Comput Optim Appl
DOI 10.1007/s10589-016-9886-1

A dual gradient-projection method for large-scale strictly convex quadratic problems

Nicholas I. M. Gould[1] · **Daniel P. Robinson**[2]

Received: 18 February 2016

Abstract The details of a solver for minimizing a strictly convex quadratic objective function subject to general linear constraints are presented. The method uses a gradient projection algorithm enhanced with subspace acceleration to solve the bound-constrained dual optimization problem. Such gradient projection methods are well-known, but are typically employed to solve the primal problem when only simple bound-constraints are present. The main contributions of this work are threefold. First, we address the challenges associated with solving the dual problem, which is usually a convex problem even when the primal problem is strictly convex. In particular, for the dual problem, one must efficiently compute directions of infinite descent when they exist, which is precisely when the primal formulation is infeasible. Second, we show how the linear algebra may be arranged to take computational advantage of *sparsity* that is often present in the second-derivative matrix, mostly by showing how sparse updates may be performed for algorithmic quantities. We consider the case that the second-derivative matrix is explicitly available and sparse, and the case when it is available implicitly via a limited memory BFGS representation. Third, we present the details of our Fortran 2003 software package DQP, which is part of the GALAHAD

Electronic supplementary material The online version of this article (doi:10.1007/s10589-016-9886-1) contains supplementary material, which is available to authorized users.

✉ Daniel P. Robinson
daniel.p.robinson@gmail.com

Nicholas I. M. Gould
nick.gould@stfc.ac.uk

[1] Scientific Computing Department, STFC-Rutherford Appleton Laboratory, Chilton, Oxfordshire OX11 0QX, England, UK

[2] Department of Applied Mathematics and Statistics, The Johns Hopkins University, Baltimore, MD 21218, USA

Published online: 01 December 2016

suite of optimization routines. Numerical tests are performed on quadratic programming problems from the combined CUTEst and Maros and Meszaros test sets.

Keywords Convex optimization · Quadratic programming · Gradient projection · Large-scale optimization · Sparse factorizations · Dual method

1 Introduction

Quadratic problems (QPs) occur naturally in many application areas such as discrete-time stabilization, economic dispatch, finite impulse response design, optimal and fuzzy control, optimal power flow, portfolio analysis, structural analysis, support vector machines and VLSI design [40]. They are also a vital component of so-called recursive/successive/sequential quadratic programming (SQP) methods for nonlinear optimization, in which the general nonlinear problem is tackled by solving a sequence of suitable approximating QPs [9,41]. Traditionally, QPs have been solved by either active-set or interior-point methods [65, Ch. 6], but in certain cases gradient-projection methods, which are the focus of this manuscript, may be preferred.

We consider the strictly convex QP

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\; q(x) := \tfrac{1}{2}\langle x, Hx\rangle + \langle g, x\rangle \;\text{ subject to }\; Ax \geq c, \tag{1}$$

where $\langle\cdot,\cdot\rangle$ is the Euclidean inner product, the symmetric matrix $H \in \mathbb{R}^{n\times n}$ is positive definite, and $A \in \mathbb{R}^{m\times n}$ is the constraint matrix for some positive integers m and n. We are especially interested in the case when n is large and the structure of A and H may be exploited, e.g., when A and H are sparse [40] or the inverse of H is represented in the form

$$H^{-1} = H_0^{-1} + VWV^T \tag{2}$$

with the matrix $H_0^{-1} \in \mathbb{R}^{n\times n}$ sparse and positive definite (often diagonal), the matrices $V \in \mathbb{R}^{n\times t}$ and $W \in \mathbb{R}^{t\times t}$ dense with W symmetric and non-singular, and t a small even integer typically in the range 0–20 [11]. Matrices of the form (2) often arise as limited-memory secant Hessian approximations in SQP methods [44–48,61,76].

It is well known [21] that (1) has a related primal-dual problem

$$\underset{(x,y)\in\mathbb{R}^{n+m}}{\text{maximize}}\; -\tfrac{1}{2}\langle x, Hx\rangle + \langle c, y\rangle \;\text{ subject to }\; Hx - A^T y + g = 0,\quad y \geq 0. \tag{3}$$

The optimal objective values of (1) and (3) are equal. After eliminating x from (3), an optimal dual vector y_* of (3) is a solution to

$$y_* = \underset{y\in\mathbb{R}^m}{\arg\min}\, q^{\mathrm{D}}(y) := \tfrac{1}{2}\langle y, H^{\mathrm{D}} y\rangle + \langle g^{\mathrm{D}}, y\rangle \;\text{ subject to }\; y \geq 0, \tag{4}$$

which is the dual problem to (1) with

$$H^{\mathrm{D}} := AH^{-1}A^T \quad \text{and} \quad g^{\mathrm{D}} := -AH^{-1}g - c. \tag{5}$$

Once y_* is computed, the optimal x_* may be recovered by solving $Hx_* = A^T y_* - g$. The advantage of this dual formulation is that the geometry of the feasible region is simpler—finding a feasible point is trivial—but at the expense of a more complicated and likely dense Hessian H^{D}. The method that we consider does not require H^{-1} but rather that we can find $u = H^{-1}w$ by solving $Hu = w$ or by other means. Note that (1) is infeasible if and only if (4) is unbounded; the latter is possible since H^{D} may only be positive semidefinite.

1.1 Prior work and our contributions

A great number of algorithms have been proposed to solve the general QP (1). They may be classified as either interior-point algorithms, active-set methods, penalty methods, or projected gradient methods, all of which we summarize in turn.

Interior-point methods typically are primal-dual [20,49,56,60,74] in the sense that they solve a linear system during each iteration that is obtained from applying Newton's Method for zero finding to a certain primal-dual nonlinear equation based on a shifted complementary condition. For problems that are convex but perhaps not strictly convex, regularized formulations have been proposed [30] to address the issues associated with potentially solving singular linear systems, or systems that are ill-conditioned. Primal-dual interior-point methods are revered for requiring a small number of iterations in practice, but are difficult to be warm-started, i.e., to solve (1) much more efficiently when given a good estimate of a solution as compared to an arbitrary starting point.

The class of active-set methods complements the set of interior-point solvers. They typically are robust and benefit from warm-starts, but are less scalable than interior-point methods. Active-set methods solve a sequence of equality-constrained quadratic subproblems (equivalently, they solve a sequence of linear systems of equations) where the constraints are defined using the current active set (more accurately, using a working set). The diminished capacity for active-set methods to scale is rooted in the fact that the active-set estimates typically change by a single element during each iteration. Thus, it is possible that an exponential number (exponential in the number of constraints) of subproblems may need to be solved to obtain a solution to the original problem (1), which is potentially problematic when the number of constraints is large. Primal feasible active-set methods [5,34,35,43,49,56] require a primal-feasible point that is often costly to obtain, whereas a trivial feasible point exists for dual feasible active-set methods [3,10,36,67,71]. We also mention that non-traditional active-set methods have recently been developed primarily motivated by solving certain problems that arise in optimal control [17,18,52], and that dual active-set methods have also received attention [27,51].

The class of penalty methods may, in some sense, be considered as a compromise between interior-point and active-set methods. Penalty methods solve the

original generally-constrained QP by solving a sequence of QP problems with simpler constraint sets. For example, the method by Spellucci [72] uses a penalty approach to formulate a new primal-dual quadratic objective function with simple bound-constraints whose first-order solutions correspond to solutions of the original generally-constrained problem (1). Although this is a reasonable approach, it typically does not perform well on moderately ill-conditioned problems because the reformulated penalty problem has a condition number that is the cube of the condition number of the original problem. A more standard augmented Lagrangian penalty approach is used by Friedlander and Leyffer [29]. They improve upon standard augmented Lagrangian methods for general nonlinear optimization by using the structure of the QP to more efficiently update problem parameters. The augmented Lagrangian approach means that each subproblem is cheaper to solve, but to ensure convergence, a penalty parameter and an estimate of a Lagrange multiplier vector must be iteratively updated. In our experience, such penalty methods typically warm-start better than interior-point methods but not as well as traditional active-set methods, and scale better than active-set methods, but not as well as interior-point methods. In this sense, one may view them as a compromise between interior-point and active-set methods.

The final class of QP methods, which consists of gradient projection methods, is well-known and typically employed to solve the primal problem when only simple bound-constraints are present [22,23,62–64,69]. When applied to bound-constrained problems, such methods have a modest cost per iteration and usually perform well when the problems are well conditioned or moderately ill-conditioned. However, they tend to perform poorly on ill-conditioned problems. Projection methods are less commonly used to solve dual formulations. One such example is the primal-dual projection method used to solve linear-QPs that arise in dynamic and stochastic programming [77]. In this setting, however, both the primal and dual problems have nonsmooth second derivatives. A second example is the dual projection method [2], which focuses on the special structure of the QPs used to solve mixed-integer problems in predictive control. The projection method in [66] is perhaps the most similar to ours. The authors present a gradient projection method called DPG for solving the dual problem, and an accelerated version called DAPG. The formulation of their method is motivated by obtaining an optimal worst-case complexity result, and has a modest cost per iteration of $O(mn)$. They prove that the convergence rate for DPG is $O(1/k)$, and that the convergence rate for DAPG is $O(1/k^2)$, where k is the iteration number. Both of these methods require a single projected gradient iteration on the dual problem during each iteration. Whereas their method was designed to obtain an optimal complexity result, the method that we describe is designed to be efficient in practice. For example, whereas DPG and DAPG perform a single gradient projection computation, we allow for either a search along the projected gradient path or a simple projected gradient backtracking search. An additional enhancement to our method is subspace acceleration computations that are routinely used in gradient projection methods. These calculations improve upon a vanilla fixed step length gradient projection method, which is known to have a convergence rate of $O(1/k)$ in terms of the optimal objective value, so that the methods discussed in this paper inherit the same rate.

The contributions of this paper can be summarized as follows. (i) We address the challenges faced by gradient projection methods in the context of solving the dual QP problem, which is typically a convex (not strictly convex) problem even when the primal problem is strictly convex. In particular, for the dual problem, one must efficiently compute directions of infinite descent when they exist, which is precisely when the primal formulation is infeasible. (ii) We show how the linear algebra may be arranged to take computational advantage of structure that is often present in the second-derivative matrix. In particular, we consider the case that the second-derivative matrix is explicitly available and sparse, and the case when it is available implicitly via a limited memory BFGS representation. (iii) We present the details of our Fortran 2003 software package DQP, which is part of the GALAHAD suite of optimization routines. Numerical tests are performed on quadratic programming problems from the combined CUTEst and Maros and Meszaros test sets.

1.2 Notation

We let I be the appropriately-dimensioned identity matrix and e_j its jth column. The vector e will be the appropriately-dimensioned vector of ones. The standard inner product of two vectors x and y in $\mathbb{R}^m$ is denoted by $\langle x, y\rangle$, with the induced vector (Euclidean) norm being $\|x\| := \sqrt{\langle x, x\rangle}$. For any index set $\mathcal{S} \subseteq \{1, 2, \ldots, m\}$, we defined $\langle x, y\rangle_{\mathcal{S}} := \sum_{j\in\mathcal{S}} x_j y_j$ with x_j denoting the jth entry in x, i.e., $\langle x, y\rangle_{\mathcal{S}}$ is the standard inner produce over the sub-vectors of x and y that correspond to the index set $\mathcal{S}$. Throughout, $H \in \mathbb{R}^{n\times n}$ is positive definite and $A \in \mathbb{R}^{m\times n}$ is the constraint matrix. The quantity $\max(y, 0)$ is the vector whose ith component is the maximum of y_j and 0; a similar definition is used for $\min(y, 0)$.

2 The method

The method we use to solve the dual bound-constrained quadratic program (BQP) (4) is fairly standard [12,15,62] and makes heavy use of projections onto the dual feasible set. Thus, we let

$$P_{\mathcal{D}}[y] := \max(y, 0)$$

be the projection of y onto the dual feasible set

$$\mathcal{D} := \{y \in \mathbb{R}^m : y \geq 0\}$$

associated with (4). The well-studied gradient projection method for BQP is given by Algorithm 1.

Computing only the Cauchy point y^{C} during each iteration is sufficient to guarantee convergence and, under suitable assumptions, to identify the set of variables that are zero at the solution [14,58,62]. Subsequently, the subspace step Δy^{S} will identify the solution, although empirically it also accelerates convergence before the optimal active set is determined. The use of $\alpha_{\max}$ allows for additional flexibility although the

Algorithm 1 Accelerated gradient projection method for solving the BQP (4).

input: Solution estimate $y \in \mathcal{D}$.
Choose $\varepsilon > 0$.
while $\|P_{\mathcal{D}}[y - \nabla q^{\mathrm{D}}(y)] - y\| > \varepsilon$ **do**
1. (Cauchy point)
Set $d = -\nabla q^{\mathrm{D}}(y)$ and compute $\alpha^{\mathrm{C}} = \operatorname{argmin}_{\alpha>0} q^{\mathrm{D}}\big(P_{\mathcal{D}}[y + \alpha d]\big)$.
Set $y^{\mathrm{C}} = P_{\mathcal{D}}[y + \alpha^{\mathrm{C}} d]$.
2. (subspace step)
Compute $\mathcal{A} = \mathcal{A}(y^{\mathrm{C}}) = \{i : y_i^{\mathrm{C}} = 0\}$.
Compute $\Delta y^{\mathrm{S}} = \operatorname{argmin}_{\Delta y \in \mathbb{R}^m} q^{\mathrm{D}}(y^{\mathrm{C}} + \Delta y)$ subject to $[\Delta y]_{\mathcal{A}} = 0$.
3. (improved subspace point)
Select $\alpha_{\max} > 0$ and then compute $\alpha^{\mathrm{S}} = \operatorname{argmin}_{\alpha \in [0, \alpha_{\max}]} q^{\mathrm{D}}\Big(P_{\mathcal{D}}[y^{\mathrm{C}} + \alpha \Delta y^{\mathrm{S}}]\Big)$.
Set $y = P_{\mathcal{D}}[y^{\mathrm{C}} + \alpha^{\mathrm{S}} \Delta y^{\mathrm{S}}]$.
end while

choice $\alpha_{\max} = \infty$ would be common. Another choice would be to set $\alpha_{\max}$ to the largest α value satisfying $(y^{\mathrm{C}} + \alpha \Delta y^{\mathrm{S}}) \in \mathcal{D}$ so that α^{S} can easily be computed in closed form.

The Cauchy point and improved subspace point computations both require that we compute a minimizer of the convex dual objection function along a piecewise linear arc. Of course, the exact minimizer suffices, but there may be a computational advantage in accepting suitable approximations. We consider both possibilities in turn.

2.1 An exact Cauchy point in Algorithm 1

The Cauchy point may be calculated by stepping along the piecewise linear arc $P_{\mathcal{D}}[y + \alpha d]$ while considering the objective function on each linear segment [15, §3]. The entire behavior can be predicted while moving from one segment to the next by evaluating and using the product $H^{\mathrm{D}} p$, where p is a vector whose non-zeros occur only in positions corresponding to components of y that become zero as the segment ends; thus, p usually has a single nonzero entry. The required product can be obtained as follows:

$$H^{\mathrm{D}} p = Au, \text{ where } Hu = w \text{ and } w = A^T p.$$

Note that w is formed as a linear combination of the columns of A^T indexed by the non-zeros in p.

The details of the search along the piecewise linear path is given as Algorithm 2, which is based on [16, Alg. 17.3.1 with typo corrections]. The segments are defined by "breakpoints", and the first and second derivatives of q^{D} on the arc at the start of the ith segment are denoted by q_i' and q_i'', respectively.

Algorithm 2 seems to need a sequence of products with A and A^T and solves with H to form the products with H^{D}, but fortunately, we may simplify the process considerably. We consider two cases.

Algorithm 2 Finding the Cauchy point (preliminary version).

input: Solution estimate $y \in \mathcal{D}$ and search direction d.
Compute the gradient $\nabla g^{\mathrm{D}}(y) = -AH^{-1}(g - A^T y) - c$ and Hessian $H^{\mathrm{D}} = AH^{-1}A^T$.
Compute the vector of breakpoints

$$\alpha_j^{\mathrm{B}} = \begin{cases} -y_j/d_j & \text{if } d_j < 0 \\ \infty & \text{if } d_j \geq 0 \end{cases} \quad \text{for all } j = 1, \ldots n.$$

Compute the index sets $\mathcal{I}^0 = \{j : \alpha_j^{\mathrm{B}} = 0\}$ and $\mathcal{A}^0 = \mathcal{I} \cup \{j : y_j = 0 \text{ and } d_j = 0\}$.
Set $\alpha^0 = 0$, $e^0 = \sum_{j \in \mathcal{I}^0} d_j e_j$, $d^0 = d - e^0$, and $s^0 = 0$.
Compute $\ell^0 = \langle \nabla g^{\mathrm{D}}(y), d^0 \rangle$ and $H^{\mathrm{D}} d^0$.
Set $q_0' = \ell^0$ and compute $q_0'' = \langle d^0, H^{\mathrm{D}} d^0 \rangle$.
for $i = 0, 1, 2, \ldots$ **do**

1. (find the next breakpoint)
Determine the next breakpoint α^{i+1}, and then set $\Delta\alpha^i = \alpha^{i+1} - \alpha^i$.

2. (check the current interval for the Cauchy point)
if $q_i' \geq 0$ **then** return the exact Cauchy point $y^{\mathrm{C}} = y + s^i$. **end**
if $q_i'' > 0$ **then** set $\Delta\alpha = -q_i'/q_i''$ **else** set $\Delta\alpha = \infty$. **end**
if $q_i'' > 0$ and $\Delta\alpha < \Delta\alpha^i$ **then** return the Cauchy point $y^{\mathrm{C}} = y + s^i + \Delta\alpha d^i$. **end**

3. (prepare for the next interval)
Compute the index sets $\mathcal{I}^{i+1} = \{j : \alpha_j^{\mathrm{B}} = \alpha^{i+1}\}$ and $\mathcal{A}^{i+1} = \mathcal{A}^i \cup \mathcal{I}^{i+1}$.
Compute $e^{i+1} = \sum_{j \in \mathcal{I}^{i+1}} d_j e_j$, $s^{i+1} = s^i + \Delta\alpha^i d^i$, and $d^{i+1} = d^i - e^{i+1}$.
Compute $H^{\mathrm{D}} e^{i+1}$ and $\langle \nabla g^{\mathrm{D}}(y), e^{i+1} \rangle$.
Update $\ell^{i+1} = \ell^i - \langle \nabla g^{\mathrm{D}}(y), e^{i+1} \rangle$ and $H^{\mathrm{D}} d^{i+1} = H^{\mathrm{D}} d^i - H^{\mathrm{D}} e^{i+1}$.

4. (compute the slope and curvature for the next interval)
Use $H^{\mathrm{D}} d^{i+1}$ to compute $\gamma^{i+1} = \langle s^{i+1}, H^{\mathrm{D}} d^{i+1} \rangle$ and $q_{i+1}'' = \langle d^{i+1}, H^{\mathrm{D}} d^{i+1} \rangle$.
Compute $q_{i+1}' = \ell^{i+1} + \gamma^{i+1}$.

end for

2.1.1 Sparse H

Suppose that H is sparse and that we have the (sparse) Cholesky factorization

$$H = LL^T, \tag{6}$$

where L is a suitable row permutation of a lower triangular matrix. By defining $x := -H^{-1}(g - A^T y)$, $h^i := L^{-1} A^T d^i$, and $p^i := L^{-1} A^T s^i$, and rearranging the inner products, we obtain a simplified method (Algorithm 3), in which products with A and back-solves with L^T are avoided in the main loop.

Notice that the product $r^i = A^T e^i$ is with a sparse vector, and that each row of A is accessed at most a single time during the entire solve. Advantage may also be taken of sparsity in r^i when performing the forward solve $Lw^i = r^i$ since in many cases w^i will be sparse when L is very sparse. This depends, of course, on the specific sparse factorization used, and in particular nested-dissection ordering favors sparse forward solutions [31]. New subroutines to provide this (sparse right-hand side) functionality have been added to each of the HSL packages `MA57` [24], `MA87` [53] and `MA97` [55]. There may also be some gain on high-performance machines in performing a block solve

Algorithm 3 Finding the Cauchy point (preliminary sparse version).

input: Solution estimate $y \in \mathcal{D}$ and search direction d.
Solve $LL^T x = A^T y - g$ and then set $\nabla g^{\mathrm{D}}(y) = Ax - c$.
Compute the vector of breakpoints

$$\alpha_j^{\mathrm{B}} = \begin{cases} -y_j/d_j & \text{if } d_j < 0, \\ \infty & \text{if } d_j \geq 0 \end{cases} \quad \text{for all } j = 1, \ldots n.$$

Compute the index sets $\mathcal{I}^0 = \{j : \alpha_j^{\mathrm{B}} = 0\}$ and $\mathcal{A}^0 = \mathcal{I} \cup \{j : y_j = 0 \text{ and } d_j = 0\}$.
Set $\alpha^0 = 0$, $e^0 = \sum_{j \in \mathcal{I}^0} d_j e_j$, $d^0 = d - e^0$, and $p^0 = 0$.
Compute $\ell^0 = \langle \nabla g^{\mathrm{D}}(y), d^0 \rangle$ and $r^0 = A^T d^0$, and solve $Lh^0 = r^0$.
Set $q_0' = \ell^0$ and compute $q_0'' = \langle h^0, h^0 \rangle$.
for $i = 0, 1, 2, \ldots$ **do**

1. (find the next breakpoint)
Determine the next breakpoint α^{i+1}, and then set $\Delta\alpha^i = \alpha^{i+1} - \alpha^i$.

2. (check the current interval for the Cauchy point)
if $q_i'' > 0$ **then** set $\Delta\alpha := -q_i'/q_i''$ **else** set $\Delta\alpha = \infty$. **end**
if $q_i' \geq 0$, or $q_i'' > 0$ and $\Delta\alpha < \Delta\alpha^i$ **then** define s component-wise by

$$s_j = \begin{cases} \alpha^\ell d_j \text{ for } j \in \mathcal{I}^\ell \text{ for each } \ell = 0, \ldots, i, \\ \alpha^i d_j \text{ for } j \notin \cup_{l=0}^{i} \mathcal{I}^\ell. \end{cases}$$

end
if $q_i' \geq 0$ **then** return the exact Cauchy point $y^{\mathrm{C}} = y + s$. **end**
if $q_i'' > 0$ and $\Delta\alpha < \Delta\alpha^i$ **then** return the Cauchy point $y^{\mathrm{C}} = y + s + \Delta\alpha d^i$. **end**

3. (prepare for the next interval)
Compute $\mathcal{I}^{i+1} = \{j : \alpha_j^{\mathrm{B}} = \alpha^{i+1}\}$ and $\mathcal{A}^{i+1} = \mathcal{A}^i \cup \mathcal{I}^{i+1}$.
Compute $e^{i+1} = \sum_{j \in \mathcal{I}^{i+1}} d_j e_j$, $d^{i+1} = d^i - e^{i+1}$, and $\ell^{i+1} = \ell^i - \langle \nabla g^{\mathrm{D}}(y), e^{i+1} \rangle$.
Compute $r^{i+1} = A^T e^{i+1}$, and then solve $Lw^{i+1} = r^{i+1}$.
Update $p^{i+1} = p^i + \Delta\alpha^i h^i$ and $h^{i+1} = h^i - w^{i+1}$.

4. (compute the slope and curvature for the next interval)
Compute $\gamma^{i+1} = \langle p^{i+1}, h^{i+1} \rangle$, $q_{i+1}' = \ell^{i+1} + \gamma^{i+1}$, and $q_{i+1}'' = \langle h^{i+1}, h^{i+1} \rangle$.
end for

$$L\left(w^i \ldots w^{i+j}\right) = \left(r^i \ldots r^{i+j}\right)$$

in anticipation of future w^{i+j} for $j > 0$, as such solves may take advantage of machine cache. The breakpoints may be found, as required, very efficiently using a heapsort [75].

While this algorithm is an improvement on its predecessor, it may still be inefficient. In particular, the vectors $\{h^i\}$ will generally be dense, and this means that the computation of γ^{i+1} and q_{i+1}'', as well as the update for p^i, may each require $O(n)$ operations. As we have already noted, by contrast w^i is generally sparse, and so our aim must be to rearrange the computation in Algorithm 3 so that products and updates involve w^i rather than h^i.

Note that, using the recurrence $h^{i+1} = h^i - w^{i+1}$ allows us to write

$$\langle h^{i+1}, h^{i+1} \rangle = \langle h^i, h^i \rangle + \langle w^{i+1} - 2h^i, w^{i+1} \rangle,$$

while this and the recurrence $p^{i+1} = p^i + \Delta\alpha^i\, h^i$ give

$$\langle p^{i+1}, h^{i+1}\rangle = \langle p^i, h^i\rangle + \Delta\alpha^i \langle h^i, h^i\rangle - \langle p^{i+1}, w^{i+1}\rangle.$$

Combining this with the relationship $q_i'' = \langle h_i, h_i\rangle$ yields the recursions

$$\gamma^{i+1} = \gamma^i + \Delta\alpha^i q_i'' - \langle p^{i+1}, w^{i+1}\rangle \;\text{ and }\; q_{i+1}'' = q_i'' + \langle w^{i+1} - 2h^i, w^{i+1}\rangle, \quad (7)$$

where we only need the inner products over components j for which $w_j^{i+1} \neq 0$.

Now let

$$u^{i+1} = p^{i+1} - \alpha^{i+1} h^i, \;\text{ with } u^0 = 0. \quad (8)$$

Using $p^{i+1} = p^i + \Delta\alpha^i h^i$, $\alpha^{i+1} = \alpha^i + \Delta\alpha^i$, and $h^{i+1} = h^i - w^{i+1}$ we have

$$\begin{aligned} u^{i+1} &= p^i + \Delta\alpha^i\, h^i - \alpha^{i+1} h^i = p^i + \left(\Delta\alpha^i - \alpha^{i+1}\right) h^i \\ &= p^i - \alpha^i h^i = p^i - \alpha^i \left(h^{i-1} - w^i\right) = u^i + \alpha^i w^i. \end{aligned}$$

Thus, rather than recurring p^i, we may instead recur u^i and obtain $p^{i+1} = u^{i+1} + \alpha^{i+1} h^i$ from (8) as needed. The important difference is that the recursions for u^i and h^i only involve the likely-sparse vector w^i. Note that by substituting for p^{i+1}, the recurrence for γ^{i+1} in (7) becomes

$$\gamma^{i+1} = \gamma^i + \Delta\alpha^i q_i'' - \langle u^{i+1}, w^{i+1}\rangle - \alpha^{i+1} \langle h^i, w^{i+1}\rangle,$$

which only involves inner products with the likely-sparse vector w^{i+1}.

Rearranging the steps in Algorithm 3 using the above equivalent formulations gives our final method stated as Algorithm 4. We note that in practice, we compute q'_{i+1} afresh when $|q'_{i+1}/q'_i|$ becomes small to guard against possible accumulated rounding errors in the recurrences.

Remark 1 We use this remark to highlight for the reader the differences between Algorithms 2–4. The preliminary sparse Algorithm 3 differs from Algorithm 2 by using the factorization (6) of the sparse matrix H along with the paragraph immediately following (6) to efficiently compute the matrix-vector products with H^{D}. While Algorithm 3 was an improvement, it still involved computations with the dense vectors $\{h^i\}$. This weakness was overcome in Algorithm 4 by showing that the calculations could be rearranged to only involve calculations with the often sparse vectors $\{w^i\}$.

2.1.2 Structured H

Suppose that H has the structure (2). We assume that we can cheaply obtain

$$H_0^{-1} = BB^T \quad (9)$$

Algorithm 4 Finding the Cauchy point (sparse version).

input: Solution estimate $y \in \mathcal{D}$ and search direction d.
Solve $LL^T x = A^T y - g$ and then set $\nabla g^{\mathrm{D}}(y) = Ax - c$.
Compute the vector of breakpoints

$$\alpha_j^{\mathrm{B}} = \begin{cases} -y_j/d_j & \text{if } d_j < 0, \\ \infty & \text{if } d_j \geq 0 \end{cases} \quad \text{for all } j = 1, \ldots n.$$

Compute the index sets $\mathcal{I}^0 = \{j : \alpha_j^{\mathrm{B}} = 0\}$ and $\mathcal{A}^0 = \mathcal{I} \cup \{j : y_j = 0 \text{ and } d_j = 0\}$.
Set $\alpha^0 = 0$, $e^0 = \sum_{j \in \mathcal{I}^0} d_j e_j$, $d^0 = d - e^0$, $u^0 = 0$, and $r^0 = A^T d^0$.
Solve $Lw^0 = r^0$ and set $h^0 = w^0$.
Compute $q_0' = \langle \nabla g^{\mathrm{D}}(y), d^0 \rangle$ and $q_0'' = \langle h^0, h^0 \rangle$.
for $i = 0, 1, 2, \ldots$ **do**

1. (find the next breakpoint)
Determine the next breakpoint α^{i+1}, and then set $\Delta\alpha^i = \alpha^{i+1} - \alpha^i$.

2. (check the current interval for the Cauchy point)
if $q_i'' > 0$ **then** set $\Delta\alpha := -q_i'/q_i''$ else set $\Delta\alpha = \infty$. **end**
if $q_i' \geq 0$, or $q_i'' > 0$ and $\Delta\alpha < \Delta\alpha^i$ **then** define s component-wise by

$$s_j = \begin{cases} \alpha^\ell d_j \text{ for } j \in \mathcal{I}^\ell \text{ for each } \ell = 0, \ldots, i, \\ \alpha^i d_j \text{ for } j \notin \cup_{l=0}^{i} \mathcal{I}^\ell. \end{cases}$$

end
if $q_i' \geq 0$ **then** return the Cauchy point $y^{\mathrm{C}} = y + s$. **end**
if $q_i'' > 0$ and $\Delta\alpha < \Delta\alpha^i$ **then** return the Cauchy point $y^{\mathrm{C}} = y + s + \Delta\alpha d^i$. **end**
if $\Delta\alpha^i = \infty$ and $q_i'' \leq 0$ **then** the problem is unbounded below. **end**

3. (prepare for the next interval)
Compute $\mathcal{I}^{i+1} = \{j : \alpha_j^{\mathrm{B}} = \alpha^{i+1}\}$ and $\mathcal{A}^{i+1} = \mathcal{A}^i \cup \mathcal{I}^{i+1}$.
Compute $e^{i+1} = \sum_{j \in \mathcal{I}^{i+1}} d_j e_j$, $d^{i+1} = d^i - e^{i+1}$, and $u^{i+1} = u^i + \alpha^i w^i$.
Compute $r^{i+1} = A^T e^{i+1}$, and then solve $Lw^{i+1} = r^{i+1}$.

4. (compute the slope and curvature for the next interval)
Compute $q_{i+1}' = q_i' - \langle \nabla g^{\mathrm{D}}(y), e^{i+1} \rangle + \Delta\alpha^i q_i'' - \langle u^{i+1} + \alpha^{i+1} h^i, w^{i+1} \rangle$.
Compute $q_{i+1}'' = q_i'' + \langle w^{i+1} - 2h^i, w^{i+1} \rangle$, and $h^{i+1} = h^i - w^{i+1}$.

end for

for some sparse matrix B (this is trivial in the common case that H^{-1} is diagonal). Now, consider Algorithm 2 and define the following:

$$h_{\mathrm{B}}^i := B^T A^T d^i \in \mathbb{R}^n, \qquad p_{\mathrm{B}}^i := B^T A^T s^i \in \mathbb{R}^n,$$
$$h_{\mathrm{V}}^i := V^T A^T d^i \in \mathbb{R}^t, \qquad p_{\mathrm{V}}^i := V^T A^T s^i \in \mathbb{R}^t.$$

The key, as in Sect. 2.1.1, is to efficiently compute

$$q_i' = \ell^i + \langle p_{\mathrm{B}}^i, h_{\mathrm{B}}^i \rangle + \langle p_{\mathrm{V}}^i, W h_{\mathrm{V}}^i \rangle \text{ and } q_i'' = \langle h_{\mathrm{B}}^i, h_{\mathrm{B}}^i \rangle + \langle h_{\mathrm{V}}^i, W h_{\mathrm{V}}^i \rangle.$$

The terms $\langle p_{\mathrm{V}}^i, W h_{\mathrm{V}}^i \rangle$ and $\langle h_{\mathrm{V}}^i, W h_{\mathrm{V}}^i \rangle$ involve vectors of length t and a matrix of size $t \times t$, so are of low computational cost; the updates $s^{i+1} = s^i + \Delta\alpha^i d^i$ and

$d^{i+1} = d^i - e^{i+1}$ show that h_{V}^{i+1} and p_{V}^{i+1} may be recurred via

$$\begin{aligned} h_{\mathrm{V}}^{i+1} &= h_{\mathrm{V}}^i - w_{\mathrm{V}}^{i+1}, \\ p_{\mathrm{V}}^{i+1} &= p_{\mathrm{V}}^i + \Delta\alpha^i h_{\mathrm{V}}^i, \end{aligned} \quad \text{where } w_{\mathrm{V}}^{i+1} = V^T r^{i+1} \text{ and } r^{i+1} = A^T e^{i+1}.$$

Note that w_{V}^{i+1} is formed from the product of the likely-sparse vector r^{i+1} and the $t \times n$ matrix V^T. Thus, we focus on how to efficiently compute

$$q'_{\mathrm{B},i} := \ell^i + \langle p_{\mathrm{B}}^i, h_{\mathrm{B}}^i \rangle \text{ and } q''_{\mathrm{B},i} := \langle h_{\mathrm{B}}^i, h_{\mathrm{B}}^i \rangle$$

since h_{B}^i and p_{B}^i are generally dense. It follows from $d^{i+1} = d^i - e^{i+1}$ that

$$h_{\mathrm{B}}^{i+1} = h_{\mathrm{B}}^i - w_{\mathrm{B}}^{i+1} \text{ and } p_{\mathrm{B}}^{i+1} = p_{\mathrm{B}}^i + \Delta\alpha^i h_{\mathrm{B}}^i, \text{ where } w_{\mathrm{B}}^{i+1} = B^T r^{i+1}. \quad (10)$$

Note that w_{B}^{i+1} is likely to be sparse since it is formed from the product of the likely-sparse vector r^{i+1} and the sparse matrix B^T. We then follow precisely the reasoning that lead to (7) to see that

$$\begin{aligned} q'_{\mathrm{B},i+1} &= q'_{\mathrm{B},i} - \langle \nabla g^{\mathrm{D}}(y), e^{i+1} \rangle + \Delta\alpha^i q''_{\mathrm{B},i} - \langle p_{\mathrm{B}}^{i+1}, w_{\mathrm{B}}^{i+1} \rangle \text{ and} \\ q''_{\mathrm{B},i+1} &= q''_{\mathrm{B},i} + \langle w_{\mathrm{B}}^{i+1} - 2h_{\mathrm{B}}^i, w_{\mathrm{B}}^{i+1} \rangle, \end{aligned}$$

where we only need to take the inner products over components j for which $[w_{\mathrm{B}}^{i+1}]_j \neq 0$. The only remaining issue is the dense update to p_{B}^i, which is itself required to compute $\langle p_{\mathrm{B}}^{i+1}, w_{\mathrm{B}}^{i+1} \rangle$. However, we may proceed as before to define $u_{\mathrm{B}}^{i+1} = p_{\mathrm{B}}^{i+1} - \alpha^{i+1} h_{\mathrm{B}}^i$ with $u_{\mathrm{B}}^i = 0$ so that

$$u_{\mathrm{B}}^{i+1} = u_{\mathrm{B}}^i + \alpha^i w_{\mathrm{B}}^i \quad (11)$$

and

$$q'_{\mathrm{B},i+1} = q'_{\mathrm{B},i} - \langle \nabla g^{\mathrm{D}}(y), e^{i+1} \rangle + \Delta\alpha^i q''_{\mathrm{B},i} - \langle u_{\mathrm{B}}^{i+1}, w_{\mathrm{B}}^{i+1} \rangle - \alpha^{i+1} \langle h_{\mathrm{B}}^i, w_{\mathrm{B}}^{i+1} \rangle.$$

This recurrence for $q'_{\mathrm{B},i+1}$ and the previous one for $q''_{\mathrm{B},i+1}$ merely require u_{B}^{i+1} and h_{B}^i, which may themselves be recurred using the same likely-sparse w_{B}^{i+1} from (10) and (11). We summarize our findings in Algorithm 5.

2.2 An approximate Cauchy point in Algorithm 1

In some cases, it may be advantageous to approximate the Cauchy point using a backtracking projected linesearch [62]. The basic idea is to pick an initial step size $\alpha_{\text{init}} > 0$, a reduction factor $\beta \in (0, 1)$ and a decrease tolerance $\eta \in (0, \frac{1}{2})$, and then compute the smallest nonnegative integer i for which

$$q^{\mathrm{D}}\left(y^{i+1}\right) \leq q^{\mathrm{D}}(y) + \eta \langle \nabla g^{\mathrm{D}}(y), d^{i+1} \rangle \quad (12)$$

Algorithm 5 Finding the Cauchy point (structured version).

input: Solution estimate $y \in \mathcal{D}$ and search direction d.
Compute $x = (BB^T + VWV^T)(A^T y - g)$ and then set $\nabla g^{\mathrm{D}}(y) = Ax - c$.
Compute the vector of breakpoints

$$\alpha_j^{\mathrm{B}} = \begin{cases} -y_j/d_j & \text{if } d_j < 0, \\ \infty & \text{if } d_j \geq 0 \end{cases} \quad \text{for all } j = 1, \ldots n.$$

Compute $\mathcal{I}^0 = \{j : \alpha_j^{\mathrm{B}} = 0\}$ and $\mathcal{A}^0 = \mathcal{I}^0 \cup \{j : y_j = 0 \text{ and } d_j = 0\}$.
Set $\alpha^0 = 0$, $e^0 = \sum_{j \in \mathcal{I}^0} d_j e_j$, $d^0 = d - e^0$, $u_{\mathrm{B}}^0 = 0$, and $p_{\mathrm{V}}^0 = 0$.
Compute $r^0 = A^T d^0$, $h_{\mathrm{B}}^0 = B^T r^0$, $h_{\mathrm{V}}^0 = V^T r^0$.
Compute $q_0' = \langle \nabla g^{\mathrm{D}}(y), d^0 \rangle$ and $q_0'' = \langle h_{\mathrm{B}}^0, h_{\mathrm{B}}^0 \rangle + \langle h_{\mathrm{V}}^0, W h_{\mathrm{V}}^0 \rangle$.
for $i = 0, 1, 2, \ldots$ **do**

1. (find the next breakpoint)
Determine the next breakpoint α^{i+1}, and then set $\Delta\alpha^i = \alpha^{i+1} - \alpha^i$.

2. (check the current interval for the Cauchy point)
if $q_i'' > 0$ **then** set $\Delta\alpha := -q_i'/q_i''$ **else** set $\Delta\alpha = \infty$. **end**
if $q_i' \geq 0$, or $q_i'' > 0$ and $\Delta\alpha < \Delta\alpha^i$ **then** define s component-wise by

$$s_j = \begin{cases} \alpha^\ell d_j \text{ for } j \in \mathcal{I}^\ell \text{ for each } \ell = 0, \ldots, i, \\ \alpha^i d_j \text{ for } j \notin \cup_{l=0}^{i} \mathcal{I}^\ell. \end{cases}$$

end
if $q_i' \geq 0$ **then** return the exact Cauchy point $y^{\mathrm{C}} = y + s^i$. **end**
if $q_i'' > 0$ and $\Delta\alpha < \Delta\alpha^i$ **then** return the Cauchy point $y^{\mathrm{C}} = y + s^i + \Delta\alpha d^i$. **end**

3. (prepare for the next interval)
Update the index sets $\mathcal{I}^{i+1} = \{j : \alpha_j^{\mathrm{B}} = \alpha^{i+1}\}$ and $\mathcal{A}^{i+1} = \mathcal{A}^i \cup \mathcal{I}^{i+1}$.
Compute $e^{i+1} = \sum_{j \in \mathcal{I}^{i+1}} d_j e_j$, $d^{i+1} = d^i - e^{i+1}$, and $r^{i+1} = A^T e^{i+1}$.
Compute, $w_{\mathrm{B}}^{i+1} = B^T r^{i+1}$, $w_{\mathrm{V}}^{i+1} = V^T r^{i+1}$, and $u_{\mathrm{B}}^{i+1} = u_{\mathrm{B}}^i + \alpha^i w_{\mathrm{B}}^i$.
Set $p_{\mathrm{V}}^{i+1} = p_{\mathrm{V}}^i + \Delta\alpha^i h_{\mathrm{V}}^i$ and $h_{\mathrm{V}}^{i+1} = h_{\mathrm{V}}^i - w_{\mathrm{V}}^{i+1}$.

4. (compute the slope and curvature for the next interval)
Compute $q_{\mathrm{B},i+1}' = q_i' - \langle \nabla g^{\mathrm{D}}(y), e^{i+1} \rangle + \Delta\alpha^i q_i'' - \langle u_{\mathrm{B}}^{i+1} + \alpha^{i+1} h_{\mathrm{B}}^i, w_{\mathrm{B}}^{i+1} \rangle$.
Compute $q_{\mathrm{B},i+1}'' = q_i'' + \langle w_{\mathrm{B}}^{i+1} - 2h_{\mathrm{B}}^i, w_{\mathrm{B}}^{i+1} \rangle$.
Compute $q_{i+1}' = q_{\mathrm{B},i+1}' + \langle p_{\mathrm{V}}^{i+1}, W h_{\mathrm{V}}^{i+1} \rangle$ and $q_{i+1}'' = q_{\mathrm{B},i+1}'' + \langle h_{\mathrm{V}}^{i+1}, W h_{\mathrm{V}}^{i+1} \rangle$.
Compute $h_{\mathrm{B}}^{i+1} = h_{\mathrm{B}}^i - w_{\mathrm{B}}^{i+1}$.

end for

with

$$y^{i+1} = P_{\mathcal{D}}\big[y + \alpha^i d\big], \quad \alpha^i = (\beta)^i \alpha_{\mathrm{init}}, \quad \text{and} \quad d^{i+1} = y^{i+1} - y,$$

for $i \geq 0$. Thus, we must efficiently compute y^{i+1}, $q_{i+1} := q^{\mathrm{D}}(y^{i+1})$, and $q_{i+1}' = \langle \nabla g^{\mathrm{D}}(y), d^{i+1} \rangle$. To this end, we define $\Delta y^i := y^{i+1} - y^i$ for $i \geq 1$. We can then observe, for all $i \geq 1$, that

$$q_{i+1}' = \langle \nabla g^{\mathrm{D}}(y), d^{i+1} \rangle = \langle \nabla g^{\mathrm{D}}(y), y^i - y + \Delta y^i \rangle = q_i' + \langle \nabla g^{\mathrm{D}}(y), \Delta y^i \rangle. \quad (13)$$

To achieve efficiency, we take advantage of the structure of Δy^i and basic properties of the backtracking projected line search. In particular, we know that once a component,

say the jth, satisfies $y_j^i > 0$, then it will also hold that $y_j^\ell > 0$ for all $\ell \geq i$. Thus, in contrast to an exact Cauchy point search that moves forward along the piecewise projected gradient path, the projected backtracking line search only starts to free up variables as the iterations proceed. With this in mind, we compute the index sets

$$\begin{aligned} \mathcal{A} &= \{j : y_j = 0 \text{ and } d_j \leq 0\}, \\ \mathcal{F} &= \{j : d_j > 0, \text{ or } d_j = 0 < y_j\}, \quad \text{and} \\ \mathcal{U} &= \{1, 2, \ldots, m\} \setminus (\mathcal{A} \cup \mathcal{F}), \end{aligned} \tag{14}$$

at the beginning of the process, and maintain the index sets

$$\mathcal{A}^i := \{j \in \mathcal{U} : y_j^i = 0\} \text{ and } \mathcal{F}^i := \{j \in \mathcal{U} : y_j^i > 0\} \quad \text{for} \quad i \geq 1, \tag{15}$$

as well as, for all $i \geq 2$, the index sets

$$\begin{aligned} \mathcal{S}_1^i &:= \mathcal{F}^{i+1} \cap \mathcal{A}^i, \\ \mathcal{S}_2^i &:= \mathcal{F}^{i+1} \cap \mathcal{F}^i \cap \mathcal{A}^{i-1}, \text{ and} \\ \mathcal{S}_3^i &:= \mathcal{F}^{i+1} \cap \mathcal{F}^i \cap \mathcal{F}^{i-1}. \end{aligned} \tag{16}$$

The set $\mathcal{A}$ ($\mathcal{F}$) contains the indices of variables that we know are active (free) at the Cauchy point that will be computed. We also know that

$$\mathcal{F}^i \subseteq \mathcal{F}^{i+1} \text{ and } \mathcal{A}^{i+1} \subseteq \mathcal{A}^i \quad \text{for all} \quad i \geq 1$$

as a consequence of the approximate Cauchy point search. Using these inclusions, it follows that

$$\mathcal{S}_3^{i+1} = \mathcal{S}_3^i \cup \mathcal{S}_2^i, \quad \mathcal{S}_2^{i+1} = \mathcal{S}_1^i, \quad \text{and} \quad \mathcal{S}_1^{i+1} = \mathcal{F}^{i+2} \cap \mathcal{A}^{i+1}.$$

These index sets become useful when noting that, for $i \geq 2$, the following hold:

$$\Delta y_j^i = \begin{cases} 0 & \text{if } j \in \mathcal{A} \cup \mathcal{A}^{i+1}, \\ \beta \Delta y_j^{i-1} & \text{if } j \in \mathcal{F} \cup \mathcal{S}_3^i, \\ y_j^{i+1} - y_j^i & \text{if } j \in \mathcal{S}_1^i \cup \mathcal{S}_2^i. \end{cases} \tag{17}$$

Also, for future reference, note that it follows from (17) that

$$\Delta y^i = \beta \Delta y^{i-1} + \delta y^{i-1}, \text{ with } \delta y_j^{i-1} := \begin{cases} 0 & \text{if } j \in \mathcal{A} \cup \mathcal{A}^{i+1} \cup \mathcal{F} \cup \mathcal{S}_3^i, \\ \Delta y_j^i - \beta \Delta y_j^{i-1} & \text{if } j \in \mathcal{S}_1^i \cup \mathcal{S}_2^i, \end{cases} \tag{18}$$

where the vector δy^{i-1} is usually sparse. Second, the index sets are useful when computing the inner products that involve Δy^i (e.g., see (13)) as shown in Algorithm 6. By combining the recursion performed for $g^i = \langle \nabla g^{\mathrm{D}}(y), \Delta y^i \rangle$ in Algorithm 6 with (13)

Algorithm 6 Efficiently computing $c^i = \langle \Delta y^i, c\rangle$ and $g^i := \langle \nabla g^{\mathrm{D}}(y), \Delta y^i\rangle$.

input: y^1, y^2, Δy^1, $\mathcal{F}$, $\mathcal{A}$, $\mathcal{F}^1$, $\mathcal{A}^1$, $\mathcal{F}^2$, and $\mathcal{A}^2$.
Compute $\mathcal{S}_1^1 = \mathcal{F}^2 \cap \mathcal{A}^1$, $\mathcal{S}_2^1 = \emptyset$, and $\mathcal{S}_3^1 = \mathcal{F}^1 \cap \mathcal{F}^2$.
Compute $g_F^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1\rangle_{\mathcal{F}}$, $g_1^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1\rangle_{\mathcal{S}_1^1}$.
Compute $g_2^1 = 0$, and $g_3^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1\rangle_{\mathcal{S}_3^1}$.
Compute $c_F^1 = \langle c, \Delta y^1\rangle_{\mathcal{F}}$, $c_1^1 = \langle c, \Delta y^1\rangle_{\mathcal{S}_1^1}$, $c_2^1 = 0$, and $c_3^1 = \langle c, \Delta y^1\rangle_{\mathcal{S}_3^1}$.
Set $g^1 = g_F^1 + g_1^1 + g_2^1 + g_3^1$ and $c^1 = c_F^1 + c_1^1 + c_2^1 + c_3^1$.
for $i = 2, 3, \ldots$ **do**
1. (Compute required elements of y^{i+1} and update active sets.)
Set $y_j^{i+1} = y_j + \alpha^i d_j$ for all $j \in \mathcal{S}_1^{i-1}$ and $y_j^{i+1} = P_{\mathcal{D}}(y_j + \alpha^i d_j)$ for all $j \in \mathcal{A}^i$.
Compute $\mathcal{C}^i = \{j \in \mathcal{A}^i : y_j^{i+1} > 0\}$.
Set $\mathcal{F}^{i+1} = \mathcal{F}^i \cup \mathcal{C}^i$ and $\mathcal{A}^{i+1} = \mathcal{A}^i \setminus \mathcal{C}^i$.
2. (Compute required elements of Δy^i.)
Set $\mathcal{S}_1^i = \mathcal{F}^{i+1} \cap \mathcal{A}^i$.
Set $\mathcal{S}_3^i = \mathcal{S}_3^{i-1} \cup \mathcal{S}_2^{i-1}$ and $\mathcal{S}_2^i = \mathcal{S}_1^{i-1}$.
Compute $\Delta y_j^i = y_j^{i+1} - y_j^i$ for all $j \in \mathcal{S}_1^i \cup \mathcal{S}_2^i$.
3. (Perform recursion.)
Set $g_F^i = \beta g_F^{i-1}$.
Set $g_1^i = \langle \nabla g^{\mathrm{D}}(y), \Delta y^i\rangle_{\mathcal{S}_1^i}$, $g_2^i = \langle \nabla g^{\mathrm{D}}(y), \Delta y^i\rangle_{\mathcal{S}_2^i}$, and $g_3^i = \beta(g_2^{i-1} + g_3^{i-1})$.
Set $c_F^i = \beta c_F^{i-1}$, $c_1^i = \langle c, \Delta y^i\rangle_{\mathcal{S}_1^i}$, $c_2^i = \langle c, \Delta y^i\rangle_{\mathcal{S}_2^i}$, and $c_3^i = \beta(c_2^{i-1} + c_3^{i-1})$.
Set $g^i = g_F^i + g_1^i + g_2^i + g_3^i$ and $c^i = c_F^i + c_1^i + c_2^i + c_3^i$.
end for

we obtain an efficient recursion for the sequence $\{q_i'\}$. The other sequence $\{c^i\}$ that is recurred in Algorithm 6 will be used when considering the different representations for H in the next two sections.

2.2.1 Sparse H

We may use (4), (5), (6), and $c^i = \langle \Delta y^i, c\rangle$ introduced in Algorithm 6 to derive

$$
\begin{aligned}
& q_{i+1} \\
&= q^{\mathrm{D}}\big(y^{i+1}\big) = q^{\mathrm{D}}\big(y^i + \Delta y^i\big) \\
&= q^{\mathrm{D}}(y^i) - \langle \Delta y^i, AH^{-1}\big(g - A^T y\big) + c\rangle + \tfrac{1}{2}\langle \Delta y^i, AH^{-1}A^T \Delta y^i\rangle \\
&= q^{\mathrm{D}}(y^i) - \langle \Delta y^i, c\rangle + \big\langle L^{-1}A^T \Delta y^i L^{-1}\big(A^T y - g\big)\big\rangle + \tfrac{1}{2}\langle L^{-1}A^T \Delta y^i, L^{-1}A^T \Delta y^i\rangle \\
&= q^{\mathrm{D}}(y^i) - \langle \Delta y^i, c\rangle + \langle s^i, h\rangle + \tfrac{1}{2}\langle s^i, s^i\rangle = q_i - \langle \Delta y^i, c\rangle + \langle s^i, h\rangle + \tfrac{1}{2}\langle s^i, s^i\rangle \\
&= q_i - c^i + \langle s^i, h\rangle + \tfrac{1}{2}\langle s^i, s^i\rangle, \qquad (19)
\end{aligned}
$$

where

$$
Ls^i = A^T \Delta y^i \quad \text{and} \quad Lh = A^T y - g. \qquad (20)
$$

Note that s^i is not likely to be sparse since Δy^i is usually not sparse, which makes the computation in (19) inefficient. However, by making use of (18), we can see that

$$s^{i+1} = \beta s^i + \delta s^i, \text{ where } L(\delta s^i) = A^T(\delta y^i) \tag{21}$$

since $Ls^{i+1} = \beta Ls^i + L(\delta s^i) = \beta A^T \Delta y^i + A^T(\delta y^i) = A^T\big(\beta \Delta y^i + (\delta y^i)\big) = A^T \Delta y^{i+1}$. Moreover, since δy^i is usually sparse, it follows from (21) that the vector δs^i will likely be sparse when L and A are sparse. We can also use (21) to obtain

$$\begin{aligned}\langle s^{i+1}, h\rangle &= \beta\langle s^i, h\rangle + \langle \delta s^i, h\rangle \text{ and}\\ \langle s^{i+1}, s^{i+1}\rangle &= \beta^2\langle s^i, s^i\rangle + \langle \delta s^i, 2s^i + \delta s^i\rangle,\end{aligned}$$

which allow us to perform the sparse updates required to compute (19). These observations are combined with our previous comments to form Algorithm 7.

2.2.2 Structured H

When H is structured according to (2) and (9), an argument similar to (19) shows that

$$\begin{aligned} q_{i+1} &= q^{\mathrm{D}}\big(y^{i+1}\big) = q^{\mathrm{D}}\big(y^i + \Delta y^i\big)\\ &= q^{\mathrm{D}}(y^i) - \langle \Delta y^i, c\rangle + \big\langle A^T \Delta y^i H^{-1}\big(A^T y - g\big)\big\rangle + \tfrac{1}{2}\langle A^T \Delta y^i, H^{-1} A^T \Delta y^i\rangle\\ &= q_i - \langle \Delta y^i, c\rangle + \langle h_{\mathrm{B}}, s^i_{\mathrm{B}}\rangle + \langle h_{\mathrm{V}}, W s^i_{\mathrm{V}}\rangle + \tfrac{1}{2}\langle s^i_{\mathrm{B}}, s^i_{\mathrm{B}}\rangle + \tfrac{1}{2}\langle s^i_{\mathrm{V}}, W s^i_{\mathrm{V}}\rangle,\\ &= q_i - c^i + \langle h_{\mathrm{B}}, s^i_{\mathrm{B}}\rangle + \langle h_{\mathrm{V}}, W s^i_{\mathrm{V}}\rangle + \tfrac{1}{2}\langle s^i_{\mathrm{B}}, s^i_{\mathrm{B}}\rangle + \tfrac{1}{2}\langle s^i_{\mathrm{V}}, W s^i_{\mathrm{V}}\rangle, \end{aligned} \tag{22}$$

where

$$\begin{aligned} s^i_{\mathrm{B}} &= B^T A^T \Delta y^i, \qquad & h_{\mathrm{B}} &= B^T\big(A^T y - g\big),\\ s^i_{\mathrm{V}} &= V^T A^T \Delta y^i, \qquad & h_{\mathrm{V}} &= V^T\big(A^T y - g\big). \end{aligned} \tag{23}$$

Similar to (21), we can use (18) to obtain

$$s^{i+1}_{\mathrm{B}} = \beta s^i_{\mathrm{B}} + \delta s^i_{\mathrm{B}}, \quad \text{with} \quad \delta s^i_{\mathrm{B}} = B^T A^T(\delta y^i), \tag{24}$$

which in turn leads to

$$\begin{aligned}\langle s^{i+1}_{\mathrm{B}}, h_{\mathrm{B}}\rangle &= \beta\langle s^i_{\mathrm{B}}, h_{\mathrm{B}}\rangle + \langle \delta s^i_{\mathrm{B}}, h_{\mathrm{B}}\rangle \text{ and}\\ \langle s^{i+1}_{\mathrm{B}}, s^{i+1}_{\mathrm{B}}\rangle &= \beta^2\langle s^i_{\mathrm{B}}, s^i_{\mathrm{B}}\rangle + \langle \delta s^i_{\mathrm{B}}, 2s^i_{\mathrm{B}} + \delta s^i_{\mathrm{B}}\rangle.\end{aligned}$$

These updates allow for the sparse updates required to compute (22). These observations are combined with our previous comments to form Algorithm 8.

2.3 The subspace step in Algorithm 1

Let $\mathcal{A}$ be the active set at the Cauchy point (see Algorithm 1) and $\mathcal{F} = \{1, 2, \ldots, m\}\backslash\mathcal{A}$ with $m_{\mathcal{F}} = |\mathcal{F}|$. By construction, the components of y^{C} that correspond to $\mathcal{A}$ are

Algorithm 7 Finding an inexact Cauchy point (sparse version).

input: Current point $y \in \mathcal{D}$ and search direction d.
Choose constants $\alpha_{\text{init}} > 0$, $\beta \in (0, 1)$, and $\eta \in (0, \frac{1}{2})$.
Solve $Lh = A^T y - g$ and $L^T x = h$, and then set $\nabla g^{\mathrm{D}}(y) = Ax - c$.
Solve $Lw = A^T y$, set $v = w - h$, solve $L^T z = v$, and then set $q^{\mathrm{D}}(y) = \frac{1}{2}\langle w, w\rangle - \langle y, c + Az\rangle$.
Define $\mathcal{A}$, $\mathcal{F}$, and $\mathcal{U}$ using (14).
Set $y_j^1 = y_j + d_j$ for $j \in \mathcal{F}$ and $y_j^1 = P_{\mathcal{D}}[y_j + \alpha d_j]$ for $j \in \mathcal{U}$.
Compute $\mathcal{F}^1$ and $\mathcal{A}^1$ using (15).
Solve $Lv = g$, then solve $L^T z = v$.
Set $q_1 = \frac{1}{2}\langle w, w\rangle - \langle y^1, c + Az\rangle$ and $q_1' = \langle \nabla q^{\mathrm{D}}(y), y^1 - y\rangle_{\mathcal{F}\cup\mathcal{U}}$.
if $q_1 \leq q^{\mathrm{D}}(y) + \eta q_1'$ **then** return the approximate Cauchy point y^1. **end**
Compute $y_j^2 = y_j + \alpha^2 d_j$ for all $j \in \mathcal{F} \cup \mathcal{F}^1$ and $y_j^2 = P_{\mathcal{D}}(y_j + \alpha^2 d_j)$ for all $j \in \mathcal{A}^1$.
Compute $\mathcal{C}^1 = \{j \in \mathcal{A}^1 : y_j^2 > 0\}$.
Set $\mathcal{F}^2 = \mathcal{F}^1 \cup \mathcal{C}^1$ and $\mathcal{A}^2 = \mathcal{A}^1 \setminus \mathcal{C}^1$.
Compute $\mathcal{S}_1^1 = \mathcal{F}^2 \cap \mathcal{A}^1$, $\mathcal{S}_2^1 = \emptyset$, and $\mathcal{S}_3^1 = \mathcal{F}^1 \cap \mathcal{F}^2$.
Compute $\Delta y_j^1 = y_j^2 - y_j^1$ for all $j \in \mathcal{F} \cup \mathcal{S}_1^1 \cup \mathcal{S}_3^1$.
Compute $g_F^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1\rangle_{\mathcal{F}}$.
Compute $g_1^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1\rangle_{\mathcal{S}_1^1}$, $g_2^1 = 0$, and $g_3^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1\rangle_{\mathcal{S}_3^1}$.
Compute $c_F^1 = \langle c, \Delta y^1\rangle_{\mathcal{F}}$, $c_1^1 = \langle c, \Delta y^1\rangle_{\mathcal{S}_1^1}$, $c_2^1 = 0$, and $c_3^1 = \langle c, \Delta y^1\rangle_{\mathcal{S}_3^1}$.
Set $g^1 = g_F^1 + g_1^1 + g_2^1 + g_3^1$ and $c^1 = c_F^1 + c_1^1 + c_2^1 + c_3^1$.
Solve for s^1 using (20), and then compute $\langle s^1, h\rangle$ and $\langle s^1, s^1\rangle$.
Set $q_2 = q_1 - c^1 + \langle s^1, h\rangle + \frac{1}{2}\langle s^1, s^1\rangle$ and $q_2' = q_1' + g^1$.
for $i = 2, 3, \ldots$ **do**
  if $q_i \leq q^{\mathrm{D}}(y) + \eta q_i'$ **then** return the approximate Cauchy point y^i. **end**
  Set $\alpha^i = (\beta)^i \alpha_{\text{init}}$.
  Compute $y_j^{i+1} = y_j + \alpha^i d_j$ for all $j \in \mathcal{S}_1^{i-1}$ and $y_j^{i+1} = P_{\mathcal{D}}(y_j + \alpha^i d_j)$ for all $j \in \mathcal{A}^i$.
  Compute $\mathcal{C}^i = \{j \in \mathcal{A}^i : y_j^{i+1} > 0\}$.
  Set $\mathcal{F}^{i+1} = \mathcal{F}^i \cup \mathcal{C}^i$ and $\mathcal{A}^{i+1} = \mathcal{A}^i \setminus \mathcal{C}^i$.
  Set $\mathcal{S}_1^i = \mathcal{F}^{i+1} \cap \mathcal{A}^i$, $\mathcal{S}_3^i = \mathcal{S}_3^{i-1} \cup \mathcal{S}_2^{i-1}$, and $\mathcal{S}_2^i = \mathcal{S}_1^{i-1}$.
  Set $\Delta y_j^i = y_j^{i+1} - y_j^i$ for all $j \in \mathcal{S}_1^i \cup \mathcal{S}_2^i$.
  Set $g_F^i = \beta g_F^{i-1}$.
  Set $g_1^i = \langle \nabla g^{\mathrm{D}}(y), \Delta y^i\rangle_{\mathcal{S}_1^i}$, $g_2^i = \langle \nabla g^{\mathrm{D}}(y), \Delta y^i\rangle_{\mathcal{S}_2^i}$, and $g_3^i = \beta(g_2^{i-1} + g_3^{i-1})$.
  Set $c_F^i = \beta c_F^{i-1}$, $c_1^i = \langle c, \Delta y^i\rangle_{\mathcal{S}_1^i}$, $c_2^i = \langle c, \Delta y^i\rangle_{\mathcal{S}_2^i}$, and $c_3^i = \beta(c_2^{i-1} + c_3^{i-1})$.
  Set $g^i = g_F^i + g_1^i + g_2^i + g_3^i$ and $c^i = c_F^i + c_1^i + c_2^i + c_3^i$.
  Compute δs^{i-1} from (21) with δy^{i-1} defined by (18).
  Compute $\langle s^i, h\rangle = \beta\langle s^{i-1}, h\rangle + \langle \delta s^{i-1}, h\rangle$.
  Compute $\langle s^i, s^i\rangle = \beta^2\langle s^{i-1}, s^{i-1}\rangle + \langle \delta s^{i-1}, 2s^{i-1} + \delta s^{i-1}\rangle$.
  Set $q_{i+1} = q_i - c^i + \langle s^i, h\rangle + \frac{1}{2}\langle s^i, s^i\rangle$ and $q_{i+1}' = q_i' + g^i$.
end for

zero, and those corresponding to $\mathcal{F}$ are strictly positive. With $\Delta y = (\Delta y^A, \Delta y^F)$, the subspace phase requires that we approximately

$$\underset{\Delta y \in \mathbb{R}^m}{\text{minimize}} \ \tfrac{1}{2}\langle \Delta y, H^{\mathrm{D}} \Delta y\rangle + \langle \Delta y, g^{\mathrm{D}} + H^{\mathrm{D}} y^{\mathrm{C}}\rangle \ \text{ subject to } \ \Delta y^A = 0,$$

Algorithm 8 Finding an inexact Cauchy point (structured version).

input: Current point $y \in \mathcal{D}$ and search direction d.
Choose constants $\alpha_{\text{init}} > 0$, $\beta \in (0, 1)$, and $\eta \in (0, \frac{1}{2})$.
Compute $x = (BB^T + VWV^T)(A^T y - g)$, and then set $\nabla g^{\mathrm{D}}(y) = Ax - c$.
Compute $w = B^T A^T y$, $z = V^T A^T y$, $b_g = B^T g$, and $v_g = V^T g$.
Compute $h_{\mathrm{B}} = w - b_g$ and $h_{\mathrm{V}} = z - v_g$.
Set $q^{\mathrm{D}}(y) = \frac{1}{2}\langle w, w\rangle + \frac{1}{2}\langle z, Wz\rangle - \langle y, c\rangle - \langle w, b_g\rangle - \langle z, Wv_g\rangle$.
Define $\mathcal{A}$, $\mathcal{F}$, and $\mathcal{U}$ using (14).
Set $y_j^1 = y_j + d_j$ for $j \in \mathcal{F}$ and $y_j^1 = P_{\mathcal{D}}[y_j + \alpha d_j]$ for $j \in \mathcal{U}$.
Compute $\mathcal{F}^1$ and $\mathcal{A}^1$ using (15).
Compute $q_1 = \frac{1}{2}\langle y^1, A(BB^T + VWV^T)A^T y^1\rangle + \langle A(BB + VWV^T)^T g + c, y^1\rangle$.
Compute $q_1' = \langle \nabla q^{\mathrm{D}}(y), y^1 - y\rangle_{\mathcal{F}\cup\mathcal{U}}$.
if $q_1 \leq q^{\mathrm{D}}(y) + \eta q_1'$ **then** return the approximate Cauchy point y^1. **end**
Compute $y_j^2 = y_j + \alpha^2 d_j$ for all $j \in \mathcal{F} \cup \mathcal{F}^1$ and $y_j^2 = P_{\mathcal{D}}(y_j + \alpha^2 d_j)$ for all $j \in \mathcal{A}^1$.
Compute $\mathcal{C}^1 = \{j \in \mathcal{A}^1 : y_j^2 > 0\}$.
Set $\mathcal{F}^2 = \mathcal{F}^1 \cup \mathcal{C}^1$ and $\mathcal{A}^2 = \mathcal{A}^1 \setminus \mathcal{C}^1$.
Compute $\mathcal{S}_1^1 = \mathcal{F}^2 \cap \mathcal{A}^1$, $\mathcal{S}_2^1 = \emptyset$, and $\mathcal{S}_3^1 = \mathcal{F}^1 \cap \mathcal{F}^2$.
Compute $\Delta y_j^1 = y_j^2 - y_j^1$ for all $j \in \mathcal{F} \cup \mathcal{S}_1^1 \cup \mathcal{S}_3^1$.
Compute $g_F^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1\rangle_{\mathcal{F}}$.
Compute $g_1^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1\rangle_{\mathcal{S}_1^1}$, $g_2^1 = 0$, and $g_3^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1\rangle_{\mathcal{S}_3^1}$.
Compute $c_F^1 = \langle c, \Delta y^1\rangle_{\mathcal{F}}$, $c_1^1 = \langle c, \Delta y^1\rangle_{\mathcal{S}_1^1}$, $c_2^1 = 0$, and $c_3^1 = \langle c, \Delta y^1\rangle_{\mathcal{S}_3^1}$.
Set $g^1 = g_F^1 + g_1^1 + g_2^1 + g_3^1$ and $c^1 = c_F^1 + c_1^1 + c_2^1 + c_3^1$.
Solve for s_{B}^1 using (23), and then compute $\langle s_{\mathrm{B}}^1, h_{\mathrm{B}}\rangle$ and $\langle s_{\mathrm{B}}^1, s_{\mathrm{B}}^1\rangle$.
Set $q_2 = q_1 - c^1 + \langle s_{\mathrm{B}}^1, h_{\mathrm{B}}\rangle + \frac{1}{2}\langle s_{\mathrm{B}}^1, s_{\mathrm{B}}^1\rangle$ and $q_2' = q_1' + g^1$.
for $i = 2, 3, \ldots$ **do**
  if $q_i \leq q^{\mathrm{D}}(y) + \eta q_i'$ **then** return the approximate Cauchy point y^i. **end**
  Set $\alpha^i = (\beta)^i \alpha_{\text{init}}$.
  Compute $y_j^{i+1} = y_j + \alpha^i d_j$ for all $j \in \mathcal{S}_1^{i-1}$ and $y_j^{i+1} = P_{\mathcal{D}}(y_j + \alpha^i d_j)$ for all $j \in \mathcal{A}^i$.
  Compute $\mathcal{C}^i = \{j \in \mathcal{A}^i : y_j^{i+1} > 0\}$.
  Set $\mathcal{F}^{i+1} = \mathcal{F}^i \cup \mathcal{C}^i$ and $\mathcal{A}^{i+1} = \mathcal{F}^i \setminus \mathcal{C}^i$.
  Set $\mathcal{S}_1^i = \mathcal{F}^{i+1} \cap \mathcal{A}^i$, $\mathcal{S}_3^i = \mathcal{S}_3^{i-1} \cup \mathcal{S}_2^{i-1}$, and $\mathcal{S}_2^i = \mathcal{S}_1^{i-1}$.
  Compute $\Delta y_j^i = y_j^{i+1} - y_j^i$ for all $j \in \mathcal{S}_1^i \cup \mathcal{S}_2^i$.
  Set $g_F^i = \beta g_F^{i-1}$.
  Set $g_1^i = \langle \nabla g^{\mathrm{D}}(y), \Delta y^i\rangle_{\mathcal{S}_1^i}$, $g_2^i = \langle \nabla g^{\mathrm{D}}(y), \Delta y^i\rangle_{\mathcal{S}_2^i}$, and $g_3^i = \beta(g_2^{i-1} + g_3^{i-1})$.
  Set $c_F^i = \beta c_F^{i-1}$, $c_1^i = \langle c, \Delta y^i\rangle_{\mathcal{S}_1^i}$, $c_2^i = \langle c, \Delta y^i\rangle_{\mathcal{S}_2^i}$, and $c_3^i = \beta(c_2^{i-1} + c_3^{i-1})$.
  Set $g^i = g_F^i + g_1^i + g_2^i + g_3^i$ and $c^i = c_F^i + c_1^i + c_2^i + c_3^i$.
  Compute $\delta s_{\mathrm{B}}^{i-1}$ from (24) with δy^{i-1} defined by (18).
  Compute $\langle s_{\mathrm{B}}^i, h_{\mathrm{B}}\rangle = \beta\langle s_{\mathrm{B}}^{i-1}, h_{\mathrm{B}}\rangle + \langle \delta s_{\mathrm{B}}^{i-1}, h_{\mathrm{B}}\rangle$.
  Compute $\langle s_{\mathrm{B}}^i, s_{\mathrm{B}}^i\rangle = \beta^2\langle s_{\mathrm{B}}^{i-1}, s_{\mathrm{B}}^{i-1}\rangle + \langle \delta s_{\mathrm{B}}^{i-1}, 2s_{\mathrm{B}}^{i-1} + \delta s_{\mathrm{B}}^{i-1}\rangle$.
  Set $q_{i+1} = q_i - c^i + \langle s_{\mathrm{B}}^i, h_{\mathrm{B}}\rangle + \frac{1}{2}\langle s_{\mathrm{B}}^i, s_{\mathrm{B}}^i\rangle$ and $q_{i+1}' = q_i' + g^i$.
end for

whose solution (when it exists) we denote by $\Delta y_* = (\Delta y_*^A, \Delta y_*^F) = (0, \Delta y_*^F)$ with

$$\Delta y_*^F = \underset{\Delta y^F \in \mathbb{R}^{m_{\mathcal{F}}}}{\operatorname{argmin}} \, q^F\left(\Delta y^F\right) := \tfrac{1}{2}\langle \Delta y^F, H^F \Delta y^F\rangle + \langle \Delta y^F, g^F\rangle, \tag{25}$$

where

$$H^F := A^F H^{-1} (A^F)^T \quad \text{and} \quad g^F := -A^F H^{-1}\Big(g - (A^F)^T (y^{\mathrm{C}})^F\Big) - c^F. \tag{26}$$

here, A^F and $(y^{\mathrm{C}})^F/c^F$ denote, respectively, the rows of A and components of y^{C}/c that correspond to the index set $\mathcal{F}$. There are then two distinct possibilities. Either (25) has a finite solution or $q^F(\Delta y^F)$ is unbounded below. Moreover, we may attempt to find such a solution using either a direct (factorization-based) or iterative approach. We consider these aspects over the next several sections.

2.3.1 Finite subspace minimizer

Since the objective in (25) is convex and quadratic, when $q^F(\Delta y^F)$ is bounded below its stationarity conditions give that

$$H^F \Delta y_*^F = -g^F, \tag{27}$$

that is to say

$$A^F H^{-1} (A^F)^T \Delta y_*^F = A^F H^{-1}\Big(g - (A^F)^T (y^{\mathrm{C}})^F\Big) + c^F. \tag{28}$$

Given x, if we then define Δx_* via

$$x + \Delta x_* = H^{-1}\Big[(A^F)^T\Big((y^{\mathrm{C}})^F + \Delta y_*^F\Big) - g\Big],$$

then we have

$$\begin{pmatrix} H & (A^F)^T \\ A^F & 0 \end{pmatrix} \begin{pmatrix} x + \Delta x_* \\ -(y^{\mathrm{C}})^F - \Delta y_*^F \end{pmatrix} = \begin{pmatrix} -g \\ c^F \end{pmatrix} \tag{29}$$

or equivalently

$$\begin{pmatrix} H & (A^F)^T \\ A^F & 0 \end{pmatrix} \begin{pmatrix} x + \Delta x_* \\ -\Delta y_*^F \end{pmatrix} = \begin{pmatrix} (A^F)^T (y^{\mathrm{C}})^F - g \\ c^F \end{pmatrix}. \tag{30}$$

Notice that, so long as (29) (equivalently (30)) is consistent, we have

$$\Delta x_* = \underset{\Delta x \in \mathbb{R}^n}{\operatorname{argmin}} \, \tfrac{1}{2}\langle x + \Delta x, H(x + \Delta x)\rangle + \langle x + \Delta x, g\rangle \;\text{ subject to }\; A^F(x + \Delta x) = c^F,$$

with $(y^{\mathrm{C}})^F + \Delta y_*^F$ the Lagrange multiplier vector. Of course the previous optimization problem is nothing other than a subproblem that would be solved by a primal active-set method for a correction Δx to a given x, and this indicates that the dual generalized Cauchy point may alternatively be viewed as a mechanism for identifying primal active constraints. Finally, we note that Δy_*^F may be computed using any of the many direct or iterative methods for solving (28) or (30) [4].

2.3.2 Subspace minimizer at infinity

Since H^F is positive semi-definite and possibly singular, the linear system (28) (equivalently (27)) may be inconsistent. We shall use the following well-known generic result.[1] (The proof follows by minimizing $\|Mu - b\|_2^2$ and letting $v = b - Mu$ whenever $Mu \neq b$).

Theorem 1 [The Fredholm Alternative [57, Thm 4, p. 174]] *Let $M \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Then, either there exists $u \in \mathbb{R}^n$ for which $Mu = b$ or there exists $v \in \mathbb{R}^m$ such that $M^T v = 0$ and $\langle b, v\rangle > 0$.*

It follows from Theorem 1 that if (27) is inconsistent, there must be a direction of linear infinite descent [13], i.e., a vector Δy_∞^F for which

$$H^F \Delta y_\infty^F = 0 \quad \text{and} \quad \langle \Delta y_\infty^F, g^F\rangle < 0 \tag{31}$$

along which

$$q^F\left(\alpha \Delta y_\infty^F\right) = \alpha \langle \Delta y_\infty^F, g^F\rangle$$

decreases linearly to minus infinity as α increases. Alternatively, examining (26), it is clear that inconsistency of (28) is only possible when c^F does not lie in the range of A^F. In this case, the Fredholm alternative implies that there exists a Δy_∞^F satisfying

$$(A^F)^T \Delta y_\infty^F = 0 \quad \text{and} \quad \langle \Delta y_\infty^F, c^F\rangle > 0, \tag{32}$$

which is also a direction of linear infinite descent since

$$q^F\left(\alpha \Delta y_\infty^F\right) = -\alpha \langle \Delta y_\infty^F, c^F\rangle.$$

We now consider how to use H^F to find a Δy_∞^F that satisfies (31), and leave the details of how we might instead satisfy (32) to the Supplementary Material.

To find Δy_∞^F satisfying (31), we require $H^F \Delta y_\infty^F \equiv A^F H^{-1} (A^F)^T \Delta y_\infty^F = 0$. Now, if we define $\Delta w_\infty := H^{-1}(A^F)^T \Delta y_\infty^F$, then we have that

$$\begin{pmatrix} H & (A^F)^T \\ A^F & 0 \end{pmatrix} \begin{pmatrix} \Delta w_\infty \\ -\Delta y_\infty^F \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{33}$$

[1] Note that the sign of the inner product $\langle b, v\rangle$ is arbitrary, since, for $-v$, $M^T(-v) = 0$ and $\langle b, -v\rangle < 0$. We shall refer to a *negative Fredholm alternative* as that for which the signs of the components of v are flipped.

Moreover, the second requirement in (31) is that

$$0 > \langle \Delta y_\infty^F, g^F \rangle = \langle -\Delta y_\infty^F, c^F \rangle + \langle \Delta w_\infty, (A^F)^T (y^C)^F - g \rangle, \tag{34}$$

where we have used (26) and the definition of Δw_∞. Notice that (33)–(34) together satisfy the negative Fredholm alternative to (30), which can be seen by applying Theorem 1 with the following data:

$$M = K := \begin{pmatrix} H & (A^F)^T \\ A^F & 0 \end{pmatrix}, \qquad u = \begin{pmatrix} x + \Delta x \\ -\Delta y^F \end{pmatrix},$$
$$b := \begin{pmatrix} (A^F)^T (y^C)^F - g \\ c^F \end{pmatrix}, \qquad v = \begin{pmatrix} \Delta w \\ -\Delta y_\infty^F \end{pmatrix}.$$

To see how we might compute the required direction of linear infinite descent in this case, suppose that

$$K = LBL^T, \tag{35}$$

where L is a permuted unit-lower-triangular matrix and B is a symmetric block diagonal matrix comprised of one-by-one and two-by-two blocks—the sparse symmetric-indefinite factorization packages `MA27` [25], `MA57` [24], `MA77` [68], `MA86` [54], `MA97` [55], `PARDISO` [70] and `WSMP` [50] offer good examples. Crucially, any singularity in K is reflected solely in B. Our method will return either a solution to (30) or to (33)–(34). To this end, we define w so that

$$Lw = b,$$

and then consider trying to solve the system

$$Bz = w. \tag{36}$$

If the system (36) is consistent, and thus there is a z satisfying (36), the vector u for which $L^T u = z$ also solves $Ku = b$ and thus its components solve (30). By contrast, if (36) is inconsistent, Theorem 1 implies that there is a vector p for which

$$Bp = 0 \ \text{ and } \ \langle p, w \rangle > 0. \tag{37}$$

In this case, the vector v for which $L^T v = p$ also satisfies

$$Kv = LBL^T v = LBp = 0$$

and

$$\langle v, b \rangle = \langle v, Lw \rangle = \langle L^T v, w \rangle = \langle p, w \rangle > 0$$

because of (37). Thus the Fredholm alternative for data K and b may be resolved by considering the Fredholm alternative for the far-simpler block-diagonal B and w.

To investigate the consistency of (36), we form the spectral factorizations $B_i = Q_i D_i Q_i^T$ for an orthonormal Q_i and diagonal D_i for each of the ℓ (say) one-by-one and two-by-two diagonal blocks, and build the overall factorization

$$B = \begin{pmatrix} Q_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & Q_\ell \end{pmatrix} \begin{pmatrix} D_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & D_\ell \end{pmatrix} \begin{pmatrix} Q_1^T & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & Q_\ell^T \end{pmatrix} = QDQ^T.$$

Singularity of B is thus readily identifiable from the block diagonal matrix D, and the system (36) is consistent if and only if $(Q^T w)_i = 0$ whenever $D_{ii} = 0$, where D_{ii} is the ith diagonal entry of the matrix D. If the system is inconsistent, then a direction p of linear infinite descent satisfying (37) can be obtained by defining $\mathcal{J} = \{j : D_{jj} = 0 \text{ and } (Q^T w)_j \neq 0\} \neq \emptyset$ and then setting $p = Qs$, where

$$s_j = \begin{cases} 0 & \text{for} \quad j \notin \mathcal{J}, \\ (Q^T w)_j & \text{for} \quad j \in \mathcal{J}. \end{cases}$$

In particular, it follows that

$$\langle p, w \rangle = \langle Qs, w \rangle = \langle s, Q^T w \rangle = \sum_{j \in \mathcal{J}} \left(Q^T w\right)_j^2 > 0 \quad \text{and}$$
$$Bp = QDQ^T p = QDs = 0$$

since $Ds = 0$ by construction. Thus, we have verified that this choice for p satisfies (37) as required. New subroutines to implement the Fredholm alternative as just described have been added to the HSL packages `MA57`, `MA77` and `MA97`, and versions will be added to `MA27` and `MA86` in due course.

When H^{-1} is structured according to (2), it is easy to show that

$$H = H_0 - ZU^{-1}Z^T,$$

where $U = W^{-1} + Y^T H_0 Y \in \mathbb{R}^{t \times t}$ and $Z = H_0 Y \in \mathbb{R}^{n \times t}$. Although the decomposition (35) is possible for such an H, it will most likely result in a dense factor L since H is dense, and thus a sparse alternative is preferable. The trick is to see that (30), with $(\Delta x, \Delta y^F) = (\Delta x_*, \Delta y_*^F)$, may be expanded to give

$$\begin{pmatrix} H_0 & (A^F)^T & Z \\ A^F & 0 & 0 \\ Z^T & 0 & U \end{pmatrix} \begin{pmatrix} x + \Delta x \\ -\Delta y^F \\ -\Delta z \end{pmatrix} = \begin{pmatrix} (A^F)^T (y^C)^F - g \\ c^F \\ 0 \end{pmatrix}, \tag{38}$$

where we introduced the variables $\Delta z := U^{-1} Z^T (x + \Delta x)$. The leading $(n + m_{\mathcal{F}}) \times (n + m_{\mathcal{F}})$ block of this system is sparse, and only the last t rows and columns are dense. If we consider Theorem 1 with data

Algorithm 9 The conjugate gradient method for solving (25).

input: g^F and H^F as defined in (26).
Set $\Delta y_0^F = 0$, $g_0 = g^F$, and $p_0 = -g_0$, and then choose $\varepsilon > 0$.
for $j = 0, 1, 2, \ldots$ **do**
  if $\|g_j\| \leq \varepsilon$ **then** return the subspace step $\Delta y_*^F := \Delta y_j$. **end**
  if $\langle p_j, H^F p_j \rangle \leq \varepsilon$ **then** return the direction of linear infinite descent $\Delta y_\infty^F = p_j$. **end**
  Set $\alpha_j = \|g_j\|_2^2 / \langle p_j, H^F p_j \rangle$.
  Set $\Delta y_{j+1}^F = \Delta y_j^F + \alpha_j p_j$.
  Set $g_{j+1} = g_j + \alpha_j H^F p_j$.
  Set $\beta_j = \|g_{j+1}\|_2^2 / \|g_j\|_2^2$.
  Set $p_{j+1} = -g_{j+1} + \beta_j p_j$.
end for

$$M = K_+ := \begin{pmatrix} H_0 & (A^F)^T & Z \\ A^F & 0 & 0 \\ Z^T & 0 & U \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} (A^F)^T (y^C)^F - g \\ c^F \\ 0 \end{pmatrix},$$

we can conclude that there either exists a vector

$$u = \begin{pmatrix} x + \Delta x \\ -\Delta y^F \\ -\Delta z \end{pmatrix} \quad \text{or} \quad v = \begin{pmatrix} \Delta w \\ -\Delta y_\infty^F \\ -\Delta z_\infty \end{pmatrix}$$

such that the first two block components of u and v provide, respectively, either a solution to (30) or its Fredholm alternative (33)–(34). A decomposition of the form (35) for K_+ and its factors then reveals the required solution to the subspace problem or a direction of linear infinite descent precisely as outlined above for the sparse H case. Sparse factorizations of K_+ generally aim to preserve sparsity, with the few dense rows pivoted to the end of the factors.

2.3.3 Iterative methods

The obvious iterative approach for solving (25) is to use the conjugate gradient method to generate a sequence $\{\Delta y_j^F\}$ for $j \geq 0$ starting from $\Delta y_0^F = 0$ (see Algorithm 9). Here, $g_j = \nabla q^F(\Delta y_j^F)$ so that $g_0 = g^F$. The iteration is stopped with $\Delta y_*^F = \Delta y_j^F$ as an approximate solution to (25) when the gradient $g_j = \nabla q^F(\Delta y_j^F)$ is small, or with $\Delta y_\infty^F = p_j$ as a linear infinite descent direction when $\langle p_j, H^F p_j \rangle$ is small. We note that a basic property of CG is that $q^F(\Delta y_j^F) < q^F(0)$ for all $j \geq 1$.

The input gradient g^F and each product $H^F p_j$ requires products with A, A^T and H^{-1} and as always the latter are obtained using the Cholesky factors (6) or the structured decomposition (2). Note that if $\langle p_j, H^F p_j \rangle = 0$ then $H^F p_j = 0$ since H^F is positive semi-definite, and in this case

$$\langle p_j, g^F \rangle = \langle p_j, \nabla q^F(\Delta y_j^F) \rangle - \langle p_j, H^F \Delta y_j^F \rangle = \langle p_j, \nabla q^F(\Delta y_j^F) \rangle = \langle p_j, g_j \rangle < 0,$$

where we have used $\nabla q^F(\Delta y_j^F) = H^F \Delta y_j^F + g^F$ and the well-known fact that the CG iterations satisfy $\langle p_j, g_j \rangle < 0$. Thus, the vector $y_\infty^F = p_j$ is a direction of linear infinite descent satisfying (31) at y^{C}. On the other hand, when $\langle p_j, H^F p_j \rangle > 0$, then

$$q^F\left(\Delta y_j^F\right) - q^F\left(\Delta y_{j+1}^F\right) = \tfrac{1}{2}\langle p_j, g_j \rangle^2 / \langle p_j, H^F p_j \rangle, \tag{39}$$

which follows from $\Delta y_{j+1}^F = \Delta y_j^F + \alpha_j p_j$ and the fact that α_j in Algorithm 9 can be equivalently written as $\alpha_j = -\langle g_j, p_j \rangle / \langle p_j, H^F p_j \rangle$. Thus, for some small $\epsilon_\infty > 0$, we stop the iteration whenever

$$\langle p_j, H^F p_j \rangle \leq \tfrac{1}{2}\epsilon_\infty \langle p_j, g_j \rangle^2$$

since (39) then gives a decrease in the dual objective function of at least $1/\epsilon_\infty$, which indicates that it is likely unbounded below.

3 The method for the general problem formulation

In practice most problems have the general form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\, q(x) = \tfrac{1}{2}\langle x, Hx \rangle + \langle g, x \rangle \;\text{ subject to }\; c_L \leq Ax \leq c_U,\; x_L \leq x \leq x_U, \tag{40}$$

where any of the components for c_L, c_U, x_L, and x_U may be infinite (i.e., we permit one-sided or free constraints/variables) and individual pairs may be equal (i.e., we allow equality constraints/fixed variables). We now briefly describe how our algorithm applies to this general formulation.

The constraints in (40) may be written as

$$A_g := \begin{pmatrix} A \\ -A \\ I \\ -I \end{pmatrix} x \geq \begin{pmatrix} c_L \\ -c_U \\ x_L \\ -x_U \end{pmatrix} =: c_g, \quad \text{with dual variables } \; y_g := \begin{pmatrix} y_L \\ -y_U \\ z_L \\ -z_U \end{pmatrix}.$$

Then using the recipe (4), we have the dual problem

$$\underset{y_g}{\text{minimize}}\, \tfrac{1}{2} y_g^T H_g^{\mathrm{D}} y_g + y_g^T g_g^{\mathrm{D}} \;\text{ subject to }\; y_g \geq 0, \tag{41}$$

with

$$H_g^{\mathrm{D}} = A_g H^{-1} A_g^T \;\text{ and }\; g_g^{\mathrm{D}} = -A_g H^{-1} g - c_g. \tag{42}$$

If we now introduce the notation

$$v = \begin{pmatrix} y_L \\ y_U \\ z_L \\ z_U \end{pmatrix}, \quad J = \begin{pmatrix} A \\ A \\ I \\ I \end{pmatrix}, \quad b = \begin{pmatrix} c_L \\ c_U \\ x_L \\ x_U \end{pmatrix}, \quad \text{and} \quad \begin{aligned} H^{\mathrm{D}} &:= JH^{-1}J^T, \\ g^{\mathrm{D}} &:= -JH^{-1}g - b, \end{aligned} \tag{43}$$

then it is easy to show that

$$y_g^T H_g^{\mathrm{D}} y_g = v^T JH^{-1}J^T v = v^T H^{\mathrm{D}} v \quad \text{and} \quad y_g^T g_g^{\mathrm{D}} = v^T(-JH^{-1}g - b) = v^T g^{\mathrm{D}}.$$

Using these relationships, we see that problem (41) is equivalent to the problem

$$v_* = \underset{v \in \mathbb{R}^{n_v}}{\operatorname{argmin}} \, q^{\mathrm{D}}(v) := \tfrac{1}{2} v^T H^{\mathrm{D}} v + v^T g^{\mathrm{D}} \ \text{ subject to } \ (y_L, z_L) \geq 0, \ (y_U, z_U) \leq 0, \tag{44}$$

where $n_v = 2m + 2n$, in the sense that a solution to (41) is trivially obtained from a solution to (44).

Notice that since

$$\begin{aligned} q^{\mathrm{D}}(v) = \tfrac{1}{2}\big[(A^T(y_L + y_U) + (z_L + z_U)\big]^T H^{-1}\big[A^T(y_L + y_U) + z_L + z_U\big] \\ - \big[(y_L + y_U)^T A + (z_L + z_U)^T\big]H^{-1}g - \Big(c_L^T y_L + c_U^T y_U + x_L^T z_L + x_U^T z_U\Big), \end{aligned}$$

that if $(c_L)_i = (c_U)_i$, for any i (i.e., the ith constraint is an equality constraint), then we may replace the corresponding variables $(y_L)_i$ and $(y_U)_i$ in (44) by $y_i = (y_L)_i + (y_U)_i$, where y_i is not restricted in sign. Also, anytime an infinite bound occurs, we omit the relevant dual variable, its bound and the corresponding row of A or I from the formal description above.

To solve (44), we simply generalize the method used to solve (4). Let

$$P_{\mathcal{D}}[v] = [\max(y_L, 0), \min(y_U, 0), \max(z_L, 0), \min(z_U, 0)]^T$$

be the projection of $v = [y_L, y_U, z_L, z_U]^T$ onto the feasible region

$$\mathcal{D} = \{v = [y_L, y_U, z_L, z_U]^T : (y_L, z_L) \geq 0 \ \text{ and } \ (y_U, z_U) \leq 0\}$$

for (44). Then we apply Algorithm 10.

The only significant differences for finding the Cauchy and improved subspace points using the obvious extension of Algorithm 4—aside from the implicit increase in dimension when considering v rather than y—are that (i) we need to compute additional breakpoints for the upper bounded variables using

$$\alpha_i^{\mathrm{B}} = \begin{cases} -v_j/d_j & \text{if } \ d_j > 0, \\ \infty & \text{if } \ d_j \leq 0, \end{cases}$$

Algorithm 10 Gradient projection method for solving the BQP (44).

input: Solution estimate $v \in \mathcal{D}$.
while $P_{\mathcal{D}}[v - \nabla q^{\mathrm{D}}(v)] \neq v$ **do**
1. (Cauchy point)
Set $d = -\nabla q^{\mathrm{D}}(v)$ and compute $\alpha^{\mathrm{C}} = \operatorname{argmin}_{\alpha>0} q^{\mathrm{D}}\big(P_{\mathcal{D}}[v + \alpha d]\big)$.
Set $v^{\mathrm{C}} = P_{\mathcal{D}}[v + \alpha^{\mathrm{C}} d]$.
2. (subspace step)
Compute $\mathcal{A} = \mathcal{A}(v^{\mathrm{C}}) := \{i : v_i^{\mathrm{C}} = 0\}$.
Compute $\Delta v^{\mathrm{S}} = \operatorname{argmin}_{\Delta v \in \mathbb{R}^{n_v}} q^{\mathrm{D}}(v^{\mathrm{C}} + \Delta v)$ subject to $[\Delta v]_{\mathcal{A}} = 0$.
3. (improved subspace point)
Select $\alpha_{\max} > 0$ and then compute $\alpha^{\mathrm{S}} = \operatorname{argmin}_{\alpha \in [0, \alpha_{\max}]} q^{\mathrm{D}}\Big(P_{\mathcal{D}}[v^{\mathrm{C}} + \alpha \Delta v^{\mathrm{S}}]\Big)$.
Set $v = P_{\mathcal{D}}[v^{\mathrm{C}} + \alpha^{\mathrm{S}} \Delta v^{\mathrm{S}}]$.
end while

(ii) the index sets $\mathcal{I}_{i+1}$ need to take into account these extra breakpoints, and (iii) any mention of A and c should now refer to J and b from (43).

The computation of the subspace step is likewise very similar. The Cauchy point fixes components of v at zero, and this results in expanded systems of the form

$$\begin{pmatrix} H & (J^F)^T \\ J^F & 0 \end{pmatrix} \begin{pmatrix} x + \Delta x \\ -\Delta v^F \end{pmatrix} = \begin{pmatrix} (J^F)^T (v^{\mathrm{C}})^F - g \\ b^F \end{pmatrix} \tag{45}$$

for (30), and

$$\begin{pmatrix} H_0 & (J^F)^T & Z \\ J^F & 0 & 0 \\ Z^T & 0 & U \end{pmatrix} \begin{pmatrix} x + \Delta x \\ -\Delta v^F \\ -\Delta z \end{pmatrix} = \begin{pmatrix} (J^F)^T (v^{\mathrm{C}})^F - g \\ b^F \\ 0 \end{pmatrix}$$

for (38), where J^F and b^F consist of the components of (43) that are free at the Cauchy point v^{C}.

3.1 A special block-diagonal Hessian

An important special case occurs when H has a particular block-diagonal structure, and covers applications that include performing ℓ_2-projections onto a polytope [1], solving bound-constrained least-squares problems [6,8, §7.7], and globalization strategies within modern SQP algorithms [28,46]. An obvious simplification is that any solve involving H can be decomposed into blocks, and is trivial if H is diagonal. A more significant advantage comes in the subspace phase. Suppose, without loss of generality, that

$$J^F = \begin{pmatrix} A_1^F & A_2^F \\ I & 0 \end{pmatrix}, \quad H = \begin{pmatrix} H_1 & 0 \\ 0 & H_2 \end{pmatrix}, \quad b^F = \begin{pmatrix} c^F \\ x_1 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ and } g = \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}$$

with $A_1^F \in \mathbb{R}^{m_1 \times n_1}$, $H_1 \in \mathbb{R}^{n_1 \times n_1}$ and $g_1 \in \mathbb{R}^{n_1}$. Problem (45) can then be written as

$$\begin{pmatrix} H_1 & 0 & (A_1^F)^T & I \\ 0 & H_2 & (A_2^F)^T & 0 \\ A_1^F & A_2^F & 0 & 0 \\ I & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 + \Delta x_1 \\ x_2 + \Delta x_2 \\ -(y^{\mathrm{C}})^F - \Delta y^F \\ -(z^{\mathrm{C}})^F - \Delta z^F \end{pmatrix} = \begin{pmatrix} -g_1 \\ -g_2 \\ c^F \\ x_1 \end{pmatrix}, \tag{46}$$

or equivalently $\Delta x_1 = 0$, $\Delta z^F = g_1 + H_1 x_1 - (A_1^F)^T \big((y^{\mathrm{C}})^F + \Delta y^F\big) - (z^{\mathrm{C}})^F$, and

$$\begin{pmatrix} H_2 & (A_2^F)^T \\ A_2^F & 0 \end{pmatrix} \begin{pmatrix} x_2 + \Delta x_2 \\ -(y^{\mathrm{C}})^F - \Delta y^F \end{pmatrix} = \begin{pmatrix} -g_2 \\ c^F - A_1^F x_1 \end{pmatrix}. \tag{47}$$

Thus the subspace phase is equivalent to finding a Fredholm alternative to the "reduced" system (47) or equivalently a Fredholm alternative to

$$A_2^F H_2^{-1} \big(A_2^F\big)^T \Delta y^F = A_2^F H_2^{-1} \Big(g_2 - (A^F)^T (y^{\mathrm{C}})^F\Big) + c^F - A_1^F x_1,$$

where we subsequently recover Δx_2 by solving trivially

$$H_2(x_2 + \Delta x_2) = \big(A_2^F\big)^T \Big((y^{\mathrm{C}})^F + \Delta y^F\Big) - g_2.$$

4 Regularized problems

A related problem of interest—for example, in Sl_pQP methods for constrained optimization [26]—is to

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\, q(x) + \sigma \|(Ax - c)^-\|_1, \tag{48}$$

for given $\sigma > 0$, where $w^- = \max(0, -w)$ for any given w. Introducing the auxiliary vector v allows us to write (48) equivalently as

$$\underset{(x,v) \in \mathbb{R}^{n+m}}{\text{minimize}}\, q(x) + \sigma e^T v \;\; \text{subject to} \;\; Ax + v \geq c, \;\; v \geq 0.$$

This problem has the associated primal-dual problem

$$\begin{aligned} \underset{(x,v,y,z) \in \mathbb{R}^{2n+2m}}{\text{maximize}} \quad & -\tfrac{1}{2}\langle x, Hx\rangle + \langle c, y\rangle \\ \text{subject to} \quad & Hx - A^T y + g = 0, \;\; y + z = \sigma e, \;\; (y, z) \geq 0, \end{aligned}$$

whose optimal y value can equivalently be computed from the problem

$$\underset{y \in \mathbb{R}^m}{\text{minimize}}\, \tfrac{1}{2}\langle A^T y - g, H^{-1}\big(A^T y - g\big)\rangle - \langle c, y\rangle \;\; \text{subject to} \;\; 0 \leq y \leq \sigma e. \tag{49}$$

We may thus apply essentially the same accelerated gradient-projection method as before, but now we project (trivially) into $[0, \sigma e]$. Similarly, if we wish to

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\ q(x) + \sigma \|(Ax - c)\|_1, \tag{50}$$

we may instead solve the problem

$$\underset{y \in \mathbb{R}^m}{\text{minimize}}\ \tfrac{1}{2}\langle A^T y - g, H^{-1}\big(A^T y - g\big)\rangle - \langle c, y\rangle \ \text{ subject to } \ -\sigma e \le y \le \sigma e \tag{51}$$

using gradient projection onto $[-\sigma e, \sigma e]$, and recover x from $Hx = A^T y - g$.

If we consider the same problem in the infinity norm, namely

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\ q(x) + \sigma \|(Ax - c)^-\|_\infty, \tag{52}$$

then by introducing the auxiliary vector v we see that (52) is equivalent to

$$\underset{(x,v) \in \mathbb{R}^{n+1}}{\text{minimize}}\ q(x) + \sigma v \ \text{ subject to } \ Ax + ve \ge c,\ v \ge 0. \tag{53}$$

The primal-dual problem to (53) is then

$$\begin{aligned} &\underset{(x,v,y,\xi) \in \mathbb{R}^{n+m+2}}{\text{maximize}}\ -\tfrac{1}{2}\langle x, Hx\rangle + \langle c, y\rangle \\ &\text{subject to } \ Hx - A^T y + g = 0,\ \langle e, y\rangle + \xi = \sigma,\ (y, \xi) \ge 0, \end{aligned}$$

whose optimal y value may be equivalently computed as the solution to

$$\underset{y \in \mathbb{R}^m}{\text{minimize}}\ \tfrac{1}{2}\langle A^T y - g, H^{-1}\big(A^T y - g\big)\rangle - \langle c, y\rangle \ \text{ subject to } \ y \ge 0,\ \langle e, y\rangle \le \sigma. \tag{54}$$

Once again, we may apply the accelerated gradient-projection method, but now the projection is onto the orthogonal simplex $\mathcal{S}_m(\sigma) := \{y \in \mathbb{R}^m : y \ge 0 \ \text{ and } \ \langle e, y\rangle \le \sigma\}$ for which there are nonetheless efficient projection algorithms [73]. In addition, in the subspace step computation (30), the defining matrix may have an additional row and column e whenever the dual constraint $\langle e, y\rangle \le \sigma$ is active. Similarly, to

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\ q(x) + \sigma \|(Ax - c)\|_\infty, \tag{55}$$

we can solve the optimization problem

$$\underset{(y_L, y_U) \in \mathcal{S}_{2m}(\sigma)}{\text{minimize}}\ \tfrac{1}{2}\langle A^T(y_L - y_U) - g, H^{-1}\big(A^T(y_L - y_U) - g\big)\rangle - \langle c, y_L - y_U\rangle \tag{56}$$

using accelerated gradient projection within $\mathcal{S}_{2m}(\sigma)$, and subsequently obtain x from $Hx = A^T(y_L - y_U) - g$.

5 Computational experience

5.1 Implementation and test problems

We implemented the algorithm outlined in Sects. 2 and 3 for strictly-convex quadratic programs, in the general form (40), as a Fortran 2003 package called DQP that is available as part of GALAHAD [38]. Details, including a complete description of user control parameters, are provided in the package documentation provided as part of GALAHAD.[2] Features such as problem pre-processing, removal of dependent constraints, general strategies for solving symmetric systems, exploiting parallelism in the linear algebra, presolve strategies, and problem scaling are similar to those described in [37] for the interior-point QP solver CQP within GALAHAD; package default values are chosen unless otherwise specified. DQP offers a choice of (dual) starting point strategies. These include allowing the user to specify the starting point, picking a point that is free from each dual bound $(y_L, z_L) \geq 0$ and $(y_U, z_U) \leq 0$, a point for which every component is equal to one of its bounds, and the point that solves a separable approximation to (44) in which H^{D} is replaced by the zero or identity matrix; our experiments start with all dual values at zero.

Both folklore and empirical evidence [62] on bound-constrained QPs suggest that the dual active set $\mathcal{A}^{(k)}$ (equivalently the fixed variables/constraints at the Cauchy point) changes rapidly. Thus solving (45) (or finding its Fredholm alternative) is unlikely to benefit from matrix factorization updating techniques [33] usually associated with active-set methods, and it is better to solve successive systems (45) *ab initio*. While we can confirm this behavior on well-conditioned problems, our experience with more difficult ones is that although there can be rapid changes in the active set between iterations, in many cases the active set changes gradually in some phases of its iteration history, especially towards the end. Thus while our initial instinct was not to provide special code to cope with gradual active-set changes, we are now convinced that there should be some provision to update factorizations if requested.

Specifically, during the kth iteration of DQP, if the coefficient matrix for (45) is of full rank (thus we are not required to seek a Fredholm alternative), and if the *change* in the active set at iteration $k+1$ is modest, we may use the Schur-complement updating technique [7,32,33,42] implemented as SCU in GALAHAD to solve (45) at this new iteration rather than resorting to refactorization. The details are quite standard [42, §3], but we choose to remove constraints that are present in $\mathcal{A}^{(k)}$ but not in $\mathcal{A}^{(k+1)}$ before we add constraints that are in $\mathcal{A}^{(k+1)}$ but not in $\mathcal{A}^{(k)}$ since removing constraints maintains full rank, while adding them may result in a rank-deficient system. If rank-deficiency is detected, the Schur-complement update is abandoned, and the Fredholm alternative is sought instead. In addition, we limit the size of the Schur complement

[2] Available from http://galahad.rl.ac.uk/galahad-www/ along with a Matlab interface.

to 100 new rows (by default) before refactorization; setting the limit to zero disables Schur-complement updating. We stop each run when

$$\|P_{\mathcal{D}}[v - \nabla q^{\mathrm{D}}(v)] - v\| \leq \max(\varepsilon_a, \varepsilon_r \cdot \|P_{\mathcal{D}}[v_0 - \nabla q^{\mathrm{D}}(v_0)] - v_0\|), \tag{57}$$

where $\varepsilon_a = 10^{-6}$ and $\varepsilon_r = 10^{-16}$, so long as a limit of 30 min or 10,000 iterations has not already been reached.

Our tests involved the quadratic programming examples from the combined CUTEst [39] and Maros and Meszaros [59] test sets—multiple instances of similar nature were excluded. We only considered the 68 strictly convex problems, which ranged from the smallest (`DUAL4`) with $n = 75$ and $m = 1$ to the largest (`QPBAND`) with $n = 50{,}000$ and $m = 25{,}000$; note that most instances have additional simple-bound constraints on the primal variables, as shown in the general formulation (40). (See Table 1 in the Supplementary Material for additional details.) A problem was identified as being strictly convex by using the GALAHAD package SLS, which provides a common interface to a variety of solvers, from the Harwell Subroutine Library (HSL) and elsewhere, for dense and sparse symmetric linear systems of equations. We used the HSL solver MA97 [55]—a direct method designed for solving sparse symmetric systems—to determine if the Hessian matrix H was numerically positive definite.

All numerical experiments were performed on eight cores of a workstation comprised of thirty-two Intel Xeon ES-2687W CPUs (3.1GHz, 1200MHz FSB, 20MB L3 Cache) with 65.9 GiB of RAM. GALAHAD and its dependencies were compiled in double precision with gfortran 4.6 using fast (`-O3`) optimization and OpenMP enabled (`-fopenmp`).

5.2 Evaluation of DQP

We first considered the performance of DQP when approximate solutions to the subspace subproblem were computed using the iterative CG method (see Sect. 2.3.3). The complete set of detailed results can be found in Table 1 of the Supplementary Material. This instance of DQP failed on 12 problems, where a failure means that the termination test (57) was never achieved. Of the 12 failures, 10 were because the maximum allowed iteration limit was reached and 2 was because the maximum allotted time limit was reached.

For comparison, we also considered the performance of DQP in the case when high accuracy solutions to the subspace subproblem were computed using factorizations (see Sects. 2.3.1 and 2.3.2). The complete set of results can be found in Table 2 of the Supplementary Material. (Note that problems `FIVE20B` and `FIVE20C` were excluded because of an error in the factorization subroutine.) We can see that this instance of DQP failed on 25 problems, which is twice as many as when CG was used. However, note that 18 of these failures were because the maximum allowed time limit was reached, 2 because the maximum allowed iterations was reached, and 5 because local infeasibility was detected. Thus we conclude that the degradation in

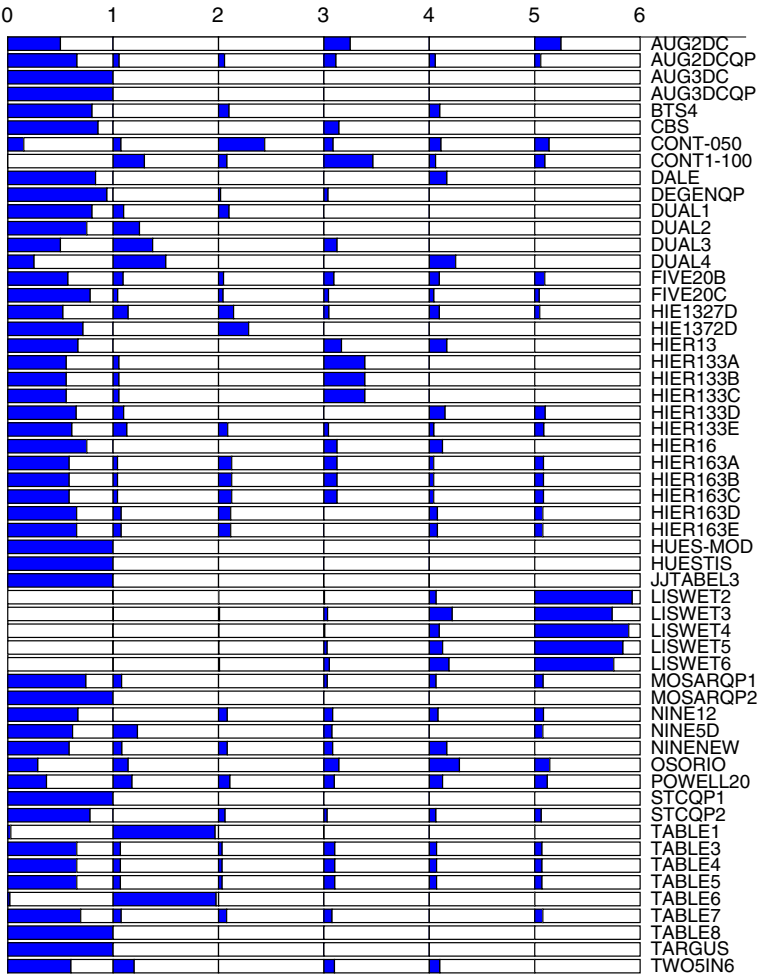

Fig. 1 A comparison of iterations for DQP when using an iterative subproblem solver

performance is primarily due to reaching the time limit, which is a consequence of using the more expensive factorizations in lieu of CG.

Interestingly, between the two variants of DQP, the only problems not solved were `CONT1-200`, `LASER`, and `QPBAND`. Together with our previous discussion, this highlights the trade-off between using iterative and direct methods for solving the subspace subproblem. Namely, that the direct methods tend to be more expensive and more frequently struggle to solve problems in the time allotted, while iterative methods are much cheaper per iteration but more frequently find it difficult to achieve the requested accuracy.

The previous paragraph motivates us to investigate how effectively DQP can obtain approximate solutions for various levels of accuracy. To answer this problem we tracked the iterates computed by DQP and saved the total number of iterations and

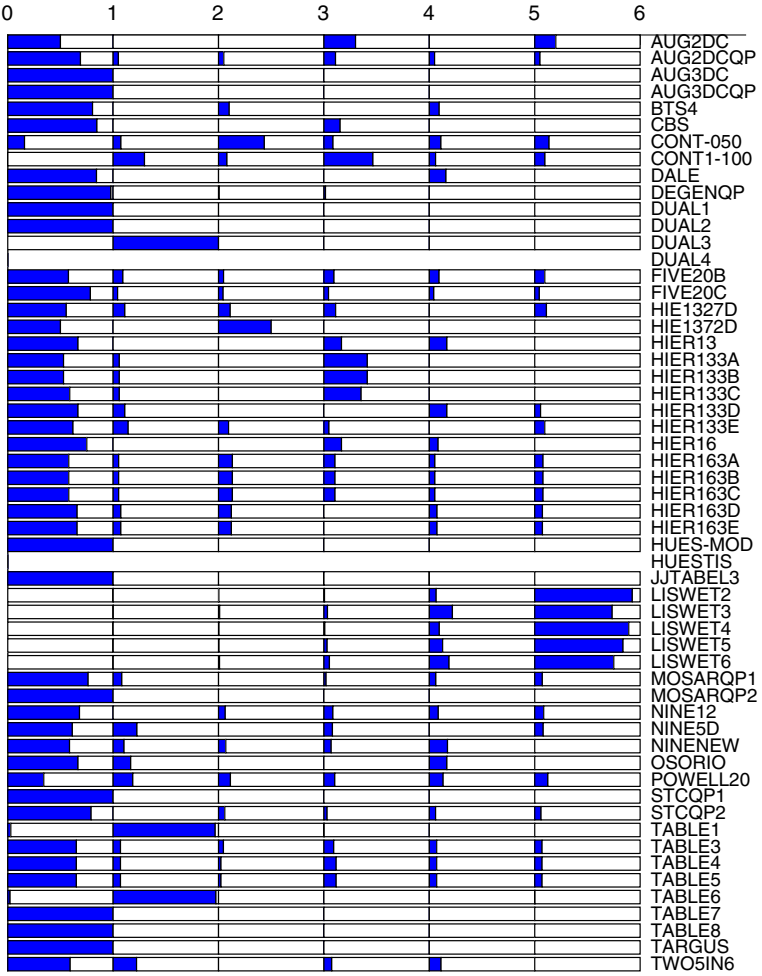

Fig. 2 A comparison of times for DQP when using an iterative subproblem solver

total time required to satisfy the termination condition (57) for the values $\epsilon_r = 0$ and $\epsilon_a \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. The full set of results for when CG is used as the subproblem solver is given in Table 4 of the Supplementary Material, while the results for the case when factorizations are used can be found in Table 5 of the Supplementary Material. For illustrative purposes, we represent the data found in these tables in the form of stacked bar graphs in Figs. 1 and 2 for the CG case, and Figs. 3 and 4 for the factorization case. (We only include problems that DQP was able to achieve the finest stopping tolerance of 10^{-6}.) These plots have a stack of 6 rectangles for each test problem. For each of these 6 rectangles, the fraction filled with blue represents the fraction of the total iterations (Figs. 1 and 3) or total time (Figs. 2 and 4) needed to achieve the various accuracy levels: the jth stacked block (counting

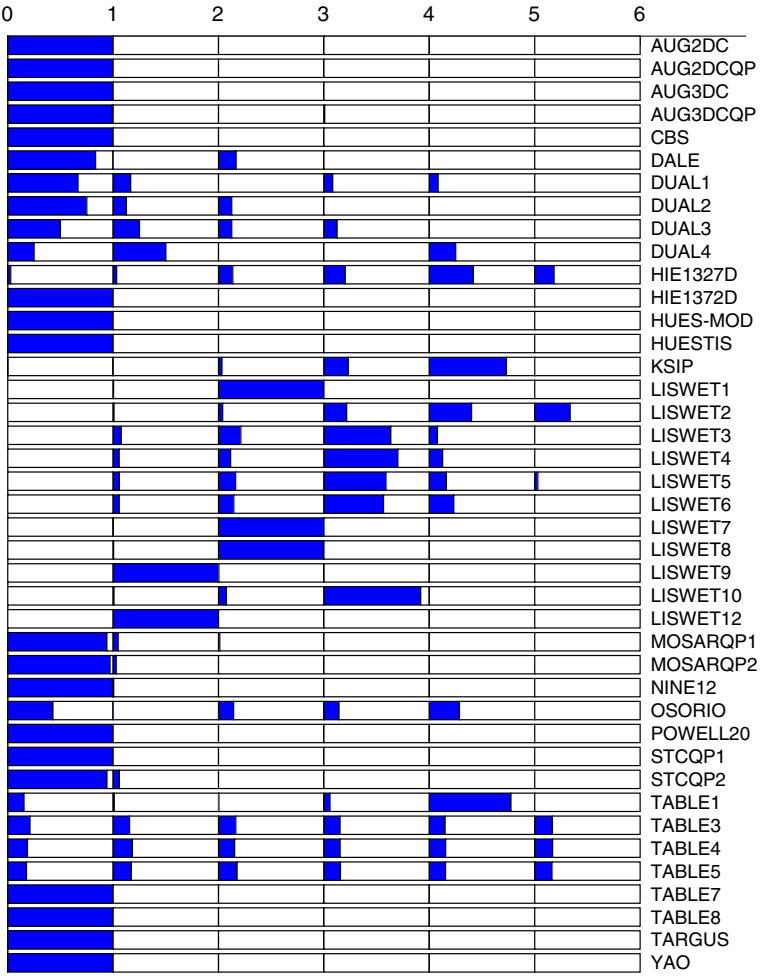

Fig. 3 A comparison of iterations for DQP when using a direct method to solve each subproblem

from left to right) for each problem corresponds to the accuracy level 10^{-j} for each $j \in \{1, 2, 3, 4, 5, 6\}$.

To help clarify the meaning of the figures, we describe a few illustrative examples. We comment that additional details for any problem can be obtained from Section 3 of the Supplementary Material.

– NINENEW From Section 3 of the Supplementary material, we see that the number of iterations required to reach the 6 different tolerance levels are 7, 8, 9, 10, 12, and 12, respectively. This means that it took 7 iterations to reach the tolerance level 10^{-1}, 8 iterations to reach the tolerance level 10^{-2}, 9 iterations to reach the tolerance level 10^{-3}, 10 iterations to reach the tolerance level 10^{-4}, and then iteration 12 was the first to fall below 10^{-5} and, in fact, it also fell below 10^{-6}.

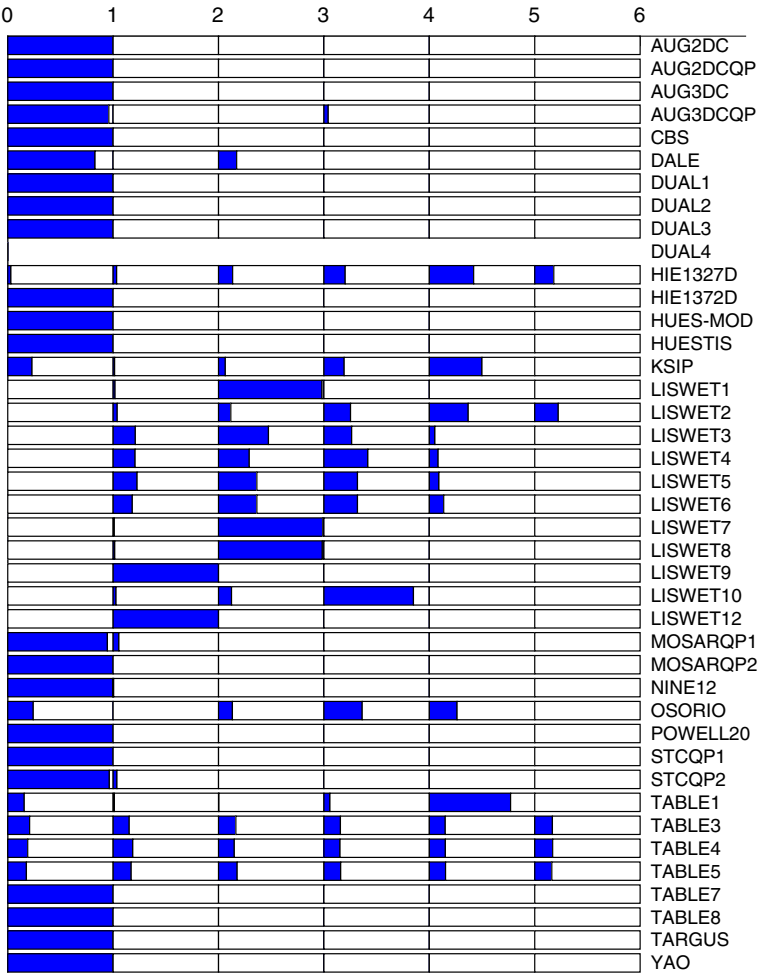

Fig. 4 A comparison of times for DQP when using a direct method to solve each subproblem

– `LISWET2` From Section 3 of the Supplementary material, we have that the number of iterations required to reach the 6 different tolerance levels are 0, 4, 22, 54, 441, and 5920, respectively. The initial point satisfied the tolerance 10^{-1} but not 10^{-2}. The level of 10^{-2} was reached by iteration 4. Bars 2–6 are *all* nonempty, but since 5920 iterations were needed to achieve the final accuracy of 10^{-6}, bars 2–4 appear empty to the eye.
– `HUESTIS` From Section 3 of the Supplementary material, we see that the number of iterations needed to reach the 6 different tolerance levels are 4, 4, 4, 4, 4, and 4, respectively. This means the 4th iterate was the first to satisfy every tolerance, i.e., the errors associated with the first 3 iterates were all greater than 10^{-1}, and then iterate 4 had an error of less than 10^{-6}. Such performance is not completely uncommon for active-set methods.

Figures 1 and 2 for DQP when CG was used as the subspace solver clearly show a very tight relationship between the number of iterations and times required to reach the 6 different accuracies. In fact, for most problems, the difference in the bars for the fraction of iterations (Fig. 1) and the times (Fig. 2) are indistinguishable; two exceptions are DUAL3 and TABLE7. We also remark that two problems in Fig. 2 do not have any stacked bars because no significant time was needed to reach the final desired accuracy. As expected, observe that in most cases the majority of iterations are needed to reach the first optimality tolerance of 10^{-1}, although LISWET2–LISWET6 are exceptions.

Figures 3 and 4, which are based on using factorizations to solve the subspace subproblem, also illustrate the close relationship between the number of iterations (Fig. 3) and times required (Fig. 4) to reach the 6 different accuracy levels; there is somewhat more variability here when compared to using CG. As before, there is one problem (DUAL4) in Fig. 4 that does not have any stacked bars because no significant time was needed to reach the final desired accuracy. One can also observe that the figures associated with the use of a direct method (Figs. 3 and 4) tend to be "denser" towards the bottom when compared to the use of CG (Figs. 1 and 2). This is perhaps no surprise since it is often the case that only a couple of iterations are required to obtain a high accuracy solution for direct methods once the active set at the solution has been identified; this appears to often be the case once the tolerance of 10^{-2} is reached. We also mention that obtaining high accuracy solutions for problems LISWET2–LISWET6 seems to be somewhat challenging, much as we observed when CG was used.

Overall, we are satisfied with these results. They clearly show the trade-off that exists between using iterative and direct subproblem solvers. By allowing for the use of both, we are able to solve 65 of the 68 problems to an accuracy of at least 10^{-6}.

6 Final comments and conclusions

We presented the details of a solver for minimizing a strictly convex quadratic objective function subject to general linear constraints. The method uses a gradient projection strategy enhanced by subspace acceleration calculations to solve the bound-constrained dual optimization problem. The main contributions of this work are threefold. First, we address the challenges associated with solving the dual problem, which is usually a convex problem even when the primal problem is strictly convex. Second, we show how the linear algebra may be arranged to take computational advantage of *sparsity* that is often present in the second-derivative matrix. In particular, we consider the case that the second-derivative matrix is explicitly available and sparse, and the case when it is available implicitly via a limited memory BFGS representation. Third, we present the details of our Fortran 2003 software package DQP, which is part of the GALAHAD suite of optimization routines. Numerical tests showed the trade-off between using an iterative subproblem solver versus a direct factorization method. In particular, iterative subproblem solvers are typically computationally cheaper per iteration but less reliable at achieving high accuracy solutions, while direct methods are typically more expensive per iteration but more reliable at

obtaining high accuracy solutions given enough time. Both options are available in the package DQP.

The numerical results showed that DQP is often able to obtain high accuracy solutions, and is very reliable at obtaining low accuracy solutions. This latter fact makes DQP an attractive option as the subproblem solver in the recently developed inexact SQO method called iSQO [19]. The solver iSQO is one of the few SQO methods that allows for inexact subproblem solutions, a feature that is paramount for large-scale problems. The conditions that iSQO requires to be satisfied by an approximate subproblem solution can readily be obtained by DQP.

Although not presented in this paper, we experimented with using DQP as the second stage of a cross-over method with the first stage being an interior-point solver. Specifically, we used the GALAHAD interior-point solver CQP and changed over to DQP once the optimality measures were below 10^{-2}. Our tests showed that DQP was not especially effective in this capacity. The reason seemed to be because CQP [37] was designed to be effective on degenerate problems and have the capacity of obtaining high accuracy solutions; this was achieved by using nonstandard parameterizations and high-order Taylor approximations of the central path. As a consequence, we believe that CQP remains the best solver for solving QP problems when good estimates of the solution are not known in advance. However, we still believe that when solving a sequence of QP problems (e.g., in SQO methods) or more generally anytime a good solution estimate is available, the method DQP is an attractive option.

In terms of cross-over methods, one could also consider using DQP as a first stage solver that provides a starting point to a traditional active-set method. Although we have not yet used DQP in this capacity, it does have the potential to be efficient and more stable than using DQP alone. The potential for improved stability is because the performance of DQP is tied to its ability to identify the optimal active set. Although such a feature holds under certain assumptions for gradient projection methods, performance often steeply degrades when such assumptions do not hold; this is typically not true of a traditional active-set method.

Finally, although duality via (3) holds more generally when H is positive semi-definite, it is not always then possible to eliminate the variables x as in (4) to arrive at a dual with simple non-negativity constraints. In particular, the dual may involve additional general linear inequality constraints on y, and projection into this region may prove to be very expensive—indeed, the projection may itself be posed as a strictly-convex QP. Fortunately for problems involving regularization, such as those discussed in Sect. 4, very minor modifications are required to fit our basic framework.

Acknowledgements The authors are very grateful to Iain Duff for providing extensions to `MA57` to cope with both sparse forward solution and the Fredholm alternative, and to Jonathan Hogg and Jennifer Scott for doing the same for `MA77` and `MA97`. We are also grateful to Iain, Jonathan, Jennifer, Mario Arioli and Tyrone Rees for helpful discussions on Fredholm issues. N.I.M. Gould's research supported by EPSRC Grants EP/I013067/1 and EP/M025179/1.

References

1. Arioli, M., Laratta, A., Menchi, O.: Numerical computation of the projection of a point onto a polyhedron. J. Optim. Theory Appl. **43**(4), 495–525 (1984)

2. Axehill, D. Hansson, A.: A dual gradient projection quadratic programming algorithm tailored for model predictive control. In: Proceedings of the 47th IEEE Conference on Decision and Control, pp. 3057–3064, Cancun (2008)
3. Bartlett, R.A., Biegler, L.T.: QPSchur: a dual, active-set, Schur-complement method for large-scale and structured convex quadratic programming. Optim. Eng. **7**(1), 5–32 (2006)
4. Benzi, M., Golub, G.H., Liesen, J.: Numerical solution of saddle point problems. Acta Numer. **14**, 1–137 (2005)
5. Betts, J.T., Frank, P.D.: A sparse nonlinear optimization algorithm. J. Optim. Theory Appl. **82**, 519–541 (1994)
6. Bierlaire, M., Toint, P.L., Tuyttens, D.: On iterative algorithms for linear least squares problems with bound constraints. Linear Algebra Appl. **143**, 111–143 (1991)
7. Bisschop, J., Meeraus, A.: Matrix augmentation and partitioning in the updating of the basis inverse. Math. Program. **13**(3), 241–254 (1977)
8. Björck, Å.: Numerical Methods for Least Squares Problems. SIAM, Philadelphia (1996)
9. Boggs, P.T., Tolle, J.W.: Sequential quadratic programming. Acta Numer. **4**, 1–51 (1995)
10. Boland, N.L.: A dual-active-set algorithm for positive semi-definite quadratic programming. Math. Program. Ser. A **78**(1), 1–27 (1997)
11. Byrd, R.H., Nocedal, J., Schnabel, R.B.: Representations of quasi-Newton matrices and their use in limited memory methods. Math. Program. **63**(2), 129–156 (1994)
12. Calamai, P.H., Moré, J.J.: Projected gradient methods for linearly constrained problems. Math. Program. **39**(1), 93–116 (1987)
13. Conn, A.R., Gould, N.I.M.: On the location of directions of infinite descent for nonlinear programming algorithms. SIAM J. Numer. Anal. **21**(6), 302–325 (1984)
14. Conn, A.R., Gould, N.I.M., Toint, P.L.: Global convergence of a class of trust region algorithms for optimization with simple bounds. SIAM J. Numer. Anal. **25**(2), 433–460 (1988). See also same journal **26**, 764–767 (1989)
15. Conn, A.R., Gould, N.I.M., Toint, P.L.: Testing a class of methods for solving minimization problems with simple bounds on the variables. Math. Comput. **50**, 399–430 (1988)
16. Conn, A.R., Gould, N.I.M., Toint, P.L.: Trust-Region Methods. SIAM, Philadelphia (2000)
17. Curtis, F.E., Han, Z.: Globally Convergent Primal-Dual Active-Set Methods with Inexact Subproblem Solves. Technical Report 14T-010, COR@L Laboratory, Department of ISE, Lehigh University, 2014. In second review for SIAM Journal on Optimization
18. Curtis, F.E., Han, Z., Robinson, D.P.: A globally convergent primal-dual active-set framework for large-scale convex quadratic optimization. Comput. Optim. Appl. (2014). doi:10.1007/s10589-014-9681-9
19. Curtis, F.E., Johnson, T.C., Robinson, D.P., Wachter, A.: An inexact sequential quadratic optimization algorithm for nonlinear optimization. SIAM J. Optim. **24**(3), 1041–1074 (2014)
20. Dominguez, J., González-Lima, M.D.: A primal-dual interior-point algorithm for quadratic programming. Numer. Algorithms **42**(1), 1–30 (2006)
21. Dorn, W.: Duality in quadratic programming. Q. Appl. Math. **18**, 155–162 (1960)
22. Dostál, Z.: Optimal Quadratic Programming Algorithms: With Applications to Variational Inequalities. Springer Optimization and Its Applications, vol. 23. Springer, New York (2009)
23. Dostál, Z., Schöberl, J.: Minimizing quadratic functions subject to bound constraints with the rate of convergence and finite termination. Comput. Optim. Appl. **30**(1), 23–43 (2005)
24. Duff, I.S.: MA57—a code for the solution of sparse symmetric definite and indefinite systems. ACM Trans. Math. Softw. **30**(2), 118–144 (2004)
25. Duff, I.S., Reid, J.K.: The multifrontal solution of indefinite sparse symmetric linear equations. ACM Trans. Math. Softw. **9**(3), 302–325 (1983)
26. Fletcher, R.: An ℓ_1 penalty method for nonlinear constraints. In: Boggs, P.T., Byrd, R.H., Schnabel, R.B. (eds.) Numerical Optimization 1984, pp. 26–40. SIAM, Philadelphia (1985)
27. Forsgren, A., Gill, P.E., Wong, E.: Primal and dual active-set methods for convex quadratic programming. Math. Program. **159**(1), 469–508 (2016)
28. Friedlander, M.P., Gould, N.I.M., Leyffer, S., Munson, T.: A filter active-set trust-region method. Technical Report Preprint ANL/MCS-P1456-0907, Argonne National Laboratory, Illinois (2007)
29. Friedlander, M.P., Leyffer, S.: Global and finite termination of a two-phase augmented Lagrangian filter method for general quadratic programs. SIAM J. Sci. Comput. **30**(4), 1706–1729 (2008)
30. Friedlander, M.P., Orban, D.: A primal-dual regularized interior-point method for convex quadratic programs. Math. Program. Comput. **4**(1), 71–107 (2012)

31. George, A., Liu, J.W.H.: Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall, Englewood Cliffs (1981)
32. Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H.: A Schur-complement method for sparse quadratic programming. In: Cox, M.G., Hammarling, S.J. (eds.) Reliable Scientific Computation, pp. 113–138. Oxford University Press, Oxford (1990)
33. Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H.: Inertia-controlling methods for general quadratic programming. SIAM Rev. **33**(1), 1–36 (1991)
34. Gill, P.E., Murray, W., Saunders, M.A.: User's guide for QPOPT 1.0: a Fortran package for quadratic programming. Report SOL 95-4, Department of Operations Research, Stanford University, Stanford (1995)
35. Gill, P.E., Murray, W., Saunders, M .A., Wright, M .H.: A Schur-complement method for sparse quadratic programming. In: Cox, M .G., Hammarling, S .J. (eds.) Reliable Numerical Computation, pp. 113–138. Oxford University Press, Oxford (1990)
36. Goldfarb, D., Idnani, A.: A numerically stable dual method for solving strictly convex quadratic programs. Math. Program. **27**(1), 1–33 (1983)
37. Gould, N.I.M., Orban, D., Robinson, D.P.: Trajectory-following methods for large-scale degenerate convex quadratic programming. Math. Program. Comput. **5**(2), 113–142 (2013)
38. Gould, N.I.M., Orban, D., Toint, P.L.: GALAHAD—a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. ACM Trans. Math. Softw. **29**(4), 353–372 (2003)
39. Gould, N.I.M., Orban, D., Toint, P.L.: CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. Comput. Optim. Appl. **60**(3), 545–557 (2015)
40. Gould, N.I.M., Toint, P.L.: A quadratic programming bibliography. Numerical Analysis Group Internal Report 2000-1, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2000. See http://www.numerical.rl.ac.uk/qp/qp.html
41. Gould, N.I.M., Toint, P.L.: SQP methods for large-scale nonlinear programming. In: Powell, M.J.D., Scholtes, S. (eds.) System Modelling and Optimization. Methods, Theory and Applications, pp. 149–178. Kluwer Academic Publishers, Dordrecht (2000)
42. Gould, N.I.M., Toint, P.L.: An iterative working-set method for large-scale non-convex quadratic programming. Appl. Numer. Math. **43**(1–2), 109–128 (2002)
43. Gould, N.I.M., Toint, P.L.: Numerical methods for large-scale non-convex quadratic programming. In: Siddiqi, A.H., Kočvara, M. (eds.) Trends in Industrial and Applied Mathematics, pp. 149–179. Kluwer Academic Publishers, Dordrecht (2002)
44. Gould, N.I.M., Loh, Y., Robinson, D.P.: A filter method with unified step computation for nonlinear optimization. SIAM J. Optim. **24**(1), 175–209 (2014)
45. Gould, N.I.M., Loh, Y., Robinson, D.P.: A filter SQP method: local convergence and numerical results. SIAM J. Optim. **25**(3), 1885–1911 (2015)
46. Gould, N.I.M., Robinson, D.P.: A second derivative SQP method: global convergence. SIAM J. Optim. **20**(4), 2023–2048 (2010)
47. Gould, N.I.M., Robinson, D.P.: A second derivative SQP method: local convergence and practical issues. SIAM J. Optim. **20**(4), 2049–2079 (2010)
48. Gould, N.I.M., Robinson, D.P.: A second derivative SQP method with a 'trust-region-free' predictor step. IMA J. Numer. Anal. **32**(2), 580–601 (2012)
49. Gu, Z., Rothberg, E., Bixby, R.: Gurobi Optimizer, version 5.5. 0. Software program (2013)
50. Gupta, A.: WSMP: Watson Sparse Matrix Package Part I—Direct Solution of Symmetric Sparse System. Research Report RC 21886, IBM T. J. Watson Research Center, Yorktown Heights (2010)
51. Hager, W.W., Hearn, D.W.: Application of the dual active set algorithm to quadratic network optimization. Comput. Optim. Appl. **1**(4), 349–373 (1993)
52. Hintermüller, M., Kunisch, K.: The primal-dual active set strategy as a semismooth Newton method. SIAM J. Optim. **13**(3), 865–888 (2002). (electronic) (2003)
53. Hogg, J.D., Reid, J.K., Scott, J.A.: Design of a multicore sparse Cholesky factorization using DAGs. SIAM J. Sci. Comput. **32**(6), 36273649 (2010)
54. Hogg, J.D., Scott, J.A.: An indefinite sparse direct solver for large problems on multicore machines. Technical Report RAL-TR-2010-011, Rutherford Appleton Laboratory, Chilton (2010)
55. Hogg, J.D., Scott, J.A.: HSL_MA97: a bit-compatible multifrontal code for sparse symmetric systems. Technical Report RAL-TR-2011-024, Rutherford Appleton Laboratory, Chilton (2011)
56. ILOG CPLEX. High-performance software for mathematical programming and optimization (2005)

57. Lancaster, P., Tismenetsky, M.: The Theory of Matrices: With Applications, 2nd edn. Academic Press, London (1985)
58. Lescrenier, M.: Convergence of trust region algorithms for optimization with bounds when strict complementarity does not hold. SIAM J. Numer. Anal. **28**(2), 476–495 (1991)
59. Maros, I., Meszaros, C.: A repository of convex quadratic programming problems. Optim. Methods Softw. **11–12**, 671–681 (1999)
60. Mészáros, C.: The BPMPD interior point solver for convex quadratic problems. Optim. Methods Softw. **11**(1–4), 431–449 (1999)
61. Morales, J.L., Nocedal, J., Wu, Y.: A sequential quadratic programming algorithm with an additional equality constrained phase. IMA J. Numer. Anal. **32**, 553–579 (2012)
62. Moré, J.J., Toraldo, G.: On the solution of large quadratic programming problems with bound constraints. SIAM J. Optim. **1**(1), 93–113 (1991)
63. Moré, J.J., Thuente, D.J.: Line search algorithms with guaranteed sufficient decrease. ACM Trans. Math. Softw. **20**(3), 286–307 (1994)
64. Moyh-ud Din, H., Robinson, D.P.: A solver for nonconvex bound-constrained quadratic optimization. SIAM J. Optim. **25**(4), 2385–2407 (2015)
65. Nocedal, J., Wright, S.J.: Numerical Optimization. Series in Operations Research, 2nd edn. Springer, Heidelberg (2006)
66. Polyak, R.A., Costa, J., Neyshabouri, S.: Dual fast projected gradient method for quadratic programming. Optim. Lett. (2012). doi:10.1007/s11590-012-0476-6
67. Powell, M.J.D.: ZQPCVX a FORTRAN subroutine for convex quadratic programming. University, Department of Applied Mathematics and Theoretical Physics (1983)
68. Reid, J.K., Scott, J.A.: An out-of-core sparse Cholesky solver. ACM Trans. Math. Softw. **36**(2), 9 (2009)
69. Robinson, D.P., Feng, L., Nocedal, J., Pang, J.-S.: Subspace accelerated matrix splitting algorithms for asymmetric and symmetric linear complementarity problems. SIAM J. Optim. **23**(3), 1371–1397 (2013)
70. Schenk, O., Gärtner, K.: On fast factorization pivoting methods for symmetric indefinite systems. Electron. Trans. Numer. Anal. **23**, 158–179 (2006)
71. Schmid, C., Biegler, L.T.: Quadratic programming methods for reduced hessian sqp. Comput. Chem. Eng. **18**(9), 817–832 (1994)
72. Spellucci, P.: Solving general convex QP problems via an exact quadratic augmented Lagrangian with bound constraints. Techn. Hochsch, Fachbereich Mathematik (1993)
73. Stefanov, S.M.: Polynomial algorithms for projecting a point onto a region defined by a linear constraint and box constraints in $\mathbb{R}^n$. J. Appl. Math. **2004**(5), 409–431 (2004)
74. Vanderbei, R.J.: LOQO: an interior point code for quadratic programming. Optim. Methods Softw. **11**(1–4), 451–484 (1999)
75. Williams, J.W.J.: Algorithm 232. Heapsort. Commun. ACM **7**, 347–348 (1964)
76. Yuan, G., Lu, S., Wei, Z.: A modified limited SQP method for constrained optimization. Appl. Math. **1**(1), 8–17 (2010)
77. Zhu, C., Rockafellar, R.T.: Primal-dual projected gradient algorithms for extended linear-quadratic programming. SIAM J. Optim. **3**(4), 751–783 (1993)