A Dual Gradient-Projection Method for Large-Scale Strictly Convex Quadratic Problems

Nicholas I. M. Gould * and Daniel P. Robinson

February 18, 2016

Abstract

The details of a solver for minimizing a strictly convex quadratic objective function subject to general linear constraints is 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 three-fold. 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 suite of optimization routines. Numerical tests are performed on quadratic programming problems from the combined CUTEst and Maros and Meszaros test sets.

1 Introduction

Quadratic problems 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 [43]. 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 quadratic problems [10, 44]. Traditionally, quadratic problems have been solved by either active-set or interior-point methods [68, Ch.6], but in certain cases gradient-projection methods may be preferred, which is the focus of this manuscript.

We consider the strictly convex quadratic problem (QP)

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad q(x) := \tfrac{1}{2}\langle x, Hx\rangle + \langle g, x\rangle \;\text{ such that }\; Ax \geq c, \tag{1.1}$$

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 [43] or the inverse of H is represented in the form

$$H^{-1} = H_0^{-1} + VWV^T \tag{1.2}$$

*This author was supported by EPSRC grants EP/I013067/1 and EP/M025179/1.

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 [12]. Matrices of the form (1.2) often arise as limited-memory secant Hessian approximations in SQP methods [47–51, 64, 79].

It is well known [22] that (1.1) has a related primal-dual problem

$$\underset{(x,y)\in\mathbb{R}^{n+m}}{\text{maximize}} \; -\tfrac{1}{2}\langle x, Hx\rangle + \langle c, y\rangle \;\text{ such that }\; Hx - A^T y + g = 0 \;\text{ and }\; y \geq 0 \tag{1.3}$$

for which the optimal objective values of (1.1) and (1.3) are equal. After eliminating x from (1.3), an optimal dual vector y_* of (1.3) is a solution to the dual problem

$$y_* = \underset{y\in\mathbb{R}^m}{\arg\min}\; q^{\text{D}}(y) := \tfrac{1}{2}\langle y, H^{\text{D}} y\rangle + \langle g^{\text{D}}, y\rangle \;\text{ such that }\; y \geq 0, \tag{1.4}$$

where

$$H^{\text{D}} := AH^{-1}A^T \;\text{ and }\; g^{\text{D}} := -AH^{-1}g - c. \tag{1.5}$$

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. Notice that (1.1) is infeasible if and only if (1.4) 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 quadratic problem (1.1). 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 [21, 52, 59, 63, 77] 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 shifted primal-dual linear system of equations. For problems that are convex but perhaps not strictly convex, regularized formulations have been proposed [32] 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 their cheap iteration cost, but are difficult to be warm-started, i.e., to solve problem (1.1) 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 (1.1), which is potentially problematic when the number of constraints is large. Primal feasible active-set methods [6, 37, 38, 46, 52, 59] require a primal-feasible point that is often costly to obtain, whereas a trivial feasible point exists for dual feasible active-set methods [4, 11, 39, 70, 74]. We also mention that non-traditional active-set methods have recently been developed primarily motivated by solving certain problems that arise in optimal control [18, 19, 55], and that dual active-set methods have also received attention [29, 54].

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 [75] 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 (1.1). 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 Friendlander and Leyffer [31]. 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 [23,24,65–67,72]. 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 liner-quadratic problems that arise in dynamic and stochastic programming [80]. In this setting, however, both the primal and dual problems have nonsmooth second derivatives. A second example is the dual projection method [3], which focuses on the special structure of the QPs used to solve mixed-integer problems in predictive control. The projection method in [69] 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.

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) (1.4) is fairly standard [13, 16, 65] 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 (1.4). The well-studied gradient projection method for BQP is given by Algorithm 2.1.

Algorithm 2.1 Accelerated gradient projection method for solving the BQP (1.4).

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 $\alpha^{\mathrm{C}} = \operatorname{argmin}_{\alpha>0} q^{\mathrm{D}}(P_{\mathcal{D}}[y + \alpha d])$.
   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 $\Delta y^{\mathrm{S}} = \operatorname{argmin}_{\Delta y \in \mathbb{R}^m} q^{\mathrm{D}}(y^{\mathrm{C}} + \Delta y)$ such that $[\Delta y]_{\mathcal{A}} = 0$.
3. **(improved subspace point)**
   Select $\alpha_{\max} > 0$ and then compute $\alpha^{\mathrm{S}} = \operatorname{argmin}_{\alpha\in[0,\alpha_{\max}]} q^{\mathrm{D}}(P_{\mathcal{D}}[y^{\mathrm{C}} + \alpha\Delta y^{\mathrm{S}}])$.
   Set $y = P_{\mathcal{D}}[y^{\mathrm{C}} + \alpha^{\mathrm{S}}\Delta y^{\mathrm{S}}]$.

end while

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 [15, 61, 65]. 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 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 2.1

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 [16, §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 nonzeros 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.

Algorithm 2.2 Finding the Cauchy point (preliminary version).

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

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

Compute the index sets $\mathcal{I}^0 = \{j : \alpha_j^{\text{B}} = 0\}$ 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 $\ell^0 = \langle \nabla g^{\text{D}}(y), d^0 \rangle$ and $H^{\text{D}} d^0$.
Set $q_0' = \ell^0$ and compute $q_0'' = \langle d^0, H^{\text{D}} d^0 \rangle$.
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)**
   Set $\Delta\alpha := -q_i'/q_i''$.
   if $q_i' \geq 0$, **then** return the exact Cauchy point $y^{\text{C}} = y + s^i$. **end if**
   if $q_i'' > 0$ and $\Delta\alpha < \Delta\alpha^i$, **then** return the exact Cauchy point $y^{\text{C}} = y + s^i + \Delta\alpha d^i$. **end if**
3. **(prepare for the next interval)**
   Compute the index sets $\mathcal{I}^{i+1} = \{j : \alpha_j^{\text{B}} = \alpha^{i+1}\}$ 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 $H^{\text{D}} e^{i+1}$ and $\langle \nabla g^{\text{D}}(y), e^{i+1} \rangle$.
   Update $\ell^{i+1} = \ell^i - \langle \nabla g^{\text{D}}(y), e^{i+1} \rangle$ and $H^{\text{D}} d^{i+1} = H^{\text{D}} d^i - H^{\text{D}} e^{i+1}$.
4. **(compute the slope and curvature for the next interval)**
   Use $H^{\text{D}} d^{i+1}$ to compute $\gamma^{i+1} = \langle s^{i+1}, H^{\text{D}} d^{i+1} \rangle$, $q_{i+1}' = \ell^{i+1} + \gamma^{i+1}$, and $q_{i+1}'' = \langle d^{i+1}, H^{\text{D}} d^{i+1} \rangle$.

end for

The details of the search along the piecewise linear path is given as Algorithm 2.2, which is based on [17, 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 2.2 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.

2.1.1 Sparse H

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

$$H = LL^T, \tag{2.1}$$

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 2.3), 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 need be 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 [33]. New subroutines to provide this (sparse right-hand side) functionality have been added to each of the HSL packages `MA57` [25], `MA87` [56] and `MA97` [58]. There may also be some gain on high-performance machines in performing a block solve

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

Algorithm 2.3 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

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

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

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 [78].

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 2.3 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

$$\gamma^{i+1} = \gamma^i + \Delta\alpha^i q_i'' - \langle p^{i+1}, w^{i+1} \rangle \quad \text{and} \quad q_{i+1}'' = q_i'' + \langle w^{i+1} - 2h^i, w^{i+1} \rangle, \tag{2.2}$$

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

Now let

$$u^{i+1} = p^{i+1} - \alpha^{i+1} h^i, \tag{2.3}$$

where $u^0 = 0$. Then it follows from $p^{i+1} = p^i + \Delta\alpha^i h^i$, $\alpha^{i+1} = \alpha^i + \Delta\alpha^i$, and $h^{i+1} = h^i - w^{i+1}$ that

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

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 (2.3) 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 (2.2) 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 2.3 using the above equivalent formulations gives our final method stated as Algorithm 2.4. 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.

2.1.2 Structured H

Suppose that H has the structured form (1.2). We assume that we can cheaply obtain

$$H_0^{-1} = BB^T \tag{2.4}$$

for some sparse matrix B (this is trivial in the common case that H^{-1} is diagonal). Now, consider Algorithm 2.2 and define

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

The key, as in §2.1.1, is to efficiently compute

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

The terms $\langle p_{\text{V}}^i, W h_{\text{V}}^i \rangle$ and $\langle h_{\text{V}}^i, W h_{\text{V}}^i \rangle$ 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 the required vectors h_{V}^{i+1} and p_{V}^{i+1} may be recurred via

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

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

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

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

$$h_{\text{B}}^{i+1} = h_{\text{B}}^i - w_{\text{B}}^{i+1} \quad \text{and} \quad p_{\text{B}}^{i+1} = p_{\text{B}}^i + \Delta\alpha^i h_{\text{B}}^i, \quad \text{where} \quad w_{\text{B}}^{i+1} = B^T r^{i+1}. \tag{2.5}$$

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 (2.2) to see that

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

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

$$u_{\text{B}}^{i+1} = u_{\text{B}}^i + \alpha^i w_{\text{B}}^i \tag{2.6}$$

Algorithm 2.4 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

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

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)**
   Set $\Delta\alpha := -q'_i/q''_i$.
   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
   if $q'_i \geq 0$, **then** return the exact Cauchy point $y^{\mathrm{C}} = y + s$. **end if**
   if $q''_i > 0$ and $\Delta\alpha < \Delta\alpha^i$, **then** return the exact Cauchy point $y^{\mathrm{C}} = y + s + \Delta\alpha d^i$. **end if**
   if $\Delta\alpha^i = \infty$ and $q''_i \leq 0$ **then** the problem is unbounded below. **end if**
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

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 (2.5) and (2.6).

We summarize our findings in Algorithm 2.5.

2.2 An approximate Cauchy point in Algorithm 2.1

In some cases, it may be advantageous to approximate the Cauchy point using a backtracking projected linesearch [65]. 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

$$q^{\mathrm{D}}(y^{i+1}) \leq q^{\mathrm{D}}(y) + \eta \langle \nabla g^{\mathrm{D}}(y), d^{i+1} \rangle \tag{2.7}$$

with

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

Algorithm 2.5 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

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

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

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 that

$$q_{i+1}' = \langle \nabla g^{\mathrm{D}}(y), d^{i+1} \rangle = \langle \nabla g^{\mathrm{D}}(y), y^i - y + \Delta y^i \rangle = q_i' + \langle \nabla g^{\mathrm{D}}(y), \Delta y^i \rangle \quad \text{for } i \geq 1. \tag{2.8}$$

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

$$\mathcal{A} = \{j : y_j = 0 \text{ and } d_j \leq 0\}, \quad \mathcal{F} = \{j : d_j > 0, \text{ or } d_j = 0 < y_j\}, \quad \text{and } \mathcal{U} = \{1, 2, \ldots, m\} \setminus (\mathcal{A} \cup \mathcal{F}), \tag{2.9}$$

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

$$\mathcal{A}^i := \{j \in \mathcal{U} : y_j^i = 0\} \quad \text{and} \quad \mathcal{F}^i := \{j \in \mathcal{U} : y_j^i > 0\} \quad \text{for } i \geq 1, \tag{2.10}$$

as well as the index sets

$$\mathcal{S}_1^i := \mathcal{F}^{i+1} \cap \mathcal{A}^i, \quad \mathcal{S}_2^i := \mathcal{F}^{i+1} \cap \mathcal{F}^i \cap \mathcal{A}^{i-1}, \quad \text{and} \quad \mathcal{S}_3^i := \mathcal{F}^{i+1} \cap \mathcal{F}^i \cap \mathcal{F}^{i-1} \quad \text{for all } i \geq 2. \tag{2.11}$$

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

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

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

$$\Delta y_j^i = \begin{cases} 0 & \text{if } j \in \mathcal{A} \cup \mathcal{A}^{i+1}, \\ \beta \Delta y_j^{i-1} & \text{if } j \in \mathcal{F} \cup \mathcal{S}_3^i, \\ y_j^{i+1} - y_j^i & \text{if } j \in \mathcal{S}_1^i \cup \mathcal{S}_2^i. \end{cases} \tag{2.12}$$

Also, for future reference, note that it follows from (2.12) that

$$\Delta y^i = \beta \Delta y^{i-1} + \delta y^{i-1}, \text{ with } \delta y_j^{i-1} := \begin{cases} 0 & \text{if } j \in \mathcal{A} \cup \mathcal{A}^{i+1} \cup \mathcal{F} \cup \mathcal{S}_3^i, \\ \Delta y_j^i - \beta \Delta y_j^{i-1} & \text{if } j \in \mathcal{S}_1^i \cup \mathcal{S}_2^i, \end{cases} \tag{2.13}$$

where the vector δy^{i-1} is usually a sparse vector. Second, they are useful when computing the inner products that involve Δy^i (e.g., see (2.8)) as shown in Algorithm 2.6. Note that by combining the recursion performed for $g^i = \langle \nabla g^{\mathrm{D}}(y), \Delta y^i \rangle$ in Algorithm 2.6 with (2.8) we obtain an efficient recursion for the sequence $\{q_i'\}$. The other sequence $\{c^i\}$ that is recurred in Algorithm 2.6 will be used when considering the different representations for H in the next two sections.

Algorithm 2.6 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 $g_F^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1 \rangle_{\mathcal{F}}$, $g_1^1 = \langle \nabla g^{\mathrm{D}}(y), \Delta y^1 \rangle_{\mathcal{S}_1^1}$, $g_2^1 = 0$, 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.)
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 $\mathcal{A}^{i+1} = \mathcal{A}^i \setminus \mathcal{C}^i$.
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 $g_F^i = \beta g_F^{i-1}$, $g_1^i = \langle \nabla g^{\mathrm{D}}(y), \Delta y^i \rangle_{\mathcal{S}_1^i}$, $g_2^i = \langle \nabla g^{\mathrm{D}}(y), \Delta y^i \rangle_{\mathcal{S}_2^i}$, 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

Algorithm 2.7 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 $\nabla g^{\text{D}}(y) = Ax - c$.
Solve $Lw = A^T y$, set $v = w - h$, solve $L^T z = v$, and then set $q^{\text{D}}(y) = \frac{1}{2}\langle w, w\rangle - \langle y, c + Az\rangle$.
Define $\mathcal{A}$, $\mathcal{F}$, and $\mathcal{U}$ using (2.9).
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 (2.10).
Solve $Lv = g$, then solve $L^T z = v$, and then set $q_1 = \frac{1}{2}\langle w, w\rangle - \langle y^1, c + Az\rangle$ and $q_1' = \langle \nabla q^{\text{D}}(y), y^1 - y\rangle_{\mathcal{F}\cup\mathcal{U}}$.
if $q_1 \le q^{\text{D}}(y) + \eta q_1'$ **then** return the approximate Cauchy point y^1. **end if**
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 $g_F^1 = \langle \nabla g^{\text{D}}(y), \Delta y^1\rangle_{\mathcal{F}}$, $g_1^1 = \langle \nabla g^{\text{D}}(y), \Delta y^1\rangle_{\mathcal{S}_1^1}$, $g_2^1 = 0$, and $g_3^1 = \langle \nabla g^{\text{D}}(y), \Delta y^1\rangle_{\mathcal{S}_3^1}$.
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 (2.15), 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 $q_i \le q^{\text{D}}(y) + \eta q_i'$ **then** return the approximate Cauchy point y^i. **end if**
  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 $g_F^i = \beta g_F^{i-1}$, $g_1^i = \langle \nabla g^{\text{D}}(y), \Delta y^i\rangle_{\mathcal{S}_1^i}$, $g_2^i = \langle \nabla g^{\text{D}}(y), \Delta y^i\rangle_{\mathcal{S}_2^i}$, 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 (2.16) with δy^{i-1} defined by (2.13).
  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

2.2.1 Sparse H

When H is sparse, we may use (1.4), (1.5), (2.1), and $c^i = \langle \Delta y^i, c\rangle$ introduced in Algorithm 2.6 to derive

$$\begin{aligned} q_{i+1} &= q^{\text{D}}(y^{i+1}) = q^{\text{D}}(y^i + \Delta y^i) = q^{\text{D}}(y^i) - \langle \Delta y^i, AH^{-1}(g - A^T y) + c\rangle + \tfrac{1}{2}\langle \Delta y^i, AH^{-1}A^T \Delta y^i\rangle \\ &= q^{\text{D}}(y^i) - \langle \Delta y^i, c\rangle + \langle L^{-1}A^T \Delta y^i, L^{-1}(A^T y - g)\rangle + \tfrac{1}{2}\langle L^{-1}A^T \Delta y^i, L^{-1}A^T \Delta y^i\rangle \\ &= q^{\text{D}}(y^i) - \langle \Delta y^i, c\rangle + \langle s^i, h\rangle + \tfrac{1}{2}\langle s^i, s^i\rangle \\ &= q_i - \langle \Delta y^i, c\rangle + \langle s^i, h\rangle + \tfrac{1}{2}\langle s^i, s^i\rangle \\ &= q_i - c^i + \langle s^i, h\rangle + \tfrac{1}{2}\langle s^i, s^i\rangle, \end{aligned} \tag{2.14}$$

where

$$Ls^i = A^T \Delta y^i \quad \text{and} \quad Lh = A^T y - g. \tag{2.15}$$

Note that s^i is not likely to be sparse since Δy^i is usually not sparse, which makes the computation in (2.14) inefficient. However, by making use of (2.13), we can see that

$$s^{i+1} = \beta s^i + \delta s^i, \text{ where } L(\delta s^i) = A^T(\delta y^i) \tag{2.16}$$

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 (2.16) that the vector δs^i will likely be sparse when L and A are sparse. We can also use (2.16) 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 (2.14). These observations are combined with our previous comments to form Algorithm 2.7.

2.2.2 Structured H

When H is structured according to (1.2) and (2.4), an argument similar to (2.14) shows that

$$\begin{aligned} q_{i+1} &= q^{\text{D}}(y^{i+1}) = q^{\text{D}}(y^i + \Delta y^i) \\ &= q^{\text{D}}(y^i) - \langle \Delta y^i, c\rangle + \langle A^T \Delta y^i, H^{-1}(A^T y - g)\rangle + \tfrac{1}{2}\langle A^T \Delta y^i, H^{-1} A^T \Delta y^i\rangle \\ &= q_i - \langle \Delta y^i, c\rangle + \langle h_{\text{B}}, s^i_{\text{B}}\rangle + \langle h_{\text{V}}, W s^i_{\text{V}}\rangle + \tfrac{1}{2}\langle s^i_{\text{B}}, s^i_{\text{B}}\rangle + \tfrac{1}{2}\langle s^i_{\text{V}}, W s^i_{\text{V}}\rangle, \\ &= q_i - c^i + \langle h_{\text{B}}, s^i_{\text{B}}\rangle + \langle h_{\text{V}}, W s^i_{\text{V}}\rangle + \tfrac{1}{2}\langle s^i_{\text{B}}, s^i_{\text{B}}\rangle + \tfrac{1}{2}\langle s^i_{\text{V}}, W s^i_{\text{V}}\rangle, \end{aligned} \tag{2.17}$$

where

$$s^i_{\text{B}} = B^T A^T \Delta y^i, \quad h_{\text{B}} = B^T(A^T y - g), \quad s^i_{\text{V}} = V^T A^T \Delta y^i \text{ and } h_{\text{V}} = V^T(A^T y - g). \tag{2.18}$$

Similar to (2.16), we can use (2.13) to obtain

$$s^{i+1}_{\text{B}} = \beta s^i_{\text{B}} + \delta s^i_{\text{B}}, \quad \text{with} \quad \delta s^i_{\text{B}} = B^T A^T(\delta y^i), \tag{2.19}$$

which in turn leads to

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

These updates allow us to perform the sparse updates required to compute (2.17). These observations are combined with our previous comments to form Algorithm 2.8.

2.3 The subspace step in Algorithm 2.1

Let $\mathcal{A}$ be the active set at the Cauchy point (see Algorithm 2.1) and $\mathcal{F} = \{1, 2, \ldots, m\} \setminus \mathcal{A}$ with $m_{\mathcal{F}} = |\mathcal{F}|$. By construction, the components of y^{C} that correspond to $\mathcal{A}$ are zero, and those corresponding to the set $\mathcal{F}$ are strictly positive. With $\Delta y = (\Delta y^{\mathcal{A}}, \Delta y^{\mathcal{F}})$, the subspace phase requires that we approximately

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

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

$$\Delta y^{\mathcal{F}}_* = \underset{\Delta y^{\mathcal{F}} \in \mathbb{R}^{m_{\mathcal{F}}}}{\text{argmin}} \ q^{\mathcal{F}}(\Delta y^{\mathcal{F}}) := \tfrac{1}{2}\langle \Delta y^{\mathcal{F}}, H^{\mathcal{F}} \Delta y^{\mathcal{F}}\rangle + \langle \Delta y^{\mathcal{F}}, g^{\mathcal{F}}\rangle, \tag{2.20}$$

where

$$H^{\mathcal{F}} := A^{\mathcal{F}} H^{-1} (A^{\mathcal{F}})^T \ \text{ and } \ g^{\mathcal{F}} := -A^{\mathcal{F}} H^{-1}\Big(g - (A^{\mathcal{F}})^T (y^{\text{C}})^{\mathcal{F}}\Big) - c^{\mathcal{F}}. \tag{2.21}$$

Here, $A^{\mathcal{F}}$ and $(y^{\text{C}})^{\mathcal{F}}/c^{\mathcal{F}}$ 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 (2.20) has a finite solution or $q^{\mathcal{F}}(\Delta y^{\mathcal{F}})$ 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.

Algorithm 2.8 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 $\nabla g^{\text{D}}(y) = Ax - c$.
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 $h_{\text{B}} = w - b_g$ and $h_{\text{V}} = z - v_g$, and then set $q^{\text{D}}(y) = \frac{1}{2}\langle w, w\rangle + \frac{1}{2}\langle z, Wz\rangle - \langle y, c\rangle - \langle w, b_g\rangle - \langle z, Wv_g\rangle$.
Define $\mathcal{A}$, $\mathcal{F}$, and $\mathcal{U}$ using (2.9).
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 (2.10).
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 $q_1' = \langle \nabla q^{\text{D}}(y), y^1 - y\rangle_{\mathcal{F}\cup\mathcal{U}}$.
if $q_1 \leq q^{\text{D}}(y) + \eta q_1'$ **then** return the approximate Cauchy point y^1. **end if**
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 $g_F^1 = \langle \nabla g^{\text{D}}(y), \Delta y^1\rangle_{\mathcal{F}}$, $g_1^1 = \langle \nabla g^{\text{D}}(y), \Delta y^1\rangle_{\mathcal{S}_1^1}$, $g_2^1 = 0$, and $g_3^1 = \langle \nabla g^{\text{D}}(y), \Delta y^1\rangle_{\mathcal{S}_3^1}$.
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 (2.18), and then compute $\langle s_{\text{B}}^1, h_{\text{B}}\rangle$ and $\langle s_{\text{B}}^1, s_{\text{B}}^1\rangle$.
Set $q_2 = q_1 - c^1 + \langle s_{\text{B}}^1, h_{\text{B}}\rangle + \frac{1}{2}\langle s_{\text{B}}^1, s_{\text{B}}^1\rangle$ and $q_2' = q_1' + g^1$.
for $i = 2, 3, \ldots$ **do**
  if $q_i \leq q^{\text{D}}(y) + \eta q_i'$ **then** return the approximate Cauchy point y^i. **end if**
  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 $g_F^i = \beta g_F^{i-1}$, $g_1^i = \langle \nabla g^{\text{D}}(y), \Delta y^i\rangle_{\mathcal{S}_1^i}$, $g_2^i = \langle \nabla g^{\text{D}}(y), \Delta y^i\rangle_{\mathcal{S}_2^i}$, 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 $\delta s_{\text{B}}^{i-1}$ from (2.19) with δy^{i-1} defined by (2.13).
  Compute $\langle s_{\text{B}}^i, h_{\text{B}}\rangle = \beta\langle s_{\text{B}}^{i-1}, h_{\text{B}}\rangle + \langle \delta s_{\text{B}}^{i-1}, h_{\text{B}}\rangle$.
  Compute $\langle s_{\text{B}}^i, s_{\text{B}}^i\rangle = \beta^2\langle s_{\text{B}}^{i-1}, s_{\text{B}}^{i-1}\rangle + \langle \delta s_{\text{B}}^{i-1}, 2s_{\text{B}}^{i-1} + \delta s_{\text{B}}^{i-1}\rangle$.
  Set $q_{i+1} = q_i - c^i + \langle s_{\text{B}}^i, h_{\text{B}}\rangle + \frac{1}{2}\langle s_{\text{B}}^i, s_{\text{B}}^i\rangle$ and $q_{i+1}' = q_i' + g^i$.
end for

2.3.1 Finite subspace minimizer

Since the objective in (2.20) is convex and quadratic, when $q^F(\Delta y^F)$ is bounded below its stationarity conditions give that

$$H^F \Delta y_*^F = -g^F, \tag{2.22}$$

that is to say

$$A^F H^{-1} (A^F)^T \Delta y_*^F = A^F H^{-1}\Big(g - (A^F)^T (y^{\text{C}})^F\Big) + c^F. \tag{2.23}$$

Given x, if we then define Δx_* via

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

then we have

$$\begin{pmatrix} H & (A^F)^T \\ A^F & 0 \end{pmatrix} \begin{pmatrix} x + \Delta x_* \\ -(y^{\mathrm{C}})^F - \Delta y_*^F \end{pmatrix} = \begin{pmatrix} -g \\ c^F \end{pmatrix} \tag{2.24}$$

or equivalently

$$\begin{pmatrix} H & (A^F)^T \\ A^F & 0 \end{pmatrix} \begin{pmatrix} x + \Delta x_* \\ -\Delta y_*^F \end{pmatrix} = \begin{pmatrix} (A^F)^T (y^{\mathrm{C}})^F - g \\ c^F \end{pmatrix}. \tag{2.25}$$

Notice that, so long as (2.24) (equivalently (2.25)) is consistent, we have

$$\Delta x_* = \underset{\Delta x \in \mathbb{R}^n}{\operatorname{argmin}} \tfrac{1}{2}\langle x + \Delta x, H(x + \Delta x)\rangle + \langle x + \Delta x, g\rangle \;\text{ such that }\; A^F(x + \Delta x) = c^F, \tag{2.26}$$

with $(y^{\mathrm{C}})^F + \Delta y_*^F$ the Lagrange multiplier vector. Of course (2.26) 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. Significantly, to find Δy_*^F, we may use any of the many direct or iterative methods for solving (2.23) or (2.25) [5].

2.3.2 Subspace minimizer at infinity

Since H^F is positive semi-definite and possibly singular, the linear system (2.23) (equivalently (2.22)) 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 2.1. *[The Fredholm Alternative [60, 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 2.1 that if (2.22) is inconsistent, there must be a direction of linear infinite descent [14], i.e., a vector Δy_∞^F for which

$$H^F \Delta y_\infty^F = 0 \;\text{ and }\; \langle \Delta y_\infty^F, g^F\rangle < 0 \tag{2.27}$$

along which

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

decreases linearly to minus infinity as α increases. Alternatively, examining (2.21), it is clear that inconsistency of (2.23) 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

$$(A^F)^T \Delta y_\infty^F = 0 \;\text{ and }\; \langle \Delta y_\infty^F, c^F\rangle > 0, \tag{2.28}$$

which is also a direction of linear infinite descent since

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

We now consider how to use H^F to find a Δy_∞^F that satisfies (2.27), and leave the details of how we might instead satisfy (2.28) to Appendix A.1.

To find a Δy_∞^F satisfying (2.27), we require that $H^F \Delta y_\infty^F \equiv A^F H^{-1} (A^F)^T \Delta y_\infty^F = 0$. Now, if we define $\Delta w_\infty := H^{-1}(A^F)^T \Delta y_\infty^F$, then we have that

$$\begin{pmatrix} H & (A^F)^T \\ A^F & 0 \end{pmatrix} \begin{pmatrix} \Delta w_\infty \\ -\Delta y_\infty^F \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{2.29}$$

[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 (2.27) is that

$$0 > \langle \Delta y^F_\infty, g^F \rangle = \langle -\Delta y^F_\infty, c^F \rangle + \langle \Delta w^{(k)}, (A^F)^T (y^C)^F - g \rangle, \tag{2.30}$$

where we have used (2.21) and the definition of Δw_∞. Notice that (2.29)–(2.30) together satisfy the negative Fredholm alternative to (2.25), which can be seen by applying Theorem 2.1 with data

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

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

$$K = LBL^T, \tag{2.31}$$

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` [26], `MA57` [25], `MA77` [71], `MA86` [57], `MA97` [58], `PARDISO` [73] and `WSMP` [53] offer good examples. Crucially, any singularity in K is reflected solely in B. Our method will return either a solution to (2.25) or to (2.29)–(2.30). To this end, we define w so that

$$Lw = b,$$

and then consider trying to solve the system

$$Bz = w. \tag{2.32}$$

If the system (2.32) is consistent, and thus there is a z satisfying (2.32), the vector u for which $L^T u = z$ also solves $Ku = b$ and thus its components solve (2.25). By contrast, if (2.32) is inconsistent, Theorem 2.1 implies that there is a vector p for which

$$Bp = 0 \ \text{ and } \ \langle p, w \rangle > 0. \tag{2.33}$$

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 (2.33). 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 (2.32), 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 (2.32) 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 (2.33) 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

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

In particular, it follows that

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

since $Ds = 0$ by construction. Thus, we have verified that this choice for p satisfies (2.33) 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 (1.2), 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 (2.31) 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 (2.25), with $(\Delta x, \Delta y^F) = (\Delta x_*, \Delta y_*^F)$, may be expanded to give

$$\begin{pmatrix} H_0 & (A^F)^T & Z \\ A^F & 0 & 0 \\ Z^T & 0 & U \end{pmatrix} \begin{pmatrix} x + \Delta x \\ -\Delta y^F \\ -\Delta z \end{pmatrix} = \begin{pmatrix} (A^F)^T (y^{\mathrm{C}})^F - g \\ c^F \\ 0 \end{pmatrix}, \tag{2.34}$$

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 2.1 with data

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

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 (2.25) or its Fredholm alternative (2.29)–(2.30). A decomposition of the form (2.31) 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 (2.20) 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 2.9). 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 (2.20) when the gradient $g_j = q^F(\Delta y_j^F)$ 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 (2.1) or the structured decomposition (1.2) as appropriate. 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 (2.27) at y^{C}. On the other hand, when $\langle p_j, H^F p_j\rangle > 0$, then

$$q^F(\Delta y_j^F) - q^F(\Delta y_{j+1}^F) = \tfrac{1}{2}\langle p_j, g_j\rangle^2 / \langle p_j, H^F p_j\rangle, \tag{2.35}$$

Algorithm 2.9 The conjugate gradient method for solving (2.20).

input: g^F and H^F as defined in (2.21).
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**
  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 if**
  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

which follows from $\Delta y_{j+1}^F = \Delta y_j^F + \alpha_j p_j$ and the fact that α_j in Algorithm 2.9 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 (2.35) 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

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad q(x) = \tfrac{1}{2} \langle x, Hx \rangle + \langle g, x \rangle \quad \text{such that} \quad c_L \leq Ax \leq c_U \quad \text{and} \quad x_L \leq x \leq x_U, \tag{3.1}$$

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 (3.1) may be written as

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

Then using the recipe (1.4), we have the dual problem

$$\underset{y_g}{\text{argmin}} \quad \tfrac{1}{2} y_g^T H_g^{\text{D}} y_g + y_g^T g_g^{\text{D}} \quad \text{such that} \quad y_g \geq 0, \tag{3.2}$$

with

$$H_g^{\text{D}} = A_g H^{-1} A_g^T \quad \text{and} \quad g_g^{\text{D}} = -A_g H^{-1} g - c_g. \tag{3.3}$$

If we now introduce the notation

$$v = \begin{pmatrix} y_L \\ y_U \\ z_L \\ z_U \end{pmatrix}, \quad J = \begin{pmatrix} A \\ A \\ I \\ I \end{pmatrix}, \quad b = \begin{pmatrix} c_L \\ c_U \\ x_L \\ x_U \end{pmatrix}, \quad H^{\text{D}} := J H^{-1} J^T \quad \text{and} \quad g^{\text{D}} := -J H^{-1} g - b \tag{3.4}$$

then it is easy to show that

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

Using these relationships, we see that problem (3.2) is equivalent to the problem

$$v_* = \operatorname*{argmin}_{v \in \mathbb{R}^{n_v}} q^{\mathrm{D}}(v) := \tfrac{1}{2} v^T H^{\mathrm{D}} v + v^T g^{\mathrm{D}} \;\text{ such that }\; (y_L, z_L) \geq 0 \;\text{ and }\; (y_U, z_U) \leq 0, \tag{3.5}$$

where $n_v = 2m + 2n$, in the sense that a solution to (3.2) is trivially obtained from a solution to (3.5).

Notice that since

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

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 (3.5) 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 (3.5), we simply generalize the method used to solve (1.4). 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 (3.5). Then we apply Algorithm 3.1.

Algorithm 3.1 Gradient projection method for solving the BQP (3.5).

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 $\alpha^{\mathrm{C}} = \operatorname{argmin}_{\alpha > 0} q^{\mathrm{D}}(P_{\mathcal{D}}[v + \alpha d])$.

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 $\Delta v^{\mathrm{S}} = \operatorname{argmin}_{\Delta v \in \mathbb{R}^{n_v}} q^{\mathrm{D}}(v^{\mathrm{C}} + \Delta v)$ such that $[\Delta v]_{\mathcal{A}} = 0$.

3. (improved subspace point)

Select $\alpha_{\max} > 0$ and then compute $\alpha^{\mathrm{S}} = \operatorname{argmin}_{\alpha \in [0, \alpha_{\max}]} q^{\mathrm{D}}(P_{\mathcal{D}}[v^{\mathrm{C}} + \alpha \Delta v^{\mathrm{S}}])$.

Set $v = P_{\mathcal{D}}[v^{\mathrm{C}} + \alpha^{\mathrm{S}} \Delta v^{\mathrm{S}}]$.

end while

The only significant differences for finding the Cauchy and improved subspace points using the obvious extension of Algorithm 2.4—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

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

(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 (3.4).

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

$$\begin{pmatrix} H & (J^F)^T \\ J^F & 0 \end{pmatrix} \begin{pmatrix} x + \Delta x \\ -\Delta v^F \end{pmatrix} = \begin{pmatrix} (J^F)^T (v^{\mathrm{C}})^F - g \\ b^F \end{pmatrix} \tag{3.6}$$

for (2.25), 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 (2.34), where J^F and b^F consist of the components of (3.4) that are free at the Cauchy point v^{C}.

3.1 The special case of a diagonal Hessian

An important special case occurs when H is diagonal, and covers applications that include performing ℓ_2-projections onto a polytope [2], solving bound-constrained least-squares problems [7, 9, §7.7], and globalization strategies within modern SQP algorithms [30, 49]. An obvious simplification is that any solve involving H becomes trivial. A more significant advantage comes in the subspace phase. Suppose, without loss of generality, that

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

where $A_1^F \in \mathbb{R}^{m_1 \times n_1}$, $H_1 \in \mathbb{R}^{n_1 \times n_1}$ and $g_1 \in \mathbb{R}^{n_1}$. Then, problem (3.6) can be written as

$$\begin{pmatrix} H_1 & 0 & (A_1^F)^T & I \\ 0 & H_2 & (A_2^F)^T & 0 \\ A_1^F & A_2^F & 0 & 0 \\ I & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 + \Delta x_1 \\ x_2 + \Delta x_2 \\ -(y^{\text{C}})^F - \Delta y^F \\ -(z^{\text{C}})^F - \Delta z^F \end{pmatrix} = \begin{pmatrix} -g_1 \\ -g_2 \\ c^F \\ x_1 \end{pmatrix}, \tag{3.7}$$

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

$$\begin{pmatrix} H_2 & (A_2^F)^T \\ A_2^F & 0 \end{pmatrix} \begin{pmatrix} x_2 + \Delta x_2 \\ -(y^{\text{C}})^F - \Delta y^F \end{pmatrix} = \begin{pmatrix} -g_2 \\ c^F - A_1^F x_1 \end{pmatrix}. \tag{3.8}$$

Thus the subspace phase is equivalent to finding a Fredholm alternative to the "reduced" system (3.8) or equivalently a Fredholm alternative to

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

where we subsequently recover Δx_2 by solving trivially

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

4 Regularized Problems

A related problem of interest—for example, in Sl_pQP methods for constrained optimization [28]—is to

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \;\; q(x) + \sigma \|(Ax - c)^-\|_1, \tag{4.1}$$

for given $\sigma > 0$, where $w^- = \max(0, -w)$ for any given w. Introducing the auxiliary vector v allows us to write (4.1) equivalently as

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

This has the dual

$$\underset{(x,v,y,z) \in \mathbb{R}^{2n+2m}}{\text{maximize}} \; -\tfrac{1}{2}\langle x, Hx \rangle + \langle c, y \rangle \text{ such that } Hx - A^T y + g = 0,\; y + z = \sigma e \text{ and } (y, z) \geq 0$$

or equivalently

$$\underset{y \in \mathbb{R}^m}{\text{minimize}} \;\; \tfrac{1}{2}\langle A^T y - g, H^{-1}(A^T y - g) \rangle - \langle c, y \rangle \;\text{ such that }\; 0 \leq y \leq \sigma e. \tag{4.2}$$

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

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \;\; q(x) + \sigma \|(Ax - c)\|_1, \tag{4.3}$$

we may instead solve the dual

$$\underset{y\in\mathbb{R}^m}{\text{minimize}} \quad \tfrac{1}{2}\langle A^T y - g, H^{-1}(A^T y - g)\rangle - \langle c, y\rangle \;\text{ such that }\; -\sigma e \leq y \leq \sigma e \tag{4.4}$$

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

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

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad q(x) + \sigma\|(Ax - c)^-\|_\infty, \tag{4.5}$$

then by introducing the auxiliary vector ν we can see that (4.5) is equivalent to

$$\underset{(x,\nu)\in\mathbb{R}^{n+1}}{\text{minimize}} \quad q(x) + \sigma\nu \;\text{ such that }\; Ax + \nu e \geq c \;\text{ and }\; \nu \geq 0. \tag{4.6}$$

The dual problem to (4.6) is then

$$\underset{(x,\nu,y,\xi)\in\mathbb{R}^{n+m+2}}{\text{maximize}} \; -\tfrac{1}{2}\langle x, Hx\rangle + \langle c, y\rangle \text{ such that } Hx - A^T y + g = 0,\; \langle e, y\rangle + \xi = \sigma \text{ and } (y, \xi) \geq 0$$

or equivalently

$$\underset{y\in\mathbb{R}^m}{\text{minimize}} \quad \tfrac{1}{2}\langle A^T y - g, H^{-1}(A^T y - g)\rangle - \langle c, y\rangle \;\text{ such that }\; y \geq 0 \;\text{ and }\; \langle e, y\rangle \leq \sigma. \tag{4.7}$$

Once again, we may apply the accelerated gradient-projection method, but now the projection is into the slightly more awkward scaled, orthogonal simplex $\mathcal{S}_m(\sigma) := \{y \in \mathbb{R}^m : y \geq 0 \;\text{ and }\; \langle e, y\rangle \leq \sigma\}$ for which there are nonetheless efficient projection algorithms [76]. In addition, in the subspace step computation (2.25), the defining matrix may have an additional row and column e whenever the dual constraint $\langle e, y\rangle \leq \sigma$ is active. Similarly, to

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad q(x) + \sigma\|(Ax - c)\|_\infty, \tag{4.8}$$

we can solve the dual

$$\underset{(y_L, y_U)\in\mathcal{S}_{2m}(\sigma)}{\text{minimize}} \quad \tfrac{1}{2}\langle A^T(y_L - y_U) - g, H^{-1}(A^T(y_L - y_U) - g)\rangle - \langle c, y_L - y_U\rangle \tag{4.9}$$

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 §2 and §3 for strictly-convex quadratic programs, in the general form (3.1), as a Fortran 2003 package, DQP, that is available as part of GALAHAD [41]. Further 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 [40] for the interior-point QP solver CQP within GALAHAD; package default values are chosen unless otherwise specified. Algorithm 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, as well as the point that solves a separable approximation to

[2] Available from `http://galahad.rl.ac.uk/galahad-www/`. A Matlab interface is also provided.

(3.5) 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 [65] 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 (3.6) (or finding its Fredholm alternative) is unlikely to benefit from matrix factorization updating techniques [36] usually associated with active-set methods, and it is better to solve successive systems (3.6) *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 more 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, the coefficient matrix for (3.6) 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 [8, 35, 36, 45] implemented as SCU in GALAHAD to solve (3.6) at this new iteration rather than resorting to refactorization. The details are quite standard [45, §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 to 100 new rows (by default) before refactorization; setting the limit to zero disables Schur-complement updating. We stop each run as soon as

$$\|P_{\mathcal{D}}[v - \nabla q^{\mathrm{D}}(v)] - v\| \leq \max(\varepsilon_a, \varepsilon_r \cdot \|P_{\mathcal{D}}[v_0 - \nabla q^{\mathrm{D}}(v_0)] - v_0\|), \tag{5.1}$$

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

Our tests involved the quadratic programming examples from the combined CUTEst [42] and Maros and Meszaros [62] test sets—multiple instances of similar nature were excluded. We only considered the 68 strictly convex problems (see Table A.2.1). 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 [58]—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 in the case when approximate solutions to the subspace subproblem were computed using the iterative CG method (see Section 2.3.3). The complete set of detailed results can be found in Table A.2.1. This instance of DQP failed on 12 problems, where a failure means that the termination test (5.1) 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 achieved.

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 Sections 2.3.1 and 2.3.2). The complete set of results can be found in Table A.2.2. (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 can conclude that the degradation in performance is primarily due to reaching the time limit, which is a consequence of using the more expense factorizations in liu 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 total time required to reach the following optimality tolerance levels: 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, and 10^{-6}. The full set of results for when CG is used as the subproblem solver is given in Table A.3.1, while the results for the case when factorizations are used can be found in Table A.3.2. For illustrative purposes, we represent the data found in these tables in the form of stacked bar graphs in Figures 5.1 and 5.2 for the CG case, and Figures 5.3 and 5.4 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 (Figures 5.1 and 5.3) or total time (Figures 5.2 and 5.4) needed to achieve the various accuracy levels: the jth stacked block (counting from the bottom up) for each problem corresponds to the accuracy level 10^{-j} for each $j \in \{1, 2, 3, 4, 5, 6\}$.

Figures 5.1 and 5.2 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 (Figure 5.1) and the times (Figure 5.2) are indistinguishable; two exceptions are `DUAL3` and `TABLE7`. We also remark that two problems in Figure 5.2 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 problems `LISWET2`–`LISWET6` are exceptions.

Figures 5.3 and 5.4, which are based on using factorizations to solve the subspace subproblem, also illustrate the close relationship between the number of iterations (Figure 5.3) and times required (Figure 5.4) 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 Figure 5.4 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 (Figures 5.3 and 5.4) tend to be "denser" towards the bottom when compared to the use of CG (Figures 5.1 and 5.2). 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 three-fold. 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

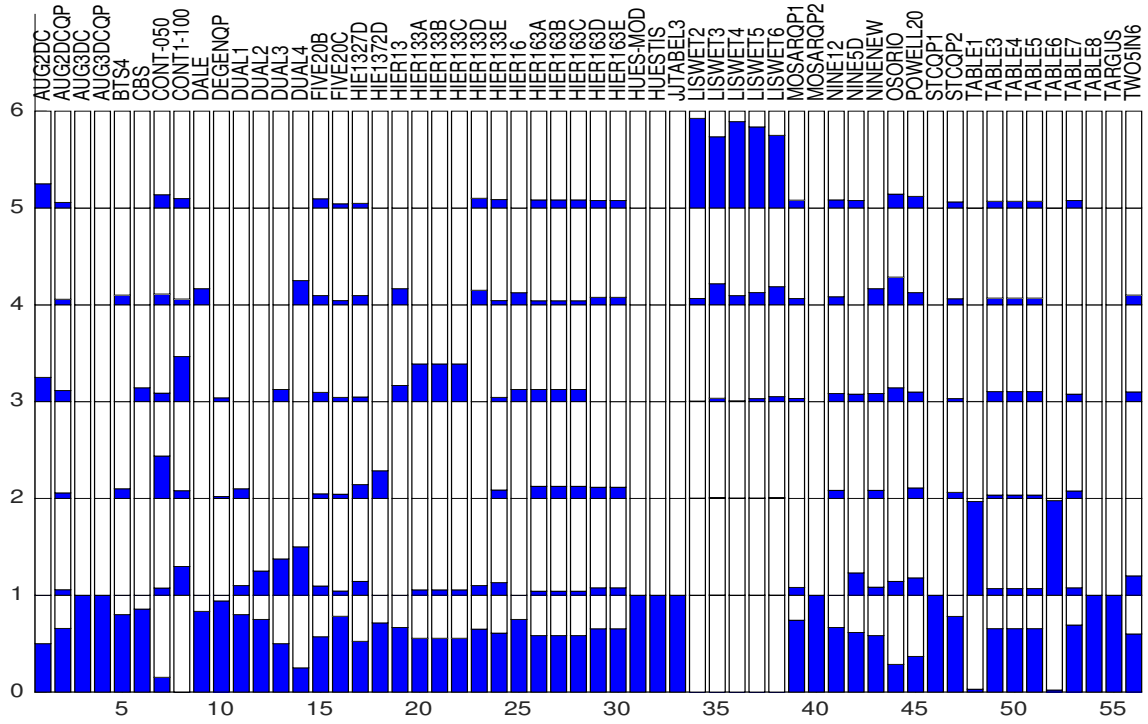

Figure 5.1: A comparison of iterations for DQP when using an iterative subproblem solver.

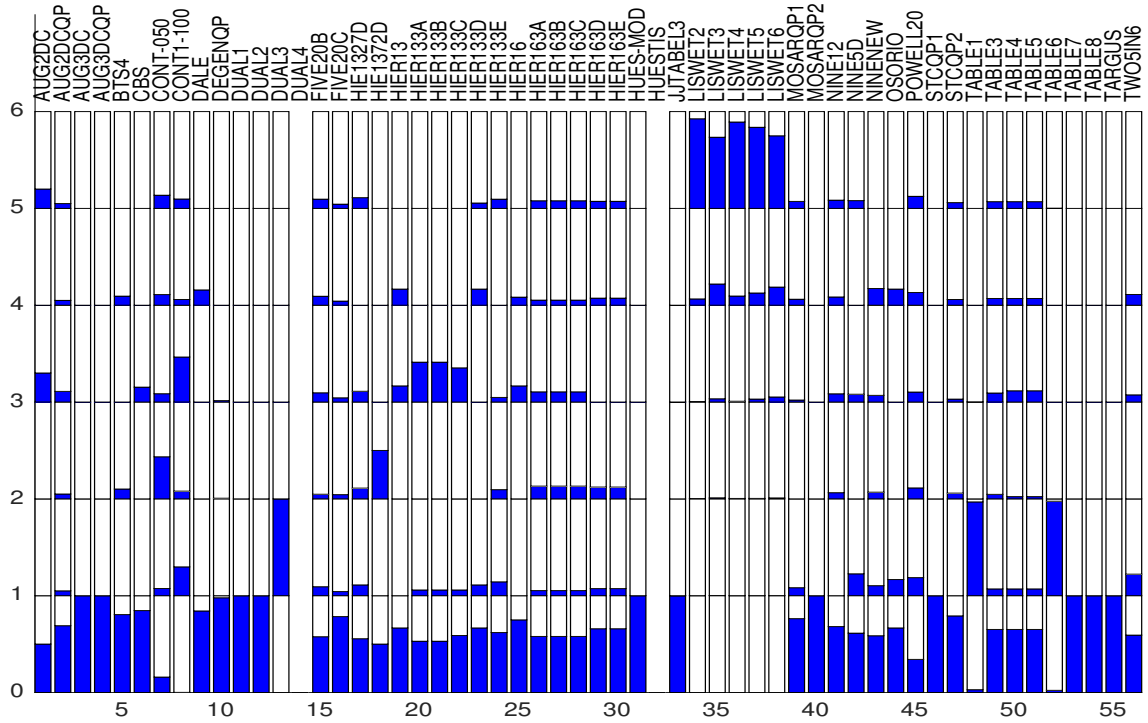

Figure 5.2: A comparison of times for DQP when using an iterative subproblem solver.

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

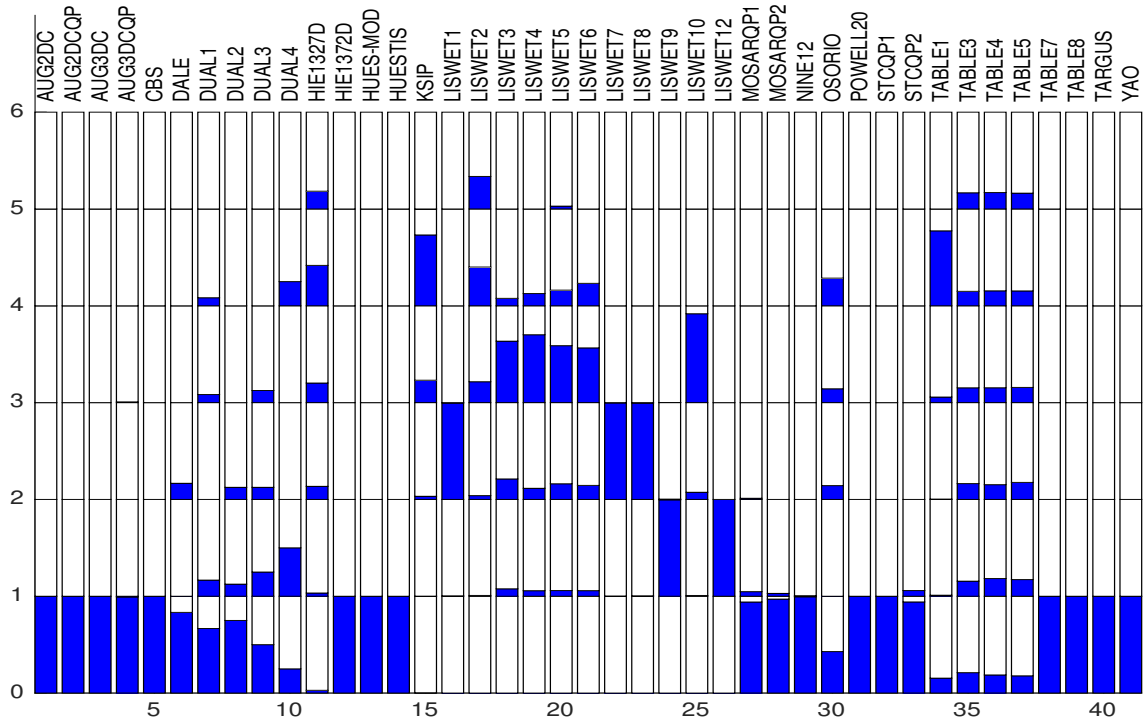

Figure 5.3: A comparison of iterations for DQP when using a direct method to solve each subproblem.

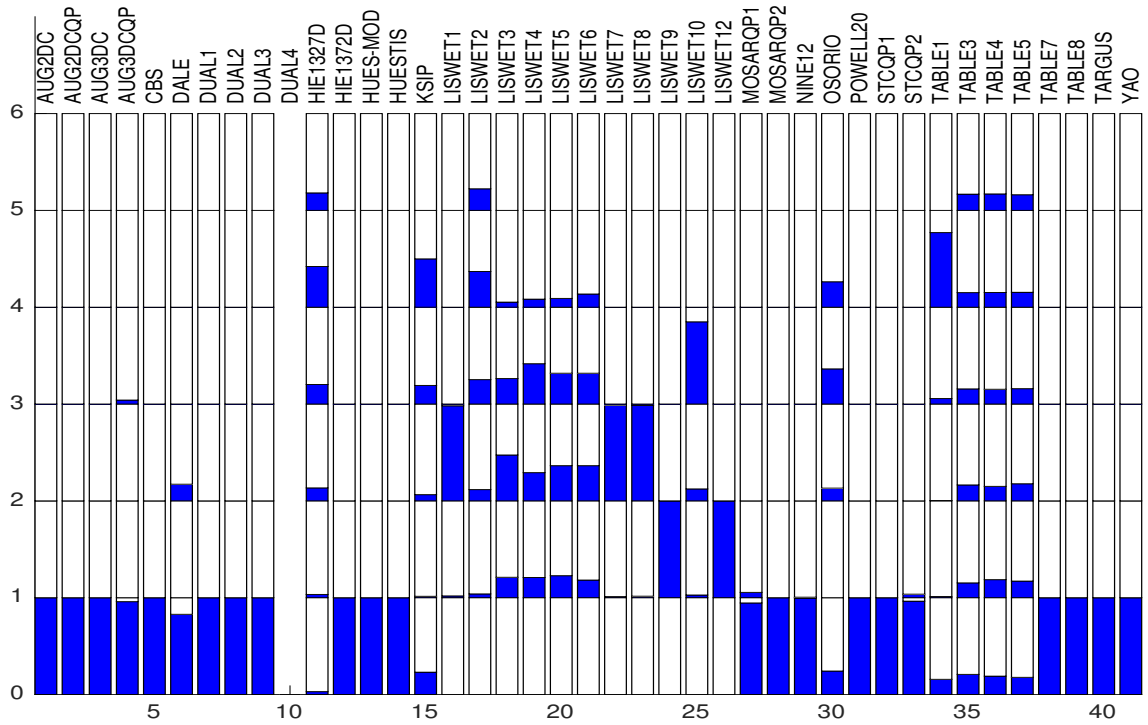

Figure 5.4: A comparison of times for DQP when using a direct method to solve each subproblem.

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 sequential quadratic optimization (SQO) method called iSQO [20]. 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.

We also considered how effective DQP could be when used 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 [40] 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 (1.3) holds more generally when H is positive semi-definite, it is not always then possible to eliminate the variables x as in (1.4) 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 §4, very minor modifications are required to fit our basic framework.

Acknowledgement

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.

References

[1] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[2] M. Arioli, A. Laratta, and O. Menchi. Numerical computation of the projection of a point onto a polyhedron. *Journal of Optimization Theory and Applications*, 43(4):495–525, 1984.

[3] D. Axehill and A. Hansson. A dual gradient projection quadratic programming algorithm tailored for model predictive control. In *Proceedings of the 47th IEEE Conference on Decision and Control*, pages 3057–3064, Cancun, Mexico, January 2008.

[4] R. A. Bartlett and L. T. Biegler. QPSchur: a dual, active-set, Schur-complement method for large-scale and structured convex quadratic programming. *Optimization and Engineering*, 7(1):5–32, 2006.

[5] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005.

[6] J. T. Betts and P. D. Frank. A sparse nonlinear optimization algorithm. *J. Optim. Theory Appl.*, 82:519–541, 1994.

[7] M. Bierlaire, Ph. L. Toint, and D. Tuyttens. On iterative algorithms for linear least squares problems with bound constraints. *Linear Algebra and its Applications*, 143:111–143, 1991.

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

[9] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, USA, 1996.

[10] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1995.

[11] N. L. Boland. A dual-active-set algorithm for positive semi-definite quadratic programming. *Mathematical Programming, Series A*, 78(1):1–27, 1997.

[12] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(2):129–156, 1994.

[13] P. H. Calamai and J. J. Moré. Projected gradient methods for linearly constrained problems. *Mathematical Programming*, 39(1):93–116, 1987.

[14] A. R. Conn and N. I. M. Gould. On the location of directions of infinite descent for nonlinear programming algorithms. *SIAM Journal on Numerical Analysis*, 21(6):302–325, 1984.

[15] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, 25(2):433–460, 1988. See also same journal **26**, 764-767, 1989.

[16] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50:399–430, 1988.

[17] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. SIAM, Philadelphia, 2000.

[18] F. E. Curtis and Z. Han. 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.

[19] F. E. Curtis, Z. Han, and D. P. Robinson. A Globally Convergent Primal-Dual Active-Set Framework for Large-Scale Convex Quadratic Optimization. *Computational Optimization and Applications*, DOI: 10.1007/s10589-014-9681-9, 2014.

[20] F. E. Curtis, T. C. Johnson, D. P. Robinson, and A. Wachter. An inexact sequential quadratic optimization algorithm for nonlinear optimization. *SIAM Journal on Optimization*, 24(3):1041–1074, 2014.

[21] J. Dominguez and M. D. González-Lima. A primal-dual interior-point algorithm for quadratic programming. *Numerical Algorithms*, 42(1):1–30, 2006.

[22] W. Dorn. Duality in quadratic programming. *Quarterly of Applied Mathematics*, 18:155–162, 1960.

[23] Z. Dostál. *Optimal quadratic programming algorithms*, volume 23 of *Springer Optimization and Its Applications*. Springer, New York, 2009. With applications to variational inequalities.

[24] Z. Dostál and J. Schöberl. Minimizing quadratic functions subject to bound constraints with the rate of convergence and finite termination. *Comput. Optim. Appl.*, 30(1):23–43, 2005.

[25] I. S. Duff. MA57 - a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30(2):118–144, 2004.

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

[27] I. S. Duff and J. K. Reid. The design of MA48: a code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Transactions on Mathematical Software*, 22(2):187–226, 1996.

[28] R. Fletcher. An ℓ_1 penalty method for nonlinear constraints. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel, editors, *Numerical Optimization 1984*, pages 26–40, Philadelphia, USA, 1985. SIAM.

[29] A. Forsgren, P. E Gill, and E. Wong. Primal and dual active-set methods for convex quadratic programming. *Mathematical Programming*, pages 1–40, 2015.

[30] M. P. Friedlander, N. I. M. Gould, S. Leyffer, and T. Munson. A filter active-set trust-region method. Technical Report Preprint ANL/MCS-P1456-0907, Argonne National Laboratory, Illinois, USA, 2007.

[31] M. P. Friedlander and S. Leyffer. Global and finite termination of a two-phase augmented Lagrangian filter method for general quadratic programs. *SIAM Journal on Scientific Computing*, 30(4):1706–1729, 2008.

[32] M. P. Friedlander and D. Orban. A primal–dual regularized interior-point method for convex quadratic programs. *Mathematical Programming Computation*, 4(1):71–107, 2012.

[33] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1981.

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

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

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

[37] P. E. Gill, W. Murray, and M. A. Saunders. User's guide for QPOPT 1.0: a Fortran package for quadratic programming. Report SOL 95-4, Department of Operations Research, Stanford University, Stanford, CA, 1995.

[38] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A Schur-complement method for sparse quadratic programming. In M. G. Cox and S. J. Hammarling, editors, *Reliable Numerical Computation*, pages 113–138. Oxford University Press, 1990.

[39] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27(1):1–33, 1983.

[40] N. I. M. Gould, D. Orban, and D. P. Robinson. Trajectory-following methods for large-scale degenerate convex quadratic programming. *Mathematical Programming Computation*, 5(2):113–142, 2013.

[41] N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD—a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. *ACM Transactions on Mathematical Software*, 29(4):353–372, 2003.

[42] N. I. M. Gould, , D. Orban, and Ph. L. Toint. CUTEst : a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 2015.

[43] N. I. M. Gould and Ph. L. Toint. 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".

[44] N. I. M. Gould and Ph. L. Toint. SQP methods for large-scale nonlinear programming. In M. J. D. Powell and S. Scholtes, editors, *System Modelling and Optimization, Methods, Theory and Applications*, pages 149–178, Dordrecht, The Netherlands, 2000. Kluwer Academic Publishers.

[45] N. I. M. Gould and Ph. L. Toint. An iterative working-set method for large-scale non-convex quadratic programming. *Applied Numerical Mathematics*, 43(1–2):109–128, 2002.

[46] N. I. M. Gould and Ph. L. Toint. Numerical methods for large-scale non-convex quadratic programming. In A. H. Siddiqi and M. Kočvara, editors, *Trends in Industrial and Applied Mathematics*, pages 149–179, Dordrecht, The Netherlands, 2002. Kluwer Academic Publishers.

[47] N. I. M. Gould, Y. Loh, and D. P. Robinson. A filter method with unified step computation for nonlinear optimization. *SIAM Journal on Optimization*, 24(1):175–209, 2014.

[48] N. I. M. Gould, Y. Loh, and D. P. Robinson. A filter SQP method: local convergence and numerical results. *SIAM Journal on Optimization*, 25(3):1885-1911, 2015.

[49] N. I. M. Gould and D. P. Robinson. A second derivative SQP method: global convergence. *SIAM Journal on Optimization*, 20(4):2023–2048, 2010.

[50] N. I. M. Gould and D. P. Robinson. A second derivative SQP method: Local convergence and practical issues. *SIAM Journal on Optimization*, 20(4):2049–2079, 2010.

[51] N. I. M. Gould and D. P. Robinson. A second derivative SQP method with a "trust-region-free" predictor step. *IMA Journal of Numerical Analysis*, 32(2):580–601, 2012.

[52] Z. Gu, E. Rothberg, and R. Bixby. Gurobi optimizer, version 5.5. 0. *Software program*, 2013.

[53] A. Gupta. 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, NY, USA, 2010.

[54] W. W Hager and D. W Hearn. Application of the dual active set algorithm to quadratic network optimization. *Computational Optimization and Applications*, 1(4):349–373, 1993.

[55] M. Hintermüller, K. Ito, and K. Kunisch. The primal-dual active set strategy as a semismooth Newton method. *SIAM J. Optim.*, 13(3):865–888 (electronic) (2003), 2002.

[56] J. D. Hogg, J. K. Reid, and J. A. Scott. Design of a multicore sparse Cholesky factorization using DAGs. *SIAM Journal on Scientific Computing*, 32(6):36273649, 2010.

[57] J. D. Hogg and J. A. Scott. An indefinite sparse direct solver for large problems on multicore machines. Technical Report RAL-TR-2010-011, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2010.

[58] J. D. Hogg and J. A. Scott. HSL_MA97: a bit-compatible multifrontal code for sparse symmetric systems. Technical Report RAL-TR-2011-024, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2011.

[59] ILOG CPLEX. High-performance software for mathematical programming and optimization, 2005.

[60] P. Lancaster and M. Tismenetsky. *The Theory of Matrices: with Applications*. Academic Press, London, second edition, 1985.

[61] M. Lescrenier. Convergence of trust region algorithms for optimization with bounds when strict complementarity does not hold. *SIAM Journal on Numerical Analysis*, 28(2):476–495, 1991.

[62] I. Maros and C. Meszaros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11-12:671–681, 1999.

[63] C. Mészáros. The BPMPD interior point solver for convex quadratic problems. *Optimization Methods and Software*, 11(1-4):431–449, 1999.

[64] J. L. Morales, J. Nocedal, and Y. Wu. A sequential quadratic programming algorithm with an additional equality constrained phase. *IMA Journal of Numerical Analysis*, 32:553–579, 2012.

[65] J. J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, 1(1):93–113, 1991.

[66] J. J. Moré and D. J. Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Math. Software*, 20(3):286–307, 1994.

[67] H. Moyh-ud Din and D. P. Robinson. A solver for nonconvex bound-constrained quadratic optimization. *Submitted to SIAM Journal on Optimization*, 2015.

[68] J. Nocedal and S. J. Wright. *Numerical Optimization*. Series in Operations Research. Springer Verlag, Heidelberg, Berlin, New York, second edition, 2006.

[69] R. A. Polyak, J. Costa, and S. Neyshabouri. Dual fast projected gradient method for quadratic programming. *Optimization Letters*, 2012. DOI: 10.1007/s11590-012-0476-6.

[70] M. J. D. Powell. *ZQPCVX a FORTRAN subroutine for convex quadratic programming*. Department of Applied Mathematics and Theoretical Physics, University, 1983.

[71] J. K. Reid and J. A. Scott. An out-of-core sparse Cholesky solver. *ACM Transactions on Mathematical Software*, 36(2), 2009. Article 9.

[72] D. P. Robinson, L. Feng, J. Nocedal, and J.-S. Pang. Subspace accelerated matrix splitting algorithms for asymmetric and symmetric linear complementarity problems. *SIAM Journal on Optimization*, 23(3):1371–1397, 2013.

[73] O. Schenk and K. Gärtner. On fast factorization pivoting methods for symmetric indefinite systems. *Electronic Transactions on Numerical Analysis*, 23:158–179, 2006.

[74] C. Schmid and L. T. Biegler. Quadratic programming methods for reduced hessian sqp. *Computers & chemical engineering*, 18(9):817–832, 1994.

[75] P. Spellucci. *Solving general convex QP problems via an exact quadratic augmented Lagrangian with bound constraints*. Techn. Hochsch., Fachbereich Mathematik, 1993.

[76] S. M. Stefanov. Polynomial algorithms for projecting a point onto a region defined by a linear constraint and box constraints in $\mathbb{R}^n$. *Journal of Applied Mathematics*, 2004(5):409–431, 2004.

[77] R. J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization methods and software*, 11(1-4):451–484, 1999.

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

[79] G. Yuan, S. Lu, and Z. Wei. A modified limited SQP method for constrained optimization. *Applied Mathematics*, 1(1):8–17, 2010.

[80] C. Zhu and R. T. Rockafellar. Primal-dual projected gradient algorithms for extended linear-quadratic programming. *SIAM Journal on Optimization*, 3(4):751–783, 1993.

Appendix A

A.1 Directions of linear infinite descent using A^F

We show how one might find a suitable Δy^F_∞ satisfying (2.28) when c^F is not in the range space of A^F (i.e., when (2.23) is inconsistent). Recall that A^F has m_k rows, and suppose that

$$A^F = P \begin{pmatrix} L \\ M \end{pmatrix} (U \;\; N), \tag{A.1.1}$$

where P is a permutation matrix, L and U are, respectively, r_k by r_k unit lower and upper triangular matrices, and the rank $r_k \leq \min(n, m_k)$; such a "rectangular" LU factorization is implemented in many modern sparse-matrix packages (e.g., `LUSOL` [34], `MA48` [27] and `MUMPS` [1]). Let

$$\begin{pmatrix} c^F_1 \\ c^F_2 \end{pmatrix} = P^T c^F, \tag{A.1.2}$$

where c^F_1 has r_k components. The following holds.

Lemma A.1.1. *Suppose that c^F does not lie in the range of A^F. Then*

$$d^F := c^F_2 - ML^{-1}c^F_1 \neq 0. \tag{A.1.3}$$

Proof. For a contrapositive proof, suppose that $c^F_2 = ML^{-1}c^F_1$. Writing $w = L^{-1}c^F_1$, we have

$$\begin{pmatrix} L \\ M \end{pmatrix} w = \begin{pmatrix} c^F_1 \\ c^F_2 \end{pmatrix}$$

or equivalently $A^F v = c^F$, where $v = \begin{pmatrix} U^{-1}w \\ 0 \end{pmatrix}$. This completes the proof since this shows that c^F lies in the range of A^F. □

This result allows us to determine whether there is a direction of linear infinite descent by checking whether d^F is zero or not; in practice small rounding errors need to be taken into account.

Next, let d^F be nonzero, D be any diagonal matrix for which $D_{ii} \geq 0$, and $Dd^F \neq 0$, and then define

$$\Delta y^F_{D2} := Dd^F \tag{A.1.4}$$

and

$$\Delta y^F_{D1} := -L^{-T}M^T \Delta y^F_{D2}. \tag{A.1.5}$$

Then we have the following crucial result.

Lemma A.1.2. *Suppose that c^F does not lie in the range of A^F, and that Δy^F_{D2} and Δy^F_{D1} are defined by (A.1.4) and (A.1.5). Then, the vector*

$$\Delta y^F_\infty = P \begin{pmatrix} \Delta y^F_{D1} \\ \Delta y^F_{D2} \end{pmatrix} \tag{A.1.6}$$

satisfies (2.28).

Proof. It follows trivially from (A.1.1), (A.1.6), and (A.1.5) that

$$(A^F)^T \Delta y^F_\infty = \begin{pmatrix} U^T \\ N^T \end{pmatrix} (L^T \;\; M^T) P^T \Delta y^F_\infty = \begin{pmatrix} U^T \\ N^T \end{pmatrix} (L^T \Delta y^F_{D1} + M^T \Delta y^F_{D2}) = 0.$$

Moreover (A.1.5) implies that

$$\langle \Delta y_{D1}^F, c_1^F \rangle = -\langle L^{-T} M^T \Delta y_{D2}^F, c_1^F \rangle = -\langle \Delta y_{D2}^F, M L^{-1} c_1^F \rangle,$$

which together with (A.1.6), (A.1.2), and the definition of D shows that

$$\langle \Delta y_\infty^F, c^F \rangle = \quad \langle \Delta y_{D1}^F, c_1^F \rangle + \langle \Delta y_{D2}^F, c_2^F \rangle = \langle \Delta y_{D2}^F, c_2^F - M L^{-1} c_1^F \rangle = \langle D d^F, d^F \rangle > 0,$$

where we used Lemma A.1.1 to derive the last inequality. This completes the proof. □

Of course the action of the inverses in (A.1.3) and (A.1.5) would be implemented in practice as forward and back solves with the triangular L and its transpose. The most obvious choices for D are the identity matrix, in which case $\Delta y_{D2}^F = d^F$, or the rank-one matrix $e_j e_j^T$ for some j satisfying $d_j^F \neq 0$, in which case $y_{D2}^F = d_j^F e_j$. We summarize our procedure in terms of general data A and c in Algorithm A.1.1.

Algorithm A.1.1 Computing a feasible direction or a direction of infinite descent satisfying (2.28).

---

procedure $[\Delta x, \Delta y, flag]$ = DOLID(A,c)

input: $A \in \mathbb{R}^{m \times n}$ and $c \in \mathbb{R}^m$.

1. (compute the LU factorization of A)

Compute nonsingular matrices L and U in $\mathbb{R}^{r \times r}$ and a permutation matrix P so that

$$A = P \begin{pmatrix} L \\ M \end{pmatrix} (U \;\; N),$$

2. (compute a feasible point for $A(x + \Delta x) = c$ or a direction of linear infinite descent)

Compute

$$d = c_2 - Mw, \;\; \text{where} \;\; \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = P^T c, \;\; Lw = c_1, \;\; \text{and} \;\; c_1 \in \mathbb{R}^r.$$

if $d = 0$ **then**

Set $flag$ to `false` and $\Delta y = 0$, and compute the feasible point

$$x + \Delta x = \begin{pmatrix} U^{-1} w \\ 0 \end{pmatrix}.$$

else

Compute diagonal matrix D such that $D_{ii} \geq 0$ and $Dd \neq 0$.

Set $flag$ to `true` and $\Delta x = 0$, and compute the direction of linear infinite descent

$$\Delta y = P \begin{pmatrix} \Delta y_1 \\ \Delta y_2 \end{pmatrix}, \;\; \text{where} \;\; \Delta y_2 = Dd \;\; \text{and} \;\; \Delta y_1 = -L^{-T} M^T \Delta y_2.$$

end if

end procedure

In practice, the LU factorization of A^F is preceded by block triangularization. Specifically, row and column permutations P and Q are applied so that

$$A^F = P \begin{pmatrix} A_{11}^F & 0 & \cdots & 0 \\ A_{21}^F & A_{22}^F & \ddots & 0 \\ \vdots & \vdots & \ddots & 0 \\ A_{\ell 1}^F & A_{\ell 2}^F & \cdots & A_{\ell\ell}^F \end{pmatrix} Q, \tag{A.1.7}$$

where each diagonal block, except perhaps the last, is square and order n_i, say. Vitally, rather than computing the LU factors of A^F, only the diagonal blocks are factorized. That is, rather than finding (A.1.1), the factorizations

$$A^F_{ii} = \begin{pmatrix} L_{ii} \\ M_{ii} \end{pmatrix} (U_{ii} \;\; N_{ii}), \quad \text{for } \; i = 1, \ldots, \ell, \tag{A.1.8}$$

involving triangular matrices L_{ii} and U_{ii} of order $r_i \leq n_i$ are obtained (any further row permutations have been absorbed into P). To discover whether there are directions of infinite descent and when they exist to find one, we proceed by considering the blocks.

To illustrate the general procedure, suppose we have (after suitable row and column permutations)

$$A = \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix} \;\text{ and }\; c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}.$$

We first use Algorithm A.1.1 to compute $[\Delta x_1, \Delta y_1, flag] = \text{DOLID}(A_{11}, c_1)$, and if $flag$ is true, then

$$\Delta y := \begin{pmatrix} \Delta y_1 \\ 0 \end{pmatrix}$$

is a direction of linear infinite descent. If $flag$ has the value false, then the vector c_1 lies in the range of A_{11}. So now suppose there is a direction of linear infinite descent

$$\Delta y = \begin{pmatrix} \Delta y_1 \\ \Delta y_2 \end{pmatrix}$$

with $\Delta y_2 \neq 0$. In particular, this means that $A^T \Delta y = 0$, or equivalently that

$$A^T_{22} \Delta y_2 = 0 \;\text{ and }\; A^T_{11} \Delta y_1 = -A^T_{21} \Delta y_2.$$

In this case $\Delta y_1 = -A^{+T}_{11} A^T_{21} \Delta y_2$, where A^+_{11} is the generalized inverse of A_{11}, and

$$\langle c, \Delta y \rangle = \langle c_2 - A_{21} A^+_{11} c_1, \Delta y_2 \rangle.$$

Thus we seek Δy_2 for which

$$A^T_{22} \Delta y_2 = 0 \;\text{ and }\; \langle c^{(2)}_2, \Delta y_2 \rangle \neq 0, \;\text{ where }\; c^{(2)}_2 := c_2 - A_{21} A^+_{11} c_1.$$

We may check this by calling $\text{DOLID}(A_{22}, c^{(2)}_2)$. Clearly this idea may be applied recursively by partitioning A_{21}, A_{22}, and c_2. We summarize the procedure in Algorithm A.1.2.

Note that Algorithm A.1.2 only provides the direction of linear infinite descent or confirms that there is no such direction. In the latter case, one has then to solve (2.25) to recover a solution (2.22). Clearly it is inefficient to check for a direction of linear infinite descent when it is unlikely that there will be one, and equally it is wasteful to try to compute a finite minimizer when this is unlikely to happen. Pragmatically, therefore, we would apply the method of §2.3.1 when $m_k \leq n$ and only resort to that from §2.3.2 when (2.25) is reported to be inconsistent. When $m_k > n$, the method of §2.3.2 takes precedence, and we only use (2.25) when we have failed to find a direction of linear infinite descent.

Algorithm A.1.2 Computing a feasible direction or a direction of infinite descent satisfying (2.28).

input: Matrix A^F factorized as in (A.1.7) and c^F.

Let c_j denote the jth block of components of $P^T c^F$ for $j = 1, 2, \ldots \ell$ and some ℓ.

for $i = 1, \ldots, \ell$ **do**

Set $c_i \leftarrow c_i - \sum_{j=1}^{i-1} A_{ij}\Delta x_j$ and compute $[\Delta x_i, \Delta y_i, flag]$ =**dolid**(A_{ii}, c_i).

if $flag$ has the value true **then**

Back solve by performing $A_{jj}^T \Delta y_j = -\sum_{k=j+1}^{i} A_{jk}^T \Delta y_k$ for $j = i-1, \ldots, 1$.

return the direction of infinite descent

$$\Delta y^F = P \begin{pmatrix} \Delta y_1 \\ \vdots \\ \Delta y_i \\ 0 \end{pmatrix}.$$

end if

if $i = \ell$ **then**

return there does not exist a direction of infinite descent.

end if

end for

A.2 Detailed Summary of Numerical Results

Tables A.2.1 (iterative subproblem solver) and A.2.2 (direct subproblem solver) summarize the results of DQP for the test problems. We provide values for the objective value (Obj. Value), primal violation (Primal Viol.), dual violation (Dual Viol.), and complementarity violation (Comp. Viol.) at the final iterate. The number of iterations computed (Iters.) and time required (Time) are also supplied. The final column (Status) indicates the outcome of the solution process; see Table A.2.3 for the meaning of each status value.

Table A.2.1: Results for package DQP when an iterative subproblem solver is used.

Name	Obj. Value	Primal Viol.	Dual Viol.	Comp. Viol.	Iters.	Time	Status
AUG2DC	1.81836806E+06	5.5E-08	5.7E-14	5.3E-06	4	0.20	0
AUG2DCQP	6.49813468E+06	4.7E-07	2.3E-13	1.3E-03	35	1.00	0
AUG3DC	2.76540711E+04	1.2E-07	1.8E-15	1.8E-06	1	0.03	0
AUG3DCQP	6.15603837E+04	1.6E-08	3.6E-15	5.9E-07	16	0.56	0
BTS4	6.90514312E+07	6.8E-08	2.3E-13	1.4E-04	10	1.29	0
CBS	5.36161663E+06	4.7E-07	1.1E-13	1.7E-04	7	0.13	0
CONT-050	-4.56385182E+00	9.4E-07	4.1E-17	7.8E-08	1694	13.01	0
CONT1-100	-4.64440164E+00	9.9E-07	1.1E-16	1.1E-07	28743	894.66	0
CONT1-200	-4.73584419E+00	2.9E-01	2.8E-18	2.0E-03	-13999	-1800.10	-19
DALE	1.82596990E+04	3.0E-08	1.8E-15	5.2E-08	6	0.20	0
DEGENQP	1.13968254E-09	1.1E-08	5.9E-14	5.5E-09	102	7.46	0
DUAL1	3.50129677E-02	2.2E-11	3.3E-15	3.2E-10	10	0.51	0
DUAL2	3.37336714E-02	8.9E-16	2.0E-15	3.2E-17	8	0.01	0
DUAL3	1.35755832E-01	3.7E-12	6.0E-15	2.7E-10	8	0.01	0
DUAL4	7.46090652E-01	4.9E-11	3.8E-15	1.6E-09	4	0.00	0
FIVE20B	1.37339055E+08	9.7E-07	2.3E-13	1.5E-03	21	3.36	0
FIVE20C	1.62654515E+08	1.6E-07	2.3E-13	1.9E-03	46	8.03	0
HIE1327D	2.48905243E+06	7.3E-08	1.1E-13	9.6E-06	21	0.09	0
HIE1372D	8.06298962E+05	8.4E-08	5.7E-14	3.8E-05	7	0.02	0
HIER13	7.50704295E+06	8.1E-08	1.1E-13	6.4E-04	6	0.06	0
HIER133A	7.28073418E+06	3.9E-07	2.3E-13	1.5E-03	18	0.17	0
HIER133B	7.28073418E+06	3.9E-07	2.3E-13	1.5E-03	18	0.17	0
HIER133C	7.28073418E+06	3.9E-07	2.3E-13	1.5E-03	18	0.17	0
HIER133D	2.99559854E+07	4.6E-07	4.5E-13	1.7E-03	20	0.18	0
HIER133E	2.99559854E+07	7.5E-08	2.3E-13	3.9E-04	23	0.21	0
HIER16	1.04347325E+07	8.2E-09	1.1E-13	5.7E-05	8	0.12	0
HIER163A	8.69609949E+06	3.5E-09	1.1E-13	6.1E-06	24	0.38	0
HIER163B	8.69609949E+06	3.5E-09	1.1E-13	6.1E-06	24	0.38	0
HIER163C	8.69609949E+06	3.5E-09	1.1E-13	6.1E-06	24	0.38	0
HIER163D	3.47843980E+07	3.1E-07	1.1E-13	1.7E-04	26	0.41	0
HIER163E	3.47843980E+07	3.1E-07	1.1E-13	1.7E-04	26	0.41	0
HUES-MOD	3.48244898E+07	3.0E-12	4.4E-16	2.5E-07	4	0.01	0
HUESTIS	3.48245418E+10	1.9E-12	0.0E+00	1.8E-04	4	0.00	0
JJTABEL3	1.30980707E+14	3.7E-07	9.3E-10	1.2E+00	6544	50.55	0
KSIP	5.75815430E-01	5.3E-07	2.2E-16	1.1E-05	-100001	-21.28	-18
LASER	1.14592952E+02	3.5E+02	9.5E-07	4.4E+02	-100001	-1722.22	-18
LISWET1	5.07215863E+00	3.6E-05	2.8E-14	1.1E-02	-100001	-410.40	-18
LISWET2	4.99798638E+00	9.0E-07	5.6E-16	1.7E-06	5920	24.17	0
LISWET3	4.99776948E+00	9.3E-07	2.2E-16	1.5E-06	1649	6.53	0
LISWET4	4.99779938E+00	9.7E-07	2.2E-16	2.5E-06	5786	22.83	0
LISWET5	4.99782483E+00	9.8E-07	6.2E-16	2.0E-06	3208	12.77	0
LISWET6	4.99788651E+00	9.6E-07	4.4E-16	1.3E-06	2360	9.40	0
LISWET7	5.00215132E+00	1.4E-04	4.1E-14	5.4E-02	-100001	-411.09	-18
LISWET8	5.19206925E+00	1.4E-02	2.4E-13	1.4E-01	-100001	-397.44	-18
LISWET9	2.21560671E+01	7.5E-03	8.2E-13	1.7E+00	-100001	-398.40	-18
LISWET10	5.02426900E+00	1.6E-03	2.8E-14	6.0E-03	-100001	-400.56	-18
LISWET12	1.21770710E+01	8.0E-03	6.8E-13	3.4E+00	-100001	-401.34	-18
MOSARQP1	-3.82140982E+03	8.7E-07	7.1E-15	9.6E-07	62	0.98	0
MOSARQP2	-5.05259194E+03	2.7E-07	7.1E-15	2.3E-06	20	0.37	0
NINE12	3.27678548E+07	5.2E-07	1.1E-13	2.0E-04	12	0.47	0
NINE5D	3.79459871E+07	1.1E-07	1.1E-13	3.6E-05	13	0.62	0
NINENEW	2.73604976E+07	4.7E-07	2.3E-13	6.6E-04	12	0.30	0
OSORIO	1.13368430E+01	6.5E-09	2.5E-16	2.7E-08	7	0.07	0
POWELL20	5.20895823E+10	1.0E-06	0.0E+00	5.0E+00	24611	419.11	0
QPBAND	-3.21962814E+06	0.0E+00	3.3E-01	2.9E+02	-398	-1800.56	-19
STCQP1	3.67100485E+05	1.2E-08	1.8E-12	2.9E-06	4	0.79	0
STCQP2	3.71892743E+04	2.4E-07	4.8E-13	1.9E-05	32	2.88	0
TABLE1	2.78623226E+12	1.1E-08	2.3E-10	4.2E-03	1122	4.14	0
TABLE3	2.93000638E+09	6.8E-09	1.8E-12	1.1E-04	29	0.43	0
TABLE4	2.93000638E+09	6.8E-09	1.8E-12	1.1E-04	29	0.43	0
TABLE5	2.93000638E+09	6.8E-09	1.8E-12	1.1E-04	29	0.43	0
TABLE6	2.78623226E+12	4.7E-08	1.2E-10	3.6E-02	1098	4.06	0
TABLE7	7.79587339E+08	8.1E-08	3.6E-12	7.8E-05	13	0.02	0
TABLE8	1.91933038E+02	8.9E-09	1.7E-15	9.2E-10	3	0.01	0
TARGUS	1.68744463E+07	1.3E-07	4.5E-13	2.7E-04	185	0.11	0
TOYSARAH	5.89122424E+19	4.0E+02	1.2E-07	2.1E+11	-100001	-894.75	-18
TWO5IN6	1.91002932E+07	3.5E-07	2.3E-13	2.6E-04	10	0.27	0
YAO	2.48349703E+01	5.7E-04	3.1E-13	8.0E-01	-100001	-411.01	-18

Table A.2.2: Results for package DQP when a direct subproblem solver is used.

Name	Obj. Value	Primal Viol.	Dual Viol.	Comp. Viol.	Iters.	Time	Status
AUG2DC	1.81836807E+06	5.7E-13	5.7E-14	3.6E-10	1	0.07	0
AUG2DCQP	6.49813474E+06	1.4E-12	2.3E-13	3.8E-09	6815	217.79	0
AUG3DC	2.76540711E+04	1.5E-13	1.8E-15	6.1E-13	1	0.08	0
AUG3DCQP	6.15603837E+04	3.6E-14	3.6E-15	1.9E-12	135	8.76	0
BTS4	6.90514292E+07	5.8E-05	2.3E-13	2.3E-01	-4398	-1800.10	-19
CBS	5.36161663E+06	1.4E-12	1.7E-13	3.8E-11	6	0.26	0
CONT-050	0.00000000E+00	6.8E+00	3.2E+62	4.6E+62	-2429	-45.54	-5
CONT1-100	0.00000000E+00	9.7E+00	2.7E+61	1.7E+62	-3716	-339.94	-5
CONT1-200	0.00000000E+00	8.7E+00	1.0E+62	1.8E+62	-3622	-1443.41	-5
DALE	1.82596990E+04	1.1E-10	1.8E-15	2.9E-10	6	0.16	0
DEGENQP	2.35828854E+01	7.6E+00	4.5E+03	1.2E+01	-17642	-1800.10	-19
DUAL1	3.50129678E-02	2.1E-13	4.8E-15	7.6E-15	12	0.01	0
DUAL2	3.37336714E-02	6.5E-14	2.1E-15	2.3E-15	8	0.01	0
DUAL3	1.35755833E-01	5.6E-14	4.0E-15	8.1E-15	8	0.02	0
DUAL4	7.46090649E-01	2.2E-15	4.6E-15	1.9E-15	4	0.00	0
HIE1327D	2.48905242E+06	8.5E-07	1.1E-13	3.1E-03	3062	255.06	0
HIE1372D	8.06298962E+05	5.4E-11	2.8E-14	7.3E-08	6	0.06	0
HIER13	7.50699445E+06	9.2E-04	2.3E-13	1.5E+01	-4251	-1800.17	-19
HIER133A	7.28065677E+06	1.2E-03	1.1E-13	9.3E+00	-3409	-1800.35	-19
HIER133B	7.28066894E+06	2.3E-03	2.3E-13	1.0E+01	-3433	-1800.45	-19
HIER133C	7.28068850E+06	8.2E-04	1.1E-13	6.3E+00	-3410	-1800.31	-19
HIER133D	2.99558560E+07	1.7E-03	2.3E-13	2.7E+01	-3694	-1800.31	-19
HIER133E	2.99555414E+07	7.9E-03	2.3E-13	7.6E+01	-3443	-1800.12	-19
HIER16	1.04280522E+07	2.4E-02	1.1E-13	9.4E+02	-903	-1801.11	-19
HIER163A	8.68585956E+06	4.1E-02	5.7E-14	8.7E+02	-796	-1801.97	-19
HIER163B	8.68605451E+06	5.3E-02	1.1E-13	8.4E+02	-787	-1800.68	-19
HIER163C	8.68488527E+06	6.0E-02	5.7E-14	9.5E+02	-789	-1800.34	-19
HIER163D	3.47533066E+07	6.1E-02	2.3E-13	2.6E+03	-794	-1801.22	-19
HIER163E	3.47729480E+07	3.4E-02	2.3E-13	1.0E+03	-795	-1801.47	-19
HUES-MOD	3.48244898E+07	2.7E-12	4.4E-16	2.6E-07	4	0.07	0
HUESTIS	3.48245418E+10	1.1E-12	0.0E+00	1.1E-04	4	0.02	0
JJTABEL3	1.30980708E+14	4.0E-01	4.7E-10	1.1E+07	-100001	-839.10	-18
KSIP	5.75797349E-01	5.6E-07	2.8E-17	1.1E-16	81304	23.90	0
LASER	0.00000000E+00	2.1E+02	2.5E+62	1.5E+64	-64	-0.61	-5
LISWET1	7.22189448E+00	1.3E-11	1.3E-12	1.5E-07	1169	2.23	0
LISWET2	4.99808204E+00	5.0E-07	2.8E-15	4.6E-08	482	1.03	0
LISWET3	4.99777877E+00	1.6E-15	2.1E-16	1.8E-15	52	0.19	0
LISWET4	4.99781511E+00	2.2E-15	2.6E-16	6.8E-15	87	0.24	0
LISWET5	4.99783185E+00	2.7E-15	6.1E-16	5.9E-15	68	0.22	0
LISWET6	4.99789739E+00	2.2E-15	2.2E-16	4.8E-15	69	0.22	0
LISWET7	9.98951642E+01	8.7E-11	7.3E-12	7.6E-06	1659	3.62	0
LISWET8	1.43130575E+02	8.7E-11	7.3E-12	5.9E-06	897	2.63	0
LISWET9	3.92920161E+02	8.7E-11	7.8E-12	6.6E-06	905	4.00	0
LISWET10	9.89648678E+00	1.1E-11	1.0E-12	1.2E-07	622	1.46	0
LISWET12	3.47518978E+02	8.7E-11	7.3E-12	7.0E-06	936	3.63	0
MOSARQP1	-3.82140980E+03	5.4E-15	7.1E-15	3.0E-15	84	0.55	0
MOSARQP2	-5.05259194E+03	2.2E-14	7.1E-15	7.4E-15	34	0.21	0
NINE12	3.27678548E+07	2.9E-08	2.3E-13	1.1E-03	176	201.30	0
NINE5D	9.75187214E+06	3.7E+02	1.1E-13	1.5E+05	-168	-1807.39	-19
NINENEW	2.73588127E+07	2.2E-02	1.1E-13	6.0E+02	-2776	-1800.03	-19
OSORIO	1.13368430E+01	1.8E-14	4.4E-16	1.8E-14	7	0.33	0
POWELL20	5.20895828E+10	9.3E-10	0.0E+00	5.8E-03	2501	38.16	0
QPBAND	0.00000000E+00	0.0E+00	2.3E+62	4.9E+65	-4210	-1491.66	-5
STCQP1	3.67100485E+05	3.2E-13	2.3E-12	7.7E-10	4	2.90	0
STCQP2	3.71892743E+04	1.8E-14	6.5E-13	1.3E-12	17	2.39	0
TABLE1	2.78623226E+12	4.3E-08	2.9E-11	3.3E-02	279	1.25	0
TABLE3	2.93000638E+09	6.7E-07	3.6E-12	2.7E-02	21840	505.33	0
TABLE4	2.93000638E+09	8.1E-07	3.6E-12	2.4E-02	22498	533.54	0
TABLE5	2.93000638E+09	3.9E-07	1.8E-12	1.7E-02	24521	573.97	0
TABLE6	2.78623225E+12	1.4E-04	1.2E-10	1.5E+07	-100001	-438.35	-18
TABLE7	7.79587339E+08	2.2E-11	9.1E-13	4.6E-07	22	0.04	0
TABLE8	1.91933038E+02	1.8E-14	1.8E-15	1.8E-13	3	0.01	0
TARGUS	1.68744463E+07	1.7E-12	2.3E-13	8.3E-09	201	0.07	0
TOYSARAH	5.88881391E+19	5.9E+05	1.2E-07	2.9E+15	-73844	-1800.00	-19
TWO5IN6	9.47197659E+06	1.3E+02	1.1E-13	1.3E+05	-366	-1801.26	-19
YAO	1.97704616E+02	2.3E-10	1.6E-11	3.2E-05	3	0.01	0

Status	Meaning
0	Optimality conditions met.
-5	Problem was determined to be locally infeasible.
-18	Maximum iteration limit was reached.
-19	Maximum time limit was reached.

Table A.2.3: Meaning of the status column in Tables A.2.1 and A.2.2.

A.3 Required Iterations and Time to Reach Various Accuracies

Tables A.3.1 (iterative subproblem solver) and A.3.2 (direct subproblem solver) give the total number of iterations (Iters.) and total time (Time) needed to meet termination tolerances 10^{-1} through 10^{-6}.

Table A.3.1: Checkpoint information for package DQP when an iterative subproblem solver is used.

	Iters.						Time					
Name	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
AUG2DC	2	2	2	3	3	4	0.05	0.05	0.05	0.08	0.08	0.10
AUG2DCQP	23	25	27	31	33	35	0.69	0.74	0.79	0.90	0.95	1.00
AUG3DC	1	1	1	1	1	1	0.02	0.02	0.02	0.02	0.02	0.02
AUG3DCQP	16	16	16	16	16	16	0.54	0.54	0.54	0.54	0.54	0.54
BTS4	8	8	9	9	10	10	1.03	1.03	1.16	1.16	1.28	1.28
CBS	6	6	6	7	7	7	0.11	0.11	0.11	0.13	0.13	0.13
CONT-050	258	384	1126	1274	1462	1694	2.07	3.03	8.68	9.80	11.23	13.00
CONT1-100	34	8565	10864	24240	25950	28743	1.13	267.10	338.58	754.25	807.37	894.13
CONT1-200	2583	2703	-1	-1	-1	-1	355.00	370.17	-1.00	-1.00	-1.00	-1.00
DALE	5	5	5	5	6	6	0.16	0.16	0.16	0.16	0.19	0.19
DEGENQP	96	96	98	102	102	102	7.28	7.28	7.33	7.44	7.44	7.44
DUAL1	8	9	10	10	10	10	0.01	0.01	0.01	0.01	0.01	0.01
DUAL2	6	8	8	8	8	8	0.01	0.01	0.01	0.01	0.01	0.01
DUAL3	4	7	7	8	8	8	0.00	0.01	0.01	0.01	0.01	0.01
DUAL4	1	3	3	3	4	4	0.00	0.00	0.00	0.00	0.00	0.00
FIVE20B	12	14	15	17	19	21	1.93	2.24	2.40	2.72	3.03	3.35
FIVE20C	36	38	40	42	44	46	6.28	6.62	6.97	7.32	7.66	8.01
HIE1327D	11	14	17	18	20	21	0.05	0.06	0.07	0.08	0.08	0.09
HIE1372D	5	5	7	7	7	7	0.01	0.01	0.02	0.02	0.02	0.02
HIER13	4	4	4	5	6	6	0.04	0.04	0.04	0.05	0.06	0.06
HIER133A	10	11	11	18	18	18	0.09	0.10	0.10	0.17	0.17	0.17
HIER133B	10	11	11	18	18	18	0.09	0.10	0.10	0.17	0.17	0.17
HIER133C	10	11	11	18	18	18	0.10	0.11	0.11	0.17	0.17	0.17
HIER133D	13	15	15	15	18	20	0.12	0.14	0.14	0.14	0.17	0.18
HIER133E	14	17	19	20	21	23	0.13	0.16	0.18	0.19	0.19	0.21
HIER16	6	6	6	7	8	8	0.09	0.09	0.09	0.11	0.12	0.12
HIER163A	14	15	18	21	22	24	0.22	0.24	0.29	0.33	0.35	0.38
HIER163B	14	15	18	21	22	24	0.22	0.24	0.29	0.33	0.35	0.38
HIER163C	14	15	18	21	22	24	0.22	0.24	0.29	0.33	0.35	0.38
HIER163D	17	19	22	22	24	26	0.27	0.30	0.35	0.35	0.38	0.41
HIER163E	17	19	22	22	24	26	0.27	0.30	0.35	0.35	0.38	0.41
HUES-MOD	4	4	4	4	4	4	0.01	0.01	0.01	0.01	0.01	0.01
HUESTIS	4	4	4	4	4	4	0.00	0.00	0.00	0.00	0.00	0.00
JJTABEL3	6541	6541	6541	6543	6544	6544	50.49	50.49	50.49	50.51	50.52	50.52
KSIP	5	103	1519	8884	-1	-1	0.02	0.28	1.50	3.68	-1.00	-1.00
LASER	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
LISWET1	0	6	21	295	-1	-1	0.00	0.02	0.08	1.21	-1.00	-1.00
LISWET2	0	4	22	54	441	5920	0.00	0.01	0.09	0.22	1.79	24.15
LISWET3	0	4	20	77	436	1649	0.00	0.01	0.08	0.30	1.73	6.53
LISWET4	0	4	29	76	625	5786	0.00	0.01	0.11	0.30	2.46	22.81
LISWET5	0	4	17	117	522	3208	0.00	0.01	0.07	0.46	2.07	12.76
LISWET6	0	4	26	148	590	2360	0.00	0.01	0.10	0.59	2.35	9.40
LISWET7	0	5	16	783	-1	-1	0.00	0.02	0.06	3.21	-1.00	-1.00
LISWET8	0	4	316	-1	-1	-1	0.00	0.01	1.25	-1.00	-1.00	-1.00
LISWET9	0	37	18665	-1	-1	-1	0.00	0.14	73.61	-1.00	-1.00	-1.00
LISWET10	0	5	60	1767	-1	-1	0.00	0.02	0.24	7.05	-1.00	-1.00
LISWET12	0	40	3395	-1	-1	-1	0.00	0.15	13.34	-1.00	-1.00	-1.00
MOSARQP1	46	51	51	53	57	62	0.74	0.82	0.82	0.84	0.90	0.97
MOSARQP2	20	20	20	20	20	20	0.37	0.37	0.37	0.37	0.37	0.37
NINE12	8	8	9	10	11	12	0.32	0.32	0.35	0.39	0.43	0.47
NINE5D	8	11	11	12	12	13	0.38	0.52	0.52	0.57	0.57	0.62
NINENEW	7	8	9	10	12	12	0.17	0.20	0.22	0.24	0.29	0.29
OSORIO	2	3	3	4	6	7	0.04	0.05	0.05	0.05	0.06	0.06
POWELL20	9054	13469	16136	18570	21666	24611	143.16	221.31	268.59	311.74	366.65	418.85
QPBAND	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
STCQP1	4	4	4	4	4	4	2.90	2.90	2.90	2.90	2.90	2.90
STCQP2	25	25	27	28	30	32	15.72	15.72	16.90	17.50	18.68	19.87
TABLE1	33	1119	1119	1121	1121	1122	0.12	4.12	4.12	4.13	4.13	4.13
TABLE3	19	21	22	25	27	29	0.28	0.31	0.33	0.37	0.40	0.43
TABLE4	19	21	22	25	27	29	0.28	0.31	0.32	0.37	0.40	0.43
TABLE5	19	21	22	25	27	29	0.28	0.31	0.32	0.37	0.40	0.43
TABLE6	23	1096	1097	1097	1097	1098	0.09	4.05	4.05	4.05	4.05	4.06
TABLE7	9	10	11	12	12	13	0.02	0.02	0.02	0.02	0.02	0.02
TABLE8	3	3	3	3	3	3	0.01	0.01	0.01	0.01	0.01	0.01
TARGUS	185	185	185	185	185	185	0.11	0.11	0.11	0.11	0.11	0.11
TOYSARAH	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
TWO5IN6	6	8	8	9	10	10	0.16	0.22	0.22	0.24	0.27	0.27
YAO	47	71	3778	-1	-1	-1	0.19	0.29	15.45	-1.00	-1.00	-1.00

Table A.3.2: Checkpoint information for package DQP when a direct subproblem solver is used.

	Iters.						Time					
Name	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
AUG2DC	1	1	1	1	1	1	0.27	0.27	0.27	0.27	0.27	0.27
AUG2DCQP	6815	6815	6815	6815	6815	6815	890.77	890.77	890.77	890.77	890.77	890.77
AUG3DC	1	1	1	1	1	1	0.31	0.31	0.31	0.31	0.31	0.31
AUG3DCQP	134	134	134	135	135	135	33.76	33.76	33.76	35.20	35.20	35.20
BTS4	486	1314	2484	4030	-1	-1	863.80	2321.94	4362.33	7062.49	-1.00	-1.00
CBS	6	6	6	6	6	6	1.35	1.35	1.35	1.35	1.35	1.35
CONT-050	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
CONT1-100	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
CONT1-200	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
DALE	5	5	6	6	6	6	0.53	0.53	0.64	0.64	0.64	0.64
DEGENQP	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
DUAL1	8	10	10	11	12	12	0.01	0.01	0.01	0.01	0.01	0.01
DUAL2	6	7	8	8	8	8	0.01	0.01	0.01	0.01	0.01	0.01
DUAL3	4	6	7	8	8	8	0.01	0.01	0.01	0.01	0.01	0.01
DUAL4	1	3	3	3	4	4	0.00	0.00	0.00	0.00	0.00	0.00
HIE1327D	87	189	603	1222	2500	3062	20.99	44.45	138.72	280.88	577.09	704.46
HIE1372D	6	6	6	6	6	6	0.15	0.15	0.15	0.15	0.15	0.15
HIER13	1006	2481	-1	-1	-1	-1	1132.75	2767.33	-1.00	-1.00	-1.00	-1.00
HIER133A	965	1829	-1	-1	-1	-1	1340.32	2544.38	-1.00	-1.00	-1.00	-1.00
HIER133B	793	1926	-1	-1	-1	-1	1096.55	2643.27	-1.00	-1.00	-1.00	-1.00
HIER133C	1302	2060	-1	-1	-1	-1	1827.50	2873.03	-1.00	-1.00	-1.00	-1.00
HIER133D	1997	2743	-1	-1	-1	-1	2599.48	3525.58	-1.00	-1.00	-1.00	-1.00
HIER133E	1645	2716	-1	-1	-1	-1	2267.25	3750.37	-1.00	-1.00	-1.00	-1.00
HIER16	846	-1	-1	-1	-1	-1	5554.50	-1.00	-1.00	-1.00	-1.00	-1.00
HIER163A	721	-1	-1	-1	-1	-1	5508.16	-1.00	-1.00	-1.00	-1.00	-1.00
HIER163B	731	-1	-1	-1	-1	-1	5602.53	-1.00	-1.00	-1.00	-1.00	-1.00
HIER163C	763	-1	-1	-1	-1	-1	5823.36	-1.00	-1.00	-1.00	-1.00	-1.00
HIER163D	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
HIER163E	381	-1	-1	-1	-1	-1	2889.72	-1.00	-1.00	-1.00	-1.00	-1.00
HUES-MOD	4	4	4	4	4	4	0.07	0.07	0.07	0.07	0.07	0.07
HUESTIS	4	4	4	4	4	4	0.02	0.02	0.02	0.02	0.02	0.02
JJTABEL3	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
KSIP	290	367	3005	21811	81299	81304	6.38	6.77	8.55	13.87	27.70	27.70
LASER	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
LISWET1	0	5	1169	1169	1169	1169	0.00	0.04	2.21	2.21	2.21	2.21
LISWET2	0	4	23	127	320	482	0.00	0.04	0.16	0.42	0.80	1.03
LISWET3	0	4	15	48	52	52	0.00	0.04	0.13	0.18	0.19	0.19
LISWET4	0	5	15	76	87	87	0.00	0.05	0.12	0.22	0.24	0.24
LISWET5	0	4	15	55	66	68	0.00	0.05	0.13	0.20	0.22	0.22
LISWET6	0	4	14	53	69	69	0.00	0.04	0.12	0.19	0.22	0.22
LISWET7	0	4	1659	1659	1659	1659	0.00	0.04	3.61	3.61	3.61	3.61
LISWET8	0	4	897	897	897	897	0.00	0.04	2.63	2.63	2.63	2.63
LISWET9	0	901	904	905	905	905	0.00	3.99	3.99	3.99	3.99	3.99
LISWET10	0	5	51	622	622	622	0.00	0.04	0.22	1.46	1.46	1.46
LISWET12	0	936	936	936	936	936	0.00	3.62	3.62	3.62	3.62	3.62
MOSARQP1	79	83	84	84	84	84	0.52	0.55	0.55	0.55	0.55	0.55
MOSARQP2	33	34	34	34	34	34	0.21	0.21	0.21	0.21	0.21	0.21
NINE12	175	176	176	176	176	176	479.41	481.79	481.79	481.79	481.79	481.79
NINE5D	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
NINENEW	646	2706	-1	-1	-1	-1	1037.07	4417.92	-1.00	-1.00	-1.00	-1.00
OSORIO	3	3	4	5	7	7	0.46	0.46	0.71	1.40	1.90	1.90
POWELL20	2501	2501	2501	2501	2501	2501	186.30	186.30	186.30	186.30	186.30	186.30
QPBAND	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
STCQP1	4	4	4	4	4	4	6.09	6.09	6.09	6.09	6.09	6.09
STCQP2	16	17	17	17	17	17	6.41	6.65	6.65	6.65	6.65	6.65
TABLE1	43	46	47	63	279	279	0.80	0.86	0.88	1.17	5.11	5.11
TABLE3	4606	8008	11582	14922	18186	21840	456.39	791.98	1153.32	1497.13	1829.26	2198.03
TABLE4	4208	8320	11752	15204	18682	22498	439.86	872.43	1219.74	1573.94	1927.13	2322.22
TABLE5	4366	8589	12901	16749	20515	24521	444.67	874.73	1318.60	1717.90	2104.66	2511.45
TABLE6	60	63	64	68	161	-1	1.05	1.10	1.12	1.19	2.76	-1.00
TABLE7	22	22	22	22	22	22	0.04	0.04	0.04	0.04	0.04	0.04
TABLE8	3	3	3	3	3	3	0.01	0.01	0.01	0.01	0.01	0.01
TARGUS	201	201	201	201	201	201	0.07	0.07	0.07	0.07	0.07	0.07
TOYSARAH	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
TWO5IN6	-1	-1	-1	-1	-1	-1	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
YAO	3	3	3	3	3	3	0.01	0.01	0.01	0.01	0.01	0.01