2015

Primal-Dual Active-Set Methods for Convex Quadratic Optimization with Applications

Zheng Han
Lehigh University

Primal-Dual Active-Set Methods for Convex Quadratic Optimization with Applications

by

Zheng Han

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Industrial Engineering

Lehigh University

August 2015

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Date

Professor Frank E. Curtis, Dissertation Director

Accepted Date

Committee Members

Professor Frank E. Curtis, Committee Chair

Professor Jorge Nocedal

Professor Daniel P. Robinson

Professor Tamás Terlaky

Professor Luis F. Zuluaga

Acknowledgements

My gratitude towards my PhD advisor, Professor Frank E. Curtis, for his selfless support during the five years is beyond words. He has always been thoughtful in advising, generous (both financially and spiritually) in investing, and patient in mentoring. He also gives me much freedom and encouragement to pursue projects that best suit my ability and interest. I am fortunate to benefit tremendously not only from his thorough professional knowledge but also from his adherence to highest standard and meticulousness to research. He makes a paragon of the researcher, professor, and advisor one may possibly dream to become. His effort for advising me on being a researcher is priceless!

It is my great pleasure to have Professor Daniel P. Robinson, Professor Jorge Nocedal, Professor Tamás Terlaky, and Professor Luis F. Zuluaga in my committee who have been providing insightful discussions and suggestions that help me stand at the frontier of optimization research. I would also like to thank Professor Katya Scheinberg for exposing me to machine learning topics, Professor Ted Ralphs for pushing me to the world of advanced computing, and Rita Frey for her assistance on all graduate affairs. Additionally, sincerely thank Dr. Haifeng Chen for mentoring me on my summer research project at NEC Laboratories America.

I also owe many thanks to my colleagues who taught me that there are good times, hard times, but never bad times. Thank also the members at COR@L lab: Julio Góez, Aykut Bulut, Xiaocheng Tang, Jie Liu, and Seyedalireza Yektamaram for providing strong technical support without which most of the numerical results would not be available. Thanks also to my colleagues at the ISE department: Hao Wang, Lin He, Yunfei Song, Xi Bai, Wei Guo, Jiadong Wang, Choat Inthawongse, Murat Mut, Dan Li, Xiaocun Que, and many others. These years at Lehigh is a memorable experience of my life.

Finally, heartfelt thanks to my parents and my sister, who have been extremely supportive and respectful to every decision I have made.

Contents

List of Tables

List of Figures

Abstract

Primal-dual active-set (PDAS) methods are developed for solving quadratic optimization problems (QPs). Such problems arise in their own right in optimal control and statistics–two applications of interest considered in this dissertation–and as subproblems when solving nonlinear optimization problems. PDAS methods are promising as they possess the same favorable properties as other active-set methods, such as their ability to be warm-started and to obtain highly accurate solutions by explicitly identifying sets of constraints that are active at an optimal solution. However, unlike traditional active-set methods, PDAS methods have convergence guarantees despite making rapid changes in active-set estimates, making them well suited for solving large-scale problems.

Two PDAS variants are proposed for efficiently solving generally-constrained convex QPs. Both variants ensure global convergence of the iterates by enforcing montonicity in a measure of progress. Besides identifying an estimate set estimate, a novel uncertain set is introduced into the framework in order to house indices of variables that have been identified as being susceptible to cycling. The introduction of the uncertainty set guarantees convergence of the algorithm, and with techniques proposed to keep the set from expanding quickly, the practical performance of the algorithm is shown to be very efficient.

Another PDAS variant is proposed for solving certain convex QPs that commonly arise when discretizing optimal control problems. The proposed framework allows inexactness in the subproblem solutions, which can significantly reduce computational cost in large-scale settings. By controlling the level inexactness either by exploiting knowledge of an upper bound of a matrix inverse or by dynamic estimation of such a value, the method achieves convergence guarantees and is shown to outperform a method that employs exact solutions computed by direct factorization techniques.

Finally, the application of PDAS techniques for applications in statistics, variants are proposed for solving isotonic regression (IR) and trend filtering (TR) problems. It is shown that PDAS can solve an IR problem with n data points with only $\mathcal{O}(n)$ arithmetic operations. Moreover, the method is shown to outperform the state-of-the-art method for solving IR problems, especially when warm-starting is considered. Enhancements to the method are proposed for solving general TF problems, and numerical results are presented to show that PDAS methods are viable for a broad class of such problems.

Chapter 1

Introduction

Nonlinear optimization (NLO) models capture the complex nature of real-world problems. As a result, they have been widely used, and play an important role in areas such as machine learning [116, 105, 91, 96, 107, 6], optimal control [74, 75, 8], portfolio selection [89, 84], and many others [18, 53]. To get a sense of NLO models in practice, consider the following examples.

Support Vector Machine (SVM)

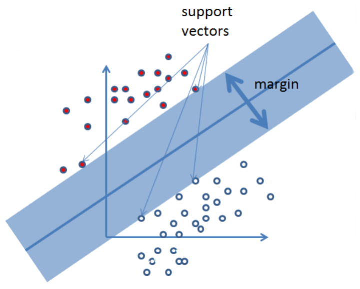


Figure 1.1: Maximize the margin between two classes [1]

A classical machine learning example is the support vector machine (SVM) problem which attempts to find a hyperplane that "best" separates two classes of labeled training data. In the separable case, there are infinitely many such hyperplanes, in which case an SVM model typically picks the one with the largest separation margin, a choice which

[1]Figure from Chapter 10, [6]

can be motivated by learning theory. Simply put, the SVM is designed to determine a hyperplane that minimizes the misclassification error of unseen data. For example, in the case illustrated in Figure 1.1, the labeled samples are linearly separable, and hence there exists $(\boldsymbol{w}, b)$ such that

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x} + b &\geq 1 \quad \forall \boldsymbol{x} \in \text{Class}_1, \\ \boldsymbol{w}^T\boldsymbol{x} + b &\leq -1 \ \forall \boldsymbol{x} \in \text{Class}_2. \end{aligned}$$

Suppose there are n labeled samples $(\boldsymbol{x}_i, y_i)$ for $i = 1, \ldots, n$, the hyperplane is then parameterized by the pair $(\boldsymbol{w}, b)$ and the SVM problem is formulated [116] as an NLP:

$$\begin{aligned} &\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &\text{s.t. } y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \ i = 1, \ldots, n. \end{aligned} \tag{1.1}$$

In particular, (1.1) is an NLP with the objective function being quadratic and constraints affine. Such an NLP is also called a quadratic optimization problem (QP) and in the non-separable case the SVM problem is similarly [116] formulated as a QP.

Model Predictive Control (MPC)

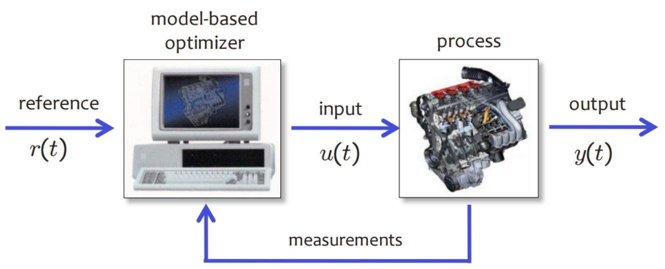


Figure 1.2: Minimize predicted performance cost [2]

Model predictive control (MPC) is an example of using online optimization techniques in industrial practice. MPC aims to model the behavior of complex dynamical systems, predict future control inputs and outputs, and minimize the predicted performance cost at regular intervals. The complex dynamical system is usually approximated in discrete-time

[2] Figure from [7]

and the optimal control problem over a finite future horizon of N steps can be formulated as an NLP in the form of

$$\begin{aligned} &\min_{u_k} \sum_{k=0}^{N-1} L(y_k - r(t+k), u_k) \\ &\text{s.t. } x_{k+1} = f(x_k, u_k), \text{ for } k = 0, 1, \ldots, N-1 \\ &\quad\; y_k = g(x_k, y_k), \text{ for } k = 0, 1, \ldots, N-1 \\ &\quad\; u_{min} \leq u_k \leq u_{max}, \text{ for } k = 0, 1, \ldots, N-1 \\ &\quad\; y_{min} \leq y_k \leq y_{max}, \text{ for } k = 0, 1, \ldots, N-1 \\ &\quad\; x_0 = x(t). \end{aligned} \tag{1.2}$$

The predicting error and the actuation are two factors of the performance cost function L of (1.2). Specifically, $y_k - r(t+k)$ represents the predicting error and u_k the actuation.

Portfolio Selection

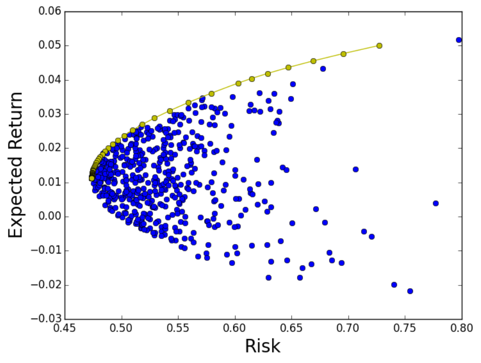


Figure 1.3: Trade-off between risk and expected return

Consider the decision of an investment on n assets each with return r_i as a random variable for $i = 1, \ldots, n$. Denote that the expected return $\mu_i = E[r_i]$, variance $\sigma_i^2 = E[(r_i - \mu_i)^2]$, and covariance between each pair (i, j) as

$$\rho_{ij} = \frac{E[(r_i - \mu_i)(r_j - \mu_j)]}{\sigma_i \sigma_j}, \text{ for } i, j = 1, 2, \ldots, n.$$

It is desirable to create a portfolio $x \in \mathbb{R}^n$ where x_i represents the fraction of funds invested in asset i such that the portfolio $R = x^T r$ has relatively high expected return and low risk (measured by variance) which are usually at odds with each other. Notice that for portfolio R we have

$$E[R] = E[\sum_{i=1}^{n} x_R R_i] = x^T \mu,$$
$$Var[R] = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j \sigma_i \sigma_j \rho_{ij} = x^T G x.$$

To trade-off between expected return and risk, one may employ $\kappa > 0$ and formulate the following optimization model

$$\begin{aligned} \max_{x} \ & x^T \mu - \kappa x^T G x \\ \text{s.t.} \ & \sum_{i=1}^{n} x_i = 1, \ \ x \geq 0. \end{aligned}$$

Each x is associated with a risk and expected return value as illustrated in Figure 1.3. By selecting different values of κ, the solutions constitute the efficient frontier (yellow curve).

1.1 Overview

As shown in previous examples, a typical nonlinear optimization problem (NLP) is formulated to maximize/minimize an objective function subject to potential constraints on the decision variables. Mathematically, an NLP is expressed as

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \ & f(x) \\ \text{s.t.} \ & c^{\mathcal{E}}(x) = 0, c^{\mathcal{I}}(x) \leq 0, \end{aligned} \tag{1.3}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$, $c^{\mathcal{E}} : \mathbb{R}^n \mapsto \mathbb{R}^{|\mathcal{E}|}$, and $c^{\mathcal{I}} : \mathbb{R}^n \mapsto \mathbb{R}^{|\mathcal{I}|}$ are differentiable. Some seemingly nonsmooth optimization problems could also be cast in the form of (1.3).

A good NLP model should be complex enough to capture the intricacy of the real-world problem yet simple enough to be solved relatively quickly. A quadratic optimization problem (QP) is such a model that is formulated as

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \ & \frac{1}{2}x^T Hx + c^T x \\ \text{s.t. } & Ax = b, \ \ell \le x \le u. \end{aligned} \tag{1.4}$$

where $H \in \mathbb{R}^{n\times n}$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $\{\ell, u\} \subset \bar{\mathbb{R}}^n$(i.e., the extended n-dimensional reals that include infinite values). QPs are one of the most common types of NLPs formulated and solved in practice, and have been studied extensively in numerous areas of applied mathematics [15, 16, 53, 74, 75, 85, 88, 110, 116]. Moreover, there are well-developed techniques [43, 44, 50, 51, 52, 97] for (1.3) by iteratively solving a series of QPs. A bound constrained QP (BQP) is a special QP with only bound constraints on the variables:

$$\begin{aligned} \min_{x} \ & \frac{1}{2}x^T Hx + c^T x \\ \text{s.t. } & \ell \le x \le u. \end{aligned} \tag{1.5}$$

Since the computational efficiency of the solver for QPs is of paramount importance for an NLP solver, in much of this dissertation we focus on algorithms for solving large-scale QPs.

Three popular classes of NLO algorithms are interior-point methods (IPM), augmented Lagrangian (AL) methods, and active-set methods. IPMs draw on the power of Newton's method to solve systems of nonlinear equations by solving a series of linearized equation systems. The resulting algorithms run in polynomial-time [95, 94, 119, 33], and thus are believed to be effective for large-scale problems. Augmented Lagrangian methods represent a classical approach that are regaining favor in large-scale settings. The strength of augmented Lagrangian methods is that they only require solving unconstrained or bound-constrained subproblems [68, 102, 46] which, in some situations, can be cheaper than solving linear systems or QPs as in IPMs or active-set methods respectively. Active-set methods follow a different paradigm and have many distinctive features. The main advantage of active-set methods is their ability to warm-start with a good initial point. These methods have achieved great success in Sequential Quadratic Optimization(SQO) methods [43, 44, 50, 51, 52, 97]. Additionally, they also can yield very accurate solutions once the optimal active set is identified. Active-set methods are most used in solving small- or medium-scale problems yet all their desirable properties motivate us to research further the use of active-set methods for large-scale problems.

Algorithms for solving convex QPs have been studied for decades and they generally fall into the categories of active-set [11, 36, 97] and interior-point methods [83, 95, 115, 119]. There are also a variety of methods designed exclusively for BQPs. These include active-set [39, 45], interior-point [22, 67], gradient projection [10, 29, 40, 97], or some combination of these methods [13, 64, 92].

Most active-set methods are initialized by making an estimate of the optimal active set, then iteratively change the estimate until optimality is reached. Hence, the ability to identify the optimal active set quickly is crucial in the design of efficient active-set algorithms. Since active-set methods for NLP typically employ active-set methods to solve QP subproblems, this dissertation will focus on active-set methods for QPs. In particular, we focus on primal-dual active-set (PDAS) methods for solving convex QPs in the favor of the methods' promising performance in large-scale settings. A PDAS-like method was proposed [1] as a Newton-type method to solve linear complementarity problems (LCPs), see also in [78, 100] as block pivoting algorithms and [25] for a comprehensive overview of pivoting algorithms for LCPs. Recently the method received much attention as it is shown to replicate semismooth Newton method [69] as well as for its impressive practical performance.

The algorithmic research of PDAS methods roughly follows two lines: the investigation of conditions under which a plain PDAS method without safeguards assumes global convergence and the extension (with enhancements and safeguards) of PDAS variants for more general QPs. For the former research, see [69, 86, 76] that impose conditions on the Hessian of BQP (1.5) to guarantee global convergence and [42, 27] that provide examples of LCP and BQP that PDAS cycles. Moreover, inexactly solving the subproblems might greatly improve the efficiency of PDAS [26] for certain QPs. Extending the PDAS methods to more general problems entails safeguards on the algorithm design. Straightforward approaches include [81, 21, 78] that incorporate heuristics to detect suspicious cycles and switch between PDAS and Murty's method [93] where each update of the active-set only adds/removes one constraint. A variant enforces convergence by encouraging primal feasibility and a potentially recursive subproblem solution method is proposed in [73] for strictly convex BQPs. Finally, there are also numerous articles on the use of PDAS methods for various applications [31, 55, 70, 71, 72, 87, 108].

1.2 Contributions of This Dissertation

Overall, the goals of the research in this dissertation are to enhance the theoretical and algorithmic development of PDAS methods so that they may be effective for solving general convex QPs, both on their own and in the context of solving NLPs. We summarize our main contributions toward these ends as the following.

1. We propose a novel PDAS algorithmic framework that extends the PDAS method of [69] to general convex QPs of the form (1.4). Global convergence is guaranteed via the introduction of an uncertainty set that houses indices of variables that are suspected of cycling. The framework allows rapid updates of the active-set estimate, and thus it is well-suited for solving large-scale problems

2. We propose enhancements of a PDAS method for solving certain convex QPs arising in various applications such as optimal control. By exploiting the special structure of the QPs of interest, one only needs to solve the subproblem involved in each PDAS iteration *inexactly*, yet global convergence is still guaranteed. We show that incorporating inexactness can significantly reduce the overall computational costs compared to a similar strategy that employs exact subproblem solutions obtained via direct factorizations.

3. We demonstrate that PDAS methods can be extremely efficient when solving a broad class of large-scale optimization problems arising in statistical learning. In particular, we prove that a PDAS method is able to solve an isotonic regression problem [12] of n points in $\mathcal{O}(n)$ elementary arithmetic operationsthe best possible complexity and outperforms the state-of-the-art method for solving such problems. We also propose

a novel safeguarding strategy that leads to a PDAS variant that shows competitive performance when solving a related class of trend filtering problems [82].

4. We provide a software package known as `pypdas`, which includes a generic PDAS framework that allows inexact subproblem solves. The package is implemented in Python and available as open-source, which should facilitate other researchers efforts in the study of PDAS methods.

The rest of this dissertation is organized in the following order. We first provide the background of active-set methods and review several active-set methods as well as closely-related algorithms for QPs in §2. Then a globally convergent PDAS framework for general strictly convex QPs is described in §3. We investigate in §4 a special class of convex QPs in optimal control and propose PDAS methods that only require solving inexactly the subproblems. Finally, we illustrate in §5 that the PDAS methods could be applied to a class of statistical learning applications where PDAS methods show very competitive performance. The generic PDAS framwork is implemented in the open-source package `pypdas` to facilitate study of the PDAS methods. A brief instruction of calling the package for solving general convex QPs is attached in the Appendix B.

Chapter 2

Background and Related Algorithms

2.1 Active-Set Methods for QP

In this section, we review briefly several known active-set methods for solving convex QPs. The primal and dual active-set methods are able to solve general convex QPs whereas the gradient projection method and primal-dual active-set method were mostly used to solve convex BQPs. We start by introducing necessary concepts and notation that would be used throughout this dissertation. For convenience, we denote the set of variable indices and equality constraint indices of (1.4), respectively, as

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

We also define a *partition* $\mathcal{P}(x)$ as a triplet of sets $(\mathcal{A}^\ell(x), \mathcal{A}^u(x), \mathcal{I}(x))$ identified by x, where

$$\mathcal{A}^\ell(x) := \{i \in \mathcal{N} : \; x_i = \ell_i\}, \tag{2.1a}$$

$$\mathcal{A}^u(x) := \{i \in \mathcal{N} : \; x_i = u_i\}, \tag{2.1b}$$

$$\mathcal{I}(x) := \mathcal{N} \backslash (\mathcal{A}^\ell \cup \mathcal{A}^u). \tag{2.1c}$$

Hereinafter, we use $\mathcal{P}, \mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}$ instead of $\mathcal{P}(x), \mathcal{A}^\ell(x), \mathcal{A}^u(x), \mathcal{I}(x)$ for simplicity. A partition identified by the optimal solution x_* is called the optimal partition and is denoted by $\mathcal{P}_* = (\mathcal{A}^\ell_*, \mathcal{A}^u_*, \mathcal{I}_*)$. We further denote the index set $\mathcal{A}$ as the union of lower active-set $\mathcal{A}^\ell$ and upper active-set $\mathcal{A}^u$, or in other words $\mathcal{A} := \mathcal{A}^\ell \cup \mathcal{A}^u$. The complement of $\mathcal{A}$, i.e., $\mathcal{I}$ is called the *inactive-set.* A *working set* $\mathcal{W}$ is defined as $\mathcal{A} \cup \mathcal{M}$ that includes additionally the equality constraints.

Notation We use a counter as a subscript to denote iteration number and use a set as a subscript to denote (sets of) elements of a vector or matrix; e.g., with x_k we denote the vector x at the kth iteration and with $x_{\mathcal{S}}$ we denote the vector composed of the elements in the vector x corresponding to those indices in the ordered set $\mathcal{S}$. Similarly, 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. When we refer to (sets of) elements of a vector with an additional subscript, such as x_*, we denote sets of elements after appending brackets, such as in $[x_*]_{\mathcal{S}}$. We also occasionally denote a vector composed of stacked subvectors as an ordered tuple of vectors, i.e., for vectors a and b we occasionally write $(a, b) := [a^T \; b^T]^T$. For a square matrix S, we write $S \succ 0$ ($S \succeq 0$) to indicate that S is positive definite (semidefinite). Finally, e denotes a vector of ones whose size is determined by the context in which it appears.

Optimality Conditions

We use Karush-Kuhn-Tucker (KKT) conditions to characterize the optimal solution x_* of (1.4). Specifically, for x_* to be optimal, there must exist dual variables $y_* \in \mathbb{R}^m$ and

$\{z_*^\ell, z_*^u\} \subset \mathbb{R}^n$ associated with x_* such that:

$$\begin{aligned} Hx_* + A^T y_* - z_*^\ell + z_*^u + c &= 0, && (2.2a) \\ Ax_* &= b, && (2.2b) \\ (x_* - \ell) \circ z_*^\ell &= 0, && (2.2c) \\ (u - x_*) \circ z_*^u &= 0, && (2.2d) \\ (x_* - \ell, u - x_*, z_*^\ell, z_*^u) &\geq 0, && (2.2e) \end{aligned}$$

where $\circ$ represents an element-wise product. The KKT conditions serve naturally as an optimality verification criteria of a primal-dual solution (x, y, z^ℓ, z^u). In particular, we can employ a KKT error measure $\mathrm{KKT}(x, y, z^\ell, z^u)$ to evaluate how close a given point is to optimality. Specifically, for (x, y, z^ℓ, z^u) to be optimal, we must have

$$0 = \mathrm{KKT}(x, y, z^\ell, z^u) := \begin{pmatrix} Hx + A^T y + c - z^\ell + z^u \\ Ax - b \\ \min\{x - \ell, z^\ell\} \\ \min\{u - x, z^u\} \end{pmatrix}. \tag{2.3}$$

For future reference, when equality constraints are not present so that (1.4) reduces to (1.5), the dual variable y is also not necessary and we refer to the residual simply as $\mathrm{KKT}(x, z^\ell, z^u)$.

Once the optimal partition $\mathcal{P}$ is known for (1.4), we could simply fix $x_{\mathcal{A}^\ell} = \ell_{\mathcal{A}^\ell}$, $x_{\mathcal{A}^u} = u_{\mathcal{A}^u}$ and drop off all bound constraints on $x_{\mathcal{I}}$. Problem (1.4) then becomes an equality constrained QP which is much easier to solve. This is an important concept to which we refer numerous times throughout the dissertation. To be consistent with this idea, we define the *subspace minimizer* of a partition. Given a partition $\mathcal{P} = (\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I})$ of a strictly convex QP (1.4), a subspace minimizer of $\mathcal{P}$ is a primal-dual solution (x, y, z^ℓ, z^u)

such that

$$x_{\mathcal{A}^\ell} = \ell_{\mathcal{A}^\ell}, \tag{2.4a}$$

$$x_{\mathcal{A}^u} = u_{\mathcal{A}^u}, \tag{2.4b}$$

$$z^\ell_{\mathcal{I}\cup\mathcal{A}^u} = 0, \tag{2.4c}$$

$$z^u_{\mathcal{I}\cup\mathcal{A}^\ell} = 0, \tag{2.4d}$$

$$Hx + A^T y - z^\ell + z^u + c = 0, \tag{2.4e}$$

$$Ax = b. \tag{2.4f}$$

The Dual Problem

Sometimes the dual problem of (1.4) may shed some light on the primal problem that is to be solved. Suppose H of (1.4) is positive semi-definite, i.e., (1.4) is a convex QP, the dual problem is then

$$\begin{aligned} \max_{x,y,z}\ & \frac{1}{2}x^T Hx + (c + A^T y + z^u - z^\ell)^T x - b^T y - u^T z^u + \ell^T z^\ell \\ \text{s.t. } & Hx + c + A^T y + z^u - z^\ell = 0 \\ & z^u, z^\ell \geq 0. \end{aligned} \tag{2.5}$$

Suppose further that H is strictly positive definite, we can eliminate x and yield a convex BQP:

$$\begin{aligned} \min_{y,z^u,z^\ell}\ & \frac{1}{2}(c + A^T y + z^u - z^\ell)^T H^{-1}(c + A^T y + z^u - z^\ell) + b^T y + u^T z^u - \ell^T z^\ell \\ \text{s.t. } & z^u, z^\ell \geq 0. \end{aligned} \tag{2.6}$$

Now we are in a position to review several popular active-set methods: primal active-set methods and dual active-set methods for QPs (1.4); gradient projection methods and primal-dual active-set methods for BQPs (1.5). All of these methods seek to find a stationary point $(x_*, y_*, z^\ell_*, z^u_*)$ satisfying the optimality conditions of (2.2).

2.1.1 Primal Active-Set Methods for QP

Primal active-set methods are seen as extension of simplex methods to nonlinear optimization and have been extensively studied [39, 45, 41]. The algorithms keep primal variables x feasible on all iterations, and alternate between minimizing in the primal space and updating dual variables until all variables become feasible. In classic active-set algorithms, an initial feasible point x_0 must be specified which is obtainable via linear optimization, i.e., the so-called Phase-I. Two types of procedures, namely a primal-procedure and dual-procedure, are involved in the algorithm iterations. The primal-procedure searches in a subspace of the feasible region and expands the active set when the boundary of the feasible region is hit. When a subspace minimizer is achieved, the dual-procedure is invoked which either terminates the algorithm if optimality is detected or shrinks the active set by one unit otherwise. Under certain assumptions, the algorithm is globally convergent and performs well when served with a good initial point. Furthermore, since the algorithm treats active inequalities as equalities, it only searches in the reduced space hence it can save computational as well as storage costs. The algorithm framework is shown in Algorithm 1.

Now we list several drawbacks of primal active-set methods. First, a primal active-set method can only iterate on the feasible region. However, finding a feasible initial solution may require a considerable amount of computation for large-scale problems. In addition, a primal active-set method behaves badly if the problem is ill-conditioned and/or the initial point poorly chosen. Furthermore, it is difficult to obtain an efficient implementation of the classic active-set method as one has to deal with degeneracy and other numerical issues.

Figure 2.1 demonstrates the slow update of the active-set in classic active-set method. The algorithm adds or drops only one active constraint per update, and thus it may take numerous iterations to adapt a badly estimated active set to the optimal one when n is large. Much can be improved if we allow dramatic changes of the active set per update.

Algorithm 1 Primal active-set method for convex QP

```
1: Input an initial feasible solution $x_0$ and initial active-set $\mathcal{A}_0 = \mathcal{A}_0^\ell \cup \mathcal{A}_0^u$
2: for $k = 0, 1, 2, \dots$ do
3:     Solve
           $$\begin{aligned} \min_{p_k \in \mathbb{R}^n} \ & \frac{1}{2} p_k^T H p_k + (c + Hx_k)^T p_k \\ \text{s.t. } & Ap_k = 0 \\ & [x_k]_{\mathcal{A}_k^\ell} + [p_k]_{\mathcal{A}_k^\ell} = \ell_{\mathcal{A}_k^\ell} \\ & [x_k]_{\mathcal{A}_k^u} + [p_k]_{\mathcal{A}_k^u} = u_{\mathcal{A}_k^u} \end{aligned} \quad (2.7)$$
4:     if $p_k = 0$ then
5:         Let $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}) = (\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k)$ and set
               $$[z^\ell]_{\mathcal{A}^u \cup \mathcal{I}} \leftarrow 0, \ [z^u]_{\mathcal{A}^\ell \cup \mathcal{I}} \leftarrow 0. \quad (2.8)$$
           Compute dual variables $(y, z^\ell, z^u)$ satisfying
               $$A^T y - \begin{pmatrix} 0 \\ z^\ell_{\mathcal{A}^\ell} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ z^u_{\mathcal{A}^u} \end{pmatrix} = -(Hx_k + c). \quad (2.9)$$
6:         if $(z^\ell, z^u) \geq 0$ then
7:             Stop with solution $x_* = x_k$.
8:         else
9:             Set $j = \operatorname{argmin}_{j \in \mathcal{A}} [z^\ell + z^u]_j$
               $x_{k+1} = x_k$; $\mathcal{A}_{k+1}^\ell = \mathcal{A}_k^\ell \backslash \{j\}$, $\mathcal{A}_{k+1}^u = \mathcal{A}_k^u \backslash \{j\}$, $\mathcal{I}_{k+1} = \mathcal{I}_k \cup \{j\}$.
10:        end if
11:    else
12:        Compute
               $$\alpha_k := \min\left(1, \min_{i \in \mathcal{I}_k}\left(\max\left(\frac{u_j - [x_k]_j}{[p_k]_j}, \frac{l_j - [x_k]_j}{[p_k]_j}\right)\right)\right) \quad (2.10)$$
13:        $x_{k+1} = x_k + \alpha_k p_k$;
14:        if there are blocking constraints then
15:            Obtain $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u)$ by moving one of the blocking constraints from $\mathcal{I}_k$
16:        else
17:            $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}) \leftarrow (\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k)$
18:        end if
19:    end if
20: end for
```

2.1.2 Dual Active-Set Methods for QP

Dual active-set methods have much theoretical appeal [62, 47, 4, 106]. Complicated constraints are potentially replaced by dual variables in much simpler form as seen in (2.6). Unlike primal active-set method, specifying a feasible initial dual solution is trivial. For example, fixing all dual variables to 0 will suffice. A dual active-set method framework for QPs was proposed in [47]. We illustrate the method in Algorithm 2 which maintains dual feasibility and updates primal variables until optimality is reached. Both primal and dual active-set method suffer from the slow update of the active set. However in certain

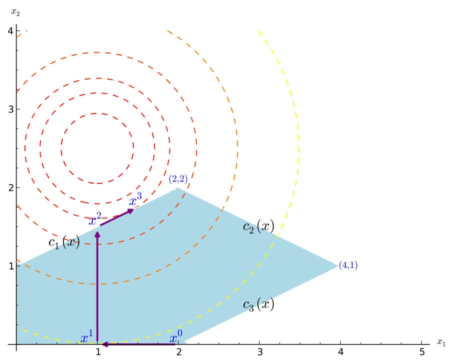


Figure 2.1: Illustration of classic active-set method

applications dual active-set methods appear more favorable in practice.

We next turn to discuss two efficient active-set methods, namely the gradient projection methods and the primal-dual active-set methods. In contrast to primal active-set methods and dual active-set methods, both algorithms to be reviewed are able to make rapid changes of the active-set estimate. In particular, the PDAS method described herein is motivated by the work of [69] that solves BQPs satisfying some special properties.

2.1.3 Gradient Projection Methods for BQP

Unlike the primal (dual) active-set method, Gradient Projection (GP) methods allow rapid changes of the active-set estimate. Assuming that the initial point is feasible, GP methods either search along the path of a projected direction [97] or along a direction to a projected point [65, 30]. The former methods require the projection of a path, while the latter ones usually only require the projection of a point. Our description will focus on the former framework, but first we refer to some relevant work of the latter variety for the readers. Hager and Zhang [65] proposed a GP algorithm for bound constrained NLPs. Their algorithm is designed by a set of rules to branch between a nonmonotone gradient projection phase and an unconstrained optimization phase. The algorithm converges globally to a stationary point even for degenerate problems. Dai and Fletcher [30] designed a GP method for BQP with an additional linear constraint. Both algorithms employed Barzilai-Borwein [5] steps and a nonmonotonic line search[57].

When projection is cheap as in the case of BQP, GP methods are well-suited and often

Algorithm 2 Dual active-set method

```
1: Input an initial feasible dual solution $(y_0, z_0^\ell, z_0^u)$ and partition $(\mathcal{I}_0, \mathcal{A}_0^\ell, \mathcal{A}_0^u)$ consistent with $(y_0, z_0^\ell, z_0^u)$.
2: for $k = 0, 1, 2, \dots$ do
3:     Let $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}) = (\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k)$, then compute $(y, z^\ell, z^u)$ by setting
           $$[z^\ell]_{\mathcal{A}^u \cup \mathcal{I}} \leftarrow 0,\ [z^u]_{\mathcal{A}^\ell \cup \mathcal{I}} \leftarrow 0, \tag{2.11}$$
       and solving
           $$\begin{aligned} \min\ & \frac{1}{2}x^T Hx + c^T x \\ \text{s.t. }\ & Ax = b \\ & x_{\mathcal{A}^\ell} = \ell_{\mathcal{A}^\ell} \\ & x_{\mathcal{A}^u} = u_{\mathcal{A}^u} \end{aligned} \tag{2.12}$$
4:     if $x$ is feasible then
5:         $x_* \leftarrow x$; Stop.
6:     else
7:         Compute step length $\alpha_k$
           $$\alpha_k = \max\{t :\ z_k^\ell + t(z^\ell - z_k^\ell) \geq 0, z_k^u + t(z^u - z_k^u) \geq 0\}, \tag{2.13}$$
8:         if $\alpha_k < 1$ then
9:             Set
               $$\begin{aligned} x_{k+1} &\leftarrow x_k + \alpha_k(x - x_k) \\ z_{k+1}^\ell &\leftarrow z_k^\ell + \alpha_k(z^\ell - z_k^\ell) \\ x_{k+1} &\leftarrow x_k + \alpha_k(x - x_k) \end{aligned} \tag{2.14}$$
10:            Shrink $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u)$ by dropping one of the blocking dual constraints into $\mathcal{I}_k$.
11:        else
12:            Set $(x_{k+1}, z_{k+1}^\ell, z_{k+1}^u) = (x, z^\ell, z^u)$
13:            Expand $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u)$ by adding one of the blocking primal constraints from $\mathcal{I}_k$.
14:        end if
15:    end if
16: end for
```

outperform other methods. For convenience of description, we define the projection of a point x onto the feasible region of (1.5) by

$$P(x, \ell, u) = \text{Median}(x, \ell, u), \tag{2.15}$$

where "Median" takes the element-wise median of x, ℓ and u. The search direction at a point x_k is usually chosen as the projected steepest descent direction which is denoted as the piecewise linear path

$$x(\alpha_k) := P(x_k - \alpha_k \nabla f(x_k), \ell, u). \tag{2.16}$$

We define the Cauchy-point x_k^C as the solution on the path (2.16) that minimizes the objective function. For BQP, x_k^C can be computed efficiently [97] and a typical GP framework is presented in Algorithm 3.

Algorithm 3 Gradient projection method for BQP

```
1: Input an initial feasible point $x_0$;
   set $k \leftarrow 0$, $z_0^\ell = \max\{0, Hx_0 + c\}$, $z_0^u = \max\{0, -Hx_0 - c\}$;
2: while $r(x_k, z_k^\ell, z_k^u) \neq 0$ do
3:     Find the Cauchy point $x_k^C$;
4:     Find an approximate solution $x^+$ of
```

$$\min_x f(x) = \frac{1}{2}x^T Hx + c^T x \tag{2.17a}$$

$$\text{s.t. } x_{\mathcal{A}(x_k^C)} = [x_k^C]_{\mathcal{A}(x_k^C)} \tag{2.17b}$$

$$\ell_{\mathcal{I}(x_k^C)} \leq x_{\mathcal{I}(x_k^C)} \leq u_{\mathcal{I}(x_k^C)} \tag{2.17c}$$

```
       such that $f(x^+) \leq f(x_k^C)$ and $x^+$ is feasible; $x_{k+1} \leftarrow x^+$;
5:     Compute $z_{k+1}^\ell = \max(0, Hx_{k+1} + c)$, $z_{k+1}^u = \max(0, -Hx_{k+1} - c)$;
       set $k \leftarrow k + 1$.
6: end while
```

An exact solution of the subspace minimization problem (2.17) is not necessary to ensure convergence. Thus, for large-scale problems, one typically employs an approach to obtain an approximate solution. The Cauchy point x_k^C is a feasible solution of (2.17) that is usually considered a candidate. One popular strategy to balance between choosing the Cauchy point and solving problem (2.17) exactly is to apply the conjugate gradient (CG) method to (2.17a) and (2.17b) and terminate once (2.17c) is satisfied. Alternatively, instead of terminating immediately upon satisfaction of (2.17c), we can continue solving with more CG iterations to achieve a more accurate solution and project the solution to the feasible region of (2.17c).

The iterate directions of GP methods are usually confined to projected steepest descent directions [65, 97]. Such a limited selection ensures global convergence, but also may inhibit the algorithm from converging quickly. Some methods choose projected Newton directions, but usually have to switch to the steepest descent direction when the projected Newton direction is not a descent one [79]. Moreover, when the projection operation is expensive, GP is not an appropriate choice.

2.1.4 Primal-Dual Active-Set Methods for BQP

The primal-dual active-set (PDAS) method discussed here was motivated by the work of Hintermüler, Ito, and Kunisch [69] in solving the BQP (1.5) with an additional assumption that H is a (perturbed) M-matrix. An M-matrix is a positive definite matrix with non-positive off-diagonal entries. The key feature of this method is its rapid adaptation of the active-set estimate. In contrast to the description in [69], we illustrate the PDAS method via updates to partitions. For convenience, we denote (z^ℓ, z^u) as dual variables associated with the lower and upper bounds, respectively. Without equality constraints, the KKT conditions (2.2) now become:

$$\begin{aligned} Hx_* - z_*^\ell + z_*^u + c &= 0, && (2.18a)\\ (x_* - \ell) \circ z_*^\ell &= 0, && (2.18b)\\ (u - x_*) \circ z_*^u &= 0, && (2.18c)\\ (x_* - \ell, u - x_*, z_*^\ell, z_*^u) &\geq 0. && (2.18d) \end{aligned}$$

We rephrase the PDAS method of [69] in Algorithm 4, but in a more general form that includes both lower and upper bounds on variables.

Algorithm 4 alternates between two phases: a subspace minimization phase (step 3–5) and a partition update phase (step 9). The subspace minimization phase fixes $x_{\mathcal{A}^\ell}$ and $x_{\mathcal{A}^u}$ to their lower and upper bounds respectively, and sets $z^\ell_{\mathcal{I}\cup\mathcal{A}^u}$ and $z^u_{\mathcal{I}\cup\mathcal{A}^\ell}$ to 0 (step 3). The remaining unknowns, namely $x_{\mathcal{I}}$, $z^\ell_{\mathcal{A}^\ell}$ and $z^u_{\mathcal{A}^u}$, are set by solving a linear system of equations (step 4–5). Step 6 verifies optimality and terminates if the optimal solution is found and turns to the partition update phase otherwise. Every partition $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I})$ defines a unique subspace minimizer (x, z^ℓ, z^u) based on which we can decide whether it is optimal or an update to the partition should be applied.

When the variables of problem (1.5) are only bounded above and the Hessian H is an M-matrix, then global convergence of Algorithm 4 is guaranteed. Many results of this dissertation are inspired by the convergence results thus we state them in Theorem 3.2.1 and provide a brief proof highlighting the behaviors of Algorithm 4.

Algorithm 4 Primal-Dual Active-Set Method for BQP

```
1: Input $(\mathcal{A}_0^\ell, \mathcal{A}_0^u, \mathcal{I}_0)$ and set $k \leftarrow 0$.
2: loop
3:     Let $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}) = (\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k)$ and set
           $$x_{\mathcal{A}^\ell} \leftarrow \ell_{\mathcal{A}^\ell},\ x_{\mathcal{A}^u} \leftarrow u_{\mathcal{A}^u},\ z^\ell_{\mathcal{I}\cup\mathcal{A}^u} \leftarrow 0, \text{ and } z^u_{\mathcal{I}\cup\mathcal{A}^\ell} \leftarrow 0. \quad (2.19)$$
4:     Let $\mathcal{A} \leftarrow \mathcal{A}^\ell \cup \mathcal{A}^u$ and set $x_\mathcal{I}$ as the solution of
           $$H_{\mathcal{I}\mathcal{I}} x_\mathcal{I} = -c_\mathcal{I} - H_{\mathcal{I}\mathcal{A}} x_\mathcal{A}. \quad (2.20)$$
5:     Set
           $$z^\ell_{\mathcal{A}^\ell} \leftarrow [Hx + c]_{\mathcal{A}^\ell} \text{ and } z^u_{\mathcal{A}^u} \leftarrow -[Hx + c]_{\mathcal{A}^u}. \quad (2.21)$$
       Let $(x_k, z_k^\ell, z_k^u) = (x, z^\ell, z^u)$.
6:     if $r(x_k, z_k^\ell, z_k^u) = 0$ then
7:         return $(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k)$ and $(x_k, z_k^\ell, z_k^u)$
8:     else
9:         Set
           $$\mathcal{A}^\ell \leftarrow \{i : i \in \mathcal{A}^\ell \text{ and } z^\ell \ge 0, \text{ or } i \in \mathcal{I} \text{ and } x_i < \ell_i\}, \quad (2.22a)$$
           $$\mathcal{A}^u \leftarrow \{i : i \in \mathcal{A}^u \text{ and } z^u \ge 0, \text{ or } i \in \mathcal{I} \text{ and } u_i - x_i < 0\}, \quad (2.22b)$$
           $$\mathcal{I} \leftarrow \mathcal{N} \backslash (\mathcal{A}^\ell \cup \mathcal{A}^u). \quad (2.22c)$$
10:    end if
11: end loop
```

Theorem 2.1.1. *([69, Theorem 3.2]) Assume that H is an M-matrix and $\ell = -\infty$. Then $(x_k, z_k^u) \to (x_*, z_*^u)$ for any arbitrary initial partition. Moreover,*

$$x_* \le x_{k+1} \le x_k \qquad \textit{for all } k \ge 0 \tag{2.23a}$$

$$x_k \le u \qquad \textit{for all } k \ge 1 \tag{2.23b}$$

Proof. Since $\ell = -\infty$, we choose $\mathcal{A}^\ell = \emptyset$, set $z^\ell = \mathbf{0}$ as a constant and denote $\mathcal{A}^\ell$ as $\mathcal{A}$ and z^u as z. We first investigate the monotonic properties of x. The iteration where (x_{k-1}, z_{k-1}) determines $(\mathcal{A}_k, \mathcal{I}_k)$ which in turn determines (x_k, z_k) is considered. Denote $\Delta x := x_k - x_{k-1}$ and $\Delta z := z_k - z_{k-1}$. By the updating strategy (2.22), we have

$$\Delta x_{\mathcal{A}_k} = [x_k - x_{k-1}]_{\mathcal{A}_k} \le 0, \tag{2.24a}$$

$$\Delta z_{\mathcal{I}_k} = [z_k - z_{k-1}]_{\mathcal{I}_k} \ge 0. \tag{2.24b}$$

Monotonicity of x: Notice from Algorithm 4 that the KKT equation (2.18a) holds for all iterations. In particular, in computing (x_k, z_k) from (x_{k-1}, z_{k-1}) we conclude that

$$\begin{pmatrix} H_{\mathcal{I}_k\mathcal{I}_k} & H_{\mathcal{I}_k\mathcal{A}_k} \\ H_{\mathcal{A}_k\mathcal{I}_k} & H_{\mathcal{A}_k\mathcal{A}_k} \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{I}_k} \\ \Delta x_{\mathcal{A}_k} \end{pmatrix} + \begin{pmatrix} \Delta z_{\mathcal{I}_k} \\ \Delta z_{\mathcal{A}_k} \end{pmatrix} = 0, \tag{2.25}$$

from which we obtain

$$\Delta x_{\mathcal{I}_k} = -H^{-1}_{\mathcal{I}_k\mathcal{I}_k} H_{\mathcal{I}_k\mathcal{A}_k} \Delta x_{\mathcal{A}_k} - H^{-1}_{\mathcal{I}_k\mathcal{I}_k} \Delta z_{\mathcal{I}_k}. \tag{2.26}$$

Since H is an M-matrix, we have $H^{-1}_{\mathcal{I}_k\mathcal{I}_k} \geq 0$ and $H^{-1}_{\mathcal{I}_k\mathcal{I}_k} H_{\mathcal{I}_k\mathcal{A}_k} \leq 0$. Combining additionally (2.24) and (2.26), this yields immediately that $\Delta x_{\mathcal{I}_k} \leq 0$. Therefore, $x_{k+1} \leq x_k$ for $k \geq 0$. Next, we prove that $x_* \leq x_{k+1}$. Denote an arbitrary partition as $(\mathcal{A}, \mathcal{I})$ and its corresponding solution as (x, z). It is easily seen that

$$[x_* - x]_{\mathcal{A}} = [x_* - u]_{\mathcal{A}} \leq 0.$$

From (2.26), we also obtain that

$$\begin{aligned} [x_* - x]_{\mathcal{I}} &= -[H_{\mathcal{I}\mathcal{I}}]^{-1} H_{\mathcal{I}\mathcal{A}} [x^* - x]_{\mathcal{A}} - [H_{\mathcal{I}\mathcal{I}}]^{-1} [z^* - z]_{\mathcal{I}} \\ &= -[H_{\mathcal{I}\mathcal{I}}]^{-1} H_{\mathcal{I}\mathcal{A}} [x^* - u]_{\mathcal{A}} - [H_{\mathcal{I}\mathcal{I}}]^{-1} [z^* - 0]_{\mathcal{I}} \\ &\leq 0, \end{aligned}$$

thus completing the proof of (2.23a).

Feasibility of x: Next we show that x becomes feasible, i.e. $x \leq u$, after the first iteration. Denote the set of variables of x_0 that violate their upper bounds as $\mathcal{V}_0 := \{i \in \mathcal{N} : [x_0]_i > u_i\}$. For any index $i \in \mathcal{V}_0$, we must have $i \in \mathcal{I}_0$, which by the updating strategy (2.22) means that we have $i \in \mathcal{A}_1$ and $[x_1]_i = u_i$. Whereas for any $i \in \mathcal{N}\backslash\mathcal{V}_0$, from (2.23b) we conclude that $[x_1]_i \leq [x_0]_i \leq u_i$. In summary, we have shown that $x_1 \leq u$ and by (2.23a) it follows that (2.23b) must hold as well.

Convergence: We prove the convergence of Algorithm 4 by showing that z becomes feasible, i.e. $z \geq 0$ after a finite number of iterations. Suppose there exists an index i that

$[z_k]_i < 0$ for some iteration $k \geq 0$, then necessarily $i \in \mathcal{A}_k \cap \mathcal{I}_{k+1}$ and thus $[z_{k+1}]_i = 0$ becomes feasible. By the feasibility of x and the updating strategy (2.22), i remains in $\mathcal{I}$ thereafter. Therefore, z will become feasible for all indices after some iterations. By (2.23b), (x, z) will be feasible, i.e. (2.18d) will be satisfied after a finite number of iterations. Additionally since it is guaranteed that the subspace minimizer (x_k, z_k) satisfies (2.18a)-(2.18c) for all $k \geq 0$, Algorithm 4 reaches optimality once (x, z) becomes feasible. $\square$

Notice that an M-matrix, call it M, is not computationally stable in the sense that an arbitrary perturbation ε may result in a non-M-matrix $M + \varepsilon$. The following theorem shows that Algorithm 4 preserves global convergence in the presence of small perturbations.

Theorem 2.1.2. *([69, Theorem 3.4]) Assume that $H = M + K$ with M an M-matrix and with K an $n \times n$ matrix. Assume again that $\ell = -\infty$. Then if $\|K\|_1$ is sufficiently small, Algorithm 4 is well-defined and converges to the optimal solution from any arbitrary initial partition $(\mathcal{A}_0^u, \mathcal{I}_0)$.*

We briefly compare Algorithm 4 with the GP methods. Both algorithms in one iteration can make dramatic changes of the active-set. The computation of Algorithm 4 mainly involves solving the linear equations (2.20) which, if solved exactly, is usually more expensive than solving a BQP subproblem inexactly as in GP methods. Algorithm 4 usually converges faster to the optimal solution than Algorithm 3 which we attribute to its replication of the semi-smooth Newton steps [69]. Moreover, the design of Algorithm 4 requires no line search or trust region techniques, and thus are easier to implement.

We conclude that the global convergence guarantees provided above do not hold for general convex BQPs as cycles may exist. We illustrate this potential cycling behavior through the following simple problem.

Example 2.1.3 (An example that Algorithm 4 fails to converge)**.**

$$H = \begin{pmatrix} 4 & 5 & -5 \\ 5 & 9 & -5 \\ -5 & -5 & 7 \end{pmatrix}, c = \begin{pmatrix} 2 \\ 1 \\ -3 \end{pmatrix}, \ell = \begin{pmatrix} -\infty \\ -\infty \\ -\infty \end{pmatrix}, \text{ and } u = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \tag{2.27}$$

Assume the initial partition of index sets is

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

We show in Table 2.1 that Algorithm 4 cycles:

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

Table 2.1: Illustration that Algorithm 4 may cause failure on a strictly-convex BQP.

Table 2.1 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, there are 8 possible initial index sets for this problem and 6 of them will lead to failure as illustrated in the following figure.

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 [42, Proposition 4.1].

[1]Figure from [73].

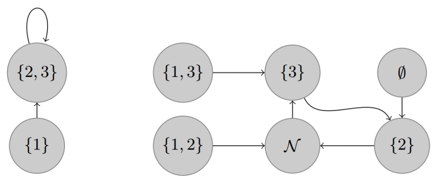


Figure 2.2: Active-set transition graph of applying Algorithm 4 on (2.27) [1].

Despite the fact that Algorithm 4 fails to solve the strictly-convex BQP (2.27), its practical performance on randomly generated problems is impressive and its ability to adapt the active set rapidly is attractive for large-scale problems. A focus of our work will be to enhance the ability of Algorithm 4 to solve strictly-convex BQPs as well as more general QPs without sacrificing overall efficiency.

2.2 Relevant Algorithms

We summarize in this section several relevant algorithms. Parametric active-set methods exemplify the successful application of active-set methods on a series of closely related QPs. Augmented Lagrangian (AL) methods and alternating direction methods multipliers (ADMM), just as PDAS, are primal-dual methods but are designed to solve more general NLPs and recently gained favor for their ability to solve large-scale problems. In practice, AL and ADMM are rarely used to solve convex QPs but more general NLPs. We summarize these methods merely due to the fact that, like PDAS methods, they allow aggressive updates in the active-set estimate.

2.2.1 Parametric Active-Set Methods

Parametric active-set methods (PASM) trace back to the work of Best [11] on parametric quadratic programming and is more recently applied in model predictive control (MPC) by Ferreau et al. [101]. Successful implementation of PASM is included in [37] and applications in [35, 38]. More recently, relevant analysis of invariant optimal active-set that involves more than one parameter [60, 59] is shown to be a powerful tool for multiobjective optimization [103]. We will describe the PASM following the framework of [101]. The key idea of PASM for solving (1.4) is to follow a homotopy path where each point corresponds to the optimal solution of a QP. The homotopy could be viewed as morphing a QP with known optimal solution to a QP to be solved. One salient feature of PASM is that no Phase-I, i.e., the procedure to obtain an initial feasible solution, is required.

Throughout this section, suppose that H is positive semi-definite. Let the homotopy be parameterized by $\tau \in [0,1]$, in PASM a series of QPs parameterized by τ would be solved as

$$\begin{aligned} \min_{x(\tau)\in\mathbb{R}^n} \ & \frac{1}{2}x(\tau)Hx(\tau) + c(\tau)^T x(\tau) \\ \text{s.t. } & \ell(\tau) \le Fx(\tau) \le u(\tau) \end{aligned} \tag{2.28}$$

where c, ℓ, and u are continuous functions of τ. Note that we have rewritten QP of (1.4) in the form of (2.28) such that $F \in \mathbb{R}^{(m+n)\times n}$ contains coefficients of both equality and inequality constraints of (1.4). For each τ, the optimal primal and dual solution pair is also parameterized as $(x(\tau), z(\tau))$. Suppose further that $c \in \mathcal{H}^n, \ell \in \mathcal{H}^{m+n}$, and $u \in \mathcal{H}^{m+n}$ are affine functions defined by

$$\mathcal{H}^k = \big\{ f : [0,1] \mapsto \mathbb{R}^k \mid f(\tau) = (1-\tau)f(0) + \tau f(1), \tau \in [0,1] \big\}.$$

It follows from [11] that the solution $(x(\tau), z(\tau))$ of (2.28) is piecewise linear but not necessarily continuous on τ. On each line segment, the active set of the optimal solution is constant. PSAM follows the solution path parameterized by τ and jumps from one line segment to another until the QP of interest is solved. In the design of most parametric active-set algorithms, $\tau = 0$ usually corresponds to a QP whose optimal solution is known or otherwise is easily solvable and $\tau = 1$ corresponds to the QP to be solved. For example, one may simply choose $\ell(0) = -M\boldsymbol{e}$ and $u(0) = M\boldsymbol{e}$ where M is a sufficiently large number such that the initial working set is empty and the optimal solution is obtained by solving a linear system.

The typical parametric active-set method is summarized in Algorithm 5. We discuss the details of several key steps of Algorithm 5 below.

Step 3: Computation of step direction. Let $F_{\mathcal{W}}$ represent the rows of F with index

Algorithm 5 Parametric active-set method for convex QP

```
1: Input an initial $c(0), \ell(0), u(0)$, and the optimal solution $(x(0), z(0))$ with working set $\mathcal{W}$.
2: while $\tau < 1$ do
3:     Compute step direction $\Delta s = (\Delta x, \Delta z)$ with current working set $\mathcal{W}$.
4:     Determine maximum step $\Delta\tau$.
5:     if $\Delta\tau \geq 1 - \Delta\tau$ then return $x(1) = x(\tau) + (1-\tau)\Delta x$, and $z(1) = z(\tau) + (1-\tau)\Delta z$.
6:     Set $\tau^+ := \tau + \Delta\tau$, $x(\tau^+) = x(\tau) + \Delta\tau\Delta x$, $z(1) = z(\tau) + (1-\tau)\Delta z$, and $\mathcal{W}^+ = \mathcal{W}$.
7:     if constraint $i$ is blocking constraint then
8:         Set $\mathcal{W}^+ = \mathcal{W}^+ \cup \{i\}$.
9:         Linear independence test for $\mathcal{W}^+$.
10:        if linear dependent then
11:            Try to find exchange index $k$.
12:            if not possible then return infeasible.
13:            Adjust dual variable $z(\tau^+)$.
14:            Set $\mathcal{W}^+ = \mathcal{W}^+ \backslash \{k\}$.
15:        end if
16:    else if $i$-th dual variable is blocking then
17:        Set $\mathcal{W}^+ = \mathcal{W}^+ \backslash \{i\}$.
18:        Test for curvature of $H$ on new working set $\mathcal{W}^+$.
19:        if nonpositive curvature then
20:            Try to find exchange index $k$.
21:            if not possible then return unbounded.
22:            Adjust primal variables $x(\tau^+)$.
23:            Set $\mathcal{W}^+ = \mathcal{W}^+ \cup \{k\}$.
24:        end if
25:    end if
26:    Set $\tau = \tau^+$ and $\mathcal{W} = \mathcal{W}^+$.
27:    Possibly update matrix decomposition.
28: end while
```

belonging to $\mathcal{W}$. The step direction is obtained by solving

$$\begin{pmatrix} H & F_{\mathcal{W}}^T \\ F_{\mathcal{W}} & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ -\Delta z_{\mathcal{W}} \end{pmatrix} = \begin{pmatrix} -(c(1) - c(\tau)) \\ F_{\mathcal{W}}(1) - F_{\mathcal{W}}(\tau) \end{pmatrix}, \tag{2.29}$$

and setting $\Delta z_{\overline{\mathcal{W}}} = 0$.

Step 4: Determination of step length. As in simplex methods for linear optimization, the step length $\Delta\tau$ is determined via a *ratio test*. For the sake of brevity, we only describe the idea of the ratio test and refer interested reader to [11, 101] for details. The primal-dual solution pair $(x(\tau), z(\tau))$ is updated in the direction $(\Delta x, \Delta z)$ until an inactive constraint becomes active (a block constraint is met) or a dual variable in the working set becomes 0 (a dual variable is blocking) whichever happens earlier.

Step 9: Linear independence test. Deciding whether adding i to $\mathcal{W}$ will lead to linear dependence is done by solving the following equation

$$\begin{pmatrix} H & F_{\mathcal{W}}^T \\ F_{\mathcal{W}} & 0 \end{pmatrix} \begin{pmatrix} s \\ \xi_{\mathcal{W}} \end{pmatrix} = \begin{pmatrix} F_i \\ 0 \end{pmatrix}. \tag{2.30}$$

It then becomes clear that F_i is linearly dependent on $F_{\mathcal{W}}$ if and only if $s = 0$ is the solution of (2.30).

Step 11: Determination of exchange index. Since $s = 0$, from (2.30) we have $F_i = \sum_{j\in\mathcal{W}} \xi_j F_j$. Suppose $\lambda > 0$, adding $\lambda(\sum_{j\in\mathcal{W}} \xi_j F_j - F_i)$ to the KKT equation of problem (2.28) would yield

$$Hx(\tau^+) + c(\tau^+) = \sum_{j\in\mathcal{W}} z_j(\tau^+)F_j^T + \lambda(\sum_{j\in\mathcal{W}} \xi_j F_j - F_i) = -\lambda F_i + \sum_{j\in\mathcal{W}} (z_j(\tau^+) + \lambda\xi_j F_j).$$

Hence the index k could be selected by another ratio test by selecting the maximum λ that keeps the sign of $z_i(\tau^+)$ correct.

Step 13: Jump in dual variables. The dual variable is simply updated as

$$\tilde{z}_j = \begin{cases} \lambda \text{ or } -\lambda & \text{for } j = i, \\ z_j(\tau^+) + \lambda\xi_j \text{ or } z_j(\tau^+) - \lambda\xi_j & \text{for } j \notin \mathcal{W}, \end{cases}$$

and set $z(\tau^+) := \tilde{z}$. Note that the sign of the change of $\tilde{z}_j$ depends on which of $\mathcal{A}^\ell$ and $\mathcal{A}^u$ that j belongs to.

Step 18: Curvature test. Removing index i from $\mathcal{W}$ might lead to zero curvature on the null space of the new working set, which lead further to the singularity of the linear system to be solved in Step 3. This could be done via solving the following linear system

$$\begin{pmatrix} H & F_{\mathcal{W}}^T \\ F_{\mathcal{W}} & 0 \end{pmatrix} \begin{pmatrix} s \\ \xi_{\mathcal{W}} \end{pmatrix} = \begin{pmatrix} 0 \\ -[e_i]_{\mathcal{W}} \end{pmatrix}, \tag{2.31}$$

where e_i represent the i-th column of the $(m+n)$-by-$(m+n)$ identify matrix. H is singular in the null space of the new working set if and only if $\xi = 0$.

Step 20: Determination of exchange index. Since (2.31) yields

$$Hs = 0, F_i s = -1, \text{ and } F_{\mathcal{W}^+} s = 0.$$

It turns out that s is a non-increase direction and we can find the maximum step length σ such that $x(\tau^+) + \sigma s$ is feasible.

Step 22: Jump in primal variables. The primal variable could be simply updated by $x(\tau^+) = x(\tau^+) + \sigma s$.

Step 27: Update matrix decomposition. Notice that (2.29), (2.30), and (2.31) have identical coefficient matrix, of which the factorization could be reused. When the working set changes by addition, removal, or substitution of constraints, the factorization could be updated relatively cheaply [11, 101] and alternatively iterative methods would be applicable when the matrix is sparse.

2.2.2 Augmented Lagrangian Methods

Augmented Lagrangian (AL) methods was originally proposed to solve constrained nonlinear optimization [68, 102] and recently gained favor for their ability in solving large-scale nonlinear problems. One big advantage of AL methods is that they usually possess fast local convergence guarantees under relatively weak assumptions. We briefly review a classic AL method in this section and an alternative in §2.2.3. Consider the general nonlinear optimization problem

$$\begin{aligned} &\min_x \; f(x) \\ &\text{s.t. } c(x) = 0 \\ &\qquad \ell \le x \le u, \end{aligned} \tag{2.32}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ and $g : \mathbb{R}^m \mapsto \mathbb{R}$ are twice continuously differentiable. Let $\rho > 0$ be an arbitrary positive number, the augmented Lagrange of (2.32) is defined as

$$L_\rho(x, y) = f(x) + y^T c(x) + \frac{\rho}{2}\|c(x)\|_2^2.$$

The basic AL iteration involves two steps:

$$x^{k+1} := \underset{x}{\operatorname{argmin}}\ L_\rho(x, y^k) \text{ subject to } \ell \le x \le u; \tag{2.33a}$$

$$y^{k+1} := y^k + \rho c(x^{k+1}). \tag{2.33b}$$

We describe how problem (1.4) could be solved via repeated application of (2.33). For a convex QP (1.4), Step (2.33a) equivalents solving a bound-constrained QP

$$\underset{x}{\operatorname{argmin}}\ \frac{1}{2}x^T Hx + c^T x + c(x)^T y^k + \frac{\rho}{2}x^T A^T Ax \text{ subject to } \ell \le x \le u,$$

which could be solved efficiently for certain structured problems. Furthermore, the bound-constrained QP subproblem does not need to be solved exactly [34, 28] and ρ could be dynamically updated [28] to speed up the performance of AL methods.

2.2.3 Alternating Direction Methods of Multipliers

The Alternating Direction Methods of Multipliers (ADMM) could be viewed as Augmented Lagrangian Methods with inexact subproblem solves and in particular by applying one single Gauss-Seidel pass. A nice survey of ADMM for large-scale nonlinear optimization is seen in [17]. In this section we focus on applying ADMM to solve (1.4) since ADMM somehow can also make aggressive update on the active set estimate.

ADMM is proposed to solve convex optimization problems of the form

$$\begin{aligned} \min_{x\in\mathbb{R}^n, y\in\mathbb{R}^m}\ & f(x) + g(y) \\ \text{s.t. } & Ax + By = c, \end{aligned} \tag{2.34}$$

where $A \in \mathbb{R}^{p\times n}$, $B \in \mathbb{R}^{p\times m}$, and $c \in \mathbb{R}^p$. The convergence assumptions of ADMM only require f and g to be convex. Let $\rho > 0$ be an arbitrary positive number, the augmented Lagrangian of (2.34) is defined as

$$L_\rho(x, y, z) = f(x) + g(y) + z^T(Ax + By - c) + (\frac{\rho}{2})\|Ax + By - z\|_2^2.$$

A basic ADMM iteration consists of the following steps

$$x^{k+1} := \underset{x}{\operatorname{argmin}}\ L_\rho(x, y^k, z^k), \tag{2.35a}$$

$$y^{k+1} := \underset{y}{\operatorname{argmin}}\ L_\rho(x^{k+1}, y, z^k), \tag{2.35b}$$

$$z^{k+1} := z^k + \rho(Ax^{k+1} + By^{k+1} - c). \tag{2.35c}$$

Under relatively mild assumptions (e.g., L_0 has a saddle point) [17], applying ADMM iterations on problem (2.34) converges to the optimal solution.

Now we confine ourselves to the application of ADMM to the QP problem (1.4), which can be formulated as

$$\begin{aligned} &\min_{x,y}\ f(x) + g(y) \\ &\text{s.t. } x - y = 0, \end{aligned}$$

where $f(x) = \frac{1}{2}x^T Hx + c^T x$, $\text{dom} f = \{x | Ax = b\}$, and g is the indicator function $\Omega(y)$ of the bounds $\ell \leq y \leq u$.

Next, we illustrate how each of the steps of (2.35) is carried out. Starting from Step (2.35a), we have

$$\underset{x}{\operatorname{argmin}} L_\rho(x, y^k, z^k) = \underset{x:Ax=b}{\operatorname{argmin}}\ \frac{1}{2}x^T Hx + c^T x + \Omega(y^k) + (x - y^k)^T z^k + (\frac{\rho}{2})\|x - y^k\|_2^2,$$

which equivalents solving an equality constrained QP or a linear system. In solving Step (2.35b), by dismissing the constants of quadratic terms of x we have

$$\underset{y}{\operatorname{argmin}} L_\rho(x^{k+1}, y, z^k) = \underset{y}{\operatorname{argmin}}\ \Omega(y) + (x^{k+1} - y)^T z^k + (\frac{\rho}{2})\|x^{k+1} - y\|_2^2,$$

which turns out to be separable and have a closed-form solution

$$y^{k+1} = \text{Median}(\ell, x^{k+1} - z^k/\rho, u),$$

where “Median” represents the element-wise median. Notice that in taking the element-wise median operations, multiple entries of y might be added or removed from the active set. The computation involved in Step (2.35c) is trivial.

The biggest advantage of ADMM is that for some problems the steps of (2.35) might be carried out in parallel thus greatly expand user's ability to solve huge problems. For QPs (1.4), ADMM can only achieve local linear convergence rate [66, 14], this is in contrast to PDAS, which is super-linear [69]. Therefore, ADMM might not be well-suited when warm-start or highly accurate solutions are desired.

Chapter 3

Globalization of PDAS for Convex QP

The PDAS method described in §2.1.1, despite being extremely efficient, is only guaranteed to work for certain classes of convex BQPs. The most distinctive feature of the method is its ability to make rapid updates to the estimate of the optimal active-set, which may make the method particularly favorable when solving large-scale problems. Indeed, this feature has sparked numerous efforts to enhance the method to solve more general convex BQPs [73, 82, 21] and LCPs [78]. In order to guarantee convergence, most of these efforts involve the incorporation of safeguards to avoid cycling. However, additional challenges besides cycling need to be addressed. It remains a challenge to design an efficient and globally convergent PDAS algorithmic framework for solving general convex QPs.

In this chapter, we propose enhancements to the PDAS framework of Algorithm 4 so that it will be applicable for solving generally-constrained strictly convex QPs. To do this, we address two main issues that might occur when applying a PDAS method for solving such problems. The first issue is that the PDAS iterates may encounter partitions at which a subspace minimizer is not well-defined. Another issue, as illustrated in the convex BQP in Example 2.1.3, is that a straightforward PDAS method might cycle.

Our proposed enhancements address these issues in the following ways. First, the issue of ill-defined subspace minimizers may arise when the subproblem associated with the partition is infeasible. This usually happens when the active-set estimate contains so many indices that fixing the values of the corresponding variables leads to a set of equality constraints that are not satisfiable. In such a situation, no feasible solution exists and a transformation of the partition is necessary to make further progress. To handle this issue, we propose a procedure that detects feasibility of a subproblem by solving a linear optimization problem for each newly obtained active-set estimate. Whenever an infeasible subproblem is detected, our enhancement adaptively expands the search space by shrinking the active-set estimate until a feasible subproblem is obtained.

We address the latter issue by proposing novel safeguards that attempt to preemptively avoid cycling of the algorithm iterates. Since cycling is typically caused by a few indices switching back and forth among different index sets, we introduce an uncertain set to house indices that display such erratic behavior. Then, in the subspace minimization phase, the bounds for variables corresponding to the uncertain set are enforced explicitly, resulting in a BQP of reduced size. The uncertain set is updated dynamically and deliberating maintained to have a small size, which results in only modest additional costs in the algorithm. Two techniques for updating the uncertain set are proposed: one that monitors changes in the membership of each index and one that monitors progress toward reducing the overall KKT error. Global convergence of our framework using either strategy is proved and the practical performance of the resulting approaches are demonstrated through extensive numerical tests.

This chapter is organized as follows. In §3.1, 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 strictly convex QPs. In §3.2, we discuss two implementation variants of the framework and their relationship to the method in [69]. In §3.3, we describe the details of our implementation and then present numerical experiments in §3.4. These experiments illustrate that an implementation of our framework is efficient when employed to strictly convex QPs, at least those with many degrees of freedom relative to n. Finally, in §3.5 we summarize our findings and comment on additional advantages of our framework, such as the potential incorporation of inexact reduced subproblem solves.

3.1 Algorithmic Framework

In this section, we motivate, describe, and prove a global convergence result for our framework for solving the strictly convex QP (1.4). Throughout this chapter we assume that $H \in \mathbb{R}^{n\times n}$ is symmetric and positive definite. Additionally, 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 (1.4) 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 (1.4), which is to say that the value of $\text{KKT}(x_*, y_*, z_*^\ell, z_*^u)$ defined in (2.3) vanishes. (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 (3.6) later on.) On the other hand, if problem (1.4) 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., [45]. For simplicity in the remainder of our algorithmic development and analysis, we ignore the possibility of having an infeasible instance of problem (1.4), but note that we have implemented such an infeasibility detection strategy in our implementation described in §3.3.

The iterates of our framework constitute a sequence of index sets $\{(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)\}_{k\geq 0}$, 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 3.1.2 (on page 39) and the results in §3.2, our use of $\mathcal{U}_k$ allows for global convergence guarantees for our framework, while the numerical results in §3.4 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

$$A_{\mathcal{M},\mathcal{I}} x_\mathcal{I} + A_{\mathcal{M},\mathcal{U}} x_\mathcal{U} = b - A_{\mathcal{M},\mathcal{A}^\ell} \ell_{\mathcal{A}^\ell} - A_{\mathcal{M},\mathcal{A}^u} u_{\mathcal{A}^u} \quad \text{and} \quad \ell_\mathcal{U} \leq x_\mathcal{U} \leq u_\mathcal{U}. \tag{3.1}$$

Algorithm 6, below, is employed at the beginning of each iteration of our framework in order to transform a given iterate $(\mathcal{A}^\ell_k, \mathcal{A}^u_k, \mathcal{I}_k, \mathcal{U}_k)$ 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 6; we leave such details until §3.3 where the approach employed in our implementation is described.) Note that if $m = 0$, then (3.1) reduces to $\ell_\mathcal{U} \leq x_\mathcal{U} \leq u_\mathcal{U}$. In such cases, preprocessing the data for problem (1.4) (i.e., to ensure $\ell < u$) has guaranteed that each iterate is a feasible partition, so running Algorithm 6 is unnecessary. Otherwise, if $m > 0$, then Algorithm 6 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 (1.4) implies that (3.1) is satisfiable.

Algorithm 6 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 (3.1) 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 7 to compute primal-dual variable values corresponding to the current index sets. The procedure computes the primal-dual variables by minimizing the objective of (1.4) over subsets of the original variables and constraints, where we have already ensured via Algorithm 6 that the reduced problem (3.3) is feasible. Again, we drop the iteration number index in Algorithm 7 as it is inconsequential in this subroutine.

Algorithm 7 Subspace minimization (SM)

```
1: Input $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U})$ such that (3.1) is satisfiable (i.e., such that (3.3) is feasible).
2: Set
       $$x_{\mathcal{A}^\ell} \leftarrow \ell_{\mathcal{A}^\ell}, \quad x_{\mathcal{A}^u} \leftarrow u_{\mathcal{A}^u}, \quad z^\ell_{\mathcal{I}\cup\mathcal{A}^u} \leftarrow 0, \text{ and } z^u_{\mathcal{I}\cup\mathcal{A}^\ell} \leftarrow 0. \tag{3.2}$$
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
       $$\begin{aligned} \underset{x_\mathcal{F}\in\mathbb{R}^{|\mathcal{F}|}}{\text{minimize}} \quad & \tfrac{1}{2}x_\mathcal{F}^T H_{\mathcal{F},\mathcal{F}} x_\mathcal{F} + x_\mathcal{F}^T(c_\mathcal{F} + H_{\mathcal{F},\mathcal{A}} x_\mathcal{A}) \\ \text{subject to} \quad & A_{\mathcal{M},\mathcal{F}} x_\mathcal{F} = b - A_{\mathcal{M},\mathcal{A}} x_\mathcal{A}, \;\; \ell_\mathcal{U} \le x_\mathcal{U} \le u_\mathcal{U}. \end{aligned} \tag{3.3}$$
4: Set
       $$z^\ell_{\mathcal{A}^\ell} \leftarrow [Hx + c - A^T y]_{\mathcal{A}^\ell} \text{ and } z^u_{\mathcal{A}^u} \leftarrow -[Hx + c - A^T y]_{\mathcal{A}^u}. \tag{3.4}$$
5: Return $(x, y, z^\ell, z^u) =: \text{SM}(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U})$.
```

The steps of Algorithm 7 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 (3.3) reduces to the solution of

$$\begin{pmatrix} H_{\mathcal{I},\mathcal{I}} & A^T_{\mathcal{M},\mathcal{I}} \\ A_{\mathcal{M},\mathcal{I}} & 0 \end{pmatrix} \begin{pmatrix} x_\mathcal{I} \\ -y \end{pmatrix} = - \begin{pmatrix} c_\mathcal{I} + H_{\mathcal{I},\mathcal{A}} x_\mathcal{A} \\ A_{\mathcal{M},\mathcal{A}} x_\mathcal{A} - b \end{pmatrix}. \tag{3.5}$$

This observation is critical as it shows that, whenever $\mathcal{U}_k = \emptyset$ in our framework, the computational cost of Algorithm 7 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 7. (Recall that we require the vector norm in equation (3.6) to be that used in the definition of the KKT function defined in equation (2.3).)

Lemma 3.1.1. *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*

$$r(x, y, z^\ell, z^u) := \left\| \begin{pmatrix} \min\{z^\ell_{\mathcal{A}^\ell}, 0\} \\ \min\{z^u_{\mathcal{A}^u}, 0\} \\ \min\{0, [x-\ell]_{\mathcal{I}}, [u-x]_{\mathcal{I}}\} \end{pmatrix} \right\|. \tag{3.6}$$

Proof. The proof follows by straightforward comparison of $\mathrm{KKT}(x, y, z^\ell, z^u)$ with (3.2), the optimality conditions of (3.3), and (3.4). 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 3.1.1 that if the input $(\mathcal{A}^\ell, \mathcal{A}^u, \mathcal{I}, \mathcal{U})$ to Algorithm 7 yields (x, y, z^ℓ, z^u) with $\ell_{\mathcal{I}} \leq x_{\mathcal{I}} \leq u_{\mathcal{I}}$, $z^\ell_{\mathcal{A}^\ell} \geq 0$, and $z^u_{\mathcal{A}^u} \geq 0$, then (x, y, z^ℓ, z^u) is the solution to (1.4). This follows as the procedure in Algorithm 7 ensures that the resulting primal-dual vector satisfies the first two blocks of equations in (2.3) as well as complementarity of the primal and dual variables. The only conditions in (2.3) that it does not guarantee for each partition are primal and dual variable bounds.

We now state our algorithmic framework in Algorithm 8.

Although we have yet to state specific strategies for updating the index sets in Step 6 of Algorithm 8, we can prove that it terminates and returns the optimal solution of (1.4) as long as, for some $k \geq 0$, we have

$$\mathcal{A}^\ell_k \subseteq \mathcal{A}^\ell_*, \quad \mathcal{A}^u_k \subseteq \mathcal{A}^u_*, \quad \text{and} \quad \mathcal{I}_k \subseteq \mathcal{I}_*. \tag{3.7}$$

Algorithm 8 Primal-dual active-set framework (PDAS)

```
1: Input $(\mathcal{A}_0^\ell, \mathcal{A}_0^u, \mathcal{I}_0, \mathcal{U}_0)$ and initialize $k \leftarrow 0$.
2: loop
3:     Set $(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k) \leftarrow \text{Feas}(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$ by Algorithm 6.
4:     Set $(x_k, y_k, z_k^\ell, z_k^u) \leftarrow \text{SM}(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$ by Algorithm 7.
5:     If $r(x_k, y_k, z_k^\ell, z_k^u) = 0$, then break.
6:     Choose $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$, then set $k \leftarrow k + 1$.
7: end loop
8: Return $(x_k, y_k, z_k^\ell, z_k^u)$.
```

In other words, Algorithm 8 produces $(x_*, y_*, z_*^\ell, z_*^u)$ satisfying (2.3) if, for some iteration number $k \geq 0$, the algorithm generates subsets of the optimal index sets.

Theorem 3.1.2. *If problem* (1.4) *is feasible and the* k*th iterate of Algorithm 8 satisfies* (3.7)*, then* $r(x_k, y_k, z_k^\ell, z_k^u) = 0$ *and* $(x_k, y_k, z_k^\ell, z_k^u)$ *solves problem* (1.4)*.*

Proof. Our strategy of proof is as follows. First, we will show that if (3.7) 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 6 has no effect on such an iterate in Step 3 of Algorithm 8. Second, we will show that $(x_*, y_*, z_*^\ell, z_*^u)$ satisfies (3.2)–(3.4), where in the case of (3.3) 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 (3.2)–(3.4), 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 (3.7) holds, then Algorithm 6 will have no effect on the partition, we will show that—due to (3.7)—the point $(x_*, y_*, z_*^\ell, z_*^u)$ satisfies (3.1). Since this will imply that (3.1) is satisfiable, it will follow that the **while** loop in Algorithm 6 will not be entered, and hence the initial partition given to Algorithm 6 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 7). It follows from the KKT conditions (2.3) that $Ax_* = b$, which with condition (3.7) implies

$$\begin{aligned} A_{\mathcal{M},\mathcal{F}_k}[x_*]_{\mathcal{F}_k} &= b - A_{\mathcal{M},\mathcal{A}_k^\ell}[x_*]_{\mathcal{A}_k^\ell} - A_{\mathcal{M},\mathcal{A}_k^u}[x_*]_{\mathcal{A}_k^u} \\ &= b - A_{\mathcal{M},\mathcal{A}_k^\ell}\ell_{\mathcal{A}_k^\ell} - A_{\mathcal{M},\mathcal{A}_k^u}u_{\mathcal{A}_k^u}. \end{aligned} \tag{3.8}$$

In addition, (2.3) implies $\ell \leq x_* \leq u$, with which we find

$$\ell_{\mathcal{U}_k} \leq [x_*]_{\mathcal{U}_k} \leq u_{\mathcal{U}_k}. \tag{3.9}$$

Hence, $(x_*, y_*, z_*^\ell, z_*^u)$ satisfies (3.1), so Algorithm 6 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 (3.2)–(3.4). It follows from (3.7) that

$$[x_*]_{\mathcal{A}_k^\ell} = \ell_{\mathcal{A}_k^\ell}, \quad [x_*]_{\mathcal{A}_k^u} = u_{\mathcal{A}_k^u}, \quad [z_*^\ell]_{\mathcal{I}_k \cup \mathcal{A}_k^u} = 0, \quad \text{and} \quad [z_*^u]_{\mathcal{I}_k \cup \mathcal{A}_k^\ell} = 0, \tag{3.10}$$

so (3.2) is satisfied. It then also follows from (2.3) and (3.10) that

$$\begin{pmatrix} H_{\mathcal{I}_k\mathcal{I}_k} & H_{\mathcal{I}_k\mathcal{U}_k} \\ H_{\mathcal{U}_k\mathcal{I}_k} & H_{\mathcal{U}_k\mathcal{U}_k} \end{pmatrix} \begin{pmatrix} [x_*]_{\mathcal{I}_k} \\ [x_*]_{\mathcal{U}_k} \end{pmatrix} + \begin{pmatrix} H_{\mathcal{I}_k\mathcal{A}_k} \\ H_{\mathcal{U}_k\mathcal{A}_k} \end{pmatrix} [x_*]_{\mathcal{A}_k} + \begin{pmatrix} c_{\mathcal{I}_k} \\ c_{\mathcal{U}_k} \end{pmatrix} - \begin{pmatrix} A^T_{\mathcal{M},\mathcal{I}_k} \\ A^T_{\mathcal{M},\mathcal{U}_k} \end{pmatrix} y_* + \begin{pmatrix} 0 \\ [z_*^u - z_*^\ell]_{\mathcal{U}_k} \end{pmatrix} = 0. \tag{3.11}$$

Furthermore,

$$\text{if } \ i \in \mathcal{U}_k \cap \mathcal{I}_*, \ \text{ then } [x_*]_i \in (l_i, u_i) \text{ and } [z_*^\ell]_i = [z_*^u]_i = 0; \tag{3.12a}$$

$$\text{if } \ i \in \mathcal{U}_k \cap \mathcal{A}_*^\ell, \ \text{ then } [x_*]_i = \ell_i,\ [z_*^\ell]_i \geq 0, \text{ and } [z_*^u]_i = 0; \tag{3.12b}$$

$$\text{if } \ i \in \mathcal{U}_k \cap \mathcal{A}_*^u, \ \text{ then } [x_*]_i = u_i,\ [z_*^\ell]_i = 0, \text{ and } [z_*^u]_i \geq 0. \tag{3.12c}$$

Thus, it follows from (3.8)–(3.12) that $[x_*]_{\mathcal{F}_k}$ is the unique solution to (3.3) with associated dual values $(y_*, [z_*^\ell]_{\mathcal{U}_k}, [z_*^u]_{\mathcal{U}_k})$. Finally, from (2.3) and (3.10) we have

$$[z_*^\ell]_{\mathcal{A}_k^\ell} = \ \ [Hx_* + c - A^T y_* + z_*^u]_{\mathcal{A}_k^\ell} = \ \ [Hx_* + c - A^T y_*]_{\mathcal{A}_k^\ell} \tag{3.13a}$$

$$\text{and} \quad [z_*^u]_{\mathcal{A}_k^u} = -[Hx_* + c - A^T y_* - z_*^\ell]_{\mathcal{A}_k^u} = -[Hx_* + c - A^T y_*]_{\mathcal{A}_k^u}. \tag{3.13b}$$

We may now conclude from (3.8)–(3.13) that $(x_*, y_*, z_*^\ell, z_*^u)$ satisfies (3.2)–(3.4), which are uniquely satisfied by $(x_k, y_k, z_k^\ell, z_k^u)$. Thus, $(x_k, y_k, z_k^\ell, z_k^u) = (x_*, y_*, z_*^\ell, z_*^u)$. $\square$

Note that due to potential degeneracy, condition (3.7) is not necessary for Algorithm 8 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 (3.3). This means that Algorithm 8 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 §3.2.

Now that our algorithmic framework has been established in Algorithm 8, we comment further on comparisons of our approach and a few related methods in the literature. First, the algorithms in [8, 9, 86] have a similar form to Algorithm 8 and employ index set updates that are similar to that in Algorithm 9, 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 (1.4). 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 [63]—which is based on the method proposed in [61]—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 (1.4), but with some primal variables fixed at their bounds and some bound constraints from (1.4) 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 (1.4) differently: DASA uses a proximal point approach to avoid unbounded dual subproblems, whereas PDAS employs an initial infeasibility detection phase and Algorithm 6 to guarantee that (3.3) is feasible.

3.2 Strategies for Updating the Indexing Sets

In this section, we describe several strategies for updating the index sets in Algorithm 8. That is, we describe subroutines to be employed in Step 6 of Algorithm 8 to choose iterate $k+1$. Recall that Theorem 3.1.2 shows that if an update yields (3.7), then Algorithm 8 will terminate with a solution to problem (1.4). We begin by describing an extension of the strategy of Hintermüller, Ito, and Kunisch [69] 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.

3.2.1 Hintermüller, Ito, and Kunisch update

The first strategy we describe, written as Algorithm 9 below, is an extension of the technique used in the method introduced by Hintermüller, Ito, and Kunisch [69]. In particular, if $m = 0$, $\ell_i = -\infty$ for all $i \in \mathcal{N}$, and $\mathcal{U}_0 \leftarrow \emptyset$, then Algorithm 8 with Step 6 employing Algorithm 9 is identical to the algorithm in [69, Section 2].

Algorithm 9 Updating strategy inspired by Hintermüller, Ito, and Kunisch [69]

```
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
```

$$\mathcal{U}_{k+1} \leftarrow \mathcal{U}_k, \tag{3.14a}$$
$$\mathcal{A}_{k+1}^\ell \leftarrow \{i \in (\mathcal{N} \setminus \mathcal{U}_{k+1}) : [x_k]_i < \ell_i, \ \text{ or } \ i \in \mathcal{A}_k^\ell \ \text{ and } \ [z_k^\ell]_i \geq 0\}, \tag{3.14b}$$
$$\mathcal{A}_{k+1}^u \leftarrow \{i \in (\mathcal{N} \setminus \mathcal{U}_{k+1}) : [x_k]_i > u_i, \ \text{ or } \ i \in \mathcal{A}_k^u \ \text{ and } \ [z_k^u]_i \geq 0\}, \tag{3.14c}$$
$$\text{and } \ \mathcal{I}_{k+1} \leftarrow \mathcal{N} \setminus (\mathcal{U}_{k+1} \cup \mathcal{A}_{k+1}^\ell \cup \mathcal{A}_{k+1}^u). \tag{3.14d}$$

```
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 [69, 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 3.2.1. *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 (1.4) has a unique solution (x_*, z_*^ℓ, z_*^u) and Algorithm 8 with Step 6 employing Algorithm 9 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 8 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 [69, Theorem 3.1].

The main disadvantage of the strategy in Algorithm 9 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 3.2.1 makes the assumption that H is a (perturbed) M-matrix, which is restrictive. The three-dimensional Example (2.27) 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.

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.

3.2.2 Update based on monitoring index set changes

The goal of our first new updating strategy for the index sets is to ensure that (3.7) is eventually satisfied. This is desirable as then convergence of the associated iterates is guaranteed by Theorem 3.1.2. The strategy is based on a simple observation: Since there are only a finite number of distinct choices of the index sets, condition (3.7) 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 10. (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 §3.3.) The motivation for the strategy is to mimic Algorithm 9, 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 (3.3), we remark that it is desirable to keep the elements of the sequence $\{|\mathcal{U}_k|\}$ small.)

The following lemma shows that Algorithm 10 guarantees that every iteration will either move a nonempty subset of indices into the uncertain set, or our index counters will increase.

Algorithm 10 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 (3.14).
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 $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.
```

Lemma 3.2.2. *Suppose that problem (1.4) is feasible. If Algorithm 8 does not terminate before or in iteration k, then by employing Algorithm 10 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 (3.1) is satisfiable and as a result of solving (3.3) we obtain $(x_*, y_*, z_*^\ell, z_*^u)$ solving (1.4). Consequently, Algorithm 8 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 10 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, (3.14) will change at least one index from some index set to another.

To derive a contradiction, suppose that $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1}) = (\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$ is yielded by (3.14). By the procedure in Algorithm 7, it follows that

$$[x_k]_{\mathcal{A}_k^\ell} = [\ell]_{\mathcal{A}_k^\ell}, \quad [x_k]_{\mathcal{A}_k^u} = [u]_{\mathcal{A}_k^u}, \quad [z_k^\ell]_{\mathcal{I}_k \cup \mathcal{A}_k^u} = 0, \quad \text{and} \quad [z_k^u]_{\mathcal{I}_k \cup \mathcal{A}_k^\ell} = 0, \tag{3.15}$$

and the optimality conditions for subproblem (3.3) ensure that

$$\min([x_k - \ell]_{\mathcal{U}_k}, [z_k^\ell]_{\mathcal{U}_k}) = \min([u - x_k]_{\mathcal{U}_k}, [z_k^u]_{\mathcal{U}_k}) = 0. \tag{3.16}$$

Algorithm 10 and the assumption that there are no set changes imply that

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

from which we conclude that

$$[\ell]_{\mathcal{I}_k} \leq [x_k]_{\mathcal{I}_k} \leq [u]_{\mathcal{I}_k}. \tag{3.17}$$

Similarly, Algorithm 10 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

$$[z^\ell_k]_{\mathcal{A}^\ell_k} \geq 0 \ \text{ and } \ [z^u_k]_{\mathcal{A}^u_k} \geq 0. \tag{3.18}$$

With the residual function r defined by (3.6), it follows from (3.15)–(3.18) 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 8 does not terminate in iteration k. □

The global convergence of Algorithm 8 with the updating strategy in Algorithm 10 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 §3.4, we rarely find the auxiliary set ever including more than a few indices.

Theorem 3.2.3. *If problem (1.4) is feasible, then Algorithm 8 with Step 6 employing Algorithm 10 solves problem (1.4) in a finite number of iterations.*

Proof. To derive a contradiction, suppose that Algorithm 8 computes infinitely many iterates. Then, by Lemma 3.2.2, 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 3.2.2, this means that (3.1) will be satisfiable, solving (3.3) will yield $(x_*, y_*, z^\ell_*, z^u_*)$ solving (1.4), and the algorithm will terminate, contradicting the supposition that an infinite number of iterations will be performed. □

Table 3.1 shows the behavior of Algorithm 8 on the problem in Example 2.27 given on page 24. By using Algorithm 10 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 3.1: Result of Algorithm 8 employed to solve the problem in Example 2.1 when iterates are updated via Algorithm 10.

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

3.2.3 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 [24, 51, 56, 112, 113]. 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 9, 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 11 can be summarized as follows. First, a trial iterate is chosen as $(\mathcal{A}^\ell_{k+1}, \mathcal{A}^u_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$ via (3.14) and then the Feas operator, i.e., Algorithm 6, is applied to produce a feasible partition. If the resulting index sets yield (through the SM operator, i.e., Algorithm 7) 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

$(\mathcal{A}^{\ell}_{k+1}, \mathcal{A}^{u}_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$. Otherwise, $(\mathcal{A}^{\ell}_{k+1}, \mathcal{A}^{u}_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$ is reset to $(\mathcal{A}^{\ell}_{k}, \mathcal{A}^{u}_{k}, \mathcal{I}_{k}, \mathcal{U}_{k})$ 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 $(\mathcal{A}^{\ell}_{k+1}, \mathcal{A}^{u}_{k+1}, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$ 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 $\mathcal{A}^{\ell}_{k+1}$, $\mathcal{A}^{u}_{k+1}$, 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 11, we have the following lemma.

Lemma 3.2.4. *Suppose that problem (1.4) is feasible. If Algorithm 8 does not terminate before or in iteration k, then by employing Algorithm 11 in Step 6, iteration $k+1$ yields*

$$r(x_{k+1}, y_{k+1}, z^{\ell}_{k+1}, z^{u}_{k+1}) < \max_{j\in\{1,\ldots,p\}} \{r(x_{k+1-j}, y_{k+1-j}, z^{\ell}_{k+1-j}, z^{u}_{k+1-j})\}. \tag{3.19}$$

Proof. Under the assumption that Algorithm 8 has not yet terminated, we have

$$\max_{j\in\{1,\ldots,p\}} \{r(x_{k+1-j}, y_{k+1-j}, z^{\ell}_{k+1-j}, z^{u}_{k+1-j})\} > 0. \tag{3.20}$$

If the condition in Step 5 holds (i.e., (3.19) 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 (3.6)) to the index set $\mathcal{U}_{k+1}$ until a strict reduction has been obtained (i.e., until (3.19) 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

Algorithm 11 Updating strategy based on ensuring KKT residual decrease

```
1: Input $(\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$, $p \geq 1$, and $\{(x_{k+1-j}, y_{k+1-j}, z_{k+1-j}^\ell, z_{k+1-j}^u)\}_{j\in\{1,\dots,p\}}$.
2: Set $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$ by (3.14).
3: Set $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1}) \leftarrow \text{Feas}(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.
4: Set $(x_{k+1}, y_{k+1}, z_{k+1}^\ell, z_{k+1}^u) \leftarrow \text{SM}(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.
5: if $r(x_{k+1}, y_{k+1}, z_{k+1}^\ell, z_{k+1}^u) \geq \max_{j\in\{1,\dots,p\}}\{r(x_{k+1-j}, y_{k+1-j}, z_{k+1-j}^\ell, z_{k+1-j}^u)\}$ then
6:     Reset $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1}) \leftarrow (\mathcal{A}_k^\ell, \mathcal{A}_k^u, \mathcal{I}_k, \mathcal{U}_k)$.
7:     repeat
8:         Choose $\mathcal{S} \leftarrow \mathcal{S}^\ell \cup \mathcal{S}^u \cup \mathcal{S}^\mathcal{I} \neq \emptyset$ with
               $\mathcal{S}^\ell \subseteq \{i \in \mathcal{A}_{k+1}^\ell : z_k^\ell < 0\}$
               $\mathcal{S}^u \subseteq \{i \in \mathcal{A}_{k+1}^u : z_k^u < 0\}$
           and $\mathcal{S}^\mathcal{I} \subseteq \{i \in \mathcal{I}_{k+1} : \min\{[x_k - \ell]_i, [u - x_k]_i\} < 0\}$.
9:         Set $\mathcal{A}_{k+1}^\ell \leftarrow \mathcal{A}_{k+1}^\ell \backslash \mathcal{S}^\ell$, $\mathcal{A}_{k+1}^u \leftarrow \mathcal{A}_{k+1}^u \backslash \mathcal{S}^u$, $\mathcal{I}_{k+1} \leftarrow \mathcal{I}_{k+1} \backslash \mathcal{S}^\mathcal{I}$, and $\mathcal{U}_{k+1} \leftarrow \mathcal{U}_{k+1} \cup \mathcal{S}$.
10:        Set $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1}) \leftarrow \text{Feas}(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.
11:        Set $(x_{k+1}, y_{k+1}, z_{k+1}^\ell, z_{k+1}^u) \leftarrow \text{SM}(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.
12:    until $r(x_{k+1}, y_{k+1}, z_{k+1}^\ell, z_{k+1}^u) < \max_{j\in\{1,\dots,p\}}\{r(x_{k+1-j}, y_{k+1-j}, z_{k+1-j}^\ell, z_{k+1-j}^u)\}$
13: end if
14: (Optional) Choose $\mathcal{T} \leftarrow \mathcal{T}^\ell \cup \mathcal{T}^u \cup \mathcal{T}^\mathcal{I}$ with
        $\mathcal{T}^\ell \subseteq \left\{ i \in \bigcap_{l=k+2-p}^{k+1} \mathcal{U}_l : [x_l]_i = \ell_i \text{ for } l \in \{k+2-p, \dots, k+1\} \right\}$,
        $\mathcal{T}^u \subseteq \left\{ i \in \bigcap_{l=k+2-p}^{k+1} \mathcal{U}_l : [x_l]_i = u_i \text{ for } l \in \{k+2-p, \dots, k+1\} \right\}$,
    and $\mathcal{T}^\mathcal{I} \subseteq \left\{ i \in \bigcap_{l=k+2-p}^{k+1} \mathcal{U}_l : \ell_i < [x_l]_i < u_i \text{ for } l \in \{k+2-p, \dots, k+1\} \right\}$,
    then set $\mathcal{A}_{k+1}^\ell \leftarrow \mathcal{A}_{k+1}^\ell \cup \mathcal{T}^\ell$, $\mathcal{A}_{k+1}^u \leftarrow \mathcal{A}_{k+1}^u \cup \mathcal{T}^u$, $\mathcal{I}_{k+1} \leftarrow \mathcal{I}_{k+1} \cup \mathcal{T}^\mathcal{I}$, and $\mathcal{U}_{k+1} \leftarrow \mathcal{U}_{k+1} \backslash \mathcal{T}$.
15: Return $(\mathcal{A}_{k+1}^\ell, \mathcal{A}_{k+1}^u, \mathcal{I}_{k+1}, \mathcal{U}_{k+1})$.
```

in the vectors defining $r(x_{k+1}, y_{k+1}, z^{\ell}_{k+1}, z^{u}_{k+1})$ and the r values in (3.20). $\square$

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 (3.21)), the KKT error corresponding to the algorithm iterates will eventually vanish.

Theorem 3.2.5. *If problem* (1.4) *is feasible, then Algorithm 8 with Step 6 employing Algorithm 11 solves problem* (1.4) *in a finite number of iterations.*

Proof. The result follows since by Lemma 3.2.4 we have that Algorithm 11 guarantees

$$\left\{ \max_{j\in\{1,\ldots,p+1\}} \{r(x_{k+1-j}, y_{k+1-j}, z^{\ell}_{k+1-j}, z^{u}_{k+1-j})\} \right\} \tag{3.21}$$

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 (3.19), there can be at most p consecutive iterations where the right-hand side of (3.19) 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 $r(x^{\ell}_k, y^{u}_k, z^{\ell}_k, z^{u}_k) = 0$. $\square$

Table 3.2 contains the output from solving the strictly convex QP in Example 2.27 on page 24 using Algorithm 8 with $p = 1$.

Table 3.2: Result of Algorithm 8 employed to solve the problem in Example 2.1 when iterates are updated via Algorithm 11. In the algorithm, the ℓ_∞-norm is used in the definition of the residual function r (recall (3.6)) 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

3.3 Implementation details

In this paper, we have proposed and analyzed two instances of the generic framework provided as Algorithm 8, in addition to an instance representing an extension of the algorithm in [69]. 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 8 paired with the updating strategy in Algorithm 9 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 10 and 11, 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 8. First, Algorithm 6 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 (3.3):

$$\begin{aligned} &\underset{\overline{x}_{\mathcal{F}},\overline{r},\overline{s}}{\text{minimize}} \quad e^T(\overline{r}+\overline{s}) \\ &\text{subject to} \quad A_{\mathcal{M},\mathcal{F}}\overline{x}_{\mathcal{F}} = b - A_{\mathcal{M},\mathcal{A}}x_{\mathcal{A}} + \overline{r} - \overline{s}, \;\; \ell_{\mathcal{U}} \le \overline{x}_{\mathcal{U}} \le u_{\mathcal{U}}, \;\; (\overline{r},\overline{s}) \ge 0, \end{aligned} \tag{3.22}$$

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 7, and $x_{\mathcal{A}}$ is defined as it is set in Step 2 of Algorithm 7. 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 (3.3); 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

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

CPLEX's primal simplex method is applied to solve (3.22). In many cases, (3.22) is solved at this initial point. In general, however, (3.22) is solved via an LP solution method and, when (3.1) 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 6 is motivated by the fact that infeasibility of (3.1) 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 7 is relatively straightforward. Indeed, the only step that requires specification is the method employed to solve subproblem (3.3). If $|\mathcal{U}| = 0$, then the solution of (3.3) is obtained by solving the reduced linear system (3.5); in such cases, we employ Matlab's built-in "\" routine to solve this system. If $|\mathcal{U}| \neq 0$, then problem (3.3) is a generally-constrained QP and we employ the active-set method implemented in the qpOASES package [37].

We now turn to the details of our implementation of our strategies in Algorithms 10 and 11. 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 10, we implement Step 3 by choosing $\mathcal{S}$ as the smallest index in $\{i \in \mathcal{N} : q_k^i = \max_{j\in\mathcal{N}} \ q_k^j\}$, and we implement Step 5 by setting $q_{k+1}^i \leftarrow 0$ for all $i \in \mathcal{N}$. Similarly, in Algorithm 11, we implement Step 8 by choosing $\mathcal{S}$ as the index in $\mathcal{A}_{k+1}^\ell \cup \mathcal{A}_{k+1}^u \cup \mathcal{I}_{k+1}$ corresponding to the element of the vector defining $r(x_{k+1}, y_{k+1}, z_{k+1}^\ell, z_{k+1}^u)$ (recall (3.6)) with the largest absolute value. (If there are more than one such indices in $\mathcal{A}_{k+1}^\ell \cup \mathcal{A}_{k+1}^u \cup \mathcal{I}_{k+1}$, then we choose the smallest such index.) Finally, due to numerical inaccuracies in the SM routine, the idealized conditions in Step 12 of Algorithm 11 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

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

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

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

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.

3.4 Numerical Results

We tested our implementation of Algorithm 8 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 10 was run with $q^{\max} \leftarrow 5$ and Algorithm 11 (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 (3.22)) was set to 10^{-8} and for Algorithm 11 we set $\varepsilon_{\mathcal{A}} \leftarrow 10^{-8}$ and $\varepsilon_{\mathcal{I}} \leftarrow 10^{-2}$.

Hereinafter, we refer to Algorithm 8 paired with the updating strategy in Algorithm 9 simply as "Algorithm 9", and similarly for Algorithms 10 and 11. 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 6, 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 3.3–3.7, 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 3.3; see the caption for that table.

In Tables 3.3–3.6, we present results for Algorithms 9, 10, and 11. 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 7 (μ(#SM)). In Tables 3.3 and 3.4, the number of calls to Algorithm 7 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 3.5 and 3.6 the number of calls to Algorithm 7 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 7. 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 9 maintains $\mathcal{U}_k = \emptyset$ for all k, we omit columns (that would otherwise appear in Table 3.3) corresponding to qpOASES iterations and cardinalities of the uncertain set since these values are all zero.

Table 3.3 shows that Algorithm 9 is generally very efficient, but may not converge. On the other hand, Algorithms 10 and 11 (see Tables 3.4–3.6) solved all problems in

Table 3.3: Results of Algorithm 9 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‡

our experiments, illustrating that their global convergence guarantees are beneficial in practice. Note that in Tables 3.4–3.5, a "μ(Last-$|\mathcal{U}|$)" equal to zero indicates that the algorithm is behaving identically to Algorithm 9. As this occurs often, particularly in Table 3.5, this illustrates that Algorithms 10 and 11 are as efficient as Algorithm 9 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 11 yields some benefits 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 3.5 and 3.6.

As a means of comparison for the results in Tables 3.3–3.6, we present in Table 3.7 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 3.3–3.6 since in many iterations our implementations of Algorithms 9, 10, and 11 solve a linear system directly rather than calling qpOASES. That being said, the results in Table 3.7 illustrate the typical behavior of a classic active-set method with which

Table 3.4: Results of Algorithm 10 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

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 3.3–3.6.

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 6 (μ(#Feas)), the average number of times that a call to Algorithm 6 actually modified the index set partition (μ(#Feas-Mod)), the average total number of simplex pivots (in CPLEX) summed over all calls to Algorithm 6 (μ(#Feas-Pvt)), and the standard deviation of the total number of simplex pivots (σ(#Feas-Pvt)). These results are provided in Tables 3.8–3.10.

Tables 3.8–3.10 illustrate that Algorithms 10 and 11 (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

Table 3.5: Results of Algorithm 11 (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

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 3.8, the consistent values `6.00e+00` and `0.00e+00` for μ(#LS) and σ(#LS), respectively, indicate that Algorithm 10 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 iterations. 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 3.11.

Finally, we test the practical performance of our proposed algorithms on the Maros and Meszaros Convex QP test set [90]. The problems that are not solved within the time limit are not reported. We employ `State` to indicate the state when the algorithm terminates, with `State` being 0 meaning successful solve and -1 meaning the problem violates feasibility tolerance. We set time limit of 48 hours and report the numerical

Table 3.6: Results of Algorithm 11 (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

Table 3.7: 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

results of the solved problems in Table 3.12–3.14. We observe again that for the solved problems PDAS usually can converge in a few iterations.

Table 3.8: Results of Algorithm 10 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-$\|\mathcal{U}\|$)	μ(Avg-$\|\mathcal{U}\|$)	σ(Avg-$\|\mathcal{U}\|$)
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

Table 3.9: Results of Algorithm 11 (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-$\|\mathcal{U}\|$)	μ(Avg-$\|\mathcal{U}\|$)	σ(Avg-$\|\mathcal{U}\|$)
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 3.10: Results for Algorithm 11 (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	

Table 3.11: 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

Table 3.12: Results of Algorithm 10 employed to solve Maros and Meszaros problems.

Name	n	m	State	#Iter	#SM	#Feas	#Feas-Mod	Last-$\lvert\mathcal{U}\rvert$	Avg-$\lvert\mathcal{U}\rvert$	#Feas-Pvt	#QP-Iter	#LS
AUG2D	20200	10000	0	0	1	1	0	0	0	161	0.00e+00	1
AUG3DCQP	3873	1000	0	12	13	13	3	1	5.83e-01	141190	4.54e+03	6
CVXQP1_S	100	50	0	0	1	1	1	0	0	2	0.00e+00	1
CVXQP2_S	100	25	0	10	11	11	3	1	4.00e-01	29	3.00e+01	7
CVXQP3_M	1000	750	0	2	3	3	3	0	0	2849	0.00e+00	3
DUAL1	85	1	0	0	1	1	1	0	0	0	0.00e+00	1
DUAL2	96	1	0	0	1	1	1	0	0	0	0.00e+00	1
GENHS28	10	8	0	0	1	1	0	0	0	0	0.00e+00	1
HS21	2	1	0	2	3	3	0	0	0	0	0.00e+00	3
HS268	5	5	0	0	1	1	0	0	0	0	0.00e+00	1
HS35MOD	3	1	0	2	3	3	0	0	0	0	0.00e+00	3
HS35	3	1	0	7	8	8	0	1	1.43e-01	0	0	7
HS51	5	3	0	0	1	1	0	0	0	0	0.00e+00	1
HS52	5	3	0	0	1	1	0	0	0	0	0.00e+00	1
HS53	5	3	0	0	1	1	0	0	0	0	0.00e+00	1
HUES-MOD	10000	2	0	12	13	13	1	1	5.83e-01	4	21	6
LISWET12	10002	10000	0	0	1	1	0	0	0	0	0.00e+00	1
LISWET4	10002	10000	0	0	1	1	0	0	0	0	0.00e+00	1
MOSARQP1	2500	700	0	249	250	250	182	40	2.02e+01	54072	3.31e+04	6
MOSARQP2	900	600	0	14	15	15	4	2	7.86e-01	1052	2.70e+02	6
PRIMAL2	649	96	0	7	8	8	0	1	2.86e-01	136	3.68e+02	6
PRIMALC1	230	9	0	11	12	12	2	1	5.45e-01	10	3.20e+01	6
PRIMALC8	520	8	0	0	1	1	0	0	0	0	0.00e+00	1

Table 3.13: Results of Algorithm 11 (without step 14) employed to solve Maros and Meszaros problems.

Name	n	m	State	#Iter	#SM	#Feas	#Feas-Mod	Last-$\|\mathcal{U}\|$	Avg-$\|\mathcal{U}\|$	#Feas-Pvt	#QP-Iter	#LS
AUG2D	20200	10000	0	0	1	1	0	0	0	161	0.00e+00	1
CONT-050	2597	2401	-1	1	0	1	1	0	0	112904	0.00e+00	0
CONT-101	10197	10098	-1	1	0	1	1	0	0	135309	0.00e+00	0
CVXQP1_M	1000	500	0	0	1	1	1	0	0	33	0.00e+00	1
CVXQP1_S	100	50	0	0	1	1	1	0	0	2	0.00e+00	1
CVXQP2_S	100	25	0	6	8	8	1	1	1	12	5.30e+01	2
CVXQP3_M	1000	750	0	2	3	3	3	0	0	2849	0.00e+00	3
CVXQP3_S	100	75	0	11	13	13	2	1	1	98	8.26e+02	2
DPKLO1	133	77	0	0	1	1	0	0	0	0	0.00e+00	1
DUAL1	85	1	0	0	1	1	1	0	0	0	0.00e+00	1
DUAL2	96	1	0	0	1	1	1	0	0	0	0.00e+00	1
DUAL3	111	1	0	0	1	1	1	0	0	0	0.00e+00	1
DUALC1	9	215	0	29	38	38	25	8	3.41e+00	1187	216	12
DUALC2	7	229	0	13	17	17	7	3	1	60	23	8
DUALC5	8	278	0	13	16	16	9	2	1	276	53	8
EXDATA	3000	3001	0	0	1	1	0	0	0	1	0.00e+00	1
GENHS28	10	8	0	0	1	1	0	0	0	0	0.00e+00	1
GOULDQP2	699	349	0	97	131	131	1	33	1.47e+01	1	9.64e+04	7
GOULDQP3	699	349	0	19	25	25	8	5	2.63e-01	27	7.10e+01	20
HS118	15	17	0	38	60	60	33	21	6	305	489	8
HS21	2	1	0	2	3	3	0	0	0	0	0.00e+00	3
HS268	5	5	0	0	1	1	0	0	0	0	0.00e+00	1
HS35MOD	3	1	0	2	3	3	0	0	0	0	0.00e+00	3
HS35	3	1	0	7	9	9	0	1	1.43e-01	0	0	8
HS51	5	3	0	0	1	1	0	0	0	0	0.00e+00	1
HS52	5	3	0	0	1	1	0	0	0	0	0.00e+00	1
HS53	5	3	0	0	1	1	0	0	0	0	0.00e+00	1
HS76	4	3	0	8	10	10	1	1	1.25e-01	0	2	9
HUES-MOD	10000	2	0	11	13	13	1	1	3.64e-01	4	1.60e+01	9
HUESTIS	10000	2	0	19	21	21	1	1	2.11e-01	4	1.60e+01	17
LISWET11	10002	10000	0	0	1	1	0	0	0	0	0.00e+00	1
LISWET12	10002	10000	0	0	1	1	0	0	0	0	0.00e+00	1
LISWET3	10002	10000	0	0	1	1	0	0	0	0	0.00e+00	1
LISWET4	10002	10000	0	0	1	1	0	0	0	0	0.00e+00	1
LISWET5	10002	10000	0	0	1	1	0	0	0	0	0.00e+00	1
LOTSCHD	12	7	0	9	11	11	2	1	3.33e-01	9	0.00e+00	8
MOSARQP1	2500	700	0	231	497	497	34	265	1.28e+02	75869	8.58e+04	7
MOSARQP2	900	600	0	44	65	65	7	20	7.45e+00	3525	4.32e+03	19
PRIMAL2	649	96	0	5	7	7	0	1	2.00e-01	103	1.84e+02	6
PRIMALC1	230	9	0	12	14	14	3	1	3.33e-01	14	2.70e+01	10
PRIMALC8	520	8	0	0	1	1	0	0	0	0	0.00e+00	1

Table 3.14: Results of Algorithm 11 (with step 14) employed to solve Maros and Meszaros problems.

Name	n	m	State	#Iter	#SM	#Feas	#Feas-Mod	Last-$\lvert\mathcal{U}\rvert$	Avg-$\lvert\mathcal{U}\rvert$	#Feas-Pvt	#QP-Iter	#LS
AUG2D	20200	10000	0	0	1	1	0	0	0	161	0.00e+00	1
CONT-101	10197	10098	-1	1	0	1	1	0	0	135309	0.00e+00	0
CVXQP2_M	1000	250	0	61	237	237	4	0	3.23e+00	3994	1.39e+04	50
CVXQP2_S	100	25	0	13	18	18	1	0	3.08-01	23	3.40e+01	14
CVXQP3_S	100	75	0	2	4	4	2	0	5.00e-01	25	1.24e+02	3
DUAL1	85	1	0	0	1	1	1	0	0	0	0.00e+00	1
DUAL2	96	1	0	0	1	1	1	0	0	0	0.00e+00	1
DUAL3	111	1	0	0	1	1	1	0	0	0	0.00e+00	1
GENHS28	10	8	0	0	1	1	0	0	0	0	0.00e+00	1
GOULDQP2	699	349	0	129	224	224	1	29	6.33e+00	1	1.25e+05	38
GOULDQP3	699	349	0	19	25	25	8	3	2.63e-01	27	7.10e+01	20
HS21	2	1	0	2	3	3	0	0	0	0	0.00e+00	3
HS268	5	5	0	0	1	1	0	0	0	0	0.00e+00	1
HS35MOD	3	1	0	2	3	3	0	0	0	0	0.00e+00	3
HS35	3	1	0	7	9	9	0	0	1.43e-01	0	0	8
HS51	5	3	0	0	1	1	0	0	0	0	0.00e+00	1
HS53	5	3	0	0	1	1	0	0	0	0	0.00e+00	1
HUES-MOD	10000	2	0	22	27	27	1	0	1.82e-01	3	1.60e+01	23
LISWET12	10002	10000	0	0	1	1	0	0	0	0	0.00e+00	1
LOTSCHD	12	7	0	15	20	20	2	0	2.67e-01	15	0	16
MOSARQP1	2500	700	0	317	823	823	80	11	5.79e+00	104427	2.87e+04	170
PRIMAL2	649	96	0	5	7	7	0	0	2.00e-01	103	1.84e+02	6
PRIMALC1	230	9	0	15	20	20	3	0	2.67e-01	14	2.90e+01	16
PRIMALC8	520	8	0	0	1	1	0	0	0	0	0.00e+00	1

3.5 Concluding Remarks

Motivated by the impressive practical performance of the primal-dual active-set method proposed by Hintermüller, Ito, and Kunisch [69] 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 [69].

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 [69] often converges extremely quickly when solving a convex BQP, despite the limitations of the method's theoretical convergence guarantees. Second, when the method in [69] 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 [69] 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 [69] (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 §3.4 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 [69].

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. In §4, we prove that for certain convex QPs the level of inexactness is indeed tractable to ensure global convergence. We also illustrate how incorporating inexactness in subproblem solves could effectively reduce the overall computational demand.

Chapter 4

PDAS with Inexact Subproblem Solves

In §3, we proposed a globally convergent PDAS algorithmic framework for generally-constrained strictly convex QPs. The main computational cost occurs in solving the subproblem which is either a reduced linear system (as in the case of Algorithm 4) or a reduced BQP (as in Algorithm 8). When the subproblem is a linear system, we propose in this chapter extra enhancements that can further reduce the computational cost. The novelty of the enhanced algorithms is that they allow inexactness in the reduced linear system solves at all partitions (except optimal ones). Such a feature is particularly important in large-scale settings when one employs iterative Krylov subspace methods [19, 98, 104, 114] to solve these systems. We establish theoretical foundations for the PDAS framework with inexact subproblem solves. We illustrate that incorporating inexactness can significantly reduce the overall cost of PDAS methods when solving large-scale problems.

We propose three primal-dual active-set methods for solving large-scale instances of an important class of QPs. The first algorithm is convergent on problems for which properties of the Hessian can be exploited to derive explicit bounds to be enforced on the reduced linear system residuals, whereas our second and third algorithms employ dynamic parameters to control the residuals to avoid the computation of such explicit bounds. We prove that, when applied to solve an important class of convex QPs, our algorithms converge from any initial partition. We also illustrate their practical behavior by providing the results of numerical experiments on a pair of discretized optimal control problems.

We remark that a straightforward heuristic involving a dynamic accuracy tolerance that decreases as the optimization process proceeds will not ensure convergence without strong assumptions on the employed linear system solver. Indeed, such is the approach employed in our third algorithm, for which our convergence guarantees are significantly weaker than for our first two algorithms. (We also provide an example illustrating why the convergence guarantees for such an algorithm must be weaker.) In short, our first two algorithms are able to attain stronger convergence guarantees as they involve procedures for computing an upper bound on the norm of the inverse of a particular submatrix during each iteration; our first algorithm computes such an upper bound explicitly, whereas our second algorithm incorporates a dynamic parameter that (effectively) replaces this upper bound.

This chapter is organized as follows. In §4.1, we state our problem of interest, basic concepts and definitions, and outline our algorithmic framework. In §4.2, we present our proposed algorithms and corresponding subroutines. We also prove that the algorithms attain convergence guarantees for a certain class of problems of interest. We then discuss an implementation of our algorithm in §4.3 and provide the results of numerical experiments on discretized optimal control problems in §4.4 to show that our inexact PDAS methods have advantages over a similar strategy that employs exact linear system solves. Finally, concluding remarks are provided in §4.5.

4.1 Fundamentals

For a positive integer n and nonnegative integer m, we define an index set of (upper) bounded variables $\mathcal{N} := \{1, \ldots, n\}$, index set of free variables $\mathcal{F} := \{n+1, \ldots, n+m\}$, and index set of equality constraints $\mathcal{M} := \{1, \ldots, m\}$. Then, given problem data in terms of $c \in \mathbb{R}^{n+m}$, $H \in \mathbb{R}^{(n+m)\times(n+m)}$, $A \in \mathbb{R}^{m\times(n+m)}$, $b \in \mathbb{R}^m$, and $u \in \mathbb{R}^n$, we consider the quadratic optimization problem

$$\min_{x\in\mathbb{R}^{n+m}} \; c^T \begin{bmatrix} x_{\mathcal{N}} \\ x_{\mathcal{F}} \end{bmatrix} + \tfrac{1}{2} \begin{bmatrix} x_{\mathcal{N}} \\ x_{\mathcal{F}} \end{bmatrix}^T H \begin{bmatrix} x_{\mathcal{N}} \\ x_{\mathcal{F}} \end{bmatrix} \quad \text{s.t.} \quad A \begin{bmatrix} x_{\mathcal{N}} \\ x_{\mathcal{F}} \end{bmatrix} = b, \; x_{\mathcal{N}} \leq u. \tag{4.1}$$

Throughout this chapter, we make the following assumption about the matrices in the problem data for a given instance of (4.1).

Assumption 4.1.1. *In (4.1), the Hessian of the objective function satisfies $H \succeq 0$ and $H_{\mathcal{N}\mathcal{N}} \succ 0$, and the constraint data submatrix $A_{\mathcal{M}\mathcal{F}}$ is invertible.*

Under Assumption 4.1.1, it follows that (4.1) is feasible and there exists a unique primal point x and unique Lagrange multipliers (y, z) satisfying the Karush-Kuhn-Tucker (KKT) optimality conditions for (4.1), which can be written as

$$0 = \text{KKT}(x, y, z) := \left(c + H \begin{bmatrix} x_{\mathcal{N}} \\ x_{\mathcal{F}} \end{bmatrix} + A^T y + \begin{bmatrix} z \\ 0 \end{bmatrix}, \; A \begin{bmatrix} x_{\mathcal{N}} \\ x_{\mathcal{F}} \end{bmatrix} - b, \; \min\{u - x_{\mathcal{N}}, z\} \right).$$

We define a partition $(\mathcal{A}, \mathcal{I})$ of the index set of bounded variables as a pair of mutually exclusive and exhaustive subsets of $\mathcal{N}$, where $\mathcal{A}$ represents an *active* set of variables (i.e., variables equal to their upper bounds) and $\mathcal{I} = \mathcal{N} \backslash \mathcal{A}$ represents the corresponding *inactive* set. Corresponding to a partition $(\mathcal{A}, \mathcal{I})$, we define a *subspace solution*, call it (x, y, z), by the following sequence of operations:

$$\text{Set} \;\; x_{\mathcal{A}} \leftarrow u_{\mathcal{A}} \;\; \text{and} \;\; z_{\mathcal{I}} \leftarrow 0, \tag{4.2a}$$

$$\text{then solve} \;\; \begin{bmatrix} H_{\mathcal{I}\mathcal{I}} & H_{\mathcal{I}\mathcal{F}} & [A_{\mathcal{M}\mathcal{I}}]^T \\ H_{\mathcal{F}\mathcal{I}} & H_{\mathcal{F}\mathcal{F}} & [A_{\mathcal{M}\mathcal{F}}]^T \\ A_{\mathcal{M}\mathcal{I}} & A_{\mathcal{M}\mathcal{F}} & 0 \end{bmatrix} \begin{bmatrix} x_{\mathcal{I}} \\ x_{\mathcal{F}} \\ y \end{bmatrix} = - \begin{bmatrix} c_{\mathcal{I}} \\ c_{\mathcal{F}} \\ -b \end{bmatrix} - \begin{bmatrix} H_{\mathcal{I}\mathcal{A}} \\ H_{\mathcal{F}\mathcal{A}} \\ A_{\mathcal{M}\mathcal{A}} \end{bmatrix} u_{\mathcal{A}} \tag{4.2b}$$

$$\text{for} \;\; (x_{\mathcal{I}}, x_{\mathcal{F}}, y),$$

$$\text{then set} \;\; z_{\mathcal{A}} \leftarrow -H_{\mathcal{A}\mathcal{N}} x_{\mathcal{N}} - H_{\mathcal{A}\mathcal{F}} x_{\mathcal{F}} - [A_{\mathcal{M}\mathcal{A}}]^T y - c_{\mathcal{A}}. \tag{4.2c}$$

Under Assumption 4.1.1, the matrix on the left-hand side of (4.2b) is nonsingular, and hence the subspace solution corresponding to $(\mathcal{A}, \mathcal{I})$ is unique. We call $(\mathcal{A}, \mathcal{I})$ an *optimal partition* if its subspace solution (x, y, z) satisfies $\text{KKT}(x, y, z) = 0$, i.e., if (x, y, z) is the optimal primal-dual solution for (4.1). Otherwise, the partition $(\mathcal{A}, \mathcal{I})$ and its corresponding subspace solution (x, y, z) are *suboptimal*. We remark that while the optimal primal-dual solution is unique for an instance of (4.1), there may be more than one optimal partition. Indeed, more generally, for a given instance of (4.1), multiple partitions may correspond to the same subspace solution.

Given a partition $(\mathcal{A}, \mathcal{I})$ and its corresponding subspace solution (x, y, z), we define the *violated sets* of indices of variables as given by

$$\mathcal{V}_P := \{i \in \mathcal{I} : x_i > u_i\} \quad \text{and} \quad \mathcal{V}_D := \{i \in \mathcal{A} : z_i < 0\}. \tag{4.3}$$

(Clearly, these violated sets depend on the partition $(\mathcal{A}, \mathcal{I})$. However, for brevity in our presentation, we do not indicate this dependence in the notation for $\mathcal{V}_P$ and $\mathcal{V}_D$. In all cases, the partition of interest will be clear from the context.) The following result has important consequences that we will use extensively.

Theorem 4.1.2. *Let (x, y, z) be the subspace solution corresponding to a given partition $(\mathcal{A}, \mathcal{I})$. Then, $(\mathcal{A}, \mathcal{I})$ is optimal for (4.1) if and only if $\mathcal{V}_P \cup \mathcal{V}_D = \emptyset$.*

Proof. By straightforward verification of the KKT conditions for (4.1), the subspace solution defined by (5.3) satisfies all KKT conditions, except perhaps subsets of $\min\{u - x_{\mathcal{N}}, z\} = 0$ corresponding to the bounds $x_{\mathcal{I}} \leq u_{\mathcal{I}}$ and $z_{\mathcal{A}} \geq 0$. Subsets of these bounds are violated if and only if the set $\mathcal{V}_P \cup \mathcal{V}_D$ is nonempty. $\square$

Consider the framework for solving (4.1) that is stated as Algorithm 12 below. Each iteration of Algorithm 12 involves the computation of a subspace solution as defined in (5.3). If the corresponding primal-dual solution estimate yields a zero (or sufficiently small, corresponding to an arbitrary vector norm $\|\cdot\|$) KKT residual, then the solution is (approximately) optimal and the algorithm terminates. Otherwise, subsets of the corresponding violated sets—the union of which is guaranteed by Theorem 4.1.2 to be nonempty—are chosen, the indices of which are switched from active to inactive, or vice versa, to create a new partition. This algorithm represents a generic framework that allows much flexibility, such as in the choices for $\mathcal{C}_P$ and $\mathcal{C}_D$ in Step 8. In the subsequent sections of this chapter, we propose three algorithms related to Algorithm 12 that allow inexactness in each subspace solution; along with these algorithms, details are provided for all algorithmic computations, including how one may consider choosing the sets $\mathcal{C}_P$ and $\mathcal{C}_D$.

Algorithm 12 Primal-Dual Active-Set (PDAS) Framework

```
1: Input an initial partition $(\mathcal{A}, \mathcal{I})$ and optimality tolerance $\varepsilon_{opt} \geq 0$.
2: loop
3:    Compute the subspace solution $(x, y, z)$ by (5.3).
4:    if $\|\mathrm{KKT}(x, y, z)\| \leq \varepsilon_{opt}$ then
5:       Terminate and return $(x, y, z)$.
6:    end if
7:    Set $\mathcal{V}_P$ and $\mathcal{V}_D$ by (4.3).
8:    Choose $\mathcal{C}_P \subseteq \mathcal{V}_P$ and $\mathcal{C}_D \subseteq \mathcal{V}_D$ such that $\mathcal{C}_P \cup \mathcal{C}_D \neq \emptyset$.
9:    Set $\mathcal{A} \leftarrow (\mathcal{A} \backslash \mathcal{C}_D) \cup \mathcal{C}_P$ and $\mathcal{I} \leftarrow (\mathcal{I} \backslash \mathcal{C}_P) \cup \mathcal{C}_D$.
10: end loop
```

The algorithm in [69] can be viewed as a special case of Algorithm 12. In particular, for the case when $m = 0$ (i.e., $\mathcal{F} = \emptyset$ and $\mathcal{M} = \emptyset$), it corresponds to Algorithm 12 with the choice $\mathcal{C}_P \leftarrow \mathcal{V}_P$ and $\mathcal{C}_D \leftarrow \mathcal{V}_D$ in Step 8 in each iteration. The authors of [69] provide convergence results for their algorithm that are similar to those in Theorem 4.1.3 below. For our purposes, we state the theorem in a more general setting so that it applies for Algorithm 12 above. A proof is given afterwards. For the result, recall that a real symmetric matrix is a P-matrix if all of its principal minors are positive (implying that the matrix is positive definite), and a P-matrix is called an M-matrix if all of its off-diagonal entries are nonpositive. We define $[A]_+ := \max\{0, A\}$ (where the max should be understood component-wise), the KKT system matrix

$$K := \begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix}, \tag{4.4}$$

and the following submatrix of K with its Schur complement with respect to K:

$$Q := \begin{bmatrix} H_{\mathcal{FF}} & [A_{\mathcal{MF}}]^T \\ A_{\mathcal{MF}} & 0 \end{bmatrix} \quad \text{and} \quad R := H_{\mathcal{NN}} - \begin{bmatrix} H_{\mathcal{FN}} \\ A_{\mathcal{MN}} \end{bmatrix}^T Q^{-1} \begin{bmatrix} H_{\mathcal{FN}} \\ A_{\mathcal{MN}} \end{bmatrix}. \tag{4.5}$$

Theorem 4.1.3. *Suppose that Assumption 4.1.1 holds and that the matrix R in (4.5) satisfies at least one of the following conditions:*

(a) R is a P-matrix and, corresponding to any partition $(\mathcal{A}, \mathcal{I})$, we have that $\|[R_{\mathcal{II}}^{-1} R_{\mathcal{IA}}]_+\|_1 < 1$ and $e^T R_{\mathcal{II}}^{-1} w \geq 0$ for any $w \geq 0$, where the latter inequality holds strictly, i.e., $e^T R_{\mathcal{II}}^{-1} w > 0$, whenever $w \neq 0$.

(b) $R = M + E$, where M is an M-matrix and $\|E\|_1$ is sufficiently small.

Then, with $\varepsilon_{opt} \geq 0$ and any initial partition, Algorithm 12 terminates in a finite number of iterations. In particular, if $\varepsilon_{opt} = 0$, then Algorithm 12 terminates in a finite number of iterations with a KKT point for (4.1).

We prove Theorem 4.1.3 by proving two theorems; the first corresponds to condition (a) of the theorem and the second corresponds to condition (b).

Theorem 4.1.4. *Suppose Assumption 4.1.1 holds, R is a P-matrix, and, corresponding to any partition $(\mathcal{A}, \mathcal{I})$, we have that $\|[R_{\mathcal{II}}^{-1} R_{\mathcal{IA}}]_+\|_1 < 1$ and $e^T R_{\mathcal{II}}^{-1} w \geq 0$ for any $w \geq 0$, where the latter inequality holds strictly, i.e., $e^T R_{\mathcal{II}}^{-1} w > 0$, whenever $w \neq 0$. Then, with $\varepsilon_{opt} \geq 0$ and any initial partition, Algorithm 12 terminates in a finite number of iterations. In particular, if $\varepsilon_{opt} = 0$, then Algorithm 12 terminates in a finite number of iterations with a KKT point for (4.1).*

Proof. It suffices to prove the result for $\varepsilon_{opt} = 0$. Thus, in the proof, we show that Algorithm 12 yields a KKT point in a finite number of iterations.

In order to derive a contradiction, suppose that Algorithm 12 generates an infinite number of partitions. For a given partition $(\mathcal{A}, \mathcal{I})$ considered in the algorithm, let $(\mathcal{A}^+, \mathcal{I}^+)$ be the subsequent partition in the algorithm. Furthermore, let (x, y, z) and (x^+, y^+, z^+), respectively, be the subspace solutions corresponding to these partitions. For any index $i \in \mathcal{A}^+$, we have by Step 9 of Algorithm 12 that either

$$\begin{aligned} & i \in \mathcal{A} \implies x_i = u_i = x_i^+ \\ \text{or} \;\; & i \in \mathcal{I} \implies x_i > u_i = x_i^+. \end{aligned}$$

Hence, it follows that $[x^+ - x]_{\mathcal{A}^+} \leq 0$. Similarly, for any index $i \in \mathcal{I}^+$, we have by Step 9 of Algorithm 12 that either

$$\begin{aligned} & i \in \mathcal{I} \implies z_i = 0 = z_i^+ \\ \text{or} \;\; & i \in \mathcal{A} \implies z_i < 0 = z_i^+. \end{aligned}$$

Hence, it follows that $[z^+ - z]_{\mathcal{I}^+} \geq 0$. Now, since (x, y, z) and (x^+, y^+, z^+) are subspace solutions, it follows from (5.3) that

$$R[x^+ - x]_{\mathcal{N}} + [z^+ - z] = 0, \tag{4.6}$$

which implies that

$$[x^+ - x]_{\mathcal{I}^+} = -R^{-1}_{\mathcal{I}^+\mathcal{I}^+}(R_{\mathcal{I}^+\mathcal{A}^+}[x^+ - x]_{\mathcal{A}^+} + [z^+ - z]_{\mathcal{I}^+}).$$

Moreover, from nonsingularity of the matrix R, it follows that if $[x^+ - x]_{\mathcal{A}^+}$ and $[z^+ - z]_{\mathcal{I}^+}$ are both zero, then $[x^+ - x]_{\mathcal{I}^+}$ and $[z^+ - z]_{\mathcal{A}^+}$ are both zero. By the partition update rule in Step 9 of Algorithm 12, this occurs if and only if the violated sets $\mathcal{V}_P$ and $\mathcal{V}_D$ corresponding to (x, y, z) satisfy $\mathcal{V}_P \cup \mathcal{V}_D$, which, by Theorem 4.1.2, occurs if and only if (x, y, z) is a KKT point for (4.1), i.e., $\text{KKT}(x, y, z) = 0$. However, in such a case, the algorithm would have terminated with (x, y, z), contradicting the supposition that an infinite number of partitions are generated. Hence, it follows that $[x^+ - x]_{\mathcal{A}^+}$ and $[z^+ - z]_{\mathcal{I}^+}$ cannot both be zero.

For brevity, we now define $\Delta x := x^+ - x$ and $\Delta z := z^+ - z$. From the discussion above and letting $e \in \mathbb{R}^n$ denote a vector of ones, we have

$$\begin{aligned}
& e^T \Delta x_{\mathcal{N}} \\
=\ & e_{\mathcal{A}^+}^T [\Delta x]_{\mathcal{A}^+} + e_{\mathcal{I}^+}^T [\Delta x]_{\mathcal{I}^+} \\
=\ & e_{\mathcal{A}^+}^T [\Delta x]_{\mathcal{A}^+} - e_{\mathcal{I}^+}^T R_{\mathcal{I}^+\mathcal{I}^+}^{-1} R_{\mathcal{I}^+\mathcal{A}^+} [\Delta x]_{\mathcal{A}^+} - e_{\mathcal{I}^+}^T R_{\mathcal{I}^+\mathcal{I}^+}^{-1} [\Delta z]_{\mathcal{I}^+} \\
\leq\ & -(1 - \|[R_{\mathcal{I}^+\mathcal{I}^+}^{-1} R_{\mathcal{I}^+\mathcal{A}^+}]_+\|_1) \|[\Delta x]_{\mathcal{A}^+}\|_1 - e_{\mathcal{I}^+}^T R_{\mathcal{I}^+\mathcal{I}^+}^{-1} [\Delta z]_{\mathcal{I}^+} < 0,
\end{aligned}$$

where, by the conditions of the theorem, the last inequality is strict since $[\Delta x]_{\mathcal{A}^+}$ and $[\Delta z]_{\mathcal{I}^+}$ cannot both be zero. Thus, the quantity $e^T x_{\mathcal{N}}$ strictly decreases in each iteration of Algorithm 12. However, this is a contradiction to the supposition that an infinite number of iterates are generated since x is uniquely determined by the partition and there are only a finite number of partitions of $\mathcal{N}$. Consequently, we have proved that Algorithm 12 must terminate finitely with a KKT point. □

Theorem 4.1.5. *Suppose Assumption 4.1.1 holds and R satisfies $R = M + E$, where M is an M-matrix and $\|E\|_1$ is sufficiently small. Then, with $\varepsilon_{opt} \geq 0$ and any initial partition, Algorithm 12 terminates in a finite number of iterations. In particular, if $\varepsilon_{opt} = 0$, then Algorithm 12 terminates in a finite number of iterations with a KKT point for (4.1).*

Proof. As in the proof of Theorem 4.1.4, it suffices to prove the result for $\varepsilon_{opt} = 0$. Furthermore, again as in the proof of Theorem 4.1.4, we suppose—in order to derive a contradiction—that Algorithm 12 generates an infinite number of partitions. Borrowing notation and conclusions from the proof of Theorem 4.1.4, we have $[\Delta x]_{\mathcal{A}^+} \leq 0$ and $[\Delta z]_{\mathcal{I}^+} \geq 0$. Moreover, for sufficiently small $\|E\|_1$, the matrix R is nonsingular,

$$R_{\mathcal{I}^+\mathcal{I}^+}^{-1} R_{\mathcal{I}^+\mathcal{A}^+} = M_{\mathcal{I}^+\mathcal{I}^+}^{-1} M_{\mathcal{I}^+\mathcal{A}^+} + \mathcal{O}(\|E\|_1) \quad \text{and} \quad R_{\mathcal{I}^+\mathcal{I}^+}^{-1} = M_{\mathcal{I}^+\mathcal{I}^+}^{-1} + \mathcal{O}(\|E\|_1).$$

Since M is an M-matrix, we have $M_{\mathcal{I}^+\mathcal{I}^+}^{-1} M_{\mathcal{I}^+\mathcal{A}^+} \leq 0$ and $M_{\mathcal{I}^+\mathcal{I}^+}^{-1} \geq 0$. Hence, since

$[\Delta x]_{\mathcal{A}^+}$ and $[\Delta z]_{\mathcal{I}^+}$ cannot both be zero, we have for sufficiently small $\|E\|_1$ that

$$\begin{aligned}
&e^T \Delta x_{\mathcal{N}} \\
&= e_{\mathcal{A}^+}^T [\Delta x]_{\mathcal{A}^+} + e_{\mathcal{I}^+}^T [\Delta x]_{\mathcal{I}^+} \\
&= e_{\mathcal{A}^+}^T [\Delta x]_{\mathcal{A}^+} - e_{\mathcal{I}^+}^T R_{\mathcal{I}^+\mathcal{I}^+}^{-1} R_{\mathcal{I}^+\mathcal{A}^+} [\Delta x]_{\mathcal{A}^+} - e_{\mathcal{I}^+}^T R_{\mathcal{I}^+\mathcal{I}^+}^{-1} [\Delta z]_{\mathcal{I}^+} \\
&= e_{\mathcal{A}^+}^T [\Delta x]_{\mathcal{A}^+} - e_{\mathcal{I}^+}^T M_{\mathcal{I}^+\mathcal{I}^+}^{-1} M_{\mathcal{I}^+\mathcal{A}^+} [\Delta x]_{\mathcal{A}^+} - e_{\mathcal{I}^+}^T M_{\mathcal{I}^+\mathcal{I}^+}^{-1} [\Delta z]_{\mathcal{I}^+} + \mathcal{O}(\|E\|_1) < 0.
\end{aligned}$$

The remainder of the proof follows in the same manner as that of Theorem 4.1.4. □

4.2 Algorithm Descriptions

In this section, we propose three algorithms for solving (4.1). Each algorithm has the same basic structure as Algorithm 12, but allows inexactness in the reduced linear system solves. In the first algorithm that we propose, a tolerance for inexactness is set based on an upper bound on a norm of a particular submatrix. We illustrate that such a bound can be computed efficiently in certain cases of interest. In the second and third algorithms, the inexactness tolerance is set based on a parameter that is updated dynamically within the algorithm. For all algorithms, we prove that the guarantees of Theorem 4.1.3 are maintained.

The algorithms in this section employ an extension of the operations specified in (5.3). In particular, corresponding to a partition $(\mathcal{A}, \mathcal{I})$, we define an *inexact subspace solution*, call it $(\tilde{x}, \tilde{y}, \tilde{z})$, by the following operations (where by "≈" in (4.7b) we are indicating that the "solve" may be approximate):

$$\text{Set} \quad \tilde{x}_{\mathcal{A}} \leftarrow u_{\mathcal{A}} \quad \text{and} \quad \tilde{z}_{\mathcal{I}} \leftarrow 0, \tag{4.7a}$$

$$\text{then solve} \quad \begin{bmatrix} H_{\mathcal{I}\mathcal{I}} & H_{\mathcal{I}\mathcal{F}} & [A_{\mathcal{M}\mathcal{I}}]^T \\ H_{\mathcal{F}\mathcal{I}} & H_{\mathcal{F}\mathcal{F}} & [A_{\mathcal{M}\mathcal{F}}]^T \\ A_{\mathcal{M}\mathcal{I}} & A_{\mathcal{M}\mathcal{F}} & 0 \end{bmatrix} \begin{bmatrix} x_{\mathcal{I}} \\ x_{\mathcal{F}} \\ y \end{bmatrix} \approx - \begin{bmatrix} c_{\mathcal{I}} \\ c_{\mathcal{F}} \\ -b \end{bmatrix} - \begin{bmatrix} H_{\mathcal{I}\mathcal{A}} \\ H_{\mathcal{F}\mathcal{A}} \\ A_{\mathcal{M}\mathcal{A}} \end{bmatrix} u_{\mathcal{A}} \tag{4.7b}$$

$$\text{for} \quad (\tilde{x}_{\mathcal{I}}, \tilde{x}_{\mathcal{F}}, \tilde{y}),$$

$$\text{then set} \quad \tilde{z}_{\mathcal{A}} \leftarrow -H_{\mathcal{A}\mathcal{N}}\tilde{x}_{\mathcal{N}} - H_{\mathcal{A}\mathcal{F}}\tilde{x}_{\mathcal{F}} - [A_{\mathcal{M}\mathcal{A}}]^T\tilde{y} - c_{\mathcal{A}} \tag{4.7c}$$

$$\text{and} \quad \begin{bmatrix} \tilde{r}_{\mathcal{N}} \\ \tilde{r}_{\mathcal{F}} \\ \tilde{t} \end{bmatrix} \leftarrow \begin{bmatrix} c_{\mathcal{N}} \\ c_{\mathcal{F}} \\ -b \end{bmatrix} + \begin{bmatrix} H_{\mathcal{N}\mathcal{N}} & H_{\mathcal{N}\mathcal{F}} & [A_{\mathcal{M}\mathcal{N}}]^T \\ H_{\mathcal{F}\mathcal{N}} & H_{\mathcal{F}\mathcal{F}} & [A_{\mathcal{M}\mathcal{F}}]^T \\ A_{\mathcal{M}\mathcal{N}} & A_{\mathcal{M}\mathcal{F}} & 0 \end{bmatrix} \begin{bmatrix} \tilde{x}_{\mathcal{N}} \\ \tilde{x}_{\mathcal{F}} \\ \tilde{y} \end{bmatrix} + \begin{bmatrix} \tilde{z} \\ 0 \\ 0 \end{bmatrix}. \tag{4.7d}$$

The vector $(\tilde{r}, \tilde{t})$ is the residual in the (reduced) linear system solve that produces $(\tilde{x}, \tilde{y}, \tilde{z})$ via (4.7b). Under Assumption 4.1.1, we find by comparing (5.3) and (4.7) that $(\tilde{x}, \tilde{y}, \tilde{z}) = (x, y, z)$ if and only if $(\tilde{r}, \tilde{t}) = 0$.

In each of the algorithms proposed in this section, we iteratively solve the linear system in (4.7b) until either it is verified that the partition $(\mathcal{A}, \mathcal{I})$ is optimal (with respect to a tolerance $\varepsilon_{opt} \geq 0$) or the inexact subspace solution is sufficiently accurate in that it leads to a productive update of the partition. In our first algorithm, we provide a strategy in which we identify subsets of the violated sets $\mathcal{V}_P$ and $\mathcal{V}_D$ corresponding to the *exact* subspace solution (x, y, z) *without* having to explicitly compute this exact solution. In this manner, the algorithm fits into the framework of Algorithm 12. In our other algorithms, we do not necessarily identify subsets of these violated sets, though we can still ensure convergence guarantees by employing and appropriately updating a dynamic algorithmic parameter.

4.2.1 An Algorithm with a Partition-Defined Subproblem Tolerance

Our first algorithm imposes a tolerance on the residual $(\tilde{r}, \tilde{t})$ defined in (4.7d) that is based on a partition-defined value with which we can guarantee that at any suboptimal partition a subset of the union $\mathcal{V}_P \cup \mathcal{V}_D$ corresponding to the *exact* subspace solution (x, y, z) will be identified. In order to motivate the tolerance that we employ, we first explore, for a given partition $(\mathcal{A}, \mathcal{I})$, the relationship between the subspace solution (x, y, z) defined by (5.3) and an inexact subspace solution $(\tilde{x}, \tilde{y}, \tilde{z})$ defined by (4.7). Then, once the tolerance is established, we present our first inexact PDAS algorithm, followed by subsections in which we discuss details of subroutines of the algorithm. It is important to note that we assume that a subroutine is available for computing *exact* solutions of linear systems with Q in (4.5); i.e., we assume products with Q^{-1} can be computed. This is a reasonable assumption in certain applications, especially since the matrix Q is fixed, i.e., it does not depend on the partition.

Given a partition $(\mathcal{A}, \mathcal{I})$ and recalling (4.4), consider the decomposition

$$K = \begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} H_{\mathcal{A}\mathcal{A}} \end{bmatrix} & \begin{bmatrix} H_{\mathcal{A}\mathcal{I}} & S_{[\mathcal{A}]}^T \end{bmatrix} \\ \begin{bmatrix} H_{\mathcal{I}\mathcal{A}} \\ S_{[\mathcal{A}]} \end{bmatrix} & \begin{bmatrix} H_{\mathcal{I}\mathcal{I}} & S_{[\mathcal{I}]}^T \\ S_{[\mathcal{I}]} & Q \end{bmatrix} \end{bmatrix}, \tag{4.8}$$

where (using a subscript $[\cdot]$ to indicate dependence on an index set) we define

$$K_{[\mathcal{I}]} := \begin{bmatrix} H_{\mathcal{I}\mathcal{I}} & S_{[\mathcal{I}]}^T \\ S_{[\mathcal{I}]} & Q \end{bmatrix}, \quad S_{[\mathcal{A}]} := \begin{bmatrix} H_{\mathcal{F}\mathcal{A}} \\ A_{\mathcal{M}\mathcal{A}} \end{bmatrix}, \quad \text{and} \quad S_{[\mathcal{I}]} := \begin{bmatrix} H_{\mathcal{F}\mathcal{I}} \\ A_{\mathcal{M}\mathcal{I}} \end{bmatrix}.$$

(Note that $K_{[\mathcal{N}]} = K$ and $R_{\mathcal{I}\mathcal{I}} = H_{\mathcal{I}\mathcal{I}} - S_{[\mathcal{I}]}^T Q^{-1} S_{[\mathcal{I}]}$ is the Schur complement of Q with respect to $K_{[\mathcal{I}]}$.) Observing (5.3) and (4.7), it follows that $\tilde{x}_{\mathcal{A}} = u_{\mathcal{A}} = x_{\mathcal{A}}$ and $\tilde{z}_{\mathcal{I}} = 0 = z_{\mathcal{I}}$. In addition, defining the residual subvector $\tilde{v} := (\tilde{r}_{\mathcal{F}}, \tilde{t})$, we have

$$x_{\mathcal{I}} = \tilde{x}_{\mathcal{I}} - R_{\mathcal{I}\mathcal{I}}^{-1} \tilde{r}_{\mathcal{I}} + R_{\mathcal{I}\mathcal{I}}^{-1} S_{[\mathcal{I}]}^T Q^{-1} \tilde{v} \tag{4.9a}$$

$$\begin{aligned} \text{and} \quad z_{\mathcal{A}} = \tilde{z}_{\mathcal{A}} &+ (H_{\mathcal{A}\mathcal{I}} - S_{[\mathcal{A}]}^T Q^{-1} S_{[\mathcal{I}]}) R_{\mathcal{I}\mathcal{I}}^{-1} \tilde{r}_{\mathcal{I}} \\ &- (H_{\mathcal{A}\mathcal{I}} R_{\mathcal{I}\mathcal{I}}^{-1} S_{[\mathcal{I}]}^T - S_{[\mathcal{A}]}^T - S_{[\mathcal{A}]}^T Q^{-1} S_{[\mathcal{I}]} R_{\mathcal{I}\mathcal{I}}^{-1} S_{[\mathcal{I}]}^T) Q^{-1} \tilde{v}. \end{aligned} \tag{4.9b}$$

Observing (4.9), it follows that the violated sets $\mathcal{V}_P$ and $\mathcal{V}_D$ corresponding to (x, y, z)

can be defined in terms of an inexact subspace solution $(\tilde{x}, \tilde{y}, \tilde{z})$ and its residual $(\tilde{r}, \tilde{t})$. In particular, for an index $i \in \mathcal{I}$, the ith element of x violates its upper bound if the corresponding element on the right-hand side of (4.9a) is greater than u_i, and, for an index $i \in \mathcal{A}$, the ith element of z violates its lower bound (of zero) if the corresponding element on the right-hand side of (4.9b) is negative. This revised viewpoint of the elements of $x_{\mathcal{I}}$ and $z_{\mathcal{A}}$ does not immediately yield any benefits since the evaluation of the terms on the right-hand sides of (4.9) is equivalent to that of solving (4.7b) exactly. However, it reveals that with an inexact subspace solution $(\tilde{x}, \tilde{y}, \tilde{z})$ and bounds on the residual terms in (4.9), we may identify subsets of $\mathcal{V}_P$ and $\mathcal{V}_D$ without computing (x, y, z) explicitly. The following lemma suggests a strategy for such a procedure.

Lemma 4.2.1. *Given a partition $(\mathcal{A}, \mathcal{I})$, let (x, y, z) be the corresponding subspace solution and let $(\tilde{x}, \tilde{y}, \tilde{z})$ be an inexact subspace solution with residual $(\tilde{r}, \tilde{t})$. Furthermore, suppose that with $\tilde{v} := (\tilde{r}_{\mathcal{F}}, \tilde{t})$ we have $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ satisfying*

$$\begin{aligned} \alpha_{\mathcal{I}} &\geq R_{\mathcal{I}\mathcal{I}}^{-1} \tilde{r}_{\mathcal{I}} - R_{\mathcal{I}\mathcal{I}}^{-1} S_{[\mathcal{I}]} Q^{-1} \tilde{v} \qquad (4.10a) \\ \textit{and} \quad \beta_{\mathcal{A}} &\geq (H_{\mathcal{A}\mathcal{I}} - S_{[\mathcal{A}]}^T Q^{-1} S_{[\mathcal{I}]}) R_{\mathcal{I}\mathcal{I}}^{-1} \tilde{r}_{\mathcal{I}} \\ &\quad - (H_{\mathcal{A}\mathcal{I}} R_{\mathcal{I}\mathcal{I}}^{-1} S_{[\mathcal{I}]}^T - S_{[\mathcal{A}]}^T - S_{[\mathcal{A}]}^T Q^{-1} S_{[\mathcal{I}]} R_{\mathcal{I}\mathcal{I}}^{-1} S_{[\mathcal{I}]}^T) Q^{-1} \tilde{v}. \qquad (4.10b) \end{aligned}$$

Then, for the violated sets $\mathcal{V}_P$ and $\mathcal{V}_D$ corresponding to (x, y, z), we have

$$\tilde{\mathcal{V}}_P := \{i \in \mathcal{I} : \tilde{x}_i - \alpha_i > u_i\} \subseteq \mathcal{V}_P \quad \textit{and} \quad \tilde{\mathcal{V}}_D := \{i \in \mathcal{A} : \tilde{z}_i + \beta_i < 0\} \subseteq \mathcal{V}_D. \qquad (4.11)$$

Moreover, if $(\mathcal{A}, \mathcal{I})$ is suboptimal, then there exists $\varepsilon > 0$ such that (4.10)–(4.11) with

$$\|\alpha_{\mathcal{I}}\| = \mathcal{O}(\|(\tilde{r}, \tilde{t})\|) \quad \textit{and} \quad \|\beta_{\mathcal{A}}\| = \mathcal{O}(\|(\tilde{r}, \tilde{t})\|) \qquad (4.12)$$

yields $\tilde{\mathcal{V}}_P = \mathcal{V}_P$ and $\tilde{\mathcal{V}}_D = \mathcal{V}_D$ for any inexact subspace solution with $\|(\tilde{r}, \tilde{t})\| \leq \varepsilon$.

Proof. By (4.10) and the relationships in (4.9), we have that for $i \in \mathcal{I}$ the inequality $\tilde{x}_i - \alpha_i > u_i$ implies $x_i > u_i$, and for $i \in \mathcal{A}$ the inequality $\tilde{z}_i + \beta_i < 0$ implies $z_i < 0$. Hence, with (4.11), it follows that $\tilde{\mathcal{V}}_P \subseteq \mathcal{V}_P$ and $\tilde{\mathcal{V}}_D \subseteq \mathcal{V}_D$.

Now suppose that $(\mathcal{A},\mathcal{I})$ is suboptimal, from which it follows by Theorem 4.1.2 that $\mathcal{V}_P \cup \mathcal{V}_D \neq \emptyset$. If $\mathcal{V}_P \neq \emptyset$, then for any $i \in \mathcal{V}_P$ we have $x_i > u_i$. Moreover, by continuity of the linear transformation defined by the inverse of the matrix on the left-hand side of (4.7b), for this $i \in \mathcal{V}_P$ there exists $\varepsilon_i > 0$ such that for any $(\tilde{r},\tilde{t})$ with $\|(\tilde{r},\tilde{t})\| \leq \varepsilon_i$, the condition (4.12) implies $\tilde{x}_i - \alpha_i > u_i$. Similar analysis shows that if $\mathcal{V}_D \neq \emptyset$, then for any $i \in \mathcal{V}_D$ there exists $\varepsilon_i > 0$ such that for any $(\tilde{r},\tilde{t})$ with $\|(\tilde{r},\tilde{t})\| \leq \varepsilon_i$, the condition (4.12) implies $\tilde{z}_i + \beta_i < 0$. Since $\tilde{\mathcal{V}}_P \subseteq \mathcal{V}_P$ and $\tilde{\mathcal{V}}_D \subseteq \mathcal{V}_D$, it follows that with $\varepsilon := \min\{\varepsilon_i : i \in \mathcal{V}_P \cup \mathcal{V}_D\} > 0$, we have $\tilde{\mathcal{V}}_P = \mathcal{V}_P$ and $\tilde{\mathcal{V}}_D = \mathcal{V}_D$. □

Lemma 4.2.1 proves that at any suboptimal partition $(\mathcal{A},\mathcal{I})$, a subset of the union of violated sets $\mathcal{V}_P \cup \mathcal{V}_D$ can be identified as long as upper bounds $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ are available and are proportional (in terms of any given norm $\|\cdot\|$) to the residual vector $(\tilde{r},\tilde{t})$. On the other hand, if a partition $(\mathcal{A},\mathcal{I})$ is optimal, then with a sufficiently small residual we obtain a sufficiently accurate primal-dual solution. Motivated by these observations, we propose the inexact PDAS framework presented as Algorithm 13.

Algorithm 13 PDAS Framework with Inexact Subspace Solutions

```
 1: Input an initial partition $(\mathcal{A},\mathcal{I})$ and optimality tolerance $\varepsilon_{opt} \geq 0$.
 2: loop
 3:   Compute an inexact subspace solution $(\tilde{x},\tilde{y},\tilde{z})$ with residual $(\tilde{r},\tilde{t})$ by (4.7).
 4:   if $\|\mathrm{KKT}(\tilde{x},\tilde{y},\tilde{z})\| \leq \varepsilon_{opt}$ then
 5:     Terminate and return $(\tilde{x},\tilde{y},\tilde{z})$.
 6:   end if
 7:   Compute $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ satisfying (4.10) and (4.12).
 8:   Set $\tilde{\mathcal{V}}_P$ and $\tilde{\mathcal{V}}_D$ by (4.11).
 9:   if $\tilde{\mathcal{V}}_P \cup \tilde{\mathcal{V}}_D \neq \emptyset$ then
10:     Choose $\mathcal{C}_P \subseteq \tilde{\mathcal{V}}_P$ and $\mathcal{C}_D \subseteq \tilde{\mathcal{V}}_D$ such that $\mathcal{C}_P \cup \mathcal{C}_D \neq \emptyset$.
11:     Set $\mathcal{A} \leftarrow (\mathcal{A}\backslash\mathcal{C}_D) \cup \mathcal{C}_P$ and $\mathcal{I} \leftarrow (\mathcal{I}\backslash\mathcal{C}_P) \cup \mathcal{C}_D$.
12:   end if
13: end loop
```

We have the following result, indicating conditions on the inexact subspace solutions computed in the algorithm—or, more precisely, on their corresponding residual vectors—that are necessary to ensure convergence.

Theorem 4.2.2. *Suppose that the conditions of Theorem 4.1.3 hold for the partitions generated by Algorithm 13. Then, the following hold:*

(i) If, for any partition, repeated executions of Step 3 yield $(\tilde{r}, \tilde{t}) \to 0$, then, with $\varepsilon_{opt} > 0$, Algorithm 13 terminates after a finite number of partition updates.

(ii) If there exists a positive integer J such that, for any partition, at most J executions of Step 3 yields $(\tilde{r}, \tilde{t}) = 0$, then, with $\varepsilon_{opt} \geq 0$, Algorithm 13 terminates in a finite number of iterations. In particular, if $\varepsilon_{opt} = 0$, then Algorithm 13 terminates in a finite number of iterations with a KKT point for (4.1).

Proof. Given any partition, it follows by the conditions in (i) and Lemma 4.2.1 that after a finite number of executions of Step 3, the sets $\tilde{\mathcal{V}}_P \subseteq \mathcal{V}_P$ and $\tilde{\mathcal{V}}_D \subseteq \mathcal{V}_D$ defined by (4.11) satisfy $\tilde{\mathcal{V}}_P \cup \tilde{\mathcal{V}}_D \neq \emptyset$. The result in (i) then follows from Theorem 4.1.3 and the fact that after a finite number of partition updates, a partition is identified such that repeated executions of Step 3 yield $(\tilde{x}, \tilde{y}, \tilde{z})$ satisfying the condition in Step 5 of the algorithm. The result in (ii) follows in a similar manner due to the additional observation that, given any partition, at most J executions of Step 3 occur before the condition in Step 5 is satisfied or the partition is modified. $\square$

There remain many details that need to be specified for a practical implementation of Algorithm 13. These details are the subjects of the following three subsections. First, due to the observation that upper bounds $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ in Step 7 are easily computed once one obtains an upper bound for the norm of the matrix $R_{\mathcal{I}\mathcal{I}}^{-1}$, we present an algorithm for computing such a bound in certain cases of interest. Second, we present a generic technique for computing $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ once such a bound is obtained. Third, we outline conditions that one may choose to impose—in addition to the condition that $\tilde{\mathcal{V}}_P \cup \tilde{\mathcal{V}}_D \neq \emptyset$—in the **if** statement in Step 9 of Algorithm 13.

Obtaining an upper bound for $\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1$

In this subsection, given an index set $\mathcal{I}$, we present a technique for computing an upper bound for $\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1$. Our technique amounts to solving a linear system of equations by an iterative process, where the computed upper bound may become tighter as the linear system is solved more accurately. (Indeed, under certain conditions, an exact solution of the linear system reveals $\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1$.) In this manner, it is clear how the upper bounds $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ required in Step 7 may be improved during the loop of Algorithm 13: For a given partition $(\mathcal{A}, \mathcal{I})$, repeated executions of the algorithm in this subsection eventually lead to a tighter upper bound for $\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1$, which in turn eventually lead to tighter upper bounds $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ via the algorithm in the following subsection.

Given a symmetric $p \times p$ matrix $B \succ 0$, consider the set

$$\mathcal{P}(B) := \{w \in \mathbb{R}^p : w > 0,\ Bw > 0,\ \|w\|_\infty = 1\}$$

and the symmetric $p \times p$ matrix $\mathfrak{M}(B)$ defined, for all $\{i, j\} \subseteq \{1, \dots, p\}$, by

$$[\mathfrak{M}(B)]_{ij} = \begin{cases} |B_{ii}| & \text{if } i = j \\ -|B_{ij}| & \text{if } i \neq j. \end{cases}$$

The matrix B is called an H-matrix if and only if $\mathfrak{M}(B)$ is a nonsingular M-matrix. Moreover, according to [118, eq. (5)], the following three statements are equivalent:

1. B is an H-matrix.
2. $\mathfrak{M}(B)$ is an M-matrix.
3. $\mathcal{P}(B)$ is nonempty.

We also have the following result, which is a special case of [118, Theorem 1]. (In [118], the author discusses upper bounds for the ℓ_∞-norm of a matrix inverse. However, since our matrix is symmetric, we can equivalently refer to its ℓ_1-norm.)

Lemma 4.2.3. *If $B \in \mathbb{R}^{p \times p}$ is a symmetric H-matrix, then, for any $w \in \mathcal{P}(B)$,*

$$\|B^{-1}\|_1 \leq \|\mathfrak{M}(B)^{-1}\|_1 \leq (\min\{[Aw]_i : 1 \leq i \leq p\})^{-1},$$

where the second inequality holds as an equality when $w = B^{-1}e/\|B^{-1}e\|_\infty \in \mathcal{P}(B)$.

By Lemma 4.2.3, if $R_{\mathcal{I}\mathcal{I}}$ is an H-matrix, then we can obtain an upper bound for $\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1$ by iteratively solving the linear system $\mathfrak{M}(R_{\mathcal{I}\mathcal{I}})w = e$ for $w \in \mathbb{R}^{|\mathcal{I}|}$, terminating whenever an element of $\mathcal{P}(R_{\mathcal{I}\mathcal{I}})$ is obtained. We formalize this strategy as Algorithm 14, for which we have the following result.

Algorithm 14 Subroutine for Obtaining an Upper Bound of $\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1$

```
1: Input $R_{\mathcal{I}\mathcal{I}}$.
2: **loop**
3:   Compute an inexact solution $w$ of $\mathfrak{M}(R_{\mathcal{I}\mathcal{I}})w = e$.
4:   **if** $\mathfrak{M}(R_{\mathcal{I}\mathcal{I}})w > 0$ **then**
5:     Terminate and **return**
         $\|w\|_\infty(\min\{[R_{\mathcal{I}\mathcal{I}}w]_i : 1 \le i \le |\mathcal{I}|\})^{-1}.$
6:   **end if**
7: **end loop**
```

Lemma 4.2.4. *Suppose Assumption 4.1.1 holds and R satisfies $R = M + E$ where M is an M-matrix and $\|E\|_1$ is sufficiently small. Then, for any $\mathcal{I} \subseteq \mathcal{N}$, if repeated executions of Step 3 of Algorithm 14 yield $\mathfrak{M}(R_{\mathcal{I}\mathcal{I}})w \to e$, then Algorithm 14 will terminate and return an upper bound for $\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1$.*

Proof. Since $R = M + E$, we have $R_{\mathcal{I}\mathcal{I}} = M_{\mathcal{I}\mathcal{I}} + E_{\mathcal{I}\mathcal{I}}$ for some M-matrix $M_{\mathcal{I}\mathcal{I}}$. Since $M_{\mathcal{I}\mathcal{I}}$ is an M-matrix, it is also an H-matrix since $\mathfrak{M}(M_{\mathcal{I}\mathcal{I}}) = M_{\mathcal{I}\mathcal{I}}$. Then, since $M_{\mathcal{I}\mathcal{I}}$ is an H-matrix, it follows that for sufficiently small $\|E\|_1$ we have sufficiently small $\|E_{\mathcal{I}\mathcal{I}}\|_1$ such that $R_{\mathcal{I}\mathcal{I}}$ is also an H-matrix. The result then follows by Lemma 4.2.3 since, under the conditions of the lemma, the algorithm eventually computes w satisfying $\mathfrak{M}(R_{\mathcal{I}\mathcal{I}})w > 0$. □

We close this subsection by remarking that if $R_{\mathcal{I}\mathcal{I}}$ is strictly diagonally dominant, then by computing $w = e$ in Step 3, Algorithm 14 would terminate in the first iteration of the **loop** and return the upper bound given by

$$\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1 \le (\min\{[R_{\mathcal{I}\mathcal{I}}e]_i : 1 \le i \le |\mathcal{I}|\})^{-1}.$$

This is known as the Ahlberg-Nilson-Varah bound [2, 117].

Obtaining upper bounds $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$

Given a partition $(\mathcal{A}, \mathcal{I})$, an inexact subspace solution $(\tilde{x}, \tilde{y}, \tilde{z})$ with residual $(\tilde{r}, \tilde{t})$, and an upper bound for a norm of $R_{\mathcal{I}\mathcal{I}}^{-1}$ (say, computed via Algorithm 14), in this subsection we present a generic method for computing upper bounds $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ satisfying (4.10) and (4.12).

We present our technique as the following algorithm. For the approach, we recall that since Q is independent of the partition, we suppose that this matrix is formed and factorized at the start of the optimization process so that products with Q^{-1} are available. Thus, the computational cost of executing the algorithm is relatively low, especially if the matrix $Q^{-1}S_{[\mathcal{N}]}$ is precomputed.

Algorithm 15 Subroutine for Obtaining $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ Satisfying (4.10) and (4.12)

```
1: Input $p \geq 1$, $q \geq 1$, and $\gamma \geq \|R_{\mathcal{I}\mathcal{I}}^{-1}\|_p$ such that $\|\cdot\|_p$ and $\|\cdot\|_q$ are dual norms.
2: Terminate and **return**
       $\alpha_{\mathcal{I}} \leftarrow (\gamma\|\tilde{r}_{\mathcal{I}} - S_{[\mathcal{I}]}Q^{-1}\tilde{v}\|_q)e$
   and $\beta_{\mathcal{A}} \leftarrow [H_{\mathcal{A}\mathcal{I}} - S_{[\mathcal{A}]}^TQ^{-1}S_{[\mathcal{I}]}]_+\alpha_{\mathcal{I}} - [H_{\mathcal{A}\mathcal{I}} - S_{[\mathcal{A}]}^TQ^{-1}S_{[\mathcal{I}]}]_-\alpha_{\mathcal{I}} + S_{[\mathcal{A}]}Q^{-1}\tilde{v}$.
```

The fact that the upper bounds $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ generated by Algorithm 15 satisfy (4.10) and (4.12) follows as a consequence of the following lemma. Recall that we define e as a vector of ones whose length is determined by the context in which it appears. For the proof of the lemma, we also define e_i as the corresponding ith unit vector.

Lemma 4.2.5. *Consider $B \in \mathbb{R}^{m\times n}$ with ith row denoted by B_i for $i \in \{1,\ldots,m\}$, a vector $w \in \mathbb{R}^n$, and a vector norm $\|\cdot\|_p$ with corresponding dual norm $\|\cdot\|_q$. Then, $B_iw \leq \|B_i^T\|_p\|w\|_q$ for all $i \in \{1,\ldots,m\}$. Furthermore, $Bw \leq \|B^T\|_p\|w\|_q e$.*

Proof. Hölder's inequality implies that $|B_iw| \leq \|B_i^T\|_p\|w\|_q$ for all $i \in \{1,\ldots,m\}$, as desired. Then, the remainder of the results follows since

$$|[Bw]_i| = |B_iw| \leq \|B_i^T\|_p\|w\|_q \leq \|B^T\|_p\|w\|_q \quad \text{for all } i \in \{1,\ldots,m\},$$

where the last inequality follows from the fact that $\|B_i\|_p = \|B^Te_i\|_p \leq \|B^T\|_p$. □

We remark that when employing Algorithm 14 to compute an upper bound for $\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1$, it is natural to employ Algorithm 15 with $p = 1$ and $q = \infty$. This is the approach used in our implementation and numerical experiments.

Conditions for executing a partition update

We close our discussion of Algorithm 13 by describing conditions that one may choose to impose—in addition to the (generally very loose) requirement that $\tilde{\mathcal{V}}_P \cup \tilde{\mathcal{V}}_D \neq \emptyset$—in the **if** statement in Step 9. As already shown in Theorem 4.2.2, the convergence of the algorithm is guaranteed with the condition as it is stated in the algorithm. Therefore, the addition of extra conditions is not necessary to ensure convergence. However, in our experience, it is worthwhile to impose extra conditions to ensure that any update to the partition that is performed will lead to substantial progress toward a solution, which can be expected to be the case when either the inexact subspace solution is sufficiently accurate and/or the modification to the partition will involve a large number of indices switching from active to inactive, or vice versa. We have found the conditions that we state in this section to work well in practice, though one may imagine other possible conditions that could be imposed.

Let $(\tilde{x}', \tilde{y}', \tilde{z}')$ be a given inexact subspace solution with residual $(\tilde{r}', \tilde{t}')$. For example, one may consider $(\tilde{x}', \tilde{y}', \tilde{z}') = (0, 0, 0)$ or the inexact subspace solution corresponding to primal-dual variable values as computed in the previous iteration of Algorithm 13. Given a tolerance $\epsilon_{res} \in (0, 1)$ and a vector norm $\|\cdot\|$, a condition that one may choose to impose is the following, similar to conditions typically found in inexact Newton methods for solving systems of equations [32]:

$$\|(\tilde{r}, \tilde{t})\| \leq \epsilon_{res} \|(\tilde{r}', \tilde{t}')\|. \tag{4.13}$$

That is, one may choose not to modify the partition until the residual vector $(\tilde{r}, \tilde{t})$ is sufficiently small in norm compared to the reference residual $(\tilde{r}', \tilde{t}')$ corresponding to the reference solution $(\tilde{x}', \tilde{y}', \tilde{z}')$. If the right-hand side of (4.13) is zero, then $(x, y, z) = (\tilde{x}', \tilde{y}', \tilde{z}')$; otherwise, (4.13) will eventually be satisfied as long as the employed iterative solver ensures that the residual vanishes, i.e., $(\tilde{r}, \tilde{t}) \to 0$.

In our experience, we have also found it beneficial to avoid modifying the partition until there is consistency between the sets $\tilde{\mathcal{V}}_P$ and $\tilde{\mathcal{V}}_D$ in (4.11) and

$$\tilde{\mathcal{V}}'_P := \{i \in \mathcal{I} : \tilde{x}_i > u_i\} \text{ and } \tilde{\mathcal{V}}'_D := \{i \in \mathcal{A} : \tilde{z}_i < 0\}, \tag{4.14}$$

potentially with the latter set replaced by

$$\tilde{\mathcal{V}}'_D := \{i \in \mathcal{A} : \tilde{z}_i + [S_{[\mathcal{A}]} Q^{-1} \tilde{v}]_i < 0\}. \tag{4.15}$$

Specifically, we have found it beneficial to avoid modifying the partition until the number of elements of the violated sets that have been identified (as measured by $|\tilde{\mathcal{V}}_P \cup \tilde{\mathcal{V}}_D|$) is proportional to the number of elements of the primal-dual components that violate their bounds (as measured by $|\tilde{\mathcal{V}}'_P \cup \tilde{\mathcal{V}}'_D|$). Given a parameter $\theta \in (0, 1]$, we write this condition as

$$|\tilde{\mathcal{V}}_P \cup \tilde{\mathcal{V}}_D| \geq \theta |\tilde{\mathcal{V}}'_P \cup \tilde{\mathcal{V}}'_D|. \tag{4.16}$$

Observe that Algorithm 15 yields $\alpha_{\mathcal{I}} \geq 0$ and $\beta_{\mathcal{A}} \geq 0$, from which it follows that $|\tilde{\mathcal{V}}_P \cup \tilde{\mathcal{V}}_D| \leq |\tilde{\mathcal{V}}'_P \cup \tilde{\mathcal{V}}'_D|$. This justifies the restriction that $\theta \in (0, 1]$.

4.2.2 Algorithms with Dynamic Subproblem Tolerances

We are now prepared to present our second and third inexact PDAS algorithms. For a given partition $(\mathcal{A}, \mathcal{I})$, the main idea underlying our first method—i.e., Algorithm 13 presented in §4.2.1—was to use properties of inexact subspace solutions and their corresponding residuals in order to construct explicit subsets of the violated sets $\mathcal{V}_P$ and $\mathcal{V}_D$ corresponding to the exact subspace solution, all without having to explicitly compute this exact solution. Unfortunately, however, the procedure that we proposed required an explicit upper bound for a norm of $R^{-1}_{\mathcal{I}\mathcal{I}}$, which may be expensive to compute in certain situations, especially when a tight bound is needed to identify elements of the violated sets. By contrast, the algorithms that we propose in this section do not require explicit upper bounds of this type; instead, they involve dynamic parameters either to estimate such an upper bound or control inexactness directly.

Our second algorithm, stated as Algorithm 16 below, employs a dynamic parameter that plays a role similar to the upper bound for a norm of $R_{\mathcal{I}\mathcal{I}}^{-1}$ as employed in Algorithm 13. In the worst case, this dynamic parameter will increase large enough such that it is, in fact, an upper bound for a norm of $R_{\mathcal{I}\mathcal{I}}^{-1}$ (for any $\mathcal{I}$); indeed, the convergence guarantees that we present for Algorithm 16 are based on this feature. However, we have designed the update for this dynamic parameter such that we rarely see such behavior in practice. Indeed, in practice, we often observe that the algorithm terminates for relatively small values of this dynamic parameter. As in Algorithm 13, the norm used in the optimality test in Step 5 can be any vector norm; we also add that, in the context of this algorithm, it is natural to use the same norm in Step 13. On the other hand, the norm to which we refer in Step 8 should be the norm $\|\cdot\|_p$ with $p \geq 1$ employed in Algorithm 15 for computing the values $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$.

Algorithm 16 PDAS Framework with Inexact Subspace Solutions (Dynamic)

```
1: Input an initial partition $(\mathcal{A},\mathcal{I})$, optimality tolerance $\varepsilon_{opt} > 0$, dynamic parameter
   $\gamma > 0$, update factor $\delta_\gamma > 1$, optimality tolerance history length $\bar{j} \in \mathbb{N}$, and sufficient
   reduction factor $\kappa \in (0,1)$.
2: Initialize a partition update counter $j \leftarrow 0$ and $\text{KKT}_j \leftarrow \infty$ for $j \in \{-1,\dots,-\bar{j}\}$.
3: **loop**
4:    Compute an inexact subspace solution $(\tilde{x},\tilde{y},\tilde{z})$ with residual $(\tilde{r},\tilde{t})$ by (4.7).
5:    **if** $\|\text{KKT}(\tilde{x},\tilde{y},\tilde{z})\| \leq \varepsilon_{opt}$ **then**
6:       Terminate and **return** $(\tilde{x},\tilde{y},\tilde{z})$.
7:    **end if**
8:    Compute $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$ by Algorithm 15 with input $\gamma$ (even if $\gamma \not\geq \|R_{\mathcal{I}\mathcal{I}}^{-1}\|_p$).
9:    Set $\tilde{\mathcal{V}}_P$ and $\tilde{\mathcal{V}}_D$ by (4.11) (even if $\tilde{\mathcal{V}}_P \not\subseteq \mathcal{V}_P$ and/or $\tilde{\mathcal{V}}_D \not\subseteq \mathcal{V}_D$).
10:   **if** $\tilde{\mathcal{V}}_P \cup \tilde{\mathcal{V}}_D \neq \emptyset$ **then**
11:      Choose $\mathcal{C}_P \subseteq \tilde{\mathcal{V}}_P$ and $\mathcal{C}_D \subseteq \tilde{\mathcal{V}}_D$ such that $\mathcal{C}_P \cup \mathcal{C}_D \neq \emptyset$.
12:      Set $\mathcal{A} \leftarrow (\mathcal{A}\backslash\mathcal{C}_D) \cup \mathcal{C}_P$ and $\mathcal{I} \leftarrow (\mathcal{I}\backslash\mathcal{C}_P) \cup \mathcal{C}_D$.
13:      Set $\text{KKT}_j \leftarrow \|\text{KKT}(\tilde{x},\tilde{y},\tilde{z})\|$.
14:      **if** $\text{KKT}_j \geq \kappa \max\{\text{KKT}_{j-1},\dots,\text{KKT}_{j-\bar{j}}\}$ **then**
15:         Set $\gamma \leftarrow \delta_\gamma \gamma$.
16:      **end if**
17:      Set $j \leftarrow j+1$.
18:   **end if**
19: **end loop**
```

The purpose of the sequence $\{\text{KKT}_j\}$ computed in Algorithm 16 is to monitor progress in reducing the KKT error over the sequence of iterations in which the partition is modified. Specifically, if a KKT error computed in Step 13 is not less than the most recent $\bar{j}$ such computed KKT errors, then the dynamic parameter γ is increased. As can be seen in the procedure in Algorithm 15, this has the effect of yielding larger values for $\alpha_{\mathcal{I}}$ and $\beta_{\mathcal{A}}$, which in turn has the effect of producing more conservative estimates (i.e., $\tilde{\mathcal{V}}_P$ and $\tilde{\mathcal{V}}_D$) of the violated sets (i.e., $\mathcal{V}_P$ and $\mathcal{V}_D$).

We have the following theorem related to convergence properties of Algorithm 16.

Theorem 4.2.6. *Suppose that the conditions of Theorem 4.1.3 hold for the partitions generated by Algorithm 16. Then, the following hold:*

(i) If, for any partition, repeated executions of Step 4 yield $(\tilde{r}, \tilde{t}) \to 0$, then, with $\varepsilon_{opt} > 0$, Algorithm 16 terminates after a finite number of partition updates.

(ii) If there exists a positive integer J such that, for any partition, at most J executions of Step 4 yields $(\tilde{r}, \tilde{t}) = 0$, then, with $\varepsilon_{opt} \geq 0$, Algorithm 16 terminates in a finite number of iterations. In particular, if $\varepsilon_{opt} = 0$, then Algorithm 16 terminates in a finite number of iterations with a KKT point for (4.1).

Proof. Given any partition, it follows by the conditions in either (i) or (ii) that after a finite number of executions of Step 4, the sets $\tilde{\mathcal{V}}_P$ and $\tilde{\mathcal{V}}_D$ defined by (4.11) satisfy $\tilde{\mathcal{V}}_P \cup \tilde{\mathcal{V}}_D \neq \emptyset$ (despite the fact that we may have $\tilde{\mathcal{V}}_P \not\subseteq \mathcal{V}_P$ and/or $\tilde{\mathcal{V}}_D \not\subseteq \mathcal{V}_D$). Consequently, given any partition, Algorithm 16 will eventually either terminate or a partition update will be performed. If the algorithm terminates finitely, then there is nothing left to prove. Hence, in order to derive a contradiction, suppose that an infinite number of partition updates are performed. If $\{\text{KKT}_j\} \to 0$, then, under the conditions in either (i) or (ii), the optimality condition in Step 5 eventually will be satisfied; this would cause the algorithm to terminate finitely, a contradiction to our supposition that an infinite number of partition updates are performed. Thus, we may assume that $\{\text{KKT}_j\}$ is bounded below by a positive constant, which, by the condition in Step 14, implies that $\gamma \to \infty$. However, once

$$\gamma \geq \bar{\gamma} := \max_{\mathcal{I} \subseteq \mathcal{N}} \|R_{\mathcal{I}\mathcal{I}}^{-1}\|_p, \tag{4.17}$$

it follows that we will always have $\tilde{\mathcal{V}}_P \subseteq \mathcal{V}_P$ and $\tilde{\mathcal{V}}_D \subseteq \mathcal{V}_D$, implying that convergence can be guaranteed as the same manner as in the proof of Theorem 13. Since this contradicts our supposition that an infinite number of partition updates are performed, we have that the algorithm will terminate finitely, as desired. $\square$

One important observation about Algorithm 16 is that it is nontrivial to choose an initial value for the dynamic parameter γ such that all iterations performed in the algorithm may be identical to those that would be performed by Algorithm 13. For example, assuming that it may be computed efficiently, one may consider an initial value of $\gamma \leftarrow \|R^{-1}\|_1$ (independently of any partition), but this value does not necessarily satisfy (4.17). To see this, consider the following example.

Example 4.2.7. *Let*

$$R = \begin{bmatrix} \frac{7}{6} & \frac{1}{6} & \frac{29}{12} \\ \frac{1}{6} & \frac{1}{6} & \frac{5}{12} \\ \frac{29}{12} & \frac{5}{12} & \frac{143}{24} \end{bmatrix} \quad \textit{and} \quad \mathcal{I} = \{1, 2\}.$$

One can verify that $\|R^{-1}\|_1 = \frac{31}{4} < 8 = \|R^{-1}_{\mathcal{I}\mathcal{I}}\|_1$.

Our third inexact PDAS algorithm, presented as Algorithm 17 below, employs the straightforward heuristic of defining a dynamic tolerance for the residuals in the (reduced) linear system solves that decreases as the optimization process proceeds. In this algorithm, it is reasonable to choose the norm in Step 8 as the same norm used in the KKT residual checks in Steps 5 and 12. (We also note that a relative residual test—rather than an absolute residual test—could be employed, as we do in our implementation and numerical experiments.)

We have the following convergence result for Algorithm 17.

Theorem 4.2.8. *Suppose that the conditions of Theorem 4.1.3 hold for the partitions generated by Algorithm 17. If, after a finite number of partition updates, the executions of Step 4 ensure that the condition in Step 8 only ever holds true when* $\|(\tilde{r}, \tilde{t})\| = 0$*, then, with* $\varepsilon_{opt} \geq 0$*, Algorithm 17 terminates in a finite number of iterations. In particular, if*

Algorithm 17 PDAS Framework with Inexact Subspace Solutions (Dynamic)

```
 1: Input an initial partition $(\mathcal{A},\mathcal{I})$, optimality tolerance $\varepsilon_{opt} > 0$, dynamic parameter
    $\zeta > 0$, update factor $\delta_\zeta > 1$, optimality tolerance history length $\bar{j} \in \mathbb{N}$, and sufficient
    reduction factor $\kappa \in (0,1)$.
 2: Initialize a partition update counter $j \leftarrow 0$ and $\text{KKT}_j \leftarrow \infty$ for $j \in \{-1,\dots,-\bar{j}\}$.
 3: loop
 4:   Compute an inexact subspace solution $(\tilde{x},\tilde{y},\tilde{z})$ with residual $(\tilde{r},\tilde{t})$ by (4.7).
 5:   if $\|\text{KKT}(\tilde{x},\tilde{y},\tilde{z})\| \leq \varepsilon_{opt}$ then
 6:     Terminate and return $(\tilde{x},\tilde{y},\tilde{z})$.
 7:   end if
 8:   if $\|(\tilde{r},\tilde{t})\| \leq \zeta$ then
 9:     Set $\tilde{\mathcal{V}}'_P$ and $\tilde{\mathcal{V}}'_D$ by (4.14) (even if $\tilde{\mathcal{V}}'_P \not\subseteq \mathcal{V}_P$ and/or $\tilde{\mathcal{V}}'_D \not\subseteq \mathcal{V}_D$).
10:     Choose $\mathcal{C}_P \subseteq \tilde{\mathcal{V}}'_P$ and $\mathcal{C}_D \subseteq \tilde{\mathcal{V}}'_D$ such that $\mathcal{C}_P \cup \mathcal{C}_D \neq \emptyset$.
11:     Set $\mathcal{A} \leftarrow (\mathcal{A}\backslash\mathcal{C}_D) \cup \mathcal{C}_P$ and $\mathcal{I} \leftarrow (\mathcal{I}\backslash\mathcal{C}_P) \cup \mathcal{C}_D$.
12:     Set $\text{KKT}_j \leftarrow \|\text{KKT}(\tilde{x},\tilde{y},\tilde{z})\|$.
13:     if $\text{KKT}_j \geq \kappa \max\{\text{KKT}_{j-1},\dots,\text{KKT}_{j-\bar{j}}\}$ then
14:       Set $\zeta \leftarrow \zeta/\delta_\zeta$.
15:     end if
16:     Set $j \leftarrow j+1$.
17:   end if
18: end loop
```

$\varepsilon_{opt} = 0$, *then Algorithm 17 terminates in a finite number of iterations with a KKT point for* (4.1).

Proof. Under the conditions of the theorem, it follows that after a finite number of partition updates the algorithm behaves as if (exact) subspace solutions were computed via (5.3). Hence, the result follows similarly as Theorem 4.1.3. □

Despite the fact that Algorithm 17 employs a straightforward heuristic for controlling inexactness and has the advantage that a factorization of Q is not required, it has two key disadvantages vis-à-vis Algorithms 13 and 16. First, as a practical matter, our experience suggests that it is much more difficult to choose (initial) values for ζ and δ_ζ that lead to good performance on a wide range of problems. Second, our convergence guarantee for Algorithm 17 is significantly weaker than those for Algorithms 13 and 16. The following example illustrates why it is not possible to obtain the same convergence guarantees in Theorem 4.2.8 as we have stated in Theorems 4.2.2 and 4.2.6. In particular, the example shows that in order to ensure that $\tilde{\mathcal{V}}'_P \subseteq \mathcal{V}_P$, $\tilde{\mathcal{V}}'_D \subseteq \mathcal{V}_D$, and $\tilde{\mathcal{V}}'_P \cup \tilde{\mathcal{V}}'_D \neq \emptyset$, one may need a linear system residual that is exactly zero.

Example 4.2.9. *Let*

$$m = 0, \quad H = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}, \quad \text{and} \quad u = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

With $(\mathcal{A}, \mathcal{I}) = (\{1,2\},\{3\})$, it follows from (5.3) that the subspace solution is

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{and} \quad z = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix},$$

from which it follows that $\mathcal{V}_P = \emptyset$ and $\mathcal{V}_D = \{1\}$. In particular, $(\mathcal{A}, \mathcal{I})$ is suboptimal. Moreover, from (4.7), any inexact subspace solution $(\tilde{x}, \tilde{z})$ satisfies

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \tilde{x}_3 - x_3 \end{bmatrix} + \begin{bmatrix} \tilde{z}_1 - z_1 \\ \tilde{z}_2 - z_2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \tilde{r}_3 \end{bmatrix},$$

which implies that $(\tilde{z}_1 - z_1) = 0$ and $\tilde{r}_3 = 2(\tilde{x}_3 - x_3) = 2(\tilde{z}_2 - z_2)$. Hence, in order to have $\tilde{\mathcal{V}}'_P \subseteq \mathcal{V}_P$, $\tilde{\mathcal{V}}'_D \subseteq \mathcal{V}_D$, and $\tilde{\mathcal{V}}'_P \cup \tilde{\mathcal{V}}'_D \neq \emptyset$, one must have

$$\begin{aligned} \tilde{x}_3 &= x_3 + \tfrac{1}{2}\tilde{r}_3 = 1 + \tfrac{1}{2}\tilde{r}_3 \leq 1 \\ \text{and} \quad \tilde{z}_2 &= z_2 + \tfrac{1}{2}\tilde{r}_2 = 0 + \tfrac{1}{2}\tilde{r}_3 \geq 0, \end{aligned}$$

i.e., one must have $\tilde{r}_3 = 0$. (On the other hand, one can verify that with $\gamma = 1 \geq \|H_{33}^{-1}\|_1$, Algorithms 13 and 16 obtain $\tilde{\mathcal{V}}_P = \emptyset = \mathcal{V}_P$ and $\tilde{\mathcal{V}}_D = \{1\} = \mathcal{V}_D$ for $|\tilde{r}_3| < 3$.)

4.3 An Implementation

We have written implementations of Algorithms 12, 13, 16, and 17 along with the subroutines described as Algorithms 14 and 15. We discuss common and distinguishing details of the implementations in this section.

All algorithms are implemented in a single piece of software in Python 2.7.3. The software uses the infrastructure in `cvxopt` for matrix storage and manipulation, as well as the implementation of MINRES [98] provided in `Scipy` for (approximately) solving the linear systems (4.2b) and (4.7b). For Algorithms 13 and 16, the matrix Q is factored at the start of each run using `cvxopt`'s interface to `LAPACK`; specifically, the `sytrf` subroutine is called to compute an LDL^T factorization of Q, the factors of which are employed to compute products with Q^{-1} as they are needed.

Our use of MINRES as the iterative solver for the linear systems (4.2b) and (4.7b) is not required; any iterative method for solving symmetric indefinite systems could be employed. It should also be noted that while preconditioning would be a critical aspect of any efficient implementation of either of our algorithms, we did not implement a preconditioner in our software. We claim that this is reasonable as the purpose of our numerical experiments is merely to illustrate the convergence behavior of our algorithms despite inexactness in the subspace solutions; in particular, the speed at which these inexact subspace solutions are obtained is not a focus of our experiments, which only serve as a "proof of concept" for our algorithms.

In all algorithms, the initial point provided to MINRES in the first PDAS iteration is a randomly generated vector, whereas, in subsequent PDAS iterations, the initial point is set by extracting the corresponding elements of the primal-dual solution corresponding to the previous PDAS iterate. For Algorithm 12 in which "exact" solutions are required, each run of MINRES terminates once the ℓ_∞-norm of the residual for the linear system (4.2b) is reduced below a prescribed tolerance $\epsilon_{num} > 0$. Similarly, for Algorithm 17, MINRES terminates once either (4.13) holds (with ϵ_{res} replaced by ζ) or the ℓ_∞-norm of the residual for the linear system (4.7b) is reduced below ϵ_{num}. Finally, for Algorithms 13 and 16, MINRES terminates either once (4.13) and (4.16) both hold or the ℓ_∞-norm of the residual for the linear system (4.7b) is reduced below ϵ_{num}. (The only exceptions occur when $\|(\tilde{r},\tilde{t})\|_\infty \leq \epsilon_{num}$, but $|\tilde{\mathcal{V}}_P \cup \tilde{\mathcal{V}}_D| = |\tilde{\mathcal{V}}'_P \cup \tilde{\mathcal{V}}'_D| = 0$ and $\|\text{KKT}(\tilde{x},\tilde{y},\tilde{z})\|_\infty > \epsilon_{opt}$, in which case MINRES is forced to continue.)

In terms of Algorithms 13 and 16, the evaluation of vectors in Algorithm 15 requires products with Q^{-1} (i.e., solves with the factors of Q), meaning that it is not economical to run this subroutine after every MINRES iteration. Hence, our software performs 100 MINRES iterations in Step 3 of Algorithm 13 and Step 4 of Algorithm 16. In both cases, we use $p = 1$ in Algorithm 15.

Finally, for Algorithm 16, we implemented an additional strategy for updating γ that utilizes intermediate vectors computed by MINRES. (The purpose of this strategy is to use problem information to quickly adjust γ if the initial value is set too low for a given run of the algorithm.) In particular, with an intermediate solution $\tilde{x}_{\mathcal{I}}$ and product $R_{\mathcal{I}\mathcal{I}}\tilde{x}_{\mathcal{I}}$, both provided at no extra cost by MINRES, we set

$$\gamma \leftarrow \max\left\{\gamma, \frac{\|\tilde{x}_{\mathcal{I}}\|_1}{\|R_{\mathcal{I}\mathcal{I}}\tilde{x}_{\mathcal{I}}\|_1}\right\}.$$

This update is motivated by the fact that

$$\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1 = \max_{\|x\|_1=1} \|R_{\mathcal{I}\mathcal{I}}x\|_1^{-1}. \tag{4.18}$$

4.4 Numerical Results

In this section, we report on the performance of our implementations of Algorithms 12, 13, 16, and 17 when they were employed to solve two optimal control test problems. In fact, the problems we consider are the same as those considered in [69]. We remark at the outset that, for consistency with notation commonly used in the context of optimal control, the decision variables in the test problems presented in this section are a state variable y and a control variable u. In addition, we use $x = (x_1, x_2)$ to denote the coordinate axes in $\mathbb{R}^2$. Overall, the reader should be aware that the pair (x, y) in this section should not be confused with the primal-dual variable pair (x, y) used in the previous sections.

Given a domain $\Omega \in \mathbb{R}^2$, reference function $z \in L^2(\Omega)$, upper bound function $\psi \in$

$L^2(\Omega)$, and regularization parameter $\beta > 0$, we consider the test problems

$$\begin{aligned} &\min_{y,u}\ \tfrac{1}{2}\|y - z\|^2_{L^2(\Omega)} + \tfrac{\beta}{2}\|u\|^2_{L^2(\Omega)} \\ &\text{s.t.} \begin{cases} -\Delta y = u & \text{in } \ \Omega \\ y = 0 & \text{on } \ \partial\Omega \\ u \le \psi & \text{in } \ \Omega, \end{cases} \end{aligned} \tag{4.19}$$

and, with n denoting the unit outer normal to Ω along $\partial\Omega$,

$$\begin{aligned} &\min_{y,u}\ \tfrac{1}{2}\|y - z\|^2_{L^2(\Omega)} + \tfrac{\beta}{2}\|u\|^2_{L^2(\partial\Omega)} \\ &\text{s.t.} \begin{cases} -\Delta y + y = 0 & \text{in } \ \Omega \\ \frac{\partial y}{\partial n} = u & \text{on } \ \partial\Omega \\ u \le \psi & \text{on } \ \partial\Omega. \end{cases} \end{aligned} \tag{4.20}$$

In particular, as in [69], we let

$$\Omega = [0,1]^2, \quad z(x_1, x_2) = \sin(5x_1) + \cos(4x_2), \quad \psi = 0, \quad \text{and} \quad \beta = 10^{-5}.$$

In order to illustrate the performance of our implementation of our algorithms on instances of various sizes, we generated discretized versions of problems (4.19) and (4.20) at various levels of discretization. In particular, we generated instances of both problems with the numbers of grid points along each dimension in the set $\{20, 40, 60, 80, 100\}$. A five-point-star discretization of Δ was used and the functions z, ψ, y, and u were discretized by means of grid functions at the nodal points. It is easily verified that all of the resulting problem instances have the form (4.1) satisfying Assumption 4.1.1. Table 4.1 contains the sizes of each problem instance in terms of the numbers of grid points per dimension (`g`), variables (`n`), and equality constraints (`m`). For each instance of each problem, all algorithms were initialized with the same initial partition, which was generated randomly for all problem instances.

For our experiments, we used the input parameters in Table 4.2. These values were

Table 4.1: Problem sizes.

	(4.19)		(4.20)	
g	n	m	n	m
20	800	400	560	480
40	3200	1600	1920	1760
60	7200	3600	4080	3840
80	12800	6400	7040	6720
100	20000	10000	10800	10400

used as they lead to good performance in our experiments, though it should be noted that, in any application, these parameters should be tuned for optimal performance. As for the dynamic parameter $\gamma > 0$ in Algorithm 16, it was initialized to 10^2, and as for the dynamic parameter $\zeta > 0$ in Algorithm 17, it was initialized to ϵ_{res}.

Table 4.2: Input parameters for our implementations of Algorithms 12, 13, 16, and 17.

Parameter	Value	Algorithm(s)
ϵ_{opt}	10^{-6}	12, 13, 16, 17
ϵ_{num}	10^{-6}	12, 13, 16, 17
ϵ_{res}	$\{10^{-2}, 10^{-3}\}$	13, 16, 17
θ	0.5	13, 16
δ_γ	1.2	16
δ_ζ	1.2	17
$\bar{j}$	5	16, 17
κ	0.9	16, 17

The performance of the algorithms in solving problem (4.19) is reported in Tables 4.3–4.9 below. In each table, we report the numbers of grid points per dimension and PDAS iterations required before termination (`Iter.`) for each instance. We also report, to illustrate the accuracy with which the (reduced) linear systems were solved in each run of the algorithm, the minimum (`min`), median (`med`), and maximum (`min`) relative residual (`Rel.Res.`) and absolute residual (`Abs.Res.`) over all PDAS iterations. (It should be noted that, due to diagonal dominance of the matrix involved, all runs of Algorithm 13 required only one Krylov iteration per PDAS iteration to compute the upper bound on $\|R_{\mathcal{I}\mathcal{I}}^{-1}\|_1$ in Algorithm 14. Hence, this subroutine did not lead to a significant increase in computational expense.)

Table 4.3: Algorithm 12 when solving problem (4.19).

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	7	2.82e-07	1.88e-06	2.94e-05	7.57e-07	9.22e-07	9.98e-07
40	8	2.44e-07	7.74e-06	5.08e-03	7.30e-07	9.88e-07	9.99e-07
60	8	2.92e-07	1.75e-05	1.71e-03	8.75e-07	9.97e-07	1.00e-06
80	8	3.20e-07	2.96e-05	4.97e-03	9.61e-07	9.97e-07	1.00e-06
100	8	3.22e-07	3.55e-05	6.99e-03	9.66e-07	9.99e-07	1.00e-06

Table 4.4: Algorithm 13 when solving problem (4.19) with $\epsilon_{res} = 10^{-2}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	10	1.71e-04	8.47e-04	8.96e-03	8.51e-07	2.84e-04	2.39e-02
40	12	3.21e-04	3.41e-03	1.31e-02	9.63e-07	2.44e-04	2.89e-02
60	11	8.70e-04	3.33e-03	5.08e-01	9.95e-07	9.98e-05	2.55e-02
80	12	2.10e-03	4.67e-03	8.11e-02	9.95e-07	8.76e-05	2.77e-02
100	13	3.30e-03	8.44e-03	4.81e-01	9.99e-07	1.21e-04	2.58e-02

Table 4.5: Algorithm 13 when solving problem (4.19) with $\epsilon_{res} = 10^{-3}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	9	1.71e-04	6.17e-04	9.20e-04	8.51e-07	2.28e-04	1.90e-03
40	10	3.24e-04	7.10e-04	1.31e-02	9.63e-07	4.86e-05	2.24e-03
60	10	1.60e-04	8.43e-04	5.08e-01	9.95e-07	2.37e-05	2.14e-03
80	9	2.26e-04	6.95e-04	9.49e-03	9.95e-07	1.53e-05	2.98e-03
100	9	2.89e-04	7.53e-04	2.65e-02	9.99e-07	1.02e-05	2.94e-03

Table 4.6: Algorithm 16 when solving problem (4.19) with $\epsilon_{res} = 10^{-2}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	9	3.05e-05	6.25e-03	8.96e-03	8.51e-07	1.26e-03	2.39e-02
40	9	6.43e-04	8.75e-03	1.03e-02	9.97e-07	4.34e-04	2.89e-02
60	10	1.91e-03	6.91e-03	4.65e-02	9.95e-07	2.52e-04	2.55e-02
80	9	2.29e-03	6.26e-03	9.39e-03	9.95e-07	1.24e-04	2.77e-02
100	9	3.40e-03	8.36e-03	9.31e-03	1.00e-06	1.79e-04	2.58e-02

Table 4.7: Algorithm 16 when solving problem (4.19) with $\epsilon_{res} = 10^{-3}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	8	1.71e-04	5.96e-04	6.66e-04	8.51e-07	9.37e-05	1.90e-03
40	8	2.19e-04	5.21e-04	1.03e-02	9.97e-07	5.42e-05	2.24e-03
60	8	1.26e-04	4.52e-04	1.72e-03	9.95e-07	2.38e-05	2.14e-03
80	8	3.87e-04	6.47e-04	5.00e-03	9.95e-07	2.09e-05	2.98e-03
100	9	2.35e-04	7.33e-04	4.81e-01	9.99e-07	1.50e-05	2.94e-03

Table 4.8: Algorithm 17 when solving problem (4.19) with $\epsilon_{res} = 10^{-2}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	11	5.38e-04	9.74e-03	1.07e-02	8.51e-07	1.05e-03	2.87e-02
40	9	5.26e-03	9.80e-03	9.99e-03	9.97e-07	4.58e-04	2.93e-02
60	10	9.39e-03	9.93e-03	9.55e-02	9.95e-07	2.86e-04	2.97e-02
80	11	9.53e-03	9.98e-03	6.26e-01	9.95e-07	2.18e-04	2.97e-02
100	10	9.51e-03	9.97e-03	7.66e-01	1.00e-06	2.22e-04	2.99e-02

Table 4.9: Algorithm 17 when solving problem (4.19) with $\epsilon_{res} = 10^{-3}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	8	7.35e-04	9.49e-04	1.07e-02	8.51e-07	2.80e-04	2.67e-03
40	9	3.38e-04	9.61e-04	1.16e-01	9.63e-07	8.33e-05	2.87e-03
60	9	9.46e-04	9.82e-04	9.39e-03	9.95e-07	4.43e-05	2.93e-03
80	8	9.62e-04	9.97e-04	1.60e-03	9.95e-07	3.29e-05	2.99e-03
100	9	9.00e-04	9.92e-04	4.81e-01	9.99e-07	2.40e-05	3.00e-03

The results in Tables 4.3–4.9 provide evidence for the computational benefits of allowing inexactness in the subspace solutions. In particular, when applied to solve instances of various sizes, Algorithms 13, 16, and 17 converge in a number of PDAS iterations that is comparable to that required by Algorithm 12; however, this convergence is attained with substantially larger relative and absolute residuals in the (reduced) linear system solves. With a reasonable preconditioner for the linear systems, such larger relative and absolute residuals would be obtainable with significantly fewer Krylov iterations, potentially yielding significant savings in computational expense.

The performance of the algorithms in solving instances of problem (4.20) is reported in Tables 4.10–4.16. (It is worthwhile to note that, again, the computational expense of applying Algorithm 14 was negligible in the context of Algorithm 13.)

Table 4.10: Algorithm 12 when solving problem (4.20).

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	8	2.06e-07	4.60e-07	2.97e-06	7.81e-07	9.66e-07	1.00e-06
40	12	1.33e-07	4.02e-07	2.56e-06	8.85e-07	9.91e-07	1.00e-06
60	17	9.02e-08	3.01e-07	2.78e-06	9.53e-07	9.94e-07	1.00e-06
80	22	7.10e-08	2.61e-07	2.86e-06	9.65e-07	9.92e-07	9.99e-07
100	26	5.75e-08	2.37e-07	3.05e-06	9.64e-07	9.97e-07	1.00e-06

Table 4.11: Algorithm 13 when solving problem (4.20) with $\epsilon_{res} = 10^{-2}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	9	3.67e-07	1.47e-03	8.22e-03	1.18e-07	9.78e-04	3.79e-02
40	20	9.93e-08	5.50e-03	9.96e-03	3.91e-08	2.90e-03	3.69e-02
60	38	6.58e-07	3.06e-03	9.49e-03	2.18e-07	1.95e-03	3.13e-02
80	50	6.25e-07	3.50e-03	9.92e-03	1.93e-07	1.60e-03	3.53e-02
100	72	2.30e-06	5.60e-03	9.97e-03	5.52e-07	1.57e-03	2.21e-02

Table 4.12: Algorithm 13 when solving problem (4.20) with $\epsilon_{res} = 10^{-3}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	8	3.53e-07	7.36e-06	8.19e-04	1.18e-07	1.66e-05	3.78e-03
40	14	1.11e-07	5.68e-04	9.95e-04	3.91e-08	1.05e-03	4.79e-03
60	21	6.58e-07	7.78e-04	9.93e-04	2.18e-07	1.25e-03	4.16e-03
80	34	5.63e-07	7.33e-04	9.98e-04	1.93e-07	8.59e-04	3.45e-03
100	39	1.80e-06	7.23e-04	9.98e-04	5.52e-07	1.38e-03	4.05e-03

Table 4.13: Algorithm 16 when solving problem (4.20) with $\epsilon_{res} = 10^{-2}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	8	3.53e-07	1.19e-05	8.22e-03	1.18e-07	2.46e-05	3.79e-02
40	16	1.09e-07	1.19e-04	7.99e-03	3.91e-08	1.02e-04	3.69e-02
60	21	2.45e-07	3.14e-05	6.79e-03	2.18e-07	9.16e-05	3.13e-02
80	28	2.70e-07	2.27e-05	4.96e-03	1.93e-07	4.81e-05	2.29e-02
100	33	2.30e-07	5.09e-06	3.89e-03	5.52e-07	1.47e-05	1.79e-02

Table 4.14: Algorithm 16 when solving problem (4.20) with $\epsilon_{res} = 10^{-3}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	8	3.53e-07	1.19e-05	8.19e-04	1.18e-07	2.46e-05	3.78e-03
40	16	1.11e-07	4.67e-05	8.40e-04	3.91e-08	5.46e-05	3.88e-03
60	21	2.45e-07	2.96e-05	9.02e-04	2.18e-07	9.16e-05	4.16e-03
80	28	2.70e-07	2.23e-05	7.49e-04	1.93e-07	4.81e-05	3.45e-03
100	33	2.30e-07	4.76e-06	8.78e-04	5.52e-07	1.47e-05	4.05e-03

Table 4.15: Algorithm 17 when solving problem (4.20) with $\epsilon_{res} = 10^{-2}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	19	3.06e-06	6.72e-03	9.96e-03	9.16e-07	9.48e-03	4.58e-02
40	31	2.67e-06	3.34e-03	9.98e-03	9.52e-07	4.53e-03	4.60e-02
60	44	2.76e-06	2.78e-03	1.00e-02	9.84e-07	1.23e-03	1.18e-01
80	62	2.88e-06	1.93e-03	9.98e-03	9.94e-07	1.47e-03	1.26e-01
100	69	2.90e-06	1.12e-03	1.00e-02	9.77e-07	8.44e-04	1.01e-01

Table 4.16: Algorithm 17 when solving problem (4.20) with $\epsilon_{res} = 10^{-3}$.

		Rel.Res.			Abs.Res.		
g	Iter.	min	med	max	min	med	max
20	9	2.77e-06	9.65e-04	9.99e-04	9.16e-07	1.69e-03	5.85e-03
40	16	2.67e-06	5.77e-04	9.99e-04	9.52e-07	1.64e-03	9.19e-03
60	40	2.81e-06	4.59e-04	9.99e-04	9.84e-07	1.05e-03	4.61e-03
80	37	2.91e-06	2.12e-04	1.00e-03	9.94e-07	8.53e-04	4.60e-03
100	40	2.91e-06	1.34e-04	1.00e-03	9.77e-07	4.37e-04	4.60e-03

In contrast to those for problem (4.19), it is clear from the results for problem (4.20) that the number of PDAS iterations required before termination is mesh dependent. Thus, these results more clearly illustrate the trade-off between saving per-iteration computational costs with overall costs, as well as the sensitivity of parameter choices in our inexact PDAS algorithms. In particular, while certain runs of our algorithms involve less accurate (reduced) linear system solutions, this may may lead to a significant increase in the number of PDAS iterations required before termination. With an effective preconditioner, one should still expect to attain reduced overall computational costs by allowing inexactness in the subproblem solutions, but this may require careful selection of algorithmic parameters such as ϵ_{res}.

4.5 Concluding Remarks

In this chapter, we have proposed a set of primal-dual active-set algorithms for solving certain structured quadratic optimization problems. The distinguishing feature of the algorithms is that they attain global convergence guarantees while allowing inexactness in the (reduced) linear system solves in each iteration. In each iteration, the first algorithm sets a requirement for the accuracy in the linear system solve through the use of subroutines for computing an upper bound for the inverse of a particular submatrix, whereas the second and third make use of dynamic algorithmic parameters for controlling the level of inexactness in each iteration. We have implemented our algorithms and have provided the results of numerical experiments on a pair of optimal control test problems, illustrating that our algorithms can converge in a similar number of PDAS iterations as an algorithm that employs "exact" linear system solves, but with much lower per-iteration computational costs.

One potential topic of research would be to investigate possibilities of incorporating inexactness into the PDAS framework of §3 for generally-constrained convex QPs. In this chapter, our proposed approach ensures global convergence by monitoring the level of inexactness and bounding the solution of the linear systems. We point out that the main obstacle of extending this approach to the PDAS framework of §3 is that, when the subproblem is a BQP (for example, when $\mathcal{U}$ is nonempty) inferring a bound of the optimal solution is challenging.

Chapter 5

PDAS for Machine Learning Applications

In this chapter, we demonstrate how PDAS methods can be adapted to solve certain large-scale optimization problems arising in machine learning. In particular, we customize PDAS methods for solving Isotonic Regression (IR) and related Trend Filtering (TR) problems. We reveal that adapting PDAS techniques to solve these problems leads to novel and extremely efficient algorithms that can outperform state-of-the-art approaches.

Isotonic regression is a non-parametric method for fitting an arbitrary monotone function to a dataset [3, 20] that has recently gained favor as a calibration method for supervised learning [54, 91, 96, 121]. A well-known and efficient method for solving the IR problems is the Pool Adjacent Violators (PAV) algorithm [12]. This method is easily implemented and enjoys a convergence guarantee with a work complexity of $\mathcal{O}(n)$ where n is the dimension of the dataset. A drawback of the PAV algorithm in large-scale settings, however, is that it is inherently sequential. Consequently, in order to exploit parallelism, one has to resort to decomposing the IR problem [80], where deciding the number of processors is nontrivial. For example, a recent Spark implementation of a parallelized PAV method suffers from significant overhead [122]. In addition, since the PAV algorithm must be initialized from a particular starting point, it cannot be warm-started—a fact that is especially detrimental when a sequence of IR problems need to be solved [109] or in the online setting where data points are constantly added. As an alternative, we propose a primal-dual active-set (PDAS) method for solving the IR problems. Our PDAS algorithm also has a convergence guarantee for IR and a work complexity of $\mathcal{O}(n)$, but can be warm-started and is easily parallelized. We also provide PDAS algorithm variants for a related class of trend filtering problems. Despite the existence of interior point methods, some specialized ADMM methods, and proximal methods that are applicable to TR problems, active-set methods are able to generate very accurate solutions thus are favored for certain applications such as switch points identification and time series segmentation. The results of numerical experiments are provided for all algorithm variants which show great practical strengths of PDAS methods.

This chapter is organized in the following order. In §5.1, we give the problem descriptions of IR and TF to which we have applied PDAS methods. In §5.2, we summarize and compare the well-known Pool Adjacent Violators (PAV) algorithm and our proposed primal-dual active-set (PDAS) method for solving IR problems. PDAS variants (with safeguards) for solving related TF problems are presented in §5.3. We report the experimental results as well as our discovery in §5.4. Finally, our concluding remarks are provided in §5.5.

5.1 Problem Descriptions

Isotonic Regression

We consider the isotonic regression (IR) problem

$$\min_{\theta \in \mathbb{R}^n} \frac{1}{2} \sum_{i=1}^{n} \omega_i (y_i - \theta_i)^2 \quad \text{subject to} \quad \theta_1 \leq \ldots \leq \theta_n, \tag{IR}$$

where $y \in \mathbb{R}^n$ represents observed data and $\omega \in \mathbb{R}^n_+$ represents weights for the data fitting term. The goal of this optimization problem formulation is to determine a monotonically increasing step function that matches the observed data as closely as possible in a sense of distance defined by ω.

Trend Filtering

Problem (IR) can be viewed as a special case of the trend filtering problem

$$\min_{\theta \in \mathbb{R}^n} \phi(\theta), \quad \text{where} \quad \phi(\theta) = f(\theta) + \lambda g(\theta), \tag{TF}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is smooth and convex, $\lambda > 0$ is a regularization parameter, and the regularization function $g : \mathbb{R}^n \mapsto \mathbb{R}$ is convex but not necessarily smooth. In trend filtering or time series segmentation, f is usually chosen to measure the distance between θ and y while g is a function that imposes desired properties on the solution θ. A typical trend filtering problem has the form

$$f(\theta) = \frac{1}{2}\sum_{i=1}^{n} \omega_i (y_i - \theta_i)^2 \quad \text{with} \quad g(\theta) = \|D\theta\|_1 \quad \text{or} \quad g(\theta) = \|(D\theta)_+\|_1,$$

where D is a first-order (or higher) difference operator and $(\gamma)_+ = \max\{\gamma, 0\}$ (componentwise). Specifically, as in [82], a k-th order difference matrix $D^{(k,n)} \in \mathbb{R}^{(n-k)\times n}$ is defined recursively as $D^{(k,n)} = D^{(1,n-k+1)} D^{(k-1,n)}$. The first and second order difference matrix $D^{(1,n)}$ and $D^{(2,n)}$ is defined, respectively, as

$$D^{(1,n)} = \begin{bmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & \ddots & \ddots & \\ & & & 1 & -1 \end{bmatrix}, \text{ and } D^{(2,n)} = \begin{bmatrix} 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \end{bmatrix}.$$

Note that when $g(\theta) = \|(D^{(1,n)}\theta)\|_1$ and λ is sufficiently large, solving (TF) gives also the solution of (IR).

5.2 Algorithms for Isotonic Regression

We describe and compare two efficient algorithms for solving problem (IR); in particular, the existing PAV and our proposed PDAS algorithms are described and their corresponding theoretical properties are summarized in §5.2.1 and §5.2.2, respectively. After that, a comparison between these two algorithms is conducted in §5.2.3. Throughout this and the subsequent sections, we borrow the following notation from [12] in the algorithm descriptions.

Notation Let J represent a partition of the variable indices $\{1, 2, \ldots, n\}$ into ordered blocks $\{B_1, B_2, \ldots\}$ where each block consists of consecutive indices, i.e., each block has the form $\{p, \ldots, q\}$ for $p \leq q$. The immediate predecessor (successor) of block B is denoted as B_- (B_+). By convention B_- (B_+) equals $\emptyset$ when B is the initial (final) block. The weighted average of the elements of y in block B is denoted $\mathrm{Av}(B) := (\sum_{i=p}^{q} \omega_i y_i)/(\sum_{i=p}^{q} \omega_i)$. For each index $i \in B = \{p, \ldots, q\}$, we define the "lower" and "upper" sets

$$L_i(J) = \{p, p+1, \ldots, i\} \quad \text{and} \quad U_i(J) = \{i+1, i+2, \ldots, q\}.$$

Hereinafter, we shall use L_i and U_i for brevity when their dependence on a particular J is clear.

5.2.1 The PAV Algorithm for Isotonic Regression

The Pool Adjacent Violators (PAV) algorithm has long been studied [12, 3, 111, 58, 20, 80]. It is usually the first choice for solving isotonic regression problems due to its simplicity as well as its practical performance. We describe briefly the PAV algorithm, state its main theoretical properties, and discuss its important features in this section. To start with, we rephrase the description of the PAV algorithm in [12], where the algorithm is shown to replicate a dual active-set method for quadratic optimization.

The main idea of Algorithm 18 can be understood as follows. Initially, each index is represented by a separate block. The algorithm then sequentially visits all blocks, merging a block with its successor whenever a "violator" is met, i.e., whenever a block has a weighted average greater than its successor. Once any merge occurs, the algorithm searches backwards to perform subsequent merges in order to ensure that, at the end of any **loop** iteration, no violators exist up to the furthest visited block. Once all blocks have been visited, no violator exists and the solution θ will be monotonically increasing.

Algorithm 18 PAV for Isotonic Regression

```
1: Input the initial partition $J = \{\{1\}, \dots, \{n\}\}$ and set $C = \{1\}$
2: loop
3:   if $C_+ = \emptyset$ then
4:     Terminate and return $\theta_i = \text{Av}(B)$ for each $i \in B$ for each $B \in J$
5:   end if
6:   if $\text{Av}(C) \le \text{Av}(C_+)$ then
7:     Set $C \leftarrow C_+$
8:   else
9:     Set $J \leftarrow \big(J \backslash \{C, C_+\}\big) \cup (C \cup C_+)$ and $C \leftarrow C \cup C_+$
10:    while $\text{Av}(C_-) > \text{Av}(C)$ and $C_- \neq \emptyset$ do
11:      Set $J \leftarrow \big(J \backslash \{C_-, C\}\big) \cup (C_- \cup C)$ and $C \leftarrow C_- \cup C$
12:    end while
13:  end if
14: end loop
```

An impressive property of Algorithm 18 is that by storing an intermediate value for each block and showing that at most n merge operations may occur, one obtains an efficient implementation that solves problem (IR) within $\mathcal{O}(n)$ elementary arithmetic operations [58]. Due to this fact and its good practical performance, Algorithm 18 has been popular since its invention. However, we argue that the PAV algorithm does have critical drawbacks when it comes to solving large-scale problems of interest today. First, Algorithm 18 must be initialized with $J = \{\{1\}, \dots, \{n\}\}$, which is unfortunate when one has a better initial partition, such as when one is solving a sequence of related instances of (IR) [109]. In addition, the sequential nature of PAV makes it difficult to leverage multi-processor infrastructures.

5.2.2 The PDAS Algorithm for Isotonic Regression

Primal-dual active-set (PDAS) methods have been proposed in the literature for solving Linear Complementarity Problems (LCPs) [1], bound-constrained QPs (BQPs) [69], and more recently generally-constrained QPs [27]. To our knowledge, however, the application and theoretical analysis of a PDAS method for solving problem (IR) has not previously been studied.

In this section, we propose a PDAS method designed exclusively for (IR) and discuss its theoretical guarantees and practical benefits. We first reveal the relationship between problem (IR) and a special class of convex BQPs for which a primal-dual active-set (PDAS) method is known to be well-suited. We then propose our PDAS algorithm tailored for solving (IR). Finally, the complexity of our proposed algorithm is analyzed and its key features and properties are discussed.

Relationship Between IR and Convex BQP

The problem formulation of (IR) is a convex QP of very special structure. In order to explore further the properties of this problem that might be useful for PDAS methods, we first give the dual problem of (IR). Let $\Omega = \operatorname{diag}(\omega_1, \ldots, \omega_n)$ be the diagonal weight matrix, the dual problem of (IR) then has the form

$$\min_{z \in \mathbb{R}^{n-1}_{+}} \frac{1}{2} z^T D\Omega^{-1} D^T z - y^T D^T z, \tag{BCQP}$$

where

$$D\Omega^{-1}D^T = \begin{bmatrix} \frac{2}{\omega_1} & -\frac{1}{\omega_2} & 0 & \cdots & 0 \\ -\frac{1}{\omega_2} & \frac{2}{\omega_2} & -\frac{1}{\omega_3} & \cdots & 0 \\ 0 & -\frac{1}{\omega_3} & \frac{2}{\omega_3} & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & -\frac{1}{\omega_{n-2}} \\ 0 & 0 & \cdots & -\frac{1}{\omega_{n-2}} & \frac{2}{\omega_{n-1}} \end{bmatrix}, \quad \text{and} \quad Dy = \begin{bmatrix} y_1 - y_2 \\ y_2 - y_3 \\ \vdots \\ y_{n-1} - y_n \end{bmatrix}.$$

Since $D\Omega^{-1}D^T$ is positive definite with non-positive off-diagonal entries, it is an M-matrix, meaning that (BCQP) has a form for which a PDAS method is well-suited [69]. In descriptions of PDAS methods for BQP such as that in [69], the notion of a partition corresponds to a division of the index set for z into "active" and "inactive" sets. There is a one-to-one correspondence between a partition of (BCQP) and that of (IR); specifically, the non-zero indices in z correspond to the boundaries of the blocks of a partition for problem (IR). We state the following convergence result of applying a PDAS method to the dual problem (BCQP) since it is paramount for our convergence and complexity analysis of the proposed PDAS algorithm for (IR).

Theorem 5.2.1 (Theorem 3.2, [69])**.** *If the PDAS method from [69] is applied to solve problem* (BCQP)*, then the iterate sequence $\{z_k\}$ is nondecreasing, has $z_k \geq 0$ for all $k \geq 1$, and converges to the optimal solution of* (BCQP)*.*

A PDAS Algorithm for Isotonic Regression

One can apply the PDAS method from [69] to solve (IR) by applying the approach to its dual (BCQP). However, a straightforward application would fail to exploit the special structure of problem (IR). Algorithm 19, on the other hand, generates the same sequence of iterates as the PDAS method of [69], but is written in a much more computationally efficient form.

Algorithm 19 PDAS for Isotonic Regression

```
1: Input an initial partition $J_0$
2: For each $i \in B = \{p, \ldots, q\} \in J_0$, set
     $$\theta_i \leftarrow \frac{\sum_{i \in B} \omega_i y_i}{\sum_{i \in B} \omega_i} \quad \text{and} \quad z_i \leftarrow \begin{cases} \omega_i(y_i - \theta_i) & \text{if } i = p \\ 0 & \text{if } i = q \neq n \\ z_{i-1} + \omega_i(y_i - \theta_i) & \text{otherwise} \end{cases}$$
3: Initialize $J_1 \leftarrow J_0$
4: for each $i \in B \in J_1$ with $z_i < 0$, set $J_1 \leftarrow (J_1 \backslash B) \cup \{L_i, U_i\}$
5: for each $B \in J_1$, set $\alpha_B \leftarrow \sum_{i \in B} \omega_i y_i$, $\beta_B \leftarrow \sum_{i \in B} \omega_i$, $\mu_B \leftarrow \alpha_B / \beta_B$, and $\theta_i \leftarrow \mu_B$ for all $i \in B$
6: for $k = 1, 2, \ldots$ do
7:     Initialize $J_{k+1} \leftarrow J_k$
8:     for each $\{B_s, \ldots, B_t\} \subseteq J_k$ with $\mu_{B_{s-1}} \leq \mu_{B_s} > \cdots > \mu_{B_t} \leq \mu_{B_{t+1}}$
           $$\text{Let } N \leftarrow \bigcup_{j=s}^{t} B_j \text{ and update } J_{k+1} \leftarrow (J_{k+1} \cup N) \backslash \{B_s, \ldots, B_t\},$$
           $$\text{Set } \alpha_N \leftarrow \sum_{j=s}^{t} \alpha_{B_j}, \quad \beta_N \leftarrow \sum_{j=s}^{t} \beta_{B_j}, \quad \mu_N \leftarrow \alpha_N / \beta_N, \text{ and } \theta_i \leftarrow \mu_N \text{ for all } i \in N$$
9:     if $J_{k+1} = J_k$, then terminate and return $\theta$
10: end for
```

Algorithm 19 is more sophisticated than Algorithm 18. The major difference between these two algorithms is their way of applying merge operations. In Algorithm 18, each update of the partition is merely merging two adjacent blocks, whereas in Algorithm 19 Step 5 allows independent merge operations to happen in multiple places where each may involve more than two consecutive blocks. We can prove that Algorithm 19, like the careful implementation [58] of Algorithm 18, is able to solve (IR) in $\mathcal{O}(n)$ elementary arithmetic operations. We state and prove the complexity of Algorithm 19 in Theorem 5.2.2.

Theorem 5.2.2. *If Algorithm 19 is applied to solve problem (IR), then it will yield the* *optimal solution for* (IR) *within* $\mathcal{O}(n)$ *elementary arithmetic operations.*

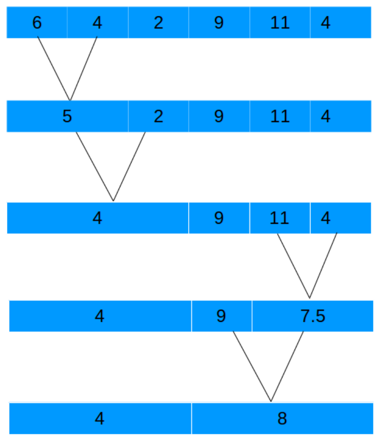
Proof. Since Algorithm 19 replicates applying the PDAS method from [69] on (BCQP), according to Theorem 5.2.1 it is guaranteed to find the optimal solution in finitely many iterations. The initialization process in Steps 1–5 requires $\mathcal{O}(n)$ elementary arithmetic operations as each step involves at most a constant number of calculations with each value from the dataset. As for the main loop involving Steps 6–9, the introduction of α and β ensures that the number of elementary arithmetic operations in merging two blocks becomes $\mathcal{O}(1)$. Thus, since the **for** loop only involves merge operations and there can be at most n merges, the desired result follows. □

5.2.3 A Comparison Between PAV and PDAS

Algorithm 19 enjoys several nice features that Algorithm 18 and other relevant algorithms do not possess. First of all, the initial partition J_0 of Algorithm 19 can be an arbitrary one. Such freedom allows Algorithm 19 to be warm-started by providing a good initial partition that is only slightly different with the optimal one. This feature is particularly appealing when a sequence of related IR problems need to be solved [109].

Another salient feature of Algorithm 19 that sets it apart from Algorithm 18 and other known active-set methods for (IR) is its potential to be parallelized. This feature is enabled because Algorithm 19 allows for multiple independent merge operations in each iterate. Specifically, this is reflected in Step 8 of Algorithm 19. As an illustrative example with $y = \{6, 4, 2, 9, 11, 4\}$ and $\omega = e$, we demonstrate in Figure 5.1 the different behavior between Algorithm 19 and Algorithm 18 when applied on this data set.

As illustrated in Figure 5.1, PAV method each time only merges two consecutive blocks whereas PDAS method allows multiple consecutive blocks to be merged and this can occur in multiple locations. Notice also that in total PDAS takes 3 division operations while PAV requires 4, despite the fact that in both methods the number of merge operations are counted as 4.



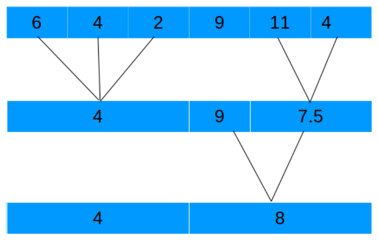


Figure 5.1: Typical merge operations in PAV (left) and PDAS (right) iteration.

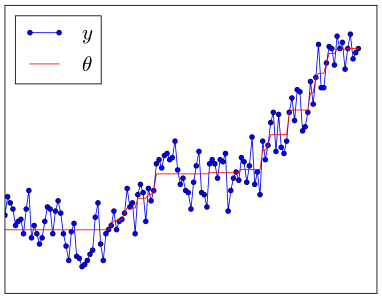
5.3 The PDAS Method for Trend Filtering

The trend filtering problem (TF) can be viewed as a generalization of problem (IR). While (IR) imposes monotonicity on the solution vector θ, variants of (TF) can impose other related properties, as illustrated in §5.3.1. Consequently, it is natural to extend PDAS for solving (TF), as we do in §5.3.2. However, since a direct application of a PDAS method may cycle when solving certain versions of (TF), we propose safeguarding strategies to ensure convergence; see §5.3.3.

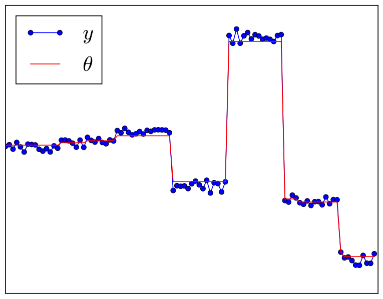
5.3.1 Regularization with Difference Operators

Common choices for the regularization function in problem (TF) are $g(\theta) = g_1(\theta) := \|D^{(d,n)}\theta\|_1$ or $g(\theta) = g_{1+}(\theta) = \|(D^{(d,n)}\theta)_+\|_1$, where $D^{(d,n)} \in \mathbb{R}^{(n-d)\times n}$ is the d-order difference matrix on $\mathbb{R}^n$. The choice of the regularization function determines the properties that one imposes on θ. We illustrate the typical behavior of θ for different choices of the regularization in Figure 5.2.

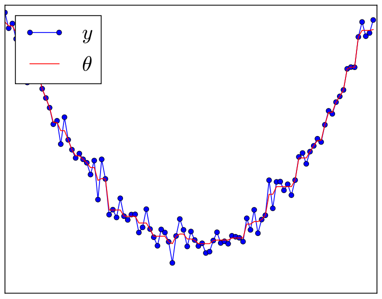
As shown in Figure 5.2, when $g = g_+$ the fitted variable θ has the property of being nearly-monotone and nearly-convex for $d = 1$ and $d = 2$, respectively. Similarly when $g = g_1$, the fitted curve would be piecewise constant and piecewise linear for $d = 1$ and $d = 2$, respectively. Higher order difference operators are applicable yet first and second order ones are more widely used in practice.



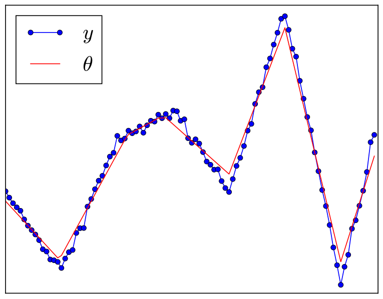
(a) $g = g_{1+}$, $d = 1$



(b) $g = g_1$, $d = 1$



(c) $g = g_{1+}$, $d = 2$



(d) $g = g_1$, $d = 2$

Figure 5.2: Trend filtering solutions for different choices of g and $D^{(d,n)}$.

5.3.2 A PDAS Framework for Trend Filtering

For brevity, we assume that D is the d-order difference matrix $D^{(d,n)}$ in this section. Denote the optimal solution of problem (TF) as (θ^*, z^*). Corresponding to this optimal solution, we may partition the indices of $D\theta^*$ as follows:

$$\mathcal{P}^* = \{j : (D\theta^*)_j > 0\}; \;\; \mathcal{N}^* = \{j : (D\theta^*)_j < 0\}; \;\; \mathcal{A}^* = \{j : (D\theta^*)_j = 0\}.$$

Active-set methods are initialized with an estimation of the optimal partition and iteratively update the estimate until the optimal one is reached. The greatest advantage of PDAS methods are their ability to apply more progressive update on a partition estimate. A typical PDAS framework consists of three generic steps: subspace minimization (SSM), termination check, and partition update. We now give the PDAS framework in Algorithm 20 and discuss the details of each of the three steps in turn.

Algorithm 20 PDAS Framework

```
1: Input an initial partition $(\mathcal{P}, \mathcal{N}, \mathcal{A})$
2: loop
3:     Compute the subspace minimizer $(\theta, z)$ corresponding to $(\mathcal{P}, \mathcal{N}, \mathcal{A})$
4:     if $(\theta, z)$ is optimal, then terminate and return $(\theta, z)$
5:     Compute a new partition $(\mathcal{P}, \mathcal{N}, \mathcal{A})$
6: end loop
```

Subspace Minimization

A subspace minimizer needs to be computed in each iteration, as in Step 3 of Algorithm 20. Given a partition $(\mathcal{P}, \mathcal{N}, \mathcal{A})$, the associated subspace minimizer is a primal-dual pair (θ, z) that can be viewed as an estimate of (θ^*, z^*). The following schematics show the processes for computing (θ, z) for problem (TF). For the convenience of expression, we denote $\mathcal{I}$ as the union $\mathcal{P} \cup \mathcal{N}$.

SSM for $g(\theta) = \|(D\theta)_+\|_1$

Set $z_j \leftarrow 0$ for $j \in \mathcal{N}$ and $z_j \leftarrow 1$ for $j \in \mathcal{P}$.

Solve for $(\theta, z_{\mathcal{A}})$:

$$\begin{bmatrix} I & \lambda D_{\mathcal{A}}^T \\ \lambda D_{\mathcal{A}} & 0 \end{bmatrix} \begin{bmatrix} \theta \\ z_{\mathcal{A}} \end{bmatrix} = \begin{bmatrix} y - \lambda D_{\mathcal{I}}^T z_{\mathcal{I}} \\ 0 \end{bmatrix}. \quad (5.1)$$

Set

$$\mathcal{V}_P \leftarrow \{j \in \mathcal{P} : (D\theta)_j < 0\};$$
$$\mathcal{V}_N \leftarrow \{j \in \mathcal{N} : (D\theta)_j > 0\};$$
$$\mathcal{V}_{AP} \leftarrow \{j \in \mathcal{A} : z_j > 1\};$$
$$\mathcal{V}_{AN} \leftarrow \{j \in \mathcal{A} : z_j < 0\}.$$

SSM for $g(\theta) = \|D\theta\|_1$

Set $z_j \leftarrow -1$ for $j \in \mathcal{N}$ and $z_j \leftarrow 1$ for $j \in \mathcal{P}$.

Solve for $(\theta, z_{\mathcal{A}})$:

$$\begin{bmatrix} I & \lambda D_{\mathcal{A}}^T \\ \lambda D_{\mathcal{A}} & 0 \end{bmatrix} \begin{bmatrix} \theta \\ z_{\mathcal{A}} \end{bmatrix} = \begin{bmatrix} y - \lambda D_{\mathcal{I}}^T z_{\mathcal{I}} \\ 0 \end{bmatrix}. \quad (5.2)$$

Set

$$\mathcal{V}_P \leftarrow \{j \in \mathcal{P} : (D\theta)_j < 0\};$$
$$\mathcal{V}_N \leftarrow \{j \in \mathcal{N} : (D\theta)_j > 0\};$$
$$\mathcal{V}_{AP} \leftarrow \{j \in \mathcal{A} : z_j > 1\};$$
$$\mathcal{V}_{AN} \leftarrow \{j \in \mathcal{A} : z_j < -1\}.$$

Observing that SSM requires solving a sparse KKT linear system (5.1) or (5.2), we discuss how it could be efficiently carried out. Notice that the linear system could be expressed as

$$\begin{bmatrix} \theta \\ z_{\mathcal{A}} \end{bmatrix} = \begin{bmatrix} I & \lambda D_{\mathcal{A}}^T \\ \lambda D_{\mathcal{A}} & 0 \end{bmatrix}^{-1} \begin{bmatrix} y - \lambda D_{\mathcal{I}}^T z_{\mathcal{I}} \\ 0 \end{bmatrix} = \begin{bmatrix} I - D_{\mathcal{A}}^T (D_{\mathcal{A}} D_{\mathcal{A}}^T)^{-1} D_{\mathcal{A}} \\ (D_{\mathcal{A}} D_{\mathcal{A}}^T)^{-1} D_{\mathcal{A}} / \lambda \end{bmatrix} \begin{bmatrix} y - \lambda D_{\mathcal{I}}^T z_{\mathcal{I}} \end{bmatrix}.$$

It turns out that the solution $(\theta, z_{\mathcal{A}})$ of system (5.1) or (5.2) could be efficiently obtained by

$$\text{solving for } z_{\mathcal{A}} \text{ from the system} \qquad D_{\mathcal{A}} D_{\mathcal{A}}^T z_{\mathcal{A}} = D_{\mathcal{A}}(y - \lambda D_{\mathcal{I}}^T z_{\mathcal{I}})/\lambda, \tag{5.3a}$$

$$\text{then setting} \qquad \theta \leftarrow y - \lambda D^T z. \tag{5.3b}$$

Since $D_{\mathcal{A}}$ is usually a first (second) order difference matrix, consequently $D_{\mathcal{A}} D_{\mathcal{A}}^T$ is a tridiagonal (quindiagonal) matrix, meaning that the coefficient matrix of the linear system (5.3a) is banded and $z_{\mathcal{A}}$ can be solved cheaply.

Termination Check

One can easily see that a subspace minimizer (θ, z) is optimal if the set $\mathcal{V} = \mathcal{V}_P \cup \mathcal{V}_N \cup \mathcal{V}_{AP} \cup \mathcal{V}_{AN}$, consisting of indices of $D\theta$ and z corresponding to violated bounds, is empty [26]. Thus, when $\mathcal{V}$ is empty, optimality has been reached and the algorithm terminates. Otherwise, these sets indicate a manner in which the partition could be updated.

Partition Update

In PDAS methods, the indices of $D\theta$ and z violating their bounds are the candidates whose membership need to be changed in a partition update. Specifically, a standard update in the PDAS method from [69] involves the following steps:

$$\begin{aligned} &\mathcal{P} \leftarrow (\mathcal{P} \backslash \mathcal{V}_P) \cup \mathcal{V}_{AP}; \\ &\mathcal{N} \leftarrow (\mathcal{P} \backslash \mathcal{V}_N) \cup \mathcal{V}_{AN}; \\ &\mathcal{A} \leftarrow \mathcal{A} \backslash (\mathcal{V}_{AP} \cup \mathcal{V}_{AN}) \cup (\mathcal{V}_P \cup \mathcal{V}_N). \end{aligned} \tag{5.4}$$

5.3.3 Safeguard

There is no convergence guarantee for Algorithm 20 for an arbitrary instance of (TF); indeed, an example illustrating that the method can cycle is given in [27] for BQPs and presented below is an example that Algorithm 20 cycles in solving (TF).

Example 5.3.1. *$y = (603, 996, 502, 19, 56, 139)^T$, $\lambda = 100$, and $g(\theta) = \|D^{(2,6)}\theta\|_1$.*

Iter	$\mathcal{P}$	$\mathcal{N}$	$\mathcal{A}$	$D\theta$	z
0	$\{2,3,4\}$	$\{1\}$	$\emptyset$	$(13,-689,820,-254)^T$	$(-1,1,1,1)^T$
1	$\{3\}$	$\emptyset$	$\{1,2,4\}$	$(0,0,4227/38,0)^T$	$(-5293/2280,-482/475,1,5201/5700)^T$
2	$\{3\}$	$\{1,2\}$	$\{4\}$	$(-787,520,-16,0)^T$	$(-1,-1,1,91/100)^T$
3	$\emptyset$	$\{1\}$	$\{2,3,4\}$	$(-887/5,0,0,0)^T$	$(-1,127/125,371/125,943/500)^T$
4	$\{2,3,4\}$	$\{1\}$	$\emptyset$	$(13,-689,820,-254)^T$	$(-1,1,1,1)^T$
$\vdots$					

Table 5.1: An illustration of Algorithm 3 cycling

Since the algorithm returns to a previously explored partition without computing an optimal solution, the algorithm cycles, i.e., it is not convergent for this problem instance from the given starting point. We have confirmed that the cycle is not caused by numerical issues since the same updates would occur with exact computation.

A simple safeguarding strategy to overcome this issue and ensure convergence is proposed in [78] and subsequently embedded in the work of [81, 21]. In particular, when $|\mathcal{V}|$ fails to decrease for several consecutive iterations, a backup procedure is invoked in which (5.4) is modified to only change partition membership of one index of $\mathcal{V}$. We rephrase this approach as a PDAS method for (TF) in Algorithm 21.

Algorithm 21 A PDAS Method with Safeguarding of [78]

```
1: Input a partition $(\mathcal{P},\mathcal{N},\mathcal{A})$, an integer $\mathtt{t_{max}}$, initialize parameter $\mathtt{V_{best}}$ as $\infty$, $t$ as 0
2: loop
3:     Compute the subspace minimizer $(\theta, z)$ corresponding to $(\mathcal{P},\mathcal{N},\mathcal{A})$
4:     if $|\mathcal{V}| = 0$, then terminate and return $(\theta, z)$
5:     if $|\mathcal{V}| < \mathtt{V_{best}}$ then
6:         Set $t \leftarrow 0$, and $\mathtt{V_{best}} \leftarrow |\mathcal{V}|$
7:     else if $|\mathcal{V}| \geq \mathtt{V_{best}}$ then
8:         Set $t \leftarrow t+1$
9:         if $t \leq \mathtt{t_{max}}$ then
10:            Apply partition update by (5.4)
11:        else if $t > \mathtt{t_{max}}$ then
12:            Set $j \leftarrow \min\{i : i \in \mathcal{V}\}$ and apply partition update by
               $$\begin{aligned} &\text{moving } j \text{ from } \mathcal{P} \text{ to } \mathcal{A}, \text{ if } j \in \mathcal{V}_P \\ &\text{moving } j \text{ from } \mathcal{N} \text{ to } \mathcal{A}, \text{ if } j \in \mathcal{V}_N \\ &\text{moving } j \text{ from } \mathcal{A} \text{ to } \mathcal{P}, \text{ if } j \in \mathcal{V}_{AP} \\ &\text{moving } j \text{ from } \mathcal{A} \text{ to } \mathcal{N}, \text{ if } j \in \mathcal{V}_{AN} \end{aligned} \tag{5.5}$$
13:        end if
14:    end if
15: end loop
```

The safeguard of Algorithm 21 essentially employs a heuristic to decide whether the partition update applies (5.4) or that of (5.5). We describe in this section an alternative strategy that we have found to perform better in our experiments. First, unlike that of [78, 81, 21], our safeguard changes the memberships of a portion of $\mathcal{V}$, where the portion size is dynamically updated. Another difference in the safeguard design is that we employ a finite queue (first-in-first-out) to store recent values of $|\mathcal{V}|$. When an element is pushed into the queue that is already full, the earliest element is removed. We use the maximum value of the elements in the queue as the reference measure. By enforcing strict decrease on the reference measure for each iteration, our proposed algorithm framework is guaranteed to converge to the optimal solution.

Algorithm 22 PDAS Framework with Safeguarding

```
1: Input $(\mathcal{P},\mathcal{N},\mathcal{A})$, queue $Q_m$ with size $m$, proportion $p \in (0,1]$, parameter $\delta_s \in (0,1)$ and
   $\delta_e \in (1,\infty)$
2: loop
3:   Compute the subspace minimizer $(\theta, z)$ corresponding to $(\mathcal{P},\mathcal{N},\mathcal{A})$
4:   if $|\mathcal{V}| = 0$, then terminate and return $(\theta, z)$
5:   Set max/min $\leftarrow$ maximum/minimum of $Q_m$
6:   if $|\mathcal{V}| >$ max then
7:     set $p \leftarrow \max(\delta_s p, \frac{1}{|\mathcal{V}|})$
8:   else if $|\mathcal{V}| <$ min then
9:     push $|\mathcal{V}|$ into $Q_m$ and set $p \leftarrow \max(\delta_e p, 1)$
10:  else
11:    push $|\mathcal{V}|$ into $Q_m$
12:  end if
13:  Sort $\mathcal{V}$ by $\max(\lambda|D\theta|, |z|)$ and apply (5.4), only changing the top $p|\mathcal{V}|$ indices
14: end loop
```

Algorithm 22 can be understood as follows. If $|\mathcal{V}| = 0$, then (θ, z) is optimal and the algorithm terminates. Otherwise, the update (5.4) is to be applied using only the $\lfloor p|\mathcal{V}|\rfloor$ indices from $\mathcal{V}$ corresponding to the largest violations. If $p = 1/|\mathcal{V}|$, then this corresponds to moving only one index as in [81], but if $p \in (1/|\mathcal{V}|, 1]$, then a higher portion of violated indices may be moved. As long as the reference value—i.e., the maximum of the values in the queue—decreases, the value for p is maintained or is increased. However, if the reference value fails to decrease, then p is decreased. Overall, since the procedure guarantees that the reference value is monotonically decreasing and that p is sufficiently reduced whenever a new value for $|\mathcal{V}|$ is not below the reference value, our strategy preserves the convergence guarantees established in [81] while applying more aggressive update on the partition estimate.

5.4 Experiments

We implemented Algorithms 19, 20, and 22 in Python 2.7, using the Numpy (version 1.8.2) and Scipy (version 0.14.0) packages for matrix operations. In the following subsections, we discuss the results of numerical experiments for solving randomly generated instances of problems (IR) and (TF). For problem (IR), merge operations for Algorithm 19 were implemented sequentially, though in Figure 5.1 we also illustrate how they can potentially be implemented in parallel. Throughout our experiments, we set ω as an all-one vector.

5.4.1 Test on Isotonic Regression

We compare the numerical performance of Algorithm 19 (`PDAS`) with the Python implementation of Algorithm 18 (`PAV`) that is integrated into scikit-learn (version 0.13.1, all later versions are re-implemented in C) [99]. The data y_i are generated by $y_i \leftarrow i + \varepsilon_i$ where $\varepsilon_i \sim \mathcal{N}(0,4)$. By default the initial partition is set as $J_0 = \{\{1\},\{2\},\ldots,\{n\}\}$ for both algorithms. We generated 10 random instances each for $n \in \{1\times10^4, 5\times10^4, \ldots, 33\times10^4\}$. A boxplot for running time in seconds (`Time (s)`) and number of merge operations (# `Merge`) are reported in Figure 5.3.

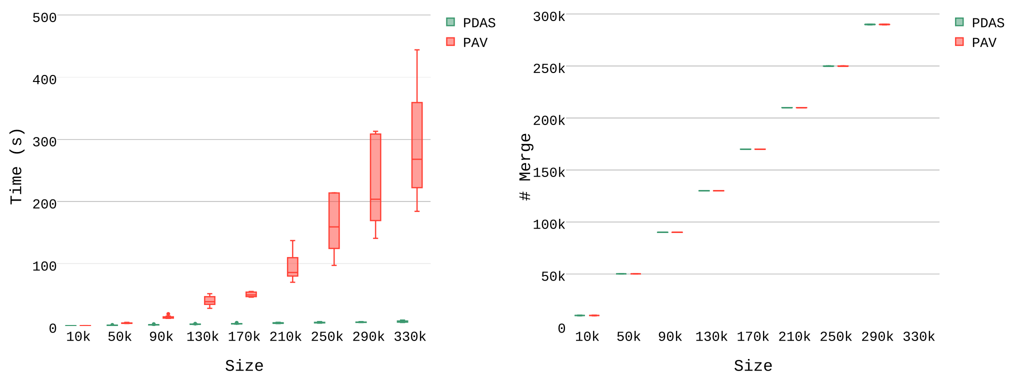


Figure 5.3: Comparison of PDAS and PAV in running time (left) and # of merges (right).

Figure 5.3 demonstrates that the implementation of `PDAS` outperforms `PAV` in terms of running time. That being said, when both use the set of singletons as the starting point, the numbers of merge operations performed by the two algorithms are nearly identical.

Warm-starting

Figure 5.3 does not show an obvious advantage of PDAS over PAV when it comes to the total number of merge operations. However, as claimed, we now show an advantage of PDAS in terms of its ability to exploit a good initial partition. We simulate warm-starting PDAS by generating an instance of (IR) as in our previous experiment, solving it with PDAS, and using the solution as the starting point for solving related instances for which the data vector y has been perturbed. In particular, for each problem size n, we generated 10 perturbed instances by adding a random variable $\epsilon_i \sim \mathcal{N}(0, 10^{-2})$ to each y_i. The results of solving the perturbed instances are reported in Figure 5.4.

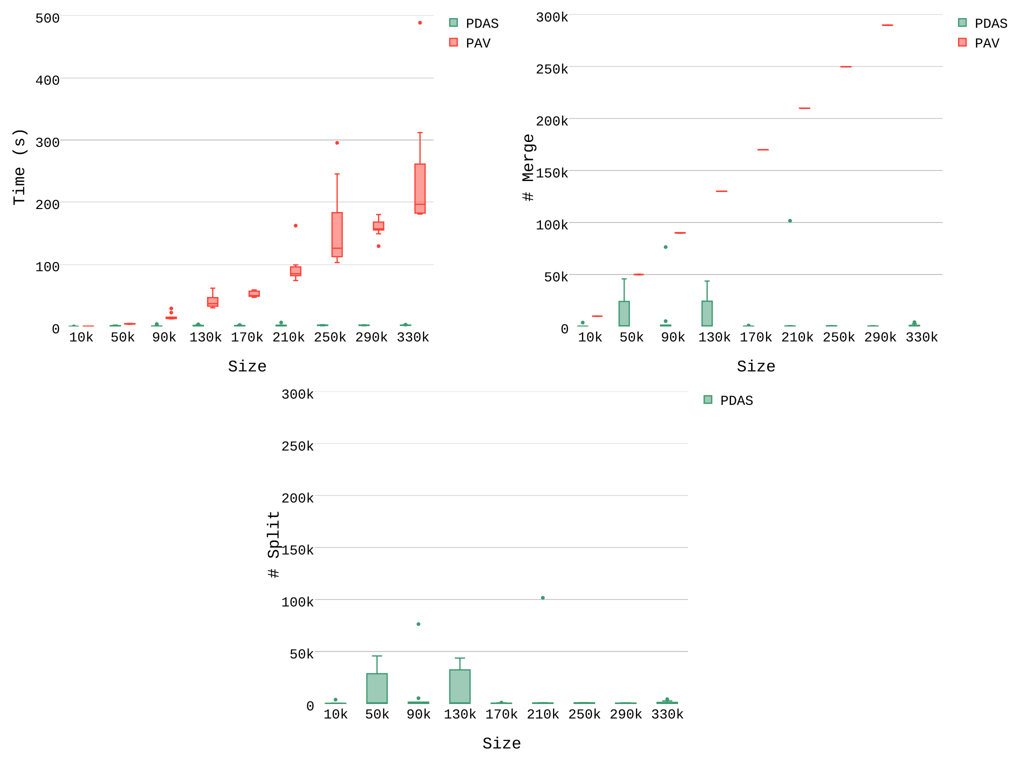


Figure 5.4: Comparison of warm-started PDAS and PAV.

Since PAV is not able to utilize a good initial partition, the required work (i.e., number of merge operations) for solving the perturbed problems is not cheaper than for the base instance. In contrast, warm-starting the PDAS algorithm can greatly reduce the computational cost as observed in the much-reduced number of merge operations (even after accounting for the added split operations).

5.4.2 Test on Trend Filtering

We now compare the performance of several PDAS variants for trend filtering. In particular, we compare the straightforward PDAS method of Algorithm 20 (`PDAS`), Algorithm 21 (`SF1`), and the PDAS method with our proposed safeguard strategy in Algorithm 22 (`SF2`). The safeguard parameter $\mathtt{t_{max}} = 5$ was chosen for `SF1` and we similarly set $m = 5$, $\delta_s = 0.9$, and $\delta_e = 1.1$ for `SF2`. We generated 10 random problem instances each for $n \in \{10^4, 1.7 \times 10^5, 3.3 \times 10^5\}$ where, for each instance, the data vector had y_i uniformly distributed in $[0, 10]$. Such datasets had minimum pattern and thus made each problem relatively difficult to solve. We considered both regularization functions g_1 and g_{1+} defined in §5.3.1 with difference matrices $D^{(1,n)}$ and $D^{(2,n)}$, setting $\lambda = 10$ in all cases. For all runs, we set an iteration limit of 800; if an algorithm failed to produce the optimal solution within this limit, then the run was considered a failure. The percentages of successful runs for each algorithm is reported in Table 5.2.

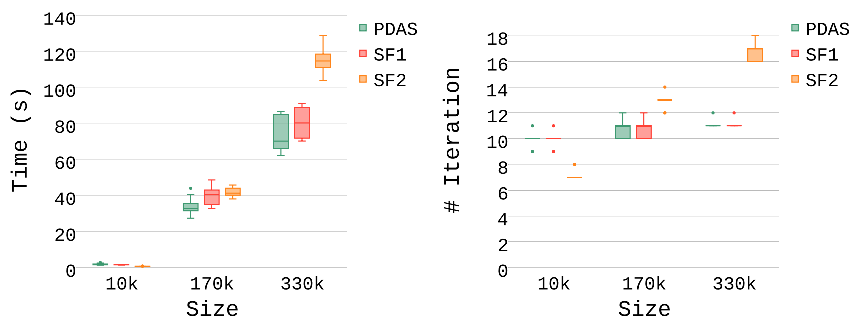
Table 5.2: Percentages of successful runs for each algorithm and problem type

n (size)	% of success											
	$g(\theta) = \|(D^{(1,n)}\theta)_+\|_1$			$g(\theta) = \|D^{(1,n)}\theta\|_1$			$g(\theta) = \|(D^{(2,n)}\theta)_+\|_1$			$g(\theta) = \|D^{(2,n)}\theta\|_1$		
	PDAS	SF1	SF2	PDAS	SF1	SF2	PDAS	SF1	SF2	PDAS	SF1	SF2
1.0e+4	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0
1.7e+5	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
3.3e+5	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0

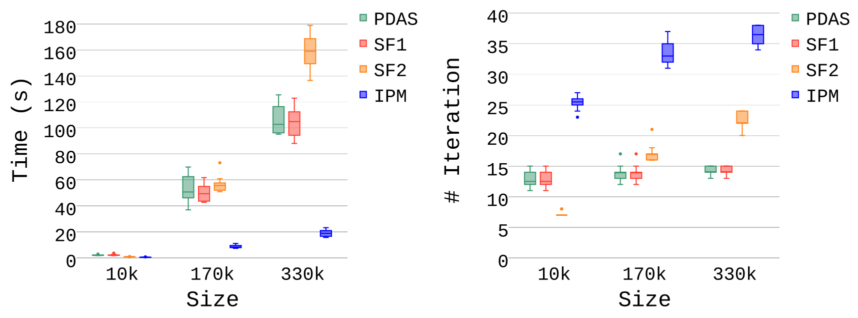
We observe from Table 5.2 that all algorithms solved all instances when the regularization function involved a first-order difference matrix, but that `PDAS` and `SF1` both had failures when a second-order difference matrix is used. By contrast, our proposed safeguard in `SF2` results in a method that is able to solve all instances within the iteration limit. This shows that our proposed safeguard, which allows more aggressive updates, can be more effective than a conservative safeguard.

To compare further the performance of the algorithms, we collected the running time and iteration number for all successfully solved instances. Figure 5.5 demonstrates that when $D = D^{(1,n)}$, all algorithms show very similar performance. However, when

$D = D^{(2,n)}$ as in Figure 5.6, the results show that `SF2` is not only more reliable than `PDAS` and `SF1`; it is also more efficient even when `SF1` is successful. We also include the results for the interior-point method (`IPM`) proposed in [82], but emphasize that this algorithm is implemented in Matlab (as opposed to Python) and is only set up to solve the instances when an ℓ_1-regularization function is used. Although the imterior point method implementation used herein has very impressive practical performance, we remark that in general it is difficult to warm-starting interior point methods [77], despite recent efforts toward this direction [120, 49, 48] .



(a) $D = D^{(1,n)}$, $g = g_{1+}$

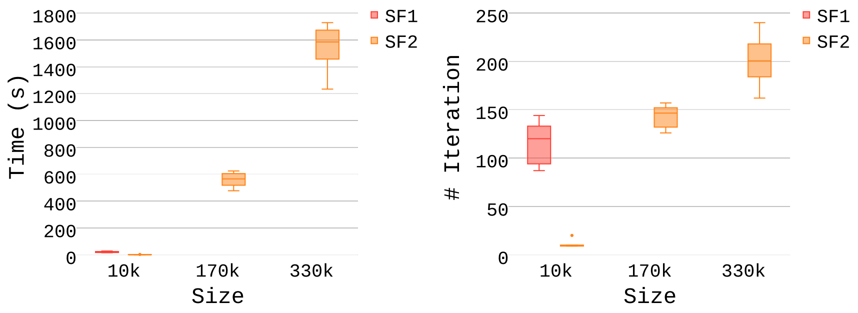


(b) $D = D^{(1,n)}$, $g = g_1$

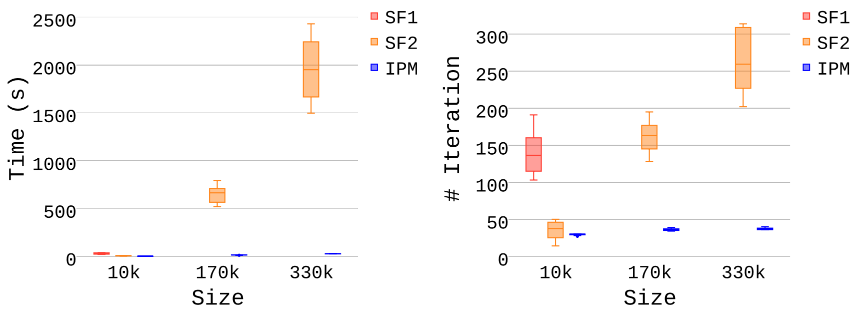
Figure 5.5: PDAS vs IPM for $D = D^{1,n}$ and different choices of g.

Warm-starting

We conclude our experiments by comparing the performance of `IPM`—which is not set up for warm-starting—and warm-started `SF2`. As in §5.4.1, we generated 10 perturbed instances for a given dataset by adding a random variable $\epsilon_i \sim \mathcal{N}(0, 10^{-2})$ to y_i. The running times and numbers of iterations for solving the perturbed problems are reported



(a) $D = D^{(2,n)}$, $g = g_{1+}$



(b) $D = D^{(2,n)}$, $g = g_1$

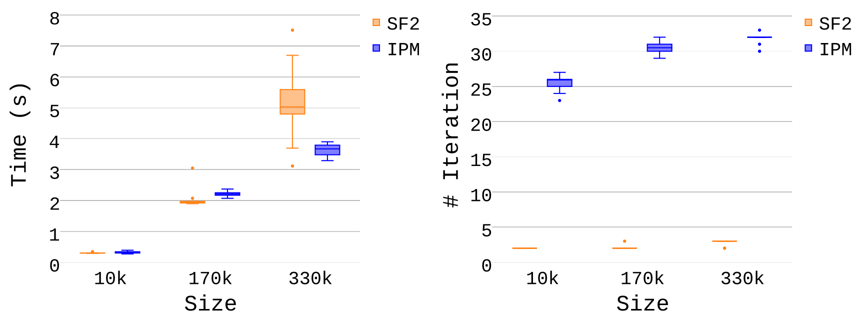
Figure 5.6: PDAS vs IPM for $D = D^{2,n}$ and different choices of g.

via boxplots in Figure 5.7.

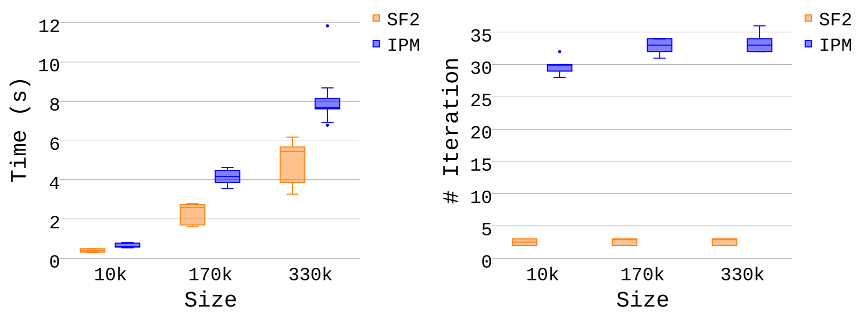
Comparing the performance of SF2 between Figures 5.5, Figures 5.6 and 5.7 shows that warm-starting SF2 can dramatically reduce the cost of solving an instance of (TF). With a cold-start, SF2 may require hundreds of iterations, while with warm-starting it requires dramatically fewer iterations. In contrast, IPM does not benefit much from a good starting point.

5.5 Concluding Remarks

We propose innovative PDAS algorithms for Isotonic Regression (IR) and Trend Filtering (TF). For IR, our PDAS method enjoys the same theoretical properties as the well-known PAV method, but also has the ability to be warm-started, can exploit parallelism, and outperforms PAV in our experiments. Our proposed safeguarding strategy for a PDAS method for TF also exhibits reliable and efficient behavior. Overall, our proposed methods show that PDAS frameworks are powerful when solving a broad class of regularization



(a) $D = D^{(1,n)}$, $g = g_1$



(b) $D = D^{(2,n)}$, $g = g_1$

Figure 5.7: PDAS (with warm-start) vs IPM.

problems. Our major discovery is that customizing PDAS method to certain machine learning problems usually results to novel methods that outperform state-of-the-art algorithms. It therefore deserves further exploration of problems to which PDAS methods are applicable.

Chapter 6

Conclusion

Primal-dual active-set (PDAS) methods show great potential in solving certain large-scale convex QPs arising in optimal control and statistics. As other active-set methods, PDAS methods possess many favorable properties such as the ability to be warm-started and to obtain highly accurate solutions. These features make PDAS methods ideal candidates when solving a nonlinear optimization problem via a series of quadratic optimization subproblems. Moreover, PDAS methods can adapt the active set estimate in a much more aggressive manner than traditional active-set methods, making them well-suited for solving large-scale problems.

In this dissertation, we have proposed various methods for improving the practical performance of PDAS methods and for enhancing their capabilities to solve larger-classes of problems. First, for solving general convex QPs, a primary challenge is that the algorithm may encounter active set estimates at which a subspace minimizer is not well-defined. In addition, even if the subspace minimizers are well-defined throughout the iterative process, a plain PDAS iteration might cycle and, thus, never converge. In §3, we have addressed these issues by proposing enhancements to arrive at a globally convergent PDAS framework for general convex QPs. In particular, the former issue is addressed by a process of solving linear optimization problems while adaptively expanding the subspace until a feasible partition is reached. To address the latter issue, we introduced into the algorithm an uncertainty set to house indices that are suspected of leading to cycling. Global convergence of the algorithm is then achieved by dynamically adapting the uncertainty set until no cycle persists. Through numerical experiments, we illustrated that our enhancements are effective and the algorithmic framework is very efficient when solving large-scale convex QPs.

We continued our proposed enhancements to PDAS methods in §4, in which we presented ideas for lowering the chief computational cost in any PDAS method, namely, the subspace minimization procedure. We proved that inexact subspace minimization can be exploited to yield productive steps under reduced computational costs. To ensure global convergence of the method, we proposed precise conditions under which a given inexact step is acceptable. These conditions involve inferring an upper bound on the norm of a matrix inverse, which we show can be done explicitly or eventually through updates for a dynamic parameter. We show through examples of discretized optimal control problems that allowing inexactness can be appealing in that iterative linear system solution methods can be terminated early with gains in overall computational cost for the algorithm. This provides evidence that PDAS methods with inexact, iterative subspace minimization steps can perform well in comparison to methods that employ direct factorization techniques.

As another area in which PDAS methods can be effective, we showed in §5 that PDAS methods are powerful in solving certain large-scale optimization problems arising in statistical learning. In particular, we showed that the isotonic regression (IR) problem over n data points is solvable by a PDAS method in O(n) elementary arithmetic operations. In addition, we showed that a sophisticated implementation of this method outperforms the state-of-the-art method for solving such problems. We also showed that safeguarded variants of PDAS methods show competitive performance in solving a broad class of related trend filtering (TF) problems.

We conclude by pointing out a few research topics that one may consider for extending the work in this dissertation. First, our convergence guarantees for PDAS methods with inexact subproblem solves in §4 were limited to cases of solving a restricted class of convex QPs. One topic of interest would be extending these techniques for solving general convex QPs, such as by merging them with the globalization strategies proposed in

§3. Another interesting research direction would be to explore parallelization techniques in implementations of PDAS methods. We have discussed how one could parallelize a PDAS method for solving IR problems in §5, but completing such an implementation and comparing it to a sequential solver remains as future work. Finally, the incorporation of a PDAS method for solving QPs into an outer algorithm for solving nonlinear optimization problems (NLPs) could lead to interesting new methods. The practical performance of such an approach as compared to one in which a traditional QP solver is employed could be groundbreaking.

Bibliography

[1] M. Aganagić. Newton's method for linear complementarity problems. *Mathematical Programming*, 28(3):349–362, 1984.

[2] J. H. Ahlberg and E. N. Nilson. Convergence properties of the spline fit. *Journal of the Society for Industrial and Applied Mathematics*, 11(1):95–104, 1963.

[3] R. E. Barlow and H. D. Brunk. The isotonic regression problem and its dual. *Journal of the American Statistical Association*, 67(337):140–147, 1972.

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

[5] J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

[6] R. Battiti and M. Brunato. *The Lion Way: Learning Plus Intelligent Optimization.* Createspace Independent Pub, 2014.

[7] A. Bemporad. Model predictive control: Basic concepts, 2015.

[8] 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.

[9] M. Bergounioux and K. Kunisch. Primal-dual strategy for state-constrained optimal control problems. *Computational Optimization and Applications*, 22(2):193–224, 2002.

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

[11] M. J. Best. An algorithm for the solution of the parametric quadratic programming problem. In *Applied Mathematics and Parallel Computing*, pages 57–76. Physica-Verlag HD, Heidelberg, Germany, 1996.

[12] M. J. Best and N. Chakravarti. Active set algorithms for isotonic regression; a unifying framework. *Mathematical Programming*, 47(1-3):425–439, 1990.

[13] E. G. Birgin and J. M. Martínez. Large-scale active-set box-constrained optimization method with spectral projected gradients. *Computational Optimization and Applications*, 23(1):101–125, 2002.

[14] D. Boley. Local linear convergence of the alternating direction method of multipliers on quadratic or linear programs. *SIAM Journal on Optimization*, 23(4):2183–2207, 2013.

[15] I. M. Bomze. On standard quadratic optimization problems. *Journal of Global Optimization*, 13(4):369–387, 1998.

[16] B .E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92*, pages 144–152, New York, NY, USA, 1992. ACM Press.

[17] S. P. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[18] S. P. Boyd and L. Vandenberghe. *Convex Optimization.* Berichte über verteilte messysteme. Cambridge University Press, 2004.

[19] P. N. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM Journal on Scientific and Statistical Computing*, 11(3):450–481, 1990.

[20] H. D. Brunk, R. E. Barlow, D. J. Bartholomew, and J. M. Bremner. Statistical inference under order restrictions.(the theory and application of isotonic regression). Technical report, DTIC Document, 1972.

[21] 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.

[22] S. Cafieri, M. D'Apuzzo, M. Marino, A. Mucherino, and G. Toraldo. Interior-point solver for large-scale quadratic programming problems with bound constraints. *Journal of Optimization Theory and Applications*, 129(1):55–75, 2006.

[23] S. T. Choi, C. C. Paige, and M. A Saunders. Minres-qlp: A krylov subspace method for indefinite or singular symmetric systems. *SIAM Journal on Scientific Computing*, 33(4):1810–1836, 2011.

[24] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-Region Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[25] R. W. Cottle, J. S. Pang, and R. E. Stone. *The Linear Complementarity Problem*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1992.

[26] F. E. Curtis and Z. Han. Globally Convergent Primal-Dual Active-Set Methods with Inexact Subproblem Solves. Technical Report 14T-010, COR@L Laboratory, Department of ISE, Lehigh University, 2014. In first review for SIAM Journal on Optimization.

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

[28] F. E. Curtis, H. Jiang, and D. P. Robinson. An adaptive augmented lagrangian method for large-scale constrained optimization. *Mathematical Programming*, pages 1–45, 2014.

[29] Y. Dai and R. Fletcher. Projected Barzilai-Borwein methods for large-scale box-constrained quadratic programming. *Numerische Mathematik*, 100(1):21–47, 2005.

[30] Y. Dai and R. Fletcher. New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds. *Math. Program.*, 106(3):403–421, May 2006.

[31] J. C. De los Reyes. A primal-dual active set method for bilaterally control constrained optimal control of the Navier-Stokes equations. *Numerical Functional Analysis and Optimization*, 25(7–8):657–683, 2005.

[32] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton Methods. *SIAM Journal on Numerical Analysis*, 19(2):400–408, 1982.

[33] A. Deza, E. Nematollahi, and T. Terlaky. How good are interior point methods. *Klee-Minty cubes tighten iteration-complexity bounds. AdvOL-Report*, 20, 2004.

[34] J. Eckstein and P. J. S. Silva. A practical relative error criterion for augmented lagrangians. *Mathematical Programming*, 141(1-2):319–348, 2013.

[35] H. J. Ferreau. An online active set strategy for fast solution of parametric quadratic programs with applications to predictive engine control. *University of Heidelberg*, 2006.

[36] H. J. Ferreau, H. G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, 18(8):816–830, 2008.

[37] 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.

[38] H. J. Ferreau, A. Kozma, and M. Diehl. A parallel active-set strategy to solve sparse parametric quadratic programs arising in mpc. In *Nonlinear Model Predictive Control*, volume 4, pages 74–79, 2012.

[39] R. Fletcher. *Practical Methods of Optimization.* Wiley-Interscience, New York, NY, USA, Second edition, 2000.

[40] A. Friedlander and J. M. Martínez. On the maximization of a concave quadratic function with box constraints. *SIAM Journal on Optimization*, 4(1):177–192, 1994.

[41] M. Fukushima and P. Tseng. An implementable active-set algorithm for computing a B-stationary point of a mathematical program with linear complementarity constraints. *SIAM Journal on Optimization*, 12(3):724–739, 2002.

[42] 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.

[43] 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.

[44] 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.

[45] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Emerald Group Publishing Limited, Bingley, UK, 1982.

[46] P. E. Gill and D. P. Robinson. A primal-dual augmented lagrangian. *Computational Optimization and Applications*, 51(1):1–25, 2012.

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

[48] J. Gondzio and A. Grothey. Reoptimization with the primal-dual interior point method. *SIAM Journal on Optimization*, 13(3):842–864, 2002.

[49] J. Gondzio and T. Terlaky. A computational view of interior point methods. *Advances in Linear and Integer Programming, Oxford Lecture Series*, (4):103–144, 1996.

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

[51] 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.

[52] 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.

[53] N. I. M. Gould and P. L. Toint. A quadratic programming bibliography. Technical Report RAL Numerical Analysis Group Internal Report 2000-1, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2000.

[54] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsofts bing search engine. In *Proceedings of the 27th International Conference on Machine Learning ICML 2010*, June 2010.

[55] R. Griesse and S. Volkwein. A primal-dual active set strategy for optimal boundary control of a nonlinear reaction-diffusion system. *SIAM Journal on Control and Optimization*, 44(2):467–494, 2005.

[56] 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.

[57] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for newton's method. *SIAM Journal on Numerical Analysis*, pages 707–716, 1986.

[58] S. J. Grotzinger and C. Witzgall. Projections onto order simplexes. *Applied mathematics and optimization*, 12(1):247–270, 1984.

[59] A. G. Hadigheh, K. Mirnia, and T. Terlaky. Active constraint set invariancy sensitivity analysis in linear optimization. *Journal of Optimization Theory and Applications*, 133(3):303–315, 2007.

[60] A. G. Hadigheh, O. Romanko, and T. Terlaky. Bi-parametric convex quadratic optimization. *Optimization Methods & Software*, 25(2):229–245, 2010.

[61] 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.

[62] W. W. Hager. The dual active set algorithm and its application to linear programming. *Comput. Optim. Appl.*, 21(3):263–275, March 2002.

[63] 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.

[64] W. W. Hager and H. Zhang. A new active set algorithm for box constrained optimization. *SIAM Journal on Optimization*, 17(2):526–557, 2006.

[65] W. W. Hager and H. Zhang. A new active set algorithm for box constrained optimization. *SIAM Journal on Optimization*, 17(2):526 – 557, 2006.

[66] D. Han and X. Yuan. Local linear convergence of the alternating direction method of multipliers for quadratic programs. *SIAM Journal on numerical analysis*, 51(6):3446–3457, 2013.

[67] M. Heinkenschloss, M. Ulbrich, and S. Ulbrich. Superlinear and quadratic convergence of affine-scaling interior-point Newton methods for problems with simple bounds without strict complementarity assumption. *Mathematical Programming*, 86(3):615–635, 1999.

[68] M. R. Hestenes. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320, 1969.

[69] 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.

[70] M. Hintermüller, V. A. Kovtunenko, and K. Kunisch. Obstacle problems with cohesion: A hemivariational inequality approach and its efficient numerical solution. *SIAM Journal on Optimization*, 21(2):491–516, 2011.

[71] S. Hüeber, A. Matei, and B. I. Wohlmuth. Efficient algorithms for problems with friction. *SIAM Journal on Scientific Computing*, 29(1):70–92, 2007.

[72] S. Hüeber, G. Stadler, and B. I. Wohlmuth. A primal-dual active set algorithm for three-dimensional contact problems with Coulomb friction. *SIAM Journal on Scientific Computing*, 30(2):572–596, 2008.

[73] P. Hungerländer and F. Rendl. A feasible active set method for strictly convex quadratic problems with simple bounds. 2015.

[74] K. Ito and K. Kunisch. Optimal control of elliptic variational inequalities. *Applied Mathematics and Optimization*, 41(3):343–364, 2000.

[75] K. Ito and K. Kunisch. The primal-dual active set method for nonlinear optimal control problems with bilateral constraints. *SIAM Journal on Control and Optimization*, 43(1):357–376, 2004.

[76] K. Ito and K. Kunisch. Convergence of the primal-dual active set strategy for diagonally dominant systems. *SIAM Journal on Control and Optimization*, 46(1):14–34, 2007.

[77] E. John and E. A. Yıldırım. Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension. *Computational Optimization and Applications*, 41(2):151–183, 2008.

[78] J. J. Júdice and F. M. Pires. A block principal pivoting algorithm for large-scale strictly monotone linear complementarity problems. *Computers & operations research*, 21(5):587–596, 1994.

[79] C. Kanzow. An active set-type newton method for constrained nonlinear systems. In *In Complementarity: Applications, Algorithms and Extensions*, pages 179–200. Kluwer Academic Publishers, 1999.

[80] A. J. Kearsley, R. A. Tapia, and M. W. Trosset. An approach to parallelizing isotonic regression. In *Applied Mathematics and Parallel Computing*, pages 141–147. Springer, 1996.

[81] J. Kim and H. Park. Fast active-set-type algorithms for l1-regularized linear regression. In *International Conference on Artificial Intelligence and Statistics*, 2010.

[82] S. Kim, K. Koh, S. Boyd, and D. Gorinevsky. l1 trend filtering. *SIAM Review*, 2009.

[83] S. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):606–617, 2007.

[84] H. Konno and H. Yamazaki. Mean-absolute deviation portfolio optimization model and its applications to tokyo stock market. *Management science*, 37(5):519–531, 1991.

[85] D. Krishnan, P. Lin, and A. M. Yip. A primal-dual active-set method for non-negativity constrained total variation deblurring problems. *IEEE Transactions on Image Processing*, 16(11):2766–2777, 2007.

[86] 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.

[87] K. Kunisch and A. Rösch. Primal-dual active set strategy for a general class of constrained optimal control problems. *SIAM Journal on Optimization*, 13(2):321–334, 2002.

[88] P. Lötstedt. Solving the minimal least squares problem subject to bounds on the variables. *BIT Numerical Mathematics*, 24(2):205–224, 1984.

[89] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[90] I. Maros and C. Mészáros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11(1-4):671–681, 1999.

[91] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and Jeremy Kubica. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2013.

[92] J. J. Moré and G. Toraldo. Algorithms for bound constrained quadratic programming problems. *Numerische Mathematik*, 55(4):377–400, 1989.

[93] K. G. Murty. Note on a bard-type scheme for solving the complementarity problem. *Opsearch*, 11(2-3):123–130, 1974.

[94] A. E. Nemirovski and M. J. Todd. Interior-point methods for optimization. *Acta Numerica*, 17:191–234, 2008.

[95] Y. Nesterov and A. Nemirovskii. *Interior point polynomial algorithms in convex programming.* Studies in Applied and Numerical Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1994.

[96] A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22Nd International Conference on Machine Learning*, pages 625–632, New York, NY, USA, 2005. ACM.

[97] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer, New York, NY, USA, Second edition, 2006.

[98] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975.

[99] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[100] 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.

[101] A. Potschka, C. Kirches, H. G. Bock, and J. P. Schlöder. Reliable solution of convex quadratic programs with parametric active set methods. *Interdisciplinary Center for Scientific Computing, Heidelberg University, Im Neuenheimer Feld*, 368:69120, 2010.

[102] M. J. D. Powell. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization*, pages 283–298. Academic Press, New York, 1969.

[103] O. Romanko, A. G. Hadigheh, and T. Terlaky. Multiobjective optimization via parametric optimization: models, algorithms, and applications. In *Modeling and Optimization: Theory and Applications*, pages 77–119. Springer New York, 2012.

[104] Y. Saad. *Iterative Methods for Sparse Linear Systems.* SIAM, Philadelphia, PA, USA, Second edition, 2003.

[105] K. Scheinberg. An efficient implementation of an active set method for svms. *Journal of Machine Learning Research*, 7:2237–2257, December 2006.

[106] C. Schmid, L.T. Biegler, and Carnegie-Mellon University. Engineering Design Research Center. *Quadratic Programming Methods for Tailored Reduced Hessian SQP*. Engineering Design Research Center, Carnegie-Mellon University, 1993.

[107] S. Sra, S. Nowozin, and S. J. Wright. *Optimization for Machine Learning.* Neural information processing series. MIT Press, 2012.

[108] G. Stadler. Semismooth Newton and augmented Lagrangian methods for a simplified friction problem. *SIAM Journal on Optimization*, 15(1):39–62, 2004.

[109] X. Suo and R. Tibshirani. An ordered lasso and sparse time-lagged regression. *arXiv preprint arXiv:1405.6447*, 2014.

[110] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[111] R. J. Tibshirani, H. Hoefling, and R. Tibshirani. Nearly-isotonic regression. *Technometrics*, 53(1):54–61, 2011.

[112] P. L. Toint. Non-monotone trust-region algorithms for nonlinear optimization subject to convex constraints. *Mathematical Programming*, 77(3):69–94, 1997.

[113] 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.

[114] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems.* Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, New York, NY, USA, 2003.

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

[116] V. Vapnik and C. Cortes. Support vector networks. *Machine Learning*, 20(3):273–297, 1995.

[117] J. M. Varah. A lower bound for the smallest singular value of a matrix. *Linear Algebra and its Applications*, 11(1):3–5, 1975.

[118] R. S. Varga. On diagonal dominance arguments for bounding $\|A^{-1}\|_\infty$. *Linear Algebra and its Applications*, 14(3):211–217, 1976.

[119] S. J. Wright. *Primal-Dual Interior-Point Methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[120] E. A. Yildirim and S. J. Wright. Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization*, 12(3):782–810, 2002.

[121] B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 694–699, New York, NY, USA, 2002.

[122] M. Zapletal. Isotonic regression implementation in apache spark, 2015.

Appendix A

Primal-Dual Active-Set as a Semi-Smooth Newton Method

In this appendix, we show that Algorithm 8 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 [69].

Theorem A.0.1. *Let $\{(x_k, y_k, z_k^\ell, z_k^u)\}$ be generated by Algorithm 8 with Step 6 employing Algorithm 9, 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* (2.3) *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*

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

Proof. To simplify the proof, let us assume that $\ell = -\infty$ so that problem (1.4) 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 8 and the semi-smooth Newton method. Furthermore, it follows from (3.2)–(3.4) that

$$Hx_0 + c - A^T y_0 + z_0^u = 0 \quad \text{and} \quad Ax_0 - b = 0. \tag{A.1}$$

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:

$$\mathrm{I} := \{i : i \in \mathcal{I}_0 \ \text{ and } \ [x_0]_i \le u_i\} \implies [z_0^u]_i = 0; \tag{A.2a}$$

$$\mathrm{II} := \{i : i \in \mathcal{A}_0^u \ \text{ and } \ [z_0^u]_i \le 0\} \implies [x_0]_i = u_i; \tag{A.2b}$$

$$\mathrm{III} := \{i : i \in \mathcal{I}_0 \ \text{ and } \ [x_0]_i > u_i\} \implies [z_0^u]_i = 0; \tag{A.2c}$$

$$\mathrm{IV} := \{i : i \in \mathcal{A}_0^u \ \text{ and } \ [z_0^u]_i > 0\} \implies [x_0]_i = u_i. \tag{A.2d}$$

Here, the implications after each set follow from Step 2 of Algorithm 7. Next, (3.14) implies

$$\mathcal{I}_1 \leftarrow \mathrm{I} \cup \mathrm{II} \ \text{ and } \ \mathcal{A}_1 \leftarrow \mathrm{III} \cup \mathrm{IV}. \tag{A.3}$$

Algorithm 8 computes the next iterate as the unique point (x_1, y_1, z_1^u) satisfying

$$[z_1^u]_{\mathcal{I}_1} = 0, \quad [x_1]_{\mathcal{A}_1} = u_{\mathcal{A}_1}, \quad Hx_1 + c - A^T y_1 + z_1^u = 0, \quad \text{and} \quad Ax_1 - b = 0. \tag{A.4}$$

Now, let us consider one iteration of the semi-smooth Newton method on the function KKT defined by (2.3) using the slant derivative function M. It follows from (A.2), Table A.1, and the definition of M that the semi-smooth Newton system may be written as

$$\begin{pmatrix} H_{\mathrm{I,I}} & H_{\mathrm{I,II}} & H_{\mathrm{I,III}} & H_{\mathrm{I,IV}} & A^T_{\mathcal{N},\mathrm{I}} & I & 0 & 0 & 0 \\ H_{\mathrm{II,I}} & H_{\mathrm{II,II}} & H_{\mathrm{II,III}} & H_{\mathrm{II,IV}} & A^T_{\mathcal{N},\mathrm{II}} & 0 & I & 0 & 0 \\ H_{\mathrm{III,I}} & H_{\mathrm{III,II}} & H_{\mathrm{III,III}} & H_{\mathrm{III,IV}} & A^T_{\mathcal{N},\mathrm{III}} & 0 & 0 & I & 0 \\ H_{\mathrm{IV,I}} & H_{\mathrm{IV,II}} & H_{\mathrm{IV,III}} & H_{\mathrm{IV,IV}} & A^T_{\mathcal{N},\mathrm{IV}} & 0 & 0 & 0 & I \\ A_{\mathcal{N},\mathrm{I}} & A_{\mathcal{N},\mathrm{II}} & A_{\mathcal{N},\mathrm{III}} & A_{\mathcal{N},\mathrm{IV}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta x_{\mathrm{I}} \\ \Delta x_{\mathrm{II}} \\ \Delta x_{\mathrm{III}} \\ \Delta x_{\mathrm{IV}} \\ -\Delta y \\ \Delta z_{\mathrm{I}} \\ \Delta z_{\mathrm{II}} \\ \Delta z_{\mathrm{III}} \\ \Delta z_{\mathrm{IV}} \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ [z_0^u]_{\mathrm{II}} \\ [u - x_0]_{\mathrm{III}} \\ 0 \end{pmatrix}. \tag{A.5}$$

Table A.1: 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 (A.5) combined with (A.1) yield

$$Ax_1 - b = A(x_0 + \Delta x) - b = Ax_0 - b + A\Delta x = 0 \text{ and} \tag{A.6a}$$

$$Hx_1 + c - A^T y_1 + z_1^u = H(x_0 + \Delta x) + c - A^T(y_0 + \Delta y) + z_0^u + \Delta z = 0, \tag{A.6b}$$

while the last four blocks of equations of (A.5) and (A.2) imply

$$\Delta z_{\mathrm{I}} = 0 \implies [z_1^u]_{\mathrm{I}} = [z_0^u + \Delta z]_{\mathrm{I}} = 0 \tag{A.7}$$

$$\Delta z_{\mathrm{II}} = -[z_0^u]_{\mathrm{II}} \implies [z_1^u]_{\mathrm{II}} = [z_0^u + \Delta z]_{\mathrm{II}} = 0 \tag{A.8}$$

$$\Delta x_{\mathrm{III}} = [u - x_0]_{\mathrm{III}} \implies [x_1]_{\mathrm{III}} = [x_0 + \Delta x]_{\mathrm{III}} = u_{\mathrm{III}} \tag{A.9}$$

$$\Delta x_{\mathrm{IV}} = 0 \implies [x_1]_{\mathrm{IV}} = [x_0 + \Delta x]_{\mathrm{IV}} = u_{\mathrm{IV}} \tag{A.10}$$

so that

$$[z_1^u]_{\mathcal{I}_1} = 0 \;\text{ and }\; [x_1]_{\mathcal{A}_1} = u_{\mathcal{A}_1}. \tag{A.11}$$

It now follows from (A.4), (A.6), and (A.11) that (x_1, y_1, z_1^u) generated by the semi-smooth Newton method is the same as that generated by Algorithm 8. □

Appendix B

The pypdas Package for Convex QP

In Chapter 4, we presented a PDAS algorithm framework for certain class of QPs that supports inexact subproblem solves. In fact, all of the implementation of the three PDAS variants described therein are built and tested on top of `pypdas`, a Python package that provides a generic PDAS method framework with support of inexact subproblem solves. We give a brief instruction of the function call of `pypdas` for solving convex QPs. The source code is maintained on the following Github repository:

https://github.com/zhh210/pypdas

The `pypdas` interface attempts to emulate that of MATLAB which assumes the convex QP to be in the form of

$$\begin{aligned} \min_{x\in\mathbb{R}^n}\ & \frac{1}{2}x^T Hx + c^T x \\ \text{s.t. } & \text{Aeq}x = \text{beq} \\ & \text{bl} \le Ax \le \text{bu} \\ & \ell \le x \le u. \end{aligned} \tag{B.1}$$

In this manner, a subspace minimizer is defined by solving a linear system with a reduced space matrix of the form (B.2). In `pypdas`, this system is solved using `MINRES` [98, 23], a Krylov iterative linear system solver available in the SciPy package. Despite a lack of theoretical guarantee when solving general problems of the form (B.1), our practical experience suggests that this procedure generally leads to a productive partition update.

$$\begin{pmatrix} H_{\mathcal{A}\mathcal{A}} & \mathrm{Aeq}_{\mathcal{A}}^T & \mathrm{Aeq}_{\mathcal{A}}^T \\ \mathrm{Aeq}_{\mathcal{A}} & 0 & 0 \\ \mathrm{A}_{\mathcal{A}} & 0 & 0 \end{pmatrix} \tag{B.2}$$

The design of `pypdas` follows the framework of Algorithm 12 but supports solving the more general QP (1.4). We list all optional parameters and their default values that are used in the `pypdas` package. Users could specify their own preferred value via the function call of `pdas` or `ipdas`.

Table B.1: Default parameter settings in `pypdas`.

Parameter	Default	Usage	Note
`OptTol`	10^{-6}	`pdas, ipdas`	ϵ_{opt}, tolerance for KKT error
`ResTol`	10^{-6}	`pdas, ipdas`	ϵ_{num}, tolerance for linear system solve
`MaxItr`	500	`pdas, ipdas`	maximum number of iteration
`CG_res_absolute_hard`	10^{-2}	`ipdas`	ϵ_{res}, hard tolerance for linear system solve
`identified_estimated_ratio`	0.5	`ipdas`	θ, minimum proportion of $\lvert\mathcal{V}\rvert$ identified
`fun_estinv` [1]	100	`ipdas`	a function to estimate matrix inverse norm

[1] This function is optionally provided by the users to utilize the structure of (B.2) for the problem of interest. The package provides optional functions such as: power iteration, or Algorithm 14 when (B.2) is a (perturbed) H-matrix.

Finally we provide a step-to-step guidance on how one could call `pypdas` to solve a convex QP in Python 2.7.

(I) Prepare Data

In the favor of the unified interface for both sparse and dense matrix operations, `pypdas` chooses the matrix data structure implemented in `cvxopt`. The following piece of code creats random sparse coefficient matrices of (B.1).

```
from pdas.toolbox import pdas,ipdas
from pdas.randutil import sprandsym, sp_rand
n = 100  # number of variables
m = 1    # number of equality
mi = 1   # number of inequality
# Generate some random data (sparse)
H    = sprandsym(n)
c    = sp_rand(n,1,1)
A    = sp_rand(mi,n,1)
bl   = sp_rand(mi,1,1)
bu   = bl + 1
Aeq  = sp_rand(m,n,0.8)
beq  = sp_rand(m,1,1)
l    = sp_rand(n,1,1)
u    = l + 1
x0   = sp_rand(n,1,0.8) # optional initial solution
```

(II) PDAS with exact subproblem solve

To solve a QP with exact subproblem solve, one simply callss function `pdas`

```
    pdas(H,c,Aeq,beq,A,bl,bu,l,u,x0)
```

The output should be somewhat

```
================================================================================
Iter     Obj           KKT        |AL|  |AU|  |I|   |V|   |cV|   invnorm  Krylov
================================================================================
   0    7.21e+02      4.71e+01      57    15    28    49     0   2.00e+00    32
   1    6.23e+02      2.97e+01      56    10    34    26     0   2.00e+00    34
   2    6.86e+02      7.03e+00      55    18    27    12     0   2.00e+00    32
   3    6.93e+02      3.23e+00      53    22    25     5     0   2.00e+00    28
   4    6.94e+02      5.86e-01      54    24    22     2     0   2.00e+00    25
   5    6.94e+02      2.88e-08      54    22    24     0     0   2.00e+00    28
--------------------------------------------------------------------------------
Problem Status          : Optimal
Total Iterations        : 5
Total Krylov-iterations : 179
Avg norm(r)/norm(r0    ): 6.25e-08
Avg norm(r)             : 3.56e-07
Time Elapsed            : 6.00e-02
--------------------------------------------------------------------------------
```

To solve a BQP with exact subproblem solve, one simply calls function `pdas`

```
pdas(H=H,c=c,l=l,u=u,x0=x0)
```

The output should be somewhat

```
================================================================================
Iter     Obj           KKT        |AL|  |AU|  |I|   |V|   |cV|   invnorm  Krylov
================================================================================
   0    7.17e+02      5.06e+01      57    15    28    45     0   2.00e+00    27
   1    5.88e+02      2.90e+01      56     9    35    22     0   2.00e+00    33
   2    6.57e+02      2.26e+00      58    17    25     8     0   2.00e+00    28
   3    6.64e+02      7.94e-01      57    18    25     2     0   2.00e+00    27
   4    6.65e+02      2.88e-02      58    19    23     1     0   2.00e+00    23
   5    6.65e+02      1.05e-03      57    19    24     1     0   2.00e+00    27
   6    6.65e+02      3.17e-13      58    19    23     0     0   2.00e+00    26
--------------------------------------------------------------------------------
Problem Status           : Optimal
Total Iterations         : 6
Total Krylov-iterations : 191
Avg norm(r)/norm(r0    ): 4.75e-08
Avg norm(r)              : 3.53e-07
Time Elapsed             : 6.00e-02
--------------------------------------------------------------------------------
```

(III) PDAS with inexact subproblem solve

To solve a QP with iexact subproblem solve, one simply calls function `ipdas`

```
    ipdas(H,c,Aeq,beq,A,bl,bu,l,u,x0,identified_estimated_ratio = 0.5)
```

Note that besides `identified_estimated_ratio` any of the parameters in Table B.1 could be specified in the function. The output should be somewhat

```
================================================================================
Iter     Obj             KKT         |AL|  |AU|  |I|   |V|   |cV|   invnorm  Krylov
================================================================================
   0    7.21e+02      4.72e+01     57    15    28    49    42   2.00e+00    19
   1    6.17e+02      2.50e+01     54    11    35    28    25   2.00e+00    17
   2    6.88e+02      8.24e+00     57    18    25    16    16   2.00e+00    23
   3    6.93e+02      3.62e+00     51    22    27     6     6   2.00e+00    22
   4    6.94e+02      1.10e-01     54    23    23     1     1   2.00e+00    20
   5    6.94e+02      2.88e-08     54    22    24     0     0   2.00e+00    28
--------------------------------------------------------------------------------
Problem Status          : Optimal
Total Iterations        : 5
Total Krylov-iterations : 129
Avg norm(r)/norm(r0)    : 5.25e-04
Avg norm(r)             : 2.34e-02
Time Elapsed            : 2.60e-01
Total Krylov4estimation : 0
--------------------------------------------------------------------------------
```

To solve a BQP with inexact subproblem solve, one simply calls function `ipdas`

```
    ipdas(H=H,c=c,l=l,u=u,x0=x0,identified_estimated_ratio = 0.5)
```

The output should be somewhat

```
================================================================================
Iter     Obj           KKT        |AL|  |AU|  |I|   |V|   |cV|   invnorm  Krylov
================================================================================
   0    7.17e+02      5.06e+01      57    15    28    45    43   2.00e+00    13
   1    5.82e+02      1.98e+01      56     9    35    22    20   2.00e+00    16
   2    6.57e+02      2.76e+00      59    16    25     8     7   2.00e+00    15
   3    6.65e+02      1.02e-03      57    19    24     1     1   2.00e+00    20
   4    6.65e+02      3.17e-13      58    19    23     0     0   2.00e+00    26
--------------------------------------------------------------------------------
Problem Status          : Optimal
Total Iterations        : 4
Total Krylov-iterations : 90
Avg norm(r)/norm(r0)    : 4.13e-04
Avg norm(r)             : 2.08e-02
Time Elapsed            : 1.80e-01
Total Krylov4estimation : 0
--------------------------------------------------------------------------------
```

Biography

Zheng Han is a Ph.D. candidate in the Department of Industrial and Systems Engineering (ISE) at Lehigh University. He is keenly interested in almost all areas of mathematical optimization and his research focus is on nonlinear optimization algorithms with applications in optimal control and statistical learning. He received his B.S. degree in Industrial Engineering with B.S. double degree in Mathematics and Applied Mathematics from Tianjin University in 2010 and worked at NEC Laboratories America on data mining of massive time series in the summer of 2014. He is a P.C. Rossin Doctoral Fellow, winner of the 2015 Van Hoesen Family ISE Best Publication Award, and winner of the 2015 American Express Machine Learning Contest. Zheng will join American Express as a Data Scientist.