Noname manuscript No.
(will be inserted by the editor)

A Globally Convergent Primal-Dual Active-Set Framework for Large-Scale Convex Quadratic Optimization

Frank E. Curtis · Zheng Han · Daniel P. Robinson

July 6, 2014

Abstract We present a primal-dual active-set framework for solving large-scale convex quadratic optimization problems (QPs). In contrast to classical active-set methods, our framework allows for multiple simultaneous changes in the active-set estimate, which often leads to rapid identification of the optimal active-set regardless of the initial estimate. The iterates of our framework are the active-set estimates themselves, where for each a primal-dual solution is uniquely defined via a reduced subproblem. Through the introduction of an index set auxiliary to the active-set estimate, our approach is globally convergent for strictly convex QPs. Moreover, the computational cost of each iteration typically is only modestly more than the cost of solving a reduced linear system. Numerical results are provided, illustrating that two proposed instances of our framework are efficient in practice, even on poorly conditioned problems. We attribute these latter benefits to the relationship between our framework and semi-smooth Newton techniques.

Keywords convex quadratic optimization, active-set methods, large-scale optimization, semi-smooth Newton methods

Mathematics Subject Classification (2000) 49M05, 49M15, 65K05, 65K10, 65K15

Frank E. Curtis
Dept. of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA.
This author was supported in part by National Science Foundation grant DMS–1016291.
E-mail: frank.e.curtis@gmail.com

Zheng Han
Dept. of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, USA.
This author was supported in part by National Science Foundation grant DMS–1016291.
E-mail: zhh210@lehigh.edu

Daniel P. Robinson
Dept. of Applied Mathematics and Statistics, Johns Hopkins University, Baltimore, MD, USA.
This author was supported in part by National Science Foundation grant DMS–1217153.
E-mail: daniel.p.robinson@jhu.edu

1 Introduction

In this paper, we introduce a primal-dual active-set framework for solving strictly convex quadratic optimization problems (QPs). The importance of having efficient algorithms for solving such problems is paramount as practitioners often seek the solution of a single quadratic optimization problem [29] or the solutions of a sequence of QPs as a means for solving nonlinear problems. This latter situation occurs, e.g., in the contexts of augmented Lagrangian (AL) [4,8] and sequential quadratic optimization (SQO) methods [14,17–20,31] for solving nonlinear optimization problems. Specific application areas requiring the solution of large-scale convex QPs include bound-constrained linear least-squares estimation problems [37], the solution of partial differential equations over obstacles [7], the journal bearing problem [10], model predictive control problems [2,3,25], and the smooth primal support vector machine problem [36].

Two popular classes of algorithms for solving convex QPs are interior-point and active-set methods. Motivated by our interests in AL and SQO methods for solving nonlinear optimization problems, we focus on methods in the latter class due to their abilities to exploit good starting points and to compute accurate solutions despite ill-conditioning and/or degeneracy. The main disadvantage of previously proposed active-set methods is their potentially high computational costs vis-à-vis that of interior-point methods, which is precisely the disadvantage that the framework in this paper is designed to overcome.

The goal of an active-set method is to identify an optimal active-set, which in our formulation refers to variables that are equal to either their lower or upper bounds at the optimal solution. Classical active-set methods [6,15,21] maintain a so-called working-set as an estimate of an optimal active-set. (Roughly speaking, a working-set is an estimate of a subset of the optimal active-set, chosen so that an associated basis matrix is nonsingular.) The iterates are forced to remain primal (or dual) feasible, and during each iteration a single index is added to, or removed from, the working-set. Provided that cycling is prevented at degenerate points, global convergence of these methods is guaranteed by monotonically decreasing (increasing) the value of the primal (dual) objective function. However, this incremental process of one-by-one changes in the working-set is the main contributing factor in the potentially high computational costs that prevent classical active-set methods from being effective general-purpose solvers for large-scale problems.

The purpose of this paper is to present, analyze, and provide numerical results for a primal-dual active-set framework for solving large-scale convex QPs. Like all active-set methods, our framework can take advantage of good initial estimates of the optimal active-set. However, unlike classical active-set methods, our strategy allows for rapid adaptation of the active-set estimate. The framework is based on the primal-dual active-set strategy described by Hintermüller, Ito, and Kunisch in [25]; see also [1] for a similar method proposed for solving linear complementarity problems. The key difference between our framework and the algorithm in [25], however, is that we introduce a set, auxiliary to the active-set estimate, to house the indices of variables whose bounds are to be enforced explicitly during a given iteration. By moving indices into this auxiliary set, our active-set estimates need only form subsets of the optimal active-set in order to produce the optimal solution. Indeed, by carefully manipulating this set, our method is globally convergent for QPs with equality and inequality constraints and with any strictly convex

quadratic objective function. This is in contrast to the method proposed in [25], which is proved to be globally convergent only for bound-constrained problems with certain types of quadratic objectives, and which has been shown to cycle on other convex QPs; e.g., see [13] and Example 1 in §3.1. We refer the reader to [2, 3, 5, 26, 28, 32] for other algorithms that allow for rapid evolution of the active set estimate, but note that they are not guaranteed to converge on all strictly convex QPs. Overall, our idea of employing an auxiliary set represents a straightforward enhancement of the methodology in [25]. However, our enhancement is novel and allows for global convergence guarantees on a much broader class of problems, while maintaining the impressive practical behavior of the algorithm in [25].

We propose two specific instances of our framework that possess these global convergence guarantees. However, perhaps more important than these guarantees is that, in our experience, our techniques often leave the auxiliary set empty. This is advantageous as in such cases our instances reduce to the algorithm in [25], which is extremely efficient when it does converge. (In particular, when our auxiliary set is empty, the cost of each iteration is the same as that in [25], i.e., that of solving a reduced linear system.) However, even for large and poorly conditioned problems—e.g., with up to 10^4 variables and objective functions whose Hessian matrices have condition number up to 10^6—we typically find that the auxiliary set needs only to include very few indices relative to n. In such cases, the resulting reduced subproblems can be solved via the enumeration of all potential active-sets or via a classical active-set method, where in the latter case the cost may only amount to a linear system solve and a few updates to the corresponding matrix factorization. Overall, our numerical experiments illustrate that the goals of our globalization mechanism (i.e., the introduction of our auxiliary set) are met in practice: the set remains small to maintain low computational costs, and it only increases in size to avoid the cycling that may occur in the algorithm from [25] when applied to solve certain convex QPs. Due to our theoretical convergence guarantees and encouraging numerical results, we believe that our framework is a promising active-set approach for solving large-scale convex QPs.

We remark that the convergence guarantees for our framework are ensured without resorting to globalization strategies typical in constrained optimization, such as an objective-function-based merit function or filter, which typically impose the use of line searches to obtain solution estimates that are acceptable according to these mechanisms. Instead, global convergence in the framework that we present is based on monotonic increases in the size of our auxiliary set, or on monotonic decreases in the optimality error (guided by manipulations of the auxiliary set). Overall, one may summarize that global convergence of primal active set algorithms [31] is guaranteed by monotonic decreases of the primal objective function, global convergence of dual active set algorithms [23, 24] is guaranteed by monotonic increases of the dual objective function, and global convergence of our primal-dual framework is guaranteed by monotonicity properties ensured by the incorporation of our auxiliary set. In fact, all of these types of methods converge in a finite number of iterations due to these monotonicity properties. It should be noted that, in the worst case, our auxiliary set could grow to include the indices of all of the primal variables. However, in the experiments summarized in this paper, it is seen that in many cases the auxiliary set remains small or even empty. Thus, we have observed that our framework is particularly efficient when there are a relatively large number of variables not at their bounds at the optimal solution.

We also briefly remark that other alternatives for solving large-scale QPs are projected gradient methods, which typically execute line searches to ensure monotonicity in an objective. Approaches of this type—especially those that involve iterative subspace minimization phases—have proved to be effective for such problems, provided they are not too ill-conditioned [11,27,30,33]. See the end of §2 for further comparisons with other related approaches.

The paper is organized as follows. In §2, we describe our framework and prove a generic global convergence theorem; in particular, we prove a result stating that our framework is globally convergent when employed to solve both bound-constrained and generally-constrained strictly convex QPs. In §3, we discuss two instances of the framework and their relationship to the method in [25]. In §4, we describe the details of our implementation and then present numerical experiments in §5. These experiments illustrate that an implementation of our framework is efficient when employed to solve bound-constrained and generally-constrained strictly convex QPs, at least those with many degrees of freedom relative to n. Finally, in §6 we summarize our findings and comment on additional advantages of our framework, such as the potential incorporation of inexact reduced subproblem solves.

Notation. We use subscripts to denote (subsets of) elements of a vector or matrix; e.g., with $x_{\mathcal{S}}$ we denote the vector composed of the elements in the vector x corresponding to those indices in the ordered set $\mathcal{S}$, and with $H_{\mathcal{S}_1,\mathcal{S}_2}$ we denote the matrix composed of the elements in the matrix H corresponding to those row and column indices in the ordered sets $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. A common exception occurs when we refer to (subsets of) elements of a vector with an additional subscript, such as x_*. In such cases, we denote (subsets of) elements after appending brackets, such as in $[x_*]_{\mathcal{S}}$.

2 Algorithmic Framework

In this section, we motivate, describe, and prove a global convergence result for our framework. The problem we consider is the strictly convex QP

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & c^Tx + \frac{1}{2}x^THx \\ \text{subject to} & Ax = b, \;\; \ell \leq x \leq u, \end{array} \tag{1}$$

where $c \in \mathbb{R}^n$, $H \in \mathbb{R}^{n\times n}$ is symmetric and positive definite, $A \in \mathbb{R}^{m\times n}$ with $m \leq n$, $b \in \mathbb{R}^m$, and $\{\ell, u\} \subset \overline{\mathbb{R}}^n$ (i.e., the set of n-dimensional extended real numbers, which includes infinite values). We assume $\ell < u$ and that A has full row-rank, all of which can be guaranteed by preprocessing the data and removing fixed variables. If (1) is feasible, then the unique solution x_* is the point at which there exist Lagrange multipliers (y_*, z^ℓ_*, z^u_*) such that $(x_*, y_*, z^\ell_*, z^u_*)$ satisfies the Karush-Kuhn-Tucker (KKT) conditions for (1), which is to say that

$$0 = \text{KKT}(x, y, z^\ell, z^u) := \left\| \begin{pmatrix} Hx + c - A^Ty - z^\ell + z^u \\ Ax - b \\ \min\{x - \ell, z^\ell\} \\ \min\{u - x, z^u\} \end{pmatrix} \right\|. \tag{2}$$

(The norm used in the definition of the function KKT can be any vector norm on $\mathbb{R}^n$; our only requirement is that the same norm is used in the definition of the residual function r defined in equation (8) later on.) On the other hand, if problem (1) is infeasible, then this can be detected by solving a traditional "Phase 1" linear optimization problem (LP) to find a point that minimizes violations of the constraints; see, e.g., [16]. For simplicity in the remainder of our algorithmic development and analysis, we ignore the possibility of having an infeasible instance of problem (1), but note that we have implemented such an infeasibility detection strategy in our implementation described in §4.

Let the sets of variable and equality constraint indices, respectively, be

$$\mathcal{N} := \{1, \ldots, n\} \text{ and } \mathcal{M} := \{1, \ldots, m\}.$$

We associate with the point x_* the mutually exclusive and exhaustive subsets

$$\begin{aligned} \mathcal{A}_*^\ell &:= \{i \in \mathcal{N} : [x_*]_i = \ell_i\}, \\ \mathcal{A}_*^u &:= \{i \in \mathcal{N} : [x_*]_i = u_i\}, \\ \text{and} \quad \mathcal{I}_* &:= \{i \in \mathcal{N} : \ell_i < [x_*]_i < u_i\}, \end{aligned}$$

representing the subsets of indices corresponding to the lower-active, upper-active, and inactive optimal primal variables, respectively.

The iterates of our framework constitute a sequence of index sets given as $\{(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)\}_{k \geq 0}$, where for each $k \geq 0$ the sets $\mathcal{A}_k^\ell$, $\mathcal{A}_k^u$, $\mathcal{I}_k$, and $\mathcal{U}_k$ are mutually exclusive and exhaustive subsets representing a *partition* of the set of primal variable indices $\mathcal{N}$. Our use of index sets as iterates makes our approach differ from many algorithms whose iterates are the primal-dual variables themselves, but we make this choice as, in our framework, values of the primal-dual variables are uniquely determined by the index sets. The first three components of each iterate, namely $\mathcal{A}_k^\ell$, $\mathcal{A}_k^u$, and $\mathcal{I}_k$, are commonly defined in active-set methods and represent estimates of $\mathcal{A}_*^\ell$, $\mathcal{A}_*^u$, and $\mathcal{I}_*$, respectively. On the other hand, the auxiliary set $\mathcal{U}_k$ (also referred to as the uncertain set) contains the indices of variables whose bounds will be enforced explicitly when the corresponding primal-dual solution is computed. As illustrated by Theorem 1 (on page 7) and the results in §3, our use of $\mathcal{U}_k$ allows for global convergence guarantees for our framework, while the numerical results in §5 illustrate that these guarantees are typically attained at modest extra computational cost (as compared to the costs when $\mathcal{U}_k$ is empty).

If equality constraints are present (i.e., if $m \neq 0$), then precautions should be taken to ensure that each iteration of our method is well-defined. Specifically, each iteration of our framework requires that we have a *feasible partition*, i.e., a partition $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U})$ such that there exists $(x_\mathcal{I}, x_\mathcal{U})$ satisfying

$$A_{\mathcal{M},\mathcal{I}} x_\mathcal{I} + A_{\mathcal{M},\mathcal{U}} x_\mathcal{U} = b - A_{\mathcal{M},\mathcal{A}^\ell} \ell_{\mathcal{A}^\ell} - A_{\mathcal{M},\mathcal{A}^u} u_{\mathcal{A}^u} \quad \text{and} \quad \ell_\mathcal{U} \leq x_\mathcal{U} \leq u_\mathcal{U}. \tag{3}$$

Algorithm 1, below, is employed at the beginning of each iteration of our framework in order to transform a given iterate $(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$ into a feasible partition. (We drop the iteration number subscript in the algorithm as it is inconsequential in this subroutine. Moreover, at this point we do not provide specific strategies for choosing the sets $\mathcal{S}^\ell$, $\mathcal{S}^u$, $\mathcal{S}^\mathcal{I}$, and $\mathcal{S}^\mathcal{U}$ in Algorithm 1; we leave such details until §4 where the approach employed in our implementation is described.) Note that if $m = 0$, then (3) reduces to $\ell_\mathcal{U} \leq x_\mathcal{U} \leq u_\mathcal{U}$. In such cases, preprocessing the

data for problem (1) (i.e., to ensure $\ell < u$) has guaranteed that each iterate is a feasible partition, so running Algorithm 1 is unnecessary. Otherwise, if $m > 0$, then Algorithm 1 is well-defined and will produce a feasible partition for any input. This can be seen by the fact that, in the worst case, the algorithm will eventually have $\mathcal{A}^\ell \cup \mathcal{A}^u = \emptyset$, in which case the feasibility of (1) implies that (3) is satisfiable.

Algorithm 1 Transformation to a feasible partition (Feas)

1: Input $(\overline{\mathcal{A}}^\ell, \overline{\mathcal{A}}^u, \overline{\mathcal{I}}, \overline{\mathcal{U}})$ and initialize $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U}) \leftarrow (\overline{\mathcal{A}}^\ell, \overline{\mathcal{A}}^u, \overline{\mathcal{I}}, \overline{\mathcal{U}})$.
2: **while** (3) is not satisfiable **do**
3: Choose any $\mathcal{S}^\ell \subseteq \mathcal{A}^\ell$ and $\mathcal{S}^u \subseteq \mathcal{A}^u$ such that $\mathcal{S} \leftarrow \mathcal{S}^\ell \cup \mathcal{S}^u \neq \emptyset$.
4: Set $\mathcal{A}^\ell \leftarrow \mathcal{A}^\ell \backslash \mathcal{S}^\ell$ and $\mathcal{A}^u \leftarrow \mathcal{A}^u \backslash \mathcal{S}^u$.
5: Choose any $(\mathcal{S}^\mathcal{I}, \mathcal{S}^\mathcal{U}) \subseteq \mathcal{S} \times \mathcal{S}$ such that $\mathcal{S}^\mathcal{I} \cup \mathcal{S}^\mathcal{U} = \mathcal{S}$ and $\mathcal{S}^\mathcal{I} \cap \mathcal{S}^\mathcal{U} = \emptyset$.
6: Set $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{S}^\mathcal{I}$ and $\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{S}^\mathcal{U}$.
7: **end while**
8: Return $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U}) =: \text{Feas}(\overline{\mathcal{A}}^\ell, \overline{\mathcal{A}}^u, \overline{\mathcal{I}}, \overline{\mathcal{U}})$.

Once a feasible partition is obtained, we use Algorithm 2 to compute primal-dual variable values corresponding to the current index sets. The procedure computes the primal-dual variables by minimizing the objective of (1) over subsets of the original variables and constraints, where we have already ensured via Algorithm 1 that the reduced problem (5) is feasible. Again, we drop the iteration number index in Algorithm 2 as it is inconsequential in this subroutine.

Algorithm 2 Subspace minimization (SM)

1: Input $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U})$ such that (3) is satisfiable (i.e., such that (5) is feasible).
2: Set
$$x_{\mathcal{A}^\ell} \leftarrow \ell_{\mathcal{A}^\ell}, \quad x_{\mathcal{A}^u} \leftarrow u_{\mathcal{A}^u}, \quad z^\ell_{\mathcal{I} \cup \mathcal{A}^u} \leftarrow 0, \text{ and } z^u_{\mathcal{I} \cup \mathcal{A}^\ell} \leftarrow 0. \tag{4}$$
3: Let $\mathcal{A} \leftarrow \mathcal{A}^\ell \cup \mathcal{A}^u$, $\mathcal{F} \leftarrow \mathcal{I} \cup \mathcal{U}$, and $(x_\mathcal{F}, y, z^\ell_\mathcal{U}, z^u_\mathcal{U})$ be the optimal primal-dual solution of
$$\begin{array}{ll} \underset{x_\mathcal{F} \in \mathbb{R}^{|\mathcal{F}|}}{\text{minimize}} & \frac{1}{2} x_\mathcal{F}^T H_{\mathcal{F},\mathcal{F}} x_\mathcal{F} + x_\mathcal{F}^T (c_\mathcal{F} + H_{\mathcal{F},\mathcal{A}} x_\mathcal{A}) \\ \text{subject to} & A_{\mathcal{M},\mathcal{F}} x_\mathcal{F} = b - A_{\mathcal{M},\mathcal{A}} x_\mathcal{A}, \quad \ell_\mathcal{U} \leq x_\mathcal{U} \leq u_\mathcal{U}. \end{array} \tag{5}$$
4: Set
$$z^\ell_{\mathcal{A}^\ell} \leftarrow [Hx + c - A^T y]_{\mathcal{A}^\ell} \text{ and } z^u_{\mathcal{A}^u} \leftarrow -[Hx + c - A^T y]_{\mathcal{A}^u}. \tag{6}$$
5: Return $(x, y, z^\ell, z^u) =: \text{SM}(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U})$.

The steps of Algorithm 2 are easily described. In Step 2, the components of x in the set $\mathcal{A}^\ell$ ($\mathcal{A}^u$) are fixed at their lower (upper) bounds, and components of z^ℓ (z^u) are fixed to zero corresponding to components of x that are not fixed at their lower (upper) bounds. In Step 3, a reduced QP is solved in the remaining primal variables, i.e., those in $\mathcal{I}$ and $\mathcal{U}$. This step also determines the dual variables for the linear equalities and the bound constraints in $\mathcal{U}$. Finally, in Step 4, we set the dual variables corresponding to those primal variables that were fixed in Step 2. Notice that when $\mathcal{U} = \emptyset$, the solution of (5) reduces to the solution of
$$\begin{pmatrix} H_{\mathcal{I},\mathcal{I}} & A^T_{\mathcal{M},\mathcal{I}} \\ A_{\mathcal{M},\mathcal{I}} & 0 \end{pmatrix} \begin{pmatrix} x_\mathcal{I} \\ -y \end{pmatrix} = - \begin{pmatrix} c_\mathcal{I} + H_{\mathcal{I},\mathcal{A}} x_\mathcal{A} \\ A_{\mathcal{M},\mathcal{A}} x_\mathcal{A} - b \end{pmatrix}. \tag{7}$$

This observation is critical as it shows that, whenever $\mathcal{U}_k = \emptyset$ in our framework, the computational cost of Algorithm 2 is dominated by that of solving a reduced linear system. In practice, the framework chooses $\mathcal{U}_0$ to be empty, and only introduces indices into $\mathcal{U}_k$ if necessary to ensure convergence.

The following lemma shows a critical feature of the output of Algorithm 2. (Recall that we require the vector norm in equation (8) to be that used in the definition of the KKT function defined in equation (2).)

Lemma 1 *If* $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U})$ *is feasible and* $(x, y, z^\ell, z^u) \leftarrow \mathrm{SM}(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U})$*, then* $r(x, y, z^\ell, z^u) = \mathrm{KKT}(x, y, z^\ell, z^u)$*, where*

$$r(x, y, z^\ell, z^u) := \left\| \begin{pmatrix} \min\{z^\ell_{\mathcal{A}^\ell}, 0\} \\ \min\{z^u_{\mathcal{A}^u}, 0\} \\ \min\{0, [x - \ell]_{\mathcal{I}}, [u - x]_{\mathcal{I}}\} \end{pmatrix} \right\|. \tag{8}$$

Proof The proof follows by straightforward comparison of $\mathrm{KKT}(x, y, z^\ell, z^u)$ with (4), the optimality conditions of (5), and (6). In particular, these conditions guarantee that certain elements of the vector in the definition of $\mathrm{KKT}(x, y, z^\ell, z^u)$ are equal to zero; the only potentially nonzero elements are those in the vector defining the residual value $r(x, y, z^\ell, z^u)$. □

It follows from Lemma 1 that if the input $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U})$ to Algorithm 2 yields (x, y, z^ℓ, z^u) with $\ell_{\mathcal{I}} \le x_{\mathcal{I}} \le u_{\mathcal{I}}$, $z^\ell_{\mathcal{A}^\ell} \ge 0$, and $z^u_{\mathcal{A}^u} \ge 0$, then (x, y, z^ℓ, z^u) is the solution to (1). This follows as the procedure in Algorithm 2 ensures that the resulting primal-dual vector satisfies the first two blocks of equations in (2) as well as complementarity of the primal and dual variables. The only conditions in (2) that it does not guarantee for each partition are primal and dual variable bounds.

We now state our algorithmic framework.

Algorithm 3 Primal-dual active-set framework (PDAS)

1: Input $(\mathcal{A}^\ell_0, \mathcal{A}^u_0, \mathcal{I}_0, \mathcal{U}_0)$ and initialize $k \leftarrow 0$.
2: **loop**
3: Set $(\mathcal{A}^\ell_k, \mathcal{A}^u_k, \mathcal{I}_k, \mathcal{U}_k) \leftarrow \mathrm{Feas}(\mathcal{A}^\ell_k, \mathcal{A}^u_k, \mathcal{I}_k, \mathcal{U}_k)$ by Algorithm 1.
4: Set $(x_k, y_k, z^\ell_k, z^u_k) \leftarrow \mathrm{SM}(\mathcal{A}^\ell_k, \mathcal{A}^u_k, \mathcal{I}_k, \mathcal{U}_k)$ by Algorithm 2.
5: If $r(x_k, y_k, z^\ell_k, z^u_k) = 0$, then **break**.
6: Choose $(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$, then set $k \leftarrow k + 1$.
7: **end loop**
8: Return $(x_k, y_k, z^\ell_k, z^u_k)$.

Although we have yet to state specific strategies for updating the index sets in Step 6 of Algorithm 3, we can prove that it terminates and returns the optimal solution of (1) as long as, for some $k \ge 0$, we have

$$\mathcal{A}^\ell_k \subseteq \mathcal{A}^\ell_*, \quad \mathcal{A}^u_k \subseteq \mathcal{A}^u_*, \text{ and } \mathcal{I}_k \subseteq \mathcal{I}_*. \tag{9}$$

In other words, Algorithm 3 produces $(x_*, y_*, z^\ell_*, z^u_*)$ satisfying (2) if, for some iteration number $k \ge 0$, the algorithm generates subsets of the optimal index sets.

Theorem 1 *If problem* (1) *is feasible and the* k*th iterate of Algorithm 3 satisfies* (9)*, then* $r(x_k, y_k, z^\ell_k, z^u_k) = 0$ *and* $(x_k, y_k, z^\ell_k, z^u_k)$ *solves problem* (1).

Proof Our strategy of proof is as follows. First, we will show that if (9) holds, then $(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k) = \text{Feas}(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$, which will imply that Algorithm 1 has no effect on such an iterate in Step 3 of Algorithm 3. Second, we will show that $(x_*, y_*, z_*^\ell, z_*^u)$ satisfies (4)–(6), where in the case of (5) we mean that the primal-dual optimality conditions for the subproblem are satisfied. Since the vector $(x_k, y_k, z_k^\ell, z_k^u)$ is *uniquely* defined by (4)–(6), it will then follow that $(x_k, y_k, z_k^\ell, z_k^u) = (x_*, y_*, z_*^\ell, z_*^u)$, which is the desired result.

To show that, if (9) holds, then Algorithm 1 will have no effect on the partition, we will show that—due to (9)—the point $(x_*, y_*, z_*^\ell, z_*^u)$ satisfies (3). Since this will imply that (3) is satisfiable, it will follow that the **while** loop in Algorithm 1 will not be entered, and hence the initial partition given to Algorithm 1 will be returned. Let $\mathcal{A}_k \leftarrow \mathcal{A}_k^\ell \cup \mathcal{A}_k^u$ and $\mathcal{F}_k \leftarrow \mathcal{I}_k \cup \mathcal{U}_k$ (as in Algorithm 2). It follows from the KKT conditions (2) that $Ax_* = b$, which with condition (9) implies

$$\begin{aligned} A_{\mathcal{M},\mathcal{F}_k}[x_*]_{\mathcal{F}_k} &= b - A_{\mathcal{M},\mathcal{A}_k^\ell}[x_*]_{\mathcal{A}_k^\ell} - A_{\mathcal{M},\mathcal{A}_k^u}[x_*]_{\mathcal{A}_k^u} \\ &= b - A_{\mathcal{M},\mathcal{A}_k^\ell}\ell_{\mathcal{A}_k^\ell} - A_{\mathcal{M},\mathcal{A}_k^u}u_{\mathcal{A}_k^u}. \end{aligned} \tag{10}$$

In addition, (2) implies $\ell \leq x_* \leq u$, with which we find

$$\ell_{\mathcal{U}_k} \leq [x_*]_{\mathcal{U}_k} \leq u_{\mathcal{U}_k}. \tag{11}$$

Hence, $(x_*, y_*, z_*^\ell, z_*^u)$ satisfies (3), so Algorithm 1 does not modify $(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$.

Next, we show that $(x_*, y_*, z_*^\ell, z_*^u)$ satisfies (4)–(6). It follows from (9) that

$$[x_*]_{\mathcal{A}_k^\ell} = \ell_{\mathcal{A}_k^\ell}, \quad [x_*]_{\mathcal{A}_k^u} = u_{\mathcal{A}_k^u}, \quad [z_*^\ell]_{\mathcal{I}_k \cup \mathcal{A}_k^u} = 0, \text{ and } [z_*^u]_{\mathcal{I}_k \cup \mathcal{A}_k^\ell} = 0, \tag{12}$$

so (4) is satisfied. It then also follows from (2) and (12) that

$$\begin{aligned} &\begin{pmatrix} H_{\mathcal{I}_k\mathcal{I}_k} & H_{\mathcal{I}_k\mathcal{U}_k} \\ H_{\mathcal{U}_k\mathcal{I}_k} & H_{\mathcal{U}_k\mathcal{U}_k} \end{pmatrix} \begin{pmatrix} [x_*]_{\mathcal{I}_k} \\ [x_*]_{\mathcal{U}_k} \end{pmatrix} + \begin{pmatrix} H_{\mathcal{I}_k\mathcal{A}_k} \\ H_{\mathcal{U}_k\mathcal{A}_k} \end{pmatrix} [x_*]_{\mathcal{A}_k} \\ &\qquad + \begin{pmatrix} c_{\mathcal{I}_k} \\ c_{\mathcal{U}_k} \end{pmatrix} - \begin{pmatrix} A_{\mathcal{M},\mathcal{I}_k}^T \\ A_{\mathcal{M},\mathcal{U}_k}^T \end{pmatrix} y_* + \begin{pmatrix} 0 \\ [z_*^u - z_*^\ell]_{\mathcal{U}_k} \end{pmatrix} = 0. \end{aligned} \tag{13}$$

Furthermore,

$$\text{if } i \in \mathcal{U}_k \cap \mathcal{I}_*, \text{ then } [x_*]_i \in (l_i, u_i) \text{ and } [z_*^\ell]_i = [z_*^u]_i = 0; \tag{14a}$$

$$\text{if } i \in \mathcal{U}_k \cap \mathcal{A}_*^\ell, \text{ then } [x_*]_i = \ell_i,\ [z_*^\ell]_i \geq 0, \text{ and } [z_*^u]_i = 0; \tag{14b}$$

$$\text{if } i \in \mathcal{U}_k \cap \mathcal{A}_*^u, \text{ then } [x_*]_i = u_i,\ [z_*^\ell]_i = 0, \text{ and } [z_*^u]_i \geq 0. \tag{14c}$$

Thus, it follows from (10)–(14) that $[x_*]_{\mathcal{F}_k}$ is the unique solution to (5) with associated dual values $(y_*, [z_*^\ell]_{\mathcal{U}_k}, [z_*^u]_{\mathcal{U}_k})$. Finally, from (2) and (12) we have

$$[z_*^\ell]_{\mathcal{A}_k^\ell} = \ [Hx_* + c - A^T y_* + z_*^u]_{\mathcal{A}_k^\ell} = \ [Hx_* + c - A^T y_*]_{\mathcal{A}_k^\ell} \tag{15a}$$

$$\text{and} \quad [z_*^u]_{\mathcal{A}_k^u} = -[Hx_* + c - A^T y_* - z_*^\ell]_{\mathcal{A}_k^u} = -[Hx_* + c - A^T y_*]_{\mathcal{A}_k^u}. \tag{15b}$$

We may now conclude from (10)–(15) that $(x_*, y_*, z_*^\ell, z_*^u)$ satisfies (4)–(6), which are uniquely satisfied by $(x_k, y_k, z_k^\ell, z_k^u)$. Thus, $(x_k, y_k, z_k^\ell, z_k^u) = (x_*, y_*, z_*^\ell, z_*^u)$. □

Note that due to potential degeneracy, condition (9) is not necessary for Algorithm 3 to terminate, though it is sufficient. For example, for $i \in \mathcal{A}_*^\ell$, we may find that with $i \in \mathcal{I}_k$ we obtain $[x_k]_i = \ell_i$ from (5). This means that Algorithm 3 may terminate with a solution despite an index corresponding to an active bound being placed in $\mathcal{I}_k$, meaning that $\mathcal{I}_k \not\subset \mathcal{I}_*$. We remark, however, that this type of case does not inhibit us from proving global convergence for our methods in §3.

Now that our algorithmic framework has been established in Algorithm 3, we comment further on comparisons of our approach and a few related methods in the literature. First, the algorithms in [2,3,28] have a similar form to Algorithm 3 and employ index set updates that are similar to that in Algorithm 4, described in the following section. However, as these algorithms do not aim to solve generally constrained convex QPs, they do not possess global convergence guarantees for problem (1). Rather, they guarantee convergence only for certain classes of convex QPs where the Hessian of the objective function satisfies certain properties. Another related method is that proposed in [24]—which is based on the method proposed in [23]—in which the authors propose a dual active-set algorithm (DASA) which can solve generally constrained convex QPs. Fundamentally, DASA and our PDAS framework differ in various ways. First, they differ in the structure of the subproblems that arise in the algorithm. Specifically, an iteration of DASA involves solving a linear system in the dual space, performing a line search, and—if the algorithm has reached a maximum of a modified dual function—solving a bound-constrained problem in the primal space to check for optimality. By contrast, the subproblems solved in PDAS involve primal and dual variables and the original equality constraints from (1), but with some primal variables fixed at their bounds and some bound constraints from (1) ignored. Second, the strategies for updating the indexing sets in DASA are quite different than those employed in PDAS: DASA uses the dual variables associated with the equality constraints to generate a new active-set estimate, whereas the strategies that we propose in the following section merely use the equality constraint multipliers to measure the KKT error. Finally, DASA and our PDAS framework handle the potential infeasibility of problem (1) differently: DASA uses a proximal point approach to avoid unbounded dual subproblems, whereas PDAS employs an initial infeasibility detection phase and Algorithm 1 to guarantee that (5) is feasible.

3 Strategies for Updating the Indexing Sets

In this section, we describe several strategies for updating the index sets in Algorithm 3. That is, we describe subroutines to be employed in Step 6 of Algorithm 3 to choose iterate $k+1$. Recall that Theorem 1 shows that if an update yields (9), then Algorithm 3 will terminate with a solution to problem (1). We begin by describing an extension of the strategy of Hintermüller, Ito, and Kunisch [25] that yields this behavior in special cases, but not for all strictly convex QPs. We then describe two novel techniques that yield global convergence in the general case.

3.1 Hintermüller, Ito, and Kunisch update

The first strategy we describe, written as Algorithm 4 below, is an extension of the technique used in the method introduced by Hintermüller, Ito, and Kunisch [25]. In particular, if $m = 0$, $\ell_i = -\infty$ for all $i \in \mathcal{N}$, and $\mathcal{U}_0 \leftarrow \emptyset$, then Algorithm 3 with Step 6 employing Algorithm 4 is identical to the algorithm in [25, Section 2].

Algorithm 4 Updating strategy inspired by Hintermüller, Ito, and Kunisch [25]

1: Input $(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$ and $(x_k, y_k, z_k^\ell, z_k^u)$.
2: Set

$$\mathcal{U}_{k+1} \leftarrow \mathcal{U}_k, \tag{16a}$$

$$\mathcal{A}_{k+1}^\ell \leftarrow \{i \in (\mathcal{N} \setminus \mathcal{U}_{k+1}) : [x_k]_i < \ell_i, \text{ or } i \in \mathcal{A}_k^\ell \text{ and } [z_k^\ell]_i \geq 0\}, \tag{16b}$$

$$\mathcal{A}_{k+1}^u \leftarrow \{i \in (\mathcal{N} \setminus \mathcal{U}_{k+1}) : [x_k]_i > u_i, \text{ or } i \in \mathcal{A}_k^u \text{ and } [z_k^u]_i \geq 0\}, \tag{16c}$$

$$\text{and } \mathcal{I}_{k+1} \leftarrow \mathcal{N} \setminus (\mathcal{U}_{k+1} \cup \mathcal{A}_{k+1}^\ell \cup \mathcal{A}_{k+1}^u). \tag{16d}$$

3: Return $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.

The following is an example of a result that can be proved with this strategy when H is assumed to be a perturbation of an M-matrix; see [25, Theorem 3.4]. A matrix M is said to be an M-matrix if it is positive definite and $M_{i,j} \leq 0$ for all $i \neq j$, from which it can be shown that $M^{-1} \geq 0$.

Theorem 2 *Suppose $m = 0$, $\ell_i = -\infty$ for all $i \in \mathcal{N}$, $\mathcal{U}_0 \leftarrow \emptyset$, and $H = M + K$ where $M \in \mathbb{R}^{n \times n}$ is an M-matrix and $K \in \mathbb{R}^{n \times n}$. If $\|K\|_1$ is sufficiently small, then problem (1) has a unique solution (x_*, z_*^ℓ, z_*^u) and Algorithm 3 with Step 6 employing Algorithm 4 yields $(x_k, z_k^\ell, z_k^u) = (x_*, z_*^\ell, z_*^u)$ for some $k \geq 0$.*

We also show in Appendix A that Algorithm 3 with $\mathcal{U}_k = \emptyset$ for all $k \geq 0$ is equivalent to a semi-smooth Newton method. Hintermüller, Ito, and Kunisch state similar results for the case $m = 0$. It should be noted, however, that their proof actually only shows that iterations *after* the first one are equivalent to a semi-smooth Newton method. This caveat is important as they use this equivalence to prove that their method converges superlinearly from any starting point sufficiently close to the solution. Such a result can only be true for the iterate *after* their initial iterate due to the manner in which their method is initialized. For example, consider a starting point obtained by perturbing the optimal solution by an arbitrarily small amount into the strict interior of the feasible region. Their algorithm would then begin by computing the *unconstrained* minimizer of the quadratic objective, which can be arbitrarily far from the solution, meaning that fast local convergence could not commence until the algorithm produces another iterate within a sufficiently small neighborhood of the optimal solution. It should also be noted that, since their algorithm converges (in special cases) in a finite number of iterations, the fact that it converges superlinearly is immediate, regardless of [25, Theorem 3.1].

The main disadvantage of the strategy in Algorithm 4 when $\mathcal{U}_0 = \emptyset$ is that it does not guarantee convergence for all strictly convex quadratic objective functions. This should not be a surprise as Theorem 2 makes the assumption that H is a (perturbed) M-matrix, which is restrictive. The following three-dimensional

example shows that the strategy may fail if H is positive definite, but not an M-matrix. We provide this example as we also use it later on to show that our techniques (which may set $\mathcal{U}_k \neq \emptyset$) yield convergence on this same problem.

Example 1 Consider problem (1) with

$$m = 0, \quad H = \begin{pmatrix} 4 & 5 & -5 \\ 5 & 9 & -5 \\ -5 & -5 & 7 \end{pmatrix}, \quad c = \begin{pmatrix} 2 \\ 1 \\ -3 \end{pmatrix}, \quad \ell = \begin{pmatrix} -\infty \\ -\infty \\ -\infty \end{pmatrix}, \quad \text{and} \quad u = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{17}$$

and the initial index sets

$$(\mathcal{A}_0^\ell, \mathcal{A}_0^u, \mathcal{I}_0, \mathcal{U}_0) = (\emptyset, \emptyset, \{1,2,3\}, \emptyset).$$

Table 1 illustrates that the updating strategy in Algorithm 4 generates a cycle and thus fails to provide the optimal solution. In particular, the iterates in iterations 1 and 4 are identical, which indicates that a cycle will continue to occur. Indeed, if any of the index partitions that define iterations 0–3 are used as the initial partition, then the updating strategy generates the same cycle. Since there are 8 possible initial index sets for this problem (with $\mathcal{U}_0 = \emptyset$), it follows that at least half of them will lead to failure.

Table 1 Result of Algorithm 3 employed to solve the problem in Example 1 when iterates are updated via Algorithm 4; the iterates cycle indefinitely.

k	$\mathcal{A}_k^\ell$	$\mathcal{A}_k^u$	$\mathcal{I}_k$	$\mathcal{U}_k$	x_k	z_k^u
0	$\emptyset$	$\emptyset$	$\{1,2,3\}$	$\emptyset$	$(-3,1,-1)$	$(0,0,0)$
1	$\emptyset$	$\{2\}$	$\{1,3\}$	$\emptyset$	$(\frac{1}{3},0,\frac{2}{3})$	$(0,\frac{2}{3},0)$
2	$\emptyset$	$\{1,2,3\}$	$\emptyset$	$\emptyset$	$(0,0,0)$	$(-2,-1,3)$
3	$\emptyset$	$\{3\}$	$\{1,2\}$	$\emptyset$	$(-\frac{13}{11},\frac{6}{11},0)$	$(0,0,-\frac{2}{11})$
4	$\emptyset$	$\{2\}$	$\{1,3\}$	$\emptyset$	$(\frac{1}{3},0,\frac{2}{3})$	$(0,\frac{2}{3},0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

We remark that it can be shown that the strategy in Algorithm 4 will lead to convergence for any strictly-convex QP when $n \leq 2$; hence, we created this example with $n = 3$. See also [13, Proposition 4.1].

To prevent cycling and ensure convergence from arbitrary initial iterates for $n \geq 3$, the following subsections focus on two updating strategies for the index sets that allow for the size of $\mathcal{U}_k$ to be changed (if needed) as the iterations proceed.

3.2 Update based on monitoring index set changes

The goal of our first new updating strategy for the index sets is to ensure that (9) is eventually satisfied. This is desirable as then convergence of the associated iterates is guaranteed by Theorem 1. The strategy is based on a simple observation: Since there are only a finite number of distinct choices of the index sets, condition (9) will eventually be satisfied if we prevent an infinite number of cycles of any length.

Of course, taking action only after a cycle has occurred may be inefficient for large-scale problems as the cycle lengths can be exceedingly large. However, we can (preemptively) avoid cycling by monitoring the number of times indices have moved between the index sets. This idea also ties in with our numerical experience as we have often found that when the method in the previous section does not yield convergence, it is primarily due to a handful of indices that do not quickly settle into an index set. Variables that change index sets numerous times may be considered sensitive and are prime candidates for membership in $\mathcal{U}_k$.

To keep track of the number of times each index changes between index sets, we define a counter sequence $\{q_k^i\}$ for each $i \in \mathcal{N}$. Using these counters to monitor the number of times each index changes set membership, we obtain the updating strategy in Algorithm 5. (The algorithm is written to be as generic as possible, but precise strategies for choosing $\mathcal{S}^\ell$, $\mathcal{S}^u$, and $\mathcal{S}^\mathcal{I}$ and updating the counters in Step 5 is given in §4.) The motivation for the strategy is to mimic Algorithm 4, but to add an index (or indices) to $\mathcal{U}_{k+1}$ only if an index has changed index set membership too many times as determined by a tolerance $q^{\max} \geq 1$. Note that when an index is moved into $\mathcal{U}_{k+1}$, the counters (for all indices) may be reduced or reset to avoid indices being moved into $\{\mathcal{U}_k\}$ too frequently. (Again, due to the computational costs of solving instances of (5), we remark that it is desirable to keep the elements of the sequence $\{|\mathcal{U}_k|\}$ small.)

Algorithm 5 Updating strategy based on monitoring index set changes

1: Input $(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$, $(x_k, y_k, z_k^\ell, z_k^u)$, $(q_k^1, \ldots, q_k^n)$, and $q^{\max} \geq 1$.
2: **if** $q_k^i \geq q^{\max}$ for some $i \in \mathcal{N}$ **then**
3: Choose $\mathcal{S}^\ell \subseteq \mathcal{A}_k^\ell$, $\mathcal{S}^u \subseteq \mathcal{A}_k^u$, and $\mathcal{S}^\mathcal{I} \subseteq \mathcal{I}_k$ such that $\mathcal{S} \leftarrow \mathcal{S}^\ell \cup \mathcal{S}^u \cup \mathcal{S}^\mathcal{I} \neq \emptyset$.
4: Set $\mathcal{A}_{k+1}^\ell \leftarrow \mathcal{A}_k^\ell \backslash \mathcal{S}^\ell$, $\mathcal{A}_{k+1}^u \leftarrow \mathcal{A}_k^u \backslash \mathcal{S}^u$, $\mathcal{I}_{k+1} \leftarrow \mathcal{I}_k \backslash \mathcal{S}^\mathcal{I}$, and $\mathcal{U}_{k+1} \leftarrow \mathcal{U}_k \cup \mathcal{S}$.
5: Set $q_{k+1}^i \in \{0, \ldots, q_k^i\}$ for all $i \in \mathcal{N}$.
6: **else**
7: Set $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$ by (16).
8: Set $q_{k+1}^i \leftarrow q_k^i + 1$ for all $i \in \mathcal{A}_{k+1}^\ell$ such that $i \notin \mathcal{A}_k^\ell$.
9: Set $q_{k+1}^i \leftarrow q_k^i + 1$ for all $i \in \mathcal{A}_{k+1}^u$ such that $i \notin \mathcal{A}_k^u$.
10: Set $q_{k+1}^i \leftarrow q_k^i + 1$ for all $i \in \mathcal{I}_{k+1}$ such that $i \notin \mathcal{I}_k$.
11: **end if**
12: Return $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.

The following lemma shows that Algorithm 5 guarantees that every iteration will either move a nonempty subset of indices into the uncertain set, or our index counters will increase.

Lemma 2 *Suppose that problem* (1) *is feasible. If Algorithm 3 does not terminate before or in iteration k, then by employing Algorithm 5 in Step 6, we have that either $|\mathcal{U}_{k+1}| > |\mathcal{U}_k|$ or $q_{k+1}^i > q_k^i$ for at least some $i \in \mathcal{N}$.*

Proof If $\mathcal{U}_k = \mathcal{N}$, then (3) is satisfiable and as a result of solving (5) we obtain $(x_*, y_*, z_*^\ell, z_*^u)$ solving (1). Consequently, Algorithm 3 terminates in iteration k. Thus, we can assume that $\mathcal{U}_k \neq \mathcal{N}$. Moreover, if $q_k^i \geq q^{\max}$ for some $i \in \mathcal{N}$, then it is clear that Algorithm 5 will yield $|\mathcal{U}_{k+1}| > |\mathcal{U}_k|$. Thus, we may assume that $q_k^i < q^{\max}$ for all $i \in \mathcal{N}$. All that remains is to show that, under the assumptions of the lemma, (16) will change at least one index from some index set to another.

To derive a contradiction, suppose that (16) yields $(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1}) = (\mathcal{A}^\ell_k, \mathcal{A}^u_k, \mathcal{I}_k, \mathcal{U}_k)$. By the procedure in Algorithm 2, it follows that

$$[x_k]_{\mathcal{A}^\ell_k} = [\ell]_{\mathcal{A}^\ell_k}, \quad [x_k]_{\mathcal{A}^u_k} = [u]_{\mathcal{A}^u_k}, \quad [z^\ell_k]_{\mathcal{I}_k \cup \mathcal{A}^u_k} = 0, \text{ and } [z^u_k]_{\mathcal{I}_k \cup \mathcal{A}^\ell_k} = 0, \tag{18}$$

and the optimality conditions for subproblem (5) ensure that

$$\min([x_k - \ell]_{\mathcal{U}_k}, [z^\ell_k]_{\mathcal{U}_k}) = \min([u - x_k]_{\mathcal{U}_k}, [z^u_k]_{\mathcal{U}_k}) = 0. \tag{19}$$

Algorithm 5 and the assumption that there are no set changes imply that

$$\mathcal{I}_{k+1} = \{i : i \notin (\mathcal{A}^\ell_{k+1} \cup \mathcal{A}^u_{k+1} \cup \mathcal{U}_{k+1})\} = \mathcal{I}_k,$$

from which we conclude that

$$[\ell]_{\mathcal{I}_k} \leq [x_k]_{\mathcal{I}_k} \leq [u]_{\mathcal{I}_k}. \tag{20}$$

Similarly, Algorithm 5 and the same assumption imply that

$$\begin{aligned} \mathcal{A}^\ell_{k+1} &= \{i : [x_k]_i < \ell_i, \text{ or } i \in \mathcal{A}^\ell_k \text{ and } [z^\ell_k]_i \geq 0\} = \mathcal{A}^\ell_k \\ \text{and} \quad \mathcal{A}^u_{k+1} &= \{i : [x_k]_i > u_i, \text{ or } i \in \mathcal{A}^u_k \text{ and } [z^u_k]_i \geq 0\} = \mathcal{A}^u_k \end{aligned}$$

so that

$$[z^\ell_k]_{\mathcal{A}^\ell_k} \geq 0 \text{ and } [z^u_k]_{\mathcal{A}^u_k} \geq 0. \tag{21}$$

With the residual function r defined by (8), it follows from (18)–(21) that we have $r(x_k, y_k, z^\ell_k, z^u_k) = 0$, which contradicts the assumption of the lemma that says that Algorithm 3 does not terminate in iteration k. □

The global convergence of Algorithm 3 with the updating strategy in Algorithm 5 can now be proved from arbitrary initial iterates. The simple, key idea to the proof is that the monotonic nondecrease of the size of the auxiliary set will, in the worst case, eventually lead to $\mathcal{U}_k = \mathcal{N}$ for some large k, in which case the algorithm would produce the optimal solution (and terminate) in iteration k. We stress, however, that while the proof relies on such worst-case behavior, we have never witnessed such an occurrence in practice. Indeed, as seen in our experiments in §5, we rarely find the auxiliary set ever including more than a few indices.

Theorem 3 *If problem* (1) *is feasible, then Algorithm 3 with Step 6 employing Algorithm 5 solves problem* (1) *in a finite number of iterations.*

Proof To derive a contradiction, suppose that Algorithm 3 computes infinitely many iterates. Then, by Lemma 2, each iteration will either yield $|\mathcal{U}_{k+1}| > |\mathcal{U}_k|$ or $q^i_{k+1} > q^i_k$ for at least some $i \in \mathcal{N}$. If the algorithm continues without terminating, then eventually the increases in the components of the elements of $\{(q^1_k, \ldots, q^n_k)\}$ will yield $\mathcal{U}_k = \mathcal{N}$ for some sufficiently large k. However, as in the proof of Lemma 2, this means that (3) will be satisfiable, solving (5) will yield $(x_*, y_*, z^\ell_*, z^u_*)$ solving (1), and the algorithm will terminate, contradicting the supposition that an infinite number of iterations will be performed. □

Table 2 shows the behavior of Algorithm 3 on the problem in Example 1 given on page 11. By using Algorithm 5 to update the index sets with $q^{\max} \leftarrow 1$ (and setting q^i_{k+1} to zero whenever an element is moved into $\mathcal{U}_{k+1}$), the algorithm converges to the solution of the problem.

Table 2 Result of Algorithm 3 employed to solve the problem in Example 1 when iterates are updated via Algorithm 5.

k	$\mathcal{A}_k^\ell$	$\mathcal{A}_k^u$	$\mathcal{I}_k$	$\mathcal{U}_k$	x_k	z_k^u	(q_k^1, q_k^2, q_k^3)
0	$\emptyset$	$\emptyset$	$\{1,2,3\}$	$\emptyset$	$(-3,1,-1)$	$(0,0,0)$	$(0,0,0)$
1	$\emptyset$	$\{2\}$	$\{1,3\}$	$\emptyset$	$(\frac{1}{3},0,\frac{2}{3})$	$(0,\frac{2}{3},0)$	$(0,1,0)$
2	$\emptyset$	$\{1,3\}$	$\emptyset$	$\{2\}$	$(0,-\frac{1}{18},0)$	$(-\frac{31}{18},-\frac{1}{2},\frac{49}{18})$	$(1,0,1)$
3	$\emptyset$	$\{3\}$	$\emptyset$	$\{1,2\}$	$(-\frac{1}{2},0,0)$	$(0,\frac{3}{2},\frac{1}{2})$	$(0,0,0)$

3.3 Update based on reducing the KKT error

The updating strategy described in this section is based on a technique for ensuring reductions in the KKT residual. In particular, we adopt a nonmonotonic watch-dog strategy employed in various optimization methods [9,19,22,34,35]. Since there are only a finite number of partitions of the index sets, by ensuring that the KKT residual is reduced over sequences of iterations, the residual is eventually reduced to zero. As in the strategy in the previous subsection, the aim is to mimic Algorithm 4, but to move an index (or indices) to the uncertain set if the new KKT residual is not sufficiently small. One additional benefit of this approach is that it allows for elements to be *removed* from the uncertain set, which can be done whenever indices remain in the same index set as the residual is reduced.

The steps of Algorithm 6 can be summarized as follows. First, a trial iterate $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$ is chosen via (16) and then the Feas operator, i.e., Algorithm 1, is applied to produce a feasible partition. If the resulting index sets yield (through the SM operator, i.e., Algorithm 2) a primal-dual solution with a corresponding KKT value strictly less than the maximum of the most recent p KKT values, then the algorithm continues with $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$. Otherwise, $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$ is reset to $(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$ and indices are moved to $\mathcal{U}_{k+1}$ until the resulting feasible partition yields a corresponding KKT value less than the maximum of the most recent p KKT values. In either case, once $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$ is chosen in this manner, an optional step is available to (potentially) remove elements from $\mathcal{U}_{k+1}$. If this step is taken, then the primal variables corresponding to the most recent p elements of $\{\mathcal{U}_k\}$ are considered. Specifically, the sets $\mathcal{T}^\ell$, $\mathcal{T}^u$, and $\mathcal{T}^\mathcal{I}$ are constructed, representing indices whose variables have remained at their lower bounds, at their upper bounds, or interior to their bounds, respectively, in the last p iterations. If any of these sets are nonempty, then there is a strong indication that overall computational costs can be reduced by moving elements into $\mathcal{A}_{k+1}^\ell$, $\mathcal{A}_{k+1}^u$, or $\mathcal{I}_{k+1}$. This is done and, importantly, it has no effect on the KKT value corresponding to iterate $k+1$.

For the strategy described in Algorithm 6, we have the following lemma.

Lemma 3 *Suppose that problem* (1) *is feasible. If Algorithm 3 does not terminate before or in iteration k, then by employing Algorithm 6 in Step 6, iteration $k+1$ yields*

$$r(x_{k+1}, y_{k+1}, z_{k+1}^\ell, z_{k+1}^u) < \max_{j\in\{1,\ldots,p\}} \{r(x_{k+1-j}, y_{k+1-j}, z_{k+1-j}^\ell, z_{k+1-j}^u)\}. \quad (22)$$

Proof Under the assumption that Algorithm 3 has not yet terminated, we have

$$\max_{j\in\{1,\ldots,p\}} \{r(x_{k+1-j}, y_{k+1-j}, z_{k+1-j}^\ell, z_{k+1-j}^u)\} > 0. \quad (23)$$

Algorithm 6 Updating strategy based on ensuring KKT residual decrease

1: Input $(\mathcal{A}^\ell_k, \mathcal{A}^u_k, \mathcal{I}_k, \mathcal{U}_k)$, $p \geq 1$, and $\{(x_{k+1-j}, y_{k+1-j}, z^\ell_{k+1-j}, z^u_{k+1-j})\}_{j\in\{1,\ldots,p\}}$.
2: Set $(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$ by (16).
3: Set $(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1}) \leftarrow \text{Feas}(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.
4: Set $(x_{k+1}, y_{k+1}, z^\ell_{k+1}, z^u_{k+1}) \leftarrow \text{SM}(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.
5: **if** $r(x_{k+1}, y_{k+1}, z^\ell_{k+1}, z^u_{k+1}) \geq \max_{j\in\{1,\ldots,p\}}\{r(x_{k+1-j}, y_{k+1-j}, z^\ell_{k+1-j}, z^u_{k+1-j})\}$ **then**
6: Reset $(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1}) \leftarrow (\mathcal{A}^\ell_k, \mathcal{A}^u_k, \mathcal{I}_k, \mathcal{U}_k)$.
7: **repeat**
8: Choose $\mathcal{S} \leftarrow \mathcal{S}^\ell \cup \mathcal{S}^u \cup \mathcal{S}^\mathcal{I} \neq \emptyset$ with

$$\begin{aligned} \mathcal{S}^\ell &\subseteq \{i \in \mathcal{A}^\ell_{k+1} : z^\ell_k < 0\} \\ \mathcal{S}^u &\subseteq \{i \in \mathcal{A}^u_{k+1} : z^u_k < 0\} \\ \text{and } \mathcal{S}^\mathcal{I} &\subseteq \{i \in \mathcal{I}_{k+1} : \min\{[x_k - \ell]_i, [u - x_k]_i\} < 0\}. \end{aligned}$$

9: Set $\mathcal{A}^\ell_{k+1} \leftarrow \mathcal{A}^\ell_{k+1}\backslash\mathcal{S}^\ell$, $\mathcal{A}^u_{k+1} \leftarrow \mathcal{A}^u_{k+1}\backslash\mathcal{S}^u$, $\mathcal{I}_{k+1} \leftarrow \mathcal{I}_{k+1}\backslash\mathcal{S}^\mathcal{I}$, and $\mathcal{U}_{k+1} \leftarrow \mathcal{U}_{k+1}\cup\mathcal{S}$.
10: Set $(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1}) \leftarrow \text{Feas}(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.
11: Set $(x_{k+1}, y_{k+1}, z^\ell_{k+1}, z^u_{k+1}) \leftarrow \text{SM}(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.
12: **until** $r(x_{k+1}, y_{k+1}, z^\ell_{k+1}, z^u_{k+1}) < \max_{j\in\{1,\ldots,p\}}\{r(x_{k+1-j}, y_{k+1-j}, z^\ell_{k+1-j}, z^u_{k+1-j})\}$
13: **end if**
14: (Optional) Choose $\mathcal{T} \leftarrow \mathcal{T}^\ell \cup \mathcal{T}^u \cup \mathcal{T}^\mathcal{I}$ with

$$\begin{aligned} \mathcal{T}^\ell &\subseteq \left\{ i \in \bigcap_{l=k+2-p}^{k+1} \mathcal{U}_l : [x_l]_i = \ell_i \text{ for } l \in \{k+2-p, \ldots, k+1\} \right\}, \\ \mathcal{T}^u &\subseteq \left\{ i \in \bigcap_{l=k+2-p}^{k+1} \mathcal{U}_l : [x_l]_i = u_i \text{ for } l \in \{k+2-p, \ldots, k+1\} \right\}, \\ \text{and } \mathcal{T}^\mathcal{I} &\subseteq \left\{ i \in \bigcap_{l=k+2-p}^{k+1} \mathcal{U}_l : \ell_i < [x_l]_i < u_i \text{ for } l \in \{k+2-p, \ldots, k+1\} \right\}, \end{aligned}$$

then set $\mathcal{A}^\ell_{k+1} \leftarrow \mathcal{A}^\ell_{k+1}\cup\mathcal{T}^\ell$, $\mathcal{A}^u_{k+1} \leftarrow \mathcal{A}^u_{k+1}\cup\mathcal{T}^u$, $\mathcal{I}_{k+1} \leftarrow \mathcal{I}_{k+1}\cup\mathcal{T}^\mathcal{I}$, and $\mathcal{U}_{k+1} \leftarrow \mathcal{U}_{k+1}\backslash\mathcal{T}$.
15: Return $(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.

If the condition in Step 5 holds (i.e., (22) does not hold), then the strategy reverts to the kth partition. In such cases, the strategy iteratively moves indices corresponding to nonzero elements in the vector defining $r(x_{k+1}, y_{k+1}, z^\ell_{k+1}, z^u_{k+1})$ (recall (8)) to the index set $\mathcal{U}_{k+1}$ until a strict reduction has been obtained (i.e., until (22) holds). This procedure will terminate finitely as, in the worst-case, the method eventually has $\mathcal{U}_{k+1} = \mathcal{N}$, in which case $r(x_{k+1}, y_{k+1}, z^\ell_{k+1}, z^u_{k+1}) = 0$. Finally, observe that the procedure for removing elements from $\mathcal{U}_{k+1}$ has no effect on $r(x_{k+1}, y_{k+1}, z^\ell_{k+1}, z^u_{k+1})$ since indices are only removed if their corresponding primal and dual variables do not contribute to any nonzero values in the vectors defining $r(x_{k+1}, y_{k+1}, z^\ell_{k+1}, z^u_{k+1})$ and the r values in (23). □

We now have the following theorem. The key idea to its proof is that, by ensuring monotonic decrease in an appropriately defined merit function (see (24)), the KKT error corresponding to the algorithm iterates will eventually vanish.

Theorem 4 *If problem* (1) *is feasible, then Algorithm 3 with Step 6 employing Algorithm 6 solves problem* (1) *in a finite number of iterations.*

Proof The result follows since by Lemma 3 we have that Algorithm 6 guarantees

$$\left\{ \max_{j \in \{1,\ldots,p+1\}} \{r(x_{k+1-j}, y_{k+1-j}, z^{\ell}_{k+1-j}, z^{u}_{k+1-j})\} \right\} \tag{24}$$

is monotonically strictly decreasing. (Note that in the elements of this sequence, the max is taken from $j \in \{1, \ldots, p+1\}$.) This is the case since, due to the strict inequality in (22), there can be at most p consecutive iterations where the right-hand side of (22) does not strictly decrease, and after $p+1$ iterations there must be a strict decrease. Since there are only a finite number of partitions, we eventually obtain a sufficiently large k such that $r(x^{\ell}_k, y^{u}_k, z^{\ell}_k, z^{u}_k) = 0$. □

Table 3 contains the output from solving the strictly convex QP in Example 1 on page 11 using Algorithm 3 with $p = 1$.

Table 3 Result of Algorithm 3 employed to solve the problem in Example 1 when iterates are updated via Algorithm 6. In the algorithm, the ℓ_∞-norm is used in the definition of the residual function r (recall (8)) and we define $r_k := r(x_k, z^u_k)$.

k	$\mathcal{A}^{\ell}_k$	$\mathcal{A}^{u}_k$	$\mathcal{I}_k$	$\mathcal{U}_k$	x_k	z^u_k	r_k
0	$\emptyset$	$\emptyset$	$\{1,2,3\}$	$\emptyset$	$(-3,1,-1)$	$(0,0,0)$	1
1	$\emptyset$	$\{2\}$	$\{1,3\}$	$\emptyset$	$(\frac{1}{3},0,\frac{2}{3})$	$(0,\frac{2}{3},0)$	$\frac{2}{3}$
2	$\emptyset$	$\{1,2,3\}$	$\emptyset$	$\emptyset$	$(0,0,0)$	$(-2,-1,3)$	2
3	$\emptyset$	$\{2,3\}$	$\emptyset$	$\{1\}$	$(-\frac{1}{2},0,0)$	$(0,\frac{3}{2},\frac{1}{2})$	0

4 Implementation details

In this paper, we have proposed and analyzed two instances of the generic framework provided as Algorithm 3, in addition to an instance representing an extension of the algorithm in [25]. In this section, we describe an implementation in Matlab that incorporates all of these approaches. The main motivation for our implementation (and the numerical results in the following section) is to illustrate that while Algorithm 3 paired with the updating strategy in Algorithm 4 is extremely efficient for solving many strictly-convex QPs, it can lead to cycling, especially when applied to solve large and/or ill-conditioned problems. Our updating strategies in Algorithms 5 and 6, on the other hand, are globally convergent and require only a modest amount of additional computational effort. Specifically, our implementation incorporates the index sets $\{\mathcal{U}_k\}$ to ensure global convergence, but attempts to keep these sets small so that the subspace minimization procedure (SM) is not much more expensive than solving a reduced linear system.

We specify the details of our implementation by considering, in turn, the subroutines in Algorithm 3. First, Algorithm 1 is responsible for detecting the potential infeasibility of a partition and, if necessary, modifying the partition to a feasible one. The infeasibility detection phase involves the solution of a linear optimization problem (LP) that minimizes violations of the constraints in (5):

$$\begin{array}{l} \underset{\overline{x}_{\mathcal{F}}, \overline{r}, \overline{s}}{\text{minimize}} \;\; e^T(\overline{r} + \overline{s}) \\ \text{subject to } A_{\mathcal{M},\mathcal{F}}\overline{x}_{\mathcal{F}} = b - A_{\mathcal{M},\mathcal{A}}x_{\mathcal{A}} + \overline{r} - \overline{s}, \;\; \ell_{\mathcal{U}} \leq \overline{x}_{\mathcal{U}} \leq u_{\mathcal{U}}, \;\; (\overline{r}, \overline{s}) \geq 0, \end{array} \tag{25}$$

where $e \in \mathbb{R}^m$ is vector of ones of appropriate length, $\mathcal{A}$ and $\mathcal{F}$ are defined as in Step 3 of Algorithm 2, and $x_{\mathcal{A}}$ is defined as it is set in Step 2 of Algorithm 2. For the first component of the starting point for solving this problem, call it $\overline{x}^0_{\mathcal{F}}$, we choose the projection of the most recent primal-dual solution estimate onto the feasible region of the bound constraints in (5); i.e.,

$$\overline{x}^0_{\mathcal{F}} \leftarrow (\overline{x}^0_{\mathcal{I}}, \overline{x}^0_{\mathcal{U}}), \text{ where } \overline{x}^0_{\mathcal{I}} \leftarrow [x_{k-1}]_{\mathcal{I}} \text{ and } \overline{x}^0_{\mathcal{U}} \leftarrow \max\{\ell_{\mathcal{U}}, \min\{[x_{k-1}]_{\mathcal{U}}, u_{\mathcal{U}}\}\}.$$

We then set the initial values for the slack variables to be

$$\begin{aligned} \overline{r}^0 &= \max\{0, A_{\mathcal{M},\mathcal{A}} x_{\mathcal{A}} + A_{\mathcal{M},\mathcal{F}} \overline{x}^0_{\mathcal{F}} - b\} \\ \text{and } \overline{s}^0 &= \max\{0, b - A_{\mathcal{M},\mathcal{A}} x_{\mathcal{A}} - A_{\mathcal{M},\mathcal{F}} \overline{x}^0_{\mathcal{F}}\}. \end{aligned}$$

CPLEX's primal simplex method is applied to solve (25). In many cases, (25) is solved at this initial point. In general, however, (25) is solved via an LP solution method and, when (3) is satisfiable, the resulting solution $(\overline{x}^*_{\mathcal{F}}, \overline{r}^*, \overline{s}^*)$ yields $e^T(\overline{r}^* + \overline{s}^*) = 0$. (In fact, due to numerical inaccuracies, we consider a given partition to be feasible as long as $e^T(r^* + s^*) \leq \varepsilon$ for a small constant $\varepsilon > 0$.) If we find that this condition does not hold, then this implies that the active set $\mathcal{A}^\ell \cup \mathcal{A}^u$ should have elements removed. Algorithm 1 is motivated by the fact that infeasibility of (3) implies that too many variables are fixed to their bounds. Based on this observation, our implementation transforms the partition into a feasible one by iteratively moving an index from $\mathcal{A}^\ell \cup \mathcal{A}^u$ to $\mathcal{I}$. The index to be moved, call it j, is selected as one that, if included in $\mathcal{I}$, would potentially lead to the largest reduction in infeasibility; i.e., with a_i defined as the ith column of A, we choose

$$j \leftarrow \underset{i \in \mathcal{A}^\ell \cup \mathcal{A}^u}{\operatorname{argmin}} \left(\min_{\Delta x_i} \tfrac{1}{2} \| A_{\mathcal{M},\mathcal{A}} x_{\mathcal{A}} + a_i \Delta x_i + A_{\mathcal{M},\mathcal{F}} \overline{x}^*_{\mathcal{F}} - b \|_2^2 \right),$$

which can be computed via $|\mathcal{A}^\ell \cup \mathcal{A}^u|$ minimizations of one-dimensional convex quadratics. (If multiple indexes yield the same objective value for the inner minimization problem, then our implementation selects the smallest such index.)

Our implementation of Algorithm 2 is relatively straightforward. Indeed, the only step that requires specification is the method employed to solve subproblem (5). If $|\mathcal{U}| = 0$, then the solution of (5) is obtained by solving the reduced linear system (7); in such cases, we employ Matlab's built-in `"\"` routine to solve this system. If $|\mathcal{U}| \neq 0$, then problem (5) is a generally-constrained QP and we employ the active-set method implemented in the qpOASES package [12].

We now turn to the details of our implementation of our strategies in Algorithms 5 and 6. Due to the increased computational costs of the SM subroutine when $\mathcal{U}_k$ is large, we have implemented these strategies so that $|\mathcal{U}_{k+1}| \leq |\mathcal{U}_k| + 1$ for all k. In Algorithm 5, we implement Step 3 by choosing $\mathcal{S}$ as the smallest index in $\{i \in \mathcal{N} : q^i_k = \max_{j \in \mathcal{N}} q^j_k\}$, and we implement Step 5 by setting $q^i_{k+1} \leftarrow 0$ for all $i \in \mathcal{N}$. Similarly, in Algorithm 6, we implement Step 8 by choosing $\mathcal{S}$ as the index in $\mathcal{A}^\ell_{k+1} \cup \mathcal{A}^u_{k+1} \cup \mathcal{I}_{k+1}$ corresponding to the element of the vector defining $r(x_{k+1}, y_{k+1}, z^\ell_{k+1}, z^u_{k+1})$ (recall (8)) with the largest absolute value. (If there are more than one such indices in $\mathcal{A}^\ell_{k+1} \cup \mathcal{A}^u_{k+1} \cup \mathcal{I}_{k+1}$, then we choose the smallest such index.) Finally, due to numerical inaccuracies in the SM routine, the idealized conditions in Step 12 of Algorithm 6 are inappropriate in practice for removing

elements from the set $\mathcal{U}_{k+1}$. (In particular, since variables may never be set exactly at their bounds, those conditions would typically consider all variables to be inactive in all solutions, which would be inappropriate.) Alternatively, we perform this step by defining $0 < \varepsilon_{\mathcal{A}} < \varepsilon_{\mathcal{I}}$ and setting

$$\mathcal{T}^{\ell} \leftarrow \left\{ i \in \bigcap_{l=k+2-p}^{k+1} \mathcal{U}_l : [x_l]_i \leq \ell_i + \varepsilon_{\mathcal{A}} \text{ for } l \in \{k+2-p, \ldots, k+1\} \right\},$$

$$\mathcal{T}^{u} \leftarrow \left\{ i \in \bigcap_{i=k+2-p}^{k+1} \mathcal{U}_l : [x_l]_i \geq u_i - \varepsilon_{\mathcal{A}} \text{ for } l \in \{k+2-p, \ldots, k+1\} \right\}, \text{ and}$$

$$\mathcal{T}^{\mathcal{I}} \leftarrow \left\{ i \in \bigcap_{i=k+2-p}^{k+1} \mathcal{U}_l : \ell_i + \varepsilon_{\mathcal{I}} \leq [x_l]_i \leq u_i - \varepsilon_{\mathcal{I}} \text{ for } l \in \{k+2-p, \ldots, k+1\} \right\}.$$

That is, we choose a relatively tight (but still nonzero) tolerance for determining an index to be active and a relatively large tolerance for determining an index to be inactive. Primal variables with values between $\varepsilon_{\mathcal{A}}$ and $\varepsilon_{\mathcal{I}}$ are considered too ambiguous to be determined as active or inactive.

The numerical results in the following section support our claim that both updating strategies effectively prevent $|\mathcal{U}_k|$ from becoming large.

5 Numerical Results

We tested our implementation of Algorithm 3 by solving randomly generated problems with various numbers of variables (n), constraints (m), and condition numbers of H (H_{cond}). We generated H via Matlab's `sprandsym` routine and generated A via Matlab's `randn` routine. The problems were generated so that there would (roughly) be an equal number of lower-active, upper-active, and inactive primal variables in the optimal solution. The algorithms were tested in Matlab 7.12.0.635 (R2011a) on a 64-bit machine with 16 processors running a Linux environment.

For all of our experiments, the components $(\mathcal{A}_0^{\ell}, \mathcal{A}_0^{u}, \mathcal{I}_0)$ of the initial partition were randomly generated while $\mathcal{U}_0$ was set to $\emptyset$. Algorithm 5 was run with $q^{\max} \leftarrow 5$ and Algorithm 6 (with and without step 14) was run with $p \leftarrow 5$. These values were chosen as they resulted in good performance in our experiments. A problem was declared to be solved successfully by an algorithm if for some k it obtained $r(x_k, y_k, z_k^{\ell}, z_k^{u}) \leq 10^{-6}$, where the ℓ_∞-norm was used in the definition of r. However, we set an iteration limit for each problem as $1.1n$; i.e., if for a given problem an algorithm failed to satisfy our tolerance for the KKT error in $1.1n$ iterations, then we say that the algorithm failed to solve that problem. The tolerance parameter $\varepsilon > 0$ (see the discussion following problem (25)) was set to 10^{-8} and for Algorithm 6 we set $\varepsilon_{\mathcal{A}} \leftarrow 10^{-8}$ and $\varepsilon_{\mathcal{I}} \leftarrow 10^{-2}$.

Hereinafter, we refer to Algorithm 3 paired with the updating strategy in Algorithm 4 simply as "Algorithm 4", and similarly for Algorithms 5 and 6. We first compare the algorithms when solving strictly convex bound-constrained QPs (BQPs). (Recall that for bound-constrained problems, the Feas routine, i.e., Algorithm 1, is never invoked.) We tested problems with all combinations of number of variables (n) in $\{10^2, 10^3, 10^4\}$ and condition numbers for H (H_{cond}) in

$\{10^2, 10^4, 10^6\}$. We generated 50 problems for each of these 9 combinations and report, in Tables 4–8, averages (and some standard deviations) of performance measures over these 50 runs. For each combination, all 50 problems were solved unless otherwise indicated, and in cases when fewer than the 50 problems were solved, the statistics are computed only over those runs that were successful. In fact, this occurred only for the results in Table 4; see the caption for that table.

In Tables 4–7, we present results for Algorithms 4, 5, and 6. The first three statistics that we report are the average number of iterations (μ`(#Iter)`), the standard deviation of the number of iterations (σ`(#Iter)`), and the average number of calls to Algorithm 2 (μ`(#SM)`). In Tables 4 and 5, the number of calls to Algorithm 2 is equal to the number of iterations (plus 1 as the SM routine is called in the last iteration before computing the final KKT error), but in Tables 6 and 7 the number of calls to Algorithm 2 may be relatively larger due to potential additional calls to the SM routine while updating the index set partition.

The next statistics that we report represent a breakdown between two types of calls to Algorithm 2. In particular, if $\mathcal{U}_k = \emptyset$, then the major computational component of the call is a linear system solve, which (as has already been mentioned) is performed via Matlab's built-in `"\"` routine; otherwise, if $\mathcal{U}_k \neq \emptyset$, we solve the corresponding bound-constrained subproblem with the qpOASES package. In the former type of iteration we increment a "linear system" counter (by 1), whereas in the latter type of iteration we increment a "quadratic subproblem iteration" counter (by the number of iterations reported by qpOASES). In the tables, we report the average total number of linear systems solved (μ`(#LS)`), the standard deviation of the number of linear systems solved (σ`(#LS)`), the average total number of iterations of qpOASES summed over all calls to it (μ`(#QP-Iter)`), and the standard deviation of the total number of qpOASES iterations (σ`(#QP-Iter)`). When observing these results, it is important to note that when the initial point for solving a QP is optimal, qpOASES reports zero iterations were performed.

The last statistics that we report relate to the cardinality of the uncertain sets in the experiments. For a given run of an algorithm to solve a given problem instance, let K represent the final iteration number. In the tables, we report the average cardinality of $\mathcal{U}_K$ (μ`(Last-`$|\mathcal{U}|$`)`), the average of the mean cardinality of the elements of $\{\mathcal{U}_k\}_{k=0}^K$ (μ`(Avg-`$|\mathcal{U}|$`)`), and the standard deviation of the mean cardinality of the elements of $\{\mathcal{U}_k\}_{k=0}^K$ (σ`(Avg-`$|\mathcal{U}|$`)`).

Since Algorithm 4 maintains $\mathcal{U}_k = \emptyset$ for all k, we omit columns (that would otherwise appear in Table 4) corresponding to qpOASES iterations and cardinalities of the uncertain set since these values are all zero.

Table 4 shows that Algorithm 4 is generally very efficient, but may not converge. On the other hand, Algorithms 5 and 6 (see Tables 5–7) solved all problems in our experiments, illustrating that their global convergence guarantees are beneficial in practice. Note that in Tables 5–6, a "μ`(Last-`$|\mathcal{U}|$`)`" equal to zero indicates that the algorithm is behaving identically to Algorithm 4. As this occurs often, particularly in Table 6, this illustrates that Algorithms 5 and 6 are as efficient as Algorithm 4 for many instances in our experiments. Moreover, even when this performance measure is nonzero, it is typically very small (especially when considered relative to n) illustrating that our global convergence guarantees are attained at modest additional effort. This is further confirmed by the observation that the total qpOASES iterations (μ`(#QP-Iter)`) is often very small (especially relative to n). We also remark that the optional strategy in Algorithm 6 yields some benefits

Table 4 Results of Algorithm 4 employed to solve BQPs. Statistics followed by † were computed only over 45 (of 50) successful runs. Similarly, statistics followed by ‡ were computed only over 46 (of 50) successful runs. All other statistics were computed over 50 successful runs.

n	H_{cond}	μ(#Iter)	σ(#Iter)	μ(#SM)	μ(#LS)	σ(#LS)
1e+02	1e+02	3.78e+00	6.79e-01	4.78e+00	4.78e+00	6.79e-01
1e+02	1e+04	5.14e+00	9.69e-01	6.14e+00	6.14e+00	9.69e-01
1e+02	1e+06	6.22e+00	1.06e+00	7.22e+00	7.22e+00	1.06e+00
1e+03	1e+02	4.78e+00	6.16e-01	5.78e+00	5.78e+00	6.16e-01
1e+03	1e+04	6.64e+00	1.05e+00	7.64e+00	7.64e+00	1.05e+00
1e+03	1e+06	8.02e+00	1.06e+00	9.02e+00	9.02e+00	1.06e+00
1e+04	1e+02	5.80e+00	4.95e-01	6.80e+00	6.80e+00	4.95e-01
1e+04	1e+04	8.24e+00†	7.43e-01†	9.24e+00†	9.24e+00†	7.43e-01†
1e+04	1e+06	1.01e+01‡	1.14e+00‡	1.11e+01‡	1.11e+01‡	1.14e+00‡

Table 5 Results of Algorithm 5 employed to solve BQPs.

n	H_{cond}	μ(#Iter)	σ(#Iter)	μ(#SM)	μ(#LS)	σ(#LS)
1e+02	1e+02	3.78e+00	6.79e-01	4.78e+00	4.78e+00	6.79e-01
1e+02	1e+04	5.14e+00	9.69e-01	6.14e+00	6.04e+00	8.56e-01
1e+02	1e+06	6.20e+00	1.05e+00	7.20e+00	6.74e+00	8.03e-01
1e+03	1e+02	4.78e+00	6.16e-01	5.78e+00	5.78e+00	6.16e-01
1e+03	1e+04	6.60e+00	1.05e+00	7.60e+00	6.82e+00	8.96e-01
1e+03	1e+06	8.00e+00	1.07e+00	9.00e+00	6.60e+00	9.90e-01
1e+04	1e+02	5.80e+00	4.95e-01	6.80e+00	6.62e+00	5.30e-01
1e+04	1e+04	9.28e+00	3.89e+00	1.03e+01	6.02e+00	1.41e-01
1e+04	1e+06	1.07e+01	2.73e+00	1.17e+01	6.00e+00	0.00e+00

n	H_{cond}	μ(#QP-Iter)	σ(#QP-Iter)	μ(Last-$\lvert\mathcal{U}\rvert$)	μ(Avg-$\lvert\mathcal{U}\rvert$)	σ(Avg-$\lvert\mathcal{U}\rvert$)
1e+02	1e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+02	1e+04	1.20e-01	8.49e-01	6.00e-02	1.42e-02	6.17e-02
1e+02	1e+06	3.40e-01	9.17e-01	2.40e-01	6.12e-02	1.19e-01
1e+03	1e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+03	1e+04	2.60e-01	6.64e-01	4.20e-01	1.05e-01	1.43e-01
1e+03	1e+06	1.44e+00	2.44e+00	7.60e-01	2.83e-01	1.79e-01
1e+04	1e+02	1.20e-01	5.94e-01	1.60e-01	2.91e-02	6.92e-02
1e+04	1e+04	3.84e+00	8.35e+00	1.14e+00	4.55e-01	2.64e-01
1e+04	1e+06	6.06e+00	7.57e+00	1.14e+00	5.42e-01	1.73e-01

in our experiments; i.e., by removing elements from the uncertain set, the algorithm typically requires fewer QP iterations and maintains smaller uncertain sets. This can be seen by comparing the results in Tables 6 and 7.

As a means of comparison for the results in Tables 4–7, we present in Table 8 results when the same set of test problems are solved directly with qpOASES. We report the average number of iterations reported by qpOASES (μ(#QP-Iter)) and the standard deviation of the number of iterations reported (σ(#QP-Iter)). It is important to note that these results should not be compared directly with the similarly named columns in Tables 4–7 since in many iterations our implementations of Algorithms 4, 5, and 6 solve a linear system directly rather than calling qpOASES. That being said, the results in Table 8 illustrate the typical behavior of a classic active-set method with which the number of iterations often increases with problem size (n) and condition number of H (H_{cond}). By contrast, such a dependence appears less significant in the results in Tables 4–7.

Table 6 Results of Algorithm 6 (without step 14) employed to solve BQPs.

n	H_{cond}	μ(#Iter)	σ(#Iter)	μ(#SM)	μ(#LS)	σ(#LS)
1e+02	1e+02	3.78e+00	6.79e-01	4.78e+00	4.78e+00	6.79e-01
1e+02	1e+04	5.14e+00	9.69e-01	6.14e+00	6.14e+00	9.69e-01
1e+02	1e+06	6.22e+00	1.06e+00	7.22e+00	7.22e+00	1.06e+00
1e+03	1e+02	4.78e+00	6.16e-01	5.78e+00	5.78e+00	6.16e-01
1e+03	1e+04	6.64e+00	1.05e+00	7.64e+00	7.64e+00	1.05e+00
1e+03	1e+06	8.02e+00	1.06e+00	9.02e+00	9.02e+00	1.06e+00
1e+04	1e+02	5.80e+00	4.95e-01	6.80e+00	6.80e+00	4.95e-01
1e+04	1e+04	8.86e+00	2.27e+00	1.00e+01	9.66e+00	1.47e+00
1e+04	1e+06	1.04e+01	1.47e+00	1.15e+01	1.13e+01	1.26e+00
n	H_{cond}	μ(#QP-Iter)	σ(#QP-Iter)	μ(Last-$\lvert\mathcal{U}\rvert$)	μ(Avg-$\lvert\mathcal{U}\rvert$)	σ(Avg-$\lvert\mathcal{U}\rvert$)
1e+02	1e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+02	1e+04	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+02	1e+06	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+03	1e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+03	1e+04	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+03	1e+06	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+04	1e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+04	1e+04	9.80e-01	6.79e+00	1.80e-01	3.98e-02	2.16e-01
1e+04	1e+06	1.80e-01	1.02e+00	1.00e-01	1.59e-02	5.89e-02

Table 7 Results of Algorithm 6 (with step 14) employed to solve BQPs.

n	H_{cond}	μ(#Iter)	σ(#Iter)	μ(#SM)	μ(#LS)	σ(#LS)
1e+02	1e+02	3.78e+00	6.79e-01	4.78e+00	4.78e+00	6.79e-01
1e+02	1e+04	5.14e+00	9.69e-01	6.14e+00	6.14e+00	9.69e-01
1e+02	1e+06	6.22e+00	1.06e+00	7.22e+00	7.22e+00	1.06e+00
1e+03	1e+02	4.78e+00	6.16e-01	5.78e+00	5.78e+00	6.16e-01
1e+03	1e+04	6.64e+00	1.05e+00	7.64e+00	7.64e+00	1.05e+00
1e+03	1e+06	8.02e+00	1.06e+00	9.02e+00	9.02e+00	1.06e+00
1e+04	1e+02	5.80e+00	4.95e-01	6.80e+00	6.80e+00	4.95e-01
1e+04	1e+04	8.86e+00	2.27e+00	1.00e+01	9.86e+00	2.27e+00
1e+04	1e+06	1.04e+01	1.54e+00	1.15e+01	1.14e+01	1.54e+00
n	H_{cond}	μ(#QP-Iter)	σ(#QP-Iter)	μ(Last-$\lvert\mathcal{U}\rvert$)	μ(Avg-$\lvert\mathcal{U}\rvert$)	σ(Avg-$\lvert\mathcal{U}\rvert$)
1e+02	1e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+02	1e+04	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+02	1e+06	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+03	1e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+03	1e+04	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+03	1e+06	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+04	1e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
1e+04	1e+04	1.20e-01	7.18e-01	0.00e+00	1.11e-02	3.92e-02
1e+04	1e+06	6.00e-02	3.14e-01	0.00e+00	7.13e-03	2.52e-02

For our second set of experiments, we randomly generated problems with $n = 10^4$, but with all combinations of numbers of equality constraints (m) in $\{10, 20\}$ and condition numbers for H (H_{cond}) in $\{10^2, 10^4, 10^6\}$. For each combination we generated 10 problems and report the averages (and some standard deviations) of various performance measures. All measures that we considered were the same as for the bound-constrained problems, except that we now also report the average number of calls to Algorithm 1 (μ(#Feas)), the average number of times that a call to Algorithm 1 actually modified the index set partition (μ(#Feas-Mod)),

Table 8 Results of qpOASES employed to solve BQPs.

n	H_{cond}	μ(#QP-Iter)	σ(#QP-Iter)
1e+02	1e+02	5.40e+01	7.37e+00
1e+02	1e+04	5.90e+01	6.16e+00
1e+02	1e+06	6.22e+01	6.73e+00
1e+03	1e+02	5.39e+02	2.11e+01
1e+03	1e+04	5.70e+02	2.04e+01
1e+03	1e+06	5.94e+02	1.97e+01
1e+04	1e+02	5.33e+03	5.98e+01
1e+04	1e+04	5.70e+03	6.40e+01
1e+04	1e+06	5.97e+03	8.31e+01

the average total number of simplex pivots (in CPLEX) summed over all calls to Algorithm 1 (μ(#Feas-Pvt)), and the standard deviation of the total number of simplex pivots (σ(#Feas-Pvt)). These results are provided in Tables 9–11.

Table 9 Results of Algorithm 5 employed to solve QPs (with $n = 10^4$).

m	H_{cond}	μ(#Iter)	σ(#Iter)	μ(#SM)	μ(#LS)	σ(#LS)
1e+01	1e+02	1.20e+01	3.23e+00	1.30e+01	6.00e+00	0.00e+00
1e+01	1e+04	1.47e+01	9.38e+00	1.57e+01	6.00e+00	0.00e+00
1e+01	1e+06	2.25e+01	1.85e+01	2.35e+01	6.00e+00	0.00e+00
2e+01	1e+02	1.39e+01	5.90e+00	1.49e+01	6.00e+00	0.00e+00
2e+01	1e+04	1.77e+01	1.05e+01	1.87e+01	6.00e+00	0.00e+00
2e+01	1e+06	1.65e+01	6.85e+00	1.75e+01	6.00e+00	0.00e+00

m	H_{cond}	μ(#QP-Iter)	σ(#QP-Iter)	μ(Last-$\lvert\mathcal{U}\rvert$)	μ(Avg-$\lvert\mathcal{U}\rvert$)	σ(Avg-$\lvert\mathcal{U}\rvert$)
1e+01	1e+02	6.67e+01	2.86e+01	1.30e+00	6.31e-01	2.53e-01
1e+01	1e+04	1.01e+02	7.15e+01	1.60e+00	8.50e-01	7.93e-01
1e+01	1e+06	2.31e+02	3.15e+02	2.80e+00	1.40e+00	1.47e+00
2e+01	1e+02	1.72e+02	1.07e+02	1.60e+00	7.50e-01	3.93e-01
2e+01	1e+04	2.77e+02	2.50e+02	2.00e+00	9.28e-01	5.73e-01
2e+01	1e+06	2.37e+02	1.59e+02	2.10e+00	9.70e-01	5.76e-01

m	H_{cond}	μ(#Feas)	μ(#Feas-Mod)	μ(#Feas-Pvt)	σ(#Feas-Pvt)
1e+01	1e+02	1.30e+01	2.70e+00	1.30e+00	2.11e+00
1e+01	1e+04	1.57e+01	3.70e+00	1.90e+00	5.00e+00
1e+01	1e+06	2.35e+01	8.40e+00	1.38e+01	3.21e+01
2e+01	1e+02	1.49e+01	4.00e+00	5.90e+00	7.78e+00
2e+01	1e+04	1.87e+01	3.20e+00	1.10e+00	1.10e+00
2e+01	1e+06	1.75e+01	6.10e+00	7.10e+00	1.16e+01

Tables 9–11 illustrate that Algorithms 5 and 6 (with or without the optional step 14) successfully and efficiently solved all generated problem instances. Moreover, in all cases, the set $\mathcal{U}_k$ was maintained at a very small size relative to n. All of this being said, these results illustrate that the performance of our algorithms is less impressive when equality constraints are present. While the size of the uncertain set is typically very small relative to n, it is less often equal to zero (when compared to our results for BQPs). For example, in Table 9, the consistent values 6.00e+00 and 0.00e+00 for μ(#LS) and σ(#LS), respectively, indicate that Algorithm 5 consistently only had an empty uncertain set in the first few iterations, and afterwards the set had at least one element. This means that the solver more often relies on qpOASES, causing an increase in the total number of subproblem itera-

Table 10 Results of Algorithm 6 (without step 14) employed to solve QPs (with $n = 10^4$).

m	H_{cond}	μ(#Iter)	σ(#Iter)	μ(#SM)	μ(#LS)	σ(#LS)
1e+01	1e+02	1.88e+01	8.12e+00	2.74e+01	1.07e+01	2.00e+00
1e+01	1e+04	2.19e+01	7.87e+00	2.82e+01	1.36e+01	2.50e+00
1e+01	1e+06	2.27e+01	7.06e+00	2.79e+01	1.37e+01	2.45e+00
2e+01	1e+02	1.89e+01	5.22e+00	2.40e+01	1.06e+01	1.58e+00
2e+01	1e+04	2.08e+01	9.02e+00	2.63e+01	1.17e+01	3.53e+00
2e+01	1e+06	3.51e+01	2.92e+01	5.47e+01	1.42e+01	3.99e+00
m	H_{cond}	μ(#QP-Iter)	σ(#QP-Iter)	μ(Last-$\lvert\mathcal{U}\rvert$)	μ(Avg-$\lvert\mathcal{U}\rvert$)	σ(Avg-$\lvert\mathcal{U}\rvert$)
1e+01	1e+02	2.61e+02	4.45e+02	7.60e+00	3.05e+00	4.93e+00
1e+01	1e+04	1.72e+02	2.06e+02	5.30e+00	1.70e+00	2.54e+00
1e+01	1e+06	2.03e+02	3.55e+02	4.20e+00	1.65e+00	3.71e+00
2e+01	1e+02	2.80e+02	2.34e+02	4.10e+00	1.58e+00	2.14e+00
2e+01	1e+04	3.59e+02	5.14e+02	4.50e+00	1.64e+00	3.37e+00
2e+01	1e+06	1.68e+03	3.29e+03	1.86e+01	8.30e+00	1.65e+01
m	H_{cond}	μ(#Feas)	μ(#Feas-Mod)	μ(#Feas-Pvt)	σ(#Feas-Pvt)	
1e+01	1e+02	2.74e+01	3.70e+00	9.60e+00	1.39e+01	
1e+01	1e+04	2.82e+01	4.00e+00	1.00e+01	1.79e+01	
1e+01	1e+06	2.79e+01	2.90e+00	8.00e+00	1.93e+01	
2e+01	1e+02	2.40e+01	5.80e+00	2.07e+01	3.51e+01	
2e+01	1e+04	2.63e+01	3.20e+00	8.10e+00	2.07e+01	
2e+01	1e+06	5.47e+01	5.30e+00	5.74e+01	1.03e+02	

Table 11 Results for Algorithm 6 (with step 14) employed to solve QPs (with $n = 10^4$).

m	H_{cond}	μ(#Iter)	σ(#Iter)	μ(#SM)	μ(#LS)	σ(#LS)
1e+01	1e+02	3.29e+01	2.14e+01	5.42e+01	3.38e+01	2.12e+01
1e+01	1e+04	3.14e+01	1.64e+01	4.90e+01	3.20e+01	1.56e+01
1e+01	1e+06	3.69e+01	1.96e+01	6.04e+01	3.73e+01	1.84e+01
2e+01	1e+02	3.92e+01	2.74e+01	6.95e+01	3.98e+01	2.72e+01
2e+01	1e+04	4.39e+01	3.02e+01	7.39e+01	4.43e+01	2.90e+01
2e+01	1e+06	1.06e+02	1.13e+02	2.02e+02	1.05e+02	1.07e+02
m	H_{cond}	μ(#QP-Iter)	σ(#QP-Iter)	μ(Last-$\lvert\mathcal{U}\rvert$)	μ(Avg-$\lvert\mathcal{U}\rvert$)	σ(Avg-$\lvert\mathcal{U}\rvert$)
1e+01	1e+02	2.12e+02	2.85e+02	1.00e-01	4.71e-01	2.96e-01
1e+01	1e+04	1.32e+02	1.91e+02	4.00e-01	4.39e-01	2.61e-01
1e+01	1e+06	2.71e+02	3.27e+02	1.00e-01	4.93e-01	3.00e-01
2e+01	1e+02	6.03e+02	6.23e+02	3.00e-01	5.93e-01	2.95e-01
2e+01	1e+04	6.38e+02	8.15e+02	2.00e-01	5.12e-01	2.88e-01
2e+01	1e+06	1.92e+03	2.46e+03	5.00e-01	6.37e-01	3.76e-01
m	H_{cond}	μ(#Feas)	μ(#Feas-Mod)	μ(#Feas-Pvt)	σ(#Feas-Pvt)	
1e+01	1e+02	5.42e+01	3.70e+00	3.90e+00	5.11e+00	
1e+01	1e+04	4.90e+01	3.40e+00	3.40e+00	6.26e+00	
1e+01	1e+06	6.04e+01	5.30e+00	6.40e+00	1.25e+01	
2e+01	1e+02	6.95e+01	1.09e+01	3.26e+01	6.19e+01	
2e+01	1e+04	7.39e+01	4.60e+00	8.00e+00	2.01e+01	
2e+01	1e+06	2.02e+02	9.40e+00	5.57e+01	1.25e+02	

tions. The algorithms also involve additional work to maintain feasible partitions; work that may become significant if even more equality constraints are present. It is for these reasons that we do not present results for problems with higher numbers of equality constraints. Still, for the experiments we have performed, the results of our algorithms are strong when compared to the results obtained when applying qpOASES directly to solve the problems; see Table 12.

Table 12 Results of qpOASES employed to solve QPs (with $n = 10^4$).

m	H_{cond}	μ(#QP-Iter)	σ(#QP-Iter)
1e+01	1e+02	5.41e+03	5.65e+01
1e+01	1e+04	5.74e+03	9.20e+01
1e+01	1e+06	6.06e+03	1.14e+02
2e+01	1e+02	5.36e+03	7.45e+01
2e+01	1e+04	5.76e+03	5.99e+01
2e+01	1e+06	6.05e+03	4.82e+01

6 Conclusion

Motivated by the impressive practical performance of the primal-dual active-set method proposed by Hintermüller, Ito, and Kunisch [25] when solving certain bound-constrained QPs arising from discretized PDE-constrained optimization problems, we have proposed an algorithmic framework for solving strictly convex QPs that possesses appealing properties. In particular, we have shown that our framework is globally convergent when solving any strictly convex generally-constrained QP, and have shown in our numerical experiments that two instances of our framework achieve this theoretical behavior with only a modest increase in per-iteration computational cost as compared to the method in [25].

The novel idea underlying our framework is to introduce a set auxiliary to the traditional active-set estimate. Our techniques for handling this auxiliary set, which houses the indices of variables whose bounds will be enforced explicitly during a given iteration, have been motivated based on two observations. First, we have seen in our numerical experiments and those of others that the active-set method in [25] often converges extremely quickly when solving a convex BQP, despite the limitations of the method's theoretical convergence guarantees. Second, when the method in [25] does not converge, this behavior typically can be attributed to a small subset of variables that tend to migrate between active and inactive set estimates. Hence, by introducing our auxiliary set and devising strategies that only move indices to that set to avoid cycling/nonconvergence, we are able to attain the rapid convergence behavior of the method in [25] while solidifying a global convergence guarantee for a more general class of problems.

The biggest potential drawback of introducing our auxiliary set is that the added computational cost may become severe if the size of the set $\mathcal{U}_k$ becomes large. In such cases, rather than simply requiring the solution of a reduced linear system in each iteration as is required in [25] (and in our framework when $\mathcal{U}_k = \emptyset$), our framework requires the solution of a reduced QP with a subset of the original bound constraints. However, the numerical results that we have provided in §5 show that the set $\mathcal{U}_k$ rarely grows beyond a few indices. In fact, we often find that $\mathcal{U}_k$ remains empty throughout most iterations of a run of the algorithm, in which case our framework behaves as the method in [25].

Our framework was less efficient when solving QPs with a large number of equality constraints relative to the number of variables. This was not a surprise to us as the rapidly-adapting active-set estimates may led to infeasible partitions, which may in general lead to unacceptable increases in computational costs. Therefore, we recommend our framework when solving QPs with many degrees of

freedom, and otherwise suggest the use of classical active-set strategies, which are better tailored for problems with few degrees of freedom.

Finally, we remark that our framework lends itself to possible further enhancements, such as the use of iterative methods in place of direct matrix factorizations when solving our reduced subproblems. Maintaining global convergence guarantees when such techniques are used is not a trivial task as inexactness in the subproblem solves has to be monitored carefully so that progress is still made when updating the active-set estimates, but such details are a subject of current research.

References

1. M. Aganagić. Newton's method for linear complementarity problems. *Mathematical Programming*, 28(3):349–362, 1984.
2. M. Bergounioux, K. Ito, and K. Kunisch. Primal-dual strategy for constrained optimal control problems. *SIAM Journal on Control and Optimization*, 37(4):1176–1194, 1999.
3. M. Bergounioux and K. Kunisch. Primal-dual strategy for state-constrained optimal control problems. *Computational Optimization and Applications*, 22(2):193–224, 2002.
4. E. G. Birgin, C. A. Floudas, and J. M. Martínez. Global minimization using an Augmented Lagrangian method with variable lower-level constraints. *Mathematical Programming*, 125(1):139–162, 2010.
5. R. H. Byrd, G. M. Chin, J. Nocedal, and F. Oztoprak. A family of second-order methods for convex ℓ_1-regularized optimization. Technical report, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA, 2012.
6. L. Chen, Y. Wang, and G. He. A feasible active set QP-free method for nonlinear programming. *SIAM Journal on Optimization*, 17(2):401–429, 2006.
7. P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.
8. A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991.
9. A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.
10. C. W. Cryer. The solution of a quadratic programming problem using systematic overrelaxation. *SIAM Journal on Control*, 9(3):385–392, 1971.
11. L. Feng, V. Linetsky, J. L. Morales, and J. Nocedal. On the solution of complementarity problems arising in American options pricing. *Optimization Methods and Software*, 26(4-5):813–825, 2011.
12. H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl. qpOASES: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, pages 1–37, 2014.
13. I. B. Gharbia and J. C. Gilbert. Nonconvergence of the plain Newton-min algorithm for linear complementarity problems with a P-matrix. *Mathematical Programming*, 134(2):349–364, 2012.
14. P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: an SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.
15. P. E. Gill, W. Murray, and M. A. Saunders. *User's guide for SQOPT version 7: software for largescale linear and quadratic programming*. Systems Optimization Laboratory, Stanford University, Palo Alto, CA, USA, 2006.
16. P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Emerald Group Publishing Limited, Bingley, UK, 1982.
17. P. E. Gill and D. P. Robinson. Regularized sequential quadratic programming methods. Technical report, Department of Mathematics, University of California, San Diego, La Jolla, CA, USA, 2011.
18. N. I. M. Gould and D. P. Robinson. A second derivative SQP method: global convergence. *SIAM Journal on Optimization*, 20(4):2023–2048, 2010.
19. 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.

20. 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, 2011.
21. N. I. M. Gould and Ph. L. Toint. An iterative working-set method for large-scale nonconvex quadratic programming. *Applied Numerical Mathematics*, 43(1):109–128, 2002.
22. L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986.
23. W. W. Hager. The dual active set algorithm. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 137–142. North Holland, Amsterdam, 1992.
24. 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.
25. M. Hintermüller, K. Ito, and K. Kunisch. The primal-dual active set strategy as a semismooth Newton method. *SIAM Journal on Optimization*, 13(3):865–888, 2003.
26. M. M. Kostreva. Block pivot methods for solving the complementarity problem. *Linear Algebra and its Applications*, 21(3):207–215, 1978.
27. M. Kočvara and J. Zowe. An iterative two-step algorithm for linear complementarity problems. *Numerische Mathematik*, 68(1):95–106, 1994.
28. K. Kunisch and F. Rendl. An infeasible active set method for quadratic problems with simple bounds. *SIAM Journal on Optimization*, 14(1):35–52, 2003.
29. I. Maros and C. Mészáros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11(1-4):671–681, 1999.
30. 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.
31. J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, NY, USA, Second edition, 2006.
32. L. F. Portugal, J. J. Júdice, and L. N. Vicente. A comparison of block pivoting and interior-point algorithms for linear least squares problems with nonnegative variables. *Mathematics of Computation*, 63(208):625–643, 1994.
33. 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.
34. Ph. L. Toint. Non-monotone trust-region algorithms for nonlinear optimization subject to convex constraints. *Mathematical Programming*, 77(3):69–94, 1997.
35. M. Ulbrich and S. Ulbrich. Non-monotone trust region methods for nonlinear equality constrained optimization without a penalty function. *Mathematical programming*, 95(1):103–135, 2003.
36. V. Vapnik and C. Cortes. Support vector networks. *Machine Learning*, 20(3):273–297, 1995.
37. Y. Vardi, L. A. Shepp, and L. Kaufman. A statistical model for positron emission tomography. *Journal of the American Statistical Association*, 80(389):8–20, 1985.

A Appendix: Primal-Dual Active-Set as a Semi-Smooth Newton Method

In this appendix, we show that Algorithm 3 is equivalent to a semi-smooth Newton method under certain conditions. The following theorem utilizes the concept of a slant derivative of a slantly differentiable function [25].

Theorem 5 *Let* $\{(x_k, y_k, z_k^\ell, z_k^u)\}$ *be generated by Algorithm 3 with Step 6 employing Algorithm 4, where we suppose that, for all* k, $(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$ *with* $\mathcal{U}_k = \emptyset$ *is a feasible partition at the start of Step 3. Then,* $\{(x_k, y_k, z_k^\ell, z_k^u)\}$ *is the sequence of iterates generated by the semi-smooth Newton method for finding a zero of the function* KKT *defined by* (2) *with initial value* $(x_0, y_0, z_0^\ell, z_0^u) = \mathrm{SM}(\mathcal{A}_0^\ell, \mathcal{A}_0^u, \mathcal{I}_0, \emptyset)$ *and slant derivative* $M(a, b)$ *of the slantly differentiable function* $m(a, b) = \min(a, b)$ *defined by*

$$[M(a,b)]_{ij} = \begin{cases} 0 & \text{if } j \notin \{i, n+i\} \\ 1 & \text{if } j = i, a_i \leq b_j \\ 0 & \text{if } j = i, a_i > b_j \\ 0 & \text{if } j = n+i, a_i \leq b_j \\ 1 & \text{if } j = n+i, a_i > b_j. \end{cases}$$

Proof To simplify the proof, let us assume that $\ell = -\infty$ so that problem (1) has upper bounds only. This ensures that $z_k^\ell = 0$ and $\mathcal{A}_k^\ell = \emptyset$ for all k, so in this proof we remove all references to these quantities. The proof of the case with both lower and upper bounds follows similarly.

Under the assumptions of the theorem, the point $(x_0, y_0, z_0^u) \leftarrow \mathrm{SM}(\emptyset, \mathcal{A}_0^u, \mathcal{I}_0, \emptyset)$ is the first primal-dual iterate for both algorithms, i.e., Algorithm 3 and the semi-smooth Newton method. Furthermore, it follows from (4)–(6) that

$$Hx_0 + c - A^T y_0 + z_0^u = 0 \quad \text{and} \quad Ax_0 - b = 0. \tag{26}$$

We now proceed to show that both algorithms generate the same subsequent iterate, namely (x_1, y_1, z_1^u). The result then follows as a similar argument can be used to show that both algorithms generate the same iterate (x_k, y_k, z_k^u) for each k.

Partitioning the variable indices into four sets, namely I, II, III, and IV, we find:

$$\mathrm{I} := \{i : i \in \mathcal{I}_0 \ \text{ and } \ [x_0]_i \le u_i\} \implies [z_0^u]_i = 0; \tag{27a}$$
$$\mathrm{II} := \{i : i \in \mathcal{A}_0^u \ \text{ and } \ [z_0^u]_i \le 0\} \implies [x_0]_i = u_i; \tag{27b}$$
$$\mathrm{III} := \{i : i \in \mathcal{I}_0 \ \text{ and } \ [x_0]_i > u_i\} \implies [z_0^u]_i = 0; \tag{27c}$$
$$\mathrm{IV} := \{i : i \in \mathcal{A}_0^u \ \text{ and } \ [z_0^u]_i > 0\} \implies [x_0]_i = u_i. \tag{27d}$$

Here, the implications after each set follow from Step 2 of Algorithm 2. Next, (16) implies

$$\mathcal{I}_1 \leftarrow \mathrm{I} \cup \mathrm{II} \ \text{ and } \ \mathcal{A}_1 \leftarrow \mathrm{III} \cup \mathrm{IV}. \tag{28}$$

Algorithm 3 computes the next iterate as the unique point (x_1, y_1, z_1^u) satisfying

$$[z_1^u]_{\mathcal{I}_1} = 0, \quad [x_1]_{\mathcal{A}_1} = u_{\mathcal{A}_1}, \quad Hx_1 + c - A^T y_1 + z_1^u = 0, \quad \text{and} \quad Ax_1 - b = 0. \tag{29}$$

Now, let us consider one iteration of the semi-smooth Newton method on the function KKT defined by (2) using the slant derivative function M. It follows from (27), Table 13, and the definition of M that the semi-smooth Newton system may be written as

$$\begin{pmatrix}
H_{\mathrm{I,I}} & H_{\mathrm{I,II}} & H_{\mathrm{I,III}} & H_{\mathrm{I,IV}} & A_{\mathcal{N},\mathrm{I}}^T & I & 0 & 0 & 0 \\
H_{\mathrm{II,I}} & H_{\mathrm{II,II}} & H_{\mathrm{II,III}} & H_{\mathrm{II,IV}} & A_{\mathcal{N},\mathrm{II}}^T & 0 & I & 0 & 0 \\
H_{\mathrm{III,I}} & H_{\mathrm{III,II}} & H_{\mathrm{III,III}} & H_{\mathrm{III,IV}} & A_{\mathcal{N},\mathrm{III}}^T & 0 & 0 & I & 0 \\
H_{\mathrm{IV,I}} & H_{\mathrm{IV,II}} & H_{\mathrm{IV,III}} & H_{\mathrm{IV,IV}} & A_{\mathcal{N},\mathrm{IV}}^T & 0 & 0 & 0 & I \\
A_{\mathcal{N},\mathrm{I}} & A_{\mathcal{N},\mathrm{II}} & A_{\mathcal{N},\mathrm{III}} & A_{\mathcal{N},\mathrm{IV}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 \\
0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x_{\mathrm{I}} \\ \Delta x_{\mathrm{II}} \\ \Delta x_{\mathrm{III}} \\ \Delta x_{\mathrm{IV}} \\ -\Delta y \\ \Delta z_{\mathrm{I}} \\ \Delta z_{\mathrm{II}} \\ \Delta z_{\mathrm{III}} \\ \Delta z_{\mathrm{IV}}
\end{pmatrix}
= -
\begin{pmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ [z_0^u]_{\mathrm{II}} \\ [u - x_0]_{\mathrm{III}} \\ 0
\end{pmatrix}. \tag{30}$$

Table 13 Quantities relevant to evaluating the function KKT and computing the slant derivative M at the point (x_0, y_0, z_0^u).

Index set	$[z_0^u]_i$	$[u - x_0]_i$	$\min([z_0^u]_i, [u - x_0]_i)]$
$i \in \mathrm{I}$	0	≥ 0	0
$i \in \mathrm{II}$	≤ 0	0	$[z_0^u]_i$
$i \in \mathrm{III}$	0	< 0	$[u - x_0]_i$
$i \in \mathrm{IV}$	> 0	0	0

The first five block equations of (30) combined with (26) yield

$$Ax_1 - b = A(x_0 + \Delta x) - b = Ax_0 - b + A\Delta x = 0 \ \text{ and} \tag{31a}$$
$$Hx_1 + c - A^T y_1 + z_1^u = H(x_0 + \Delta x) + c - A^T(y_0 + \Delta y) + z_0^u + \Delta z = 0, \tag{31b}$$

while the last four blocks of equations of (30) and (27) imply

$$\Delta z_{\rm I} = 0 \implies [z_1^u]_{\rm I} = [z_0^u + \Delta z]_{\rm I} = 0 \tag{32}$$

$$\Delta z_{\rm II} = -[z_0^u]_{\rm II} \implies [z_1^u]_{\rm II} = [z_0^u + \Delta z]_{\rm II} = 0 \tag{33}$$

$$\Delta x_{\rm III} = [u - x_0]_{\rm III} \implies [x_1]_{\rm III} = [x_0 + \Delta x]_{\rm III} = u_{\rm III} \tag{34}$$

$$\Delta x_{\rm IV} = 0 \implies [x_1]_{\rm IV} = [x_0 + \Delta x]_{\rm IV} = u_{\rm IV} \tag{35}$$

so that

$$[z_1^u]_{\mathcal{I}_1} = 0 \ \text{ and } \ [x_1]_{\mathcal{A}_1} = u_{\mathcal{A}_1}. \tag{36}$$

It now follows from (29), (31), and (36) that (x_1, y_1, z_1^u) generated by the semi-smooth Newton method is the same as that generated by Algorithm 3.