An iterative working-set method for large-scale non-convex quadratic programming

Nicholas I. M. Gould[1,2] and Philippe L. Toint[3,4]

ABSTRACT

We consider a working-set method for solving large-scale quadratic programming problems for which there is no requirement that the objective function be convex. The methods are iterative at two levels, one level relating to the selection of the current working set, and the second due to the method used to solve the equality-constrained problem for this working set. A preconditioned conjugate gradient method is used for this inner iteration, with the preconditioner chosen especially to ensure feasibility of the iterates. The preconditioner is updated at the conclusion of each outer iteration to ensure that this feasibility requirement persists. The well-known equivalence between the conjugate-gradient and Lanczos methods is exploited when finding directions of negative curvature. Details of an implementation—the Fortran 90 package `QPA` in the forthcoming GALAHAD library—are given.

---

[1] Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire, OX11 0QX, England, EU.
Email : n.gould@rl.ac.uk

[2] Current reports available from "http://www.numerical.rl.ac.uk/reports/reports.html".

[3] Department of Mathematics, Facultés Universitaires ND de la Paix, 61, rue de Bruxelles, B-5000 Namur, Belgium, EU.
Email : philippe.toint@fundp.ac.be

[4] Current reports available from "http://www.fundp.ac.be/~phtoint/pht/publications.html".

Computational Science and Engineering Department
Atlas Centre
Rutherford Appleton Laboratory
Oxfordshire OX11 0QX

July 6, 2001.

1 Introduction

In this paper, we consider a working-set method for finding a second-order critical point for the so-called l_1QP problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad f(x) = \tfrac{1}{2}\langle x, Hx\rangle + \langle c, x\rangle + \rho \left\|(Ax-b)^-\right\|_1, \tag{1.1}$$

where A and H are respectively m by n and n by n symmetric matrices, ρ is a given (fixed) parameter, and the components of w^- are the minima of w_i and zero. We shall refer to the components $\langle a_i, x\rangle - b_i$ of $Ax-b$ as its *constituents*. In practice, such problems often include "equality" terms

$$\rho\left|\langle a_i, x\rangle - b_i\right|,$$

"two-sided bounds"

$$\rho\left(\min(0, \langle a_i, x\rangle - l_i) + \max(0, \langle a_i, x\rangle - u_i)\right),$$

or "simple bounds"

$$\rho\left(\min(0, x_i - l_i) + \max(0, x_i - u_i)\right),$$

but for simplicity we shall ignore theses possibilities here except as to mention that such terms provide scope for algebraic improvements. We shall make no assumption that H is positive definite, and thus cannot guarantee that any critical point found actually solves (1.1). Not withstanding, we shall refer to any second-order critical point as a solution to (1.1).

Our particular interests are twofold. Firstly, we are interested in solving nonlinear programming problems using Fletcher (1982, 1987, Section 14.4)'s Sl_1QP method, which involves a sequence of subproblems of the form (1.1). Secondly, we are interested in solving quadratic programming (QP) problems

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \tfrac{1}{2}\langle x, Hx\rangle + \langle c, x\rangle \ \text{ subject to } \ Ax \geq b$$

using an exact penalty function formulation (1.1) for some sufficiently large ρ (see, for example, Conn and Sinclair, 1975, Han, 1981).

The reader may wonder why we are considering yet another active/working set method for the problem, especially since interior-point methods are now usually considered to be superior when there are many variables. The answer is simply that there are certain circumstances in which we believe working set methods still have advantages. The most obvious is when there is good *a priori* knowledge of what the optimal active set might be, since then one may anticipate there being relatively few working set changes before the optimal set is found. Interior point methods tend not to be able to take full advantage of such *a priori* information, preferring instead to calculate the optimal active set *ab initio*. Successive quadratic programming (SQP) algorithms for general nonlinear programming (NLP) problems are cases in point. It is well known (see Robinson, 1974) that these methods tend to predict the optimal active set for the NLP as they approach a limit point, and that the optimal active set from one QP is a good starting set for the next.

1.1 Notation

The symmetric matrix M is said to be *second-order sufficient* with respect to the m by n matrix A if and only if the augmented matrix

$$K = \begin{pmatrix} M & A^T \\ A & 0 \end{pmatrix} \tag{1.2}$$

is nonsingular and has precisely m negative eigenvalues. This is equivalent to requiring that $\langle y, My \rangle > 0$ for all nonzero y satisfying $Ay = 0$, or to the reduced matrix $N^T M N$ being positive definite, where the columns of N span the null-space of A (see, for instance, Chabrillac and Crouzeix, 1984, and Gould, 1985). If M is second-order sufficient, the augmented matrix K is said to be *standard*. Otherwise, it is *nonstandard*. The *inertia* of K is the triple,

$$\mathrm{In}(K) = (k_+, k_-, k_0),$$

where k_+, k_- and k_0 are respectively the numbers of positive, negative and zero eigenvalues of K. Thus K is standard if and only if $\mathrm{In}(K) = (n, m, 0)$.

2 The basic iteration

At the start of the k-th iteration, a set of constituents is assigned to the *working* set $\mathcal{W}_k$. The constituents in the working set are chosen as a (sub)set of the *active* set $\mathcal{A}_k = \mathcal{A}(x_k)$,

$$\mathcal{A}(x) = \{ i \mid \langle a_i, x \rangle = b_i \},$$

where x_k is the current estimate of the required solution. We shall also refer to the *violated and satisfied sets*, $\mathcal{V}_k = \mathcal{V}(x_k)$ and $\mathcal{S}_k = \mathcal{S}(x_k)$ respectively, where

$$\mathcal{V}(x) = \{ i \mid \langle a_i, x \rangle < b_i \} \quad \text{and} \quad \mathcal{S}(x) = \{ i \mid \langle a_i, x \rangle > b_i \}.$$

Notice that the active, violated and satisfied sets divide $\mathcal{R}^n$ into 3^m *partitions*.

An iteration is made up of three basic tasks. Firstly, a search direction s_k is chosen to reduce the quadratic function

$$q(x_k + s) = \tfrac{1}{2}\langle s, Hs \rangle + \langle g_k, s \rangle,$$

where

$$g_k = Hx_k + c - \rho \sum_{i \in \mathcal{V}_k \cup \mathcal{A}_k \setminus \mathcal{W}_k} a_i,$$

while at the same time ensuring that the constituents in the working set stay active by requiring that $A_k s = 0$, where the rows of A_k are those of A indexed by $\mathcal{W}_k$. Secondly, a step α_k is taken in the direction s_k to reduce $f(x_k + \alpha s_k)$. Finally, x_{k+1} is set to $x_k + \alpha_k s_k$, and the working set is updated to ensure progress, if at all possible, at the next iteration. We now discuss each of these stages in turn.

2.1 Computing the search direction

The search direction subproblem is to

$$\underset{s\in\mathbb{R}^n}{\text{minimize}} \;\; \tfrac{1}{2}\langle s, Hs\rangle + \langle g_k, s\rangle \;\; \text{subject to} \;\; A_k s = 0, \tag{2.1}$$

where the rows of A_k are the vectors a_i^T, $i \in \mathcal{W}_k$. As we wish to be able to solve large problems, we do not necessarily intend to solve (2.1) very accurately. In fact, we prefer to use an iterative method, since this gives us more flexibility.

In the absence of the constraint $A_k s = 0$, the method of conjugate gradients (see Hestenes and Stiefel, 1952) would be the method of choice, particularly if applied with a suitable preconditioner. This immediately suggests that if we were able implicitly to "project" the iterates into the null-space of A_k, we might be able to recreate a conjugate gradient-like method for constrained problems. Certainly, convex quadratic programming methods based upon conjugate directions have been suggested by many authors (see, for example, Best and Ritter, 1976, Benveniste, 1979, and Hestenes, 1980). The method we shall describe here is due to Gould, Hribar and Nocedal (1998), but has its origins in the proposals of Polyak (1969) and Coleman (1994).

Let us assume, for the time being, that H is second-order sufficient with respect to A_k, and thus that the solution to (2.1) occurs at its critical point. Let P_k be a projector into the null-space of A_k. Then our method may be described as follows:

Algorithm 2.1: Preconditioned conjugate gradients for (2.1)

Given $s_{0,k} = 0$, set $g_{0,k} = g_k$, and let

$$v_{0,k} \;=\; P_k\, g_{0,k} \tag{2.2}$$

and $p_{0,k} = -v_{0,k}$. For $j = 0, 1, \ldots,$ until convergence, perform the iteration,

$$\begin{aligned}
\sigma_{j,k} &= \langle g_{j,k}, v_{j,k}\rangle / \langle p_{j,k}, Hp_{j,k}\rangle \\
s_{j+1,k} &= s_{j,k} + \sigma_{j,k} p_{j,k} \\
g_{j+1,k} &= g_{j,k} + \sigma_{j,k} H p_{j,k} \\
v_{j+1,k} &= P_k\, g_{j+1,k} \\
\beta_{j,k} &= \langle g_{j+1,k}, v_{j+1,k}\rangle / \langle g_{j,k}, v_{j,k}\rangle \\
p_{j+1,k} &= -v_{j+1,k} + \beta_{j,k} p_{j,k}
\end{aligned} \tag{2.3}$$

The matrix P_k plays the dual role of projecting the iterates into the required null-space, and preconditioning the iteration. A number of forms are possible, but the most appealing is to solve the augmented system

$$\begin{pmatrix} M_k & A_k^T \\ A_k & 0 \end{pmatrix} \begin{pmatrix} v_{j+1,k} \\ w_{j+1,k} \end{pmatrix} = \begin{pmatrix} g_{j+1,k} \\ 0 \end{pmatrix} \tag{2.4}$$

for some auxiliary vector $w_{j+1,k}$ and second-order sufficient matrix M_k. Thus, in practice, we use Algorithm 2.1, but replace (2.2)/(2.3) by (2.4). The resulting method is known as the *projected*

preconditioned conjugate gradient method. Of course, we solve (2.4) using one of the stable symmetric, indefinite factorization methods introduced by Bunch and Parlett (1971) and later improved by Bunch and Kaufman (1977) and Fletcher (1976) in the dense case, Duff, Reid, Munksgaard and Neilsen (1979) and Duff and Reid (1983) in the sparse case, and Ashcraft, Grimes and Lewis (1998) in all cases.

Notice that the first-order optimality conditions imply that the solution to (2.1) satisfies the augmented system

$$\begin{pmatrix} H & A_k^T \\ A_k & 0 \end{pmatrix} \begin{pmatrix} s \\ -y \end{pmatrix} = \begin{pmatrix} -g_k \\ 0 \end{pmatrix}. \tag{2.5}$$

Therefore Algorithm 2.1 is only appropriate if the cost of solving (2.5), is significantly greater than that for a sequence of (2.4), and the art is in choosing the preconditioner so that this is so. If M_k is diagonal, range- or null-space approaches (see Gill, Murray and Wright, 1981, Section 5.4.1) are sometimes preferable to (2.4), illustrating the flexibility of Algorithm 2.1.

We must enter a word of caution here. While Algorithm 2.1 may appear to be attractive, it is crucial that (2.4) be solved accurately, as otherwise, recurrences which rely on $v_{j+1,k}$ lying in the null-space of A_k may be invalid. A particularly troublesome case occurs when $g_{j+1,k}$ is large but $v_{j+1,k}$ is small, for then (2.4) indicates that $w_{j+1,k}$ will usually be large—such cases often occur in SQP methods when approaching the solution of a nonlinear program. In this case, although it is possible to compute the composite vector $(v_{j+1,k} \;\; w_{j+1,k})$ to high relative accuracy provided a stable factorization is used, the components $v_{j+1,k}$ may have little relative accuracy. A number of precautions, including iterative refinement, have been proposed by Gould et al. (1998), but the most effective appears to be to note that (2.4) is equivalent to

$$\begin{pmatrix} M_k & A_k^T \\ A_k & 0 \end{pmatrix} \begin{pmatrix} v_{j+1,k} \\ u_{j+1,k} \end{pmatrix} = \begin{pmatrix} g_{j+1,k} - A_k^T y_{j+1,k} \\ 0 \end{pmatrix} \tag{2.6}$$

where $w_{j+1,k} = y_{j+1,k} + u_{j+1,k}$, and to choose $y_{j+1,k}$ so that $\|g_{j+1,k} - A_k^T y_{j+1,k}\|$ is small. For then $v_{j+1,k}$ may be computed with much higher relative accuracy, and the iterates lie substantially closer to the null-space of A_k. Picking $y_{j+1,k}$ as the previously-generated $u_{j,k}$ appears to be effective in practice.

We now turn to the possibility that H is not second-order sufficient with respect to A_k. If this is the case, the problem either has a subspace of weak minimizers, or is unbounded from below. The first strategy which suggests itself is to follow the proposal of Steihaug (1983) and to monitor $\langle p_{j,k}, Hp_{j,k} \rangle$ as the iteration progresses. If this is negative, $p_{j,k}$ is a direction of negative curvature; if it is zero and $\langle g_{j,k}, v_{j,k} \rangle \neq 0$, $p_{j,k}$ is a direction of linear infinite descent. In either case, we pick $s_k = \pm p_{j,k}/\|p_{j,k}\|$, where the sign is chosen to ensure that $\langle s_k, g_k \rangle \leq 0$.

However, since such the direction is somewhat arbitrary, we prefer a slightly more sophisticated approach which aims to produce a vector which is closer to the eigenvector corresponding to the leftmost eigenvalue of H constrained to lie in the subspace $A_k s = 0$. We merely note, as have many others, that the conjugate gradient and Lanczos methods are two different ways of producing a basis for the same (Krylov) subspace, and while the former is most usually associated with solving linear systems, the latter is best known as a method for finding (in particular) extreme eigenvalues of large symmetric matrices. We shall not give details here, since our method has essentially already been defined in full by Gould, Lucidi, Roma and Toint (1999)—the (GLTR) method given in this paper, and the resulting HSL (2000) code `VF05`, also involved a trust-region

constraint, which we may either set to a large value (which will bias the solution towards the leftmost eigenvalue) or a smaller value which increases the contribution from the projected preconditioned steepest descent direction, $-P_k g_k$. Experience reported in Gould et al. (1999) has suggested that letting the Lanczos method run for a few (say 5) iterations beyond the first at which negative curvature is encountered can often significantly improve the quality of the negative curvature direction found.

2.2 Computing the step

This part is entirely standard. The objective function $\phi_k(\alpha) = f(x_k + \alpha s_k)$ is a piecewise quadratic function of α, and we aim to find its first local minimizer as α increases from zero. We call the values at which $\langle a_i, x_k + \alpha s_k \rangle = b_i$, $i \notin \mathcal{W}_k$, the *breakpoints*, and, starting from $\alpha_{0,k} = 0$, we examine the behaviour of $\phi_k(\alpha)$ between consecutive breakpoints $\alpha_{j,k} < \alpha_{j+1,k}$ until a suitable value is found. There are three possibilities.

Firstly, the slope of $\phi_k(\alpha)$ may be strictly positive for all small $\alpha \geq \alpha_{j,k}$. In this case, we choose the local minimizer $\alpha_k = \alpha_{j,k}$. Secondly, $\phi(\alpha)$ may have a minimizer $\alpha_{j,k} < \alpha^{\mathrm{M}}_{j,k} < \alpha_{j+1,k}$. If this occurs, we choose $\alpha_k = \alpha^{\mathrm{M}}_{j,k}$. Finally, the required minimizer may lie at or beyond $\alpha_{j+1,k}$, and we compute the next breakpoint $\alpha_{j+2,k}$ as well as updating the slope at $\alpha_{j+1,k}$.

To examine these possibilities, we need to be able to calculate and sort the breakpoints in increasing order, and to evaluate $\phi_k(\alpha)$ as α increases. For efficiency the sorting is perhaps best achieved using the Heapsort method (see Williams (1964)), which is particularly appropriate, as it performs a partial sort from which the $i+1$-st smallest member of a set may be found very efficiently once the first i smallest are known. Thus breakpoints beyond the minimizer need not be completely sorted. The function $\phi_k(\alpha)$ may be expressed as

$$\phi_k(\alpha) = \phi_{j,k} + \alpha \phi'_{j,k} + \tfrac{1}{2}\alpha^2 \phi''_k,$$

where $\phi''_k = \langle s_k, H s_k \rangle$, for all $\alpha_{j,k} \leq \alpha < \alpha_{j+1,k}$. A simple calculation reveals that

$$\begin{aligned} \phi'_{j+1,k} &= \phi'_{j,k} + \rho \sum_{i \in \mathcal{B}_{j+1,k}} |\langle a_i, s_k \rangle| \quad \text{and} \\ \phi_{j+1,k} &= \phi_{j,k} + \alpha_{j+1,k}(\phi'_{j,k} - \phi'_{j+1,k}), \end{aligned}$$

where $\mathcal{B}_{j+1,k}$ are the indices of constituents which define the $j+1$-st breakpoint. Of course, the initial values are

$$\phi_{0,k} = f(x_k) \ \text{ and } \ \phi'_{0,k} = \langle g_k, s_k \rangle - \rho \sum_{i \in \mathcal{B}_{j+1,0}} \langle a_i, s_k \rangle,$$

where

$$\mathcal{B}_{j+1,0} = \left\{ 1 \leq i \leq m \;\middle|\; \langle a_i, x_k \rangle < b_i, \ \text{ or } \ \langle a_i, x_k \rangle = b_i \ \text{ and } \ \langle a_i, s_k \rangle < 0 \right\}.$$

One other eventuality is that the last breakpoint is not a local minimizer, and the curvature is negative. In this case, $f(x)$ is unbounded from below along the arc $x_k + \alpha s_k$, and the algorithm should be terminated. Strictly, this might be viewed as *the* major weakness of the whole approach (particularly if we are interested in solving quadratic programs), since $f(x)$ is (globally) unbounded from below whenever H is indefinite.

We have to be slightly cautious here, since it is an open question as far as we know (and contrary to claims made by Conn and Sinclair, 1975) whether the algorithm as it stands might

actually cycle infinitely through different sets $\mathcal{V}_k \cup \mathcal{A}_k \setminus \mathcal{W}_k$ for the same working set $\mathcal{W}_k$. This might happen, for example, if the unconstrained minimizer of $\phi_k(\alpha)$ lies between breakpoints beyond the first for all $k \geq k_0$ for some k_0. In order to prevent this (remote) possibility, the simplest precaution is to stop at the breakpoint directly before the unconstrained minimizer once ever so often, since this will result in a gradual increase in the number of constituents in the working set, and once $|\mathcal{W}_k| = n$, this working set cannot reappear.

An alternative to the forward-stepping strategy given here might be to perform a Armijo-type backtracking linesearch, as suggested in broadly similar circumstances by Bertsekas (1982), Calamai and Moré (1987) and Toint (1988), to avoid stepping through a large number of breakpoints. We have not investigated this possibility.

2.3 Updating the working set

The final step of our iteration is to decide what the working set should be for the next iteration. There are four possible outcomes from the linesearch. Firstly, we may detect that $f(x)$ is unbounded from below, in which case the algorithm will be terminated. Secondly, we may stop at a breakpoint. If this is the case, *one* of the constituents which becomes active at x_{k+1} should be added to $\mathcal{W}_k$ to form $\mathcal{W}_{k+1}$. Notice that in principle it doesn't matter which constituent is added, since the gradient of each is linearly independent of the rows of A_k because $\langle a_i, s_k \rangle \neq 0$ while $As_k = 0$, but it may be wise to pick the constituent for which $|\langle a_i, s_k \rangle| / \|a_i\|_2 \|s_k\|_2$ is largest as this then gives the "most" independent a_i. The third possibility is that the linesearch stops beyond the first breakpoint, but between subsequent ones. In this case, the minimizer cannot occur within the current partition, but may nevertheless might result from the same working set. We thus simply adjust g_{k+1} to account for the change in partition, but retain $\mathcal{W}_{k+1} = \mathcal{W}_k$. The final possibility is that the linesearch stops before the first breakpoint, and it is this case that we now consider.

Ideally, the computed search direction would be the solution to (2.1). In this case, x_{k+1} is a candidate solution to the original problem. In order to investigate this possibility, we need to compute Lagrange multipliers at the solution to (2.1). If the exact solution to (2.1) has been found, the vector $v_{j+1,k}$ will be zero. As a consequence (2.4) implies that

$$A_k^T w_{j+1,k} = g_{j+1,k} = H s_{j+1,k} + g_k = H s_k + g_k,$$

and thus that $y_k = w_{j+1,k}$ are Lagrange multipliers. If $0 \leq y_k \leq 1$, standard optimality conditions (see, for example, Fletcher, 1987) imply that x_{k+1} is a first-order critical point for (1.1) (Note that we cannot be sure that this is a second-order critical point unless either $M_k = H$, in which case H is second-order sufficient with respect to A_k or the whole of the null space of A_k has been investigated by Algorithm 2.1, since then the algorithm will have established that H is positive definite in this space.) On the other hand, if there is a component $[y_k]_i \notin [0, 1]$, further progress is possible (provided other active constituents not present in the current working set do not interfere) simply by deleting the corresponding constituent from the working set. As is common in such a case, there may be more than one candidate for deletion, and there are a number of possible rules to decide which is ultimately chosen.

Of course, one of the advantages of using the preconditioned projected conjugate gradient method is the scope for terminating the iteration well before optimality. This has, unfortunately, a downside as well. So long as we solve (2.1) (and so long as a suitable anti-cycling rule is chosen),

we ensure that we cannot return to the working set W_k once we have left x_{k+1}. Thus, as there are only a finite number of possible working sets, our algorithm would be finite. If we do not solve (2.1) exactly, and we subsequently remove a constituent from the working set based on *approximate* Lagrange multiplier estimates, it may re-enter at a later stage. Unless care is taken this *zigzagging* between working sets may lead to non convergence (see, for example, Fletcher, 1987, Section 11.3). We are unaware of suitable termination rules for the conjugate gradient method that guarantee to prevent this, except, of course, to run the method until the gradient is "numerically" zero.

3 Algebraic issues

The computation is divided into a sequence of major iterations. At the start of each major iteration, a factorization of the preconditioning matrix

$$K_k = \begin{pmatrix} M_k & A_k^T \\ A_k & 0 \end{pmatrix},$$

involving the set of constituents in the current working set, is found—we shall call the set of these constituents at the start of a major iteration the *reference* set. The symmetric matrix M_k is chosen so that it is second-order sufficient, but is otherwise arbitrary. In particular, there is no requirement that M_k be positive definite, although if this were the case it would automatically be second-order sufficient. Common choices are $M_k = H$ (if this is allowed, and if K_k does not suffer significant fill-in) or $M_k = I$. We stress here that although it is desirable to choose a good approximation of H, the overriding concern is that M_k be second-order sufficient.

Having determined the factors of K_k, all subsequent linear systems during the current major iteration are solved using the Schur complement method. That is to say, if we require the solution of a system

$$\begin{pmatrix} M_\ell & A_\ell^T \\ A_\ell & 0 \end{pmatrix} \begin{pmatrix} s_\ell \\ t_\ell \end{pmatrix} = - \begin{pmatrix} g_\ell \\ c_\ell \end{pmatrix}, \tag{3.1}$$

for $\ell \geq k$, and if

$$M_k = M_\ell,$$

the solution is obtained using the factors of K_k and an appropriate Schur complement involving M_k, A_k and A_ℓ—notice here that thus far we do not allow M_ℓ to change during the course of a major iteration. To be specific, suppose without loss of generality, that

$$A_k = \begin{pmatrix} A_C \\ A_D \end{pmatrix}, \quad A_\ell = \begin{pmatrix} A_C \\ A_A \end{pmatrix} \quad \text{and} \quad c_\ell = \begin{pmatrix} c_C \\ c_A \end{pmatrix}, \tag{3.2}$$

that is that the rows A_C are common to A_k and A_ℓ, but that the rows A_D in A_k are replaced by the rows A_A in A_ℓ. In this case, the solution to (3.1) also satisfies the expanded system

$$\begin{pmatrix} M_k & A_C^T & A_D^T & A_A^T & 0 \\ A_C & 0 & 0 & 0 & 0 \\ A_D & 0 & 0 & 0 & I \\ A_A & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \end{pmatrix} \begin{pmatrix} s_\ell \\ t_C \\ t_D \\ t_A \\ u_\ell \end{pmatrix} = - \begin{pmatrix} g_\ell \\ c_C \\ 0 \\ c_A \\ 0 \end{pmatrix}, \tag{3.3}$$

where we recover

$$t_\ell = \begin{pmatrix} t_C \\ t_A \end{pmatrix}.$$

Notice that the leading 3 by 3 block of the coefficient matrix of (3.3) is simply K_k, and thus that the system may be written as

$$\begin{pmatrix} K_k & B_\ell^T \\ B_\ell & 0 \end{pmatrix} \begin{pmatrix} v_\ell \\ w_\ell \end{pmatrix} = - \begin{pmatrix} h_\ell \\ d_\ell \end{pmatrix}, \tag{3.4}$$

for the appropriately repartitioned data

$$B_\ell = \begin{pmatrix} A_A & 0 & 0 \\ 0 & 0 & I \end{pmatrix}, \quad v_\ell = \begin{pmatrix} s_\ell \\ t_C \\ t_D \end{pmatrix} \quad \text{and} \quad w_\ell = \begin{pmatrix} t_A \\ u_\ell \end{pmatrix},$$

and solution

$$h_\ell = \begin{pmatrix} g_\ell \\ c_C \\ 0 \end{pmatrix} \quad \text{and} \quad d_\ell = \begin{pmatrix} c_A \\ 0 \end{pmatrix}.$$

Thus (3.4) can be solved in the standard way using the factors of K_k and those of the Schur complement $S_l = -B_l K_k^{-1} B_\ell^T$. Crucially, the factors of S_ℓ may be updated rather than recomputed every time a constituent is added to or removed from the working set. It is usual to store the growing matrix S_ℓ and its factors as dense matrices; as a consequence each major iteration is concluded when the dimension of S_ℓ exceeds a given upper limit (default, 75), or perhaps when the cost of continuing to enlarge the Schur complement method is believed to exceed that of re-factorizing K_ℓ.

This method was first suggested by Bisschop and Meeraus (1977), and championed by Gill, Murray, Saunders and Wright (1990, 1991). We have implemented such a method as part of the package `MA39` in HSL—the package is actually designed to handle updates in the unsymmetric case, but is capable of exploiting both symmetry and even *a priori* knowledge that S_ℓ is definite. In principle, a symmetric indefinite factorization of S_ℓ is both possible, and possible to update. However, the details are complicated (see Sorensen, 1977), and we have chosen instead to use a non-symmetric (QR) factorization since updates are then relatively straightforward.

It is important to be able to check the inertia of K_ℓ at every iteration, but fortunately this can be achieved knowing those of K_k and S_ℓ using Sylvester's law. Specifically, a very minor modification of Gill et al. (1991, Lemma 7.2) shows that so long as both K_k and K_ℓ are standard,

$$\mathrm{In}(S_\ell) = (\sigma_-, \sigma_+, 0), \tag{3.5}$$

where σ_+ constituents have been added since the start of the major iteration, and σ_- have been deleted. Since we require that K_k is standard, it follows that if, at any stage, the inertia of S_ℓ does not agree with (3.5), it must be because K_ℓ is nonstandard. It is easy to check this condition since the inertia of S_ℓ may be recurred as its factors are updated (in our case, since we are using the non-symmetric QR factors, we record the determinants S_ℓ on subsequent iterations—a change in sign indicates an extra negative eigenvalue, while a repeated sign indicates an extra positive one—directly from the products of those of Q and R. We ensure by construction that $\det Q = 1$, while the eigenvalues of R are merely its diagonal entries.). We now consider the implication of adding and deleting constituents for the inertia of K_ℓ.

3.1 Adding a constituent

If K_ℓ is standard, and we add a constituent to the working set, $K_{\ell+1}$ is also standard. This follows immediately, since as we have already said K_ℓ being standard is equivalent to $N_\ell^T M_k N_\ell$ being positive definite, where the columns of N_ℓ form am orthonormal basis for the null-space of the full-rank matrix A_ℓ, the fact that

$$N_\ell = \begin{pmatrix} N_{\ell+1} \\ n^T \end{pmatrix} Q$$

for some vector n and orthonormal matrix Q (see Gill, Golub, Murray and Saunders, 1974), and the observation that $N_{\ell+1}^T M_k N_{\ell+1}$ is then a principal submatrix of the positive definite matrix $Q^T N_\ell^T M_k N_\ell Q$ and hence is itself positive definite.

3.2 Deleting a constituent

Complications arise when we delete a constituent, since then it does not automatically follow that $K_{\ell+1}$ is standard even if K_ℓ was. Fortunately, provided we are prepared to modify M_k when necessary, we can avoid this potential defect.

Suppose the columns of N form am orthonormal basis for the null-space of the full-rank matrix A, i.e., $AN = 0$. Suppose furthermore that $N^T M N$ is positive definite. Let

$$A = \begin{pmatrix} \bar{A} \\ a^T \end{pmatrix}$$

in which case

$$\bar{A}N = 0 \ \text{ and } \ a^T N = 0. \tag{3.6}$$

Then there is a vector n for which the columns of $(N \quad n)$ form an orthonormal basis for the null-space of $\bar{A}$, i.e.,

$$\bar{A}N = 0, \quad N^T n = 0 \ \text{ and } \ \bar{A}n = 0. \tag{3.7}$$

Now consider the matrix $M + \delta a a^T$ for some scalar δ. Then

$$\begin{pmatrix} N^T \\ n \end{pmatrix} (M + \delta a a^T)(N \quad n) = \begin{pmatrix} N^T M N & N^T M n \\ n^T M N & \langle n, Mn \rangle + \delta \langle a, n \rangle^2 \end{pmatrix}, \tag{3.8}$$

where we have used the fact that $N^T a = 0$ from (3.6). Since the columns of $(A^T \quad N)$ form a basis for $\mathbb{R}^n$, we may write

$$n = \bar{A}^T w + \alpha a + N v$$

for some vectors finite w and v and scalar α. Premultiplying by n^T, and using (3.7) and the orthonormality of $(N \quad n)$ yields that $1 = \alpha \langle a, n \rangle$, from which we deduce that $\langle a, n \rangle \neq 0$. Thus we can ensure that the matrix (3.8) is positive definite by, if necessary, picking δ sufficiently large.

Of course, we are not basing our method on (3.8), but rather on being able to solve augmented systems like (3.3). In order to accommodate changes to M_k of the type suggested above, we actually need to solve systems of the form

$$\begin{pmatrix} M_k & A_C^T & A_D^T & A_A^T & 0 \\ A_C & 0 & 0 & 0 & 0 \\ A_D & 0 & 0 & 0 & I \\ A_A & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & D_D \end{pmatrix} \begin{pmatrix} s_\ell \\ t_C \\ t_D \\ t_A \\ u_\ell \end{pmatrix} = - \begin{pmatrix} g_\ell \\ c_C \\ 0 \\ c_A \\ 0 \end{pmatrix}, \tag{3.9}$$

where D_D is a diagonal matrix. To see why this is appropriate, on eliminating t_D and u_ℓ and using (3.2), we obtain

$$\begin{pmatrix} M_k + A_D^T D_D A_D & A_\ell^T \\ A_\ell & 0 \end{pmatrix} \begin{pmatrix} s_\ell \\ t_\ell \end{pmatrix} = - \begin{pmatrix} g_\ell \\ c_\ell \end{pmatrix}, \tag{3.10}$$

which is (3.1) with

$$M_\ell = M_k + A_D^T D_D A_D.$$

A diagonal entry in D_D need only be nonzero if the resulting K_ℓ would otherwise be nonstandard. Crucially, as before, the leading 3 by 3 block of (3.9) is simply K_k, and thus that the system may be written as

$$\begin{pmatrix} K_k & B_\ell^T \\ B_\ell & D_\ell \end{pmatrix} \begin{pmatrix} v_\ell \\ w_\ell \end{pmatrix} = - \begin{pmatrix} h_\ell \\ d_\ell \end{pmatrix}, \tag{3.11}$$

where

$$D_\ell = \begin{pmatrix} 0 & 0 \\ 0 & D_D \end{pmatrix}.$$

Thus (3.11) can be solved in the standard way using the factors of K_k and those of the Schur complement $S_l = D_\ell - B_l K_k^{-1} B_\ell^T$, and the factors of the latter can be updated as the working set changes.

To see this, suppose (without loss of generality) that we have added constituent gradients whose Jacobian is A_A, and now intend to remove the first row from A_D. The resulting Schur complement is then

$$\begin{aligned} S(\delta) &= \begin{pmatrix} 0 & 0 \\ 0 & \delta \end{pmatrix} - \begin{pmatrix} -BK^{-1}B^T & -BK^{-1}b \\ -b^T K^{-1} B^T & -b^T K^{-1} b \end{pmatrix} \\ &= \begin{pmatrix} QR & v \\ v^T & \delta + \beta \end{pmatrix} \\ &= \begin{pmatrix} Q & 0 \\ 0 & 1 \end{pmatrix} H^T H \begin{pmatrix} R & w \\ v^T & \delta + \beta \end{pmatrix} = \bar{Q} \bar{R}(\delta), \end{aligned}$$

where

$$B = \begin{pmatrix} A_A & 0 & 0 \end{pmatrix}, \quad b^T = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix},$$
$$QR = -BK^{-1}B^T, \quad v = -BK^{-1}b, \quad \beta = -b^T K^{-1} b \text{ and } w = Q^T v,$$

and the orthonormal matrix

$$H = \begin{pmatrix} \bar{H} & h \end{pmatrix}$$

is a product of plane rotations chosen to eliminate the spike v^T. But then

$$\begin{aligned} \bar{R}(\delta) &= H \begin{pmatrix} R & w \\ v^T & \delta + \beta \end{pmatrix} = \begin{pmatrix} \bar{H}R + hv^T & \bar{H}w + \beta h \end{pmatrix} + \delta \begin{pmatrix} 0 & h \end{pmatrix} \\ &= \bar{R}(0) + \delta \begin{pmatrix} 0 & h \end{pmatrix} \end{aligned}$$

and the introduction of δ simply adds δh to the last row of the updated upper triangular factor $\bar{R}(0)$. Fortunately, the updated orthonormal matrix is

$$\bar{Q} = \begin{pmatrix} Q & 0 \\ 0 & 1 \end{pmatrix} H^T = \begin{pmatrix} Q\bar{H}^T \\ h^T \end{pmatrix},$$

and hence h is available. We can evaluate the sign of the determinant of $\bar{R}(0)$, and if this indicates that the new K is nonstandard, add a sufficiently large δ to change the sign of the last diagonal of $\bar{R}$.

3.3 Deleting a non-reference constituent which has previously been added

One case of interest is when a constituent, not contained in the reference set but added to the working set on a subsequent iteration, is now asked to leave. The obvious approach is to proceed exactly as in Section 3.2 by adding an appropriate extra row/column to the Schur complement S_{l+1} (perhaps including a nonzero diagonal term δ if this is needed to ensure that K_{l+1} is standard). An alternative is to "undo" the previous addition by removing the row and column from S_ℓ corresponding to the outgoing constituent row $\begin{pmatrix} a_A^T & 0 & 0 & 0 & 0 \end{pmatrix}$. However complications arise if we intend, once again, to ensure that K_{l+1} is standard by adding a (possible nonzero) multiple, δ, of $a_A a_A^T$ to M_l.

To see this, and to see how to avoid any difficulty, consider the analog of (3.9),

$$\begin{pmatrix} M_k & A_C^T & A_D^T & A_{N_0}^T & A_N^T & A_A^T & 0 & 0 & 0 \\ A_C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A_D & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 \\ A_{N_0} & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 \\ A_N & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I \\ A_A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & D_D & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & D_N \end{pmatrix} \begin{pmatrix} s_\ell \\ t_C \\ t_D \\ t_{N_0} \\ t_N \\ t_A \\ u_\ell \\ v_\ell \\ w_\ell \end{pmatrix} = - \begin{pmatrix} g_\ell \\ c_C \\ 0 \\ 0 \\ 0 \\ c_A \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{3.12}$$

where D_D and D_N ($\neq 0$) are diagonal matrices, and where the rows A_N and A_{N_0} correspond to non-reference constituents that are added at some stage but subsequently removed—the subscripts N and N_0 indicate those constituents for which modifications are, and are not, needed on removal, respectively. Since our intention is to avoid the introduction of the last two block rows and columns of the coefficient matrix in (3.12), eliminating the variables v_ℓ and w_ℓ (and implicitly t_{N_0}) leads to

$$\begin{pmatrix} M_k & A_C^T & A_D^T & A_N^T & A_A^T & 0 \\ A_C & 0 & 0 & 0 & 0 & 0 \\ A_D & 0 & 0 & 0 & 0 & I \\ A_N & 0 & 0 & -D_N^{-1} & 0 & 0 \\ A_A & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & D_D \end{pmatrix} \begin{pmatrix} s_\ell \\ t_C \\ t_D \\ t_N \\ t_A \\ u_\ell \end{pmatrix} = - \begin{pmatrix} g_\ell \\ c_C \\ 0 \\ 0 \\ c_A \\ 0 \end{pmatrix}, \tag{3.13}$$

which is exactly of the form (3.11) with data

$$B_\ell = \begin{pmatrix} A_N & 0 & 0 \\ A_A & 0 & 0 \\ 0 & 0 & I \end{pmatrix}, \quad D_\ell = \begin{pmatrix} -D_N^{-1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & D_D \end{pmatrix}, \quad v_\ell = \begin{pmatrix} s_\ell \\ t_C \\ t_D \end{pmatrix} \text{ and } w_\ell = \begin{pmatrix} t_N \\ t_A \\ u_\ell \end{pmatrix},$$

and solution

$$h_\ell = \begin{pmatrix} g_\ell \\ c_C \\ 0 \end{pmatrix} \text{ and } d_\ell = \begin{pmatrix} 0 \\ c_A \\ 0 \end{pmatrix}.$$

Notice that the essential difference between this case and its predecessor is that the extra diagonal terms $-D_N^{-1}$ will appear in the Schur complement S_ℓ whenever removing a non-reference

constituent might otherwise lead to a non-standard K_ℓ. We now consider how this manifests itself.

Suppose that, at iteration ℓ, a non-reference constituent is to be deleted, We might then try to remove its corresponding row and column from the p by p Schur complement S_ℓ to obtain $S_{\ell+1}$. Furthermore, suppose (without loss of generality) that it is the last row and column of S_ℓ that is to be removed, that

$$S_\ell = QR \tag{3.14}$$

for some upper triangular R and orthonormal Q—if not, standard orthogonal matrix techniques (see, for example, Gill et al., 1974) may be used to reorder the rows and columns of S_ℓ, and recover the required form of the factors, so that this is so—and that the sign of the determinant of S_ℓ is known. Given (3.14), we now choose the orthogonal matrix H, a product of plane rotations (the j-th of which involves columns $p-j$ and p, for $j = 1, \ldots, p-1$) so that

$$QH = \begin{pmatrix} \bar{Q} & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad H^T R = \begin{pmatrix} \bar{R} & r \\ l^T & \rho \end{pmatrix}.$$

for new upper triangular $\bar{R}$ and orthonormal $\bar{Q}$. Then it is safe to use $S_{\ell+1} = \bar{S}$, where

$$\bar{S} = \bar{Q}\bar{R}, \tag{3.15}$$

so long as the sign of the determinant of $\bar{S}$ is the opposite of that of S_ℓ (this follows from the inertial result (3.5))—in practice, we ensure that $\det(Q) = 1$ for all orthonormal Q, so that sign of $det(S_{\ell+1})$ is the product of the signs of the diagonal entries of $\bar{R}$. Conversely, if the signs are the same, of if $\bar{R}$ is singular, this indicates that it is not safe to remove the row/column from S_ℓ, and instead we need to introduce an extra diagonal term (c.f. $-D_N^{-1}$ in (3.13)) into S_ℓ so as to change its inertia.

At face value, this might look rather expensive, since we might have to "undo" the transformation using H if (3.15) turns out to be unsatisfactory. Fortunately, it is not actually necessary to apply H to Q or R in order to find the diagonals of $\bar{R}$. All that is needed is (1) to compute the plane rotations to reduce last row of Q to e_p^T (in temporary store), and (2) to apply these rotations to the diagonal entries of R (again in temporary store).

3.4 Adding a constituent which has previously been deleted

Another case of interest is when a constituent which is in the reference set, but which has been deleted from the working set at iteration j, now wishes to re-enter the working set. According to Section 3.1, the simplest mechanism would then be to introduce the constituent's gradient as the new last row of $K_{\ell+1}$, and we can be assured that the resulting $K_{\ell+1}$ is standard. An alternative is to "undo" the previous deletion by removing the row and column from S_ℓ corresponding to one of the artificial rows $\begin{pmatrix} 0 & 0 & I & 0 & 0 \end{pmatrix}$ which is added to K_j to effect the deletion (see equation (3.3)). However, the reader may then be concerned that, according to (3.9), the original deletion may have actually required that we add a row of $\begin{pmatrix} 0 & 0 & I & 0 & D_D \end{pmatrix}$ for nonzero D_D rather than $\begin{pmatrix} 0 & 0 & I & 0 & 0 \end{pmatrix}$ so as to ensure that the resulting K_{j+1} was standard. If we delete this row from K_ℓ, can we be sure that resulting $K_{\ell+1}$ is standard? To see that this is indeed the case,

note that if we reintroduce the row a_D^T of A_D, we must have that

$$\begin{pmatrix} M_k + A_D^T D_D A_D & A_\ell^T & a_D \\ A_\ell & 0 & 0 \\ a_D^T & 0 & 0 \end{pmatrix}$$

is standard because of our discussion in Section 3.1. Since this is equivalent to saying that

$$\langle s, (M_k + A_D^T D_D A_D) s \rangle > 0$$

for all s for which

$$\begin{pmatrix} A_\ell \\ a_D^T \end{pmatrix} s = 0,$$

we then have that

$$0 < \langle s, (M_k + A_D^T D_D A_D) s \rangle = \langle s, (M_k + \bar{A}_D^T \bar{D}_D \bar{A}_D + \delta a_D a_D^T) s \rangle = \langle s, (M_k + \bar{A}_D^T \bar{D}_D \bar{A}_D) s \rangle$$

where δ is the diagonal of D_D corresponding to the row a_D^T, and where $\bar{D}_D$ and $\bar{A}_D$ are the remaining rows of D_D and A_D respectively. Thus

$$\begin{pmatrix} M_k + \bar{A}_D^T \bar{D}_D \bar{A}_D & A_\ell^T & a_D \\ A_\ell & 0 & 0 \\ a_D^T & 0 & 0 \end{pmatrix}$$

is standard, and thus removing the row $\begin{pmatrix} 0 & 0 & 1 & 0 & \delta \end{pmatrix}$ has the desired effect.

4 Other details

As Roger Fletcher (U. Dundee) has cautioned us on a number of occasions, computational quadratic programming is all about seemingly insignificant, but, in practice, absolutely vital details. In this section we describe these details for our Fortran 90 package `QPA` (from the forthcoming GALAHAD optimization library) that implements the basic algorithm outlined in this paper. An enhanced version of `QPA`, `HSL_VE19`, in which `QPA`'s core linear algebra package `MA27` is replaced its more powerful successor `MA57` (see Duff, 2001), will be available in the next release of HSL (2001).

4.1 Constituent deletion strategies

Among the many strategies for removing constituents that have been suggested, our default is to pick the constituent whose Lagrange multiplier is furthest from the interval $[0, 1]$—ties are resolved by selecting the one with smallest index. Another strategy that is available as an option is to remove the last constituent that was added to the working set for which $[y_k]_i \notin [0, 1]$, since this gives priority to those non-reference constituents whose removal from the Schur complement is cheapest to effect. A third option is a variation on both these themes in which the last k constituents (for some user-given k) in the working set for which $[y_k]_i \notin [0, 1]$ are candidates, and the one whose Lagrange multiplier is furthest from the interval $[0, 1]$ is selected. Again, this gives some precedence to constituents whose removal may be effected most cheaply.

4.2 Simple bounds

As we mentioned in the introduction, problems frequently involve "simple bounds" of the form $\rho\,(\min(0, x_i - l_i) + \max(0, x_i - u_i))$. Although these might be exploited at various stages of the linear algebra (most particularly when updating the Schur complements as simple bound constituents enter and leave the working set) we have contented ourselves to exploit them only at the start of a major iteration. Specifically, since each system to be solved throughout the progress of the major iteration involves one or more solutions of the reference system

$$\begin{pmatrix} M_k & A_k^T \\ A_k & 0 \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} g \\ c \end{pmatrix} \tag{4.1}$$

for suitable g and c, it pays to exploit the structure of this system. So suppose (by implicitly reordering if necessary) that

$$A_k = \begin{pmatrix} A_k^{\rm FX} & A_k^{\rm FR} \\ I & 0 \end{pmatrix} \quad \text{and} \quad M_k = \begin{pmatrix} M_k^{\rm FX} & M_k^{\rm OD\,T} \\ M_k^{\rm OD} & M_k^{\rm FR} \end{pmatrix}$$

as well as

$$s = \begin{pmatrix} s^{\rm FX} \\ s^{\rm FR} \end{pmatrix}, \quad t = \begin{pmatrix} t^{\rm GC} \\ t^{\rm SB} \end{pmatrix}, \quad g = \begin{pmatrix} g^{\rm FX} \\ g^{\rm FR} \end{pmatrix} \quad \text{and} \quad c = \begin{pmatrix} c^{\rm GC} \\ c^{\rm SB} \end{pmatrix}$$

Then clearly (4.1) may be written as

$$\begin{pmatrix} M_k^{\rm FX} & M_k^{\rm OD\,T} & A_k^{\rm FX\,T} & I \\ M_k^{\rm OD} & M_k^{\rm FR} & A_k^{\rm FR\,T} & 0 \\ A_k^{\rm FX} & A_k^{\rm FR} & 0 & 0 \\ I & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} s^{\rm FX} \\ s^{\rm FR} \\ t^{\rm GC} \\ t^{\rm SB} \end{pmatrix} = \begin{pmatrix} g^{\rm FX} \\ g^{\rm FR} \\ c^{\rm GC} \\ c^{\rm SB} \end{pmatrix},$$

from which we may deduce that

$$s^{\rm FX} = g^{\rm FX}, \tag{4.2}$$

$$\begin{pmatrix} M_k^{\rm FR} & A_k^{\rm FR\,T} \\ A_k^{\rm FR} & 0 \end{pmatrix} \begin{pmatrix} s^{\rm FR} \\ t^{\rm GC} \end{pmatrix} = \begin{pmatrix} h^{\rm FR} \\ d^{\rm GC} \end{pmatrix} \stackrel{\rm def}{=} \begin{pmatrix} g^{\rm FR} - M_k^{\rm OD} g^{\rm FX} \\ c^{\rm GC} - A_k^{\rm FX} g^{\rm FX} \end{pmatrix}, \quad \text{and} \tag{4.3}$$

$$t^{\rm SB} = c^{\rm SB} - M_k^{\rm FX} s^{\rm FX} - M_k^{\rm OD\,T} s^{\rm FR} - A_k^{\rm FX\,T} t^{\rm GC}. \tag{4.4}$$

Thus rather than obtain factors of K_k, we need only find those of

$$K_k^{\rm FR} = \begin{pmatrix} M_k^{\rm FR} & A_k^{\rm FR\,T} \\ A_k^{\rm FR} & 0 \end{pmatrix}$$

and solve (4.1) via (4.2)–(4.4).

4.3 Solving the reduced reference system

The *reduced* reference system (4.3) may most obviously be solved using a direct factorization of $K_k^{\rm FR}$. However, if $M_k^{\rm FR}$ is non-singular and diagonal, it is often more efficient to find

$$s^{\rm FR} = -M_k^{\rm FR\,-1}(h^{\rm FR} - A_k^{\rm FX\,T} t^{\rm GC}), \quad \text{where} \quad A_k^{\rm FR} M_k^{\rm FR\,-1} A_k^{\rm FR\,T} t^{\rm GC} = A_k^{\rm FR} M_k^{\rm FR\,-1} h^{\rm FR} - d^{\rm GC} \tag{4.5}$$

using the factors of $A_k^{\rm FR} M_k^{\rm FR\ -1} A_k^{\rm FR\ T}$, particularly when $A_k^{\rm FR}$ has few nonzeros per column. Our default strategy is to use the alternative (4.5) whenever $M_k^{\rm FR}$ is both diagonal and positive definite and $A_k^{\rm FR}$ has less than a user-supplied number (default, 35) nonzeros per column.

Whichever approach is attempted, the HSL solver `MA27` (see, Duff and Reid, 1983) is used to factorize the relevant matrix and solve related systems. Perhaps unusually, a very small threshold tolerance is used (default, $0.1\sqrt{\epsilon_M}$), since, in our experience, this almost always results in far sparser factors, without any perceivable loss in accuracy, than with `MA27`'s default. As a precaution, the residuals of linear systems are periodically monitored, and any noticeable inaccuracies are handled by one or more steps of iterative refinement or, as a last resort, re-factorization with an increased pivot tolerance.

4.4 Preconditioners

Of course the choice of preconditioner M_k is vital. We provide a number of choices for M_k. The three "obvious" options, ranging in sophistication (and effectiveness), are to pick M_k to be H or I or a matrix made up of entries of H lying within a band of user-specified semi-bandwidth (default, 5). In addition, since we aim to use (4.2)–(4.4) rather than (4.1), we may sometimes prefer to set $M_k^{\rm FX}$ and $M_k^{\rm OD}$ to zero. In all cases (except when $M_k = I$), we have to be careful that K_k is standard. Thus, if the given $M_k^{\rm FR}$ is not second-order sufficient (this information is available after the attempted factorization of K_k), we use the simple expedient of adding $\|M_k^{\rm FR}\|_1 I$ to $M_k^{\rm FR}$, and re-factorizing. We accept that this is a rather simplistic strategy—indeed, we only really need to boost $M_k^{\rm FR}$ in the null-space of $A_k^{\rm FR}$—but it appears to be effective in practice.

The default preconditioner is chosen automatically, at the start of each major iteration, from the above, using the following heuristic. Firstly, an attempt to use $M_k = H$ is made, but this is abandoned, unless H_k is itself diagonal, if the number of nonzeros in the factors of K_k exceeds a given multiple (default, 10) of those in the K_k itself. Next, if this failure occurs, the banded approximation described above is attempted. If the banded approximation fails because there is insufficient room, the required semi-bandwidth is set to zero, and a diagonal approximation attempted. Finally, if all of the preceding fail, the identity matrix is chosen. Although this might appear *ad hoc*, such an automatic strategy has worked reasonably well in practice.

4.5 The search direction and anti-zigzagging

Our current anti-zigzagging procedure is extremely crude. The initial approximation to the solution of the search-direction problem (2.1) is a low-accuracy solution based on a few (often simply one) iterations of the GLTR method mentioned in Section 2.1. If this solution is not optimal for (2.1), and if the active set has not changed in the interim, a far more accurate solution of (2.1) is sought, typically only terminating when the "residual" $\langle g_{j,k}, v_{j,k} \rangle \leq \max(\epsilon_R \cdot \langle g_{0,k}, v_{0,k} \rangle, \epsilon_A)$ for some user-supplied relative and absolute convergence tolerances ϵ_R (default, zero) and ϵ_A (default, $\sqrt{\epsilon_M}$). We plan to investigate more sophisticated schemes in due course.

4.6 Linear independence

The whole algorithm is predicated on the constituents in the working set having linearly independent gradients. This may fail in practice for two reasons. Firstly, the initial working set provided may have dependent constituent gradients. Secondly, barely independent constituents may be

picked up as the algorithm proceeds. We thus find it periodically necessary to try to identify (and consequently) remove such rogue constituents. Our strategy is simply to factorize

$$\begin{pmatrix} I & A_k^{\mathrm{FR}\,T} \\ A_k^{\mathrm{FR}} & 0 \end{pmatrix}$$

(which may or may not be reused later as the preconditioner), using `MA27`, with a large threshold tolerance (default, 0.5), and to use relatively small diagonal entries in its block diagonal factor to predict dependent constituents—any eigenvalue of this factor smaller than a fixed factor (default, $\epsilon^{0.75}$) of the largest in absolute value is considered dependent. We apply this "pruning" strategy at the start of the algorithm, and at the start of each major iteration for which there is some suspicion that the previous one might have introduced "close to dependent" terms—this is usually apparent when factors of the Schur complement S_k become ill-conditioned.

While our strategy is not foolproof, and, admittedly, may be expensive, we believe that it is a prudent precaution that has proved most worthwhile in practice. Of course, more sophisticated strategies (such as those involving the singular value decomposition or a rank-revealing factorization) are more reliable, but they are almost always too expensive for the size of our application.

4.7 Feasibility tolerance

In practice, constituents are considered active if $|\langle a_i, x\rangle - b_i| \leq \epsilon_a$, for some user-supplied tolerance $\epsilon_a > 0$. (default, $\epsilon_M^{.75}$, where where ϵ_M is the relative machine precision).

4.8 Handling degeneracy

While there are many possible anti-cycling rules (see, for example, Chvátal, 1983, Chapter 3), we have chosen to avoid the issue altogether by randomly perturbing the data b before starting to solve the problem. Once optimality is achieved, the perturbations are gradually reduced and the problem resolved, until the perturbations have effectively vanished. Specifically, positive initial random perturbations in the range $(0, \sqrt{\epsilon_M} \cdot \max(1, \|b\|)]$ are added to b; on termination, they are reduced by a factor $0.1 \cdot \min(1, \epsilon_a/\sqrt{\epsilon_M})$, and the problem resolved, until they are smaller than $10\epsilon_M$.

4.9 Cold and warm starts

Options are provided for the user to specify which constituents are to be initially in the working set (usually known as a warm start), or for the initial point itself to be given, or for cold starts with initial working sets made up with either no active constituents, or as many active constituents as possible, or only "equality" constituents. The gradients of the active constituents at the initial point are always checked for independence, and some dependent constituents may be removed from the initial working set.

4.10 The penalty parameter

When solving QPs, the initial penalty parameter ρ is supplied by the user (default, 10). If the constraints are violated at the solution to (1.1), or if $f(x)$ has been diagnosed as being unbounded

from below, ρ is increased by a factor of user-supplied factor (default, two), and (1.1) resolved. In addition, ρ is increased by the same factor whenever the relative infeasibility fails to decrease by at least a given factor (default, 0.75) over a prescribed period (default, 100) of iterations. While this strategy is rather naive, it has worked satisfactorily. We also note that, although finding a good value of ρ can help reduce the number of iterations performed, it does not seem to be as critical for reliability as we had been lead to expect.

5 Numerical results

Preliminary numerical results obtained using the method described here are given in a companion paper (Gould and Toint, 2001*a*), in which our present proposal is compared with a competing primal-dual interior-point trust-region approach (`QPB` in GALAHAD, or `HSL_VE12` in HSL, 2000, see Conn, Gould, Orban and Toint, 2000). For brevity, we do not propose to repeat these here, but direct the reader to Gould and Toint (2001*a*) for details. Moreover, since `QPA` will eventually incorporate our quadratic programming preprocessing procedures (see, Gould and Toint, 2001*b*), we feel it is wise to report on the complete code when it is ultimately released.

We should stress that the conclusions drawn in Gould and Toint (2001*a*) are not particularly favourable for our working-set approach, since it is comprehensively outperformed for large-scale cold-started applications by its interior-point rival. Even when a good prediction of the optimal working set is available (a warm start), the working-set method described here does not always beat the interior-point approach, simply because the latter requires so few iterations, while the former can easily be fooled by just one incorrect assignment to the working set and most especially by degeneracy. Having said this, for small problems, and in some warm-started cases, the working-set approach does appear to perform better than its rival, and thus our contention that it is advantageous to have both methods available appears still to be valid.

6 Conclusions

We have presented a working-set based quadratic programming method capable of finding (at least) first-order critical points in the non-convex case. The method is designed to solve large-scale problems, and uses a suitably-preconditioned conjugate-gradient iteration at its heart. Methods for updating the preconditioner, while maintaining crucial inertial properties are described, and a large number of implementational details are provided. The method has been implemented, and versions will shortly be available as part of both the GALAHAD and HSL (2001) suites of software packages.

References

C. Ashcraft, R. G. Grimes, and J. G. Lewis. Accurate symmetric indefinite linear equation solvers. *SIAM Journal on Scientific Computing*, **20**(2), 513–561, 1998.

R. Benveniste. A quadratic programming algorithm using conjugate search directions. *Mathematical Programming*, **16**(1), 63–80, 1979.

D. P. Bertsekas. Projected Newton methods for optimization problems with simple constraints. *SIAM Journal on Control and Optimization*, **20**(2), 221–246, 1982.

M. J. Best and K. Ritter. An effective algorithm for quadratic minimization problems. Technical report 1691, University of Wisconsin, Madison, Wisconsin, U.S.A., 1976.

J. Bisschop and A. Meeraus. Matrix augmentation and partitioning in the updating of the basis inverse. *Mathematical Programming*, **13**(3), 241–254, 1977.

J. R. Bunch and L. C. Kaufman. Some stable methods for calculating inertia and solving symmetric linear equations. *Mathematics of Computation*, **31**, 163–179, 1977.

J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, **8**(4), 639–655, 1971.

P. H. Calamai and J. J. Moré. Projected gradient methods for linearly constrained problems. *Mathematical Programming*, **39**, 93–116, 1987.

Y. Chabrillac and J.-P. Crouzeix. Definiteness and semidefiniteness of quadratic forms revisited. *Linear Algebra and its Applications*, **63**, 283–292, 1984.

V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York and San Francisco, 1983.

T. F. Coleman. Linearly constrained optimization and projected preconditioned conjugate gradients. *In* J. Lewis, ed., 'Proceedings of the Fifth SIAM Conference on Applied Linear Algebra', pp. 118–122, SIAM, Philadelphia, USA, 1994.

A. R. Conn and J. W. Sinclair. Quadratic programming via a non-differentiable penalty function. Technical Report CORR 75/15, Faculty of Mathematics, University of Waterloo, 1975.

A. R. Conn, N. I. M. Gould, D. Orban, and Ph. L. Toint. A primal-dual trust-region algorithm for non-convex nonlinear programming. *Mathematical Programming*, **87**(2), 215–249, 2000.

I. S. Duff. MA57 - a new code for the solution of sparse symmetric indefinite systems. Technical Report (in preparation), Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2001.

I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, **9**(3), 302–325, 1983.

I. S. Duff, J. K. Reid, N. Munksgaard, and H. B. Neilsen. Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite. *Journal of the Institute of Mathematics and its Applications*, **23**, 235–250, 1979.

R. Fletcher. Factorizing symmetric indefinite matrices. *Linear Algebra and its Applications*, **14**, 257–272, 1976.

R. Fletcher. A model algorithm for composite nondifferentiable optimization problems. *Mathematical Programming Studies*, **17**, 67–76, 1982.

R. Fletcher. *Practical Methods of Optimization.* J. Wiley and Sons, Chichester, England, second edn, 1987.

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

P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, London, 1981.

P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A Schur-complement method for sparse quadratic programming. *In* M. G. Cox and S. J. Hammarling, eds, 'Reliable Scientific Computation', pp. 113–138, Oxford University Press, Oxford, England, 1990.

P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Inertia-controlling methods for general quadratic programming. *SIAM Review*, **33**(1), 1–36, 1991.

N. I. M. Gould. On practical conditions for the existence and uniqueness of solutions to the general equality quadratic-programming problem. *Mathematical Programming*, **32**(1), 90–99, 1985.

N. I. M. Gould and Ph. L. Toint. Numerical methods for large-scale non-convex quadratic programming. Technical Report RAL-TR-2001-017, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2001*a*.

N. I. M. Gould and Ph. L. Toint. Preprocessing for quadratic programming. Technical Report in preparation, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2001*b*.

N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic problems arising in optimization. Technical Report RAL-TR-98-069, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 1998.

N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM Journal on Optimization*, **9**(2), 504–525, 1999.

S. P. Han. Solving quadratic programs with an exact penalty function. *In* O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds, 'Nonlinear Programming, 4', pp. 25–55, Academic Press, London and New York, 1981.

M. R. Hestenes. *Conjugate Direction Methods in Optimization.* Springer Verlag, Heidelberg, Berlin, New York, 1980.

M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, **49**, 409–436, 1952.

HSL. A collection of Fortran codes for large-scale scientific computation, 2000. See http://www.numerical.rl.ac.uk/hsl.

HSL. A collection of Fortran codes for large-scale scientific computation, 2001. (Forthcoming).

B. T. Polyak. The conjugate gradient method in extremal problems. *U.S.S.R. Computational Mathematics and Mathematical Physics*, **9**, 94–112, 1969.

S. M. Robinson. Perturbed Kuhn-Tucker points and rates of convergence for a class of nonlinear programming algorithms. *Mathematical Programming*, **7**(1), 1–16, 1974.

D. C. Sorensen. Updating the symmetric indefinite factorization with applications in a modified Newton method. Technical Report ANL-77-49, Argonne National Laboratory, Illinois, USA, 1977.

T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, **20**(3), 626–637, 1983.

Ph. L. Toint. Global convergence of a class of trust region methods for nonconvex minimization in Hilbert space. *IMA Journal of Numerical Analysis*, **8**(2), 231–252, 1988.

J. W. J. Williams. Algorithm 232, Heapsort. *Communications of the ACM*, **7**, 347–348, 1964.