K)$ given by

$$\mathrm{Econd}(K) = (1 + \omega(K))\mathrm{Cond}(K), \quad \text{with} \quad \omega(K) = \frac{\max\{\|A^T G^{-1} A\|, \|AH^{-1}A^T\|\}}{\|K\|}.$$

The ability to choose P to maintain sparsity of the factors of KKT matrices is important. Sparser Cholesky-type factors typically require much less memory to store and time to compute.

4.7 Previous use of regularization for well-conditioned linear systems

Regularization has been used previously to improve the properties of linear systems solved within optimization algorithms. In this section, we review two previous approaches and compare them to the method described in this thesis.

4.7.1 Uniform primal-dual regularization

Regularization has long been used as a method for perturbing LPs and QPs to improve the condition of KKT systems arising within primal-dual interior-point methods [37, 38]. Following Saunders [94], Saunders and Tomlin [97, 96] considered the regularized QP

$$\begin{array}{ll} \underset{x,p}{\text{minimize}} & c^T x + \frac{1}{2} x^T Q x + \frac{1}{2} \|p\|^2 \\ \text{subject to} & Ax + \delta p = b, \quad \ell \leq x \leq u, \end{array} \tag{4.8}$$

where $Q = Q_0 + \gamma^2 I$ with Q_0 positive semidefinite (for an LP, $Q_0 = 0$). Here γ and δ are *uniform* regularization parameters. These parameters have fixed values, and no attempt is made to reduce them. Instead, the solution of the regularized problem is treated as an approximate solution to the original problem.

The primal-dual interior-point method applied to problem (4.8) results in KKT matrices of the form

$$K = \begin{pmatrix} -H & A^T \\ A & \delta^2 I \end{pmatrix}, \quad H \equiv D_x + Q_0 + \gamma^2 I,$$

where D_x is a positive semidefinite diagonal matrix that changes at each iteration. The matrix K is SQD when γ and δ are positive. If γ and δ are sufficiently positive, systems with K are solved with a Cholesky-type factorization. For scaled LPs with $\|A\| \simeq 1$, $\|b\| \simeq 1$ and $\|c\| \simeq 1$, they found $\gamma = \delta = 10^{-3}$ to be a reliable choice of regularization parameters that ensured a sufficiently stable Cholesky-type factorization and did not perturb the solutions too much.

This approach is continued in the PDCO solver [88] for convex optimization problems with linear constraints:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \psi(x) \\ \text{subject to} & Ax = b, \quad \ell \leq x \leq u, \end{array}$$

where $\psi(x)$ is a smooth convex function. To ensure unique primal and dual solutions (and improve solver stability), PDCO works with the regularized problem

$$\begin{array}{ll} \underset{x,r}{\text{minimize}} & \psi(x) + \frac{1}{2} \|D_1 x\|^2 + \frac{1}{2} \|r\|^2 \\ \text{subject to} & Ax + D_2 r = b, \quad \ell \leq x \leq u, \end{array}$$

where D_1 and D_2 are user-specified positive definite diagonal matrices.

The choice of uniform regularization parameters distinguishes these methods from the one described in this thesis. Because interior-point methods cannot be warm started efficiently, it would be difficult to make use of the solution of one regularized problem when solving another with reduced regularization parameters. The active-set method in this thesis relies on warm starting to efficiently solve problems with reduced regularization parameters.

4.7.2 Dynamic primal-dual regularization

In HOPDM, a primal-dual interior-point method for quadratic programming, Altman and Gondzio [3] added primal and dual regularization *dynamically*. They worked with KKT matrices of the form

$$\begin{pmatrix} -H & A^T \\ A & \end{pmatrix},$$

where as before $H \equiv D_x + Q$, with Q positive semidefinite and D_x a positive semidefinite and diagonal matrix that changes from iteration to iteration. If a small pivot arose during the factorization phase of the matrix they perturbed the above matrix to be

$$\begin{pmatrix} -H & A^T \\ A & \end{pmatrix} + \begin{pmatrix} -R_p & \\ & R_d \end{pmatrix},$$

where R_p and R_d are positive semidefinite and diagonal. They interpreted the addition of the R_p and R_d matrices as adding regularization to the primal and dual barrier subproblems separately. The primal regularization appeared in the following subproblem:

$$\begin{aligned} \underset{x,s}{\text{minimize}} \quad & c^T x + \tfrac{1}{2} x^T H x + \tfrac{1}{2}(x - x_0)^T R_p (x - x_0) - \mu \sum_{j=1}^{n} (\ln x_j + \ln s_j) \\ \text{subject to} \quad & Ax = b, \quad x + s = u \\ & x, s > 0, \end{aligned}$$

where x_0 is the vector of primal variables at the current iteration and μ is the barrier parameter. The dual regularization appeared in the corresponding dual problem:

$$\begin{aligned} \underset{x,y,z,w}{\text{maximize}} \quad & b^T y - u^T w - \tfrac{1}{2} x^T H x - \tfrac{1}{2}(y - y_0)^T R_d (y - y_0) + \mu \sum_{j=1}^{n} (\ln z_j + \ln w_j) \\ \text{subject to} \quad & A^T y + z - w - Hx = c \\ & x \geq 0, \quad z, w > 0, \end{aligned}$$

where y_0 is the vector of dual variables at the current iteration. They viewed the quadratic terms $(x - x_0)^T R_p (x - x_0)$ and $(y - y_0)^T R_d (y - y_0)$ as penalty terms on iterates far from the primal proximal point x_0 or the dual proximal point y_0. These regularizations were added to improve the condition of certain systems. To avoid solving a significantly perturbed problem, the authors did not wish to use uniform regularization as in (4.8). Instead, they chose the elements of R_p

and R_d dynamically by monitoring the diagonal pivots within their sparse indefinite Cholesky-type factorization.

The resulting method therefore differs from ours in several ways. The regularization changes during each factorization (each iteration of the interior-point method), and because the authors viewed regularization as being added to the primal and dual problem separately, there is no corresponding optimization problem being solved. In addition, their approach requires modification of the method for computing a Cholesky factorization. This conflicts with our goal of using black-box linear solvers.

Chapter 5

The quadratic penalty method

In this chapter we focus on the quadratic penalty method for handling equality constraints. As we will see, the dual regularization algorithm discussed in the previous chapter and the quadratic penalty method are equivalent. Thus, analysis of the quadratic penalty method will provide insight into, and reveal the disadvantages of, the dual regularization algorithm.

A *penalty method* is an algorithm for solving a constrained problem by solving a sequence of unconstrained (or bound constrained) problems whose objectives contain a combination of the original objective and a penalty for violating the constraints.

A *penalty function* $P(x, \rho)$ is a function with the property that $P(x, \rho) = \phi(x)$ if x satisfies the constraints. The additional argument to P indicates that the penalty function also depends on a *penalty parameter* $\rho > 0$. There are many different penalty functions. We only consider the quadratic penalty function, first proposed by Courant [21].

The quadratic penalty method may be applied to general nonconvex problems with nonlinear constraints. However, in this chapter we consider the penalty method applied to the QP

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \phi(x) \\ \text{subject to} & Ax = b \\ & x \geq 0, \end{array} \tag{5.1}$$

where ϕ is a convex quadratic objective and the matrix $A \in \mathbb{R}^{m \times n}$ has full row-rank. For simplicity we consider bounds in the form $x \geq 0$.

With linear constraints $Ax = b$ the quadratic penalty function is given by

$$P(x, \rho) = \phi(x) + \tfrac{1}{2}\rho \left\| Ax - b \right\|_2^2, \tag{5.2}$$

and its gradient and Hessian are

$$\nabla P(x,\rho) = \nabla\phi(x) + \rho A^T(Ax-b),$$
$$\nabla^2 P(x,\rho) = \nabla^2\phi(x) + \rho A^T A.$$

As we discuss the properties of the quadratic penalty method it will be useful to refer to the optimality conditions of problem (5.1). A point $(x^\star, y^\star, z^\star)$ is a solution to (5.1) if and only if

$$Ax^\star = b, \tag{5.3}$$
$$x^\star \geq 0, \tag{5.4}$$
$$\nabla\phi(x^\star) - A^T y^\star = z^\star, \tag{5.5}$$
$$x_j^\star z_j^\star = 0, \quad j = 1,\ldots,n, \tag{5.6}$$
$$z^\star \geq 0. \tag{5.7}$$

5.1 The quadratic penalty method

A method for solving (5.1) is to solve a sequence of quadratic penalty problems of the form

$$\underset{x}{\text{minimize}} \;\; P(x,\rho_k) \;\; \text{subject to} \;\; x \geq 0, \tag{PP$_k$}$$

with a sequence of positive strictly increasing penalty parameters $\{\rho_k\}$. In Section 4.3 we discussed the dual regularized problem

$$\begin{array}{ll} \underset{x,y}{\text{minimize}} & \phi(x) + \frac{1}{2}\mu_k \|y\|_2^2 \\ \text{subject to} & Ax + \mu_k y = b \\ & x \geq 0, \end{array} \tag{5.8}$$

where μ_k is the dual regularization parameter. We considered an algorithm for solving the constrained problem by solving a sequence of problems with strictly decreasing dual regularization parameters $\{\mu_k\}$. If we define $\rho_k \equiv 1/\mu_k$ and eliminate the dual variables y in problem (5.8) via $y = \rho_k(b - Ax)$ we see that this is exactly the quadratic penalty problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \phi(x) + \frac{1}{2}\rho_k \|Ax-b\|_2^2 \\ \text{subject to} & x \geq 0. \end{array}$$

We may define *exact penalization* in a similar manner as exact regularization: given a constrained problem we say that the penalized problem is exact if the solution of the penalized problem is a solution to the original constrained problem. In general the quadratic penalty function does not admit exact penalization. It only converges to a solution of the original constrained problem (5.1) in the limit as $\rho_k \to \infty$. We discuss other penalty functions that are exact in Chapter 6.

Let $(x_k^\star, z_k^\star)$ denote a solution to (PP_k). The optimality conditions of (PP_k) are

$$x_k^\star \geq 0, \tag{5.9}$$

$$\nabla P(x_k^\star, \rho_k) = z_k^\star, \tag{5.10}$$

$$[x_k^\star]_j [z_k^\star]_j = 0, \quad j = 1, \ldots, n, \tag{5.11}$$

$$z_k^\star \geq 0. \tag{5.12}$$

We see that $(x_k^\star, z_k^\star)$ satisfies the optimality conditions (5.4) and (5.6)–(5.7) for the constrained problem (5.1). From (5.10) we have

$$\nabla \phi(x_k^\star) - A^T(\rho_k r_k^\star) = z_k^\star, \tag{5.13}$$

where $r_k^\star = b - Ax_k^\star$. Thus, an estimate y_k of the optimal Lagrange multipliers $y^\star$ in (5.5) may be obtained by taking $y_k = \rho_k r_k^\star$. Furthermore, the point $(x_k^\star, y_k, z_k^\star)$ is an optimal solution for the problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \phi(x) \\ \text{subject to} & Ax = b - r_k^\star \\ & x \geq 0. \end{array} \tag{5.14}$$

Thus, as $\rho_k \to \infty$ we expect $r_k^\star \to 0$ and the point $(x_k^\star, y_k, z_k^\star)$ to become a solution to the constrained problem (5.1). Under fairly loose conditions, it can be shown (see Theorem 27 (Solution of Convex Problems by Exterior Point Algorithms) and Theorem 29 (Exterior Point Dual Convergence) in Fiacco and McCormick [31]) that

$$\lim_{\rho_k \to \infty} x_k^\star = x^\star, \quad \lim_{\rho_k \to \infty} y_k = y^\star, \tag{5.15}$$

and $\{\|r_k^\star\|\}$ is a monotonically decreasing sequence.

The numerical difficulties associated with the quadratic penalty method are well known. To compute an optimal solution to the original problem (5.1) one is required to increase ρ_k without bound. Unfortunately, the Hessian of $P(x, \rho_k)$ becomes increasingly ill-conditioned as ρ_k increases (Murray [79]). Consider an active-set method applied to (PP_k). A search direction $\Delta x_{\mathcal{M}}$ will be calculated by solving $\nabla^2 P(x_k, \rho_k)_{\mathcal{M}} \Delta x_{\mathcal{M}} = -\nabla P(x_k, \rho_k)_{\mathcal{M}}$ or

$$(H_{\mathcal{M}} + \rho_k A_{\mathcal{M}}^T A_{\mathcal{M}}) \Delta x_{\mathcal{M}} = -g_{\mathcal{M}} + \rho_k A_{\mathcal{M}}^T (Ax_k - b), \tag{5.16}$$

where $A_{\mathcal{M}} \in \mathbb{R}^{m \times |\mathcal{M}|}$. When $m < |\mathcal{M}|$, the Hessian of $P(x, \rho_k)$ is dominated by the unbounded rank-deficient matrix $\rho_k A_{\mathcal{M}}^T A_{\mathcal{M}}$.

Gould [50] developed a method for dealing with the ill-conditioned Hessian; a new variable $w = \rho_k(A_{\mathcal{M}} \Delta x_{\mathcal{M}} - Ax_k + b)$ may be defined and the search direction can then be computed

accurately by solving the augmented system

$$\begin{pmatrix} H_{\mathcal{M}} & A_{\mathcal{M}}^T \\ A_{\mathcal{M}} & -\frac{1}{\rho_k} I \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ w \end{pmatrix} = \begin{pmatrix} -g_{\mathcal{M}} \\ r_k \end{pmatrix}. \tag{5.17}$$

Unfortunately, this does not entirely fix the quadratic penalty method. The main issue can be seen by comparing the dual optimality condition of the original problem,

$$\nabla \phi(x^\star) - A^T y^\star = z^\star,$$

with the dual optimality condition satisfied by each of the subproblems:

$$\nabla \phi(x_k^\star) - A^T(\rho_k r_k^\star) = z_k^\star.$$

To approximate the optimal Lagrange multipliers $y^\star$ via $y_k = \rho_k r_k^\star$ we need $\rho_k \to \infty$ to balance $\|r_k^\star\| = \|Ax_k^\star - b\| \to 0$. A remedy for this problem is provided by the augmented Lagrangian method discussed next.

Chapter 6

The bound-constrained Lagrangian method

The quadratic penalty method discussed in the previous chapter has an important deficiency: feasibility is only obtained when ρ_k is increased without bound. In this chapter we consider the bound-constrained Lagrangian (BCL) method, which has the important property that feasibility is obtained after ρ_k has been increased past a finite threshold. The key idea behind the BCL method is to maintain explicit Lagrange multiplier estimates and to minimize an augmented Lagrangian function.

6.1 The BCL method

The BCL method is an algorithm for solving general nonlinear constrained optimization problems. The method is based on the method of multipliers or the augmented Lagrangian method proposed by Hestenes [57] and Powell [89]. Bertsekas [6] presents a comprehensive description of the theory of Lagrange multiplier methods. A globally convergent augmented Lagrangian method with bound-constrained subproblems is described in Conn *et al.* [16]. We follow Friedlander and Saunders [34] in referring to this algorithm as the BCL method.

In this chapter we consider the BCL method applied to QP (5.1). The optimality conditions of problem (5.1) are given in (5.3)–(5.7). The *augmented Lagrangian* associated with the linear constraints $Ax = b$ is

$$L(x, y, \rho) = \phi(x) - y^T(Ax - b) + \frac{1}{2}\rho \|Ax - b\|_2^2 . \tag{6.1}$$

It may be thought of as a modification to the Lagrangian, or as a shifted quadratic penalty function. Note that the augmented Lagrangian becomes the quadratic penalty function when y vanishes ($L(x, 0, \rho) = P(x, \rho)$).

The gradient and Hessian of the augmented Lagrangian (with respect to x) are

$$\begin{aligned}\nabla L(x, y, \rho) &= \nabla\phi(x) - A^T y + \rho A^T(Ax - b),\\ \nabla^2 L(x, y, \rho) &= \nabla^2\phi(x) + \rho A^T A.\end{aligned}$$

The BCL method solves (5.1) by solving a sequence of bound-constrained subproblems

$$\begin{array}{ll}\underset{x}{\text{minimize}} & L(x, y_k, \rho_k)\\ \text{subject to} & x \geq 0,\end{array} \tag{BC$_k$}$$

where the objective is the augmented Lagrangian with parameters y_k and ρ_k fixed for each subproblem. Using the augmented Lagrangian in place of the quadratic penalty function and updating a series of multiplier estimates y_k leads to an algorithm that increases ρ_k only finitely many times.

Let $(x_k^\star, z_k^\star)$ denote an optimal solution of subproblem (BC$_k$). The optimality conditions for (BC$_k$) are

$$x_k^\star \geq 0, \tag{6.2}$$
$$\nabla L(x_k^\star, y_k, \rho_k) = z_k^\star, \tag{6.3}$$
$$[x_k^\star]_j [z_k^\star]_j = 0, \quad j = 1, \ldots, n, \tag{6.4}$$
$$z_k^\star \geq 0. \tag{6.5}$$

Equations (6.2), (6.4), and (6.5) correspond to optimality conditions (5.4), (5.6) and (5.7) for the equality constrained problem (5.1). From (6.3) we have

$$\nabla\phi(x_k^\star) - A^T y_k + \rho_k A^T(Ax_k^\star - b) = z_k^\star. \tag{6.6}$$

If ρ_k is large enough so that $Ax_k^\star \simeq b$, y_k approximates the optimal Lagrange multipliers $y^\star$ for problem (5.1).

Consider an active-set method applied to (BC$_k$). A search direction $\Delta x_{\mathcal{M}}$ will be computed by solving $\nabla^2 L(x_k, y_k, \rho_k)_{\mathcal{M}} \Delta x_{\mathcal{M}} = -\nabla L(x_k, y_k, \rho_k)_{\mathcal{M}}$ or

$$(H_{\mathcal{M}} + \rho_k A_{\mathcal{M}}^T A_{\mathcal{M}})\Delta x_{\mathcal{M}} = -(g_{\mathcal{M}} - A_{\mathcal{M}}^T y_k - \rho_k A_{\mathcal{M}}^T r_k), \tag{6.7}$$

where $r_k = b - Ax_k$. As with the Hessian of the quadratic penalty function, the Hessian of the augmented Lagrangian $L(x_k, y_k, \rho_k)$ will be ill-conditioned when ρ_k is large and $|\mathcal{M}| > m$. The search direction may be computed accurately by introducing a new variable $w = -\rho_k A_{\mathcal{M}} \Delta x_{\mathcal{M}} + y_k + \rho_k r_k$ and solving the augmented system

$$\begin{pmatrix} H_{\mathcal{M}} & A_{\mathcal{M}}^T \\ A_{\mathcal{M}} & -\frac{1}{\rho_k} I \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ -w \end{pmatrix} = \begin{pmatrix} -g_{\mathcal{M}} \\ \frac{y_k}{\rho_k} + r_k \end{pmatrix}. \tag{6.8}$$

Algorithm 5 The BCL algorithm

Require: Starting point x_0, initial multiplier estimate y_0, feasibility tolerance ϵ
  Set initial feasibility and penalty parameters $\epsilon_0 < 1$, $\rho_0 > 1$.
  Set parameters $\alpha, \beta, \tau > 0$ with $\alpha < 1$ and $\tau > 1$.
  `converged` $\leftarrow$ `false`
  repeat
    Solve subproblem (BC_k) to obtain $(x_k^\star, z_k^\star)$
    Compute the residual $r_k^\star = b - Ax_k^\star$
    if $\|r_k^\star\| \leq \max(\epsilon_k, \epsilon)$ **then**
      $x_{k+1} \leftarrow x_k^\star$, $z_{k+1} \leftarrow z_k^\star$ [Update solution estimates]
      $y_{k+1} \leftarrow y_k + \rho_k r_k^\star$ [Update Lagrange multiplier estimates]
      `converged` $\leftarrow \|r_k^\star\| \leq \epsilon$
      $\rho_{k+1} \leftarrow \rho_k$ [Keep penalty parameter]
      $\epsilon_{k+1} \leftarrow \epsilon_k / \rho_{k+1}^\beta$ [Decrease feasibility parameter]
    else
      $x_{k+1} \leftarrow x_k$, $z_{k+1} \leftarrow z_k$ [Keep solution estimates]
      $y_{k+1} \leftarrow y_k$ [Keep Lagrange multiplier estimates]
      $\rho_{k+1} \leftarrow \tau \rho_k$ [Increase penalty parameter]
      $\epsilon_{k+1} \leftarrow \epsilon_0 / \rho_{k+1}^\alpha$ [Increase/decrease feasibility parameter]
    end if
    $k \leftarrow k + 1$
  until `converged`
  return (x_k, y_k, z_k)

After each BCL subproblem is solved, the size of the residual $r_k^\star = b - Ax_k^\star$ is tested against a feasibility tolerance ϵ_k. If $\|r_k^\star\| \leq \epsilon_k$ the iteration is deemed "successful" and the estimates of the Lagrange multipliers y_k are updated. If the iteration is "unsuccessful", y_k is not updated. Instead the penalty parameter ρ_k is increased. Under certain conditions, all iterations are eventually successful and the sequence $\{\rho_k\}$ remains bounded.

The updates to the Lagrange multiplier estimates are chosen based on (6.6), the dual optimality condition for the (BC_k) subproblem. The updated multiplier estimates are defined as $y_{k+1} = y_k + \rho_k r_k^\star$ so that (6.6) becomes

$$\nabla \phi(x_k^\star) - A^T y_{k+1} = z_k^\star. \tag{6.9}$$

This equation has the same form as the dual optimality condition (5.5) for the equality constrained problem (5.1), with y_{k+1} acting as an estimate of the optimal Lagrange multipliers $y^\star$. Thus, the point $(x_k^\star, y_{k+1}, z_k^\star)$ is optimal for the equality constrained problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \phi(x) \\ \text{subject to} & Ax = b - r_k^\star \\ & x \geq 0. \end{array} \tag{6.10}$$

A formal definition of the BCL method is given in Algorithm 5.

The convergence properties of the BCL method are readily evident from its relationship to the quadratic penalty method. At each unsuccessful iteration, the penalty parameter ρ_k is increased. If this happens infinitely often and y_k remains bounded, the BCL method essentially reduces to a quadratic penalty method. Because the quadratic penalty method is a convergent method, by extension so is the BCL method. It is possible to relax the bounded assumption on y_k. Convergence can still be assured if we require the quotient $\|y_k\|/\rho_k$ to converge to zero. Thus, we require that $\|y_k\|$ not increase faster than the penalty parameter. The form of the updates to ϵ_k are critical for this result to hold.

The theory describing the convergence of the BCL algorithm is given in Bertsekas [6], Conn *et al.* [16] and Friedlander [32].

6.2 The dual proximal point method

In this section we take a brief detour to discuss the equivalence between the BCL method and the proximal point method applied to the dual problem. The connection between the dual proximal point method and the augmented Lagrangian method was first noted and analyzed by Rockafellar [91].

The dual of the convex QP (5.1) is

$$\begin{array}{ll} \underset{w,y,z}{\text{maximize}} & b^T y - \frac{1}{2} w^T H w \\ \text{subject to} & c + Hw - A^T y = z \\ & z \geq 0. \end{array} \tag{6.11}$$

When H is positive definite, we may recover the solution of the primal problem (5.1) from the solution of the dual problem (6.11). Optimal $(y^\star, z^\star)$ for (6.11) correspond to the Lagrange multipliers in the optimality conditions (5.5)–(5.7) for problem (5.1).

When the proximal point method (described in Section 4.2) is applied to problem (6.11), a sequence of subproblems of the form

$$\begin{array}{ll} \underset{w,y,z}{\text{maximize}} & b^T y - \frac{1}{2} w^T H w - \frac{1}{2}\mu_k \|y - y_k\|_2^2 \\ \text{subject to} & c + Hw - A^T y = z \\ & z \geq 0 \end{array} \tag{DPPt$_k$}$$

are solved, where $\{\mu_k\}$ is a sequence of decreasing positive parameters, and the solution of (DPPt$_k$) yields the next iterate y_{k+1}. Remarkably, when $\mu_k \equiv 1/\rho_k$, the proximal point subproblem (DPPt$_k$) is the dual of the BCL subproblem (BC$_k$) and y_{k+1} satisfies the BCL update formula $y_{k+1} = y_k + \rho_k(b - Ax_k)$, where x_k is the optimal Lagrange multipliers in problem (DPPt$_k$) (or the primal variables in problem (5.1)).

Thus, we see that performing the BCL algorithm on the primal QP is equivalent to performing the proximal point method on the dual QP. That is, for a given initial y_0 and ρ_0, both algorithms will yield the same sequence of primal-dual iterates $\{x_k, y_k, z_k\}$ (under the assumption that the

subproblems are solved to optimality and all iterates are "successful").

6.3 Bound constrained quadratic subproblems

The BCL subproblem (BC_k) for the QP (1.2) is a bounded constrained QP of the form

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x - y_k^T (Ax - s) + \frac{1}{2}\rho_k \left\| Ax - s \right\|_2^2 \\ \text{subject to} & \ell \leq (x, s) \leq u. \end{array} \tag{BCQP_k}$$

The Hessian of the objective $\phi(x, s)$ in (BCQP_k) is

$$\nabla^2 \phi(x, s) = \begin{pmatrix} H + \rho_k A^T A & -\rho_k A^T \\ -\rho_k A & \rho_k I \end{pmatrix}.$$

We expect the Hessian to be ill-conditioned for large values of ρ_k and if A contains a dense row $A^T A$ will tend to be dense. This motivates us to consider the following equivalent formulation of (BCQP_k):

$$\begin{array}{ll} \underset{x,s,r}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x + y_k^T r + \frac{1}{2}\rho_k r^T r \\ \text{subject to} & Ax - s + r = 0 \\ & \ell \leq (x, s) \leq u, \end{array} \tag{$\text{QP}(\rho_k, y_k)$}$$

where we optimize the residual r of the linear equality constraints. The gradient and Hessian of the objective $\phi(x, s, r)$ in ($\text{QP}(\rho_k, y_k)$) are

$$\nabla \phi(x, s, r) = \begin{pmatrix} c + Hx \\ 0 \\ y_k + \rho_k r \end{pmatrix} \quad \text{and} \quad \nabla^2 \phi(x, s, r) = \begin{pmatrix} H & & \\ & 0 & \\ & & \rho_k I \end{pmatrix}.$$

Observe that because the Hessian is positive semidefinite, a subproblem solver would require an inertia-controlling method to prevent the KKT matrices from becoming singular.

6.4 Regularized BCL for quadratic programming

We would like to ensure that the KKT systems are nonsingular for any choice of active set. This would eliminate the need for an inertia-controlling method and KKT repair. We have already discussed a method for ensuring that a semidefinite Hessian is nonsingular: primal regularization. Thus, we apply primal regularization to ($\text{QP}(\rho_k, y_k)$) to obtain the primal regularized BCL subproblem

$$\begin{array}{ll} \underset{x,s,r}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x + \frac{1}{2}\delta_k x^T x + \frac{1}{2}\delta_k s^T s + y_k^T r + \frac{1}{2}\rho_k r^T r \\ \text{subject to} & Ax - s + r = 0 \\ & \ell \leq (x, s) \leq u. \end{array} \tag{$\text{QP}(\rho_k, y_k, \delta_k)$}$$

The gradient and Hessian of the objective $\phi(x, s, r)$ in (QP(ρ_k, y_k, δ_k)) are

$$\nabla\phi(x,s,r) = \begin{pmatrix} c + (H + \delta_k I)x \\ \delta_k s \\ y_k + \rho_k r \end{pmatrix}, \quad \nabla^2\phi(x,s,r) = \begin{pmatrix} H + \delta_k I & & \\ & \delta_k I & \\ & & \rho_k I \end{pmatrix}.$$

Note that with regularization the Hessian $\nabla^2\phi(x, s, r)$ is strictly positive definite.

Problem (QP(ρ_k, y_k, δ_k)) has several appealing properties. Because of the regularization the objective is strictly convex. The linear equality constraints are used only to avoid forming the matrix $A^T A$. As a result, if $\ell \leq u$ the problem is always feasible. Therefore, given an (x, s) satisfying $\ell \leq (x, s) \leq u$ it is trivial to find a feasible point (x, s, r): simply choose $r = -Ax + s$.

Consider an active-set algorithm applied to (QP(ρ_k, y_k, δ_k)). Suppose the algorithm is at a point (x, s, r) satisfying $Ax - s + r = 0$ with sets $\mathcal{M}$ and $\mathcal{N}$ of moving and nonmoving variables. A search direction $p_{\mathcal{M}} = (\Delta x_{\mathcal{M}}, \Delta s_{\mathcal{M}}, \Delta r)$ is found by solving the KKT system

$$\begin{pmatrix} H_{\mathcal{M}} + \delta_k I & & & A_{\mathcal{M}}^T \\ & \delta_k I & & -I_{\mathcal{M}}^T \\ & & \rho_k I & I \\ A_{\mathcal{M}} & -I_{\mathcal{M}} & I & \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ \Delta s_{\mathcal{M}} \\ \Delta r \\ -y \end{pmatrix} = \begin{pmatrix} -g_x \\ -g_s \\ -g_r \\ 0 \end{pmatrix}, \tag{6.12}$$

where g_r is the gradient of $\phi(x, s, r)$ with respect to r. We can use the equation $\rho_k \Delta r - y = -g_r$ to eliminate Δr in the above system to yield the following KKT system:

$$\begin{pmatrix} H_{\mathcal{M}} + \delta_k I & & A_{\mathcal{M}}^T \\ & \delta_k I & -I_{\mathcal{M}}^T \\ A_{\mathcal{M}} & -I_{\mathcal{M}} & -\mu_k I \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ \Delta s_{\mathcal{M}} \\ -y \end{pmatrix} = \begin{pmatrix} -g_x \\ -g_s \\ \mu_k y_k + r \end{pmatrix}, \tag{6.13}$$

where $\mu_k \equiv 1/\rho_k$. With the exception of the last component in the right-hand side, (6.13) is identical to the KKT system (4.7) for the primal-dual regularized problem (QP(δ, μ)). Thus, from Theorem 2 we have that the matrix in (6.13) is nonsingular, and since Δr is uniquely defined in terms of y, the matrix in (6.12) is also nonsingular.

6.4.1 The regularized BCL algorithm

With the properties of (QP(ρ_k, y_k, δ_k)) in mind, we now present a quadratic programming algorithm that uses the BCL method to satisfy the linear constraints, with primal regularization to ensure that the KKT systems are nonsingular for any active set. This algorithm is almost a direct application of Algorithm 5—the only differences being how the regularization parameter δ_k is updated, and an additional stopping criterion

$$\left\| \nabla\phi(x,s) - \begin{pmatrix} A^T \\ -I \end{pmatrix} y - z_l + z_u \right\|_\infty \leq \omega \tag{6.14}$$

based on the dual optimality condition of the original QP (1.2). The algorithm follows.

Algorithm 6 The regularized BCL algorithm

Require: Starting point (x_0, s_0), initial estimate y_0, feasibility tolerance ϵ, optimality tolerance ω
  Set initial feasibility, penalty, and regularization parameters $\epsilon_0, \rho_0, \delta_0$
  Set parameters $\alpha, \beta, \tau > 0$ with $\alpha < 1$ and $\tau > 1$
  `converged` $\leftarrow$ `false`
  repeat
    Solve subproblem (QP(ρ_k, y_k, δ_k)) to obtain $(x_k^\star, s_k^\star, z_{lk}^\star, z_{uk}^\star)$
    Compute the residual $r_k^\star = s_k^\star - Ax_k^\star$
    if $\|r_k^\star\| \leq \max(\epsilon_k, \epsilon)$ **then**
      $x_{k+1} \leftarrow x_k^\star$, $s_{k+1} \leftarrow s_k^\star$ [Update solution estimates]
      $z_{lk+1} \leftarrow z_{lk}^\star$, $z_{uk+1} \leftarrow z_{uk}^\star$
      $y_{k+1} \leftarrow y_k + \rho_k r_k^\star$ [Update Lagrange multiplier estimates]
      `converged` $\leftarrow \left\| \nabla\phi(x_k^\star, s_k^\star) - \begin{pmatrix} A^T \\ -I \end{pmatrix} y_k^\star - z_{lk}^\star + z_{uk}^\star \right\|_\infty \leq \omega$ and $\|r_k^\star\| \leq \epsilon$
      $\rho_{k+1} \leftarrow \rho_k$ [Keep penalty parameter]
      $\epsilon_{k+1} \leftarrow \epsilon_k / \rho_{k+1}^\beta$ [Decrease feasibility parameter]
      $\delta_{k+1} \leftarrow \delta_k / \tau$ [Decrease regularization parameter]
    else
      $x_{k+1} \leftarrow x_k$, $s_{k+1} \leftarrow s_k$ [Keep solution estimates]
      $z_{lk+1} \leftarrow z_{lk}$, $z_{uk+1} \leftarrow z_{uk}$
      $y_{k+1} \leftarrow y_k$ [Keep Lagrange multiplier estimates]
      $\rho_{k+1} \leftarrow \tau\rho_k$ [Increase penalty parameter]
      $\epsilon_{k+1} \leftarrow \epsilon_0 / \rho_{k+1}^\alpha$ [Increase/decrease feasibility parameter]
    end if
    $k \leftarrow k + 1$
  until `converged`
  return $(x_k, s_k, y_k, z_{lk}, z_{uk})$

6.5 Previous use of the BCL algorithm

The BCL algorithm was developed for nonconvex optimization problems with nonlinear constraints [6] and implemented in the large-scale LANCELOT solver [16, 18, 65]. LANCELOT solves the nonlinear optimization problem

$$\begin{aligned} (\text{locally}) \underset{x}{\text{minimize}} \quad & \varphi(x) \\ \text{subject to} \quad & c(x) = 0, \quad \ell \leq x \leq u \end{aligned} \tag{6.15}$$

by approximately solving a sequence of nonconvex bound-constrained subproblems

$$\begin{aligned} (\text{locally}) \underset{x}{\text{minimize}} \quad & \varphi(x) - y_k^T c(x) + \tfrac{1}{2}\rho_k \|c(x)\|_2^2 \\ \text{subject to} \quad & \ell \leq x \leq u. \end{aligned} \tag{6.16}$$

Here $c(x) = 0$ includes linear and nonlinear constraints. LANCELOT contains an active-set subproblem solver, SBMIN, that computes an approximate solution to (6.16). Because problem (6.16)

is nonconvex, a trust-region strategy is required. SBMIN uses a gradient-projection approach in conjunction with this trust-region strategy to make multiple changes to the active set at each iteration. Conn *et al.* [17] show how to handle *linear* constraints separately in (6.15) by generalizing (6.16) to include linear constraints as well as bounds. (However, this approach has not been implemented.)

By default, SBMIN uses the conjugate-gradient method (with a band preconditioner) to solve sparse systems of the form

$$H_{\mathcal{M}} \Delta x_{\mathcal{M}} = -g_{\mathcal{M}},$$

where $H_{\mathcal{M}}$ is a submatrix of the Hessian of the augmented Lagrangian objective function in (6.16). In addition, LANCELOT contains options to use the sparse direct solvers MA27 or MA57 [28] to factorize $H_{\mathcal{M}}$ or $H_{\mathcal{M}} + E$, where E is a diagonal modification to make the system positive definite. As $\mathcal{M}$ changes, Schur-complement updates are made in order to reuse factors.

The problem (and hence the method) in this thesis is more specialized than that considered by LANCELOT. Because we consider only convex QPs and apply primal regularization, the subproblems are strictly convex. Thus, a trust-region method is not required to solve the subproblem. In addition, we prefer to use direct methods to solve the (possibly ill-conditioned) systems that arise during the course of an optimization algorithm.

6.6 The proximal method of multipliers

A method similar to the regularized BCL algorithm was proposed by Rockafellar [91] for general convex optimization problems. This method, called the proximal method of multipliers, adds a proximal point term (instead of primal regularization) to the augmented Lagrangian function. Sequences of iterates $\{x_k\}$ and Lagrange multiplier estimates $\{y_k\}$ are obtained by solving subproblems of the form

$$\begin{array}{ll} \underset{x}{\text{minimize}} & L(x, y_k, \rho_k) + \frac{1}{2}\mu_k \left\| x - x_k \right\|_2^2 \\ & x \geq 0, \end{array} \qquad (\text{PrMoM}_k)$$

where $\mu_k \equiv 1/\rho_k$ (unlike the regularized BCL algorithm a separate parameter δ_k is not maintained). The solution of subproblem (PrMoM_k) becomes x_{k+1}. The Lagrange multiplier estimate is updated via $y_{k+1} = y_k + \rho_k(b - Ax_{k+1})$. Rockafellar proves the proximal method of multipliers converges to a primal and dual optimal solution of the original convex problem as long as the iterates x_k and y_k remain bounded. These quantities are bounded when there exist optimal solutions to the primal and dual problems with zero duality gap.

Wright [112] applied the proximal method of multipliers to LPs. As in the regularized BCL algorithm, the bound-constrained QP subproblems solved at each iteration are strictly convex. The subproblems are solved using a method that merges the gradient-projection algorithm with the conjugate-gradient method. Gradient-projection steps are taken until the active set remains unchanged for two successive iterations. At this point the method switches to performing conjugate-gradient to approximately solve $H_{\mathcal{M}} \Delta x_{\mathcal{M}} = -g_{\mathcal{M}}$ for the current active set.

Wright's method for solving the subproblem differs from the approach taken in this thesis (as

described in the next chapter). However, we recognize the advantage of using the gradient-projection method to incorporate many changes in the active set at each iteration. We discuss a gradient-projection method for solving the QP subproblems in Section 7.2, and compare the performance of the regularized BCL algorithm to the proximal method of multipliers in Section 11.3.3.

Chapter 7

The regularized QP subproblem

In this chapter we present two algorithms for solving the regularized QP subproblems that appear in the previous chapters. The first algorithm is similar to a standard active-set algorithm for linearly constrained QPs. The second is a gradient-projection algorithm for QPs with bound constraints.

Several different subproblems appeared in the previous chapters. To unify notation we will consider the bound-constrained subproblem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & c^T x + \frac{1}{2} x^T H x + \hat{y}^T (b - Ax) + \frac{1}{2} \left\| D_\rho^{1/2} (b - Ax) \right\|_2^2 \\ \text{subject to} & \ell \leq x \leq u \end{array} \tag{7.1}$$

and the equivalent linearly constrained subproblem

$$\begin{array}{ll} \underset{x,r}{\text{minimize}} & \phi(x, r) \equiv c^T x + \frac{1}{2} x^T H x + \hat{y}^T r + \frac{1}{2} r^T D_\rho r \\ \text{subject to} & Ax + r = b \\ & \ell \leq x \leq u, \end{array} \tag{7.2}$$

where A has any shape and need not have full rank, and D_ρ is positive definite and diagonal. We assume that $\ell < u$ (variables with $\ell_j = u_j$ may be eliminated). In this chapter, because we are concerned with the regularized QP subproblem, we assume that H is strictly positive definite. That is, regularization has already been applied and included in the Hessian. The subproblems discussed in the previous chapters may be transformed into either of the above forms.

The necessary and sufficient conditions for a point $(x^\star, r^\star, y^\star, z_l^\star, z_u^\star)$ to be optimal for problem (7.2) are

$$Ax^\star + r^\star = b, \tag{7.3}$$
$$\ell \leq x^\star \leq u, \tag{7.4}$$
$$c + Hx^\star = A^T y^\star + z_l^\star - z_u^\star, \tag{7.5}$$
$$\hat{y} + D_\rho r^\star = y^\star, \tag{7.6}$$
$$z_l^{\star T}(x^\star - \ell) = 0, \quad z_u^{\star T}(u - x^\star) = 0, \tag{7.7}$$
$$z_l^\star, z_u^\star \geq 0, \tag{7.8}$$

where $y^\star$ are Lagrange multipliers for the equality constraints and $z_l^\star$ and $z_u^\star$ are Lagrange multipliers for the inequality constraints.

7.1 An algorithm for solving the regularized QP subproblem

We present a primal active-set algorithm for solving the regularized subproblem (7.2). The algorithm may be viewed as an active-set method for solving the bound-constrained problem (7.1) where the search direction is computed by solving an augmented system (similar to systems (5.17) and (6.8)), or it may be viewed as a method for solving (7.2) that always satisfies the linear constraints $Ax + r = b$.

The algorithm follows the general structure of active-set methods described in Chapter 2. A prediction of the active set of inequality constraints at the optimal solution is defined in terms of the sets $\mathcal{M}$ and $\mathcal{N}$ of moving and nonmoving variables. The set of nonmoving variables is further partitioned into sets $\mathcal{L}$ and $\mathcal{U}$, with the property that if $j \in \mathcal{L}$, $x_j = \ell_j$ and if $j \in \mathcal{U}$, $x_j = u_j$. If x_j is a moving variable, it usually lies strictly between its bounds ($\ell_j < x_j < u_j$). However, at a degenerate point there will be indices $j \in \mathcal{M}$ for which $x_j = \ell_j$ or $x_j = u_j$. This is acceptable.

A feature of our approach is that (in contrast to standard active-set methods) we do not need to have nonmoving variables fixed on artificial bounds. This is because variables not on their bounds may be treated as moving variables without risk of the KKT systems becoming singular.

Note that at every iteration all variables in the vector r are allowed to move. However, by convention, the set $\mathcal{M}$ only contains indices of moving variables in the vector x.

Given an initial feasible x ($\ell \leq x \leq u$) the residual $r = b - Ax$ is computed. From this feasible point (x, r) the algorithm evolves the primal decision variables via the iteration $(x, r) = (x, r) + \alpha p$. The search direction p is defined by the solution of the problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & g_\phi^T p + \frac{1}{2} p^T H_\phi p \\ \text{subject to} & \begin{pmatrix} A & I \end{pmatrix} p = 0, \quad p_{\mathcal{N}} = 0, \end{array} \tag{7.9}$$

where g_ϕ and H_ϕ are the gradient and Hessian of the objective $\phi(x, r)$ respectively. If the search direction is partitioned into $p = (\Delta x_{\mathcal{M}}, \Delta x_{\mathcal{N}}, \Delta r)$, the constraints in the problem above keep $\Delta x_{\mathcal{N}} = 0$,

and the remaining components $\Delta x_{\mathcal{M}}$ and Δr are found by solving the KKT system

$$\begin{pmatrix} H_{\mathcal{M}} & & A_{\mathcal{M}}^T \\ & D_\rho & I \\ A_{\mathcal{M}} & I & \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ \Delta r \\ -y \end{pmatrix} = \begin{pmatrix} -g_{\mathcal{M}} \\ -g_r \\ 0 \end{pmatrix}, \tag{7.10}$$

where $g_{\mathcal{M}} = [c + Hx]_{\mathcal{M}}$ and $g_r = \hat{y} + D_\rho r$. Note that p is nonzero if the current point (x, r) is not a subspace minimizer. In the algorithm statement (Algorithm 7), the logical variable `stationary` guards against this circumstance. In the following discussion we assume that p, defined by (7.10), is nonzero.

The variable Δr may be eliminated from system (7.10) via the relation $\Delta r = D_\rho^{-1}(y - g_r)$ and the smaller system

$$\begin{pmatrix} H_{\mathcal{M}} & A_{\mathcal{M}}^T \\ A_{\mathcal{M}} & -D_\rho^{-1} \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ -y \end{pmatrix} = \begin{pmatrix} -g_{\mathcal{M}} \\ D_\rho^{-1} g_r \end{pmatrix} \tag{7.11}$$

solved. For numerical reasons we often wish to use a previous estimate of the Lagrange multipliers y_0 and solve for $\Delta y = y - y_0$ instead of y itself. Thus (7.11) becomes

$$\begin{pmatrix} H_{\mathcal{M}} & A_{\mathcal{M}}^T \\ A_{\mathcal{M}} & -D_\rho^{-1} \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ -\Delta y \end{pmatrix} = \begin{pmatrix} -g_{\mathcal{M}} + A_{\mathcal{M}}^T y_0 \\ D_\rho^{-1}(g_r - y_0) \end{pmatrix}. \tag{7.12}$$

We will continue to use (7.10) in the analysis of the algorithm. Lemma 1 proves that if p is defined by (7.10), it is a descent direction ($g_\phi^T p < 0$) for the objective, regardless of the choice of $\mathcal{M}$ and $\mathcal{N}$.

Lemma 1. *For any $\mathcal{M}$ and $\mathcal{N}$, if $p = (\Delta x_{\mathcal{M}}, \Delta x_{\mathcal{N}}, \Delta r)$ is defined by (7.10), it is a descent direction for the objective $\phi(x, r)$.*

Proof. If $\Delta x_{\mathcal{N}} = 0$, $g_\phi^T p = g_{\mathcal{M}}^T \Delta x_{\mathcal{M}} + g_r^T \Delta r$. From Theorem 2 we know that p is uniquely defined because the matrix in (7.10) is nonsingular for any choice of $\mathcal{M}$. If we multiply system (7.10) by $(\Delta x_{\mathcal{M}}^T, \Delta r^T, 0)$, we have

$$\begin{aligned} g_{\mathcal{M}}^T \Delta x_{\mathcal{M}} + g_r^T \Delta r &= -\Delta x_{\mathcal{M}}^T H_{\mathcal{M}} \Delta x_{\mathcal{M}} - \Delta r^T D_\rho \Delta r + y^T (A_{\mathcal{M}} \Delta x_{\mathcal{M}} + \Delta r) \\ &= -\Delta x_{\mathcal{M}}^T H_{\mathcal{M}} \Delta x_{\mathcal{M}} - \Delta r^T D_\rho \Delta r, \end{aligned}$$

where the last equality follows from $A_{\mathcal{M}} \Delta x_{\mathcal{M}} + \Delta r = 0$. This quantity is strictly negative because $H_{\mathcal{M}} \succ 0$, $D_\rho \succ 0$, and $\Delta x_{\mathcal{M}}$ and Δr cannot both be zero without p being zero. □

If the current point is not degenerate, it is possible to take a positive feasible step. The following lemma states that, in this case, a strict decrease in the objective will be made.

Lemma 2. *If p is defined by (7.10) and the current point x is not degenerate, there is some $\alpha > 0$ such that $\ell \leq x + \alpha p \leq u$ and $\phi(x + \alpha p) < \phi(x)$.*

Proof. From Lemma 1 we have $g_\phi^T p < 0$, and because H_ϕ is positive definite we also have $p^T H_\phi p > 0$.

Therefore,

$$\phi(x+\alpha p) = \phi(x) + \alpha g_\phi^T p + \tfrac{1}{2}\alpha^2 p^T H_\phi p < \phi(x)$$

and $\ell \le x + \alpha p \le u$ for all $\alpha > 0$ sufficiently small. Since x is not a degenerate point there is no risk of α being zero. □

The step length α that minimizes the objective along the direction p, while retaining feasibility, is calculated as discussed in Section 2.6. After a step is taken the new iterate (x, r) again satisfies $Ax + r = b$.

The algorithm distinguishes between intermediate iterates and subspace minimizers. An iterate (x, r) is a subspace minimizer if it is a solution to the problem

$$\begin{array}{lll} \underset{x,r}{\text{minimize}} & \phi(x) & \\ \text{subject to} & Ax + r = b & \\ & x_{\mathcal{L}} = \ell_{\mathcal{L}}, \quad x_{\mathcal{U}} = u_{\mathcal{U}} & \\ & \ell_{\mathcal{M}} \le x_{\mathcal{M}} \le u_{\mathcal{M}}. & \end{array} \tag{7.13}$$

Here the subspace is defined by the sets $\mathcal{M}$, $\mathcal{L}$ and $\mathcal{U}$. Subspace minimizers satisfy equations (7.3)–(7.7). A subspace minimizer is reached when the unit step ($\alpha = 1$) is taken. If an inequality constraint prevents the unit step from being taken, we say that (x, r) is an intermediate iterate. At an intermediate iterate a moving variable x_j becomes fixed on a bound and j is moved into $\mathcal{N}$.

At a subspace minimizer, $z = g - A^T y$ is calculated. The signs of the vector components z_j are tested to determine if the iterate is optimal. The algorithm is at an optimal point when $z_j \ge 0$ for all nonmoving variables with $x_j = \ell_j$, and $z_j \le 0$ for all nonmoving variables with $x_j = u_j$. The Lagrange multipliers z_l and z_u may be defined in terms of z as

$$[z_l]_j = \begin{cases} 0 & j \in \mathcal{U} \\ 0 & j \in \mathcal{M} \\ z_j & j \in \mathcal{L} \end{cases}, \quad [z_u]_j = \begin{cases} -z_j & j \in \mathcal{U} \\ 0 & j \in \mathcal{M} \\ 0 & j \in \mathcal{L}. \end{cases} \tag{7.14}$$

If a subspace minimizer is not optimal, a current nonmoving variable can be made into a moving variable and a feasible descent direction found. Lemma 3 proves that the search direction computed on the next iteration is strictly feasible with respect to this new moving variable. This ensures that the new moving variable will not immediately be fixed back on the same bound.

Lemma 3. *Suppose the current point (x, y, z) with $z = g - A^T y$ is a subspace minimizer and there is a nonmoving variable x_j on its lower (upper) bound with $z_j < 0$ ($z_j > 0$). Then x_j can be freed from its bound and Δx_j, the jth component of the next search direction, will be positive (negative).*

Proof. After the variable x_j is freed, we will have new sets $\mathcal{M} \cup \{j\}$ and $\mathcal{N} \backslash \{j\}$ of moving and

nonmoving variables. The next search direction will be defined by the system

$$\begin{pmatrix} H_{\mathcal{M}} & h_j & & A_{\mathcal{M}}^T \\ h_j^T & h_{jj} & & a_j^T \\ & & D_\rho & I \\ A_{\mathcal{M}} & a_j & I & \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ \Delta x_j \\ \Delta r \\ -y^+ \end{pmatrix} = \begin{pmatrix} -g_{\mathcal{M}} \\ -g_j \\ -g_r \\ 0 \end{pmatrix}, \tag{7.15}$$

where h_j is the jth column of the Hessian (restricted to the moving variables), h_{jj} is the jth diagonal of the Hessian, and a_j is the jth column of A.

We begin by showing that the search direction is nonzero. Because the point (x, y, z) is a subspace minimizer we have

$$\begin{pmatrix} g_{\mathcal{M}} \\ g_r \end{pmatrix} = \begin{pmatrix} A_{\mathcal{M}}^T \\ I \end{pmatrix} y. \tag{7.16}$$

From Theorem 2 we have that the system in (7.15) is nonsingular. Thus, if the right-hand side in (7.15) is zero, g_r, g_j and p are zero. Equation (7.16) would then imply that $y = 0$, and thus the quantity $z_j = g_j - a_j^T y = 0$. This contradicts the hypothesis of the lemma. If the right-hand side in (7.15) is nonzero but $p = (\Delta x_{\mathcal{M}}, \Delta x_j, \Delta r) = 0$, y^+ must be nonzero. When $\Delta r = 0$, the third row of (7.15) implies $y^+ = g_r$. From (7.16) we conclude that $y^+ = y$. The second row of (7.15) implies that $a_j^T y = g_j$. Again, this contradicts the hypothesis of the lemma. Thus, we conclude that $p \neq 0$.

From Lemma 1 we have $g^T p < 0$ where

$$g^T p = g_{\mathcal{M}}^T \Delta x_{\mathcal{M}} + g_j \Delta x_j + g_r^T \Delta r. \tag{7.17}$$

From (7.16) and (7.15) we have

$$\Delta x_{\mathcal{M}}^T g_{\mathcal{M}} + \Delta r^T g_r = y^T (A_{\mathcal{M}} \Delta x_{\mathcal{M}} + \Delta r) = -a_j^T y \Delta x_j.$$

Thus, we see that

$$g^T p = (g_j - a_j^T y) \Delta x_j.$$

The quantity $g_j - a_j^T y$ is simply z_j, so $\Delta x_j = g^T p / z_j$. Thus, when x_j is on its lower (upper) bound and $z_j < 0$ ($z_j > 0$) we have $\Delta x_j > 0$ ($\Delta x_j < 0$). □

The following lemma proves that even in the presence of degeneracy the algorithm can always take a positive step and hence obtain a strict decrease in the objective function after at most n degenerate steps.

Lemma 4. *After at most n degenerate steps the algorithm either terminates or makes strict decrease in the objective function.*

Proof. If x_k is a degenerate point there is some $j \in \mathcal{M}$ with $[x_k]_j = \ell_j$ or $[x_k]_j = u_j$. Thus, the maximum feasible step α_F may be zero. Suppose $[x_k]_j$ is the binding variable and j is removed from $\mathcal{M}$ and added to $\mathcal{N}$ and $x_{k+1} = x_k$. We can take at most n degenerate steps with

Algorithm 7 Algorithm for the regularized QP subproblem

Require: $c, H, \hat{y}, D_\rho, A, b, \ell, u, x_0, y_0, \mathcal{M}_0, \mathcal{N}_0$
  $k \leftarrow 0$
  $r_0 \leftarrow b - Ax_0$
  $g_0 \leftarrow c + Hx_0$
  $z_0 \leftarrow g_0 - A^T y_0$
  $(\texttt{optimal}, z_{l0}, z_{u0}) \leftarrow \texttt{testOptimality}(z_0, \mathcal{M}_0, \mathcal{N}_0)$
  $\texttt{stationary} \leftarrow \texttt{false}$
  while not `optimal` **do**
    $g_k \leftarrow c + Hx_k$
    if `stationary` **then**
      $z_k \leftarrow g_k - A^T y_k$
      $(\texttt{optimal}, z_{lk}, z_{uk}) \leftarrow \texttt{testOptimality}(z_k, \mathcal{M}_k, \mathcal{N}_k)$
      if not `optimal` **then**
        Choose a variable $[x_k]_j$ with corresponding $[z_k]_j$ of incorrect sign.
        $\mathcal{M}_{k+1} \leftarrow \mathcal{M}_k \cup \{j\},\ \mathcal{N}_{k+1} \leftarrow \mathcal{N}_k \backslash \{j\}$
        $\texttt{stationary} \leftarrow \texttt{false}$
      end if
    else
      Solve (7.12) for the search direction $p = (\Delta x, \Delta r)$ and Δy.
      Compute step length α. (See Section 2.6 for details.)
      if $\alpha < 1$ **then**
        Fix the moving variable $[x_k]_j$ with the binding constraint on its bound
        $\mathcal{N}_{k+1} \leftarrow \mathcal{N}_k \cup \{j\},\ \mathcal{M}_{k+1} \leftarrow \mathcal{M}_k \backslash \{j\}$
      else
        $\texttt{stationary} \leftarrow \texttt{true}$
      end if
      $x_{k+1} \leftarrow x_k + \alpha \Delta x$
      $r_{k+1} \leftarrow r_k + \alpha \Delta r$
      $y_{k+1} \leftarrow y_k + \alpha \Delta y$
    end if
    $k \leftarrow k + 1$
  end while
  return $(x_k, r_k, y_k, z_{lk}, z_{uk})$

$x_k = x_{k+1} = \cdots = x_{k+n}$ before $\mathcal{M} = \emptyset$. At this point x_{k+n} is a subspace minimizer. If z_l and z_u are nonnegative, x_{k+n} is optimal and the algorithm terminates. Otherwise, Lemma 3 shows that a nonmoving variable x_i can be freed from its bound and the resulting search direction $p = (\Delta x_i, \Delta r)$ will not place x_i back on its bound. Because there is only one moving variable x_i, Lemma 3 also implies that there is a feasible step $\alpha_F > 0$. Hence, a strict decrease will be made in the objective function. □

A formal definition of the algorithm for solving a regularized QP subproblem is presented in Algorithm 7. The function `testOptimality` in Algorithm 7 constructs the Lagrange multipliers z_l and z_u and sets the flag `optimal` if both are nonnegative. Theorem 3 proves that the algorithm terminates with a solution in a finite number of iterations.

Theorem 3. *Algorithm 7 converges to a solution of problem* (7.2) *in a finite number of iterations.*

Proof. If the current point x_k is a subspace minimizer (with respect to $\mathcal{M}$ and $\mathcal{N}$) and the vectors z_{lk} and z_{uk} are nonnegative, x_k is a solution to problem (7.2). Otherwise there is at least one negative Lagrange multiplier associated with a nonmoving variable. Lemma 3 shows that the step p_{k+1}, computed after x_j is freed from its bound, will be a descent direction for the objective. If it is possible to take a step $\alpha_{k+1} > 0$ then $\phi(x_k + \alpha_{k+1}p_{k+1}) < \phi(x_k)$. Otherwise Lemma 4 shows that after at most n steps the algorithm terminates or a strict decrease in the objective function is made. It follows that the algorithm can never return to the same sets $\mathcal{M}$ and $\mathcal{N}$, because subsequent iterates have objective values that are lower than the minimizer in the subspace defined by $\mathcal{M}$ and $\mathcal{N}$.

The algorithm encounters a subspace minimizer at least every n iterations. Thus, the algorithm must find the minimum of the objective in the subspace defined by $\mathcal{M}$ and $\mathcal{N}$ periodically and having done so, it never returns to that particular $\mathcal{M}$ and $\mathcal{N}$ again. It follows that since there are at most 3^n different possibilities for $\mathcal{M}$ and $\mathcal{N} = \mathcal{L} \cup \mathcal{U}$, the algorithm cannot iterate forever. Eventually it encounters a minimizer in the current subspace that satisfies the optimality conditions for problem (7.2) and it terminates with a solution. □

7.2 A gradient-projection method for solving the regularized subproblem

As described in Section 2.7, the gradient-projection method is an active-set algorithm particularly suited to problems such as (7.1) with simple bound constraints. The advantage of the gradient-projection algorithm over traditional active-set methods like the one described above, is that it allows the active set to change rapidly from one iteration to the next. This is a particularly desirable property for an algorithm designed to solve large-scale QPs.

A gradient-projection algorithm for solving the bound-constrained convex QP

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \phi(x) \equiv c^T x + \frac{1}{2} x^T H x \\ \text{subject to} & \ell \leq x \leq u \end{array} \tag{7.18}$$

is outlined in Algorithm 8. A detailed treatment of the gradient-projection algorithm is given by Nocedal and Wright [82].

A prototype implementation of the gradient-projection method was developed for solving the regularized subproblem (7.2). Unfortunately, when the optimal solution $x^\star$ for (7.18) is degenerate the optimal active set may oscillate, with constraints repeatedly entering and leaving the active set on successive iterations. This can prevent the algorithm from terminating. The prototype implementation exhibited this problem on several QP test problems. This undesirable behavior of the gradient-projection method is well known and various methods have been proposed to prevent it [14, 19]. Further work is needed to implement a method to control oscillation in the active set near an optimal solution.

Algorithm 8 The gradient-projection algorithm for problem (7.18)

Require: Feasible point x_0 ($\ell \leq x_0 \leq u$)
  $k \leftarrow 0$
  repeat
    if x_k satisfies the optimality conditions of (7.18) **then**
      `converged` $\leftarrow$ `true`
      $x^\star \leftarrow x_k$
    else
      $g_k \leftarrow c + Hx_k$
      Let $x(\alpha) = P(x_k - \alpha g_k, \ell, u)$, where $P(x, \ell, u)$ is the projection of the point x onto the feasible set $\{x \mid \ell \leq x \leq u\}$.
      Compute the Cauchy point x^C defined as the first local minimizer of the problem

$$\underset{\alpha}{\text{minimize}} \quad \phi(x(\alpha))$$

      Compute the active set at the Cauchy point $\mathcal{A}(x^C) = \{j \mid x_j^C = \ell_j \text{ or } x_j^C = u_j\}$
      Compute an approximate solution x^+ to the problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \phi(x) \equiv c^T x + \frac{1}{2} x^T H x \\ \text{subject to} & x_j = x_j^C, \quad j \in \mathcal{A}(x^C) \\ & \ell_j \leq x_j \leq u_j, \quad j \notin \mathcal{A}(x^C) \end{array}$$

      such that $\phi(x^+) \leq \phi(x^C)$.
      $x_{k+1} \leftarrow x^+$
    end if
    $k \leftarrow k + 1$
  until `converged`
  return $x^\star$

Chapter 8

Least squares problems

The method of least squares is a key computational tool in many fields, such as statistics, geodetics, signal processing and control [1]. In this chapter we discuss how sparse linear least squares problems may be solved efficiently and reliably using the algorithms presented in the previous chapters.

8.1 Ordinary least squares

In many applications, least squares problems arise when a vector $b \in \mathbb{R}^m$ of observations is related to an unknown vector $x \in \mathbb{R}^n$ of parameters via the linear relation

$$Ax + r = b, \tag{8.1}$$

where $A \in \mathbb{R}^{m \times n}$ is a known matrix and r is a vector of random errors. If these random errors have zero mean, are uncorrelated, and have the same variance (*i.e.* the covariance matrix for r is $\sigma^2 I$), and if A has full column-rank, the best linear unbiased estimator of the parameters x is given by the solution of the *ordinary* least squares problem

$$\underset{x}{\text{minimize}} \, \|b - Ax\|_2 \,. \tag{LS(A,b)}$$

In this chapter we consider ordinary least squares problems where A is large and sparse. Problem (LS(A,b)) is equivalent to the unconstrained convex QP

$$\underset{x}{\text{minimize}} \; \phi(x) \equiv \frac{1}{2} x^T A^T A x - b^T A x. \tag{8.2}$$

The regularized BCL algorithm (Algorithm 6) and the primal-dual regularized algorithm (Algorithm 4) for convex quadratic programming could be applied directly to this problem. These algorithms would seek a point that satisfies the optimality condition $\nabla\phi(x^\star) = 0$ or

$$A^T A x^\star - A^T b = 0.$$

These are the well known normal equations that characterize the solution of a least squares problem. The dubious nature of the normal equations has long been recognized [49]. Solving the least squares problem via the normal equations is not a backward stable algorithm, and information contained in the data matrix A may be lost when A^TA and A^Tb are formed in finite-precision. In addition, if A is sparse but contains one or more dense rows, the matrix A^TA will be dense. Even when $n \ll m$ these numerical issues may negate the computational benefits of solving with the comparatively small $n \times n$ matrix A^TA.

It is possible to transform problem (8.2) to avoid forming the normal equations by including the residual vector r in the problem and adding linear constraints

$$\begin{array}{ll} \underset{x,r}{\text{minimize}} & \frac{1}{2}r^Tr \\ \text{subject to} & Ax + r = b. \end{array} \tag{8.3}$$

While the above problem could also be solved by the regularized BCL algorithm, this would be inefficient as problem (8.3) would be transformed into

$$\begin{array}{ll} \underset{x,r,r_1}{\text{minimize}} & \frac{1}{2}\delta_k x^Tx + \frac{1}{2}(1+\delta_k)r^Tr + y_k^Tr_1 + \frac{1}{2}\rho_k r_1^Tr_1 \\ \text{subject to} & Ax + r + r_1 = b, \end{array}$$

and a sequence of problems of this form would be solved (with δ_k converging to zero and ρ_k and y_k adjusted via the BCL algorithm).

If regularization is added directly to the least squares problem, we arrive at the Tikhonov regularized least squares problem

$$\begin{array}{ll} \underset{x,r}{\text{minimize}} & \frac{1}{2}\gamma x^Tx + \frac{1}{2}r^Tr \\ \text{subject to} & Ax + r = b. \end{array} \tag{8.4}$$

The above problem is a regularized QP subproblem (7.2) and may be solved directly by the regularized QP algorithm (Algorithm 7). We note that adding Tikhonov regularization to the least squares problem may also be justified from a statistical standpoint.

Of course, the ordinary least squares problem is linear algebraic in nature, and so it is not necessary to use an optimization method. Golub [48] pioneered the use of the QR factorization for solving ordinary least squares problems. The matrix A is factored into $A = QR$, where $Q \in \mathbb{R}^{m\times m}$ is orthogonal and $R \in \mathbb{R}^{m\times n}$ is upper-triangular. When A is sparse, a sparse QR factorization may be used [24, 25]. While the orthogonal matrix Q tends to be dense it need not be stored. Instead, as it is computed, it may be applied to b. At the completion of the QR factorization algorithm the matrix R and the vector Q^Tb are returned, but the matrix Q has not been formed or stored.

An augmented system may also be used to solve the sparse least squares problem. In previous versions of MATLAB, before sparse QR algorithms were well developed, least squares problems were

solved via the augmented system

$$\begin{pmatrix} \kappa I & A \\ A^T & \end{pmatrix} \begin{pmatrix} r/\kappa \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix},$$

where $\kappa \in \mathbb{R}$ is a scaling factor chosen to reduced the condition number of the augmented matrix. (For more details about this approach see Björck [1] and the MATLAB function `spaugment` [99].)

If applied to the Tikhonov regularized least squares problem (8.4), the regularized QP algorithm would solve the system

$$\begin{pmatrix} \gamma I & A^T \\ A & -I \end{pmatrix} \begin{pmatrix} \Delta x \\ -y \end{pmatrix} = \begin{pmatrix} -\gamma x_0 \\ r_0 \end{pmatrix},$$

where x_0 is the initial guess and $r_0 = b - Ax_0$. The first iterate $(x_1, r_1) = (x_0, r_0) + (\Delta x, \Delta r)$ would satisfy the optimality conditions of problem (8.4):

$$\begin{pmatrix} \gamma I & A^T \\ A & -I \end{pmatrix} \begin{pmatrix} x_1 \\ -r_1 \end{pmatrix} = \begin{pmatrix} 0 \\ b \end{pmatrix}.$$

Thus, the regularized QP algorithm is an efficient method for solving sparse regularized least squares problems.

8.1.1 Bound-constrained least squares

Often, an unknown parameter x_j is not allowed to take on arbitrary values, but instead is constrained to lie within a certain interval. Constraints of this type lead to the bound-constrained least squares problem

$$\underset{x}{\text{minimize}} \quad \|b - Ax\|_2 \quad \text{subject to} \quad \ell \leq x \leq u.$$

Such problems arise in a variety of applications, including reconstruction problems in geodesy and tomography, contact problems in mechanical systems, and the modeling of ocean circulation [1]. Unlike ordinary least squares, a bound-constrained least squares problem must be solved with an optimization method.

If regularization is included and the bound-constrained least squares problem is transformed into

$$\begin{array}{ll} \underset{x,r}{\text{minimize}} & \frac{1}{2}\gamma x^T x + \frac{1}{2} r^T r \\ \text{subject to} & Ax + r = b \\ & \ell \leq x \leq u, \end{array}$$

the above problem may be efficiently solved with the regularized QP algorithm.

8.2 Generalized least squares

When the linear model (8.1) is generalized to allow the random errors to be correlated, the ordinary least squares problem becomes a generalized least squares problem. The covariance matrix of r is $\sigma^2 W$, where $W \in \mathbb{R}^{m\times m}$ is symmetric and positive semidefinite. If W is positive definite and A has full column-rank, Aitken [2] showed the best linear unbiased estimate of the parameters x is the solution of the problem

$$\underset{x}{\text{minimize}} \quad (b - Ax)^T W^{-1}(b - Ax). \tag{8.5}$$

When W is positive definite a Cholesky factorization $W = LL^T$ can be computed, where $L \in \mathbb{R}^{m\times m}$ is a full-rank lower triangular matrix. Problem (8.5) is then equivalent to

$$\underset{x}{\text{minimize}} \quad \left\|L^{-1}(b - Ax)\right\|_2, \tag{8.6}$$

which is an ordinary least squares problem $\text{LS}(\tilde{A}, \tilde{b})$, where $\tilde{A}$ and $\tilde{b}$ satisfy $L\tilde{A} = A$ and $L\tilde{b} = b$. We may also use the Cholesky factor to replace the model $Ax + r = b$ with the corresponding model $Ax + Lu = b$, where $u \in \mathbb{R}^m$ is a random vector with a covariance matrix $\sigma^2 I$. This yields the problem

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & u^T u \\ \text{subject to} & Ax + Lu = b. \end{array} \tag{8.7}$$

This formulation of the generalized least squares problem is discussed in detail by Paige [83, 84]. In the above problem the linear constraints are not of the form $Ax + r = b$, so (even if regularization were added) this problem is not a regularized QP subproblem (7.2). However, because L has full rank and we may perform solves with L, it would be possible to generalize the algorithm for solving problem (7.2) to handle the constraint $Ax + Lr = b$. The algorithm would then solve KKT systems (corresponding to (7.11))

$$\begin{pmatrix} H_{\mathcal{M}} & A_{\mathcal{M}}^T \\ A_{\mathcal{M}} & -LD_\rho^{-1}L^T \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{M}} \\ -y \end{pmatrix} = \begin{pmatrix} -g_{\mathcal{M}} \\ LD_\rho^{-1} g_r \end{pmatrix}. \tag{8.8}$$

When W is positive semidefinite, we can still write $W = BB^T$ (where $B \in \mathbb{R}^{m\times p}$ with $p < m$) and solve the equivalent problem

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & u^T u \\ \text{subject to} & Ax + Bu = b. \end{array} \tag{8.9}$$

Because B is not square and full-rank, this problem has general linear equality constraints and so cannot be solved with the regularized QP algorithm. However, it can be solved with the regularized BCL algorithm for convex quadratic programming.

8.2.1 Weighted least squares

A special case of the generalized least squares problem occurs when W is positive definite and diagonal. These problems are known as weighted least squares problems and arise in equilibrium problems in electrical networks and structural analysis, and finite element methods [1].

The weighted least squares problem can be written as

$$\begin{aligned} \underset{x,r}{\text{minimize}} \quad & \tfrac{1}{2} r^T D r \\ \text{subject to} \quad & Ax + r = b, \end{aligned} \tag{8.10}$$

where $D = W^{-1}$ is diagonal. If Tikhonov regularization is added, problem (8.10) is a regularized QP subproblem (7.2). Thus, it may be solved directly by the regularized QP algorithm. The algorithm for solving the regularized QP subproblem relies on D being well-conditioned to eliminate Δr in system (7.10). Thus, weighted least squares problems with ill-conditioned covariance matrices should be solved with the regularized BCL algorithm.

Chapter 9

QPBLUR implementation

The algorithms described in the previous chapters are implemented in the solver QPBLUR (Quadratic Programming with Block-LU updates and Regularization). In this chapter we discuss some of the important implementation details of QPBLUR.

9.1 Implementation overview

The QPBLUR solver has been implemented in two languages: MATLAB and Fortran 95. A prototype implementation of QPBLUR was first constructed inside the MATLAB environment for testing and evaluation. As development progressed, this implementation became a reference for a Fortran 95 version of QPBLUR. MEX interfaces were constructed to allow individual Fortran routines to be verified and tested within the MATLAB environment and compared against the reference implementation.

As a result of this development strategy, MATLAB users may choose between a pure MATLAB version of QPBLUR and a version that uses the MEX interface to the Fortran routines. Users may prefer the pure MATLAB version of QPBLUR for portability and the fact that it does not require compilation. The MEX interface version can be several times faster on certain problems. Both versions can be accessed through an optional interface that follows the same calling conventions as the Optimization Toolbox routine `quadprog` [90]. MATLAB users will benefit from using QPBLUR as a replacement for `quadprog` on sparse convex QPs, as `quadprog` is a dense method.

The Fortran version of QPBLUR has been integrated into a development version of SNOPT, a solver for sparse nonlinear programs [39]. QPBLUR may be used as an alternative to SQOPT for the QP subproblems that are used to define search directions within SNOPT. The use of QPBLUR within SNOPT's SQP method is described in further detail in Chapter 10.

Interfaces to the Fortran version of QPBLUR are available for AMPL [4] and the Standard Input Format (SIF) [18]. These interfaces were constructed to test QPBLUR on QPs in the COPS [27, 20] and CUTEr test sets [51]. The performance of QPBLUR on these test sets is described in Chapter 11. As a result of this work, QPBLUR may be used as a solver for QP and LP models described in AMPL.

Programs written in Fortran and C/C++ may also use QPBLUR by linking against the library `libqpblur` and directly calling the solver routines.

Both the MATLAB and Fortran implementations of QPBLUR make heavy use of linear algebra routines developed for the QP solver QPBLU (Huynh [59]). These routines are used to perform block-LU updates and solve KKT systems (as described in Chapter 3).

9.2 Block-LU update implementation

Several modifications were necessary to enable Huynh's block-LU update routines to be used inside QPBLUR. The most important of these modifications are described in this section.

Because QPBLUR works with regularized KKT systems, the initial KKT matrix K_0 has the form

$$K_0 = \begin{pmatrix} H_{\mathcal{M}_0} + \delta I & A_{\mathcal{M}_0}^T \\ A_{\mathcal{M}_0} & -D_\rho^{-1} \end{pmatrix}$$

instead of the traditional KKT matrix

$$\begin{pmatrix} H_{\mathcal{M}} & A_{\mathcal{M}}^T \\ A_{\mathcal{M}} & \end{pmatrix}.$$

The additional terms in the $(1,1)$ and $(2,2)$ block of K_0 required changes to the routines used to form K_0 and perform certain block-LU updates (notably Case 1 described in Section 3.1).

Several performance enhancements were also made to the block-LU routines. Originally an $O(\varrho\,|\mathcal{M}_0|^2)$ algorithm was used to form K_0. This was replaced with an $O(\varrho\,|\mathcal{M}_0|)$ algorithm, where ϱ is the maximum number of nonzeros in a column of K_0.

In QPBLU, the sparse block factors Y and Z (defined in Section 3.3) are stored in compressed sparse column format (CSC). (Davis [24] presents a detailed description of the CSC format.) Storage for each matrix is preallocated with a maximum number of nonzeros and a maximum number of columns. As changes are made to the active set, columns are added to and deleted from these matrices. A list of columns currently in each matrix is maintained. When a column is deleted (as in Cases 3 and 4 in Section 3.1), memory associated with this column is not reclaimed. Instead the index of the deleted column is simply removed from the list of columns. This approach is taken because if a deleted column is near the beginning of the CSC matrix, reclaiming the memory associated with the column requires copying nearly the entire matrix. If Y or Z contain many nonzeros, this is an expensive operation. In QPBLU, if the maximum number of columns or maximum number of nonzeros is reached, a factorization is forced to define new factors $L_0U_0 = K_0$ and reset Y and Z to zero.

Within the QPBLUR solver, the above strategy leads to excessive factorizations when many changes are made to the active set but the size $|\mathcal{M}|$ of the set of moving variables remains relatively constant. This case occurs when a nonmoving variable is released from a bound, only to become fixed on a bound again after several iterations. To prevent excessive factorizations, which are often

the most expensive operation in the algorithm, a stop-and-copy garbage collection approach is taken. When the maximum number of columns or the maximum number of nonzeros in Y or Z is reached, storage for a new matrix of larger size is allocated, the list of columns is traversed, columns that have not been deleted are copied into the new matrix, and the memory associated with the old matrix is returned to the operating system. This approach decreases the number of factorizations required, and performs few stop-and-copy operations.

As the size of the block factors grows, the work needed to solve the current KKT system increases. Thus, it is important to periodically refactor the KKT matrix from scratch and reset Y, Z, and C to zero. A parameter `nCmax` is used to bound the size of the dense matrix C. When C reaches dimension `nCmax` $\times$ `nCmax`, the KKT matrix is refactored. A value of `nCmax` $= 200$ was found to work well on a large variety of problems.

Refactorization can be particularly advantageous when the current number of moving variables $|\mathcal{M}|$ is less than the original number of moving variables $|\mathcal{M}_0|$. This is because, before refactorization, systems are solved with the factors L_0 and U_0 of dimension $|\mathcal{M}_0| \times |\mathcal{M}_0|$. After refactorization, solves are performed with new factors L_0 and U_0 of reduced size $|\mathcal{M}| \times |\mathcal{M}|$.

The KKT matrix is also refactored when an estimate of the condition number of C surpasses a threshold parameter `condCmax`. Because only the factors E and R of C ($EC = R$) are stored (not C itself), the condition number is estimated by the ratio of the largest and smallest diagonals of the upper-triangular matrix R. An additional estimate of the 1-norm condition number

$$\kappa_1(C) = \left\|C^{-1}\right\|_1 \|C\|_1$$

is obtained by estimating the norms $\left\|C^{-1}\right\|_1$ and $\|C\|_1$. The 1-norm estimator routine `dlacn2` developed by Higham [58] is used to compute these estimates. Because C is not stored, an estimate of $\|C\|_1$ is obtained via an estimate of $\|R\|_1$. This is a reasonable approximation, as E is well-conditioned. If the maximum of these two estimates exceeds `condCmax` the KKT matrix is refactored.

9.3 Sparse direct linear solvers

Huynh's routines maintain an abstraction barrier between the code that performs the block-LU updates and augmented system solves and the sparse direct linear solver. As a result, QPBLUR may use any sparse direct linear solver, provided an interface to the solver is implemented that contains the following functions:

- `LUinit`: Allocate memory and perform any initialization need by the solver
- `LUfactorize`: Compute LU factors of a KKT matrix K stored in CSC format
- `Lsolve`: Perform a solve with "L" or "L^T"
- `Usolve`: Perform a solve with "U" or "U^T"
- `LUclear`: Free memory and resources associated with the solver

- `LUstats`: Return statistics on the amount of memory used by the solver.

Huynh implemented interfaces to the following sparse solvers: LUSOL [70], MA57 [28], PARDISO [87], SuperLU (the sequential version) [102], and UMFPACK [104].

Some sparse direct solvers do not allow the user to perform separate solves with the individual L, U or L, D, L^T factors of a matrix K. These solvers only provide a routine for performing solves with K. In this case the block-LU factorization must be defined as

$$\begin{pmatrix} K & V \\ V^T & D \end{pmatrix} = \begin{pmatrix} K & \\ Z^T & I \end{pmatrix} \begin{pmatrix} I & Y \\ & C \end{pmatrix}, \tag{9.1}$$

with Y, Z and C defined as $KY = V$, $Z = V$, and $C = D - Z^T Y$. The `Lsolve` routine should then perform a solve with K and the `Usolve` routine should perform a solve with the identity matrix. Unfortunately, this causes Y, in general, to contain more nonzeros than if it were defined in terms of L by $LY = V$.

Huynh's interface to the PARDISO solver defined Y and Z in terms of K and I. A subsequent version of PARDISO has been released that allows for separate L and U solves to be performed. A second PARDISO interface was therefore developed to utilize these separate solves.

9.4 Limits on the regularization parameters

Theoretical convergence to a solution of the original QP requires that the primal regularization parameter $\delta \to 0$. Although the BCL method ensures the penalty parameter ρ is finite, it may be quite large. In practice, δ cannot be allowed to become arbitrarily small or ρ to become extremely large because the regularized KKT system

$$\begin{pmatrix} H_{\mathcal{M}_0} + \delta I & A_{\mathcal{M}_0}^T \\ A_{\mathcal{M}_0} & -\frac{1}{\rho} I \end{pmatrix}$$

must be nonsingular to within machine precision. Computational experiments indicate that δ should not fall below $\delta_{\min} \equiv 10^{-14}$ and ρ should not exceed $\rho_{\max} \equiv 10^{14}$ when machine precision is $\epsilon_{\text{mach}} \simeq 10^{-16}$. In practice, these values have proven effective even though many of the test problems considered in Chapter 11 have poorly scaled H and A.

9.5 Early termination of the regularized QP subproblem

An important property of the BCL method is that the subproblems do not have to be solved to very high precision. Early subproblems (QP(ρ_k, y_k, δ_k)) are solved to an optimality tolerance ω_k. When a BCL iterate is successful, ω_k is decreased. After several successful iterations ω_k reaches the user requested optimality tolerance ω.

An iteration limit is also imposed on the QP subproblems. This iteration limit is a small multiple of the refactorization frequency. That is, if we refactor the KKT matrix when the Schur complement

C is of size `nCmax`, we impose a limit of $\kappa \times$ `nCmax` iterations on the QP subproblem ($\kappa = 2$ works well in practice). This limit causes QPBLUR to exit the subproblem and adjust the Lagrange multiplier estimates y_k just before performing an expensive refactorization. The next subproblem resumes from the same point and active set at which the last subproblem was halted. We have found this feature useful for adjusting the Lagrange multiplier estimates when a variable is being added to $\mathcal{M}$ at every iteration.

9.6 The step length procedure

QPBLUR uses a simplified step length procedure, as described in Section 2.6. A parameter `pivtol` is used to avoid dividing by small components of the search direction when computing the maximum feasible step

$$\alpha_F = \max\left\{ \max_{j \in \mathcal{M}: p_j < -\texttt{pivtol}} \left(\frac{\ell_j - x_j}{p_j} \right), \quad \max_{j \in \mathcal{M}: p_j > \texttt{pivtol}} \left(\frac{u_j - x_j}{p_j} \right) \right\}.$$

We found `pivtol` $\equiv 10^{-11}$ to work well on a large variety of problems. An anti-cycling procedure (such as EXPAND [43]) is unnecessary. In particular, $\ell \leq x \leq u$ is maintained precisely throughout, and in contrast to most active-set solvers, QPBLUR does not need to allow the variables to be outside their bounds by some feasibility tolerance (typically 10^{-6}). Primal feasibility is measured solely by $\|r\|$, as it is in interior-point methods.

9.7 The initial point and the initial active set

When solving the problem

$$\begin{aligned} \underset{x,s}{\text{minimize}} \quad & c^T x + \tfrac{1}{2} x^T H x \\ \text{subject to} \quad & Ax - s = 0 \\ & \ell \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u, \end{aligned}$$

QPBLUR takes as input an initial point (x_0, s_0). The user may also provide initial sets $\mathcal{M}_0$, $\mathcal{L}_0$ and $\mathcal{U}_0$. In this section we discuss how this initial point is modified to ensure that it is feasible and consistent with the initial active set.

The starting point (x_0, s_0) is made to lie between the bounds ℓ and u by setting

$$\begin{pmatrix} x_0 \\ s_0 \end{pmatrix} \leftarrow \max\left(\ell, \min\left(\begin{pmatrix} x_0 \\ s_0 \end{pmatrix}, u \right) \right).$$

If the user provided $\mathcal{L}_0$ and $\mathcal{U}_0$, variables in these sets are placed on their bounds by setting

$$\left[\begin{pmatrix} x_0 \\ s_0 \end{pmatrix} \right]_j \leftarrow \ell_j, \text{ for } j \in \mathcal{L}_0 \quad \text{and} \quad \left[\begin{pmatrix} x_0 \\ s_0 \end{pmatrix} \right]_j \leftarrow u_j, \text{ for } j \in \mathcal{U}_0.$$

Variables with $\ell_j = -\infty$ ($u_j = \infty$) are not allowed to be in $\mathcal{L}_0$ ($\mathcal{U}_0$) as the initial point (x_0, s_0) must be finite. If a variable is unconstrained ($-\infty < x_j < \infty$), it is placed in $\mathcal{M}_0$ (and will remain a moving variable for the duration of the solve). However, if $j \in \mathcal{M}_0$, $[x_0]_j$ may lie on one of its bounds. Thus, QPBLUR is allowed to start from a degenerate point.

Some QPs contain variables x_j with fixed bounds ($\ell_j = u_j$). For instance, the AMPL interface creates a variable with fixed bounds to add a constant term to the QP objective so that the solver objective matches the model objective [36]. Variables with fixed bounds also arise when equality constraints are present. When slack variables are add to problem (1.1) to yield (1.2), the equality constraint $a_i^T x = \beta_i$ is transformed into $a_i^T x - s_i = 0$ with $\beta_i \leq s_i \leq \beta_i$.

In QPBLUR, the set $\mathcal{N}$ of nonmoving variables is partitioned into three sets $\mathcal{L}$, $\mathcal{E}$, and $\mathcal{U}$. Variables with fixed bounds are treated as nonmoving variables and placed in $\mathcal{E}$. If an index j is contained in $\mathcal{E}$, for all iterations $\ell_j = x_j = u_j$. These variables can never become moving variables.

9.8 Detecting infeasible and unbounded problems

Infeasible and unbounded QPs may arise in practice. Thus, QPBLUR must have a method for detecting these types of problems.

The regularized BCL algorithm (Algorithm 6) is well defined on infeasible QPs. The BCL subproblem (QP(ρ_k, y_k, δ_k)) is always feasible, even when the original QP is not. To detect infeasible problems, a threshold penalty parameter $\rho_{\max}$ and the user's requested feasibility tolerance ϵ are used. If $\rho_k \geq \rho_{\max}$, the subproblem (QP(ρ_k, y_k, δ_k)) has been solved to optimality, and the size of optimal residual $\|r_k^\star\|$ is larger than ϵ, the problem is declared infeasible.

The BCL subproblem (QP(ρ_k, y_k, δ_k)) is also bounded below, even when the original QP is not. To determine if a problem is unbounded, a tolerance $\xi_{\max}$ is used. A convex quadratic objective function $\phi(x) \to -\infty$ only when $\|x\| \to \infty$. Thus, a problem is declared unbounded if the size of the subproblem solution $\|x_k^\star\|$ is larger than $\xi_{\max}$. In practice, choosing a good value for $\xi_{\max}$ is difficult. For example, consider the unbounded QP

$$\text{minimize} \quad \beta \tag{9.2}$$

and the corresponding regularized QP

$$\text{minimize} \quad \beta + \tfrac{1}{2}\delta\beta^2.$$

The solution of the regularized QP is given by $\beta^\star = -1/\delta$. Thus, the test $\|\beta^\star\| > \xi_{\max}$ needs to take into the account the size of $\delta_{\min}$ (the smallest regularization parameter used in the regularized BCL algorithm). For instance, to detect that (9.2) is unbounded we must have $\xi_{\max} < 1/\delta_{\min}$. However, we do not want to make $\xi_{\max}$ too small as we risk classifying some QPs as unbounded when they are not.

9.9 Optimality criteria

QPBLUR takes as input an optimality tolerance ω and a feasibility tolerance ϵ. These two tolerances are used to define the optimality conditions as follows:

$$\begin{aligned} \|Ax - s\|_\infty &\leq \epsilon\,(1 + \max(\|x\|_\infty, \|s\|_\infty)) \\ \ell \leq x &\leq u \\ \left\| \nabla\phi(x,s) - \begin{pmatrix} A^T \\ -I \end{pmatrix} y - z_l + z_u \right\|_\infty &\leq \omega\,(1 + \|y\|_\infty) \\ z_l^T \left(\begin{pmatrix} x \\ s \end{pmatrix} - \ell \right) &= 0 \\ z_u^T \left(u - \begin{pmatrix} x \\ s \end{pmatrix} \right) &= 0 \\ z_l, z_u &\geq -\omega. \end{aligned}$$

If a point (x, s, y, z_l, z_u) satisfies the above conditions it is declared optimal.

9.10 Scaling

QPs are often formulated without regard to the size of the elements of the data matrix A. Row and column scaling can often improve the performance of an active-set method (although there are cases where such scaling can make a problem more difficult to solve). QPBLUR contains an option to rescale the problem so that the nonzero elements of A tend to be closer to 1.

An iterative scaling procedure is used that makes several passes through the columns and rows of A [95]. On each pass the geometric mean of the nonzeros in each (scaled) column or row is computed and used as the new scale factor. This procedure returns positive definite diagonal matrices R and C. New scaled data $\tilde{A} = R^{-1}AC^{-1}$, $\tilde{H} = C^{-1}HC^{-1}$, $\tilde{c} = C^{-1}c$,

$$\tilde{\ell} = \begin{pmatrix} C & \\ & R^{-1} \end{pmatrix} \ell, \text{ and } \tilde{u} = \begin{pmatrix} C & \\ & R^{-1} \end{pmatrix} u$$

are computed and the QP

$$\begin{aligned} \underset{\tilde{x}}{\text{minimize}} \quad & \tilde{c}^T\tilde{x} + \tfrac{1}{2}\tilde{x}^T\tilde{H}\tilde{x} \\ \text{subject to} \quad & \tilde{\ell} \leq \begin{pmatrix} \tilde{x} \\ \tilde{A}\tilde{x} \end{pmatrix} \leq \tilde{u} \end{aligned}$$

involving the scaled variables $\tilde{x} = Cx$ is solved.

Chapter 10

Quadratic programming within an SQP method

In this chapter we discuss quadratic programming within an SQP method for nonlinear programming. We begin with a brief overview of SQP methods and then consider the SNOPT solver, an implementation of an SQP method for nonlinear programs with sparse constraints and derivatives. We present the QP subproblems solved within SNOPT, and discuss its present QP solver SQOPT. We describe how QPBLUR may be used as an alternative QP solver for SNOPT.

10.1 Sequential quadratic programming methods

SQP methods have proved highly effective at solving constrained optimization problems of the form

$$\begin{aligned} \text{(locally) } \underset{x}{\text{minimize}} \quad & \varphi(x) \\ \text{subject to} \quad & \ell \leq \begin{pmatrix} x \\ c(x) \\ Ax \end{pmatrix} \leq u, \end{aligned} \tag{NLP}$$

where the objective $\varphi(x)$ and the constraints $c(x)$ are smooth, but otherwise general, nonlinear functions.

SQP methods solve problem (NLP) via a sequence of QP subproblems. The constraints of each QP subproblem are linearizations of the nonlinear constraints in the original problem, and the subproblem objective function is a quadratic approximation to a modified Lagrangian function. The solution of each subproblem defines a search direction, and the active set of constraints (at the solution) provides an estimate of the active set for the original problem.

In an SQP method there are *major* and *minor* iterations. The major iterations generate a sequence of iterates (x_k, y_k) that converge to a locally optimal solution $(x^\star, y^\star)$ of the original problem, where y is a vector of Lagrange multipliers for the nonlinear constraints. At each major

iteration a QP subproblem is solved. Minor iterations are iterations of the quadratic programming method.

SQP methods are particularly effective when the objective and constraint functions are expensive to evaluate. For example, optimizing the shape of an airplane wing to minimize drag might require a partial differential equation to be solved each time $\varphi(x)$ and $c(x)$ are evaluated. SQP methods evaluate the functions and gradients once per major iteration in order to define the QP subproblem. (These methods also evaluate the functions after the QP subproblem has been solved during a line search on some merit function.) Many minor iterations may be required to solve the QP subproblem. However, this approach may be considerably less expensive than other methods that require several function and gradient evaluations to define a search direction.

10.2 SNOPT: an SQP method for sparse problems

SNOPT (Sparse Nonlinear OPTimizer) [39, 40] is an implementation of an SQP method that exploits sparsity in the matrix A and the Jacobian J of the nonlinear constraints in (NLP). SNOPT is designed to solve problems where the gradients of the objective and constraint functions are available. Second derivatives are assumed to be unavailable or too expensive to compute. Instead, a limited-memory quasi-Newton approximation to the second derivative of the modified Lagrangian is maintained.

Let (x_k, y_k) denote the current point at the kth major iteration of the SQP algorithm, and define the current function and gradients evaluated at x_k as

$$\varphi_k = \varphi(x_k), \quad g_k = \nabla\varphi(x_k), \quad c_k = c(x_k), \quad \text{and} \quad J_k = J(x_k).$$

The *modified Lagrangian* associated with (NLP) is

$$L(x, x_k, y_k) = \varphi(x) - y_k^T d_k(x, x_k),$$

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

$$\begin{aligned} \bar{c}_k(x, x_k) &= c_k + J_k(x - x_k), \\ d_k(x, x_k) &= c(x) - \bar{c}_k(x, x_k). \end{aligned}$$

The first and second derivatives of the modified Lagrangian with respect to x are

$$\begin{aligned} \nabla L(x, x_k, y_k) &= \nabla\varphi(x) - (J(x) - J_k)^T y_k, \\ \nabla^2 L(x, x_k, y_k) &= \nabla^2\varphi(x) - \sum_i [y_k]_i \nabla^2 c_i(x). \end{aligned}$$

At $x = x_k$, the modified Lagrangian has the same value and gradient as the objective, namely $L(x_k, x_k, y_k) = \varphi_k$, and $\nabla L(x_k, x_k, y_k) = g_k$. Thus, the quadratic approximation to the modified

Lagrangian at the point x_k is

$$L_q(x, x_k, y_k) = \varphi_k + g_k^T(x - x_k) + \tfrac{1}{2}(x - x_k)^T \nabla^2 L(x_k, x_k, y_k)(x - x_k). \tag{10.1}$$

If (x_k, y_k) is an optimal solution $(x^\star, y^\star)$, the optimality conditions of the QP

$$\begin{array}{ll} \underset{x}{\text{minimize}} & L_q(x, x_k, y_k) \\ \text{subject to} & \ell \leq \begin{pmatrix} x \\ \bar{c}_k(x, x_k) \\ Ax \end{pmatrix} \leq u \end{array} \tag{10.2}$$

are identical to those of (NLP).

Because SNOPT chooses not to work with second derivatives, the $\nabla^2 L$ term in the objective is replaced by a positive semidefinite quasi-Newton approximation H_k. Note that the true Hessian of $L(x, x_k, y_k)$ is zero in rows and columns associated with linear variables. If the variables are ordered so that those appearing nonlinearly in the objective or constraints come first, the quasi-Newton Hessian approximation takes the form

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

where $\bar{H}_k = G_k^T G_k$, $G_k = \bar{H}_0^{1/2} \prod_{i=1}^{k} (I + s_i v_i^T)$, $\bar{H}_0$ is diagonal and positive definite, and $\{(s_i, v_i)\}_{i=1}^{k}$ are limited-memory update pairs. (For more details of the limited-memory quasi-Newton approximation see Section 3.2.)

Thus, at the kth major iteration, a search direction $\Delta x = x - x_k$ and $\Delta y = y - y_k$ is found by solving the convex QP

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & \varphi_k + g_k^T(x - x_k) + \tfrac{1}{2}(x - x_k)^T H_k (x - x_k) \\ \text{subject to} & c_k + J_k(x - x_k) - s = 0 \\ & \ell \leq \begin{pmatrix} x \\ s \\ Ax \end{pmatrix} \leq u, \end{array} \tag{10.3}$$

where s is a vector of slack variables for the linearized constraints. A line search is then conducted on an augmented Lagrangian merit function, regarded as a function of both x and y.

Because H_k, the quasi-Newton approximation to $\nabla^2 L$, is positive semidefinite, problem (10.3) is convex. Recent research by Gill and Wong [46, 47] seeks to develop a nonconvex QP solver that will allow SNOPT to use exact second derivatives with indefinite Hessians. Like QPBLUR, their method utilizes dual regularization and block-LU updates of the KKT factors. However, an inertia-controlling algorithm is required for general quadratic programming. Even with dual regularization and an inertia-controlling method, a basis repair (and possibly a KKT repair) procedure is necessary

for warm starts and singular or nearly singular Hessians. If such measures can be implemented, the benefit should be reduced numbers of major and minor iterations.

10.3 SQOPT: a reduced-gradient QP solver

At present SNOPT uses the convex QP solver SQOPT [41] to solve problem (10.3). SQOPT employs a two-phase active-set reduced-gradient method. An overview of this method is presented in Section 2.5.1. If n_S, the number of superbasic variables, is small, SQOPT uses a dense Cholesky factorization of the reduced Hessian:

$$LL^T = Z_{\mathcal{M}}^T H_{\mathcal{M}} Z_{\mathcal{M}}.$$

When n_S becomes large, SQOPT uses the conjugate-gradient (CG) method to approximately solve

$$(Z_{\mathcal{M}}^T H_{\mathcal{M}} Z_{\mathcal{M}} + \delta I)p_z = -Z_{\mathcal{M}}^T g_{\mathcal{M}}, \tag{10.4}$$

where $\delta \simeq 10^{-3}$ is a small regularization parameter to allow for singular reduced Hessians. By default the algorithm switches from Cholesky to CG when $n_S > 2000$. The number of CG iterations used to solve (10.4) is limited (by default the limit is 100 iterations). If this limit is exceeded, the approximate solution is used, a step is taken, and a new minor iteration is started.

When the nonlinear variables are ordered to appear first, problem (10.3) has the form

$$\begin{aligned} \underset{x,s_n,s_l,s_{\texttt{iObj}}}{\text{minimize}} \quad & \begin{pmatrix} \sigma g_{\texttt{Obj}} \\ 0 \end{pmatrix}^T (x - x_k) + \tfrac{1}{2}(x - x_k)^T \begin{pmatrix} \bar{H} & 0 \\ 0 & 0 \end{pmatrix} (x - x_k) + \sigma s_{\texttt{iObj}} \\ \text{subject to} \quad & \begin{pmatrix} J_1 & J_2 \\ J_3 & J_4 \\ l_1^T & l_2^T \end{pmatrix} x - \begin{pmatrix} s_n \\ s_l \\ s_{\texttt{iObj}} \end{pmatrix} = \begin{pmatrix} b \\ 0 \\ 0 \end{pmatrix} \\ & \ell_x \le x \le u_x, \quad \ell_n \le s_n \le u_n \quad \ell_l \le s_l \le u_l, \end{aligned} \tag{10.5}$$

with variables, data, and dimensions as described in Tables 10.1–10.2. The correspondence between problem (10.5) and (10.3) may be seen from the following definitions:

$$J_k \equiv \begin{pmatrix} J_1 & J_2 \end{pmatrix}, \; A \equiv \begin{pmatrix} J_3 & J_4 \\ l_1^T & l_2^T \end{pmatrix}, \; \ell \equiv \begin{pmatrix} \ell_x \\ \ell_n \\ \ell_l \end{pmatrix}, \; u \equiv \begin{pmatrix} u_x \\ u_n \\ u_l \end{pmatrix}, \; l \equiv \begin{pmatrix} l_1 \\ l_2 \end{pmatrix}, \; b \equiv J_k x_k - c_k.$$

The linear portion of the objective $\varphi(x)$ is represented by the vector l. The constraint $l^T x = s_{\texttt{iObj}}$, allows the linear term to be incorporated into the QP objective. The constant σ allows SNOPT to handle maximization and feasibility problems.

Although problem (10.5) is solved to compute a search direction $\Delta x = x - x_k$, this direction is not solved for directly. Instead, the new point x is computed. Note, the constant terms involving x_k

Table 10.1: Quantities used in problem (10.5)

Quantity	Description	Dimension
$g_{\texttt{Obj}}$	Gradient of the nonlinear objective	nnObj
J_1	Jacobian of the nonlinear constraints w.r.t nonlinear variables	nnCon $\times$ nnJac
J_2	Jacobian of the nonlinear constraints w.r.t linear variables	nnCon $\times$ (n $-$ nnJac)
J_3	Jacobian of the linear constraints w.r.t nonlinear variables	(m $-$ nnCon) $\times$ nnJac
J_4	Jacobian of the linear constraints w.r.t linear variables	(m $-$ nnCon) $\times$ (n $-$ nnJac)
s_n	Slack variables for the nonlinear constraints	nnCon
s_l	Slack variables for the linear constraints	m $-$ nnCon
$s_{\texttt{iObj}}$	Slack variable for the linear objective	1
l	Linear term of the objective	n
b	Right-hand side for nonlinear constraints	nnCon
$\bar{H}$	Hessian approximation	nnH $\times$ nnH
$\bar{H}_0$	Diagonal portion of Hessian approximation	nnH $\times$ nnH
σ	$= 1$ (minimize) $= -1$ (maximize) $= 0$ (feasible point)	1

Table 10.2: Description of dimensions used in problem (10.5)

Dimension	Description
m	Number of constraints
n	Number of variables
nnObj	Number of nonlinear variables in the objective
nnJac	Number of nonlinear variables in the nonlinear constraints
nnCon	Number of nonlinear constraints
nnH	max(nnObj, nnJac)
iObj	Index of linear objective row
nQNmod	Number of rank-2 modifications to the Hessian

are not eliminated from the objective. Instead, the quadratic objective and gradient are computed using Δx. This is preferable for numerical reasons.

The dense approximate Hessian $\bar{H}$ is never formed explicitly. When system (10.4) is solved by CG, only matrix-vector products $\bar{H}v$ are needed.

For a cold start, the first step of the SNOPT algorithm is to compute a point that satisfies the linear constraints in problem (NLP). This is done by solving an LP. The solution and optimal active set from this LP is used to initialize SQOPT on the first QP subproblem. As a result, SQOPT starts from a vertex with zero superbasic variables. When solving a nonlinear program with $n_S^\star$ degrees of freedom at the solution, SQOPT must perform at least $n_S^\star$ minor iterations to reach the optimal active set. When $n_S^\star$ is large, this may be very expensive.

SNOPT contains an option `New superbasic limit` that controls the number of superbasic variables that may be added during a single major iteration. By default this value is set at 99. If the number of new superbasics exceeds this value, SQOPT is terminated early and a new major iteration occurs. This limit is imposed to prevent adding many superbasic variables in early QP subproblems only to have them deleted later. However, it also causes optimization problems with many degrees of freedom to require many major iterations. Because the number of function and gradient evaluations is directly related to the number of major iterations, this is inefficient when evaluations are expensive.

The above strategy was chosen by the authors of SNOPT because they expected to encounter

problems with few degrees of freedom. If problems with many degrees of freedom are expected, SNOPT could be made to solve an initial QP (rather than an LP). This would allow SNOPT to begin with many superbasics. However, the resulting iterations of SQOPT would be expensive.

10.4 QPBLUR as an alternative to SQOPT

QPBLUR was designed to be used as an alternative to SQOPT on problems with many degrees of freedom. Instead of working with problem (10.5) it solves the following QP:

$$\begin{aligned} \underset{x,s_n,s_l}{\text{minimize}} \quad & \left(\begin{pmatrix} \sigma g_{\texttt{Obj}} \\ 0 \end{pmatrix} + \sigma l \right)^T (x - x_k) + \tfrac{1}{2}(x - x_k)^T \begin{pmatrix} \bar{H} & \\ & 0 \end{pmatrix} (x - x_k) \\ \text{subject to} \quad & \begin{pmatrix} J_1 & J_2 \\ J_3 & J_4 \end{pmatrix} x - \begin{pmatrix} s_n \\ s_l \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} \\ & \ell_x \le x \le u_x, \quad \ell_n \le s_n \le u_n \quad \ell_l \le s_l \le u_l. \end{aligned} \tag{10.6}$$

This transformation is necessary because QPBLUR does not allow a linear objective term on the slack variables (such as $\sigma s_{\texttt{iObj}}$). Like SQOPT, QPBLUR evaluates the quadratic objective and gradient in terms of the difference $\Delta x = (x - x_k)$ instead of the variables x alone. The augmented system approach described in Section 3.2.2 is used to incorporate low-rank updates to $\bar{H}$ while maintaining sparsity in the KKT matrices.

QPBLUR returns an approximate solution to (10.6) that solves the perturbed problem

$$\begin{aligned} \underset{x,s_n,s_l,r}{\text{minimize}} \quad & \left(\begin{pmatrix} \sigma g_{\texttt{Obj}} \\ 0 \end{pmatrix} + \sigma l \right)^T (x - x_k) + \tfrac{1}{2}(x - x_k)^T \begin{pmatrix} \bar{H} + \delta I & \\ & \delta I \end{pmatrix} (x - x_k) \\ & + \tfrac{1}{2}\delta \left\| s_n - s_{nk} \right\|_2^2 + \tfrac{1}{2}\delta \left\| s_l - s_{lk} \right\|_2^2 + \hat{y}^T r + \tfrac{1}{2}\rho \left\| r \right\|_2^2 \\ \text{subject to} \quad & \begin{pmatrix} J_1 & J_2 \\ J_3 & J_4 \end{pmatrix} x - \begin{pmatrix} s_n \\ s_l \end{pmatrix} + r = \begin{pmatrix} b \\ 0 \end{pmatrix} \\ & \ell_x \le x \le u_x, \quad \ell_n \le s_n \le u_n \quad \ell_l \le s_l \le u_l, \end{aligned} \tag{10.7}$$

where (x_k, s_{lk}, s_{nk}) is the current SQP iterate and slacks. The values of δ, ρ, and $\hat{y}$ are determined by the regularized BCL algorithm to ensure the solution of (10.7) approximates the solution of (10.6) to SNOPT's requested feasibility and optimality tolerance. The primal regularization in problem (10.7) may be interpreted as controlling the length of the search direction $p = (\Delta x, \Delta s_n, \Delta s_l)$, because

$$\left\| p \right\|_2^2 = \left\| x - x_k \right\|_2^2 + \left\| s_n - s_{nk} \right\|_2^2 + \left\| s_l - s_{lk} \right\|_2^2 .$$

A good initial estimate of the Lagrange multipliers for the linearized constraints in (10.6) may be obtained from SNOPT's current estimate y_k of the Lagrange multipliers for the nonlinear constraints. This estimate y_k is used to initialize the multiplier $\hat{y}$ in QPBLUR's regularized BCL algorithm.

When a problem is likely to have many degrees of freedom, it may be advantageous to use QPBLUR to solve all QP subproblems. This is because QPBLUR does not need to start from a vertex and expand the number of superbasic variables. As a result, the number of major iterations (and therefore the number of function and gradient evaluations) may be decreased. If the number of superbasics is not known in advance, SNOPT may employ SQOPT to solve the initial QP subproblems, and then switch over to QPBLUR (as an alternative to solving (10.4) with CG) when n_S becomes large.

10.5 Elastic programming

The constraints of the QP subproblem are a linearization of the nonlinear constraints in problem (NLP). This linearization may be infeasible. If SQOPT detects that problem (10.3) is infeasible, SNOPT enters *elastic* mode, in which an ℓ_1 penalty term is added to the objective of problem (NLP) to minimize the infeasibility of the nonlinear constraints:

$$\begin{array}{ll} \underset{x,v,w}{\text{minimize}} & \varphi(x) + \gamma e^T(v+w) \\ \text{subject to} & \ell \leq \begin{pmatrix} x \\ c(x) - v + w \\ Ax \end{pmatrix} \leq u, \quad v, w \geq 0, \end{array} \tag{NLP(γ))}$$

where $\gamma \geq 0$ is a penalty parameter and e is the vector of all ones. The associated elastic QP subproblem takes the form

$$\begin{array}{ll} \underset{x,s,v,w}{\text{minimize}} & \varphi_k + g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T H_k (x - x_k) + \gamma e^T(v+w) \\ \text{subject to} & c_k + J_k(x - x_k) - s + v - w = 0 \\ & \ell \leq \begin{pmatrix} x \\ s \\ Ax \end{pmatrix} \leq u, \quad v, w \geq 0. \end{array} \tag{10.8}$$

If the nonlinear constraints are later satisfied, γ is set to zero, effectively switching out of elastic mode and back to subproblem (10.3). If (NLP(γ)) is solved to optimality but the nonlinear constraints are *not* satisfied, γ is increased in stages to $O(10^{12})$ before problem (NLP) is declared infeasible.

SQOPT implements elastic programming implicitly. Any bound on the variables or slacks may be elastic. The associated variables are given a piecewise linear objective function that penalizes values outside the "real" bounds. Within SNOPT, only the bounds on the slack variables for nonlinear constraints are made elastic.

Although it is possible for QPBLUR to solve (10.8) by including the variables v and w explicitly, we do not do this. Instead, if a QP is detected to be infeasible, QPBLUR exits and the remaining QP subproblems are solved by SQOPT. Further work is needed to allow QPBLUR to treat elastic bounds implicitly.

Alternatively, SNOPT could be modified to include an ℓ_2 penalty on the infeasibility of the nonlinear constraints. This would yield the QP subproblem

$$\begin{array}{ll} \underset{x,s,r}{\text{minimize}} & \varphi_k + g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T H_k(x - x_k) + \gamma \|r\|_2^2 \\ \text{subject to} & c_k + J_k(x - x_k) - s + r = 0 \\ & \ell \leq \begin{pmatrix} x \\ s \\ Ax \end{pmatrix} \leq u. \end{array} \tag{10.9}$$

The variable r in the equality constraint and the objective can be handled implicitly with QPBLUR's regularized subproblem solver (Algorithm 7).

Chapter 11

Computational results

In this chapter we present computational results obtained with QPBLUR on a variety of test problems. We compare QPBLUR to SQOPT on two sets of convex QPs. We also compare the performance of QPBLUR and SQOPT when they are used as a QP subproblem solver inside SNOPT's sparse SQP method on a set of nonlinear programs.

11.1 Computing platform

Computational results were obtained using a Dell Precision 490 workstation (from 2006) with a 3 GHz Intel 5160 Xeon processor and 4 GB of RAM. Fortran 77 and Fortran 95 software was compiled with gfortran 4.3.2 using 64-bit floating-point arithmetic. All CPU times are in seconds and do not include the time required to load the problem data.

11.2 Comparison with QPBLU

Because QPBLUR is the successor to QPBLU, we wished to make a comprehensive comparison between the two solvers. Limitations of the current QPBLU implementation made this somewhat difficult. QPBLU is restricted to solving separable QPs (those with diagonal Hessians) and cannot start from an arbitrary point or active set provided by the user. Instead, a procedure is required to generate an initial point and active set from which the inertia-controlling method can be safely started. QPBLU is also sensitive to the scaling applied to problem, and because KKT repair is not implemented, if a singular or near singular KKT system arises it must exit.

Huynh [59] presented performance results for QPBLU on a set of 66 problems derived from a test set of LPs available from the University of Florida Sparse Matrix Collection [23]. These LPs are of the form

$$\begin{array}{ll} \underset{x}{\text{minimize}} & c^T x \\ \text{subject to} & Ax = b \\ & \ell \leq x \leq u. \end{array}$$

To construct a QP the quadratic term $\frac{1}{2}x^Tx$ was added to the linear objective. Note that the resulting objective $c^Tx + \frac{1}{2}x^Tx$ is bounded below by $-\frac{1}{2}\|c\|_2^2$.

A MEX interface was constructed to call QPBLU from MATLAB. QPBLU was allowed to choose the initial active set, starting point, and a scaling for the problem. QPBLUR was run on the same scaled problem with this initial active set and starting point. Both QPBLU and QPBLUR may use one of several sparse linear solvers. Each QP solver was run using its best performing linear solver. LUSOL was used with QPBLU and UMFPACK with QPBLUR.

QPBLU failed on 17 problems, it incorrectly classified 14 problems as unbounded, and stopped on 3 problems when a singular KKT matrix occurred. QPBLUR solved all problems. We note that QPBLU is sensitive to the scaling of the problem (when no scaling was applied it failed on 25 problems).

When it was able to solve a problem, QPBLU was generally faster than QPBLUR. Table 11.1 shows the CPU time of QPBLU and QPBLUR on these problems. QPBLUR is able to solve all problems within 10 times the amount of time required by QPBLU. We note that the initial active set and starting point chosen by QPBLU (one with zero degrees of freedom) does not allow QPBLUR to exploit its ability to start with many degrees of freedom.

This trade-off between speed and robustness can be understood, as QPBLUR must solve a sequence of regularized QPs, while QPBLU need only solve a single QP.

Table 11.1: Problem dimensions, degrees of freedom, and CPU times for QPBLU and QPBLUR

#	Problem	m	n	dof	QPBLU	QPBLUR
1	80bau3b	2262	12061	9935		49.85
2	adlittle	56	138	61	0.02	0.14
3	afiro	27	51	12	0.01	0.01
4	agg	488	615	123	0.11	0.39
5	agg2	516	758	101	0.31	1.28
6	agg3	516	758	101	0.34	1.32
7	bandm	305	472	168	0.33	1.58
8	beaconfd	173	295	131	0.05	0.14
9	blend	74	114	31		0.11
10	bnl2	2324	4486	2090		162.33
11	capri	271	482	215	0.26	1.15
12	czprob	929	3562	2493	2.14	8.46
13	d2q06c	2171	5831	2545		166.93
14	e226	223	472	186	0.24	1.09
15	etamacro	400	816	351	0.86	2.57
16	fffff800	524	1028	563		2.59
17	finnis	497	1064	581	0.77	2.62
18	fit1d	24	1049	999	0.96	3.17
19	fit1p	627	1677	855	3.31	7.78
20	fit2d	25	10524	10288	118.40	429.06
21	fit2p	3000	13525	7613	304.36	535.21
22	ganges	1309	1706	447	1.03	4.33
23	grow15	300	645	145	0.53	2.80
24	grow22	440	946	198		4.73
25	grow7	140	301	86	0.12	0.62
26	israel	174	316	85	0.19	0.86
27	kb2	43	68	22	0.01	0.02
28	lotfi	153	366	83	0.14	0.60
29	maros	846	1966	1314		11.93
30	maros_r7	3136	9408	4759		208.94
31	osa_07	1118	25067	23939	23.47	82.69
32	osa_14	2337	54797	52455	120.44	375.77
33	osa_30	4350	104374	100024	509.00	1501.12
34	osa_60	10280	243246	232969	2981.95	8519.39
35	perold	625	1506	828		33.82
36	pilot	1441	4860	3183		332.61
37	pilot4	410	1123	700		3.22
38	pilot_ja	940	2267	1283		32.20
39	pilot_we	722	2928	2077		15.90
40	pilot87	2030	6680	4428		1160.01
41	pilotnov	975	2446	1436		45.29
42	recipe	91	204	134	0.01	0.01
43	sc105	105	163	15	0.02	0.10
44	sc205	205	317	49	0.06	0.48
45	sc50a	50	78	5	0.01	0.01
46	sc50b	50	78	3	0.01	0.02
47	scagr7	129	185	29	0.05	0.27
48	scfxm1	330	600	254	0.26	1.62
49	scfxm2	660	1200	509		4.62
50	scfxm3	990	1800	778		11.87
51	scrs8	490	1275	784	0.55	2.66
52	scsd1	77	760	679	0.16	0.57
53	scsd6	147	1350	1190	0.51	2.44
54	scsd8	397	2750	2029	2.18	18.21
55	sctap1	300	660	257	0.21	1.13
56	sctap2	1090	2500	936	1.77	8.83
57	sctap3	1480	3340	1167	3.16	17.56
58	share1b	117	253	130	0.11	0.33
59	share2b	96	162	54	0.04	0.16
60	stair	356	614	217	0.60	3.56
61	standata	359	1274	962	0.18	0.73
62	standmps	467	1274	852	0.63	2.15
63	stocfor1	117	165	47	0.01	0.02
64	stocfor2	2157	3045	798	1.13	11.47
65	stocfor3	16675	23541	6137	67.64	657.73
66	woodw	1098	8418	7663	8.63	81.11

11.3 Results for convex quadratic programming

In this section we evaluate QPBLUR's ability to solve convex QPs of the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x + \tfrac{1}{2} x^T H x \\ \text{subject to} \quad & \ell \leq \begin{pmatrix} x \\ Ax \end{pmatrix} \leq u. \end{aligned}$$

These problems arise from two different test sets: the CUTEr test set and the Maros and Mészáros convex QP test set.

11.3.1 Results on convex CUTEr QPs with many degrees of freedom

CUTEr (the Constrained and Unconstrained Testing Environment, revisited) [51, 22] is a platform for testing optimization solvers. CUTEr contains a collection of more than 1000 test problems ranging from linear programming through convex and nonconvex quadratic programming to general nonlinear programming. CUTEr test problems are represented in the SIF format. A set of Fortran routines for decoding problems and calculating function values and derivatives is included in CUTEr.

We compare the performance of QPBLUR and SQOPT on a set of convex CUTEr QPs with many degrees of freedom at the solution. Recall that the number of degrees of freedom is given by $|\mathcal{M}| - m$. For SQOPT the number of degrees of freedom is equal to n_S the number of superbasic variables. To obtain these results, interfaces from CUTEr to SQOPT and the Fortran 95 version of QPBLUR were constructed. These interfaces were based on code provided by Wong [111].

In addition to problem data, CUTEr provides an initial point x_0. Both SQOPT and QPBLUR are cold started from this initial point (no initial active set is provided). SQOPT selects an initial active set (and an initial basis B) using a crash procedure [53, 74].

When QPBLUR is cold started, all variables (x, s) are initialized to be moving variables. The initialization procedure, described in Section 9.7, removes variables with fixed bounds. If some of the initial variables are on their bounds, QPBLUR will begin from a degenerate point and will require a sequence of degenerate steps to remove those variables from $\mathcal{M}$. Because we expect $|\mathcal{M}| \gg |\mathcal{N}|$ for these problems, this is an acceptable initial active set.

QPBLUR may use one of several sparse linear solvers. The results presented here were obtained using UMFPACK to factor and solve the KKT systems.

The CUTEr test set contains 172 QPs. Appendix B contains a classification of the CUTEr QPs. There are 91 convex QPs. However, most of these have few degrees of freedom. There are only 22 convex QPs with linear equality constraints and a thousand or more degrees of freedom. We selected 20 of these problems, each with more than a thousand degrees of freedom, to evaluate the performance of QPBLUR and SQOPT. (Problems `ALLINQP` and `RDW2D51U` are not included in this test set because they could not be decoded with the initial CUTEr settings selected.)

These selected problems are described in Table 11.2. The column headings there are as follows: `n` is the number of variables, `m` is the number of linear constraints (excluding bounds), `nnz(A)` is

Table 11.2: Structure and optimal objective value of convex CUTEr QPs with many degrees of freedom

Name	n	m	nnz(A)	nnz(H)	nnzOD	Objective
AUG2D	20200	10000	40000	19800	0	1.68741e+06
AUG2DC	20200	10000	40000	20200	0	1.81837e+06
AUG2DCQP	20200	10000	40000	20200	0	6.49813e+06
AUG2DQP	20200	10000	40000	19800	0	6.23701e+06
AUG3D	27543	8000	50286	22743	0	2.45615e+04
AUG3DC	27543	8000	50286	27543	0	2.76541e+04
AUG3DCQP	27543	8000	50286	27543	0	6.15604e+04
AUG3DQP	27543	8000	50286	22743	0	5.42290e+04
CVXQP1	10000	5000	14998	39984	29984	† 1.08705e+08
CVXQP2	10000	2500	7499	39984	29984	† 8.18425e+07
DTOC3	4499	2998	10493	4497	0	‡ 2.35216e+02
GOULDQP3	19999	9999	29997	39995	19999	3.27947e-05
HUESTIS	5000	2	10000	5000	0	1.74122e+11
HUES-MOD	5000	2	10000	5000	0	3.48245e+07
MOSARQP1	2500	700	3422	2545	45	‡ -3.82141e+03
MOSARQP2	2500	700	3422	2545	45	‡ -5.05259e+03
PRIMAL4	1489	75	16031	1488	0	-7.46090e-01
STCQP1	8193	4095	28665	57333	49140	3.67100e+05
STCQP2	8193	4095	28665	57333	49140	3.71893e+04
UBH1	9009	6000	24000	3003	0	1.11600e+00

the number of nonzeros in A, `nnz(H)` is the number of nonzeros in the lower-triangular portion of H, `nnzOD` is the number of nonzeros in the off-diagonal lower-triangular portion of H (if `nnzOD` is zero, the problem is separable), and `Objective` is a reference optimal objective value obtained with IPOPT and published in [108]. Some problems contain variable dimensions, and so the dimensions used here differ from those computed with IPOPT. Objective values marked with (†) and (‡) had to be obtained from other sources. Values marked with (†) are from results with BPMPD [73], and those marked with (‡) are from results with KNITRO [109].

Figure 11.1 displays the distribution of the number of variables and the number of degrees of freedom of the problems in this test set. All problems have more than a 1000 degrees of freedom, and nearly half have more than 20000 variables and over 5000 degrees of freedom.

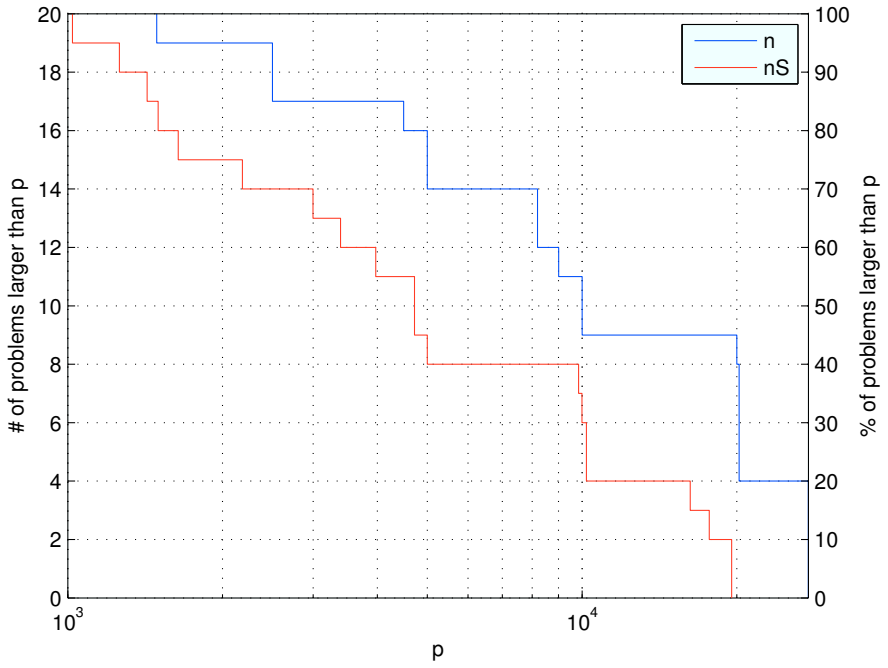

Figure 11.1: Distribution of variables (`n`) and degrees of freedom (`nS`) in convex CUTEr QPs

Table 11.3: Results for QPBLUR and SQOPT on the convex CUTEr QPs

Name	Q Objective	S Objective	Q itn	S itn	Q nS	S nS	Q sec	S sec
AUG2D	1.68741e+06	1.67751e+06	3	10616	10200	10192	0.58	2045.60
AUG2DC	1.81837e+06	1.80827e+06	3	10622	10200	10200	0.54	2024.88
AUG2DCQP	6.49813e+06	6.48803e+06	10441	14296	9994	9994	235.91	2283.09
AUG2DQP	6.23701e+06	6.22711e+06	10061	13667	9844	9801	225.96	2279.84
AUG3D	2.45615e+04	1.31900e+04	2	17647	19543	16909	1.45	3304.44
AUG3DC	2.76541e+04	1.38826e+04	2	19707	19543	19543	1.21	2854.65
AUG3DCQP	6.15604e+04	4.77889e+04	11554	22071	17677	17665	391.28	3537.12
AUG3DQP	5.42290e+04	4.28575e+04	8270	18288	16229	13714	303.57	2114.44
CVXQP1	1.08705e+08	1.08705e+08	5695	11519	1260	1276	4699.81	81.88
CVXQP2	8.18425e+07	8.18425e+07	7714	8770	2186	2210	692.52	127.96
DTOC3	2.35218e+02	2.35216e+02	2	1830	1499	1499	0.06	9.24
GOULDQP3	2.75344e-05	-3.81495e+04	5009	7511	5000	4988	5865.05	580.30
HUES-MOD‡	3.48245e+07	1.00000e+00	282	0	4721	0	1.52	0.08
HUESTIS‡	1.74122e+11	5.00000e+03	289	0	4721	0	1.63	0.06
MOSARQP1	-3.82141e+03	-3.82141e+03	1694	3492	1021	1021	7.77	6.87
MOSARQP2	-5.05259e+03	-5.05259e+03	878	2610	1640	1640	3.24	6.22
PRIMAL4	-7.46090e-01	-7.46091e-01	65	1299	1427	1140	0.34	3.70
STCQP1	3.67100e+05	3.67101e+05	995	7493	3393	5707	117.34	147.12
STCQP2	3.71893e+04	3.71893e+04	132	8080	3970	3970	6.90	179.61
UBH1‡	1.25215e+00	1.58394e+00	14	90090	2997	2260	92.15	9121.12

Table 11.3 contains the optimal objective value (`Objective`), the number of iterations (`itn`), the degrees of freedom at the optimal solution (`nS`), and the CPU seconds (`sec`) computed by QPBLUR and SQOPT on these problems. Columns beginning with `Q` correspond to QPBLUR. Columns beginning with `S` correspond to SQOPT. QPBLUR solved all problems in the test set. SQOPT failed to solve the problems marked with (‡). (Problems `HUES-MOD` and `HUESTIS` failed quickly because SQOPT could not satisfy the constraints $Ax - s = 0$ sufficiently well.) SQOPT reached the iteration limit on the problem `UBH1`. Note that the optimal objective values computed by QPBLUR agree with the reference values in Table 11.2. On all problems, QPBLUR performs fewer iterations than SQOPT.

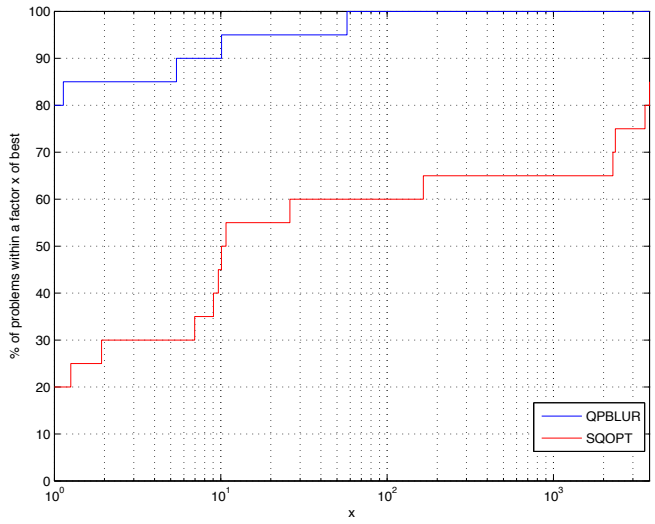

Figure 11.2: Performance profile for CPU time on 20 convex CUTEr QPs with many degrees of freedom (QPBLUR and SQOPT)

We use a performance profile to visualize the results in Table 11.3. Performance profiles are explained in depth in Dolan and Moré [26]. We provide a brief explanation here. Consider a test set $\mathcal{P}$ contain n_p problems on which we profile n_s solvers. Suppose

$$t_{p,s} = \text{time needed to solve problem } p \text{ with solver } s$$

is the quantity we wish to profile. Note that $t_{p,s}$ may be other quantities of interest (*e.g.* the number of iterations or function evaluations). Then each solver has an associated curve

$$P_s(x) = \frac{1}{n_p} \# \left\{ p \in \mathcal{P} : t_{p,s} \leq x \left(\min_{\sigma} t_{p,\sigma} \right) \right\}.$$

For a particular value $x \geq 1$, $P_s(x)$ denotes the fraction of problems in the test set solved by solver s within a factor x of the best time. $P_s(1)$ is the fraction of problems for which solver s is the fastest. Performance profiles are plotted from 1 to M, where M is the maximum factor of all solvers. $P_s(M)$ is the fraction of problems in the test set solved by solver s.

Figure 11.2 contains a performance profile of the CPU time for SQOPT and QPBLUR. We note that QPBLUR is faster than SQOPT on 80% of the test problems, and solves every problem within a factor of 60 of SQOPT's time. SQOPT is faster on 20% of the problems, but only solves 85% of the problems. On several problems, SQOPT required more than 3000 times the amount of time required by QPBLUR.

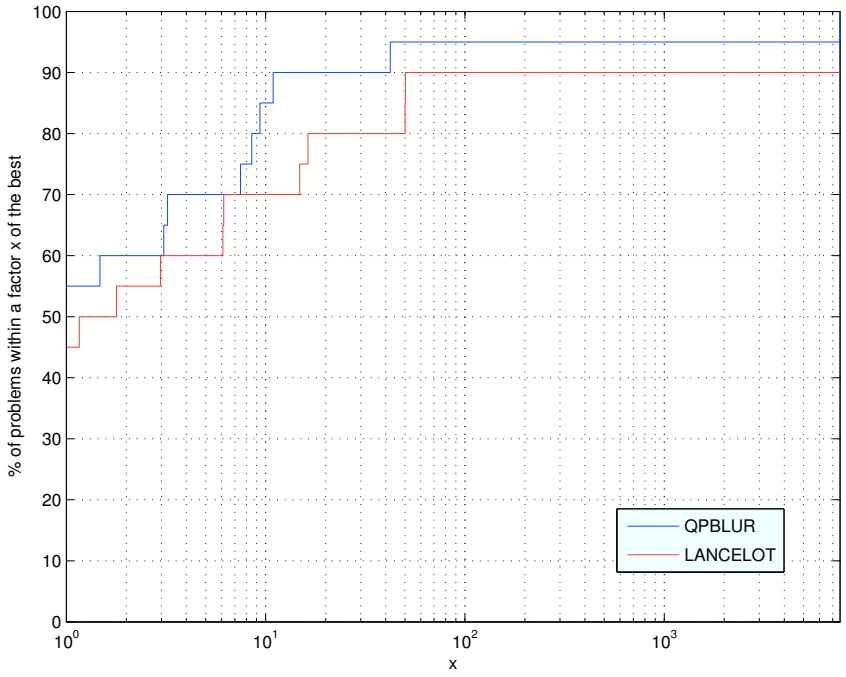

Figure 11.3: Performance profile for CPU time on 20 CUTEr QPs with many degrees of freedom (QPBLUR and LANCELOT)

11.3.2 Comparison with LANCELOT

Because LANCELOT is a popular implementation of the BCL algorithm, we thought it relevant to compare it to QPBLUR. LANCELOT was run on the previous test set of 20 CUTEr QPs with 1000 or more degrees of freedom. The option `quadratic-problem` was enabled to allow LANCELOT to take some advantage of the quadratic objective and linear constraints. The default values of all other options were used. In particular, linear systems were solved with the conjugate-gradient method using a band preconditioner with semibandwidth 5.

Figure 11.3 gives a performance profile of CPU time for LANCELOT and QPBLUR, showing a moderate lead for QPBLUR. However, both solvers have a similar profile. To solve the BCL subproblems, LANCELOT uses a gradient-projection method that allows for many changes to the active set at each iteration, while QPBLUR can only perform a single change per iteration. Indeed, LANCELOT generally performs fewer iterations than QPBLUR, indicating that the gradient-projection algorithm is beneficial. However, CG on systems involving the Hessian of the augmented Lagrangian may require many iterations to converge. This might explain why LANCELOT tends to consume more CPU time than QPBLUR.

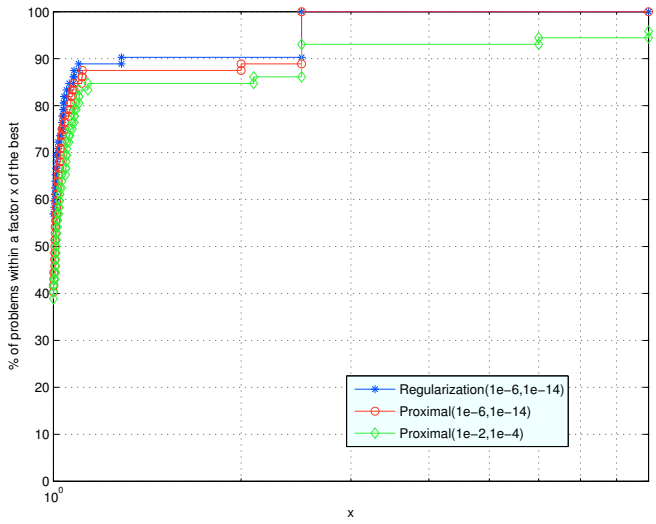

Figure 11.4: Performance profile for CPU time of regularization and proximal point methods

11.3.3 Comparison between proximal point and regularization

QPBLUR uses the regularized BCL algorithm by default, but includes an option to use the proximal method of multipliers (described in Section 6.6). The difference between the two algorithms is the penalty term applied to the primal variables. The regularized BCL algorithm adds the term $\frac{1}{2}\delta_k \|x\|_2^2$ to the augmented Lagrangian objective, while the proximal method of multipliers adds the term $\frac{1}{2}\delta_k \|x - x_k\|_2^2$, where x_k is the previous subproblem solution.

In this section we compare the performance of the two algorithms on a test set of 72 convex QPs from CUTEr. In both algorithms, two crucial parameters are the initial and final values of δ_k. These are denoted by δ_0 and $\delta_{\min}$, respectively.

Figures 11.4 and 11.5 give performance profiles of CPU time and number of iterations for regularization with $(\delta_0, \delta_{\min}) = (10^{-6}, 10^{-14})$, and the proximal method of multipliers with $(\delta_0, \delta_{\min}) = (10^{-6}, 10^{-14})$ and $(10^{-2}, 10^{-4})$. Regularization is the fastest method, but the proximal method of multipliers is nearly identical when the same small $(\delta_0, \delta_{\min})$ is used.

An advantage of the proximal method of multipliers is that larger δ may be used. This helps ensure that the perturbed Hessian $H+\delta I$ is far from singular. Unfortunately, large δ can lead to many more iterations and subproblems. On problems `PRIMALC1` and `PRIMALC8` the proximal method of multipliers with $(10^{-2}, 10^{-4})$ failed to converge within the specified number of subproblems, because the initial point x_0 was far from the solution. Problem `UBH1` is another example of where large δ can be detrimental. On `UBH1` the proximal method of multipliers with $(10^{-2}, 10^{-4})$ required 1197 iterations, compared with 17 for $(10^{-6}, 10^{-14})$. These results indicate that the proximal method of multipliers might benefit from choosing $(\delta_0, \delta_{\min})$ dynamically based on the scaling of the problem.

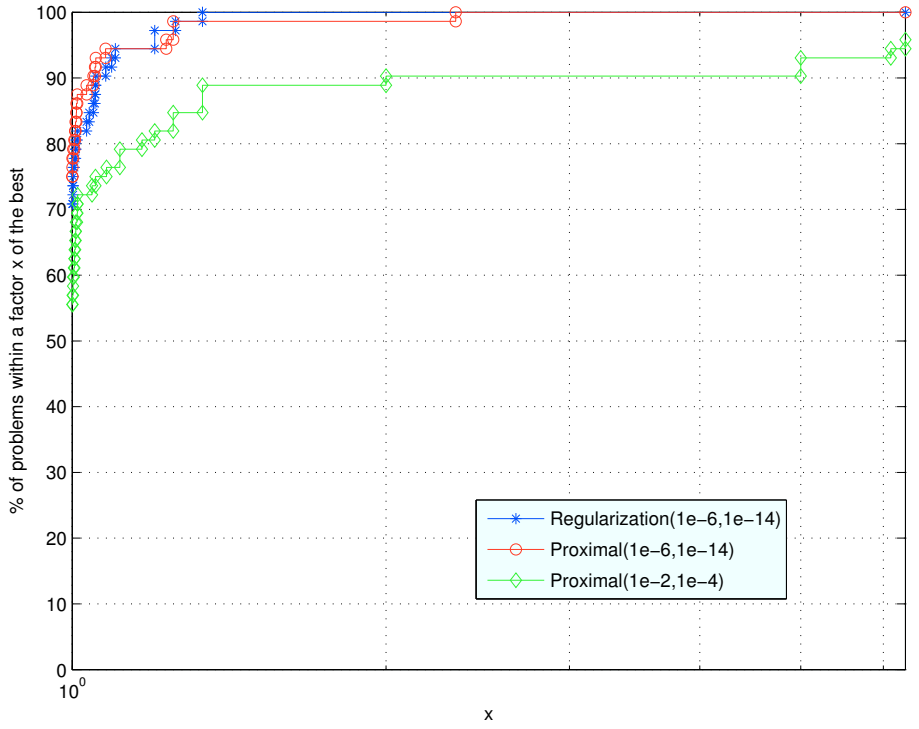

Figure 11.5: Performance profile for iterations of regularization and proximal point methods

11.3.4 Results on the Maros and Mészáros convex QP test set

The Maros and Mészáros convex QP test set [73, 72] is a collection of 138 problems. These problems originate from several sources: 76 problems come from the CUTE library (the predecessor to CUTEr), 46 were provided by the Brunel optimization group, and the remaining 16 come from miscellaneous sources. These problems were collected in order for the authors to test their interior-point QP solver BPMPD [77, 10]. The test set includes both separable and nonseparable problems, and a reference optimal objective value computed by BPMPD.

The problems are available in the QPS format. QPS is a generalization of the MPS format [80] for linear programming. A parser was implemented in Python to extract the sparse matrices and vectors from problems in QPS format and save them in MATLAB MAT-File format [76]. These extracted problems were used to test the MATLAB version of QPBLUR. Because the QPS format is a subset of SIF, the CUTEr interface discussed in the previous section may also be used to access these problems. We present results obtained with the CUTEr interface described in the previous section.

Figure 11.6 shows the distribution of variables and degrees of freedom of problems in the test set. Note that less than 20 problems in this test set have more than 1000 degrees of freedom, and less than 10 have more than 3000 degrees of freedom. Because there is some overlap with the CUTEr QP test set in the previous section, many of the problems with many degrees of freedom were already solved (although perhaps with different dimensions).

Table 11.4 presents results of running QPBLUR on 130 problems in test set. Problems `BOYD1`, `BOYD2`, `CONT-200`, `CONT-201`, `CONT-300` were excluded. Computing the LU factors of the KKT matrices in the `BOYD` and `CONT` problems was prohibitively expensive and solving these problems can take many hours. `EXDATA` caused an error to be thrown in CUTEr routine `cdimsh`. Problems `Q25FV47` and `QSHIP12L` were solved later, but are not included in this test set because of errors that occurred during decoding of their SIF files.

The columns in Table 11.4 are as follows: `m` is the number of constraints, `n` is the number of variables, `Objective` is the optimal objective value, `Residual` is the relative residual of the linear constraints, `Iter` is the number of iterations, `nS` is the degrees of freedom, `Fac` is the number of KKT factorizations performed, `Time` is the CPU time, and `Inform` is QPBLUR's return code.

All problems except `CONT-101` and `QFFFFF80` were solved. QPBLUR exceeded the iteration limit on `CONT-101` and incorrectly classified `QFFFFF80` as infeasible (probably because of poor scaling in the problem).

Because most problems in the test set contain only a few degrees of freedom, SQOPT's reduced-gradient method outperforms QPBLUR's fullspace method in almost all cases. This can be seen in the performance profile in Figure 11.7. We present these results only to show that QPBLUR is a robust quadratic programming method capable of solving a wide variety of problems.

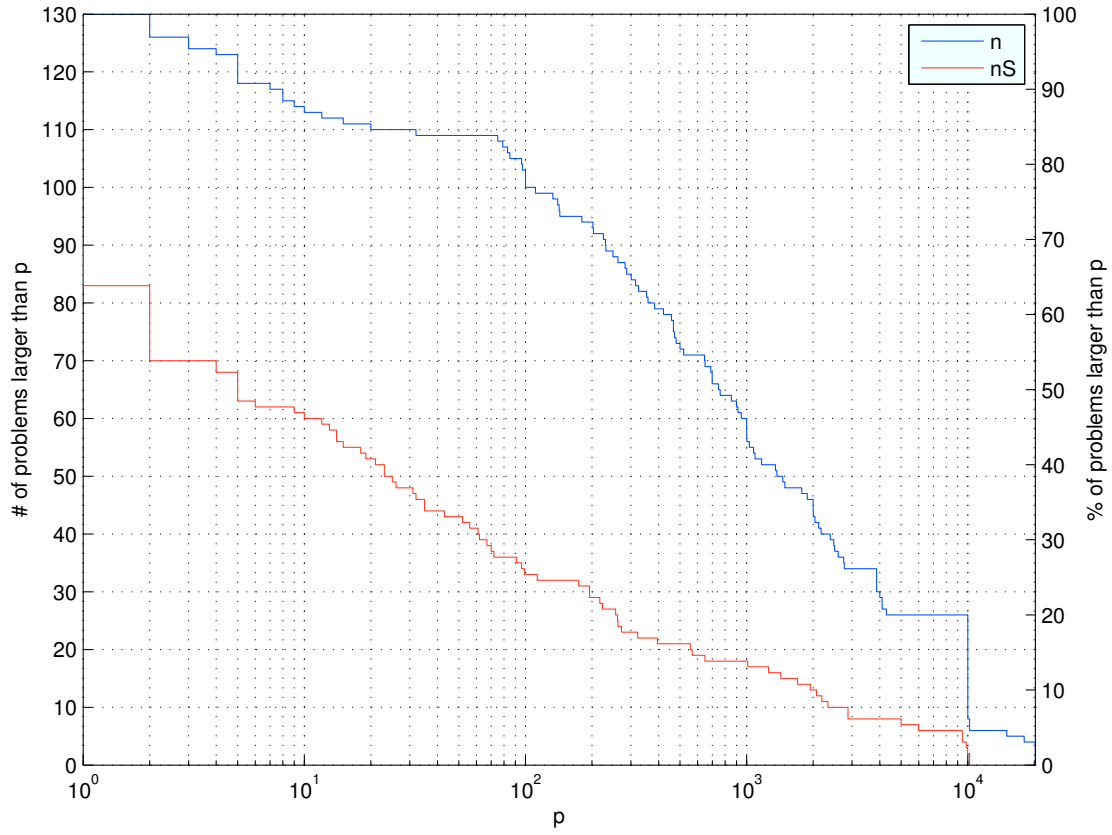

Figure 11.6: Distribution of variables (`n`) and degrees of freedom (`nS`) in the Maros and Mészáros test set

Table 11.4: Results of QPBLUR on the Maros and Mészáros convex QP test set

Name	m	n	Objective	Residual	Iter	nS	Fac	Time	Inform
AUG2D	10000	20200	1.68741e+06	5.3e-12	3	10200	3	0.55	0
AUG2DC	10000	20200	1.81837e+06	6.0e-12	3	10200	3	0.57	0
AUG2DCQP	10000	20200	6.49813e+06	2.3e-14	10441	9994	51	240.73	0
AUG2DQP	10000	20200	6.23701e+06	3.8e-14	10061	9844	45	228.72	0
AUG3D	1000	3873	5.54068e+02	2.1e-12	2	2873	2	0.07	0
AUG3DC	1000	3873	7.71262e+02	3.0e-12	2	2873	2	0.06	0
AUG3DCQP	1000	3873	9.93362e+02	2.1e-10	625	2333	4	3.85	0
AUG3DQP	1000	3873	6.75238e+02	1.3e-10	813	2072	6	5.02	0
CONT-050	2401	2597	-4.56375e+00	9.5e-10	4	195	2	0.84	0
CONT-100	9801	10197	-4.64439e+00	4.4e-11	5	395	3	26.37	0
CONT-101	10098	10197	1.37864e+06	1.6e-02	30591	0	4986	35855.97	-1
CVXQP1_L	5000	10000	1.08705e+08	4.8e-11	7389	1257	69	4516.19	0
CVXQP1_M	500	1000	1.08751e+06	6.6e-11	741	113	12	5.94	0
CVXQP1_S	50	100	1.15907e+04	1.3e-11	76	13	4	0.04	0
CVXQP2_L	2500	10000	8.18425e+07	6.0e-10	9101	2191	108	670.50	0
CVXQP2_M	250	1000	8.20155e+05	1.7e-11	908	217	10	4.24	0
CVXQP2_S	25	100	8.12094e+03	1.3e-12	86	21	3	0.04	0
CVXQP3_L	7500	10000	1.15711e+08	8.5e-12	10631	0	125	14069.08	0
CVXQP3_M	750	1000	1.36283e+06	6.5e-11	1312	35	52	17.10	0
CVXQP3_S	75	100	1.19434e+04	1.7e-11	73	1	6	0.04	0
DPKLO1	77	133	3.70131e-01	1.4e-13	2	56	2	0.01	0
DTOC3	9998	14999	2.35263e+02	3.9e-11	3	4999	3	0.32	0
DUAL1	1	85	3.50130e-02	1.8e-14	46	62	2	0.03	0
DUAL2	1	96	3.37337e-02	8.6e-15	7	91	2	0.01	0
DUAL3	1	111	1.35756e-01	6.7e-15	26	96	2	0.03	0
DUAL4	1	75	7.46091e-01	6.0e-14	16	61	2	0.01	0
DUALC1	215	9	6.15526e+03	2.0e-10	23	2	4	0.02	0
DUALC2	229	7	3.55132e+03	1.4e-10	12	2	4	0.01	0
DUALC5	278	8	4.27234e+02	5.0e-10	7	4	3	0.01	0
DUALC8	503	8	1.83094e+04	4.2e-10	11	2	4	0.02	0
GENHS28	8	10	9.27174e-01	6.7e-14	2	2	2	0.00	0
GOULDQP2	349	699	1.87272e-04	1.4e-14	5216	322	19	7.76	0
GOULDQP3	349	699	2.06309e+00	1.3e-12	179	174	2	0.87	0
HS118	17	15	6.64821e+02	3.6e-12	59	0	2	0.01	0
HS21	1	2	-9.99598e+01	9.5e-13	3	1	1	0.00	0
HS268	5	5	1.93896e-05	9.6e-13	2	5	2	0.00	0
HS35	1	3	1.11111e-01	6.2e-15	4	2	2	0.00	0
HS35MOD	1	3	2.50000e-01	1.5e-12	2	2	2	0.00	0
HS51	3	5	1.05000e-07	5.6e-13	2	2	2	0.00	0
HS52	3	5	5.32665e+00	3.9e-11	2	2	2	0.00	0
HS53	3	5	4.09302e+00	1.9e-11	2	2	2	0.00	0
HS76	3	4	-4.68182e+00	4.6e-13	8	2	2	0.00	0
HUESTIS	2	10000	3.48245e+11	2.2e-10	561	9444	7	5.44	0
HUES-MOD	2	10000	3.48245e+07	5.2e-11	559	9444	5	5.32	0
KSIP	1001	20	5.75798e-01	3.2e-15	3742	18	35	9.64	0
LASER	1000	1002	2.40960e+06	6.8e-13	1770	70	9	6.58	0
LISWET1	10000	10002	3.58915e+01	4.8e-10	10646	2	74	213.02	0
LISWET10	10000	10002	4.94522e+01	1.9e-10	11144	43	74	215.75	0
LISWET11	10000	10002	4.95182e+01	2.6e-10	11479	72	71	223.19	0
LISWET12	10000	10002	1.73692e+03	9.4e-11	12076	6	78	234.96	0
LISWET2	10000	10002	2.49980e+01	4.5e-10	11291	14	65	223.23	0
LISWET3	10000	10002	2.50012e+01	3.0e-10	10809	261	59	212.28	0
LISWET4	10000	10002	2.50001e+01	7.6e-10	10934	272	61	214.23	0
LISWET5	10000	10002	2.50343e+01	4.0e-10	10801	256	59	212.05	0
LISWET6	10000	10002	2.49957e+01	5.6e-10	10829	223	60	211.28	0
LISWET7	10000	10002	4.98817e+02	1.9e-10	10407	2	73	203.76	0
LISWET8	10000	10002	7.14437e+02	2.0e-10	11270	32	72	217.98	0
LISWET9	10000	10002	1.96325e+03	8.7e-11	11882	5	76	230.47	0
LOTSCHD	7	12	2.39842e+03	1.0e-14	13	0	3	0.00	0
MOSARQP1	700	2500	-9.52875e+02	3.5e-13	2230	1012	50	7.14	0
MOSARQP2	600	900	-1.59748e+03	8.8e-14	1365	568	33	3.10	0
POWELL20	10000	10000	5.20896e+10	7.4e-11	10006	0	56	190.45	0
PRIMAL1	85	325	-3.50130e-02	4.9e-10	76	262	2	0.13	0
PRIMAL2	96	649	-3.37337e-02	3.0e-14	101	557	2	0.25	0
PRIMAL3	111	745	-1.35756e-01	5.2e-10	107	648	2	0.64	0
PRIMAL4	75	1489	-7.46091e-01	4.2e-13	65	1427	2	0.34	0
PRIMALC1	9	230	-6.09264e+03	9.0e-13	222	14	3	0.34	0
PRIMALC2	7	231	-3.54826e+03	2.7e-12	236	1	3	0.34	0
PRIMALC5	8	287	-4.27222e+02	2.0e-12	285	5	3	0.36	0
PRIMALC8	8	520	-1.82745e+04	6.9e-12	513	15	5	0.87	0
QADLITTL	56	97	4.80319e+05	3.2e-10	325	4	9	0.08	0
QAFIRO	27	32	-1.59078e+00	1.5e-13	77	0	5	0.01	0
QBANDM	305	472	1.63545e+04	3.6e-11	2115	0	203	2.18	0
QBEACONF	173	262	1.64712e+05	3.2e-14	477	0	24	0.31	0
QBORE3D	233	315	3.10186e+03	6.8e-12	630	0	38	0.36	0

Name	m	n	Objective	Residual	Iter	nS	Fac	Time	Inform
QBRANDY	220	249	2.83758e+04	1.5e-11	1328	0	84	0.82	0
QCAPRI	271	353	6.67933e+07	1.4e-12	1510	0	131	1.27	0
QE226	223	282	2.12992e+02	5.4e-15	2221	23	183	1.86	0
QETAMACR	400	688	8.67604e+04	1.5e-10	2325	35	201	4.22	0
QFFFFF80	524	854	3.64501e+12	5.7e-07	2783	0	222	5.39	-2
QFORPLAN	161	421	7.45690e+09	1.3e-13	1109	9	33	1.26	0
QGFRDXPN	616	1092	1.00791e+11	4.0e-15	5244	0	460	6.24	0
QGROW15	300	645	-1.01512e+08	2.2e-16	2931	99	77	3.77	0
QGROW22	440	946	-1.49369e+08	2.9e-16	4788	195	171	8.42	0
QGROW7	140	301	-4.27132e+07	1.1e-15	1161	19	65	0.84	0
QISRAEL	174	142	2.53478e+07	1.7e-10	1070	5	33	0.66	0
QPCBLEND	74	83	-7.84254e-03	2.1e-10	260	0	4	0.10	0
QPCBOEI1	351	384	1.15055e+07	2.1e-10	3460	67	299	3.68	0
QPCBOEI2	166	143	8.17196e+06	3.9e-15	1102	31	116	0.78	0
QPCSTAIR	356	467	6.20439e+06	8.0e-10	1088	26	18	1.61	0
QPILOTNO	975	2172	4.73097e+06	2.6e-10	11451	0	1459	55.42	0
QPTEST	2	2	4.37188e+00	2.6e-12	7	1	2	0.00	0
QRECIPE	91	180	-2.66616e+02	3.8e-12	47	52	2	0.02	0
QSC205	205	203	-5.81395e-03	1.9e-12	237	0	4	0.30	0
QSCAGR25	471	500	2.01740e+08	4.2e-11	2516	0	92	2.58	0
QSCAGR7	129	140	2.68659e+07	4.9e-11	400	0	11	0.13	0
QSCFXM1	330	457	1.68828e+07	1.8e-11	1694	2	89	1.61	0
QSCFXM2	660	914	2.77832e+07	1.8e-12	3344	0	212	5.82	0
QSCFXM3	990	1371	3.08280e+07	1.9e-12	5175	0	320	12.18	0
QSCORPIO	388	358	1.88051e+03	4.4e-14	1514	0	42	0.99	0
QSCRS8	490	1169	9.04572e+02	4.1e-11	4371	0	222	5.25	0
QSCSD1	77	760	8.66667e+00	5.0e-12	1128	0	43	0.95	0
QSCSD6	147	1350	5.08082e+01	4.0e-13	3914	0	151	4.64	0
QSCSD8	397	2750	9.40764e+02	1.1e-14	9000	12	402	29.78	0
QSCTAP1	300	480	1.41586e+03	1.7e-14	2355	0	180	2.71	0
QSCTAP2	1090	1880	1.73503e+03	4.1e-13	8442	0	268	22.41	0
QSCTAP3	1480	2480	1.43875e+03	6.9e-14	10816	0	617	39.33	0
QSEBA	515	1028	8.14821e+07	1.4e-10	1073	0	40	1.58	0
QSHARE1B	117	225	7.25284e+05	1.7e-13	1037	10	72	0.40	0
QSHARE2B	96	79	1.17037e+04	6.9e-10	355	0	16	0.11	0
QSHELL	536	1775	1.57264e+12	2.2e-15	4142	25	139	23.37	0
QSHIP04L	402	2118	2.42002e+06	3.1e-12	2721	0	46	6.36	0
QSHIP04S	402	1458	2.42499e+06	7.9e-12	1966	0	34	3.55	0
QSHIP08L	778	4283	2.37604e+06	9.5e-10	5497	0	83	48.37	0
QSHIP08S	778	2387	2.38573e+06	2.8e-10	3139	0	43	13.18	0
QSHIP12S	1151	2763	3.05696e+06	5.3e-10	4417	0	70	23.35	0
QSIERRA	1227	2036	2.37504e+07	6.6e-10	4793	0	161	12.19	0
QSTAIR	356	467	7.98545e+06	3.7e-11	1232	0	30	1.90	0
QSTANDAT	359	1075	6.41184e+03	2.9e-11	1242	0	21	2.68	0
S268	5	5	1.93896e-05	9.6e-13	2	5	2	0.00	0
STADAT1	3999	2001	-2.85269e+07	1.5e-11	3126	0	17	36.94	0
STADAT2	3999	2001	-3.26266e+01	5.4e-12	2838	0	17	39.34	0
STADAT3	7999	4001	-3.57794e+01	2.1e-12	5389	0	33	95.97	0
STCQP1	2052	4097	1.55144e+05	9.2e-13	359	1699	4	12.28	0
STCQP2	2052	4097	2.23273e+04	1.4e-12	109	1940	3	1.87	0
TAME	1	2	7.50000e-09	2.5e-13	2	1	2	0.00	0
UBH1	12000	18009	1.37129e+00	1.8e-10	74	5997	3	759.14	0
VALUES	1	202	-1.39662e+00	2.4e-10	309	23	2	0.61	0
YAO	2000	2002	1.97704e+02	1.2e-10	2213	1	28	9.38	0
ZECEVIC2	2	2	-4.12500e+00	1.2e-12	7	1	2	0.00	0

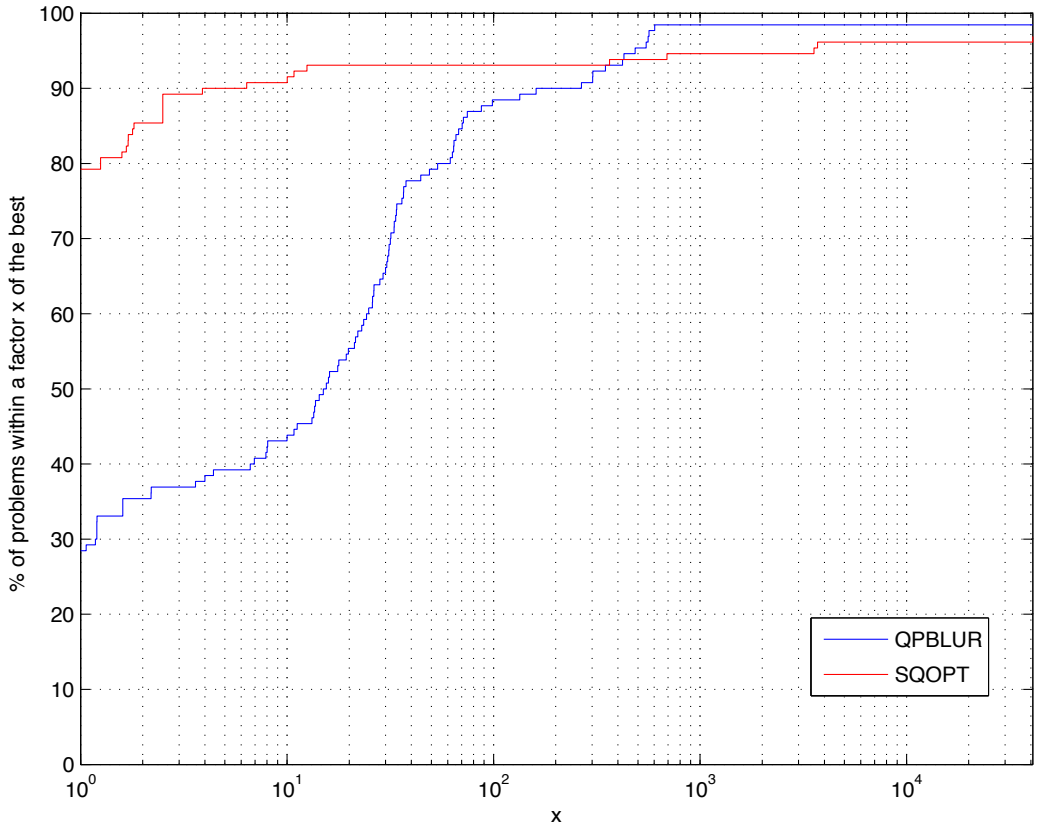

Figure 11.7: Performance profile for CPU time for QPBLUR on the Maros and Mészáros test set

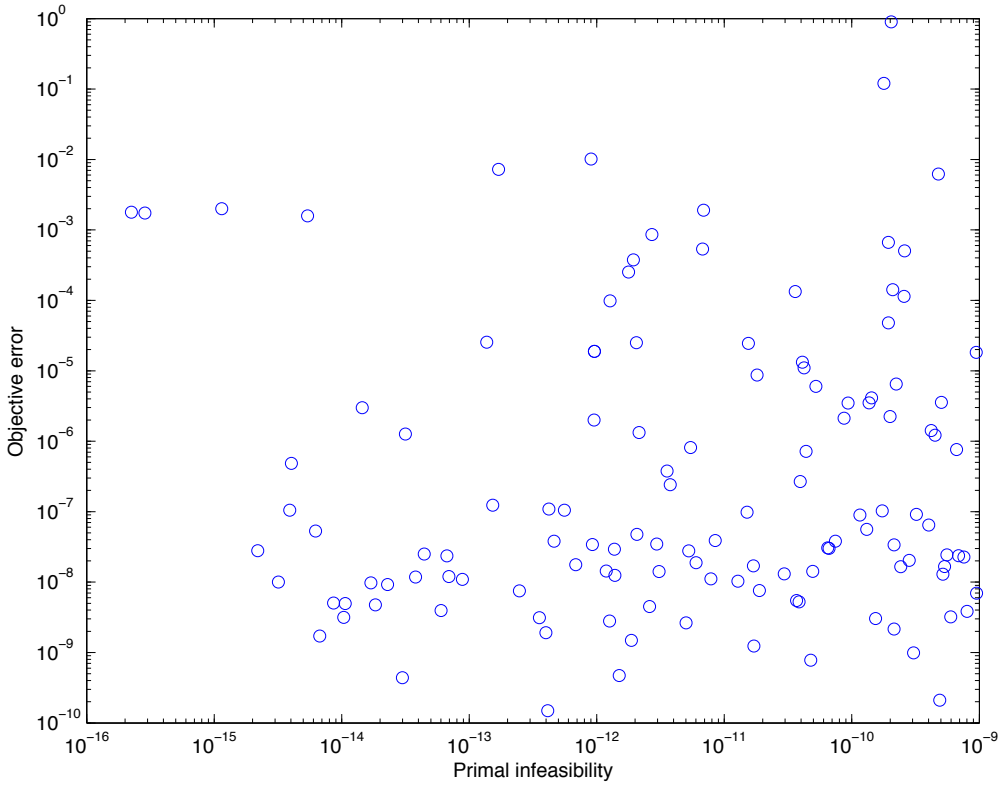

Figure 11.8: Relative objective error and relative primal infeasibility for QPBLUR on problems in the Maros and Mészáros test set

Figure 11.8 is a scatter plot of the relative error in the optimal objective value and the relative primal infeasibility for each problem in the test set. For each problem a point (ϵ_r, ϕ_r) is plotted. The relative primal infeasibility ϵ_r is given by

$$\epsilon_r = \frac{\|Ax - s\|_\infty}{1 + \|(x, s)\|_\infty}.$$

The relative error in the optimal objective value $\phi_Q^\star$ computed with QPBLUR is

$$\phi_r = \frac{\left|\phi_B^\star - \phi_Q^\star\right|}{1 + |\phi_B^\star|},$$

where $\phi_B^\star$ is the reference optimal objective value computed with BPMPD.

11.4 Results for nonlinear programming

In this section we evaluate the performance of QPBLUR when used as a QP solver inside SNOPT's sparse SQP method. SNOPT is used to solve the nonlinear program

$$\begin{aligned} &\text{(locally) } \underset{x}{\text{minimize}} && \varphi(x) \\ &\text{subject to} && \ell \leq \begin{pmatrix} x \\ c(x) \\ Ax \end{pmatrix} \leq u, \end{aligned}$$

where the objective $\varphi(x)$ and the constraints $c(x)$ are smooth nonlinear functions. Chapter 10 discusses SQP methods, SNOPT, and the QP subproblems solved by QPBLUR. Version 7.2-9 (June 2008) of SNOPT was modified to use QPBLUR. The performance of QPBLUR is evaluated in comparison to SNOPT's default QP solver SQOPT.

11.4.1 Results on the COPS 3.0 test set

COPS (a large-scale Constrained Optimization Problem Set) [20, 27] is a collection of 22 nonlinear programs arising from fluid dynamics, population dynamics, optimal design, mesh smoothing, and optimal control. These problems are available as AMPL [4] models.

The AMPL interface to SNOPT was used to solve these problems. General nonlinear programs are solved by SNOPT using QPBLUR as the QP subproblem solver. The problems `bearing` and `torsion` are QPs. The AMPL interface detects when a model is an LP or a QP and, instead of calling SNOPT, it calls SQOPT directly. To compute these results, the AMPL interface was modified to call QPBLUR instead.

Except on problem `triangle` the default options in the AMPL interface to SNOPT were used. For `triangle` the option `Penalty parameter` was set to 10^5 to prevent the objective from becoming unbounded (as described in [39]). Note that the AMPL interface uses 10 Hessian updates by default.

Table 11.5 shows the results of SNOPT-QPBLUR on the COPS 3.0 test set. The heading `Res` is the nonlinear constraint violation, `Majr` is the number of major iterations, `Minr` is the total number of QPBLUR iterations, `Func` is the number of nonlinear function evaluations, `Con` is the number of nonlinear constraint evaluations, `nS` is the number of degrees of freedom at the solution, and `EXIT` and `INFO` are SNOPT's return codes.

SNOPT-QPBLUR failed on the problems `dirichlet`, `polygon` and `rocket`. The failures in `dirichlet` and `polygon` were because the minor iteration limit was exceeded. On `polygon`, this occurs on the first QP subproblem. On `rocket`, QPBLUR detected that the second QP subproblem was infeasible and exited. SQOPT continued, but was unable to minimize the infeasibilities. SNOPT exited with an error of infeasible linear constraints in the QP subproblem.

SNOPT-QPBLUR performs poorly on the the COPS test set. We believe this is because QPBLUR spends too much time trying to solve early QP subproblems accurately.

Table 11.5: Results for SNOPT-QPBLUR on the COPS 3.0 test set

Name	m	n	Objective	Res	Majr	Minr	Func	Con	nS	Time	Exit	Info
bearing	1	5000	-1.55042e-01	0.0e+00	0	9206	0	0	3352	41.77	0	0
camshape	2401	1200	+4.21156e+00	2.0e-07	13	10826	0	26	4	53.80	0	1
catmix	3199	4398	-4.80557e-02	1.2e-08	67	3490	0	71	712	647.36	0	1
chain	2401	3199	+5.06853e+00	4.1e-06	38	149	0	47	799	273.30	0	1
channel	6398	6398	+1.00000e+00	2.3e-05	5	62	0	7	0	3898.68	0	1
dirichlet	42	8981	-1.44349e-01	1.1e+03	6	21108	0	17	8963	369.14	30	31
elec	201	600	+1.84391e+04	5.7e-11	654	1747	763	763	400	33.29	0	1
gasoil	3999	4001	+5.23660e-03	1.6e-08	23	900	26	26	3	626.88	0	1
glider	1601	1999	+1.24797e+03	2.0e-11	98	13103	0	160	359	40.98	0	1
henon	82	10801	+1.25972e+02	1.9e-06	3120	166427	0	18413	10764	6623.79	0	1
lane_emden	82	19241	+9.28490e+00	2.0e-09	80	353	0	104	19237	68.26	0	1
marine	6393	6415	+1.97465e+07	1.1e-11	88	3964	119	119	22	396.91	0	3
methanol	4798	4802	+9.02229e-03	3.1e-09	2558	14346	13488	13488	4	20357.03	0	1
minsurf	1	5000	+2.50695e+00	0.0e+00	585	105947	648	0	4843	1019.87	0	1
pinene	7996	8000	+1.98689e+01	2.4e-08	37	1431	53	53	5	1539.00	0	1
polygon	20099	398	+8.41802e-01	1.4e-01	0	20000	2	2	0	45604.63	30	31
robot	4801	7198	+9.14094e+00	4.3e-09	15	31986	0	24	0	5970.87	0	1
rocket	4801	6401	+1.00611e+00	9.4e-03	2	11104	0	4	1484	7114.50	10	15
steering	3201	3999	+5.54571e-01	6.5e-08	60	2784	0	98	799	4035.22	0	1
tetra	3946	1200	+6.05735e+03	0.0e+00	47	392	49	49	1200	24.21	0	1
torsion	1	5000	-4.18239e-01	0.0e+00	0	3596	0	0	3504	26.14	0	0
triangle	3969	3578	+4.21523e+03	0.0e+00	155	18674	195	195	3578	524.92	0	1

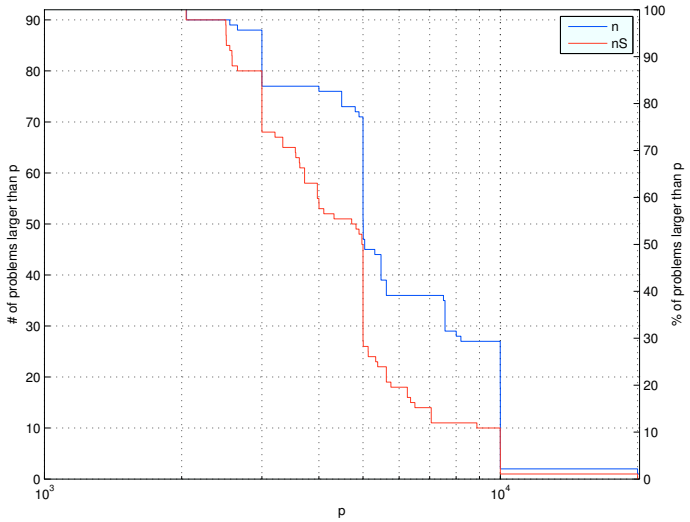

Figure 11.9: Distribution of variables (n) and degrees of freedom (nS) in the nonlinear CUTEr problems

11.4.2 Results on nonlinear CUTEr problems

We revisit the CUTEr test set to examine the performance of QPBLUR within SNOPT on nonlinear programs. CUTEr contains over 1000 nonlinear programs. To select problems likely to have many degrees of freedom at the solution we examined problems for which $n - m > 2000$. SNOPT was run on these problems, and 92 problems were selected that each contained 2000 or more superbasic variables when SNOPT finished. (Problems were chosen based on the number of superbasic variables at the point SNOPT finished, not on whether SNOPT was successful in solving the problem.) Problems with infeasible QP subproblems were eliminated from the test set. This was done because QPBLUR cannot currently handle elastic mode and must switch to SQOPT if it detects an infeasible subproblem.

Table 11.6 shows the size and number of superbasic variables (at exit) for the 92 test problems. Figure 11.9 displays the distribution of the number of variables and the number of degrees of freedom of the problems in the test set. (Note that more than half the problems have more than 5000 variables and degrees of freedom.)

Many problems in this test set have no linear or nonlinear constraints (just bounds on the variables). These problems appear with a 1 in the column labeled m because SNOPT requires at least one constraint to be present. This constraint takes the form $l^T x - s_{\texttt{iObj}} = 0$, where the vector l represents the linear portion of the nonlinear objective. Some problems such as AUG2DCQP and HUES-MOD are convex QPs. These problems were solved directly by QPBLUR in the previous test sets. However, SNOPT has no knowledge of the convexity of these problems and treats them as general nonlinear programs. SNOPT also has no knowledge of the Hessian of these problems, and must construct a quasi-Newton approximation using gradients of the objective evaluated at different iterates.

Table 11.6: Problem size and degrees of freedom for the nonlinear CUTEr test problems

Name	n	m	nS
AUG2DCQP	20200	10001	9994
BROYDN7D	5000	1	5000
CHAINWOO	4000	1	4000
CHENHARK	5000	1	2999
CLPLATEA	5041	1	4970
CLPLATEB	5041	1	4970
CRAGGLVY	5000	1	5000
DIXMAANA	3000	1	3000
DIXMAANB	3000	1	3000
DIXMAANC	3000	1	3000
DIXMAAND	3000	1	3000
DIXMAANE	3000	1	3000
DIXMAANF	3000	1	3000
DIXMAANG	3000	1	3000
DIXMAANH	3000	1	3000
DIXMAANI	3000	1	3000
DIXMAANJ	3000	1	3000
DIXMAANL	3000	1	3000
DIXON3DQ	10000	1	10000
DRCAV1LQ	4489	1	3969
DRCAV2LQ	4489	1	3969
DRCAV3LQ	4489	1	3969
EIGENBLS	2550	1	2550
EIGENCLS	2652	1	2652
FLETCBV3	5000	1	5000
FMINSRF2	5625	1	5625
FMINSURF	5625	1	5625
FREUROTH	5000	1	5000
GENHUMPS	5000	1	5000
GRIDNETB	7564	3845	3720
GRIDNETC	7564	3845	2578
GRIDNETE	7564	3845	3720
GRIDNETF	7564	3845	2581
GRIDNETH	7564	3845	3720
GRIDNETI	7564	3845	2581
HUES-MOD	5000	3	4721
JANNSON3	20000	4	19999
JANNSON4	10000	3	9999
JNLBRNG1	10000	1	6498
JNLBRNG2	10000	1	5754
JNLBRNGA	10000	1	6356
JNLBRNGB	10000	1	5383
LMINSURF	5625	1	5329
LUKVLE12	9997	7498	2500
LUKVLE13	9998	6665	3334
LUKVLE14	9998	6665	3334

Name	n	m	nS
LUKVLI12	9997	7498	9996
LUKVLI13	9998	6665	8885
LUKVLI14	9998	6665	9997
LUKVLI16	9997	7498	5625
LUKVLI17	9997	7498	6252
LUKVLI18	9997	7498	6252
MCCORMCK	5000	1	4999
MINSURFO	5306	1	4825
NOBNDTOR	5476	1	4318
NONCVXU2	5000	1	5000
NONCVXUN	5000	1	5000
NONDQUAR	5000	1	5000
NONMSQRT	4900	1	4900
NONSCOMP	5000	1	5000
OBSTCLAE	10000	1	5133
OBSTCLAL	10000	1	5130
OBSTCLBL	10000	1	7057
OBSTCLBM	10000	1	7056
OBSTCLBU	10000	1	7056
ORTHRDM2	8003	4001	4003
ORTHRDS2	5003	2501	2503
ORTHREGA	8197	4097	4101
ORTHREGC	5005	2501	2505
ORTHREGD	5003	2501	2503
ORTHREGE	7506	5001	2506
ORTHREGF	4805	1601	3205
ORTHRGDM	10003	5001	5003
POWELLSG	5000	1	5000
RAYBENDL	2050	1	2046
RAYBENDS	2050	1	2046
SCHMVETT	5000	1	5000
SCOND1LS	5002	1	5000
SCURLY10	10000	1	10000
SCURLY20	10000	1	10000
SCURLY30	10000	1	10000
SPARSINE	5000	1	5000
SPARSQUR	10000	1	10000
SPMSRTLS	4999	1	4999
SROSENBR	5000	1	5000
TESTQUAD	5000	1	5000
TOINTGSS	5000	1	5000
TORSION1	5476	1	3552
TORSION2	5476	1	3560
TORSIONA	5476	1	3624
TORSIONB	5476	1	3632
TQUARTIC	5000	1	5000

```
Begin  SNOPT-CUTEr NLP
   Major Print level          000001
   Solution                      no
   Print file                     0
   Major iterations            3000
   Iteration limit           100000
   Hessian            Limited Memory
   Hessian updates               10
   Verify level                  -1
END SNOPT-CUTEr NLP
```

Figure 11.10: The specification file containing the options used by SNOPT

Figure 11.10 gives the runtime options used for SNOPT. Note that SNOPT uses only 10 limited memory quasi-Newton updates to approximate the Hessian. Options for QPBLUR were set directly in the SNOPT-QPBLUR interface. In particular, the sparse linear solver UMFPACK was used to factor and solve the KKT systems inside QPBLUR.

Figure 11.11 shows a performance profile of the CPU time for SNOPT with QPBLUR and SQOPT. Regardless of the QP solver used, SNOPT is unable to solve around a quarter of the test problems. This is most likely because SNOPT maintains only 10 Hessian updates.

SNOPT-QPBLUR failed on 23 problems. Twenty-one of these failures were because SNOPT exceeded the major iteration limit. The other two were because SNOPT declared the objective of `FLETCBV3` to be unbounded, and could not achieve the requested accuracy on `ORTHREGA`. Note that SNOPT-SQOPT was able to achieve the requested accuracy on `ORTHREGA`.

SNOPT-SQOPT failed on 26 problems. Twenty-one of these failures were because SNOPT exceeded the major iteration limit. (Twenty of these failures also occurred with SNOPT-QPBLUR.) SNOPT exceeded the minor iteration limit on `FLETCBV3` and `LUKVLI18`, and it could not achieve the requested accuracy on `ORTHRDS2`, `LUKVLI17` and `LUKVLI12`. Note that SNOPT-QPBLUR was able to achieve the requested accuracy on `LUKVLI17` and `LUKVLI12`.

Figure 11.12 shows a portion of the performance profile plot for CPU time. SNOPT-SQOPT is faster than SNOPT-QPBLUR on slightly more problems. However, on the problems it was able to solve, SNOPT-QPBLUR is always within a factor of 10 of the time required by SNOPT-SQOPT. This is not true for SNOPT-SQOPT, which can be more than 10 times slower than SNOPT-QPBLUR on a significant portion of the problems.

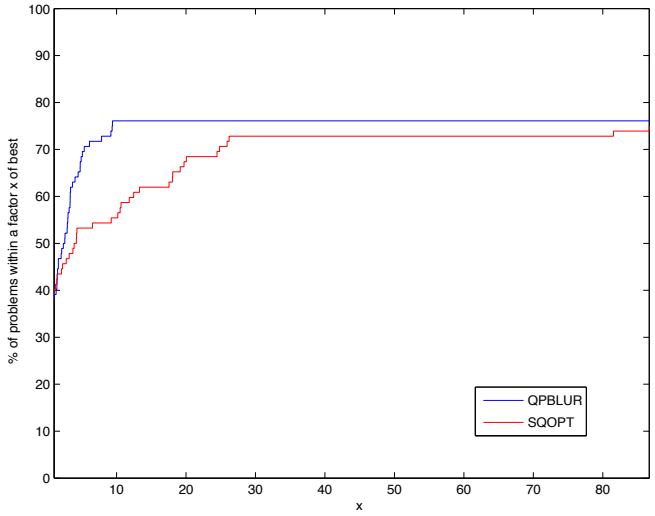

Figure 11.11: Performance profile for CPU time for SNOPT-QPBLUR and SNOPT-SQOPT on nonlinear CUTEr problems with many degrees of freedom

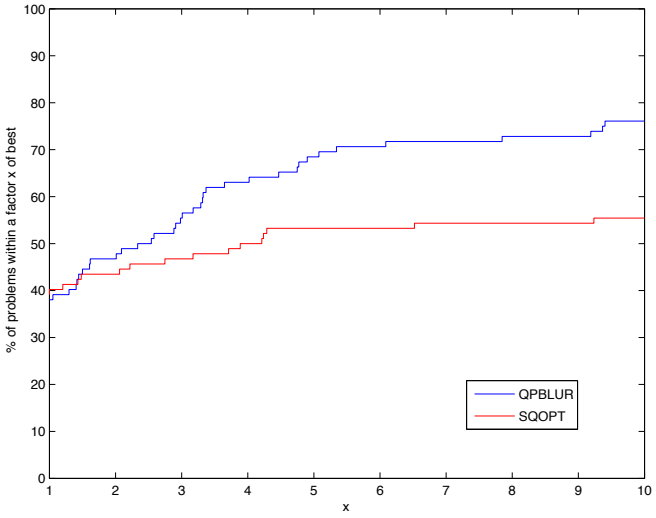

Figure 11.12: A portion of the same performance profile for CPU time on nonlinear CUTEr problems with many degrees of freedom

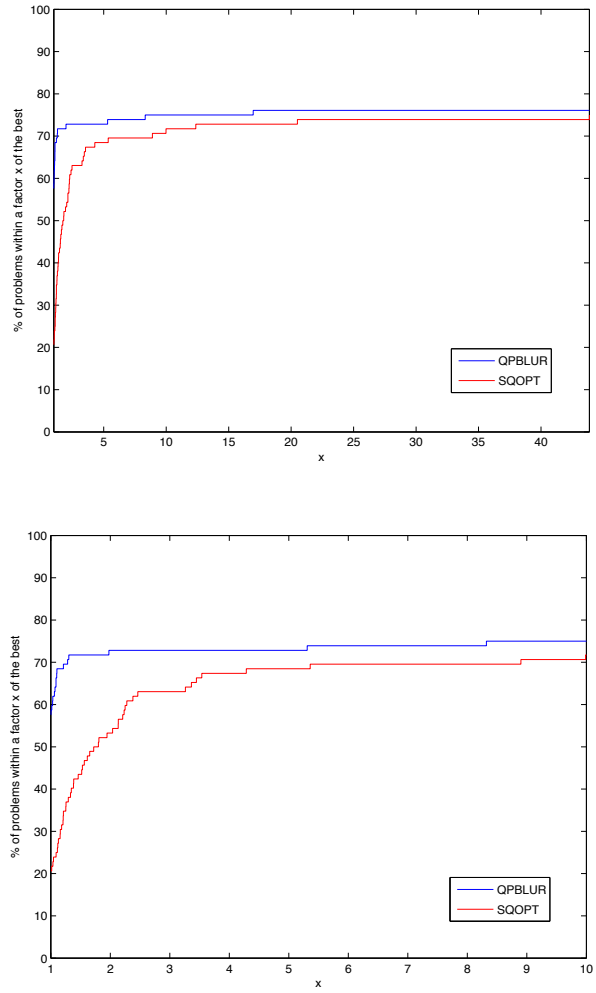

Figure 11.13: Performance profiles of function/constraint evaluations for SNOPT-QPBLUR and SNOPT-SQOPT on nonlinear CUTEr problems. The top figure is the entire profile, the bottom shows factors 1–10.

A definite advantage of using QPBLUR as the QP subproblem solver can be seen by examining the performance profile of the number of function evaluations (Figure 11.13). On nearly 60% of the problems, SNOPT-QPBLUR performs fewer function evaluations than SNOPT-SQOPT. In addition, for every factor x, SNOPT-QPBLUR has a larger fraction of test problems that are within a factor of x of the smallest number of function evaluations, compared to SNOPT-SQOPT. This decrease in evaluations is because SNOPT-QPBLUR performs fewer major iterations than SNOPT-SQOPT on many problems in this test set. We conclude that SNOPT-QPBLUR is a competitive solver on problems with many degrees of freedom for which the nonlinear objective and constraint functions are expensive to compute.

Chapter 12

Contributions and future work

The algorithms developed in this thesis utilize regularization to control the condition of the KKT systems that arise in active-set methods for convex quadratic programming. Regularization improves the linear algebraic properties of the QP algorithms, but it also perturbs the underlying optimization problem. A careful balance is therefore required.

Our contribution is to use the BCL algorithm to solve a sequence of regularized QPs with decreasing regularization parameters, in order to moderate the perturbation applied to the original QP. We use the ability of active-set methods to start efficiently from an approximate solution to solve subproblems with reduced regularization parameters. (Often, only a single iteration is required to compute the solution of an advanced subproblem.)

The BCL algorithm keeps the dual regularization parameter μ (the inverse of the penalty parameter ρ) from becoming arbitrarily close to zero, while ensuring convergence to a point that satisfies the linear equality constraints.

The regularized subproblems in our method are bound-constrained, strictly convex, QPs. Therefore, they may be solved with a simplified active-set algorithm. This algorithm does not require a Phase 1 method to become feasible, or an inertia-controlling method to prevent singular KKT systems. The algorithm uses a simplified step-length procedure without risk of cycling in the presence of degeneracy. Although the subproblems are bound-constrained we choose to compute search-directions using augmented KKT-like systems to maintain sparse and well-conditioned systems. Because these KKT-like systems are always nonsingular, they can be factored with a variety of sparse direct linear solvers, and there is no need for KKT repair, which is expensive and dependent on the linear solver. Block-LU updates of the KKT factors allow for active-set changes and Hessians with low-rank updates.

The tangible result of this work is QPBLUR, a convex QP solver with certain favorable properties. QPBLUR has been tested on a wide variety of QPs and we believe it to be a robust solver. QPBLUR may be warm started from any active set, and is particularly suited as a subproblem solver inside SQP methods such as SNOPT. QPBLUR's fullspace method complements the nullspace method used by SQOPT. It is an effective solver for QPs with many degrees of freedom.

Further work should be done to integrate QPBLUR more completely into SNOPT. In particular, infeasible linearized constraints should be handled without reverting to SQOPT. This can be accomplished by modifying QPBLUR to implement elastic bounds, or modifying SNOPT to minimize the 2-norm, rather than the 1-norm, of the constraint residuals.

It will also be interesting to investigate the performance of SNOPT when SQOPT is used to solve QP subproblems with few degrees of freedom and a switch is made to QPBLUR when the number of degrees of freedom exceeds a threshold. Currently, SQOPT switches from a dense Cholesky factorization to the conjugate-gradient method when the number of degrees of freedom becomes large. Because CG is surprisingly effective on many problems, QPBLUR should not replace it, but simply be a further option. During the QP subproblem solve, the number of CG iterations could be monitored and QPBLUR used if many CG iterations have been performed. Efforts to construct this hybrid SQOPT-QPBLUR approach are currently under way.

In addition, it would be beneficial if SNOPT were aware of the regularization applied inside QPBLUR. The QP subproblems that SNOPT solves could include primal regularization without affecting the theoretical convergence properties of the SQP method. SNOPT could also perturb the original nonlinear program by including primal regularization.

Some improvements to QPBLUR are also desirable. In particular, the stopping criterion based on satisfying the dual optimality conditions of the original problem often seems unnecessarily strict. We have observed improved performance, and fewer subproblem solves, when this criterion is slightly relaxed (without much loss of accuracy in the solution to the original problem). It would be interesting to examine the theoretical properties of a primal-dual regularized solution, in order to develop more refined stopping criteria.

It would also be useful to add interfaces to additional direct parallel solvers for sparse indefinite systems (*e.g.* MUMPS [78]). QPBLUR is currently capable of using the parallel version of PARDISO [87]. However, preliminary numerical results with PARDISO (using separate solves with L and U) indicate that on certain problems the solution of the KKT systems may have a large relative residual. As a result, QPBLUR with PARDISO is unable to solve certain QPs that are solvable with UMFPACK. UMFPACK [104] is currently the preferred linear solver for performance and reliability reasons. A detailed study of the performance of QPBLUR using five different sparse direct solvers (similar to Huynh [59]) is currently under way.

Finally, it would be interesting to develop a gradient-projection algorithm to solve the regularized QP subproblems. This would allow for many changes to be made to the active set at each iteration—a crucial property for large-scale problems. Currently, the regularized QP algorithm allows for only one change to be made to the active set per iteration. As described in Section 7.2, a prototype implementation of a gradient-projection algorithm has been developed. Further work is needed to control oscillations in the active set near a degenerate optimal solution.

Appendix A

SNOPT-QPBLUR on the JANNSON3 problem

In this appendix we present the output of SNOPT on the `JANNSON3` problem using QPBLUR as the QP subproblem solver. The `JANNSON3` problem was originally formulated by Jannson [60, Example 5.3] and is included in CUTEr.

The problem has 20000 variables, 3 nonlinear constraints, one linear constraint, and bounds on the variables. Thus, $(x, s) \in \mathbb{R}^{20004}$. Because there is a single linear equality constraint, a slack variable, say s_1, will have fixed bounds $0 \leq s_1 \leq 0$.

The following table gives the correspondence between the values printed in QPBLUR's output and the mathematical notation discussed in this thesis. Output of SNOPT is described in the SNOPT 7 user guide [40].

Printed output	Notation	Description
`delta0`	δ_0	Initial primal regularization parameter
`deltam`	$\delta_{\min}$	Primal regularization parameter limit
`rho0`	ρ_0	Initial penalty parameter
`rhomax`	$\rho_{\max}$	Penalty parameter limit
`opttol`	ω	Optimality tolerance
`featol`	ϵ	Feasibility tolerance
`update`	`nHup`	Number of rank-2 Hessian updates
`move`	$\lvert\mathcal{M}_0\rvert$	Initial number of moving variables
`nonmove`	$\lvert\mathcal{N}_0\rvert$	Initial number of nonmoving variables
`fixed`	$\lvert\mathcal{E}\rvert$	Number of variables with fixed bounds ($\ell_j = u_j$)
`bcl`	k	BCL iteration count
`rho`	ρ_k	Current penalty parameter
`delta`	δ_k	Current primal regularization parameter
`||r||`	$\lVert Ax - s\rVert_\infty$	Residual for linear equality constraints
`accept`	ϵ_k	Residual tolerance for BCL subproblem
`||y||`	$\lVert y\rVert_\infty$	Size of Lagrange multipliers for linear equality constraints
`iter`	–	Number of BCL subproblem iterations
`nfac`	–	Number LU factorizations performed during BCL subproblem solve
`nS`	n_S	Degrees of freedom $\lvert\mathcal{M}\rvert - m$
`omega`	ω_k	BCL subproblem optimality tolerance
`sigma`	σ	Smallest Lagrange multiplier
`dual`	lhs of (6.14)	Dual residual for original QP

```
 Nonlinear constraints       3     Linear constraints       1
 Nonlinear variables     20000     Linear variables         0
 Jacobian  variables     20000     Objective variables  20000
 Total constraints           4     Total variables      20000


 The user has defined   80000   out of   80000   first  derivatives


QPBLUR: Quadratic Programming with Block-LU updates and Regularization
m     =        4    n      =    20000     nnz(A) =    60004
nnz(H)=    20004    delta0 =  1.0E-06     deltam =  1.0E-14
rho0  =  1.0E+06    rhomax =  1.0E+14     opttol =  1.0E-05
featol=  1.0E-06    maxiter=    10000     update =        0
move  =    20003    nonmove=        0     fixed  =        1
solver= UMFPACK     dxform =        1
Problem   bcl  rho     delta    ||r||    accept     ||y||     iter       nS     nfac   omega   sigma    dual
SQP    0   1  1.0E+06  1.0E-06  2.0E-06  1.0E+00  0.0E+00        1    19999        1  1.0E-03  0.0E+00  4.0E+00
SQP    0   2  1.0E+06  1.0E-08  4.4E-14  1.0E-03  2.0E+00        1    19999        1  1.0E-05  0.0E+00  3.3E-07
SQP    0   3  1.0E+06  1.0E-10  1.4E-17  1.0E-06  2.0E+00        1    19999        1  1.0E-07  0.0E+00  3.3E-09

 Major Minors     Step   nCon Feasible  Optimal  MeritFunction    nS Penalty
     0      3              1  1.0E+00  6.7E-01  2.0001000E+04 19999          r

QPBLUR: Quadratic Programming with Block-LU updates and Regularization
m     =        4    n      =    20000     nnz(A) =    60004
nnz(H)=    20004    delta0 =  1.0E-06     deltam =  1.0E-14
rho0  =  1.0E+06    rhomax =  1.0E+14     opttol =  5.0E-04
featol=  1.0E-06    maxiter=    10000     update =        0
move  =    20003    nonmove=        0     fixed  =        1
solver= UMFPACK     dxform =        1
Problem   bcl  rho     delta    ||r||    accept     ||y||     iter       nS     nfac   omega   sigma    dual
SQP    1   1  1.0E+06  1.0E-06  1.0E-06  1.0E+00  2.2E+00        5    19996        1  1.0E-03  0.0E+00  1.9E-02
SQP    1   2  1.0E+06  1.0E-08  3.7E-12  1.0E-03  2.0E+00        1    19996        1  1.0E-05  0.0E+00  2.5E-06

     1      6  1.0E+00      2  2.5E-01  1.3E+00  1.9999500E+04 19996        n r

QPBLUR: Quadratic Programming with Block-LU updates and Regularization
m     =        4    n      =    20000     nnz(A) =    60004
nnz(H)=    20004    delta0 =  1.0E-06     deltam =  1.0E-14
rho0  =  1.0E+06    rhomax =  1.0E+14     opttol =  2.5E-04
featol=  1.0E-06    maxiter=    10000     update =        1
move  =    20003    nonmove=        0     fixed  =        1
solver= UMFPACK     dxform =        1
Problem   bcl  rho     delta    ||r||    accept     ||y||     iter       nS     nfac   omega   sigma    dual
SQP    2   1  1.0E+06  1.0E-06  1.0E-06  1.0E+00  2.2E+00        1    19999        1  1.0E-03  0.0E+00  2.2E-01
SQP    2   2  1.0E+06  1.0E-08  5.7E-13  1.0E-03  2.0E+00        1    19999        1  1.0E-05  0.0E+00  4.9E-08

     2      2  2.8E-01      4 (3.5E-08) 2.2E-01  1.9998596E+04 19999        s

QPBLUR: Quadratic Programming with Block-LU updates and Regularization
m     =        4    n      =    20000     nnz(A) =    60004
nnz(H)=    20004    delta0 =  1.0E-06     deltam =  1.0E-14
rho0  =  1.0E+06    rhomax =  1.0E+14     opttol =  1.3E-04
featol=  1.0E-06    maxiter=    10000     update =        2
move  =    20003    nonmove=        0     fixed  =        1
solver= UMFPACK     dxform =        1
Problem   bcl  rho     delta    ||r||    accept     ||y||     iter       nS     nfac   omega   sigma    dual
SQP    3   1  1.0E+06  1.0E-06  1.0E-06  1.0E+00  2.2E+00        1    19999        1  1.0E-03  0.0E+00  1.1E-01
```

```
SQP    3   2  1.0E+06  1.0E-08  4.9E-15  1.0E-03  2.0E+00        1    19999        1  1.0E-05  0.0E+00  1.3E-08

     3      2  1.0E+00      5 (1.5E-11) 1.1E-01  1.9998535E+04 19999

QPBLUR: Quadratic Programming with Block-LU updates and Regularization
m     =        4    n      =    20000     nnz(A) =    60004
nnz(H)=    20004    delta0 =  1.0E-06     deltam =  1.0E-14
rho0  =  1.0E+06    rhomax =  1.0E+14     opttol =  6.3E-05
featol=  1.0E-06    maxiter=    10000     update =        3
move  =    20003    nonmove=        0     fixed  =        1
solver= UMFPACK     dxform =        1
Problem   bcl  rho     delta    ||r||    accept     ||y||     iter       nS     nfac   omega   sigma    dual
SQP    4   1  1.0E+06  1.0E-06  1.0E-06  1.0E+00  2.2E+00        1    19999        1  1.0E-03  0.0E+00  4.1E-02
SQP    4   2  1.0E+06  1.0E-08  1.0E-14  1.0E-03  2.0E+00        1    19999        1  1.0E-05  0.0E+00  5.7E-09

     4      2  1.0E+00      6 (4.9E-13) 4.1E-02  1.9998521E+04 19999

QPBLUR: Quadratic Programming with Block-LU updates and Regularization
m     =        4    n      =    20000     nnz(A) =    60004
nnz(H)=    20004    delta0 =  1.0E-06     deltam =  1.0E-14
rho0  =  1.0E+06    rhomax =  1.0E+14     opttol =  3.1E-05
featol=  1.0E-06    maxiter=    10000     update =        4
move  =    20003    nonmove=        0     fixed  =        1
solver= UMFPACK     dxform =        1
Problem   bcl  rho     delta    ||r||    accept     ||y||     iter       nS     nfac   omega   sigma    dual
SQP    5   1  1.0E+06  1.0E-06  1.0E-06  1.0E+00  2.2E+00        1    19999        1  1.0E-03  0.0E+00  2.3E-02
SQP    5   2  1.0E+06  1.0E-08  1.4E-14  1.0E-03  2.0E+00        1    19999        1  1.0E-05  0.0E+00  3.1E-09

     5      2  1.0E+00      7 (2.6E-12) 2.3E-02  1.9998519E+04 19999

QPBLUR: Quadratic Programming with Block-LU updates and Regularization
m     =        4    n      =    20000     nnz(A) =    60004
nnz(H)=    20004    delta0 =  1.0E-06     deltam =  1.0E-14
rho0  =  1.0E+06    rhomax =  1.0E+14     opttol =  1.6E-05
featol=  1.0E-06    maxiter=    10000     update =        5
move  =    20003    nonmove=        0     fixed  =        1
solver= UMFPACK     dxform =        1
Problem   bcl  rho     delta    ||r||    accept     ||y||     iter       nS     nfac   omega   sigma    dual
SQP    6   1  1.0E+06  1.0E-06  1.0E-06  1.0E+00  2.2E+00        1    19999        1  1.0E-03  0.0E+00  1.5E-03
SQP    6   2  1.0E+06  1.0E-08  2.6E-16  1.0E-03  2.0E+00        1    19999        1  1.0E-05  0.0E+00  1.7E-10

     6      2  1.0E+00      8 (1.9E-12) 1.5E-03  1.9998518E+04 19999

QPBLUR: Quadratic Programming with Block-LU updates and Regularization
m     =        4    n      =    20000     nnz(A) =    60004
nnz(H)=    20004    delta0 =  1.0E-06     deltam =  1.0E-14
rho0  =  1.0E+06    rhomax =  1.0E+14     opttol =  7.8E-06
featol=  1.0E-06    maxiter=    10000     update =        6
move  =    20003    nonmove=        0     fixed  =        1
solver= UMFPACK     dxform =        1
Problem   bcl  rho     delta    ||r||    accept     ||y||     iter       nS     nfac   omega   sigma    dual
SQP    7   1  1.0E+06  1.0E-06  1.0E-06  1.0E+00  2.2E+00        1    19999        1  1.0E-03  0.0E+00  1.3E-04
SQP    7   2  1.0E+06  1.0E-08  1.2E-17  1.0E-03  2.0E+00        1    19999        1  1.0E-05  0.0E+00  1.8E-11
SQP    7   3  1.0E+06  1.0E-10  1.2E-19  1.0E-06  2.0E+00        1    19999        1  1.0E-07  0.0E+00  1.2E-14

     7      3  1.0E+00      9 (1.2E-12) 1.2E-04  1.9998518E+04 19999

QPBLUR: Quadratic Programming with Block-LU updates and Regularization
m     =        4    n      =    20000     nnz(A) =    60004
nnz(H)=    20004    delta0 =  1.0E-06     deltam =  1.0E-14
rho0  =  1.0E+06    rhomax =  1.0E+14     opttol =  3.9E-06
```

```
featol=  1.0E-06    maxiter=    10000     update =        7
move  =    20003    nonmove=        0     fixed  =        1
solver= UMFPACK     dxform =        1
Problem   bcl  rho     delta    ||r||    accept     ||y||     iter       nS     nfac   omega   sigma    dual
SQP    8   1  1.0E+06  1.0E-06  1.0E-06  1.0E+00  2.2E+00        1    19999        1  1.0E-03  0.0E+00  9.6E-07
SQP    8   2  1.0E+06  1.0E-08  1.0E-18  1.0E-03  2.0E+00        1    19999        1  1.0E-05  0.0E+00  1.6E-13
SQP    8   3  1.0E+06  1.0E-10  8.5E-21  1.0E-06  2.0E+00        1    19999        1  1.0E-07  0.0E+00  8.9E-14

     8      3  1.0E+00     10 (1.1E-12)(8.9E-07) 1.9998518E+04 19999

 SNOPTA EXIT   0 -- finished successfully
 SNOPTA INFO   1 -- optimality conditions satisfied

 Problem name                JANNSON3
 No. of iterations                 25   Objective value      1.9998518041E+04
 No. of major iterations            8   Linear objective     9.9997787828E-19
 Penalty parameter           0.000E+00   Nonlinear objective  1.9998518041E+04
 No. of calls to funobj            10   No. of calls to funcon            10
 No. of superbasics             19999   No. of basic nonlinears            0
 No. of degenerate steps            0   Percentage                      0.00
 Max x                      1 4.7E-01   Max pi                   1 2.0E+00
 Max Primal infeas          0 0.0E+00   Max Dual infeas          2 2.9E-06
 Nonlinear constraint violn    1.1E-12

 Solution not printed

 Time for MPS input                          0.00 seconds
 Time for solving problem                    9.81 seconds
 Time for solution output                    0.00 seconds
 Time for constraint functions               0.05 seconds
 Time for objective function                 0.00 seconds
```

Appendix B

Classification of CUTEr QPs

This appendix contains a classification of the 172 QPs currently in CUTEr. The columns labeled m and n denote the number of rows and columns in the matrix A. The columns labeled $\mathrm{nnz}(A)$ and $\mathrm{nnz}(H)$ denote the number of nonzero in A and the number of nonzeros in H. A problem is separable if the Hessian is diagonal. Separable problems were classified as convex if the diagonal was nonnegative. Nonseparable problems were classified as convex if it was possible to compute the Cholesky factorization of $H + 10^{-13}I$.

#	Name	m	n	$\mathrm{nnz}(A)$	$\mathrm{nnz}(H)$	convex	separable
1	AOENDNDL	15002	45006	79995	90012	N	N
2	AOENINDL	15002	45006	79995	90012	N	N
3	AOENSNDL	15002	45006	78025	90012	N	N
4	AOESDNDL	15002	45006	79996	90012	N	N
5	AOESINDL	15002	45006	79996	90012	N	N
6	AOESSNDL	15002	45006	78026	90012	N	N
7	AONNDNDL	20004	60012	115003	120024	N	N
8	AONNDNIL	20004	60012	115003	120024	N	N
9	AONNDNSL	20004	60012	113034	120024	N	N
10	AONNSNSL	20004	60012	111099	120024	N	N
11	AONSDSDL	20004	60012	115005	120024	N	N
12	AONSDSDS	2004	6012	11104	12024	N	N
13	AONSDSIL	20004	60012	115005	120024	N	N
14	AONSDSSL	20004	60012	113035	120024	N	N
15	AONSSSSL	20004	60012	111101	120024	N	N
16	A2ENDNDL	15002	45006	79995	90012	N	N
17	A2ENINDL	15002	45006	79995	90012	N	N
18	A2ENSNDL	15002	45006	78025	90012	N	N
19	A2ESDNDL	15002	45006	79996	90012	N	N
20	A2ESINDL	15002	45006	79996	90012	N	N
21	A2ESSNDL	15002	45006	78026	90012	N	N
22	A2NNDNDL	20004	60012	115003	120024	N	N
23	A2NNDNIL	20004	60012	115003	120024	N	N
24	A2NNDNSL	20004	60012	113034	120024	N	N
25	A2NNSNSL	20004	60012	111099	120024	N	N
26	A2NSDSDL	20004	60012	115005	120024	N	N
27	A2NSDSIL	20004	60012	115012	120024	N	N
28	A2NSDSSL	20004	60012	113035	120024	N	N
29	A2NSSSSL	20004	60012	111101	120024	N	N
30	A5ENDNDL	15002	45006	79995	90012	N	N
31	A5ENINDL	15002	45006	79995	90012	N	N
32	A5ENSNDL	15002	45006	78025	90012	N	N
33	A5ESDNDL	15002	45006	79996	90012	N	N
34	A5ESINDL	15002	45006	79996	90012	N	N
35	A5ESSNDL	15002	45006	78026	90012	N	N
36	A5NNDNDL	20004	60012	115003	120024	N	N
37	A5NNDNIL	20004	60012	115003	120024	N	N

#	Name	m	n	$\mathrm{nnz}(A)$	$\mathrm{nnz}(H)$	convex	separable
38	A5NNDNSL	20004	60012	113034	120024	N	N
39	A5NNSNSL	20004	60012	111099	120024	N	N
40	A5NSDSDL	20004	60012	115005	120024	N	N
41	A5NSDSDM	2004	6012	11104	12024	N	N
42	A5NSDSIL	20004	60012	115012	120024	N	N
43	A5NSDSSL	20004	60012	113035	120024	N	N
44	A5NSSNSM	2004	6012	11104	12024	N	N
45	A5NSSSSL	20004	60012	111101	120024	N	N
46	ALLINQP	25000	50000	50000	149998	Y	N
47	AUG2D	10000	20200	40000	19800	Y	Y
48	AUG2DC	10000	20200	40000	20200	Y	Y
49	AUG2DCQP	10000	20200	40000	20200	Y	Y
50	AUG2DQP	10000	20200	40000	19800	Y	Y
51	AUG3D	8000	27543	50286	22743	Y	Y
52	AUG3DC	8000	27543	50286	27543	Y	Y
53	AUG3DCQP	8000	27543	50286	27543	Y	Y
54	AUG3DQP	8000	27543	50286	22743	Y	Y
55	AVGASA	10	8	30	22	Y	N
56	AVGASB	10	8	30	22	Y	N
57	BIGGSC4	7	4	16	4	N	N
58	BLOCKQP1	5001	10010	70010	10010	N	N
59	BLOCKQP2	5001	10010	70010	10010	N	N
60	BLOCKQP3	5001	10010	70010	10010	N	N
61	BLOCKQP4	5001	10010	70010	10010	N	N
62	BLOCKQP5	5001	10010	70010	10010	N	N
63	BLOWEYA	2002	4002	10003	10003	N	N
64	BLOWEYB	2002	4002	10003	10003	N	N
65	BLOWEYC	2002	4002	10003	10003	N	N
66	CONT5-QP	40200	40601	240200	401	Y	Y
67	CVXQP1	5000	10000	14998	69968	Y	N
68	CVXQP2	2500	10000	7499	69968	Y	N
69	CVXQP3	7500	10000	22497	69968	Y	N
70	DEGENQP	125025	50	247548	50	Y	Y
71	DTOC3	2998	4499	10493	4497	Y	Y
72	DUAL1	1	85	85	7031	Y	N
73	DUAL2	1	96	96	8920	Y	N
74	DUAL3	1	111	111	12105	Y	N
75	DUAL4	1	75	75	5523	Y	N
76	DUALC1	215	9	1935	81	Y	N
77	DUALC2	229	7	1603	49	Y	N
78	DUALC5	278	8	2224	64	Y	N
79	DUALC8	503	8	4024	64	Y	N
80	FERRISDC	210	2200	6200	440000	N	N
81	GENHS28	8	10	24	28	Y	N
82	GMNCASE1	300	175	23940	23430	N	N
83	GMNCASE2	1050	175	28546	23431	Y	N
84	GMNCASE3	1050	175	28546	23431	Y	N
85	GMNCASE4	350	175	27510	30485	Y	N
86	GOULDQP2	9999	19999	29997	29995	Y	N
87	GOULDQP3	9999	19999	29997	59993	Y	N
88	HATFLDH	7	4	16	4	N	N
89	HS118	17	15	39	15	Y	Y
90	HS21	1	2	2	2	Y	Y
91	HS268	5	5	25	25	Y	N
92	HS35	1	3	3	7	Y	N
93	HS35I	1	3	3	7	Y	N
94	HS35MOD	1	3	3	7	Y	N
95	HS44	6	4	12	8	N	N
96	HS44NEW	6	4	12	8	N	N
97	HS51	3	5	7	9	Y	N
98	HS52	3	5	7	9	Y	N
99	HS53	3	5	7	9	Y	N
100	HS76	3	4	10	8	Y	N
101	HS76I	3	4	10	8	Y	N
102	HUES-MOD	2	5000	10000	5000	Y	Y
103	HUESTIS	2	5000	10000	5000	Y	Y
104	KSIP	1001	20	20001	20	Y	Y
105	LINCONT	419	1257	24045	2514	N	N
106	LISWET1	2000	2002	6000	2002	Y	Y
107	LISWET10	2000	2002	6000	2002	Y	Y
108	LISWET11	2000	2002	6000	2002	Y	Y
109	LISWET12	2000	2002	6000	2002	Y	Y
110	LISWET2	2000	2002	6000	2002	Y	Y
111	LISWET3	2000	2002	6000	2002	Y	Y

#	Name	m	n	$\mathrm{nnz}(A)$	$\mathrm{nnz}(H)$	convex	separable
112	LISWET4	2000	2002	6000	2002	Y	Y
113	LISWET5	2000	2002	6000	2002	Y	Y
114	LISWET6	2000	2002	6000	2002	Y	Y
115	LISWET7	2000	2002	6000	2002	Y	Y
116	LISWET8	2000	2002	6000	2002	Y	Y
117	LISWET9	2000	2002	6000	2002	Y	Y
118	LOTSCHD	7	12	54	6	Y	Y
119	MOSARQP1	700	2500	3422	2590	Y	N
120	MOSARQP2	700	2500	3422	2590	Y	N
121	NASH	24	72	157	144	N	N
122	NCVXQP1	5000	10000	14998	69968	N	N
123	NCVXQP2	5000	10000	14998	69966	N	N
124	NCVXQP3	5000	10000	14998	69968	N	N
125	NCVXQP4	2500	10000	7499	69968	N	N
126	NCVXQP5	2500	10000	7499	69966	N	N
127	NCVXQP6	2500	10000	7499	69968	N	N
128	NCVXQP7	7500	10000	22497	69968	N	N
129	NCVXQP8	7500	10000	22497	69966	N	N
130	NCVXQP9	7500	10000	22497	69968	N	N
131	PORTSNQP	2	100000	199999	100000	N	Y
132	PORTSQP	1	100000	100000	100000	Y	Y
133	POWELL20	5000	5000	10000	5000	Y	Y
134	PRIMAL1	85	325	5815	324	Y	Y
135	PRIMAL2	96	649	8042	648	Y	Y
136	PRIMAL3	111	745	21547	744	Y	Y
137	PRIMAL4	75	1489	16031	1488	Y	Y
138	PRIMALC1	9	230	2070	229	Y	Y
139	PRIMALC2	7	231	1617	230	Y	Y
140	PRIMALC5	8	287	2296	286	Y	Y
141	PRIMALC8	8	520	4160	519	Y	Y
142	QPBAND	25000	50000	50000	149998	Y	N
143	QPCBLEND	74	83	491	83	Y	Y
144	QPCBOEI1	351	384	3485	384	Y	Y
145	QPCBOEI2	166	143	1196	143	Y	Y
146	QPCSTAIR	356	467	3856	467	Y	Y
147	QPNBAND	25000	50000	50000	149998	N	N
148	QPNBLEND	74	83	491	83	N	Y
149	QPNBOEI1	351	384	3485	384	N	Y
150	QPNBOEI2	166	143	1196	143	N	Y
151	QPNSTAIR	356	467	3856	467	N	Y
152	RDW2D51F	65025	132098	1170450	1182722	Y	N
153	RDW2D51U	65025	132098	1170450	1182722	Y	N
154	RDW2D52B	65025	132098	1170450	1182722	Y	N
155	RDW2D52F	49	162	882	1250	Y	N
156	RDW2D52U	65025	132098	1170450	1182722	Y	N
157	S268	5	5	25	25	Y	N
158	SOSQP1	2501	5000	10000	5000	N	N
159	SOSQP2	2501	5000	10000	5000	N	N
160	STATIC3	96	434	496	1738	N	N
161	STCQP1	4095	8193	28665	106473	Y	N
162	STCQP2	4095	8193	28665	106473	Y	N
163	STNQP1	4095	8193	28665	105843	N	N
164	STNQP2	4095	8193	28665	105843	N	N
165	TAME	1	2	2	4	Y	N
166	UBH1	6000	9009	24000	3003	Y	Y
167	WALL10	0	9009	0	25065	Y	N
168	WALL100	0	9009	0	2743326	Y	N
169	WALL20	0	9009	0	105506	Y	N
170	WALL50	0	9009	0	679175	Y	N
171	YAO	2000	2002	6000	2002	Y	Y
172	ZECEVIC2	2	2	4	1	Y	Y

Bibliography

[1] Åke Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996. 55, 57, 59

[2] Alexander C. Aitken. On least squares and linear combinations of observations. *Proceedings of the Royal Society of Edinburgh, Sec. A*, 55:42–47, 1934. 58

[3] Anna Altman and Jacek Gondzio. Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization. *Optimization Methods and Software*, 11(1–4):275–302, 1999. 32

[4] AMPL modeling system. `http://www.ampl.com`. 60, 92

[5] Michele Benzi, Gene H. Golub, and Jörg Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005. 8

[6] Dimitri P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, Boston, 1996. 38, 41, 44

[7] Johannes Bisschop and Alexander Meeraus. Matrix augmentation and partitioning in updating of the basis inverse. *Mathematical Programming*, 13(1):241–254, 1977. 15

[8] Robert E. Bixby. Solving real-world linear programs: a decade and more of progress. *Operations Research*, 50(1):3–15, 2002. 14

[9] Karl-Heinz Borgwardt. *The Simplex Method, A Probabilistic Analysis*. Springer-Verlag, Berlin, 1987. 4

[10] BPMPD interior point solver. `http://www.sztaki.hu/~meszaros/bpmpd/`. 86

[11] Ken W. Brodlie, A. R. Gourlay, and John Greenstadt. Rank-one and rank-two corrections to positive definite matrices expressed in product form. *Journal of the Institute of Mathematics and its Applications*, 11:73–82, 1973. 17

[12] James R. Bunch and Linda Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31(137):163–179, 1977. 10

[13] James R. Bunch, Linda Kaufman, and Beresford N. Parlett. Decomposition of a symmetric matrix. *Numerische Mathematik*, 27(1):95–110, 1976. 10

[14] James V. Burke and Jorge J. Moré. Exposing constraints. *SIAM Journal on Optimization*, 4:573–595, 1994. 53

[15] Alan K. Cline. Two subroutine packages for the efficient updating of matrix factorizations. TR 68, Department of Computer Science and the Center for Numerical Analysis, University of Texas at Austin, Mar 1977. 21

[16] Andrew R. Conn, Nicholas I. M. Gould, Annick Sartenaer, and Philippe L. Toint. A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991. 38, 41, 44

[17] Andrew R. Conn, Nicholas I. M. Gould, Annick Sartenaer, and Philippe L. Toint. Convergence properties of an augmented Lagrangian algorithm for optimization with a combination of general equality and linear constraints. *SIAM Journal of Optimization*, 6:674–703, 1996. 45

[18] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *LANCELOT: A Fortran package for Large-scale Nonlinear Optimization (Release A)*, volume 17 of *Springer Series in Computational Mathematics*. Springer Verlag, Heidelberg and New York, 1992. 44, 60

[19] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. Testing a class of algorithms for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50:399–430, 1998. 53

[20] COPS: constrained optimization problem set. `http://www.mcs.anl.gov/~more/cops/`. 60, 92

[21] Richard Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society*, 49:1–23, 1943. 34

[22] CUTEr: constrained and unconstrained testing environment revisited. `http://cuter.rl.ac.uk/cuter-www/`. 78

[23] Timothy A. Davis. `http://www.cise.ufl.edu/research/sparse/matrices`. Submitted to ACM Transactions on Mathematical Software. 75

[24] Timothy A. Davis. *Direct Methods for Sparse Linear Systems*. Fundamentals of Algorithms. SIAM, Philadelphia, 2006. 20, 56, 61

[25] Timothy A. Davis. Multifrontal multithreaded rank-revealing sparse QR factorization. Submitted to *ACM Transactions on Mathematical Software*, 2010. 56

[26] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2), 2002. 82

[27] Elizabeth D. Dolan, Jorge J. Moré, and Todd S. Munson. Benchmarking optimization software with COPS 3.0. Tech. Report ANL/MCS-TM-273, Mathematics and Computer Science Division, Argonne National Laboratory, Feb 2004. 60, 92

[28] Iain S. Duff. MA57: a Fortran code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30(2):118–144, 2004. 45, 63

[29] Samuel K. Eldersveld and Michael A. Saunders. A block-LU update for large-scale linear programming. *SIAM Journal of Matrix Analysis and Applications*, 13:191–201, 1992. 15

[30] Michael C. Ferris. Finite termination of the proximal point algorithm. *Mathematical Programming*, 50:359–366, 1991. 26

[31] Anthony V. Fiacco and Garth P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley and Sons, New York, 1968. 36

[32] Michael P. Friedlander. *A Globally Convergent Linearly Constrained Lagrangian Method for Nonlinear Optimization*. PhD thesis, MS&E, Stanford University, 2002. 41

[33] Michael P. Friedlander. Exact regularization of linear programs. Tech. Report TR-2005-31, University of British Columbia, Dept. of Computer Science, Dec 2005. 25

[34] Michael P. Friedlander and Michael A. Saunders. A globally convergent linearly constrained Lagrangian method for nonlinear optimization. *SIAM Journal of Optimization*, 15(3):863–897, 2005. 38

[35] Michael P. Friedlander and Paul Tseng. Exact regularization of convex programs. *SIAM Journal on Optimization*, 18:1326–1350, 2007. 23, 24

[36] David M. Gay. Hooking your solver to AMPL. Tech. Report 97-4-06, Computing Sciences Research Center, Bell Laboratories, Murray Hill, NJ, April 1997. 65

[37] Philip E. Gill, Walter Murray, Dulce B. Ponceleón, and Michael A. Saunders. Solving reduced KKT systems in barrier methods for linear and quadratic programming. Tech. Report SOL 91-7, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University, 1991. 31

[38] Philip E. Gill, Walter Murray, Dulce B. Ponceleón, and Michael A. Saunders. Solving reduced KKT systems in barrier methods for linear programming. In G. A. Watson and D. Griffiths, editors, *Numerical Analysis 1993 (Dundee, 1993)*, volume 303 of *Pitman Res. Notes Math. Ser.*, pages 89–104. Longman Sci. Tech., Harlow, UK, 1994. 31

[39] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005. SIGEST article. 17, 60, 68, 92

[40] Philip E. Gill, Walter Murray, and Michael A. Saunders. User's guide for SNOPT 7: Software for large-scale nonlinear programming. `http://www.scicomp.ucsd.edu/~peg` (see Software), 2006. 68, 101

[41] Philip E. Gill, Walter Murray, and Michael A. Saunders. User's guide for SQOPT 7: Software for large-scale linear and quadratic programming. `http://www.scicomp.ucsd.edu/~peg` (see Software), 2006. 10, 70

[42] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. Sparse matrix methods in optimization. *SIAM Journal of Scientific and Statistical Computing*, 5:562–589, 1984. 15

[43] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45(1-3):437–474, 1989. 14, 64

[44] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. Inertia-controlling methods for general quadratic programming. *SIAM Review*, 33(1):1–36, 1991. 13

[45] Philip E. Gill, Michael A. Saunders, and Joseph R. Shinnerl. On the stability of Cholesky factorization for symmetric quasidefinite systems. *SIAM Journal on Matrix Analysis and Applications*, 17(1):35–46, 1996. 30

[46] Philip E. Gill and Elizabeth Wong. Methods for convex and general quadratic programming. Numerical Analysis Report 10-1, Department of Mathematics, University of California, San Diego, La Jolla, CA, 2010. 14, 69

[47] Philip E. Gill and Elizabeth Wong. Sequential quadratic programming methods. In Jon Lee and Sven Leyffer, editors, *Mixed-Integer Nonlinear Optimization: Algorithmic Advances and Applications*, The IMA Volumes in Mathematics and its Applications. Springer Verlag, Berlin, Heidelberg and New York, 2010. To appear. 14, 17, 69

[48] Gene H. Golub. Numerical methods for solving linear least squares problems. *Numerische Mathematik*, 7(3):206–216, 1965. 56

[49] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. The Johns Hopkins University Press, Baltimore, third edition, 1996. 56

[50] Nicholas I. M. Gould. On the accurate determination of search directions for simple differentiable penalty functions. *IMA Journal of Numerical Analysis*, 6:357–372, 1986. 36

[51] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. CUTEr and SifDec: A constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394, 2003. 60, 78

[52] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. GALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Transactions on Mathematical Software*, 29(4):353–372, 2003. 11

[53] Nicholas I. M. Gould and John K. Reid. New crash procedures for large-scale systems of linear constraints. *Mathematical Programming*, 45:475–501, 1989. 78

[54] Nicholas I. M. Gould and Philippe L. Toint. A quadratic programming page. `http://www.numerical.rl.ac.uk/qp/qp.html`. 2

[55] Nicholas I. M. Gould and Philippe L. Toint. A quadratic programming bibliography. Internal Report 2001, Rutherford Appleton Laboratory, 2010. 2

[56] Julian A. J. Hall and Ken I. M. McKinnon. The simplest examples where the simplex method cycles and conditions where EXPAND fails to prevent cycling. *Mathematical Programming*, 100(1):133–150, 2004. 14

[57] Magnus R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4:303–320. 38

[58] Nicholas J. Higham. Fortran codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Transactions on Mathematical Software*, 14:381–396, 1988. 62

[59] Hanh M. Huynh. *A Large-scale Quadratic Programming Solver Based On Block-LU Updates of the KKT System*. PhD thesis, SCCM, Stanford University, 2008. 4, 11, 15, 61, 75, 100

[60] Christian Jansson. Convex-concave extensions. *BIT*, 40(2):291–313, 2000. 101

[61] Linda Kaufman. Solving the quadratic programming problem arising in support vector classification. In B. Schlkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*. MIT Press, 1998. 2

[62] Andrew Kennings and Anthony Vannelli. VLSI placement using quadratic programming and network partitioning techniques. *International Transactions in Operational Research*, 4(5–6):353–364, 1997. 2

[63] Victor L. Klee and George J. Minty. How good is the simplex method? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, New York, 1972. 4

[64] M. K. Kozlov, S. P. Tarasov, and L. G. Khachiyan. The polynomial solvability of convex quadratic programming. *USSR Computational Mathematics and Mathematical Physics*, 20(5):223–228, 1980. Translated by J. Berry. 1, 4

[65] LANCELOT optimization software. `http://www.numerical.rl.ac.uk/lancelot/blurb.html`. 44

[66] Bosco H. Leung. Design methodology of decimation filters for oversampled ADC based on quadratic programming. *IEEE Transactions on Circuits and Systems*, 38(10):1121–1132, 1991. 2

[67] X. Li and Z. Xuan. An interior-point QP algorithm for structural optimization. *Structural Optimization*, 15(3–4):172–179, 1998. 2

[68] X. Liu, Y. Sun, and W. Wang. Stabilizing control of robustness for systems with maximum uncertain parameters—a quadratic programming approach. *Control Theory and Applications*, 16(5):729–732, 1999. 2

[69] LUMOD: Updating a dense square factorization $LC = U$. `http://www.stanford.edu/group/SOL/software/lumod.html`. 21

[70] LUSOL sparse matrix package. `http://www.stanford.edu/group/SOL/software.html`. 63

[71] Olvi L. Mangasarian and R. R. Meyer. Nonlinear perturbation of linear programs. *SIAM Journal on Control and Optimization*, 17(6):745–752, 1979. 24

[72] István Maros and Csaba Mészáros. CUTEr version of the Maros and Mészáros Quadratic Programming Test Problem Set. `http://www.numerical.rl.ac.uk/cuter-www/Problems/marmes.html`. 86

[73] István Maros and Csaba Mészáros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11-12:671–681, 1999. 79, 86

[74] István Maros and Gautam Mitra. Strategies for creating advanced bases for large-scale linear programming problems. *Informs Journal on Computing*, 10:248–260, 1998. 78

[75] A. D. Martin. Mathematical programming of portfolio selections. *Management Science*, 1(2):152–166, 1955. 2

[76] The Mathworks Inc., Natick, MA. *MAT-File Format*, March 2010. `http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf`. 86

[77] Csaba Mészáros. The BPMPD interior point solver for convex quadratic programs. WP 98-8, Laboratory of Operations Research and Decision Systems, Hungarian Academy of Sciences, Budapest, 1998. 86

[78] MUMPS: a multifrontal massively parallel sparse direct solver. `http://mumps.enseeiht.fr/`. 100

[79] Walter Murray. Analytical expressions for eigenvalues and eigenvectors of the Hessian matrices of barrier and penalty functions. *Journal of Optimization Theory and Applications*, 7:189–196, 1971. 36

[80] Bruce A. Murtagh. *Advanced Linear Programming: Computation and Practice*. McGraw-Hill, New York, 1981. 86

[81] Katta G. Murty and Santosh N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39:117–129, 1987. 1

[82] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Verlag, New York, second edition, 2006. 4, 12, 14, 53

[83] Christopher C. Paige. Fast numerically stable computations for generalized linear least squares problems. *SIAM Journal of Numerical Analysis*, 16(1):165–171, 1979. 58

[84] Christopher C. Paige. Covariance matrix representation in linear filtering. *Contemporary Mathematics*, 47:309–321, 1985. 58

[85] Christopher C. Paige and Michael A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal of Numerical Analysis*, 12:617–629, 1975. 10

[86] Panos M. Pardalos and Georg Schnitger. Checking local optimality in constrained quadratic progamming is NP-hard. *Operations Research Letters*, 7(1):33–35, 1988. 1

[87] PARDISO parallel sparse solver. `http://www.pardiso-project.org`. 63, 100

[88] PDCO convex optimization software (MATLAB). `http://www.stanford.edu/group/SOL/software.html`. 31

[89] Michael J. D. Powell. A method for nonlinear constraints and minimization problems. In Roger Fletcher, editor, *Optimization*, pages 283–298. Academic Press, New York, 1969. 38

[90] MATLAB documentation for `quadprog`. `http://www.mathworks.com/help/toolbox/optim/ug/quadprog.html`. 60

[91] R. Tyrrell Rockafellar. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of Operations Research*, 1(2):97–116, 1976. 26, 41, 45

[92] R. Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal of Control and Optimization*, 14(5):877–898, 1976. 26

[93] Sartaj Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262–279, 1974. 1

[94] Michael A. Saunders. Cholesky-based methods for sparse least squares: The benefits of regularization. In L. Adams and J. L. Nazareth, editors, *Linear and Nonlinear Conjugate Gradient-Related Methods*, pages 92–100, Philadelphia, 1996. SIAM. 31

[95] Michael A. Saunders. Notes 4: The primal simplex method. `http://www.stanford.edu/class/msande318/notes.html`, 2010. Class notes for CME 338: Large-scale Numerical Optimization. 66

[96] Michael A. Saunders and John A. Tomlin. Solving regularized linear programs using barrier methods and KKT systems. Tech. Report SOL 96-4, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University, 1996. 31

[97] Michael A. Saunders and John A. Tomlin. Stable reduction to KKT systems in barrier methods for linear and quadratic programming. Tech. Report SOL 96-3, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University, 1996. 31

[98] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag, Berlin, Heidelberg, New York, 2003. 4

[99] MATLAB documentation for `spaugment`. `http://www.mathworks.com/help/techdoc/ref/spaugment.html`. 57

[100] John C. Stone, Patrick H. McAllister, and George B. Dantzig. Using the PILOT model to study the effects of technological change. Tech. Report SOL 86-16, Dept. of Operations Research, Stanford University, 1986. 2

[101] B. Stott, O. Alsac, and J. L. Marinho. The optimal power flow problem. In A. M. Erisman, K. W. Neves, and M. H. Dwarakanth, editors, *Proceedings of SIAM Conference on Electric Power Problems: The Mathematical Challenge*, pages 327–351, Philadelphia, 1980. SIAM. 2

[102] SuperLU software for sparse unsymmetric systems. `http://crd.lbl.gov/~xiaoye/SuperLU/`. 63

[103] Andrey N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-posed Problems*. V. H. Winston and Sons, Washington, D.C., 1977. Translated from Russian. 22

[104] UMFPACK solver for sparse $Ax = b$. `http://www.cise.ufl.edu/research/sparse/umfpack`. 63, 100

[105] Robert J. Vanderbei. LOQO: An interior point code for quadratic programming. Report SOR-94-15, Dept. of Statistics and Operations Research, Princeton University, 1994. 30

[106] Robert J. Vanderbei. Symmetric quasi-definite matrices. *SIAM Journal on Optimization*, 5(1):100–113, 1995. 30

[107] Stephen A. Vavasis. Quadratic programming is in NP. *Information Processing Letters*, 36(2):73–77, 1990. 1

[108] Andreas Wächter and Lorenz T. Biegler. Table of results on the CUTEr test set for IPOPT, 2004. `www.research.ibm.com/people/a/andreasw/papers/Ipopt-table.ps`. 79

[109] Andreas Wächter and Lorenz T. Biegler. Table of results on the CUTEr test set for KNITRO, 2004. `www.research.ibm.com/people/a/andreasw/papers/Knitro-table.pdf`. 79

[110] James H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, London, 1965. 21

[111] Elizabeth Wong, April 2010. Private communication. 78

[112] Stephen J. Wright. Implementing proximal point methods for linear programming. *Journal of Optimization Theory and Applications*, 65(3):531–554, 1990. 45

[113] Stephen J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia, 1997. 4

[114] Yinyu Ye. Further development of the interior algorithm for convex quadratic programming. Unpublished working paper: `http://www.stanford.edu/~yyye/yyye/newqp.ps`, 1987. 1