ACCELERATED FIRST ORDER METHODS FOR CONVEX COMPOSITE QUADRATIC PROGRAMMING: THEORY, ALGORITHMS, AND GPU IMPLEMENTATION

KAIHUANG CHEN

PhD

The Hong Kong Polytechnic University

2026

The Hong Kong Polytechnic University

Department of Applied Mathematics

Accelerated First Order Methods for Convex Composite Quadratic Programming: Theory, Algorithms, and GPU Implementation

Kaihuang Chen

A thesis submitted in partial fulfilment of the requirements

for the degree of Doctor of Philosophy

August 2025

Certificate of Originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____(Signed)

Kaihuang Chen (Name of student)

To my family

Abstract

Convex composite quadratic programming (CCQP) plays a central role in machine learning, finance, and engineering. This thesis develops efficient algorithms for large-scale CCQP problems, with an emphasis on accelerated first-order methods that exploit the massive parallelism of modern GPUs.

On the theoretical side, we propose two Halpern Peaceman–Rachford (HPR) methods for solving different primal formulations of CCQP. Both enjoy an $O(1/k)$ complexity bound in terms of the Karush–Kuhn–Tucker (KKT) residual and objective error. By introducing auxiliary slack variables and appropriate proximal operators, each iteration admits an explicit closed-form update, avoiding linear system solves and making the methods well-suited for GPU acceleration. More importantly, we introduce a dual HPR method designed for solving the restricted Wolfe dual of CCQP problems, which also enjoys a fast $O(1/k)$ complexity bound in terms of the KKT residual and the objective error. One distinctive feature of the dual approach is that, instead of working with the primal formulations, it builds on the novel restricted Wolfe dual introduced in recent years. It also leverages the symmetric Gauss–Seidel technique to simplify subproblem updates without introducing auxiliary slack variables that typically lead to slow convergence. By restricting updates to the range space of the Hessian of the quadratic objective function, the dual approach employs proximal operators of smaller spectral norms to speed up the convergence. Shadow sequences are elaborately constructed to deal with the range space constraints.

On the implementation side, we introduce three key enhancements. First, leveraging the $O(1/k)$ complexity results in terms of the KKT residual of the HPR methods, we develop adaptive restart and penalty parameter updating strategies to accelerate practical convergence. Second, we design a GPU-oriented implementation that avoids linear solves and matrix inverse approximations, relying solely on matrix-vector multiplications and vector-level operations. Third, the solver operates without requiring the explicit matrix form of the quadratic objective, making it highly scalable to extremely large CCQP instances such as quadratic assignment and LASSO regression.

In the numerical study, we first present case analyses comparing GPU versus CPU performance, HPR against preconditioned alternating direction method of multipliers (without acceleration), restricted Wolfe dual versus primal formulations, and the impact of algorithmic enhancements on solver efficiency and convergence. Extensive experiments demonstrate that the proposed methods achieve superior speed and scalability compared to state-of-the-art solvers.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Defeng Sun, for his expert guidance and invaluable support throughout the course of my research. His profound knowledge in computing and applied optimization, along with his insightful advice and constructive feedback, have been instrumental in shaping this thesis. I am especially appreciative of his encouragement, patience, and mentorship, which have greatly contributed to both my academic and personal growth.

I am also sincerely thankful to my collaborators, Professor Yancheng Yuan, Guojun Zhang, and Professor Xinyuan Zhao, for their contributions to our joint work and for the stimulating discussions that enriched this research. Their collaboration has been essential in advancing the results presented in this thesis.

In addition, I would like to acknowledge Qian Li and Chiyu Ma for their collaboration and support in our research project, which laid the groundwork for subsequent developments.

Finally, I wish to express my gratitude to Alexis Montoison for his helpful introduction to Julia and GPU computation, which greatly facilitated the implementation of the algorithms developed in this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations and related topics

In this thesis, we focus on implementing efficient algorithms for solving convex composite quadratic programming (CCQP) problems. Consider the CCQP with the following formulation:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \frac{1}{2}\langle x, Qx\rangle + \langle c, x\rangle + \phi(x) \\ \text{s.t.} \quad & Ax \in \mathcal{K}, \end{aligned} \tag{1.1}$$

where $Q : \mathbb{R}^n \to \mathbb{R}^n$ is a self-adjoint positive semidefinite linear operator, $c \in \mathbb{R}^n$ is a given vector, and $\phi : \mathbb{R}^n \to (-\infty, +\infty]$ is a proper, closed, and convex function. Moreover, $A : \mathbb{R}^n \to \mathbb{R}^m$ is a linear operator, and $\mathcal{K} := \{y \in \mathbb{R}^m \mid -\infty \le l_i \le y_i \le u_i \le +\infty, 1 \le i \le m\}$ is a simple polyhedral set. This flexible framework unifies many problem classes of practical significance in large-scale applications. Below, we briefly discuss some representative instances of CCQP.

1.1.1 Representative problem classes

Convex Quadratic Programming. If function $\phi(\cdot)$ is the indicator function of a convex set $\delta_C(\cdot)$, where $\mathcal{C}$ is a similar polyhedral set to $\mathcal{K}$, as $\mathcal{C} := \{x \in \mathbb{R}^n \mid -\infty \le L_i \le x_i \le U_i \le +\infty, 1 \le i \le n\}$, then the problem is called convex quadratic

programming (CQP):

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \frac{1}{2}\langle x, Qx \rangle + \langle c, x \rangle \\ \text{s.t.} \quad & Ax \in \mathcal{K}, \\ & x \in \mathcal{C}. \end{aligned}$$

CQP problems are ubiquitous in operations research and engineering, including scheduling (Skutella, 2001), network optimization (Lavei et al., 2011), and energy systems (Chen et al., 2014). It has been extensively studied for several decades (Cottle, 1963, 1964; Dantzig, 1961; Gould and Toint, 2000; Gould et al., 2001; Nocedal and Wright, 2006).

Least absolute shrinkage and selection operator (LASSO). An important special case of problem (1.1), obtained by setting $\phi(x) = \lambda\|x\|_1$, is the LASSO problem (Tibshirani, 1996):

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1.$$

This formulation is central in high-dimensional statistics (Bühlmann and Van De Geer, 2011), compressed sensing (Donoho, 2006), and machine learning (Muthukrishnan and Rohini, 2016). The number of features n can easily reach millions in genomics, image reconstruction, and natural language processing (Chang and Lin, 2011), demanding scalable solvers.

Support Vector Machines (SVM). The training of linear SVM classifiers (Cortes and Vapnik, 1995) can be cast as a convex quadratic programming:

$$\begin{aligned}
&\min_{w,\,b,\,\xi} \ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{m}\xi_i \\
&\text{s.t.} \ \ y_i(w^\top x_i + b) \geq 1 - \xi_i, \ \text{for } i = 1, \ldots, m, \\
&\qquad \xi \geq 0,
\end{aligned}$$

where w is the classifier weight vector and ξ_i are slack variables. SVMs are among the most widely used methods in machine learning, with applications in text classification, bioinformatics, and computer vision. The LIBLINEAR (Fan et al., 2008) and LIBSVM repositories list datasets with millions of features and billions of samples.

Optimal Control and Model Predictive Control (MPC). The finite-horizon MPC (Borrelli et al., 2017) problem can be formulated in sparse stage-wise form as the quadratic programming:

$$\begin{aligned}
&\min_{x_k,u_k} \ \frac{1}{2}\sum_{k=0}^{N-1}\left\langle \begin{bmatrix} x_k \\ u_k \end{bmatrix}, \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix}\begin{bmatrix} x_k \\ u_k \end{bmatrix}\right\rangle + \frac{1}{2}\langle x_N, Px_N\rangle \\
&\text{s.t.} \ \ x_{k+1} = Ax_k + Bu_k, \quad k = 0, \ldots, N-1, \\
&\qquad F_x x_k \leq f_x, \quad k = 0, \ldots, N, \\
&\qquad F_u u_k \leq f_u, \quad k = 0, \ldots, N-1, \\
&\qquad F_f x_N \leq f_f.
\end{aligned}$$

The variable $x_k \in \mathbb{R}^n$ is the state (x_0 is given), $u_k \in \mathbb{R}^m$ is the input, $Q \succeq 0$ and $P \succeq 0$ are state weighting matrices, $R \succ 0$ is the input weighting matrix, and (F_x, f_x), (F_u, f_u), (F_f, f_f) encode polyhedral state, input, and terminal constraints, respectively. These problems are critical in robotics, aerospace, and process engineering with comprehensive studies (Garcia et al., 1989; Rawlings et al., 2020). In

industrial applications, horizons N can be large and state dimensions high, leading to structured QP problems with hundreds of thousands of variables.

Portfolio Optimization. In finance, the classical mean–variance model introduced by Markowitz (1952) is a convex quadratic programming:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \ & \frac{1}{2}\langle x, Qx \rangle - \langle \mu, x \rangle \\ \text{s.t.} \ \ & e^\top x = 1, \\ & x \geq 0, \end{aligned}$$

where Q is the covariance matrix of asset returns, μ is the expected return vector, and e is the vector of all ones in $\mathbb{R}^n$. Large-scale extensions of this model are relevant for index tracking (Benidis et al., 2017), sparse portfolios (Brodie et al., 2009), and risk-parity portfolios (Rockafellar et al., 2000). In practice, n may be in the order of tens or hundreds of thousands of assets.

Quadratic Assignment Problem (QAP) and Relaxations. The QAP (Koopmans and Beckmann, 1957) can be approximated via convex quadratic relaxations, which are special cases of CCQP. Such relaxations arise in combinatorial optimization, facility location, and graph matching problems (Burkard et al., 1997; Loiola et al., 2007; Cela, 2013; Anstreicher and Brixius, 2001). Even the convex relaxations involve extremely large matrix variables and remain computationally demanding.

QAP is defined as follows:

$$\begin{aligned} \min_{X \in \mathbb{R}^{d \times d}} \quad & \langle \mathrm{vec}(X), (\widehat{B} \otimes \widehat{A}) \mathrm{vec}(X) \rangle \\ \text{s.t.} \quad & Xe = e, \\ & X^\top e = e, \\ & X \geq 0, \\ & X \in \{0,1\}^{d \times d}, \end{aligned}$$

where $\widehat{A}, \widehat{B} \in \mathbb{S}^d$ are given matrices, e is the vector of all ones in $\mathbb{R}^d$. A lower bound of the QAP is given by the following CQP relaxation (Anstreicher and Brixius, 2001):

$$\begin{aligned} \min_{\mathrm{vec}(X) \in \mathbb{R}^{d^2}} \quad & \langle \mathrm{vec}(X), \widehat{Q}\, \mathrm{vec}(X) \rangle \\ \text{s.t.} \quad & (e^\top \otimes I_d)\, \mathrm{vec}(X) = e, \\ & (I_d \otimes e^\top)\, \mathrm{vec}(X) = e, \\ & \mathrm{vec}(X) \geq 0. \end{aligned}$$

The matrix formulation $\widehat{Q} \in \mathbb{S}^{d^2}$ is defined by $\widehat{Q} = (\widehat{B} \otimes \widehat{A} - I \otimes S - T \otimes I)$, with $S, T \in \mathbb{S}^d$ computed according to $\widehat{A}, \widehat{B}$ described in Section 5.3.

These representative problem classes illustrate the versatility of the CCQP framework. They span machine learning, finance, engineering, and operations research, and often lead to optimization instances of unprecedented scale. Developing efficient, scalable algorithms for CCQP is therefore a topic of both theoretical significance and practical urgency.

1.1.2 Related algorithms

CQP has been a central topic in optimization for decades, with developments ranging from classical theory to modern GPU-based solvers. We briefly review the main algorithmic paradigms that motivate our work.

Early Developments. The Karush–Kuhn–Tucker (KKT) conditions established necessary conditions for constrained optimality, forming the theoretical basis of CQP (Karush, 1939; Kuhn and Tucker, 1951). Early algorithms, such as Wolfe's pivoting method (Wolfe, 1959) and Fletcher's general QP algorithm (Fletcher, 1971), introduced the active-set paradigm: iteratively identifying active constraints and solving equality-constrained subproblems. These approaches achieved exact solutions for small- to medium-scale problems, and a comprehensive and practical guide can be found in Gill et al. (1981) for active-set methods.

Interior-Point Methods. A major breakthrough came from Karmarkar's polynomial-time algorithm for LP (Karmarkar, 1984), which led to primal-dual interior-point frameworks for QP (Kojima et al., 1989; Nesterov and Nemirovskii, 1994; Wright, 1997). These methods scale to much larger problems with strong complexity guarantees. Representative solvers include LOQO (Vanderbei, 1999), OOQP (Gertz and Wright, 2003), and the more recent Clarabel (Goulart and Chen, 2024). Commercial suites such as MOSEK, Gurobi, and CPLEX (MOSEK ApS, 2025; Gurobi Optimization, LLC, 2025; IBM ILOG CPLEX, 2022) integrate IPM and active-set techniques into industrial-grade packages. Although interior-point methods are well known for their fast convergence rate, a high-quality and efficient solution is required for the corresponding linear system.

Augmented Lagrangian methods. The augmented Lagrangian method (ALM), originally introduced by Hestenes (1969) and Powell (1969), has played a central role in constrained optimization, including quadratic programming (QP). By augmenting the classical Lagrangian with quadratic penalty terms, ALM overcomes the ill-conditioning issues inherent in pure penalty approaches while retaining favorable convergence properties. Rockafellar (1973, 1976) established a rigorous theoretical

foundation, showing that ALM can achieve global convergence under mild assumptions. Building on this foundation, a number of practical solvers have been developed, such as QPALM (Hermans et al., 2019), QPPAL (phase-II) (Liang et al., 2022), and ProxQP (Bambade et al., 2025).

Operator-Splitting Methods. With the growth of machine learning and control applications, first-order methods such as operator-splitting approaches (Lions and Mercier, 1979; Eckstein and Bertsekas, 1992; Boyd et al., 2011; Fazel et al., 2013) have become central tools for large-scale QPs. These methods trace back to the operator-splitting literature of the 1970s, particularly the alternating direction method of multipliers (ADMM) (Glowinski and Marroco, 1975; Gabay and Mercier, 1976), which introduced decomposable augmented Lagrangian techniques for variational inequalities and convex optimization. The key idea is to exploit separability in the objective by alternating between primal variable updates while maintaining dual feasibility through multiplier adjustments, thereby enabling decomposition across variable blocks and yielding substantial computational advantages.

Solvers such as SCS (O'donoghue et al., 2016; O'Donoghue, 2021), OSQP (Stellato et al., 2020), and QPPAL (phase I) (Liang et al., 2022) have been developed based on different QP formulations. Although their subproblems are relatively simple, performance on large-scale problems may be limited by reliance on direct linear system solvers or by iterative schemes that require well-designed preconditioners. More recently, the award-winning solver PDLP (Applegate et al., 2021, 2023; Lu and Yang, 2023), which is built on the primal–dual hybrid gradient method (Esser et al., 2010; Chambolle and Pock, 2011), has demonstrated the scalability of first-order methods for large-scale LP. Two key insights from PDLP are the effectiveness of restarting at the average iterate and the efficiency of GPU acceleration for factorization-free methods.

Accelerated Operator-Splitting Methods. Recently, by leveraging the Halpern iteration (Halpern, 1967; Sabach and Shtern, 2017; Lieder, 2021), substantial progress has been achieved to accelerate the (semi-proximal) Peaceman–Rachford (PR) method (Eckstein and Bertsekas, 1992; Lions and Mercier, 1979). Notably, Zhang et al. (2022) developed the HPR method by applying the Halpern iteration to the PR scheme without proximal terms, establishing an $O(1/k)$ convergence rate in terms of both the KKT residual and the objective error. Building on this, Sun et al. (2025) reformulated the semi-proximal PR method as a degenerate proximal point method (dPPM) (Bredies et al., 2022) with a positive semidefinite preconditioner, and further applied the Halpern iteration to derive the semi-proximal HPR method, which also enjoys $O(1/k)$ complexity guarantees. More recently, Chen et al. (2024a) proposed HPR-LP, a GPU-accelerated solver for large-scale linear programming, which outperformed the award-winning PDLP solver (Applegate et al., 2021, 2023; Lu and Yang, 2023) by a significant margin.

GPU-Oriented Solvers. GPUs provide massive parallelism well suited for matrix–vector operations that dominate iterative optimization algorithms. Over the past decade, GPU acceleration has evolved from prototypes to production-ready solvers for linear, quadratic/conic, and nonlinear problems, with speedups largely stemming from recasting iterations into memory-bound primitives such as sparse matrix–vector multiplications (SpMV), map/reduce, and axpy operations. First-order splitting methods have been particularly amenable to GPU implementation: for instance, cuOSQP (Schubiger et al., 2020), a GPU version of OSQP (Stellato et al., 2020), employs preconditioned conjugate gradients with a GPU-friendly Jacobi preconditioner; NVIDIA's cuOpt provides a GPU-accelerated PDLP for large-scale LP and routing pipelines (NVIDIA, 2024, 2025a,b); and more recent solvers such as PDQP (Lu and Yang, 2025) and PDCS (Lin et al., 2025) extend primal–dual hybrid gradi-

ent methods to quadratic and conic programs. Interior-point approaches have also adopted GPU support, with CuClarabel (Chen et al., 2024b) and MadNLP (Shin et al., 2024) exploiting GPU-accelerated linear algebra in conic and nonlinear programming. These developments underscore the increasing importance of GPU-aware design in modern large-scale optimization.

In summary, the evolution of CQP algorithms has progressed from active-set and interior-point methods to modern augmented Lagrangian methods, first-order and GPU-based solvers. Despite these advances, achieving both theoretical acceleration and practical robustness for CCQP, including CQP, remains challenging. Motivated by the strong theoretical guarantees and empirical success of HPR-based methods, as well as the rapid advancement of GPU hardware, we pursue the development of a GPU-accelerated HPR framework for solving large-scale CCQP problems, with a particular emphasis on CQP as a fundamental subclass.

1.2 Contributions

This thesis develops efficient accelerated algorithms that fully leverage GPU architectures for solving large-scale CCQP problems. The main contributions are summarized as follows:

Algorithmic Framework. We propose two HPR methods for solving different primal formulations of CCQP. The complexity bounds of both methods are $O(1/k)$ in terms of KKT residual and objective error. We further introduce a dual HPR method for solving the restricted Wolfe dual formulation. By incorporating the restricted Wolfe dual formulation and symmetric Gauss–Seidel decomposition, the algorithm achieves both robustness and computational efficiency, while inheriting the $O(1/k)$ KKT convergence guarantee of the HPR framework. The advantages of the restricted Wolfe dual formulation are discussed for both computational efficiency and

theoretical effectiveness. The shadow sequences are carefully constructed to analyze the relation between theory and practice.

Algorithmic Enhancements. Building on the $O(1/k)$ complexity results in terms of the KKT residual of the HPR method, we design and analyze practical strategies—such as adaptive restart and penalty parameter updates—that significantly enhance the convergence speed and robustness of the proposed algorithms.

GPU-Oriented Implementation. We implement a solver tailored for GPU architectures that avoids linear system solves or matrix inverse approximations, relying solely on matrix-vector multiplications and vector-level operations. It perfectly suits the case when the quadratic objective is given in an implicit form, such as in quadratic assignment and LASSO regression problems. This design ensures excellent scalability and efficient use of parallel hardware.

Extensive Numerical Evaluation. We conduct case studies comparing GPU and CPU performance, HPR and preconditioned ADMM, restricted Wolfe dual and primal formulations, and the impact of algorithmic enhancements on solver efficiency and convergence. Extensive experiments on large-scale benchmarks further demonstrate that the proposed algorithms consistently surpass state-of-the-art methods in both speed and scalability.

1.3 Thesis organization

The thesis is structured as follows. In Chapter 2, we introduce some notations that will be used in the later chapters. We introduce the restricted Wolfe dual formulation and the base method HPR that will be used throughout the thesis. In Chapter 3, we present two formulations of the primal HPR method for convex composite quadratic

programming. Their convergence properties are analyzed, and computational details are discussed. In Chapter 4, we derive the HPR method based on the restricted Wolfe dual formulation, together with the sGS decomposition. The implementation details and convergence of the proposed method are presented. In Chapter 5, we first present case studies to demonstrate the effectiveness of the numerical enhancements, the GPU-oriented implementation, and other techniques. Extensive experiments are conducted to compare the performance of the proposed methods with state-of-the-art solvers on a variety of benchmark problems. Finally, we conclude the thesis in Chapter 6 and discuss some possible future research directions.

Chapter 2

Preliminaries

2.1 Notations

1. Let $\mathbb{R}^n$ denote the n-dimensional real Euclidean space, equipped with the standard inner product $\langle x, y\rangle$ for $x, y \in \mathbb{R}^n$. The Euclidean norm induced by this inner product is $\|x\| := \sqrt{\langle x, x\rangle}$ for $x \in \mathbb{R}^n$. The infinity norm of $x \in \mathbb{R}^n$ is $\|x\|_\infty := \max_{1\le i\le n} |x_i|$.

2. For a linear operator $B : \mathbb{R}^n \to \mathbb{R}^m$, we denote its adjoint by B^* (with respect to the Euclidean inner products). For a matrix $A \in \mathbb{R}^{m\times n}$, we denote its transpose by $A^\top$. The spectral norm of A is $\|A\| := \sqrt{\lambda_{\max}(AA^\top)}$, where $\lambda_{\max}(\cdot)$ denotes the largest eigenvalue.

3. We denote the space of symmetric $n \times n$ matrices by $\mathbb{S}^n$. For a matrix variable $X \in \mathbb{R}^{m\times n}$, its vectorization $\mathrm{vec}(X) \in \mathbb{R}^{mn}$ is the column-stacking operator:

$$\mathrm{vec}(X) \;=\; \begin{bmatrix} X_{:,1} \\ X_{:,2} \\ \vdots \\ X_{:,n} \end{bmatrix}, \quad \text{equivalently} \quad \big(\mathrm{vec}(X)\big)_{i+(j-1)m} \;=\; X_{i,j}, \;\; 1 \le i \le m,\; 1 \le j \le n,$$

   where $X_{:,j} \in \mathbb{R}^m$ is the j-th column of X.

4. We denote the Kronecker product of two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ by

$A \otimes B \in \mathbb{R}^{mp \times nq}$:

$$A \otimes B \;=\; \begin{bmatrix} A_{1,1}B & \cdots & A_{1,n}B \\ \vdots & \ddots & \vdots \\ A_{m,1}B & \cdots & A_{m,n}B \end{bmatrix}.$$

5. For a vector $x \in \mathbb{R}^n$, we denote by $D = \operatorname{diag}(x) \in \mathbb{R}^{n \times n}$ the diagonal matrix with the entries of x on its diagonal:

$$D = \begin{bmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_n \end{bmatrix}.$$

6. Given a self-adjoint positive semidefinite linear operator $\mathcal{M} : \mathbb{R}^n \to \mathbb{R}^n$, the $\mathcal{M}$-seminorm of $x \in \mathbb{R}^n$ is $\|x\|_{\mathcal{M}} := \sqrt{\langle x, \mathcal{M}x \rangle}$.

7. Let $f : \mathbb{R}^n \to (-\infty, +\infty]$ be a proper, closed, convex function. Its effective domain is $\operatorname{dom}(f) := \{x \in \mathbb{R}^n \mid f(x) < +\infty\}$. The subdifferential of f at $x \in \operatorname{dom}(f)$ is

$$\partial f(x) := \big\{ g \in \mathbb{R}^n \,\big|\, f(y) \geq f(x) + \langle g, y - x \rangle \text{ for all } y \in \mathbb{R}^n \big\}.$$

   The convex conjugate of f is

$$f^*(z) := \sup_{x \in \mathbb{R}^n} \big\{ \langle x, z \rangle - f(x) \big\}, \quad z \in \mathbb{R}^n.$$

   The proximal mapping of f is

$$\operatorname{Prox}_f(x) := \arg\min_{z \in \mathbb{R}^n} \left\{ f(z) + \tfrac{1}{2}\|z - x\|^2 \right\}, \quad x \in \mathbb{R}^n,$$

   and is single-valued.

8. Let $C \subseteq \mathbb{R}^n$ be a nonempty closed convex set. Its indicator function is

$$\delta_C(x) := \begin{cases} 0, & x \in C, \\ +\infty, & x \notin C. \end{cases}$$

The Euclidean distance from $x \in \mathbb{R}^n$ to C is $\operatorname{dist}(x, C) := \inf_{z \in C} \|z - x\|$. The Euclidean projection onto C is $\Pi_C(x) := \arg\min_{z \in C} \|x - z\|$; for nonempty closed convex set C, $\Pi_C(x)$ is uniquely defined for every $x \in \mathbb{R}^n$.

2.2 Restricted Wolfe dual

The Wolfe dual (Dorn, 1960; Wolfe, 1961) formulation of the CCQP (1.1) is given by

$$\begin{aligned} \min_{y \in \mathbb{R}^m,\, w \in \mathbb{R}^n,\, z \in \mathbb{R}^n} \quad & \frac{1}{2}\langle w, Qw \rangle + \delta^*_{\mathcal{K}}(-y) + \phi^*(-z) \\ \text{s.t.} \quad & -Qw + A^\top y + z = c, \end{aligned} \tag{2.1}$$

where $\delta^*_{\mathcal{K}}$ and ϕ^* is the convex conjugate of the indicator function $\delta_{\mathcal{K}}$ and the function ϕ, respectively. Unfortunately, the solution set of (2.1), if nonempty, is always unbounded as long as Q is not positive definite (the null space of Q is uncontrollable). To address this issue, Li et al. (2018b) proposed the restricted Wolfe dual formulation, which is given by

$$\begin{aligned} \min_{y \in \mathbb{R}^m,\, w \in \mathbb{R}^n,\, z \in \mathbb{R}^n} \quad & \frac{1}{2}\langle w, Qw \rangle + \delta^*_{\mathcal{K}}(-y) + \phi^*(-z) \\ \text{s.t.} \quad & -Qw + A^\top y + z = c, \\ & w \in \mathcal{W}, \end{aligned} \tag{2.2}$$

where $\mathcal{W} := \operatorname{Range}(Q)$, the range space of Q. Notably, Q is self-adjoint and positive definite on $\mathcal{W}$. The equivalence between the (Linear Independence Constraint Qualification) strict Mangasarian–Fromovitz Constraint Qualification of the primal problem and the (strong) second-order sufficient conditions of the restricted Wolfe dual is preserved (Sun, 2020). Moreover, the dual-based approaches, such as the augmented Lagrangian method, can be employed. For instance, Li et al. (2018b) successfully extended the SDPNAL software for solving the dual of semidefinite programming to the restricted Wolfe dual of convex quadratic semidefinite programming.

2.3 Preconditioned alternating direction method of multipliers

A widely used first-order method for solving convex optimization problems is the preconditioned ADMM with semi-proximal terms (Xiao et al., 2018). Let $\mathbb{X}, \mathbb{Y}$, and $\mathbb{Z}$ denote three finite-dimensional real Euclidean spaces, each equipped with the inner product $\langle \cdot, \cdot \rangle$ and the corresponding norm $\| \cdot \|$. Consider the following convex optimization problem:

$$\begin{aligned} \min_{y \in \mathbb{Y},\, z \in \mathbb{Z}} \quad & f_1(y) + f_2(z) \\ \text{s.t.} \quad & B_1 y + B_2 z = c, \end{aligned} \tag{2.3}$$

where $f_1 : \mathbb{Y} \to (-\infty, +\infty]$, $f_2 : \mathbb{Z} \to (-\infty, +\infty]$ are proper closed convex functions, $B_1 : \mathbb{Y} \to \mathbb{X}$, $B_2 : \mathbb{Z} \to \mathbb{X}$ are given linear operators, and $c \in \mathbb{X}$ is a given point.

The augmented Lagrangian function of (2.3) is defined by, for any $(y, z, x) \in \mathbb{Y} \times \mathbb{Z} \times \mathbb{X}$

$$L_\sigma(y, z; x) = f_1(y) + f_2(z) + \langle x, B_1 y + B_2 z - c \rangle + \frac{\sigma}{2} \| B_1 y + B_2 z - c \|^2,$$

where $\sigma > 0$ is the penalty parameter. The dual of problem (2.3) is given by

$$\max_{x \in \mathbb{X}} \quad -f_1^*(-B_1^* x) - f_2^*(-B_2^* x) - \langle c, x \rangle. \tag{2.4}$$

The preconditioned ADMM with semi-proximal terms (Xiao et al., 2018) is given by Algorithm 1.

Algorithm 1 A preconditioned ADMM for solving the convex optimization (2.3)

Input: Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two self-adjoint positive semidefinite linear operators on $\mathbb{Y}$ and $\mathbb{Z}$, respectively. Choose $u^0 = (y^0, z^0, x^0) \in \mathrm{dom}(f_1) \times \mathrm{dom}(f_2) \times \mathbb{X}$. Set parameters $\rho \in (0, 2)$ and $\sigma > 0$. Denote $u = (y, z, x)$ and $\bar{u} = (\bar{y}, \bar{z}, \bar{x})$.

for $k = 0, 1, 2, \ldots$ **do**

Step 1. $\bar{z}^{k+1} = \arg\min_{z \in \mathbb{Z}} \left\{ L_\sigma(y^k, z; x^k) + \frac{1}{2}\|z - z^k\|^2_{\mathcal{S}_2} \right\}$;

Step 2. $\bar{x}^{k+1} = x^k + \sigma(B_1 y^k + B_2 \bar{z}^{k+1} - c)$;

Step 3. $\bar{y}^{k+1} = \arg\min_{y \in \mathbb{Y}} \left\{ L_\sigma(y, \bar{z}^{k+1}; \bar{x}^{k+1}) + \frac{1}{2}\|y - y^k\|^2_{\mathcal{S}_1} \right\}$;

Step 4. $u^{k+1} = \rho \bar{u}^{k+1} + (1 - \rho) u^k$;

end for

To establish the convergence of Algorithm 1, we first introduce some notations. For the closed convex functions f_1 and f_2, there exist two self-adjoint and positive semidefinite operators Σ_{f_1} and Σ_{f_2} such that, for any $y, y' \in \mathrm{dom}(f_1)$, $\phi \in \partial f_1(y)$, and $\phi' \in \partial f_1(y')$ the following hold:

$$f_1(y) \geq f_1(y') + \langle \phi', y - y' \rangle + \frac{1}{2}\|y - y'\|^2_{\Sigma_{f_1}},$$

$$\langle \phi - \phi', y - y' \rangle \geq \|y - y'\|^2_{\Sigma_{f_1}}.$$

Similarly, for any $z, z' \in \mathrm{dom}(f_2)$, $\psi \in \partial f_2(z)$, and $\psi' \in \partial f_2(z')$ the following hold:

$$f_2(z) \geq f_2(z') + \langle \psi', z - z' \rangle + \frac{1}{2}\|z - z'\|^2_{\Sigma_{f_2}},$$

$$\langle \psi - \psi', z - z' \rangle \geq \|z - z'\|^2_{\Sigma_{f_2}}.$$

According to Rockafellar (1970, Corollary 28.3.1), a point $(y^*, z^*) \in \mathbb{Y} \times \mathbb{Z}$ is an optimal solution to problem (2.3) if there exists $x^* \in \mathbb{X}$ such that the following KKT system is satisfied:

$$\begin{cases} 0 \in B_1^* x^* + \partial f_1(y^*), \\ 0 \in B_2^* x^* + \partial f_2(z^*), \\ B_1 y^* + B_2 z^* - c = 0. \end{cases} \tag{2.5}$$

We make the following assumptions:

Assumption 2.1. *There exists a vector* $(y^*, z^*, x^*) \in \mathbb{Y} \times \mathbb{Z} \times \mathbb{X}$ *that satisfies the KKT system* (2.5).

Assumption 2.2. *Both* $\Sigma_{f_1} + B_1^* B_1 + \mathcal{S}_1$ *and* $\Sigma_{f_2} + B_2^* B_2 + \mathcal{S}_2$ *are positive definite.*

The convergence result for Algorithm 1 is presented in Theorem 2.1.

Theorem 2.1 (Theorem 5.1 in Xiao et al. (2018))**.** *Suppose that Assumptions 2.1 and 2.2 hold. Then the sequence* $\{u^k\} = \{(y^k, z^k, x^k)\}$ *generated by the preconditioned ADMM with semi-proximal terms in Algorithm 1 converges to the point* $u^* = (y^*, z^*, x^*)$*, where* (y^*, z^*) *solves the problem* (2.3) *and* x^* *is a solution to problem* (2.4).

We briefly review some complexity results for Algorithm 1 that are relevant to this thesis. For the case $\rho = 1$, Monteiro and Svaiter (2013) established an ergodic $O(1/k)$ convergence rate with respect to the ε-subdifferential residual, while Davis and Yin (2016) showed a nonergodic $o(1/\sqrt{k})$ complexity bound in terms of feasibility violation and objective error. For the majorized ADMM with larger dual stepsizes, Cui et al. (2016) proved a nonergodic $O(1/\sqrt{k})$ convergence rate with respect to the KKT residual.

2.4 Halpern Peaceman–Rachford (HPR) methods

Algorithm 1 admits only an $O(1/\sqrt{k})$ complexity bound in terms of the KKT residual. Sun et al. (2025) proposed a Halpern-type acceleration scheme that attains an $O(1/k)$ rate with respect to both the KKT residual and the objective error. In particular, Algorithm 3.1 of their work, with parameters $\rho = 2$ and $\alpha = 2$, coincides with the HPR method with semi-proximal terms, presented as Algorithm 2. The HPR

method derives its name from applying the Halpern iteration to the Peaceman–Rachford splitting scheme, introduced by Zhang et al. (2022) for solving the Wasserstein barycenter problem. In this thesis, we use the HPR method as the baseline algorithm for solving CCQP problems.

Algorithm 2 An HPR method with semi-proximal terms for solving the convex optimization (2.3)

Input: Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two self-adjoint positive semidefinite linear operators on $\mathbb{Y}$ and $\mathbb{Z}$, respectively. Choose $u^0 = (y^0, z^0, x^0) \in \mathrm{dom}(f_1) \times \mathrm{dom}(f_2) \times \mathbb{X}$. Set parameter $\sigma > 0$. Denote $u = (y, z, x)$ and $\bar{u} = (\bar{y}, \bar{z}, \bar{x})$.

for $k = 0, 1, 2, \ldots$ **do**

Step 1. $\bar{z}^{k+1} = \arg\min_{z \in \mathbb{Z}} \left\{ L_\sigma(y^k, z; x^k) + \frac{1}{2}\|z - z^k\|^2_{\mathcal{S}_2} \right\}$;

Step 2. $\bar{x}^{k+1} = x^k + \sigma(B_1 y^k + B_2 \bar{z}^{k+1} - c)$;

Step 3. $\bar{y}^{k+1} = \arg\min_{y \in \mathbb{Y}} \left\{ L_\sigma(y, \bar{z}^{k+1}; \bar{x}^{k+1}) + \frac{1}{2}\|y - y^k\|^2_{\mathcal{S}_1} \right\}$;

Step 4. $\hat{u}^{k+1} = 2\bar{u}^{k+1} - u^k$;

Step 5. $u^{k+1} = \frac{1}{k+2} u^0 + \frac{k+1}{k+2} \hat{u}^{k+1}$;

end for

Under Assumption 2.2, the subproblems in Steps 1 and 3 of Algorithm 2 have unique solutions (Fazel et al., 2013). Under Assumption 2.1, solving problem (2.3) is equivalent to finding a vector u^* such that $\mathbf{0} \in \mathcal{T} u^*$, where the maximal monotone operator $\mathcal{T}$ is defined as

$$\mathcal{T} u = \begin{pmatrix} \partial f_1(y) + B_1^* x \\ \partial f_2(z) + B_2^* x \\ c - B_1 y - B_2 z \end{pmatrix}, \quad \forall u = (y, z, x) \in \mathbb{Y} \times \mathbb{Z} \times \mathbb{X}.$$

The convergence result for Algorithm 2 is presented in Theorem 2.2.

Theorem 2.2 (Corollary 3.5 in Sun et al. (2025))**.** *Suppose that Assumptions 2.1 and 2.2 hold. Then the sequence* $\{\bar{u}^k\} = \{(\bar{y}^k, \bar{z}^k, \bar{x}^k)\}$ *generated by the HPR method with semi-proximal terms in Algorithm 2 converges to the point* $u^* = (y^*, z^*, x^*)$*, where* (y^*, z^*) *solves the problem* (2.3) *and* x^* *is a solution to problem* (2.4)*.*

Note that in problem (2.3), the functions $f_1(\cdot)$ and $f_2(\cdot)$ often admit a composite decomposition

$$f_1 = g_1 + p_1, \qquad f_2 = g_2 + p_2,$$

where $g_1 : \mathbb{Y} \to (-\infty, +\infty)$ and $g_2 : \mathbb{Z} \to (-\infty, +\infty)$ are continuously differentiable convex functions (e.g., convex quadratic functions), and $p_1 : \mathbb{Y} \to (-\infty, +\infty]$, $p_2 : \mathbb{Z} \to (-\infty, +\infty]$ are proper closed convex functions. In this setting, Han et al. (2018) introduced the residual mapping associated with the KKT system (2.5), defined as

$$\mathcal{R}(w) := \begin{pmatrix} y - \mathrm{Prox}_{p_1}\left(y - \nabla g_1(y) - B_1^* x\right) \\ z - \mathrm{Prox}_{p_2}\left(z - \nabla g_2(z) - B_2^* x\right) \\ c - B_1 y - B_2 z \end{pmatrix} \quad \forall w = (y, z, x) \in \mathbb{Y} \times \mathbb{Z} \times \mathbb{X}.$$

To measure complexity in terms of the objective, for the sequence $\{(\bar{y}^k, \bar{z}^k)\}$ generated by Algorithm 2, we define the objective error as

$$h(\bar{y}^{k+1}, \bar{z}^{k+1}) := f_1(\bar{y}^{k+1}) + f_2(\bar{z}^{k+1}) - f_1(y^*) - f_2(z^*), \quad \forall\, k \geq 0,$$

where (y^*, z^*) is an optimal solution to problem (2.3). Moreover, we define the linear operator $\mathcal{M} : \mathcal{U} \to \mathcal{U}$ as

$$\mathcal{M} = \begin{bmatrix} \sigma B_1^* B_1 + \mathcal{S}_1 & 0 & B_1^* \\ 0 & \mathcal{S}_2 & 0 \\ B_1 & 0 & \frac{1}{\sigma} I \end{bmatrix}.$$

Based on Proposition 2.9, Theorem 3.7, and Remark 3.8 in Sun et al. (2025), we obtain the following iteration complexity result for Algorithm 2.

Theorem 2.3. *Suppose that Assumptions 2.1 and 2.2 hold. Let* $\{\bar{u}^k\} = \{(\bar{y}^k, \bar{z}^k, \bar{x}^k)\}$ *and* $\{u^k\} = \{(y^k, z^k, x^k)\}$ *be the sequences generated by the HPR method with semi-proximal terms in Algorithm 2, and let* $u^* = (y^*, z^*, x^*)$ *be a solution to the KKT system* (2.5). *Define* $R_0 := \|u^0 - u^*\|_{\mathcal{M}}$. *Then the following complexity bounds hold*

for all $k \geq 0$*:*

$$\|\bar{u}^{k+1} - u^k\|_{\mathcal{M}} \leq \frac{R_0}{k+1},$$

$$\|\mathcal{R}(\bar{u}^{k+1})\| \leq \left(\frac{\sigma\|B_1^*\| + 1}{\sqrt{\sigma}} + \|\sqrt{\mathcal{S}_1}\| + \|\sqrt{\mathcal{S}_2}\| \right) \frac{R_0}{k+1},$$

$$-\frac{1}{\sqrt{\sigma}}\|x^*\| \cdot \frac{R_0}{k+1} \;\leq\; h(\bar{y}^{k+1}, \bar{z}^{k+1}) \;\leq\; \left(3R_0 + \frac{1}{\sqrt{\sigma}}\|x^*\| \right) \frac{R_0}{k+1}.$$

2.5 The block symmetric Gauss-Seidel decomposition theorem

In this subsection, we recall the symmetric Gauss–Seidel (sGS) technique introduced in Li et al. (2016, 2019) for handling multi-block variables that arise in the restricted Wolfe dual of CCQP (2.1). Let $\mathcal{X}_i = \mathbb{R}^{n_i}$ for $i = 1, \ldots, s$ with integer $s \geq 2$, and define the product space $\mathcal{X} := \mathcal{X}_1 \times \cdots \times \mathcal{X}_s$. Consider the block-structured linear operator Q of the form

$$Q = \begin{bmatrix} Q_{1,1} & \cdots & Q_{1,s} \\ \vdots & \ddots & \vdots \\ Q_{1,s}^* & \cdots & Q_{s,s} \end{bmatrix}, \quad Q_{i,j} \in \mathbb{R}^{n_i \times n_j},\ i, j = 1, \ldots, s.$$

We make the following assumption on Q.

Assumption 2.3. *The operator Q is symmetric positive semidefinite, and each diagonal block $Q_{i,i}$ is symmetric positive definite for $i = 1, \ldots, s$.*

We further decompose Q into the form

$$Q = U + D + U^*,$$

where

$$U = \begin{bmatrix} 0 & Q_{1,2} & \cdots & Q_{1,s} \\ & \ddots & \ddots & \vdots \\ & & \ddots & Q_{s-1,s} \\ & & & 0 \end{bmatrix}, \qquad D = \begin{bmatrix} Q_{1,1} & 0 & \cdots & 0 \\ 0 & Q_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Q_{s,s} \end{bmatrix}.$$

Consider the proximal minimization problem

$$\min_{x\in\mathcal{X}} \; p(x_1) + \tfrac{1}{2}\langle x, Qx\rangle - \langle b, x\rangle + \tfrac{1}{2}\|x - x^k\|^2_{\mathcal{T}_Q}, \tag{2.6}$$

where $p : \mathcal{X}_1 \to (-\infty, +\infty]$ is a proper closed convex function, and the sGS operator $\mathcal{T}_Q := UD^{-1}U^*$. The sGS decomposition theorem corresponding to (2.6) is stated in Theorem 2.4.

Theorem 2.4 (sGS decomposition (Li et al., 2019, Theorem 1))**.** *Assume that Assumption 2.3 holds. Then the following holds*

$$\widehat{Q} \;:=\; Q + T_Q \;=\; (D+U)D^{-1}(D+U^*) \;\succ\; 0. \tag{2.7}$$

Denote $q(x) = \frac{1}{2}\langle x, Qx\rangle - \langle b, x\rangle$. For $i = s, \ldots, 2$, suppose we have computed $x'_i \in \mathcal{X}_i$ defined by

$$\begin{aligned} x'_i &:= \arg\min_{x_i\in\mathcal{X}_i} \; p(\bar{x}_1) + q(\bar{x}_{\le i-1};\, x_i;\, x'_{\ge i+1}) \\ &= Q_{i,i}^{-1}\left(b_i - \sum_{j=1}^{i-1} Q^*_{j,i}\bar{x}_j - \sum_{j=i+1}^{s} Q_{i,j}x'_j \right). \end{aligned}$$

Then the optimal solution x^+ for (2.6) can be computed exactly via

$$\begin{cases} x_1^+ = \arg\min_{x_1\in\mathcal{X}_1} \; p(x_1) + q(x_1;\, x'_{\ge 2}), \\ x_i^+ = \arg\min_{x_i\in\mathcal{X}_i} \; p(x_1^+) + q(x^+_{\le i-1};\, x_i;\, x'_{\ge i+1}), \quad i = 2, \ldots, s, \\ \qquad = Q_{i,i}^{-1}\left(b_i - \sum_{j=1}^{i-1} Q^*_{j,i}x_j^+ - \sum_{j=i+1}^{s} Q_{i,j}x'_j \right), \quad i = 2, \ldots, s. \end{cases}$$

Chapter 3

The primal HPR method for convex composite quadratic programming

In this chapter, we propose two HPR methods for solving different primal formulations of CCQP. Both enjoy an $O(1/k)$ complexity bound in terms of the KKT residual and objective error. By introducing auxiliary slack variables and appropriate proximal operators, each iteration admits an explicit closed-form update, avoiding linear system solves and making the methods well-suited for GPU acceleration.

3.1 Formulation I

To apply the HPR method to the primal CCQP (1.1), we first rewrite the problem with a slack variable $s \in \mathbb{R}^m$ as follows:

$$\begin{aligned} \min_{x\in\mathbb{R}^n,\, s\in\mathbb{R}^m} \quad & \frac{1}{2}\langle x, Qx\rangle + \langle c, x\rangle + \phi(x) + \delta_{\mathcal{K}}(s) \\ \text{s.t.} \quad & Ax = s. \end{aligned} \tag{3.1}$$

The augmented Lagrangian function of (3.1) is given by

$$L_\sigma^{P1}(x, s; y) = \frac{1}{2}\langle x, Qx\rangle + \langle c, x\rangle + \phi(x) + \delta_{\mathcal{K}}(s) + \langle y, s - Ax\rangle + \frac{\sigma}{2}\|Ax - s\|^2,$$

where y is the Lagrange multiplier, and $\sigma > 0$ is the penalty parameter. For convenience, we denote the tuple (x, s, y) by u, and define the space $\mathcal{U} := \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m$ in this section. The HPR method with semi-proximal terms for problem (3.1) is given by Algorithm 3:

Algorithm 3 An HPR method for solving the primal formulation (3.1)

1: **Input:** Choose $u^0 = (x^0, s^0, y^0) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m$. Set parameter $\sigma > 0$. Let $\mathcal{S}_x + Q + \sigma A^\top A \succ 0$. Denote $u = (x, s, y)$ and $\bar{u} = (\bar{x}, \bar{s}, \bar{y})$.
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: **Step 1.** $\bar{s}^{k+1} = \arg\min_{s \in \mathbb{R}^m} \{L_\sigma^{P1}(x^k, s; y^k)\}$;
4: **Step 2.** $\bar{y}^{k+1} = y^k + \sigma(\bar{s}^{k+1} - Ax^k)$;
5: **Step 3.** $\bar{x}^{k+1} = \arg\min_{x \in \mathbb{R}^n} \left\{ L_\sigma^{P1}(x, \bar{s}^{k+1}; \bar{y}^{k+1}) + \frac{1}{2}\|x - x^k\|_{\mathcal{S}_x}^2 \right\}$;
6: **Step 4.** $\hat{u}^{k+1} = 2\bar{u}^{k+1} - u^k$;
7: **Step 5.** $u^{k+1} = \frac{1}{k+2} u^0 + \frac{k+1}{k+2} \hat{u}^{k+1}$;
8: **end for**

3.1.1 Convergence and complexity results

We will establish the convergence and $O(1/k)$ complexity results with respect to KKT residual and objective error for Algorithm 3. According to Rockafellar (1970, Corollary 28.3.1), $(x^*, s^*) \in \mathbb{R}^n \times \mathbb{R}^m$ is an optimal solution to problem (3.1) if there exists $y^* \in \mathbb{R}^m$ such that the KKT system below is satisfied:

$$\begin{cases} 0 \in Qx^* + c - A^\top y^* + \partial\phi(x^*), \\ 0 \in \partial\delta_{\mathcal{K}}(s^*) + y^*, \\ Ax^* - s^* = 0. \end{cases} \tag{3.2}$$

We make the following assumption:

Assumption 3.1. *There exists a vector $(x^*, s^*, y^*) \in \mathcal{U}$ satisfying the KKT system (3.2).*

Under Assumption 3.1, solving problem (3.1) is equivalent to finding a vector $u^* \in \mathcal{U}$ such that $\mathbf{0} \in \mathcal{T}u^*$, where the maximal monotone operator $\mathcal{T}$ is defined by

$$\mathcal{T}u = \begin{pmatrix} \partial\phi(x) + Qx + c - A^\top y \\ \partial\delta_{\mathcal{K}}(s) + y \\ Ax - s \end{pmatrix} \quad \forall u = (x, s, y) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m.$$

We define the linear operator $\mathcal{M} : \mathcal{U} \to \mathcal{U}$ as

$$\mathcal{M} = \begin{bmatrix} \sigma A^\top A + \mathcal{S}_x & 0 & -A^\top \\ 0 & 0 & 0 \\ -A & 0 & \frac{1}{\sigma} I_m \end{bmatrix}. \tag{3.3}$$

Based on Theorem 2.2, the convergence result for Algorithm 3 is established as follows.

Theorem 3.1. *Suppose that Assumption 3.1 holds. Then the sequence* $\{\bar{u}^k\} = \{(\bar{x}^k, \bar{s}^k, \bar{y}^k)\}$ *generated by the HPR method with semi-proximal terms in Algorithm 3 converges to the point* $u^* = (x^*, s^*, y^*)$*, where* (x^*, s^*) *solves the primal problem* (3.1).

To establish the complexity results of Algorithm 3, we consider the residual mapping associated with the KKT system (3.2), introduced in Han et al. (2018):

$$\mathcal{R}(u) = \begin{pmatrix} x - \mathrm{Prox}_{\phi}(x - Qx - c + A^\top y) \\ s - \Pi_{\mathcal{K}}(s - y) \\ Ax - s \end{pmatrix} \quad \forall\, u = (x, s, y) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m.$$

For the sequence $\{(\bar{x}^k, \bar{s}^k\}$ generated by Algorithm 3, we define the objective error as

$$\begin{aligned} h(\bar{x}^k, \bar{s}^k) := \frac{1}{2}\langle \bar{x}^k, Q\bar{x}^k \rangle + \langle c, \bar{x}^k \rangle + \phi(\bar{x}^k) + \delta_{\mathcal{K}}(\bar{s}^k) \\ - \left(\frac{1}{2}\langle x^*, Qx^* \rangle + \langle c, x^* \rangle + \phi(x^*) + \delta_{\mathcal{K}}(s^*) \right) \quad \forall\, k \geq 1, \end{aligned}$$

where (x^*, s^*) is a solution to the primal problem (3.1). Based on the iteration complexity results in Theorem 2.3, we obtain the following iteration complexity bounds for the HPR method with semi-proximal terms for solving the primal problem (3.1).

Theorem 3.2. *Suppose that Assumption 3.1 holds. Let* $\{\bar{u}^k\} = \{(\bar{x}^k, \bar{s}^k, \bar{y}^k)\}$ *and* $\{u^k\} = \{(x^k, s^k, y^k)\}$ *be the sequences generated by the HPR method with semi-proximal terms in Algorithm 3, and let* $u^* = (x^*, s^*, y^*)$ *be a solution to the KKT system* (3.2). *Define* $R_0 := \|u^0 - u^*\|_{\mathcal{M}}$, *where* $\mathcal{M}$ *is defined in* (3.3). *Then the following complexity bounds hold for all* $k \geq 0$*:*

$$\|\bar{u}^{k+1} - u^k\|_{\mathcal{M}} \leq \frac{R_0}{k+1}, \quad \|\mathcal{R}(\bar{u}^{k+1})\| \leq \left(\frac{\sigma\|A^\top\| + 1}{\sqrt{\sigma}} + \|\sqrt{\mathcal{S}_x}\|\right)\frac{R_0}{k+1},$$

$$-\frac{1}{\sqrt{\sigma}}\|y^*\| \cdot \frac{R_0}{k+1} \;\leq\; h(\bar{x}^{k+1}, \bar{s}^{k+1}) \;\leq\; \left(3R_0 + \frac{1}{\sqrt{\sigma}}\|y^*\|\right)\frac{R_0}{k+1}.$$

3.1.2 Implementation details

Now we introduce the step-by-step implementation of Algorithm 3. In general, Algorithm 3 is not practically implementable for arbitrary choices of $\mathcal{S}_x$. For instance, if $\mathcal{S}_x = 0$, Step 3 cannot be solved efficiently when $\phi(x) \neq 0$. A practical and implementable choice is to set

$$\mathcal{S}_x := \lambda_Q I_n - Q + \sigma(\lambda_A I_n - A^\top A),$$

for some $\lambda_Q, \lambda_A > 0$ ensuring that $\mathcal{S}_x$ is positive semidefinite. With this linearization choice of $\mathcal{S}_x$, the update formula can be explicitly derived as follows.

In **Step 1**, the optimality condition of the subproblem is given by

$$0 \in \partial\delta_{\mathcal{K}}(\bar{s}^{k+1}) + y^k + \sigma(\bar{s}^{k+1} - Ax^k),$$

which is equivalent to

$$\bar{s}^{k+1} = \Pi_{\mathcal{K}}(Ax^k - \sigma^{-1}y^k).$$

For **Step 2**, the formula of the update has the following equivalence

$$\bar{y}^{k+1} = y^k + \sigma(\bar{s}^{k+1} - Ax^k) = \sigma\left(\Pi_K(Ax^k - \sigma^{-1}y^k) - (Ax^k - \sigma^{-1}y^k)\right).$$

In **Step 3**, the optimality condition of the subproblem is given by

$$0 \in \partial\phi(\bar{x}^{k+1}) + Q\bar{x}^{k+1} + c - A^\top\bar{y}^{k+1} + \sigma A^\top(A\bar{x}^{k+1} - \bar{s}^{k+1}) + \mathcal{S}_x(\bar{x}^{k+1} - x^k).$$

By substituting $\mathcal{S}_x = \lambda_Q I_n - Q + \sigma(\lambda_A I_n - A^\top A)$ into the above condition, we have

$$\begin{aligned}
\bar{x}^{k+1} &= \mathrm{Prox}_\phi\left(x^k + \frac{1}{\lambda_Q + \sigma\lambda_A}(-Qx^k - c + A^\top\bar{y}^{k+1} - \sigma A^\top(Ax^k - \bar{s}^{k+1}))\right) \\
&= \mathrm{Prox}_\phi\left(x^k + \frac{1}{\lambda_Q + \sigma\lambda_A}(-Qx^k - c + A^\top(2\bar{y}^{k+1} - y^k))\right) \\
&= \mathrm{Prox}_\phi\left(x^k + \frac{1}{\lambda_Q + \sigma\lambda_A}(-Qx^k - c + A^\top\hat{y}^{k+1})\right).
\end{aligned}$$

Remark 3.1. *The update of $\hat{y}^{k+1}$, formally given in Step 4 of Algorithm 3, can in fact be computed immediately after Step 2. Moreover, the quantity $\bar{s}^{k+1}$ need not be explicitly evaluated at every iteration, except when it is required for computing residuals used in the stopping criteria.*

The corresponding proximal operator $\mathcal{S}_x = \lambda_Q I_n - Q + \sigma(\lambda_A I_n - A^\top A)$ consists of both Q, A, and a penalty parameter σ which may be extremely small or large in practice. Practically, the proximal operator $\mathcal{S}_x$ with a large spectral norm $\|\mathcal{S}_x\|$ can significantly slow down the convergence rate.

3.2 Formulation II

In this section, we show another primal formulation where the matrices Q and A in proximal terms are separated. We rewrite the problem with two slack variables

$s \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$ as follows:

$$\begin{aligned} \min_{x\in\mathbb{R}^n,\, s\in\mathbb{R}^m,\, v\in\mathbb{R}^n} \quad & \frac{1}{2}\langle v, Qv\rangle + \langle c, x\rangle + \phi(x) + \delta_{\mathcal{K}}(s) \\ \text{s.t.} \quad & Ax = s, \\ & x = v. \end{aligned} \tag{3.4}$$

The augmented Lagrangian function of (3.4) is given by

$$\begin{aligned} L_\sigma^{P2}(x, s, v; y_1, y_2) = & \frac{1}{2}\langle v, Qv\rangle + \langle c, x\rangle + \phi(x) + \delta_{\mathcal{K}}(s) \\ & + \langle y_1, s - Ax\rangle + \frac{\sigma}{2}\|Ax - s\|^2 \\ & + \langle y_2, v - x\rangle + \frac{\sigma}{2}\|x - v\|^2, \end{aligned}$$

where y_1 and y_2 are the Lagrange multipliers, and $\sigma > 0$ is the penalty parameter. For convenience, we denote the tuple (x, s, v, y_1, y_2) by u, and define the space $\mathcal{U} := \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n$ in this section. The HPR with semi-proximal terms for solving problem (3.4) is given by Algorithm 4.

Algorithm 4 An HPR for solving the primal reformulation (3.4)

Input: Let $\mathcal{S}_x$ and $\mathcal{S}_v$ be two self-adjoint positive semidefinite operators on $\mathbb{R}^n$ such that $\mathcal{S}_x + \sigma A^\top A$ and $\mathcal{S}_v + Q$ are positive definite. Choose $u^0 = (x^0, s^0, v^0, y_1^0, y_2^0) \in \mathcal{U}$. Set parameter $\sigma > 0$. Denote $u = (x, s, v, y_1, y_2)$ and $\bar{u} = (\bar{x}, \bar{s}, \bar{v}, \bar{y}_1, \bar{y}_2)$.

for $k = 0, 1, 2, \ldots$ **do**

Step 1. $(\bar{s}^{k+1}, \bar{v}^{k+1}) = \underset{(s,v)\in\mathbb{R}^m\times\mathbb{R}^n}{\arg\min} \left\{ L_\sigma^{P2}(x^k, s, v; y_1^k, y_2^k) + \frac{1}{2}\|v - v^k\|_{\mathcal{S}_v}^2 \right\}$;

Step 2-1. $\bar{y}_1^{k+1} = y_1^k + \sigma(\bar{s}^{k+1} - Ax^k)$;

Step 2-2. $\bar{y}_2^{k+1} = y_2^k + \sigma(\bar{v}^{k+1} - x^k)$;

Step 3. $\bar{x}^{k+1} = \underset{x\in\mathbb{R}^n}{\arg\min} \left\{ L_\sigma^{P2}(x, \bar{s}^{k+1}, \bar{v}^{k+1}; \bar{y}_1^{k+1}, \bar{y}_2^{k+1}) + \frac{1}{2}\|x - x^k\|_{\mathcal{S}_x}^2 \right\}$;

Step 4. $\hat{u}^{k+1} = 2\bar{u}^{k+1} - u^k$;

Step 5. $u^{k+1} = \frac{1}{k+2}u^0 + \frac{k+1}{k+2}\hat{u}^{k+1}$;

end for

Remark 3.2. *We formulate the update in Step 1 as a joint minimization over* (s, v). *Since* s *and* v *are not coupled, it reduces to two independent minimization problems.*

3.2.1 Convergence and complexity results

Now we establish the convergence and complexity results for Algorithm 4. According to Rockafellar (1970, Corollary 28.3.1), $(x^*, s^*, v^*) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n$ is an optimal solution to problem (3.4) if there exist $y_1^* \in \mathbb{R}^m$ and $y_2^* \in \mathbb{R}^n$ such that the following KKT system is satisfied:

$$\begin{cases} 0 \in \partial\phi(x^*) + c - A^\top y_1^* - y_2^*, \\ 0 \in \partial\delta_{\mathcal{K}}(s^*) + y_1^*, \\ Qv^* + y_2^* = 0, \\ Ax^* - s^* = 0, \\ x^* - v^* = 0. \end{cases} \tag{3.5}$$

We make the following assumption:

Assumption 3.2. *There exists a vector* $(x^*, s^*, v^*, y_1^*, y_2^*) \in \mathcal{U}$ *satisfying the KKT system* (3.5).

Under Assumption 3.2, solving problem (3.4) is equivalent to finding a vector $u^* \in \mathcal{U}$ such that $\mathbf{0} \in \mathcal{T}u^*$, where the maximal monotone operator $\mathcal{T}$ is defined by

$$\mathcal{T}u = \begin{pmatrix} \partial\phi(x) + c - A^\top y_1 - y_2 \\ \partial\delta_{\mathcal{K}}(s) + y_1 \\ Qv + y_2 \\ Ax - s \\ x - v \end{pmatrix} \quad \forall u = (x, s, v, y_1, y_2) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n.$$

We define the linear operator $\mathcal{M} : \mathcal{U} \to \mathcal{U}$ as

$$\mathcal{M} = \begin{bmatrix} \sigma A^\top A + \mathcal{S}_x & 0 & 0 & -A^\top & -I_n \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathcal{S}_v & 0 & 0 \\ -A & 0 & 0 & \frac{1}{\sigma} I_m & 0 \\ -I_n & 0 & 0 & 0 & \frac{1}{\sigma} I_n \end{bmatrix}. \tag{3.6}$$

Based on Theorem 2.2, we establish the following convergence result for Algorithm 4.

Theorem 3.3. *Suppose that Assumption 3.2 holds. Then the sequence* $\{\bar{u}^k\} = \{(\bar{x}^k, \bar{s}^k, \bar{v}^k, \bar{y}_1^k, \bar{y}_2^k)\}$ *generated by the HPR method with semi-proximal terms in Algorithm 4 converges to the point* $u^* = (x^*, s^*, v^*, y_1^*, y_2^*)$*, where* (x^*, s^*, v^*) *solves the primal problem* (3.4)*.*

The residual mapping associated with the KKT system (3.5) was introduced by Han et al. (2018), as follows:

$$\mathcal{R}(u) = \begin{pmatrix} x - \mathrm{Prox}_\phi(x - c + A^\top y_1 + y_2) \\ s - \Pi_{\mathcal{K}}(s - y_1) \\ Qv + y_2 \\ Ax - s \\ x - v \end{pmatrix} \quad \forall\, u = (x, s, v, y_1, y_2) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n.$$

For the sequence $\{(\bar{x}^k, \bar{s}^k, \bar{v}^k\}$ generated by Algorithm 4, we define the objective error as

$$\begin{aligned} h(\bar{x}^k, \bar{s}^k, \bar{v}^k) := & \frac{1}{2}\langle \bar{v}^k, Q\bar{v}^k \rangle + \langle c, \bar{x}^k \rangle + \phi(\bar{x}^k) + \delta_{\mathcal{K}}(\bar{s}^k) \\ & - \left(\frac{1}{2}\langle v^*, Qv^* \rangle + \langle c, x^* \rangle + \phi(x^*) + \delta_{\mathcal{K}}(s^*) \right) \quad \forall\, k \geq 1, \end{aligned}$$

where (x^*, s^*, v^*) is a solution to the primal problem (3.4). Based on the iteration complexity results in Theorem 2.3, we obtain the following iteration complexity bounds for the HPR method with semi-proximal terms for solving the primal problem (3.4).

Theorem 3.4. *Suppose that Assumption 3.2 holds. Let* $\{\bar{u}^k\} = \{(\bar{x}^k, \bar{s}^k, \bar{v}^k, \bar{y}_1^k, \bar{y}_2^k)\}$ *and* $\{u^k\} = \{(x^k, s^k, v^k, y_1^k, y_2^k)\}$ *be the sequences generated by the HPR method with semi-proximal terms in Algorithm 4, and let* $u^* = (x^*, s^*, v^*, y_1^*, y_2^*)$ *be a solution to the KKT system* (3.5)*. Define* $R_0 := \|u^0 - u^*\|_{\mathcal{M}}$*, where* $\mathcal{M}$ *is defined in* (3.6)*. Then the following complexity bounds hold for all* $k \geq 0$*:*

$$\|\bar{u}^{k+1} - u^k\|_{\mathcal{M}} \leq \frac{R_0}{k+1}, \quad \|\mathcal{R}(\bar{u}^{k+1})\| \leq \left(\frac{\sigma\|\tilde{A}^\top\| + 1}{\sqrt{\sigma}} + \|\sqrt{\mathcal{S}_x}\| + \|\sqrt{\mathcal{S}_v}\|\right)\frac{R_0}{k+1},$$

$$-\frac{1}{\sqrt{\sigma}}\|(y_1^*, y_2^*)\| \cdot \frac{R_0}{k+1} \;\leq\; h(\bar{x}^{k+1}, \bar{s}^{k+1}, \bar{v}^{k+1}) \;\leq\; \left(3R_0 + \frac{1}{\sqrt{\sigma}}\|(y_1^*, y_2^*)\|\right)\frac{R_0}{k+1},$$

where $\tilde{A} = \begin{bmatrix} -A \\ -I_n \end{bmatrix}$.

3.2.2 Implementation details

Algorithm 4 is not practically implementable for all choices of $\mathcal{S}_x$. For example, if $\mathcal{S}_x = 0$, Step 3 cannot be solved efficiently when $\phi(x) \neq 0$. We therefore adopt the linearizations

$$\mathcal{S}_x := \sigma(\lambda_A I_n - A^\top A), \qquad \mathcal{S}_v := \lambda_Q I_n - Q,$$

with parameters $\lambda_A \geq \lambda_{\max}(A^\top A)$ and $\lambda_Q \geq \lambda_{\max}(Q)$, so that $\mathcal{S}_x, \mathcal{S}_v \succeq 0$. The corresponding update formulas are derived as follows.

In **Step 1**, the optimality conditions of the subproblems with respect to s and v are given by

$$0 \in \partial\delta_{\mathcal{K}}(\bar{s}^{k+1}) + y_1^k + \sigma(\bar{s}^{k+1} - Ax^k),$$

$$Q\bar{v}^{k+1} + y_2^k + \sigma(\bar{v}^{k+1} - x^k) + \mathcal{S}_v(\bar{v}^{k+1} - v^k) = 0,$$

which are equivalent to

$$\bar{s}^{k+1} = \Pi_{\mathcal{K}}(Ax^k - \sigma^{-1}y^k),$$

$$\bar{v}^{k+1} = v^k - \frac{1}{\sigma + \lambda_Q}(Qv^k + y_2^k + \sigma(v^k - x^k)).$$

For **Step 2-1**, the formula of update has the following equivalence

$$\bar{y}_1^{k+1} = y_1^k + \sigma(\bar{s}^{k+1} - Ax^k) = \sigma\left(\Pi_K(Ax^k - \sigma^{-1}y_1^k) - (Ax^k - \sigma^{-1}y_1^k)\right).$$

For **Step 2-2**, the formula of update is

$$\bar{y}_2^{k+1} = y_2^k + \sigma(\bar{v}^{k+1} - x^k).$$

In **Step 3**, the optimality condition of the subproblem is given by

$$0 \in \partial\phi(\bar{x}^{k+1}) + c - A^\top \bar{y}_1^{k+1} - \bar{y}_2^{k+1} + \sigma A^\top(A\bar{x}^{k+1} - \bar{s}^{k+1}) + \sigma(\bar{x}^{k+1} - \bar{v}^{k+1}) + \mathcal{S}_x(\bar{x}^{k+1} - x^k).$$

By substituting $\mathcal{S}_x = \sigma(\lambda_A I_n - A^\top A)$ into the above condition, we obtain

$$\begin{aligned}
\bar{x}^{k+1} &= \mathrm{Prox}_\phi\left(x^k + \frac{1}{\sigma + \sigma\lambda_A}(-c + A^\top \bar{y}_1^{k+1} + \bar{y}_2^{k+1} - \sigma A^\top(Ax^k - \bar{s}^{k+1}) - \sigma(x^k - \bar{v}^{k+1}))\right) \\
&= \mathrm{Prox}_\phi\left(x^k + \frac{1}{\sigma + \sigma\lambda_A}(-c + A^\top(2\bar{y}_1^{k+1} - y_1^k) + 2\bar{y}_2^{k+1} - y_2^k)\right) \\
&= \mathrm{Prox}_\phi\left(x^k + \frac{1}{\sigma + \sigma\lambda_A}(-c + A^\top \hat{y}_1^{k+1} + \hat{y}_2^{k+1})\right).
\end{aligned}$$

Remark 3.3. *The updates of $\hat{y}_1^{k+1}$ and $\hat{y}_2^{k+1}$, although written in Step 4 of Algorithm 4, can in fact be computed immediately after Step 2. Moreover, the slack variable $\bar{s}^{k+1}$ need not be explicitly evaluated at every iteration, except when it is required for computing residuals for checking the stopping criteria.*

Chapter 4

A dual HPR method for solving convex composite quadratic programming

In this chapter, we introduce a dual HPR method designed for solving the restricted Wolfe dual of CCQP problems, which also enjoys a fast $O(1/k)$ complexity bound in terms of the KKT residual and the objective error. One distinctive feature of the dual approach is that, instead of working with the primal formulations, it builds on the novel restricted Wolfe dual introduced in recent years. It also leverages the symmetric Gauss–Seidel technique to simplify subproblem updates without introducing auxiliary slack variables that typically lead to slow convergence. By restricting updates to the range space of the Hessian of the quadratic objective function, the dual approach employs proximal operators of smaller spectral norms to speed up the convergence. Shadow sequences are elaborately constructed to deal with the range space constraints.

4.1 A dual HPR method incorporating the symmetric Gauss-Seidel technique

Recall the following restricted Wolfe dual formulation of CCQP,

$$\begin{aligned} \min_{y\in\mathbb{R}^m,\, w\in\mathbb{R}^n,\, z\in\mathbb{R}^n} \quad & \frac{1}{2}\langle w, Qw\rangle + \delta^*_{\mathcal{K}}(-y) + \phi^*(-z) \\ \text{s.t.} \quad & -Qw + A^\top y + z = c, \\ & w \in \mathcal{W}. \end{aligned} \tag{4.1}$$

Since Q is positive definite on $\mathcal{W} = \text{Range}(Q)$, one may consider applying a convergent three-block semi-proximal ADMM (Li et al., 2015) to solve problem (4.1). However, its convergence guarantee requires the penalty parameter σ proportional to the smallest positive eigenvalue of Q, which may lead to slow convergence rate in practice.

In this chapter, we introduce an HPR method incorporating the symmetric Gauss-Seidel technique for solving problem (4.1). The augmented Lagrangian function, for any $(y, w, z, x) \in \mathbb{R}^m \times \mathcal{W} \times \mathbb{R}^n \times \mathbb{R}^n$, is given by

$$\begin{aligned} L_\sigma(y, w, z; x) = & \frac{1}{2}\langle w, Qw\rangle + \delta^*_{\mathcal{K}}(-y) + \phi^*(-z) \\ & + \langle x, -Qw + A^\top y + z - c\rangle \\ & + \frac{\sigma}{2}\| - Qw + A^\top y + z - c\|^2. \end{aligned}$$

For convenience, we denote the tuple (y, w, z, x) by u, and define the space $\mathcal{U} := \mathbb{R}^m \times \mathcal{W} \times \mathbb{R}^n \times \mathbb{R}^n$ in this chapter. The HPR with semi-proximal terms for solving problem (4.1) is given by Algorithm 5.

Algorithm 5 An HPR for solving the restricted Wolfe dual problem (4.1)

Input: Choose a self-adjoint positive semidefinite linear operator $\mathcal{S}_{yw}$ on $\mathbb{R}^m \times \mathcal{W}$. Set parameter $\sigma > 0$. Denote $u = (y, w, z, x)$ and $\bar{u} = (\bar{y}, \bar{w}, \bar{z}, \bar{x})$. Let $u^0 = (y^0, w^0, z^0, x^0) \in \mathbb{R}^m \times \mathcal{W} \times \mathbb{R}^n \times \mathbb{R}^n$.

for $k = 0, 1, 2, \ldots$ **do**

Step 1. $\bar{z}^{k+1} = \arg\min_{z \in \mathbb{R}^n} \{L_\sigma(y^k, w^k, z; x^k)\}$;

Step 2. $\bar{x}^{k+1} = x^k + \sigma(-Qw^k + A^\top y^k + \bar{z}^{k+1} - c)$;

Step 3. $(\bar{y}^{k+1}, \bar{w}^{k+1}) =$

$$\arg\min_{(y,w) \in \mathbb{R}^m \times \mathcal{W}} \left\{ L_\sigma(y, w, \bar{z}^{k+1}; \bar{x}^{k+1}) + \frac{1}{2}\|(y, w) - (y^k, w^k)\|^2_{\mathcal{S}_{yw}} \right\};$$

Step 4. $\hat{u}^{k+1} = 2\bar{u}^{k+1} - u^k$;

Step 5. $u^{k+1} = \frac{1}{k+2} u^0 + \frac{k+1}{k+2} \hat{u}^{k+1}$;

end for

Define the linear operator $\mathcal{H} : \mathbb{R}^m \times \mathcal{W} \to \mathbb{R}^m \times \mathcal{W}$ as follows:

$$\mathcal{H} := \sigma A_Q^\top A_Q + \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & Q \end{pmatrix} = \begin{pmatrix} \sigma A A^\top & -\sigma A Q \\ -\sigma Q A^\top & \sigma Q^2 + Q \end{pmatrix},$$

where $A_Q := [A^\top \ -Q] \in \mathbb{R}^{n \times (m+n)}$. According to Rockafellar (1970, Corollary 28.3.1), $(y^*, w^*, z^*) \in \mathbb{R}^m \times \mathcal{W} \times \mathbb{R}^n$ is an optimal solution to problem (2.2) if there exists $x^* \in \mathbb{R}^n$ such that the KKT system below is satisfied:

$$\begin{cases} 0 \in \partial \delta^*_{\mathcal{K}}(-y^*) - Ax^*, \\ Qw^* - Qx^* = 0, \\ 0 \in \partial \phi^*(-z^*) - x^*, \\ -Qw^* + A^\top y^* + z^* - c = 0. \end{cases} \tag{4.2}$$

We make the following assumptions:

Assumption 4.1. *There exists a vector $(y^*, w^*, z^*, x^*) \in \mathcal{U}$ satisfying the KKT system* (4.2).

Assumption 4.2. *The operator $\mathcal{S}_{yw}$ is a self-adjoint positive semidefinite linear operator such that $\mathcal{S}_{yw} + \mathcal{H}$ is positive definite on $\mathbb{R}^m \times \mathcal{W}$.*

To simplify the subproblem in Step 3 of Algorithm 5, we use the proximal operator $\mathcal{S}_{yw}$ based on the sGS decomposition technique (Li et al., 2016, 2019). Specifically, define the block operator

$$Q_{yw} = \begin{pmatrix} \sigma AA^\top + \sigma\mathcal{S}_y & -\sigma AQ \\ -\sigma QA^\top & \sigma Q^2 + Q + \sigma\mathcal{S}_w \end{pmatrix}.$$

This operator admits the decomposition

$$Q_{yw} = U_{yw} + D_{yw} + U_{yw}^\top,$$

where

$$U_{yw} = \begin{pmatrix} 0 & -\sigma AQ \\ 0 & 0 \end{pmatrix}, \qquad D_{yw} = \begin{pmatrix} \sigma AA^\top + \sigma\mathcal{S}_y & 0 \\ 0 & \sigma Q^2 + Q + \sigma\mathcal{S}_w \end{pmatrix}.$$

The associated sGS operator is given by

$$\mathcal{S}_{sGS} = UD^{-1}U^\top = \begin{pmatrix} \sigma^2 AQ(\sigma Q^2 + Q + \sigma\mathcal{S}_w)^{-1}QA^\top & 0 \\ 0 & 0 \end{pmatrix}. \tag{4.3}$$

By Theorem 2.4, Step 3 of Algorithm 5 can be solved with an sGS order, as described in Proposition 4.1.

Proposition 4.1. *Let $\mathcal{S}_y$ and $\mathcal{S}_w$ be two self-adjoint positive semidefinite linear operators on $\mathbb{R}^m$ and $\mathcal{W}$, respectively, such that $\mathcal{S}_y + AA^\top$ is positive definite. Define the operator*

$$\mathcal{S}_{yw} = \sigma \begin{pmatrix} \mathcal{S}_y & 0 \\ 0 & \mathcal{S}_w \end{pmatrix} + \mathcal{S}_{sGS}, \tag{4.4}$$

where $\mathcal{S}_{sGS}$ are given in (4.3)*. Then for any $k \geq 0$,*

$$(\bar{y}^{k+1}, \bar{w}^{k+1}) = \underset{(y,w)\in\mathbb{R}^m\times\mathcal{W}}{\arg\min} \left\{ L_\sigma(y, w, \bar{z}^{k+1}; \bar{x}^{k+1}) + \frac{1}{2}\|(y, w) - (y^k, w^k)\|^2_{\mathcal{S}_{yw}} \right\}$$

is equivalent to the following updates:

$$\begin{cases} \bar{w}^{k+\frac{1}{2}} = \arg\min\limits_{w\in\mathcal{W}} \left\{ L_\sigma(y^k, w, \bar{z}^{k+1}; \bar{x}^{k+1}) + \frac{\sigma}{2}\|w - w^k\|^2_{\mathcal{S}_w} \right\}, \\ \bar{y}^{k+1} = \arg\min\limits_{y\in\mathbb{R}^m} \left\{ L_\sigma(y, \bar{w}^{k+\frac{1}{2}}, \bar{z}^{k+1}; \bar{x}^{k+1}) + \frac{\sigma}{2}\|y - y^k\|^2_{\mathcal{S}_y} \right\}, \\ \bar{w}^{k+1} = \arg\min\limits_{w\in\mathcal{W}} \left\{ L_\sigma(\bar{y}^{k+1}, w, \bar{z}^{k+1}; \bar{x}^{k+1}) + \frac{\sigma}{2}\|w - w^k\|^2_{\mathcal{S}_w} \right\}. \end{cases}$$

Moreover, $\mathcal{S}_{yw} + \mathcal{H}$ is positive definite on $\mathbb{R}^m \times \mathcal{W}$.

Applying the Proposition 4.1, an HPR method with sGS technique for solving problem (4.1) is given in Algorithm 6.

Algorithm 6 An HPR method for solving the restricted Wolfe dual problem (4.1)

Input: Let $\mathcal{S}_y$ and $\mathcal{S}_w$ be two self-adjoint positive semidefinite linear operators on $\mathbb{R}^m$ and $\mathcal{W}$, respectively, and $\mathcal{S}_y + AA^\top$ is positive definite. Set $\sigma > 0$. Denote $u = (y, w, z, x)$ and $\bar{u} = (\bar{y}, \bar{w}, \bar{z}, \bar{x})$. Let $u^0 = (y^0, w^0, z^0, x^0) \in \mathbb{R}^m \times \mathcal{W} \times \mathbb{R}^n \times \mathbb{R}^n$.
for $k = 0, 1, 2, \ldots$ **do**
  Step 1. $\bar{z}^{k+1} = \arg\min\limits_{z\in\mathbb{R}^n} \left\{ L_\sigma(y^k, w^k, z; x^k) \right\}$;
  Step 2. $\bar{x}^{k+1} = x^k + \sigma(-Qw^k + A^\top y^k + \bar{z}^{k+1} - c)$;
  Step 3-1. $\bar{w}^{k+\frac{1}{2}} = \arg\min\limits_{w\in\mathcal{W}} \left\{ L_\sigma(y^k, w, \bar{z}^{k+1}; \bar{x}^{k+1}) + \frac{\sigma}{2}\|w - w^k\|^2_{\mathcal{S}_w} \right\}$;
  Step 3-2. $\bar{y}^{k+1} = \arg\min\limits_{y\in\mathbb{R}^m} \left\{ L_\sigma(y, \bar{w}^{k+\frac{1}{2}}, \bar{z}^{k+1}; \bar{x}^{k+1}) + \frac{\sigma}{2}\|y - y^k\|^2_{\mathcal{S}_y} \right\}$;
  Step 3-3. $\bar{w}^{k+1} = \arg\min\limits_{w\in\mathcal{W}} \left\{ L_\sigma(\bar{y}^{k+1}, w, \bar{z}^{k+1}; \bar{x}^{k+1}) + \frac{\sigma}{2}\|w - w^k\|^2_{\mathcal{S}_w} \right\}$;
  Step 4. $\hat{u}^{k+1} = 2\bar{u}^{k+1} - u^k$;
  Step 5. $u^{k+1} = \frac{1}{k+2}u^0 + \frac{k+1}{k+2}\hat{u}^{k+1}$;
end for

Remark 4.1. *The proposed HPR method, as outlined in Algorithm 6, which incorporates the sGS technique, provides a unified and flexible framework for solving general CCQP problems of the form* (1.1)*. In particular, when $Q = \mathbf{0}$ and $\phi(\cdot) = \delta_C(\cdot)$, it reduces to the HPR method for LP introduced in Chen et al. (2024a). Moreover, if the constraint $Ax \in \mathcal{K}$ is absent, such as for the LASSO problem (Li et al., 2018a), Algorithm 6 simplifies to the update cycle $z \to x \to w$. These special cases highlight the flexibility and broad applicability of the dual HPR method in Algorithm 6.*

Now we establish the convergence and complexity results for Algorithm 6. Under Assumption 4.1, solving problems (1.1) and (2.2) is equivalent to finding a vector $u^* \in \mathcal{U}$ such that $\mathbf{0} \in \mathcal{T}u^*$, where the maximal monotone operator $\mathcal{T}$ is defined by

$$\mathcal{T}u = \begin{pmatrix} -\partial\delta_{\mathcal{K}}^*(-y) + Ax \\ Qw - Qx \\ -\partial\phi^*(-z) + x \\ c - A^\top y + Qw - z \end{pmatrix} \quad \forall u = (y, w, z, x) \in \mathbb{R}^m \times \mathcal{W} \times \mathbb{R}^n \times \mathbb{R}^n.$$

We define the linear operator $\mathcal{M} : \mathcal{U} \to \mathcal{U}$ as

$$\mathcal{M} = \begin{bmatrix} \sigma AA^\top + \mathcal{S}_{sGS} + \mathcal{S}_y & -\sigma AQ & \mathbf{0} & A \\ -\sigma QA^\top & \sigma Q^2 + \mathcal{S}_w & \mathbf{0} & Q \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ A^\top & Q & \mathbf{0} & \frac{1}{\sigma} I_n \end{bmatrix}. \tag{4.5}$$

Under Assumption 4.2, each subproblem in Algorithm 6 admits a unique solution. Based on Theorem 2.2, we establish the following convergence result for Algorithm 6.

Theorem 4.1. *Suppose that Assumptions 4.1 and 4.2 hold. Then the sequence* $\{\bar{u}^k\} = \{(\bar{y}^k, \bar{w}^k, \bar{z}^k, \bar{x}^k)\}$ *generated by the HPR method with semi-proximal terms in Algorithm 6 converges to the point* $u^* = (y^*, w^*, z^*, x^*)$*, where* (y^*, w^*, z^*) *solves the dual problem* (4.1) *and* x^* *solves the primal problem* (1.1)*.*

The residual mapping associated with the KKT system (4.2) was introduced in Han et al. (2018), as follows:

$$\mathcal{R}(u) = \begin{pmatrix} Ax - \Pi_{\mathcal{K}}(Ax - y) \\ Qw - Qx \\ x - \mathrm{Prox}_{\phi}(x - z) \\ c - A^\top y + Qw - z \end{pmatrix} \quad \forall\, u = (y, w, z, x) \in \mathbb{R}^m \times \mathcal{W} \times \mathbb{R}^n \times \mathbb{R}^n.$$

For the sequence $\{(\bar{y}^k, \bar{w}^k, \bar{z}^k)\}$ generated by Algorithm 6, we define the objective

error as

$$h(\bar{y}^k, \bar{w}^k, \bar{z}^k) := \frac{1}{2}\langle \bar{w}^k, Q\bar{w}^k\rangle + \delta^*_{\mathcal{K}}(-\bar{y}^k) + \phi^*(-\bar{z}^k) \\ - \left(\frac{1}{2}\langle w^*, Qw^*\rangle + \delta^*_{\mathcal{K}}(-y^*) + \phi^*(-z^*)\right) \quad \forall\, k \geq 1,$$

where (y^*, w^*, z^*) is a solution to the dual problem (4.1). Based on the iteration complexity results in Theorem 2.3, we obtain the following iteration complexity bounds for the HPR method with semi-proximal terms for solving the restricted Wolfe dual problem (4.1).

Theorem 4.2. *Suppose that Assumptions 4.1 and 4.2 hold. Let* $\{\bar{u}^k\} = \{(\bar{y}^k, \bar{w}^k, \bar{z}^k, \bar{x}^k)\}$ *and* $\{u^k\} = \{(y^k, w^k, z^k, x^k)\}$ *be the sequences generated by the HPR method with semi-proximal terms in Algorithm 6, and let* $u^* = (y^*, w^*, z^*, x^*)$ *be a solution to the KKT system* (4.2). *Define* $R_0 := \|u^0 - u^*\|_{\mathcal{M}}$, *where* $\mathcal{M}$ *is given by* (4.5). *Then the following complexity bounds hold for all* $k \geq 0$*:*

$$\|\bar{u}^{k+1} - u^k\|_{\mathcal{M}} \leq \frac{R_0}{k+1}, \quad \|\mathcal{R}(\bar{u}^{k+1})\| \leq \left(\frac{\sigma\|A_Q^\top\| + 1}{\sqrt{\sigma}} + \|\sqrt{\mathcal{S}_{yw}}\|\right)\frac{R_0}{k+1},$$

$$-\frac{1}{\sqrt{\sigma}}\|x^*\| \cdot \frac{R_0}{k+1} \;\leq\; h(\bar{y}^{k+1}, \bar{w}^{k+1}, \bar{z}^{k+1}) \;\leq\; \left(3R_0 + \frac{1}{\sqrt{\sigma}}\|x^*\|\right)\frac{R_0}{k+1},$$

where $A_Q = [A^\top - Q]$ *and* S_{yw} *is defined in* (4.4).

4.2 HPR-QP: A convex composite quadratic programming solver based on the dual HPR method

We adopt Algorithm 6 as the main scheme for solving CCQP, and develop the easy-to-implement solver HPR-QP by incorporating algorithmic enhancements such as restart strategies and penalty parameter updates, as detailed in the following subsections.

4.2.1 Efficient solution of subproblems

We will introduce the computation details of Step 3 of Algorithm 6, starting from the subproblem of y in Step 3-2. In **Step 3-2**, the optimality condition for the subproblem is given by

$$0 \in \partial\delta^*_{\mathcal{K}}(-\bar{y}^{k+1}) + A\bar{x}^{k+1} + \sigma A(-Q\bar{w}^{k+\frac{1}{2}} + A^\top\bar{y}^{k+1} + \bar{z}^{k+1} - c) + \sigma\mathcal{S}_y(\bar{y}^{k+1} - y^k).$$

The above condition can be equivalent to a closed-form expression (linearization), or a linear system (direct). We will introduce two practical choices of $\mathcal{S}_y$ and the corresponding implementations.

Direct. When $\mathcal{K} = b \in \mathbb{R}^m$, we can choose $\mathcal{S}_y = \eta I_m$, where $\eta > 0$ is a constant. Then the update of $\bar{y}^{k+1}$ can be computed by solving the linear system

$$\left(AA^\top + \eta I_m\right)\bar{y}^{k+1} = \frac{1}{\sigma}(b - A(\bar{x}^{k+1} + \sigma(-Q\bar{w}^{k+\frac{1}{2}} + \bar{z}^{k+1} - c))) + \eta y^k. \tag{4.6}$$

The main computation is on the factorization of the matrix $AA^\top + \eta I_m$ (only one time) and the subsequent triangular solves.

Linearization. We can choose the linearization semiproximal operator as

$$\mathcal{S}_y = \lambda_A I_m - AA^\top, \tag{4.7}$$

where $\lambda_A \geq \lambda_{\max}(AA^\top)$ is a constant. Then the update of $\bar{y}^{k+1}$ can be computed by

$$\bar{y}^{k+1} = \frac{1}{\sigma\lambda_A}\left(\Pi_{\mathcal{K}}(r_y^k) - r_y^k\right), \tag{4.8}$$

where $r_y^k = A(\bar{x}^{k+1} + \sigma(-Q\bar{w}^{k+\frac{1}{2}} + A^\top y^k + \bar{z}^{k+1} - c)) - \sigma\lambda_A y^k$. The main computation is on the matrix-vector multiplication and vector operations.

In **Steps 3-1 and 3-3**, our target is to find a $\bar{w}^{k+\frac{1}{2}} \in \mathcal{W}$ and $\bar{w}^{k+1} \in \mathcal{W}$ which satisfies the condition

$$\begin{cases} Q\bar{w}^{k+\frac{1}{2}} - Q\bar{x}^{k+1} - \sigma Q(-Q\bar{w}^{k+\frac{1}{2}} + A^\top y^k + \bar{z}^{k+1} - c) + \sigma\mathcal{S}_w(\bar{w}^{k+\frac{1}{2}} - w^k) = 0, \\ Q\bar{w}^{k+1} - Q\bar{x}^{k+1} - \sigma Q(-Q\bar{w}^{k+1} + A^\top \bar{y}^{k+1} + \bar{z}^{k+1} - c) + \sigma\mathcal{S}_w(\bar{w}^{k+1} - w^k) = 0. \end{cases}$$

We use the computation of $\bar{w}^{k+1}$ as the example to illustrate the details, where both the Wolfe dual and the restricted Wolfe dual formulations are discussed for comparison.

Direct. For the Wolfe dual formulation $\mathcal{W} = \mathbb{R}^n$, we set $\mathcal{S}_w = \eta I_n$, then Step 3-3 is equivalent to the following positive definite linear system

$$(Q(\sigma Q + I_n) + \sigma\eta I_n)w = Q\big(\bar{x}^{k+1} + \sigma(A^\top \bar{y}^{k+1} + \bar{z}^{k+1} - c)\big) + \sigma\eta w^k. \tag{4.9}$$

The linear system (4.9) can be solved by the direct method if the explicit matrix formulation of Q is available. If Q has no explicit formulation, we can use iterative solvers to solve it.

For the restricted Wolfe dual formulation $\mathcal{W} = \text{Range}(Q)$, we set $\mathcal{S}_w = 0$, then Step 3-3 is equivalent to the following linear system

$$Q(\sigma Q + I_n)w = Q\big(\bar{x}^{k+1} + \sigma(A^\top \bar{y}^{k+1} + \bar{z}^{k+1} - c)\big), \quad w \in \mathcal{W}.$$

To handle the constraint $w \in \mathcal{W}$, we can obtain an alternative solution w_Q by the following proposition (Li et al., 2018b, Proposition 4.1).

Proposition 4.2. *Let $\mathcal{W} = \text{Range}(Q)$. The linear system*

$$Q(\sigma Q + I_n)w = Qr_w. \tag{4.10}$$

and the solution w_Q of the linear system

$$(\sigma Q + I_n)w = r_w.$$

have the following relations:

(a) $w_Q^+ = \Pi_{\mathcal{W}}(w_Q)$ *is a solution of the linear system* (4.10)*;*

(b) $Qw_Q^+ = Qw_Q$ *and* $\langle w_Q^+, Qw_Q^+\rangle = \langle w_Q, Qw_Q\rangle$*.*

These properties imply that we can obtain the alternative solutions $\bar{w}_Q^{k+1}$ by solving the linear system

$$(\sigma Q + I_n)w = \bar{x}^{k+1} + \sigma(A^\top \bar{y}^{k+1} + \bar{z}^{k+1} - c), \tag{4.11}$$

and $\bar{w}_Q^{k+\frac{1}{2}}$ by solving the linear system

$$(\sigma Q + I_n)w = \bar{x}^{k+1} + \sigma(A^\top y^k + \bar{z}^{k+1} - c). \tag{4.12}$$

Linearization. We introduce similar results for the linearization case. For the Wolfe dual formulation $\mathcal{W} = \mathbb{R}^n$, we set $\mathcal{S}_w = \tilde{\lambda}_Q I_n - (Q^2 + Q/\sigma)$, where $\tilde{\lambda}_Q \geq \lambda_{\max}(Q^2 + Q/\sigma)$ is a constant, then Step 3-3 is equivalent to the following formula

$$\bar{w}^{k+1} = \frac{1}{\tilde{\lambda}_Q}\left(\tilde{\lambda}_Q w^k + Q(\bar{x}^{k+1} - w^k + \sigma(-Qw^k + A^\top \bar{y}^{k+1} + \bar{z}^{k+1} - c))\right).$$

For the restricted Wolfe dual formulation $\mathcal{W} = \text{Range}(Q)$, we set

$$\mathcal{S}_w = Q(\lambda_Q I_n - Q), \tag{4.13}$$

where $\lambda_Q \geq \lambda_{\max}(Q)$ is a constant, then $\bar{w}^{k+1}$ is the solution of the following formula

$$Q\bar{w}^{k+1} = \frac{1}{1+\sigma\lambda_Q} Q\left(\sigma\lambda_Q w^k + \bar{x}^{k+1} + \sigma(-Qw^k + A^\top \bar{y}^{k+1} + \bar{z}^{k+1} - c)\right), \quad \bar{w}^{k+1} \in \mathcal{W}. \tag{4.14}$$

To handle the constraint $w \in \mathcal{W}$, we can obtain an alternative solution w_Q according to the following proposition.

Proposition 4.3. *Let* $\mathcal{W} = \mathrm{Range}(Q)$ *and* $r_w \in \mathbb{R}^n$. *For* w_Q *with the following formula*

$$w_Q = \frac{1}{1+\sigma\lambda_Q}\left(\sigma\lambda_Q w^k + \bar{x}^{k+1} + \sigma(-Qw^k + A^\top \bar{y}^{k+1} + \bar{z}^{k+1} - c)\right).$$

We have the following properties:

(a) $w_Q^+ = \Pi_{\mathcal{W}}(w_Q)$ *is a solution of the* (4.14)*;*

(b) $Qw_Q^+ = Qw_Q$ *and* $\langle w_Q^+, Qw_Q^+\rangle = \langle w_Q, Qw_Q\rangle$.

The proposition implies that we can obtain the alternative solutions $\bar{w}_Q^{k+1}$ by the formula

$$\bar{w}_Q^{k+1} = \frac{1}{1+\sigma\lambda_Q}\left(\sigma\lambda_Q w^k + \bar{x}^{k+1} + \sigma(-Qw^k + A^\top \bar{y}^{k+1} + \bar{z}^{k+1} - c)\right), \tag{4.15}$$

and $\bar{w}_Q^{k+\frac{1}{2}}$ by the formula

$$\bar{w}_Q^{k+\frac{1}{2}} = \frac{1}{1+\sigma\lambda_Q}\left(\sigma\lambda_Q w^k + \bar{x}^{k+1} + \sigma(-Qw^k + A^\top y^k + \bar{z}^{k+1} - c)\right). \tag{4.16}$$

Remark 4.2. *In Algorithm 6, the updates for* $\bar{w}^{k+\frac{1}{2}}$ *and* $\bar{w}^{k+1}$ *for* $k \geq 0$ *are restricted to the subspace* $\mathcal{W} = \mathrm{Range}(Q)$. *Although it may seem easier to update* $\bar{w}^{k+\frac{1}{2}}$ *and* $\bar{w}^{k+1}$ *in the full space* $\mathbb{R}^n$, *for example, under the linearization case, the proximal operator has a larger spectral norm, such as* $\mathcal{S}_w = \lambda_{max}(Q^2 + Q/\sigma)I_n - (Q^2 + Q/\sigma)$. *Restricting the update to* $\mathcal{W}$ *allows HPR-QP to employ a proximal operator with a smaller spectral norm, namely* $\mathcal{S}_w = Q(\lambda_1(Q)I_n - Q)$. *The algorithm using the proximal operator with a smaller spectral norm theoretically converges faster.*

Based on Propositions 4.2 and 4.3, an easy-to-implement dual HPR method for the restricted-Wolfe dual (4.1) is presented in Algorithm 7. The convergence of Algorithm 7 is established in Theorem 4.3.

Algorithm 7 An easy-to-implement dual HPR method for the restricted-Wolfe dual problem (4.1)

Input: Let $\mathcal{S}_w$ be the linearization proximal term $Q(\lambda_Q I_n - Q)$ or 0, and let $\mathcal{S}_y$ be the linearization proximal term $\lambda_A I_m - AA^\top$ or ηI_m with $\eta > 0$. Denote $u_Q = (y, w_Q, z, x)$ and $\bar{u}_Q = (\bar{y}, \bar{w}_Q, \bar{z}, \bar{x})$. Let $u_Q^0 = (y^0, w_Q^0, z^0, x^0) \in \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n$, and set $\sigma > 0$.

for $k = 0, 1, 2, \ldots$ **do**

Step 1. $\bar{z}^{k+1} = \arg\min_{z \in \mathbb{R}^n} \left\{ L_\sigma(y^k, w_Q^k, z; x^k) \right\}$;

Step 2. $\bar{x}^{k+1} = x^k + \sigma(-Qw_Q^k + A^\top y^k + \bar{z}^{k+1} - c)$;

Step 3-1. Update $\bar{w}_Q^{k+\frac{1}{2}}$ by (4.16) for linearization, otherwise (4.12);

Step 3-2. Update $\bar{y}^{k+1}$ by (4.8) for linearization, otherwise (4.6);

Step 3-3. Update $\bar{w}_Q^{k+1}$ by (4.15) for linearization, otherwise (4.11);

Step 4. $\hat{u}_Q^{k+1} = 2\bar{u}_Q^{k+1} - u_Q^k$;

Step 5. $u_Q^{k+1} = \dfrac{1}{k+2} u_Q^0 + \dfrac{k+1}{k+2} \hat{u}_Q^{k+1}$;

end for

Theorem 4.3. *Suppose Assumption 4.1 holds. Then the sequence* $\{(\bar{y}^k, Q\bar{w}_Q^k, \bar{z}^k, \bar{x}^k)\}$ *generated by Algorithm 7 is equivalent to the sequence* $\{(\bar{y}^k, Q\bar{w}^k, \bar{z}^k, \bar{x}^k)\}$ *produced by Algorithm 6 starting from the same initial point. Moreover, both sequences converge to the point* (y^*, Qw^*, z^*, x^*)*, where* (y^*, w^*, z^*) *solves the dual problem* (4.1) *and* x^* *solves the primal problem* (1.1)*.*

Proof. We state the proof for the linearization case, and the proof for the direct case follows similarly. According to the formulas and optimality conditions of the subproblems in Algorithm 6, for any $k \geq 0$, we have

$$\begin{cases} \mathbf{0} \in -\partial\phi^*(-\bar{z}^{k+1}) + x^k + \sigma(-Qw^k + A^\top y^k + \bar{z}^{k+1} - c), \\ \bar{x}^{k+1} = x^k + \sigma(-Qw^k + A^\top y^k + \bar{z}^{k+1} - c), \\ Q\bar{w}^{k+\frac{1}{2}} - Q\bar{x}^{k+1} - \sigma Q(-Q\bar{w}^{k+\frac{1}{2}} + A^\top y^k + \bar{z}^{k+1} - c) + \sigma\mathcal{S}_w(\bar{w}^{k+\frac{1}{2}} - w^k) = 0, \quad \bar{w}^{k+\frac{1}{2}} \in \mathcal{W}, \\ \mathbf{0} \in -\partial\delta_{\mathcal{K}}^*(-\bar{y}^{k+1}) + A\bar{x}^{k+1} + \sigma A(-Q\bar{w}^{k+\frac{1}{2}} + A^\top \bar{y}^{k+1} + \bar{z}^{k+1} - c) + \sigma\mathcal{S}_y(\bar{y}^{k+1} - y^k), \\ Q\bar{w}^{k+1} - Q\bar{x}^{k+1} - \sigma Q(-Q\bar{w}^{k+1} + A^\top \bar{y}^{k+1} + \bar{z}^{k+1} - c) + \sigma\mathcal{S}_w(\bar{w}^{k+1} - w^k) = 0, \quad \bar{w}^{k+1} \in \mathcal{W}. \end{cases} \tag{4.17}$$

Based on (4.17) and Propositions 4.2 and 4.3, we can derive that the sequence

$\{(\bar{y}^k, Q\bar{w}_Q^k, \bar{z}^k, \bar{x}^k)\}$ generated by Algorithm 7 is equivalent to $\{(\bar{y}^k, Q\bar{w}^k, \bar{z}^k, \bar{x}^k)\}$ from Algorithm 6, under the same initialization.

Furthermore, Proposition 4.1 establishes that Algorithm 6 is a special case of Algorithm 5 with $\mathcal{S}_{yw}$ defined in (4.4). Thus, the convergence result in Theorem 4.1 applies, and the sequence $\{(\bar{y}^k, \bar{w}^k, \bar{z}^k, \bar{x}^k)\}$ from Algorithm 6 converges to the optimal solution of the primal-dual pair (1.1) and (4.1). This completes the proof. □

The main computation overhead of Algorithm 7 is the matrix-vector multiplication for the linearization case, and is the linear system solving for the direct case. For the linearization case, the computational cost for each iteration of Steps 1-3 in Algorithm 7 can be further reduced. We summarize them as follows:

- **Step 1.** The update for $\bar{z}^{k+1}$ can be computed by

$$\bar{z}^{k+1} = \frac{1}{\sigma}\left(\mathrm{Prox}_{\phi}(r_z^k) - r_z^k\right),$$

  where $r_z^k = x^k + \sigma(-Qw_Q^k + A^\top y^k - c)$.

- **Step 2.** The update for $\bar{x}^{k+1}$ can be expressed by

$$\bar{x}^{k+1} = x^k + \sigma(-Qw_Q^k + A^\top y^k + \bar{z}^{k+1} - c) = \mathrm{Prox}_{\phi}(r_z^k).$$

- **Step 3-1.** The update for $\bar{w}_Q^{k+\frac{1}{2}}$ can be computed by

$$\begin{aligned}\bar{w}_Q^{k+\frac{1}{2}} &= \frac{1}{1+\sigma\lambda_Q}\left(\sigma\lambda_Q w_Q^k + \bar{x}^{k+1} + \sigma(-Qw_Q^k + A^\top y^k + \bar{z}^{k+1} - c)\right)\\ &= \frac{1}{1+\sigma\lambda_Q}\left(\sigma\lambda_Q w_Q^k + 2\bar{x}^{k+1} - x^k\right) = \frac{1}{1+\sigma\lambda_Q}\left(\sigma\lambda_Q w_Q^k + \hat{x}^{k+1}\right).\end{aligned}$$

- **Step 3-2.** The update for $\bar{y}^{k+1}$ can be evaluated by

$$\bar{y}^{k+1} = \frac{1}{\sigma\lambda_A}\left(\Pi_{\mathcal{K}}(r_y^{k+1}) - r_y^{k+1}\right),$$

where

$$r_y^{k+1} = A\left(\bar{x}^{k+1} + \sigma(-Q\bar{w}_Q^{k+\frac{1}{2}} + A^\top y^k + \bar{z}^{k+1} - c)\right) - \sigma\lambda_A y^k$$
$$= A(\hat{x}^{k+1} + \sigma(Qw_Q^k - Q\bar{w}_Q^{k+\frac{1}{2}})) - \sigma\lambda_A y^k.$$

- **Step 3-3.** The update for $\bar{w}_Q^{k+1}$ can be computed by

$$\bar{w}_Q^{k+1} = \frac{1}{1+\sigma\lambda_Q}\left(\sigma\lambda_Q w_Q^k + \bar{x}^{k+1} + \sigma(-Qw_Q^k + A^\top\bar{y}^{k+1} + \bar{z}^{k+1} - c)\right)$$
$$= \left(\bar{w}_Q^{k+\frac{1}{2}} + \frac{\sigma}{1+\sigma\lambda_Q}A^\top(\bar{y}^{k+1} - y^k)\right).$$

Remark 4.3. *The update of $\hat{x}^{k+1}$ is presented in Step 4 of Algorithm 7, but we can compute it after Step 2. The value of $\bar{z}^{k+1}$ does not have to be explicitly computed for every iteration, unless we will compute the residuals for stopping criteria in the current iteration.*

4.2.2 Restart and penalty parameter selection

In Algorithm 8, we present HPR-QP for solving problem (4.1), incorporating the adaptive restart and penalty parameter update strategies that will be introduced in this section.

Algorithm 8 HPR-QP: A dual HPR method for the CCQP (4.1)

Input: Let $\mathcal{S}_w$ be the linearization proximal operator $Q(\lambda_Q I_n - Q)$ or 0, and let $\mathcal{S}_y$ be the linearization proximal operator $\lambda_A I_m - AA^\top$ or ηI_m with $\eta > 0$. Let $u_Q = (y, w_Q, z, x)$, $\bar{u}_Q = (\bar{y}, \bar{w}_Q, \bar{z}, \bar{x})$, and initial point $u_Q^{0,0} = (y^{0,0}, w_Q^{0,0}, z^{0,0}, x^{0,0}) \in \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n$.
Initialization: Set outer loop counter $r = 0$, total iteration counter $k = 0$, and initial penalty parameter $\sigma_0 > 0$.
repeat
  Initialize inner loop: set inner counter $t = 0$;
  repeat
    $\bar{z}^{r,t+1} = \arg\min_{z \in \mathbb{R}^n} \{L_{\sigma_r}(y^{r,t}, w_Q^{r,t}, z; x^{r,t})\}$;
    $\bar{x}^{r,t+1} = x^{r,t} + \sigma_r(-Qw_Q^{r,t} + A^\top y^{r,t} + \bar{z}^{r,t+1} - c)$;
    Update $\bar{w}_Q^{r,t+\frac{1}{2}}$ by (4.16) for linearization, otherwise (4.12);
    Update $\bar{y}^{r,t+1}$ by (4.8) for linearization, otherwise (4.6);
    Update $\bar{w}_Q^{r,t+1}$ by (4.15) for linearization, otherwise (4.11);
    $\hat{u}_Q^{r,t+1} = 2\bar{u}_Q^{r,t+1} - u_Q^{r,t}$;
    $u_Q^{r,t+1} = \frac{1}{k+2} u_Q^{r,0} + \frac{k+1}{k+2} \hat{u}_Q^{r,t+1}$;
    $t \leftarrow t + 1$, $k \leftarrow k + 1$;
  until restart or termination criteria are met;
  Restart: Set $\tau_r = t$, $u_Q^{r+1,0} = \bar{u}_Q^{r,\tau_r}$;
  $\sigma_{r+1} = \textbf{SigmaUpdate}(\bar{u}_Q^{r,\tau_r}, u_Q^{r,0}, \mathcal{S}_y, \mathcal{S}_w, A, Q)$;
  $r \leftarrow r + 1$;
until termination criteria are met;
Output: $\{\bar{u}^{r,t}\}$.

Inspired by the restart strategy proposed in PDLP (Applegate et al., 2021; Lu and Yang, 2023; Lu et al., 2023), Chen et al. (2024a) introduced an adaptive restart mechanism based on the $O(1/k)$ iteration complexity with respect to KKT residual, which has demonstrated practical success on large-scale LP problems. Motivated by this, we extend this adaptive restart strategy to the CCQP (1.1) by defining a suitable merit function grounded in the complexity result established in Theorem 4.2. Specifically, we define the following idealized merit function:

$$R_{r,t} := \|u_Q^{r,t} - u^*\|_{\mathcal{M}} \quad \forall r \geq 0,\ t \geq 0,$$

where u^* denotes any solution to the KKT system (4.2). Note that $R_{r,0}$ corresponds

to the upper bound given by the complexity result in Theorem 4.2 at the start of the r-th outer iteration. Since u^* is unknown in practice, we approximate $R_{r,t}$ using the computable surrogate:

$$\widetilde{R}_{r,t} := \|u_Q^{r,t} - \bar{u}_Q^{r,t+1}\|_{\mathcal{M}}.$$

Based on this surrogate merit function, we introduce the following adaptive restart criteria for the HPR-QP method:

1. Sufficient decay:
$$\widetilde{R}_{r,t+1} \leq \alpha_1 \widetilde{R}_{r,0}; \tag{4.18}$$

2. Insufficient local progress despite overall decay:
$$\widetilde{R}_{r,t+1} \leq \alpha_2 \widetilde{R}_{r,0} \quad \text{and} \quad \widetilde{R}_{r,t+1} > \widetilde{R}_{r,t};$$

3. Excessively long inner loop:
$$t \geq \alpha_3 k; \tag{4.19}$$

where $\alpha_1 \in (0, \alpha_2)$, $\alpha_2 \in (0, 1)$, and $\alpha_3 \in (0, 1)$ are user-defined parameters. Whenever any of the above conditions is satisfied, we terminate the current inner loop and begin a new outer iteration by setting $u^{r+1,0} = \bar{u}^{r,\tau_r}$ and updating the penalty parameter σ_{r+1} accordingly.

Now, we describe the update rule for the penalty parameter σ in HPR-QP. Specifically, at the beginning of the $(r+1)$-th outer iteration, we determine σ_{r+1} by solving the following minimization problem:

$$\sigma_{r+1} = \arg\min_{\sigma>0} \|u_Q^{r+1,0} - u^*\|_{\mathcal{M}}^2, \tag{4.20}$$

where u^* denotes a solution to the KKT system (4.2), and the metric $\|u_Q^{r+1,0} - u^*\|_{\mathcal{M}}$ corresponds to the upper bound derived in Theorem 4.2. The rationale behind this

formulation is that minimizing this bound is expected to yield a smaller quantity $\|u_Q^{r+1,t} - \bar{u}_Q^{r+1,t+1}\|_{\mathcal{M}}$ for all $t \geq 0$, which reduces the KKT residual $\|\mathcal{R}(\bar{u}_Q^{r+1,t+1})\|$. This guides the adjustment of σ to enhance the practical performance of the method.

For the linearization case (similar for the direct case but some approximation needed), substituting (4.3), (4.7), (4.13), and (4.5) into (4.20), the update rule for σ_{r+1} reduces to solving the following one-dimensional minimization problem:

$$\sigma_{r+1} = \arg\min_{\sigma>0} \left\{ f(\sigma) := \theta_1 \sigma + \frac{\theta_2}{\sigma} + \frac{\sigma^2 \theta_3}{1 + \lambda_Q \sigma} \right\}, \tag{4.21}$$

where the coefficients $\theta_1, \theta_2, \theta_3$ are given by

$$\theta_1 = \lambda_A \|y^{r+1,0} - y^*\|^2 + \lambda_Q \|w_Q^{r+1,0} - w^*\|_Q^2 - 2\langle Q(w_Q^{r+1,0} - w^*), A^\top (y^{r+1,0} - y^*)\rangle,$$
$$\theta_2 = \|x^{r+1,0} - x^*\|^2, \quad \theta_3 = \|A^\top y^{r+1,0} - A^\top y^*\|_Q^2.$$

When coefficients $\theta_i > 0$, $i = 2, 3$, the function $f(\sigma)$ is strictly convex on $\sigma > 0$, since its second derivative satisfies

$$f''(\sigma) = \frac{2\theta_2}{\sigma^3} + \frac{2\theta_3}{(1 + \lambda_Q \sigma)^3} > 0 \quad \text{for all } \sigma > 0.$$

This guarantees both the existence and uniqueness of the optimal solution to problem (4.21). This scalar optimization problem can be efficiently solved using the golden section search method (Gerald, 2004). Moreover, since the exact values of θ_i involve the solution u^*, we estimate them as

$$\widetilde{\theta}_1 = \lambda_A \|\bar{y}^{r,\tau_r} - y^{r,0}\|^2 + \lambda_Q \|\bar{w}_Q^{r,\tau_r} - w_Q^{r,0}\|_Q^2 - 2\langle Q(\bar{w}_Q^{r,\tau_r} - w_Q^{r,0}), A^\top (\bar{y}^{r,\tau_r} - y^{r,0})\rangle,$$
$$\widetilde{\theta}_2 = \|\bar{x}^{r,\tau_r} - x^{r,0}\|^2, \quad \widetilde{\theta}_3 = \|A^\top \bar{y}^{r,\tau_r} - A^\top y^{r,0}\|_Q^2. \tag{4.22}$$

Using these approximations, the updated penalty parameter σ is obtained by solving the following scalar optimization problem:

$$\sigma_{\text{new}} = \arg\min_{\sigma>0} \left\{ f(\sigma) = \widetilde{\theta}_1 \sigma + \frac{\widetilde{\theta}_2}{\sigma} + \frac{\sigma^2 \widetilde{\theta}_3}{1 + \lambda_Q \sigma} \right\}. \tag{4.23}$$

To further stabilize the update and prevent the approximations from deviating significantly from the true values, we apply an exponential smoothing scheme. The complete update procedure for σ is summarized in Algorithm 9.

Algorithm 9 SigmaUpdate

1: **Input:** $\bar{u}_Q^{r,\tau_r}, u_Q^{r,0}, \lambda_A, \lambda_Q, A, Q$.
2: Compute $\widetilde{\theta}_1, \widetilde{\theta}_2, \widetilde{\theta}_3$ as defined in (4.22);
3: Ensure numerical stability: $\widetilde{\theta}_i \leftarrow \max\big(\widetilde{\theta}_i, 10^{-12}\big),\ i = 1, 2$;
4: Solve problem (4.23) using golden section search method to obtain σ_{new};
5: Compute smoothing factor: $\beta = \exp\left(-\widetilde{R}_{r,\tau_r-1}/\widetilde{R}_{0,\tau_0-1}\right)$;
6: **Output:** $\sigma_{r+1} = \exp\big(\beta \log(\sigma_{\text{new}}) + (1-\beta)\log(\sigma_r)\big)$.

Chapter 5

Numerical experiments

In this chapter, we will first present the experimental setup, including the datasets, evaluation metrics, and implementation details. Then, we will conduct some case studies to demonstrate the effect of different components of the proposed methods. At last, we will compare our methods with some state-of-the-art solvers for convex composite quadratic programming.

5.1 Experimental setup

Julia. We implement the proposed optimization algorithm in Julia (Bezanson et al., 2017), a modern programming language designed for high-performance numerical computing. Julia provides a natural balance between the ease of a high-level dynamic language and the efficiency of low-level implementations, making it particularly well-suited for large-scale optimization research. Its built-in support for parallelism, seamless GPU integration, and efficient linear algebra backends allow us to directly exploit GPU acceleration. Since our solver is dominated by matrix-vector multiplications and vector-level operations, Julia's just-in-time (JIT) compilation and array-oriented abstractions enable us to implement the algorithm in a concise yet highly efficient way, achieving scalability across both CPU and GPU architectures.

Main algorithm. Our main algorithm, termed HPR-QP and presented in Algorithm 8, is the HPR method with semi-proximal terms applied to the restricted Wolfe dual formulation of CCQP, augmented with adaptive restart, penalty parameter updating, and preconditioning techniques. For comparison, we also implement several related methods, including variations of preconditioned ADMM and the Halpern Douglas–Rachford (HDR) method.

Linearization. For the linearization case, we use the power method (Golub and Van Loan, 2013) to compute the largest eigenvalue of Q or $AA^\top$. The major computations in the algorithms are matrix-vector multiplications and vector operations.

Direct. For the direct case, the linear system $(AA^\top + \varepsilon I_m)y = \text{rhs}$ is reformulated as the quasi-definite augmented system

$$\begin{bmatrix} \varepsilon & A \\ A^\top & -I_n \end{bmatrix} \begin{bmatrix} y \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} \text{rhs} \\ 0 \end{bmatrix},$$

which usually uses less memory for large-scale problems. The computation bottleneck lies in the linear system solvers.

CPU implementation. The sparse matrices are stored in the compressed sparse column (CSC) format. We use the Julia built-in function *mul!* for sparse matrix vector multiplication, Cholesky factorization in the SuiteSparse library for the symmetric positive definite linear system, and the $LDL^\top$ factorization in the QDLDL library for the quasi-definite linear system.

GPU implementation. The sparse matrices are stored in the compressed sparse row (CSR) format. We use the function *CUDA.CUSPARSE.mv!* in the CUSPARSE library for sparse matrix vector multiplication, where the underlying algorithm is

set to CUSPARSE_SPMV_CSR_ALG2 for the reproductive results. We implement a customized sparse vector multiplication that is faster than the CUSPARSE library when the instance is small or sparse enough. We use the CUDSS library for the linear system solver, which supports the direct solver for sparse linear systems.

Computing Environment. All solvers are benchmarked on a SuperServer SYS-420GP-TNR equipped with an NVIDIA A100-SXM4-80GB GPU, an Intel Xeon Platinum 8338C CPU @ 2.60 GHz, and 256 GB of RAM. All the experiments are conducted in Julia.

Real-world Datasets. We evaluate the solvers on several real-world datasets. These include Mittelmann LP instances (https://plato.asu.edu/ftp/lpfeas.html), a benchmark for most commercial and open-source LP solvers; Maros–Mészáros QP dataset (Maros and Mészáros, 1999), a commonly used benchmark for convex QP solvers; LASSO instances from UCI Machine Learning Repository (used in Li et al. (2018a)), originally sourced from the LIBSVM datasets (Chang and Lin, 2011); QAP relaxation instances from the QAPLIB dataset (Burkard et al., 1997).

Synthetic Datasets. Following OSQP (Stellato et al., 2020), we generate 30 instances from 6 problem classes (random QP, equality constrained QP, optimal control, portfolio optimization, huber fitting), and 5 extremely large LASSO instances as a supplement for the experiment on LASSO problems. We randomly generate large-scale QAP instances to show the scalability of HPR-QP.

Parameter Setting. HPR-QP is initialized at the origin, and the initial penalty parameter is set to $\sigma_0 = \|b\|_\infty / \|c\|_\infty$, when both $\|b\|$ and $\|c\|$ lie within the range

$[10^{-16}, 10^{16}]$.[1] Otherwise, we set $\sigma_0 = 1$ to ensure numerical stability. The adaptive restart mechanism follows the criteria in (4.18)–(4.19), with parameters $\alpha_1 = 0.2$, $\alpha_2 = 0.8$, and $\alpha_3 = 0.5$. If the residual ratio satisfies $\widetilde{R}_{r,\tau_r-1}/\widetilde{R}_{0,\tau_0-1} \leq 0.1$, then α_3 is tightened to 0.2. The penalty parameter σ is updated according to Algorithm 9. We use the linearization proximal terms if not specified otherwise.

Preconditioning. The diagonal preconditioning is a commonly used technique to improve the numerical performance of first-order methods such as SCS (O'donoghue et al., 2016) and OSQP (Stellato et al., 2020). Consider the following CQP problem with an explicit matrix representation of Q:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \frac{1}{2}\langle x, Qx \rangle + \langle c, x \rangle \\ \text{s.t.} \quad & Ax \in \mathcal{K}, \\ & x \in \mathcal{C}. \end{aligned}$$

We employ a diagonal preconditioning strategy similar to PDQP (Lu and Yang, 2025) and HPR-LP (Chen et al., 2024a). In detail, we perform 10 iterations of Ruiz equilibration (Ruiz, 2001), followed by the diagonal preconditioning method of Pock and Chambolle (Pock and Chambolle, 2011) with parameter $\alpha = 1$. Either scaling technique is applied to the following system

$$G = \begin{bmatrix} Q & A^\top \\ A & 0 \end{bmatrix},$$

which results in a scaling matrix

$$D = \begin{bmatrix} D_C & 0 \\ 0 & D_R \end{bmatrix}$$

[1] The vector $b := \max(|l|, |u|)$ is taken componentwise, treating any infinite entries in l or u as zero when computing the norm.

such that the preconditioned system becomes DGD. Specifically, the preconditioned problem can be expressed as

$$\begin{aligned} \min_{\tilde{x}\in\mathbb{R}^n} \quad & \frac{1}{2}\langle \tilde{x}, \tilde{Q}\tilde{x}\rangle + \langle \tilde{c}, \tilde{x}\rangle \\ \text{s.t.} \quad & \tilde{A}\tilde{x} \in \tilde{\mathcal{K}}, \\ & \tilde{x} \in \tilde{\mathcal{C}}, \end{aligned}$$

where $\tilde{Q} = D_C Q D_C, \tilde{c} = D_C c$, $\tilde{A} = D_R A D_C$, $\tilde{\mathcal{K}} = D_R\mathcal{K}$, and $\tilde{\mathcal{C}} = D_C^{-1}\mathcal{C}$. The variable $\tilde{x} = D_C^{-1}x$.

The scaled dual problem is

$$\begin{aligned} \min_{\tilde{y}\in\mathbb{R}^m,\, \tilde{w}\in\mathbb{R}^n,\, \tilde{z}\in\mathbb{R}^n} \quad & \frac{1}{2}\langle \tilde{w}, \tilde{Q}\tilde{w}\rangle + \delta^*_{\tilde{\mathcal{K}}}(-\tilde{y}) + \delta^*_{\tilde{\mathcal{C}}}(-\tilde{z}) \\ \text{s.t.} \quad & -\tilde{Q}\tilde{w} + \tilde{A}^\top\tilde{y} + \tilde{z} = \tilde{c}, \\ & \tilde{w} \in \tilde{W}, \end{aligned}$$

where $\tilde{W} = \text{range}(\tilde{Q})$. The dual variables $\tilde{w} = D_C^{-1}w$, $\tilde{y} = D_R^{-1}y$ and $\tilde{z} = D_C z$. If Q is not stored in explicit matrix form, as in QAP relaxations or LASSO instances, preconditioning is not applied.

Termination criteria. We terminate HPR-QP when the following conditions hold for the unscaled problem with a given tolerance $\varepsilon \in (0, \infty)$

$$\eta_{\text{gap}} = \frac{\left|\frac{1}{2}\langle x, Qx\rangle + \langle c, x\rangle + \phi(x) - \left(-\frac{1}{2}\langle x, Qx\rangle - \delta^*_{\mathcal{K}}(-y) - \phi^*(-z)\right)\right|}{1 + \max\left(\left|\frac{1}{2}\langle x, Qx\rangle + \langle c, x\rangle + \phi(x)\right|, \left|\frac{1}{2}\langle x, Qx\rangle + \delta^*_{\mathcal{K}}(-y) + \phi^*(-z)\right|\right)} \le \varepsilon,$$

$$\eta_{\text{p}} = \frac{\|Ax - \Pi_{\mathcal{K}}(Ax)\|_\infty}{1 + \max\left(\|b\|_\infty, \|Ax\|_\infty\right)} \le \varepsilon, \quad \eta_{\text{d}} = \frac{\|-Qx + A^\top y + z - c\|_\infty}{1 + \max\left(\|c\|_\infty, \|A^\top y\|_\infty, \|Qx\|_\infty\right)} \le \varepsilon.$$

All other solvers are tested with the same tolerances ε. The time limit is set to 3600 seconds, and the tolerance ε is set to 10^{-8} if not specified otherwise.

Shifted Geometric Mean. To evaluate solver performance over multiple problems, we use the shifted geometric mean (SGM), as in Mittelmann benchmarks. For a shift $\Delta = 10$, the SGM10 is defined as $\left(\prod_{i=1}^{n}(t_i + \Delta)\right)^{1/n} - \Delta$, where t_i denotes the solve time (or iteration count) in seconds for the i-th instance. Unsolved instances are assigned a time limit.

Absolute Performance Profile. We also evaluate solver performance using the absolute performance profile $f_s^a : \mathbb{R}_+ \to [0, 1]$, which represents the fraction of problems solved by solver s within time τ. It is defined as $f_s^a(\tau) = \frac{1}{N}\sum_p \mathcal{I}_{\leq\tau}(t_{p,s})$, where $\mathcal{I}_{\leq\tau}(u) = 1$ if $u \leq \tau$, and 0 otherwise.

5.2 Analysis of algorithmic and implementation choices

In this section, we evaluate the impact of several algorithmic and implementation variants of HPR-QP, including:

1. HPR, HDR, and preconditioned ADMM;
2. different penalty parameters and restart frequencies;
3. restricted Wolfe dual, Wolfe dual, and primal formulations;
4. GPU and CPU implementations using either linearization or direct linear system solvers;
5. different update orders of the variables.

We report numerical results where the y-axis corresponds to a residual measure derived from the KKT conditions, defined as

$$\max\left\{\|Ax-\Pi_K(Ax-y)\|_\infty,\ \|-Qw+A^\top y+z-c\|_\infty,\ \|x-\Pi_C(x-z)\|_\infty,\ \|Qx-Qw\|_\infty\right\}.$$

5.2.1 Case study I: Halpern acceleration

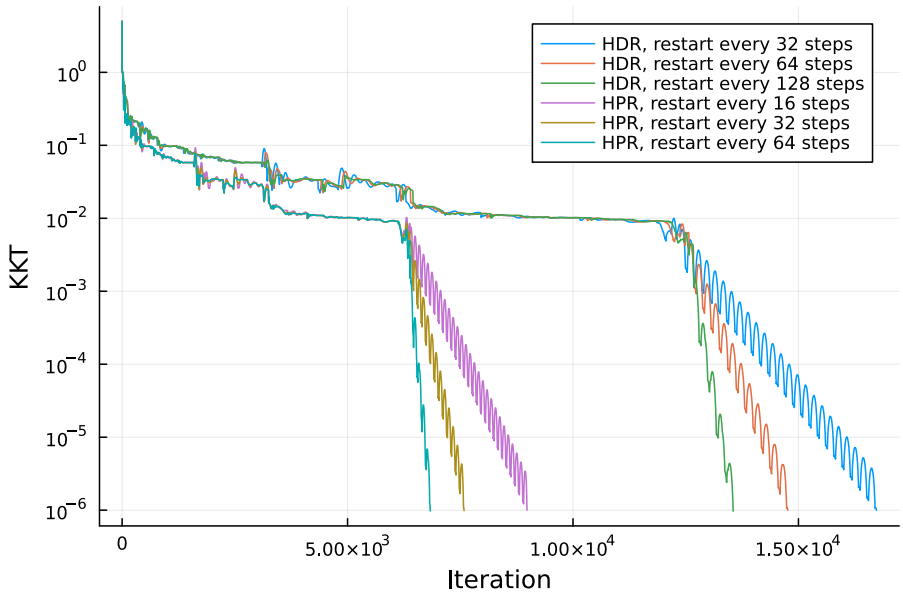
To evaluate the relative performance of the HDR method and the HPR method, we consider the following implementations:

- **HDR method:** based on Algorithm 8, except that Step 12 is modified as

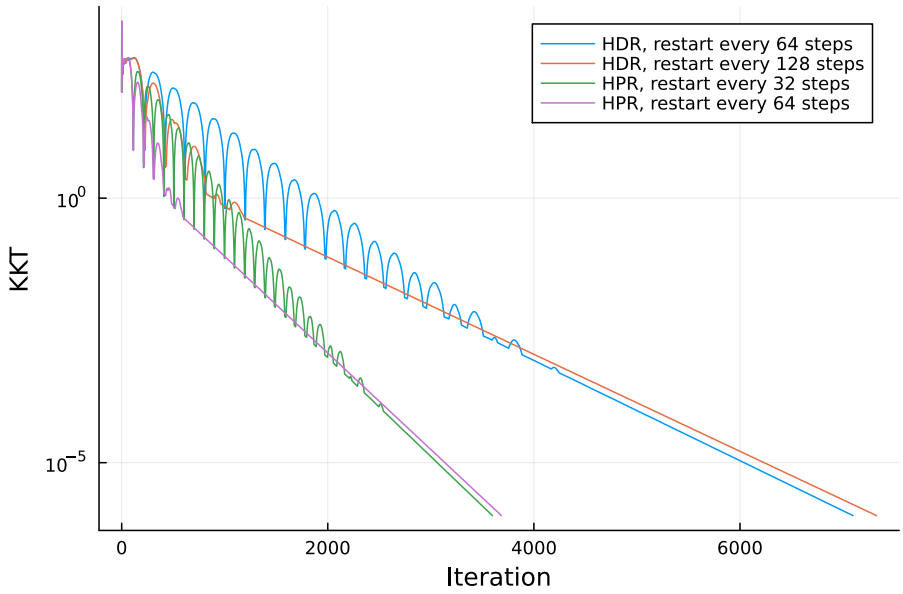
$$u_Q^{r,t+1} = \frac{1}{k+2} u_Q^{r,0} + \frac{k+1}{k+2} \bar{u}_Q^{r,t+1}.$$

- **HPR method:** Algorithm 8, applied without modification.

We implement the HPR method and the HDR method with different restart frequencies across several test instances. Since the optimal restart frequencies for the HPR method and the HDR method typically differ, we report only near-optimal configurations for each method. The penalty parameter is fixed to 1 throughout this experiment. The results are presented in Figure 5.1. For the instances in Figure 5.1a, two phases can be observed: Phase I (slow convergence) and Phase II (fast convergence). Phase I of HPR terminates earlier than that of HDR, and in Phase II, HPR exhibits a faster convergence rate. This advantage of HPR over HDR in Phase II is even more evident in Figure 5.1b.



(a) Instance QSCSD1.

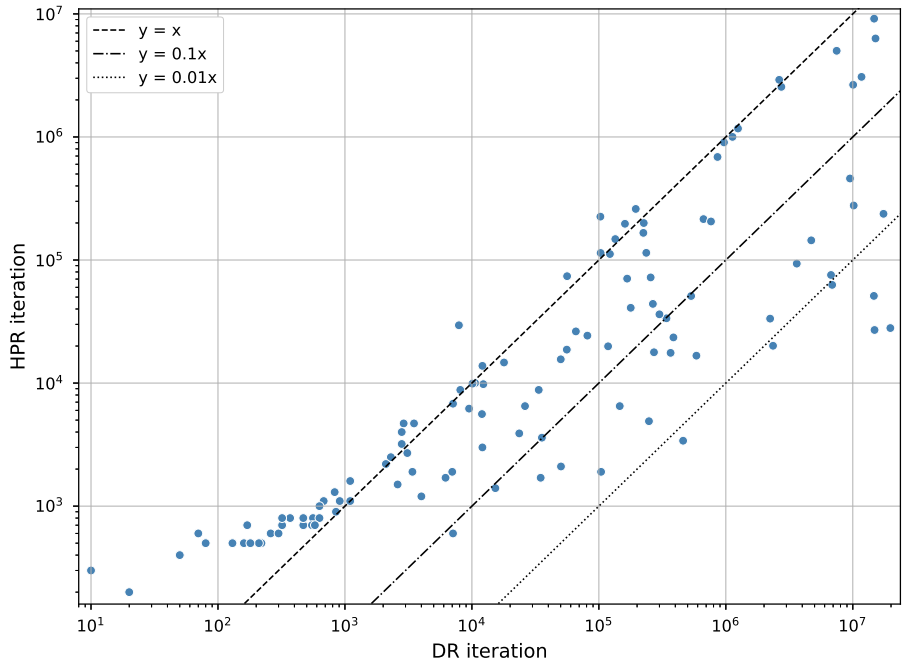


(b) Instance QSTANDAT.

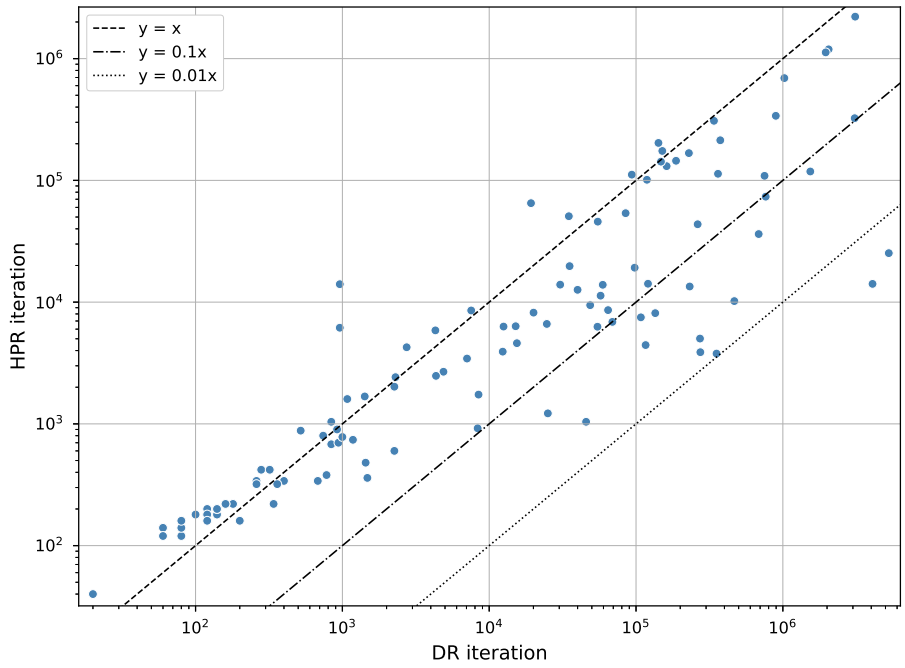
Figure 5.1: KKT residuals of HDR, and HPR with different restart frequencies on two QP problem instances.

Next, we compare the performance of HPR, HDR, and preconditioned ADMM.

The implementation of the preconditioned ADMM corresponds to Algorithm 8 with Line 12 removed and Line 11 replaced by $u^{r,t+1} = \bar{u}_Q^{r,t+1}$. We denote this method as DR, as it corresponds to the linearized Douglas–Rachford scheme. The same restart and penalty parameter update strategies are applied to DR, although the restart mechanism has no practical effect other than providing points for updating the penalty parameter. Figures 5.2a and 5.2b report the iteration counts of HPR and DR on the Maros–Mészáros dataset using the linearization and direct approaches, respectively. As shown in Figure 5.2a, HPR converges in fewer iterations than DR on approximately 60% of the problems, while for the remaining 40% the two methods exhibit comparable iteration counts. Notably, in about 20% of the problems, HPR is more than ten times faster than DR. Figure 5.2b reveals a similar trend for the direct method compared to the linearization case.



(a) Linearization. (x-axis: DR, y-axis: HPR)



(b) Direct. (x-axis: DR, y-axis: HPR)

Figure 5.2: Iteration counts of HPR vs DR with (a) linearization and (b) direct proximal terms on the Maros–Mészáros dataset.

It was shown that the HPR is around 1.7x faster than HDR for the linear programming by Chen et al. (2024a). We present the comparison between HPR and HDR for QP instances from the Maros–Mészáros dataset. Figure 5.3 shows that the HPR converges faster than HDR for most of the problems, and the median of the iteration ratio of HPR to HDR is around 1.8.

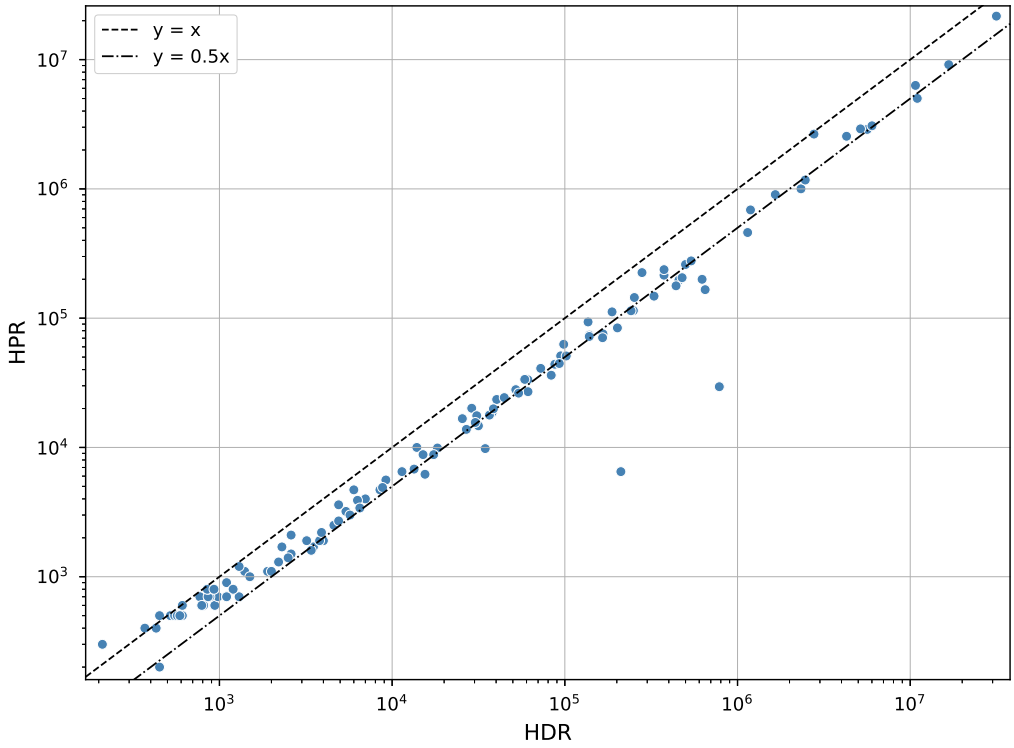
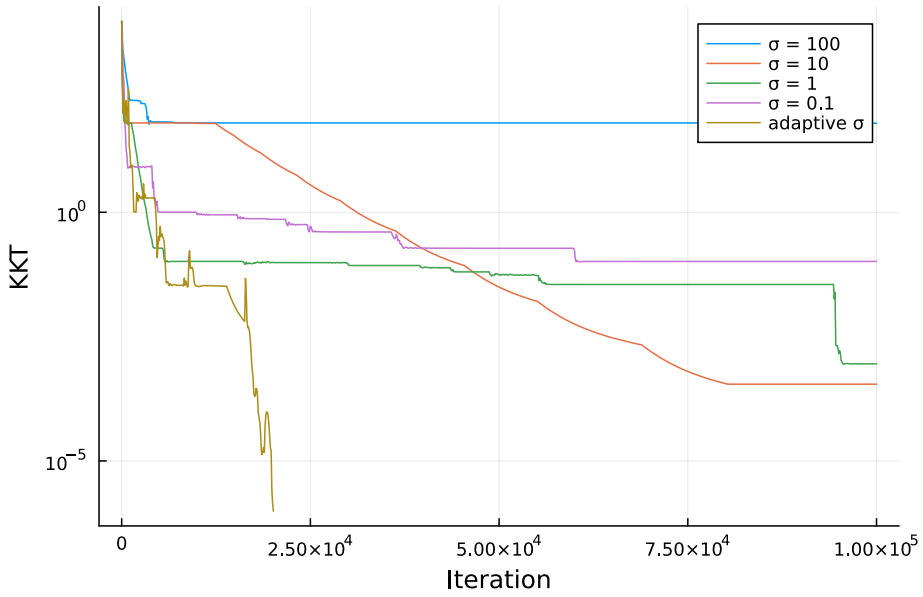


Figure 5.3: Iteration counts of HPR vs HDR on the Maros–Mészáros dataset.

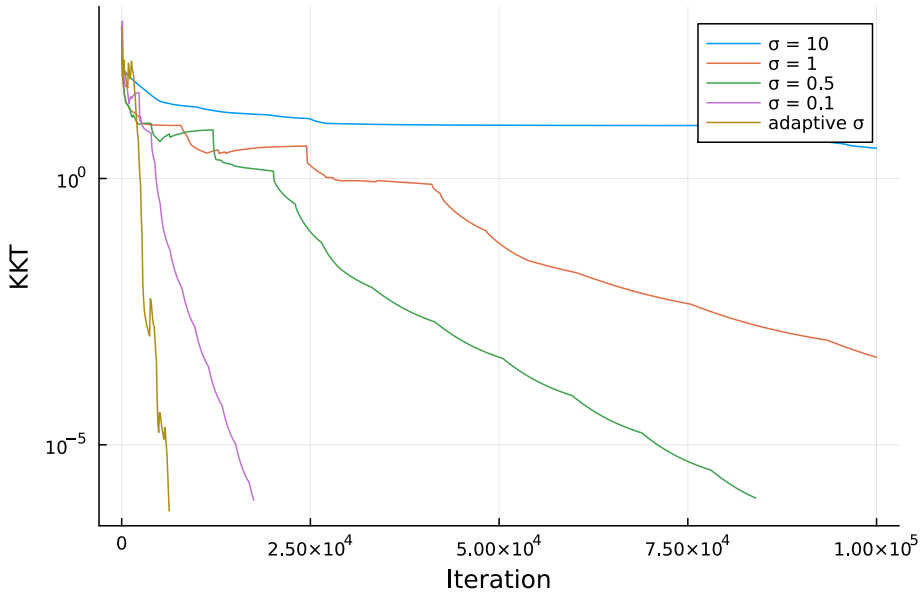
5.2.2 Case study II: Penalty parameter

We evaluate the impact of the penalty parameter on the HPR with fixed and adaptive penalty parameters introduced in Section 4.2.2 with $\sigma_0 = 1$. The adaptive restart introduced in Section 4.2.2 is applied. Figure 5.4 shows the KKT residuals of on two problem instances, QSCAGR7 and QPCSTAIR (Maros–Mészáros dataset). It shows that for HPR with most fixed σ choices has similar phase I that converges slowly as previously discussed. For the case QPCSTAIR, if the σ is chosen properly, the phase

I can be very short. For the case QPCAGR7, the different values of fixed σ cannot affect the phase I significantly except for the case $\sigma = 10$. Such behavior indicates that the choice of σ is crucial for the performance of the algorithm, especially the convergence speed in phase I. With the adaptive penalty parameter updates, the algorithms can potentially escape the slow convergence in phase I and turn to the linear convergence rate.



(a) Instance QSCAGR7.



(b) Instance QPCSTAIR.

Figure 5.4: KKT residuals of HPR with fixed and adaptive penalty parameters on two instances.

5.2.3 Case study III: Restricted Wolfe dual

We will illustrate that solving the restricted Wolfe dual formulation (2.2) is better than the Wolfe dual by two examples. The first example is a QP with a singular Q:

$$Q = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad c = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}, \quad \mathcal{K} = \mathbb{R}^3, \quad \mathcal{C} = (-\infty, 1] \times (-\infty, 1] \times (-\infty, 1], \tag{5.1}$$

where A has no influence since $\mathcal{K}$ is the whole space. We compare Algorithm 8 applied to the restricted Wolfe dual and to the Wolfe dual, using penalty parameter $\sigma = 1$ and near-optimal restart frequencies. For the restricted Wolfe dual, the proximal operator is

$$\mathcal{S}_w = Q\big(\lambda_{\max}(Q) I_n - Q\big) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

For the Wolfe dual, the proximal operator becomes

$$\mathcal{S}'_w = \lambda_{\max}(Q^2 + Q) I_n - (Q^2 + Q) = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 6 \end{pmatrix}.$$

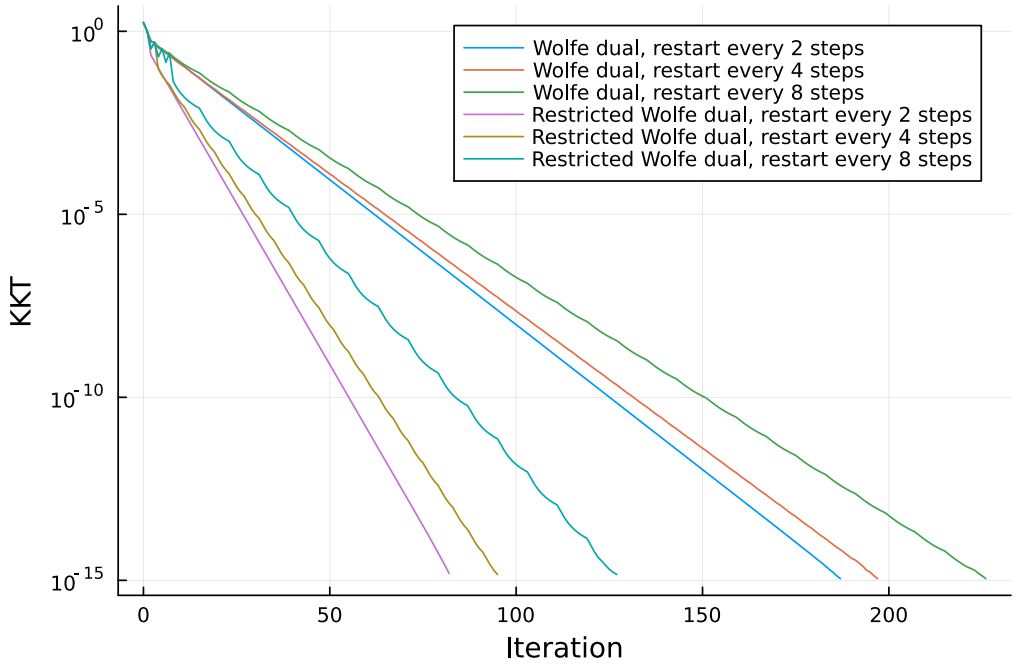
Figure 5.5a shows the KKT residual of the algorithm on the restricted Wolfe dual and Wolfe dual of the example (5.1), respectively. For the restricted Wolfe dual, the accuracy is around 10^{-15} for 80 iterations with a restart every 2 steps, while for the Wolfe dual, it needs 180 iterations with a restart every 2 steps.

The second example is a quadratic program with a strongly convex but ill-conditioned matrix Q:

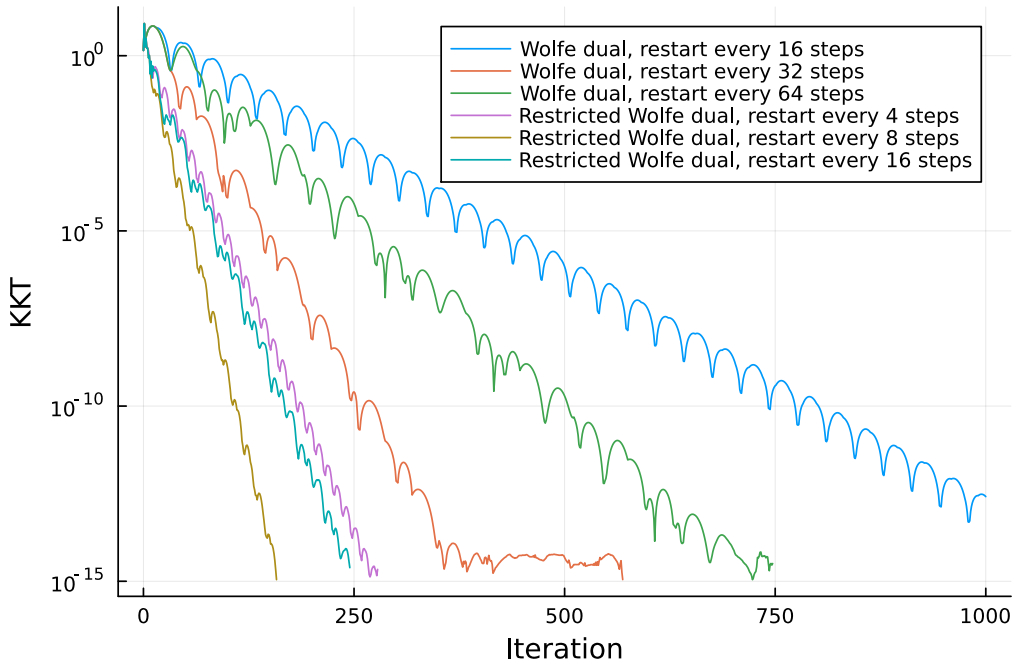
$$Q = \begin{pmatrix} 1 & 0.1 \\ 0.1 & 10 \end{pmatrix}, \quad c = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \quad \mathcal{K} = \mathbb{R}^2, \quad \mathcal{C} = \mathbb{R}^2, \tag{5.2}$$

where A has no influence since $\mathcal{K}$ is the whole space. Figure 5.5b shows the residuals of the algorithm on the restricted Wolfe dual and Wolfe dual of the example (5.2),

respectively. For the restricted Wolfe dual, the accuracy is around 10^{-15} for 150 iterations with a restart every 8 steps, while for the Wolfe dual, it needs 300 iterations with a restart every 32 steps.



(a) Example (5.1).



(b) Example (5.2).

Figure 5.5: KKT residuals of HPR for the Wolfe dual and the restricted Wolfe dual with near-optimal restart frequencies: (a) example (5.1); (b) example (5.2).

To compare the difference between the restricted Wolfe dual and primal formulations, we test the HPR-QP (Algorithm 8), the HPR-QP-primal-1 (Algorithm 3), and the HPR-QP-primal-2 (Algorithm 4). The adaptive restart and penalty parameter selection are implemented in a similar manner for HPR-QP-primal-1 and HPR-QP-primal-2. The iteration numbers of the algorithms are presented in Figure 5.6. It shows that the HPR-QP converges faster than the HPR-QP-primal-1 and HPR-QP-primal-2 for most of the problems. There are around 15% of the problems that HPR-QP is 5x faster than the HPR-QP-primal-1, and around 10% of the problems that HPR-QP is 20x faster. HPR-QP-primal-1 is faster than HPR-QP-primal-2 for most of the problems. Figure 5.7 shows the absolute performance profiles of the three algorithms. It indicates that HPR-QP can solve 80% of instances within 15 seconds, while HPR-QP-primal-1 and HPR-QP-primal-2 need around 50 seconds and 400 seconds, respectively.

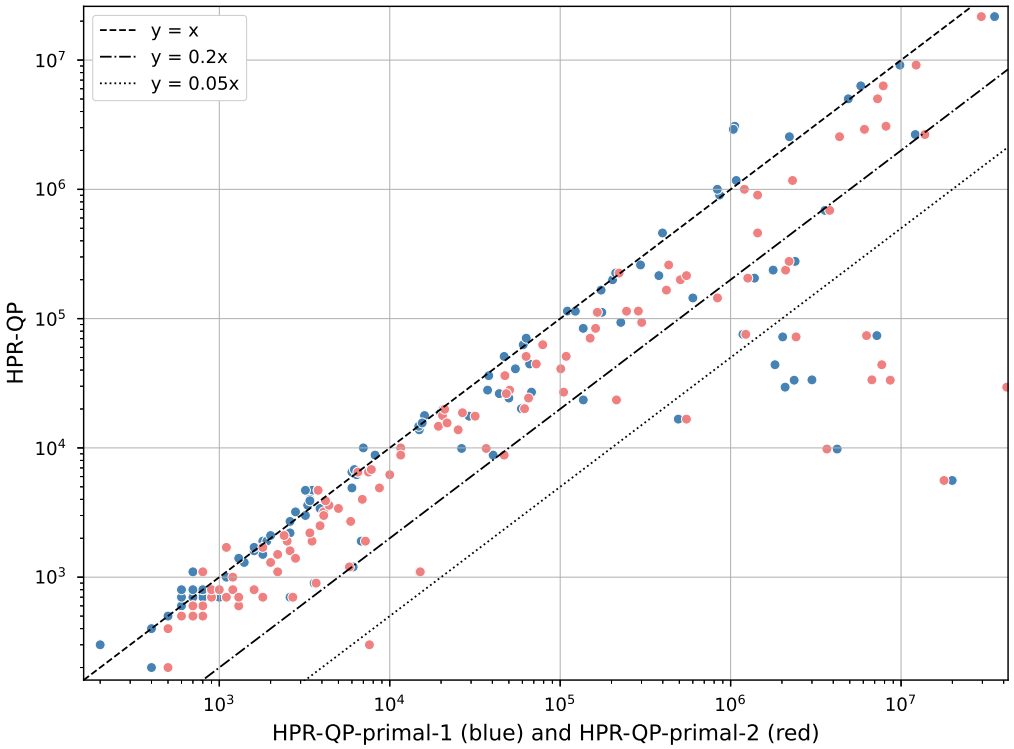


Figure 5.6: Iteration counts of HPR-QP (y-axis), HPR-QP-primal-1 (x-axis of blue points), and HPR-QP-primal-2 (x-axis of red points) on the Maros–Mészáros dataset.

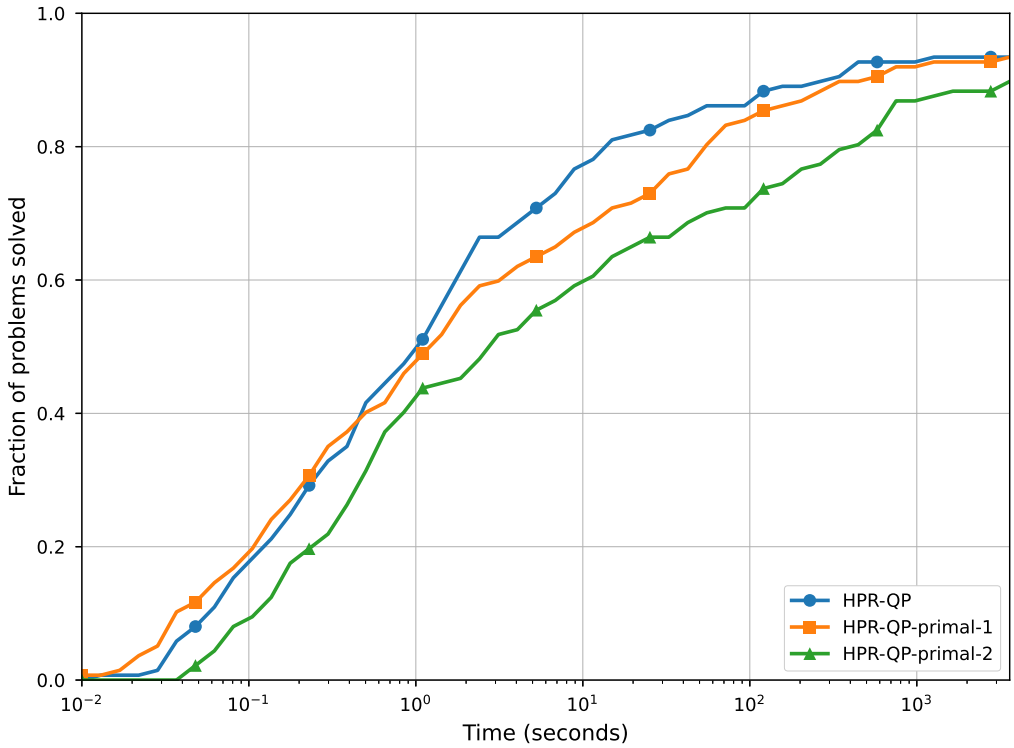


Figure 5.7: Absolute performance profiles of HPR-QP, HPR-QP-primal-1, and HPR-QP-primal-2 on the Maros–Mészáros dataset.

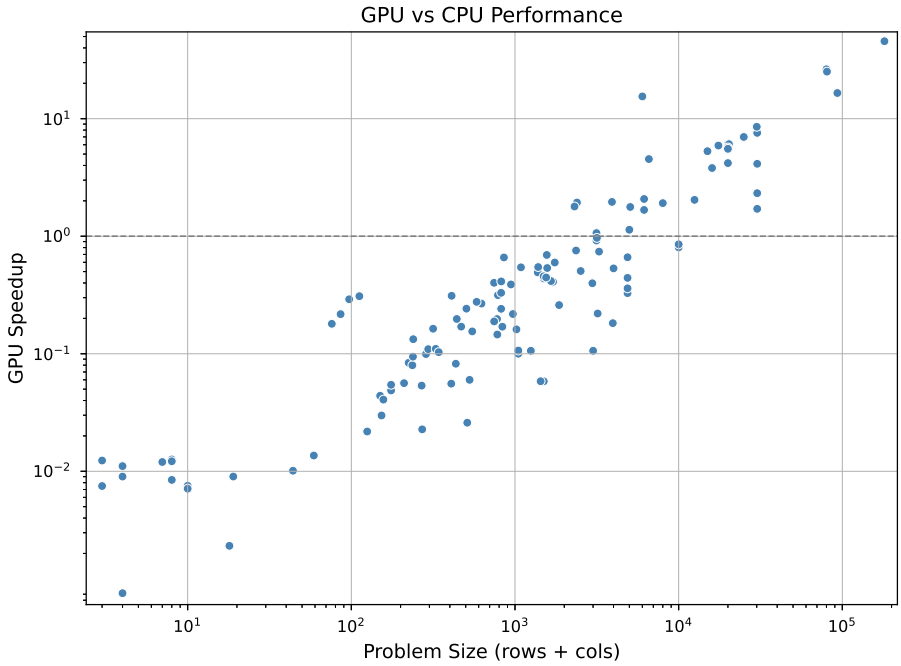
5.2.4 Case study IV: GPU vs CPU

GPU speedup. In this subsection, we evaluate the GPU speedup over CPU by testing HPR-QP (Algorithm 8) with both linearization and direct implementations on two datasets: the Maros–Mészáros QP dataset and the Mittelmann LP dataset.

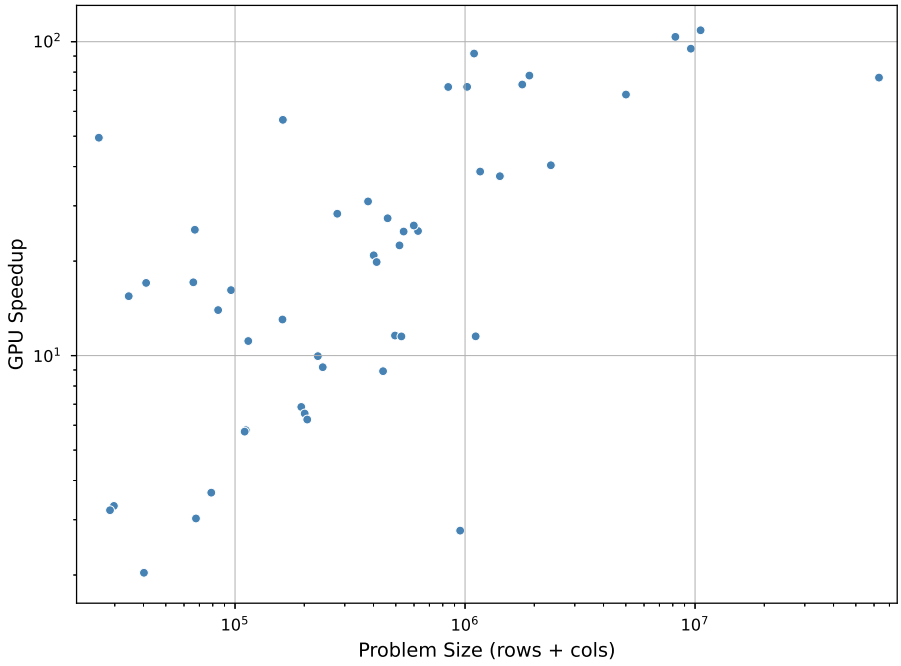
The implementation and number of threads may influence the performance of the algorithm on the CPU, especially for the linearization version. The instances in the Maros–Mészáros dataset are relatively small (dimensions up to the magnitude of 10^5), while the instances in Mittelmann LP dataset are larger (dimensions up to 10^7). For the Maros–Mészáros dataset, we implement the CPU version with the Single Instruction Multiple Data (SIMD) technique for parallel processing. For the Mittelmann LP dataset, we implement the CPU version by the multi-threading

technique with 16 threads, which is the fastest in our experiments.

To evaluate the speedup of GPU compared to CPU, we test the linearization and the direct on Maros–Mészáros QP dataset (Figures 5.8a and 5.9) and the linearization on Mittelmann LP dataset (Figure 5.8b). For the linearization version, we observe that the GPU is slower than the CPU for the instances with problem size less than 10^3, and faster with problem size larger than 10^4, up to 100 times faster with problem size larger than 10^6. For the direct version, the GPU is faster than the CPU for most of the problems with size larger than 10^4, up to 7x faster, while CPU is faster (around 10x on average) for most problems with size smaller than 10^4.



(a) Maros–Mészáros dataset.



(b) Mittelmann LP dataset.

Figure 5.8: GPU speedup of the linearization version HPR-QP on (a) Maros–Mészáros dataset and (b) Mittelmann LP dataset.

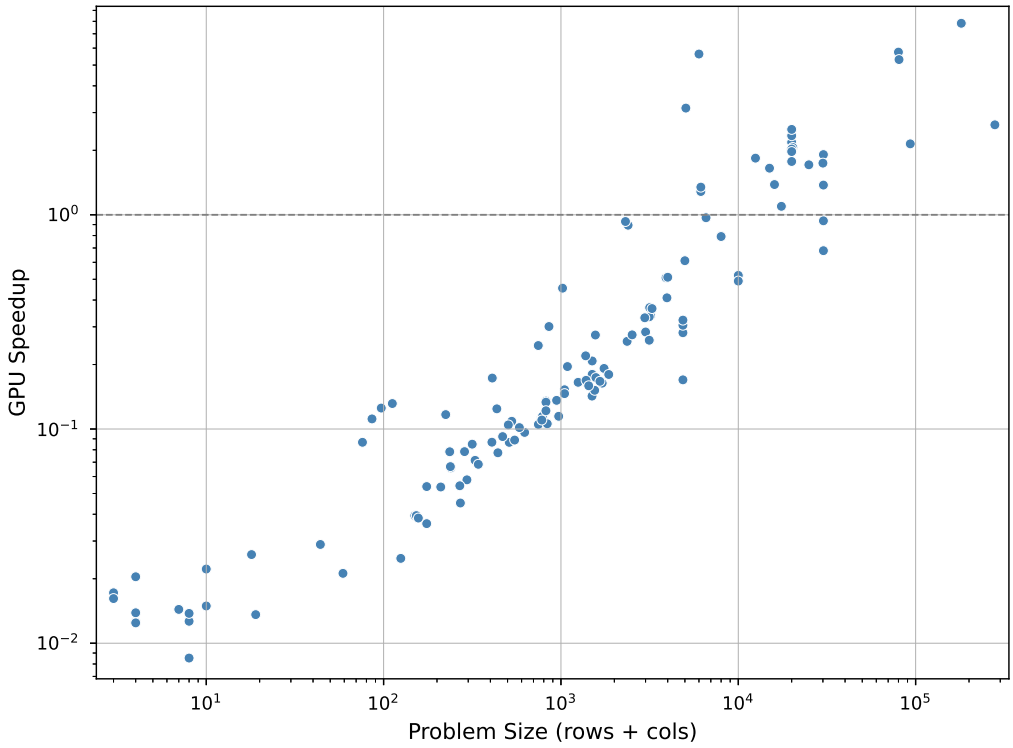
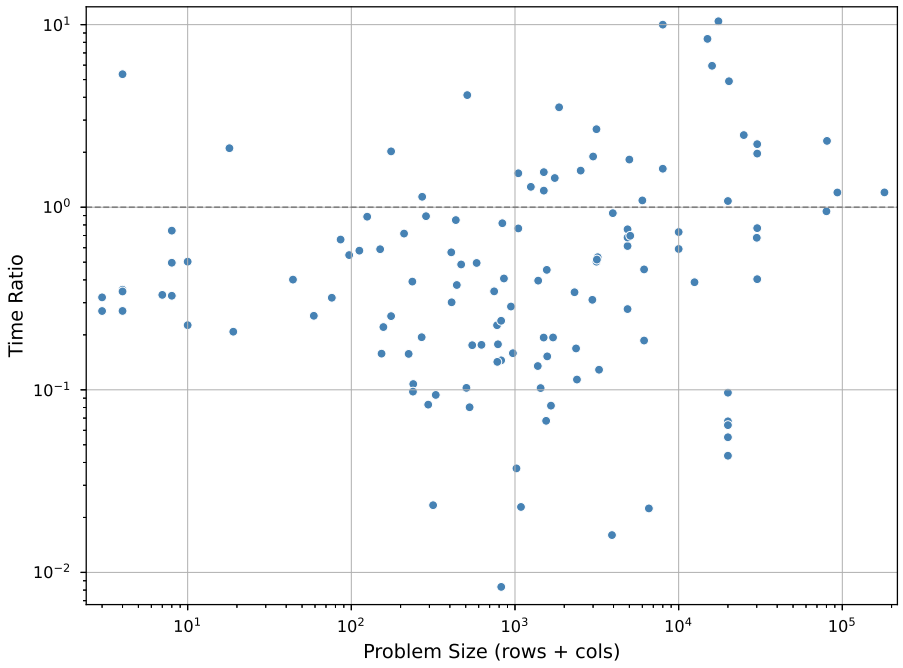
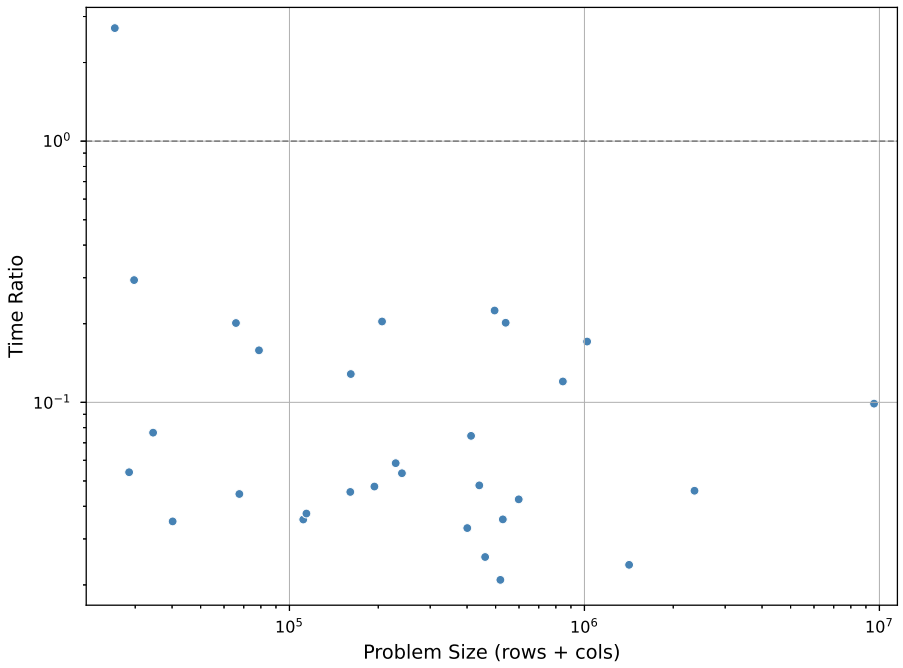


Figure 5.9: GPU speedup of the direct version HPR-QP on Maros–Mészáros dataset.

To assess the impact of the linearization technique versus direct linear system solvers on GPU, we report the runtimes for 30 instances from the Mittelmann LP dataset and 126 instances from the Maros–Mészáros dataset, restricted to those that can be solved within one hour by both methods. In Figure 5.10a, we observe that the linearization is faster for most instances, and it is 2 times faster as a median. Figure 5.10b suggests that the linearization is faster than the direct for most of the problems in Mittelmann LP dataset, up to more than 50 (median is 20) times faster.



(a) Maros–Mészáros dataset.



(b) Mittelmann LP dataset.

Figure 5.10: Time ratio (GPU) of linearization to direct on (a) Maros–Mészáros dataset and (b) Mittelmann LP dataset.

To assess the impact of the linearization technique versus direct linear system solvers using CPU, we show the runtime of 124 instances in the Maros–Mészáros dataset that can be solved within one hour for both methods. Figure 5.11 indicates that the linearization is faster for most instances, and it is 2 times faster as a median.

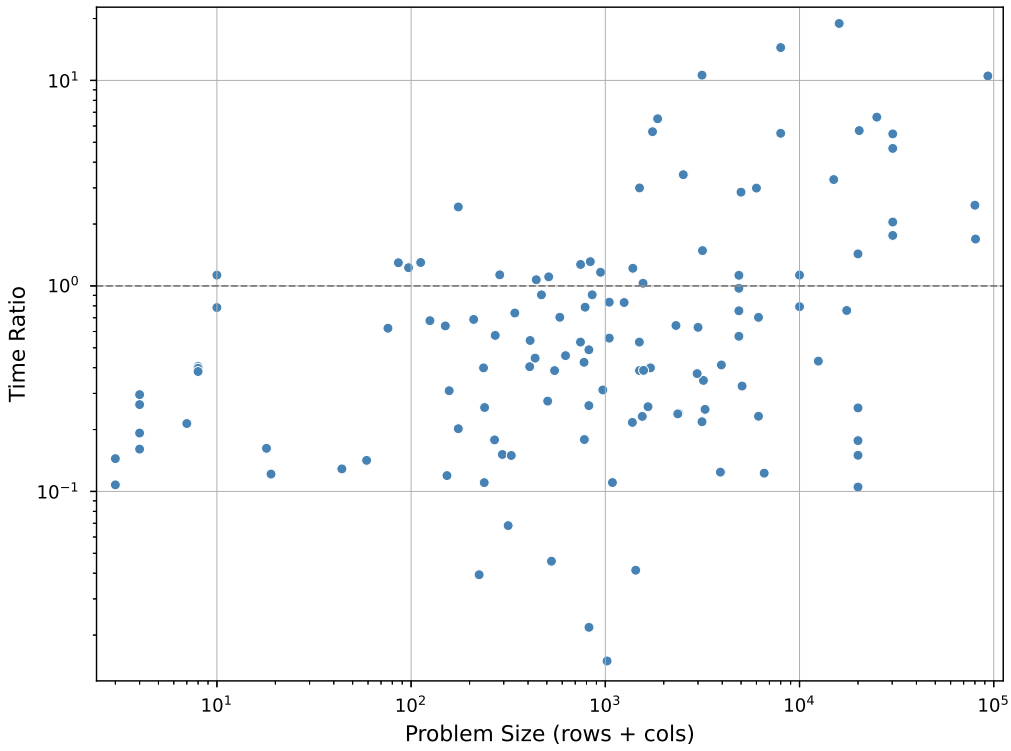


Figure 5.11: Time ratio (CPU) of linearization to direct on Maros–Mészáros dataset.

We list several representative instances (4 from Maros–Mészáros QP dataset and 4 synthetic instances) in Table 5.1, to compare the performance of linearization and direct proximal terms on GPU and CPU. For instances like STADAT2 and control_s, the direct version is faster than the linearization on the GPU, since the iteration numbers of the linearization are hundreds of times more. For instances like equality_s and huber_s, the linearization is faster than the direct version on the GPU, where the iteration numbers of them are similar. We can also observe that the GPU can have a significant (though smaller than linearization) speedup if the problem is large

enough.

Table 5.1: Comparison of linearization and direct: iteration counts, GPU runtime, and CPU runtime. Values in parentheses indicate the iteration counts when reaching 3600 seconds. The first 4 instances are from the Maros–Mészáros dataset, and the last 4 instances are synthetic.

instance	rows	cols	nnz(A)	nnz(Q)	linearization			direct		
					iter	gpu time	cpu time	iter	gpu time	cpu time
BOYD1.mps	18	93,261	558,985	93,261	148,000	20.6	348.2	4,260	17.1	41.0
DTOC3	9,998	14,999	34,993	14,997	199,800	7.6	53.3	3,880	3.1	6.7
EXDATA	3,001	3,000	7,500	2,250,000	73,900	11.7	180.3	8,540	10.7	59.2
STADAT2	3,999	4,001	11,997	2,000	6,315,900	293.6	561.5	53,780	29.4	26.4
control_5	22,000	32,000	60,022,000	4,023,950	250,400	278.3	(29,905) 3600*	1,160	51.1	963.5
equality_3	10,000	20,000	399,621	30,712,896	800	0.7	45.0	180	68.3	322.2
huber_4	5,000,000	15,050,000	16,000,000	5,000,000	800	5.7	487.4	260	791.7	886.0
portfolio_5	801	80,800	32,080,800	80,800	112,500	74.5	(61,687) 3600*	2,020	33.7	421.7

We make a brief conclusion as follows. If the problem is small (up to 10^4), the CPU is faster than the GPU. If the problem scale grows, the linearization with GPU is a robust and efficient choice, especially memory-wise. The above suggestion is just a guideline for the tested dataset, since the performance also depends on the sparsity, the condition number, and the structure of the problem.

5.2.5 Case study V: Variable update orders

Update Orders For the linear programming, Chen et al. (2024a) set the update order of the variables to be z, x, y in the HPR method, due to the complementarity relationship between the variables z and x. Following this idea, by applying the sGS technique to the restricted Wolfe dual formulation in the CCQP, one may update the variables z, x, y, w in the orders $w \to z \to w \to x \to y$ or $z \to x \to w \to y \to w$ (the one discussed in Chapter 4). There are other possible update orders, such as $z \to w \to z \to x \to y$ or $z \to x \to y \to w \to y$, which are not proved to be convergent with the current analysis, since the double-updated variable has a non-smooth objective. Interestingly, these update orders may still be useful in practice, and further research is needed to explore their convergence properties.

We show the performance profile of the update orders $z \to x \to w \to y \to w$ (main used), $w \to z \to w \to x \to y$ (another theoretically convergent), and $z \to w \to z \to x \to y$ (not yet proved convergent) for solving Maros–Mészáros dataset in Figure 5.12. It shows that the main used order $z \to x \to w \to y \to w$ outperforms the $w \to z \to w \to x \to y$, and the not yet proved convergent order $z \to w \to z \to x \to y$ also demonstrates competitive performance.

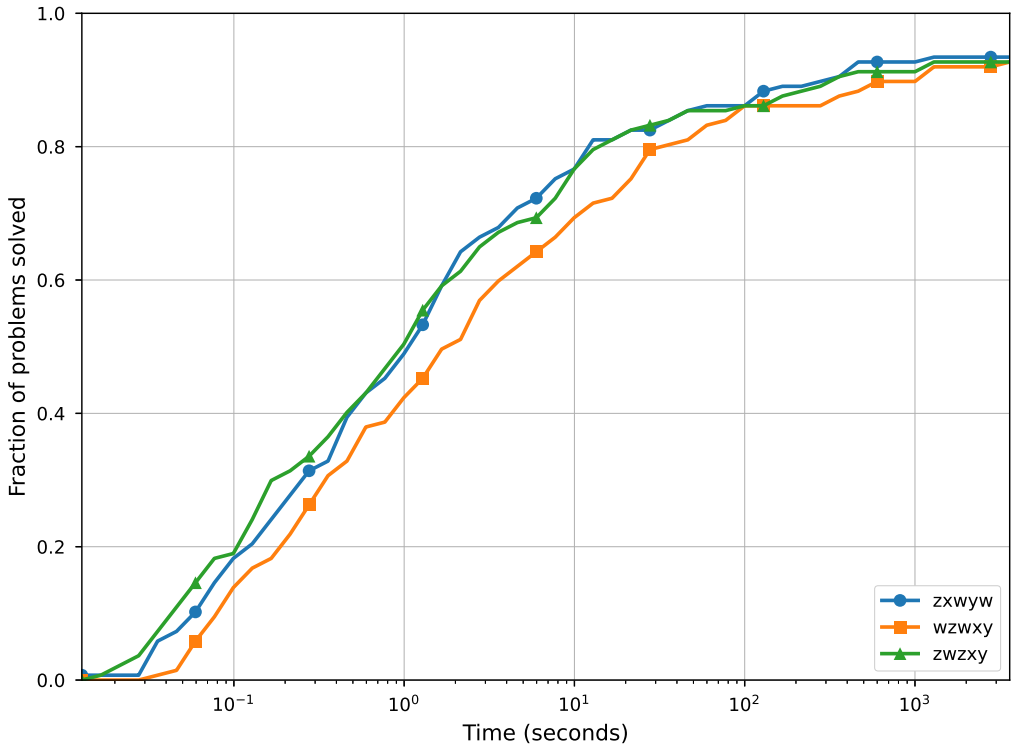


Figure 5.12: Absolute performance profiles of different update orders for HPR on the Maros–Mészáros dataset.

5.3 Comparisons with other solvers

Solvers. We compare to other open-source solvers, including PDQP (Lu and Yang, 2025), an accelerated first-order method based GPU solver for quadratic programming (https://github.com/jinwen-yang/PDQP.jl, downloaded in April 2025); cuPDLP (Lu and Yang, 2023; Applegate et al., 2021), an accelerated first-order method based GPU solver for linear programming (https://github.com/jinwen-yang/cuPDLP.jl, downloaded in July 2024); CuClarabel (Goulart and Chen, 2024; Chen et al., 2024b), an interior point solver for convex optimization including QP, SDP, and conic programming (https://github.com/oxfordcontrol/CuClarabel.jl, version 0.10.0); SCS (O'donoghue et al., 2016; O'Donoghue, 2021), a splitting based conic programming solver (https://github.com/jump-dev/SCS.jl, version 2.1.0); OSQP (Stellato et al., 2020; Schubiger et al., 2020), a split-

ting based solver for convex quadratic programming (https://github.com/osqp/OSQP.jl, version 0.8.1); Gurobi (Gurobi Optimization, LLC, 2025) (version 12.0.2, academic license), a commercial software for (mixed-integer) linear/quadratic programming. A summary of the solvers is provided in Table 5.2. SCS only supports the iterative linear system solver on the GPU, while other computations are performed on the CPU. We only test OSQP on CPU since its GPU version is not robust in our preliminary experiments. For some instances, the objective value of OSQP results is not optimal with the tolerance, which might be due to the lack of the objective gap in its stopping criteria. For QP, Gurobi also supports the simplex method, but we only test the interior point method since it is more efficient. The presolve and crossover techniques in Gurobi are turned off in our experiments. The output status other than optimal will be viewed as failed, and the time will be recorded as 3600 seconds. We turn off the infeasibility detection in SCS, CuClarabel.

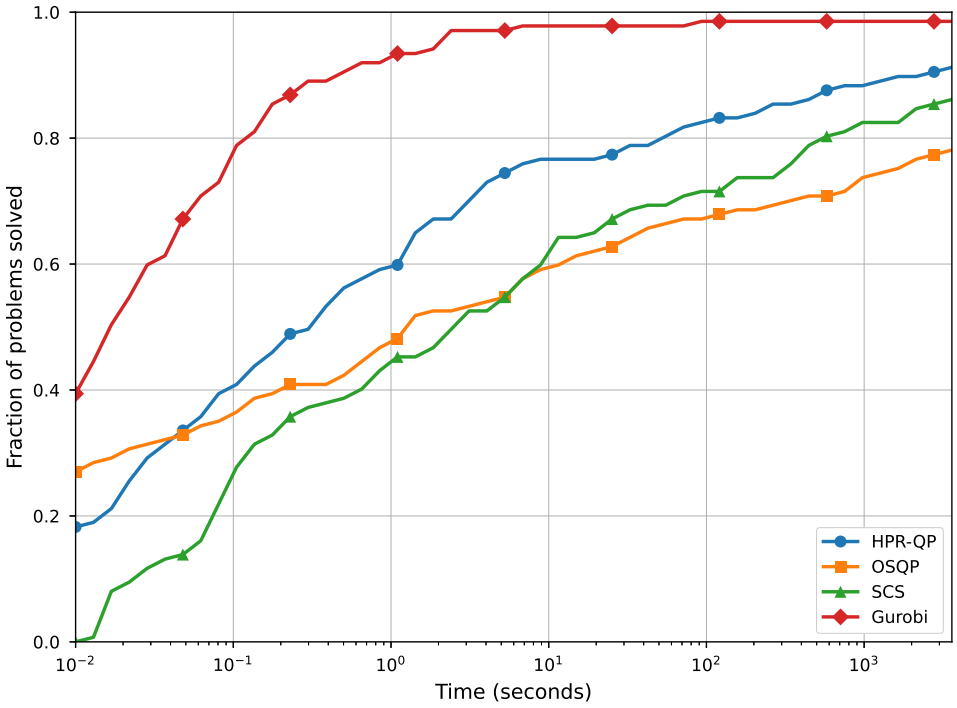
For PDQP, we exclude the time spent on data loading and preconditioning from the measured solve time. In HPR-QP, the time consumed by the power method is counted as part of the solve time, while PDQP does not include it. For SCS and CuClarabel, we record only the solve time and exclude any setup overhead. For Gurobi (Gurobi Optimization, LLC, 2025), we report the solve time as provided by the solver itself.

In the first experiment, we compare the performance of the solvers for the Maros–Mészáros QP dataset on GPU and CPU. In the later experiments, we will focus on the GPU solvers and present Gurobi's performance for reference.

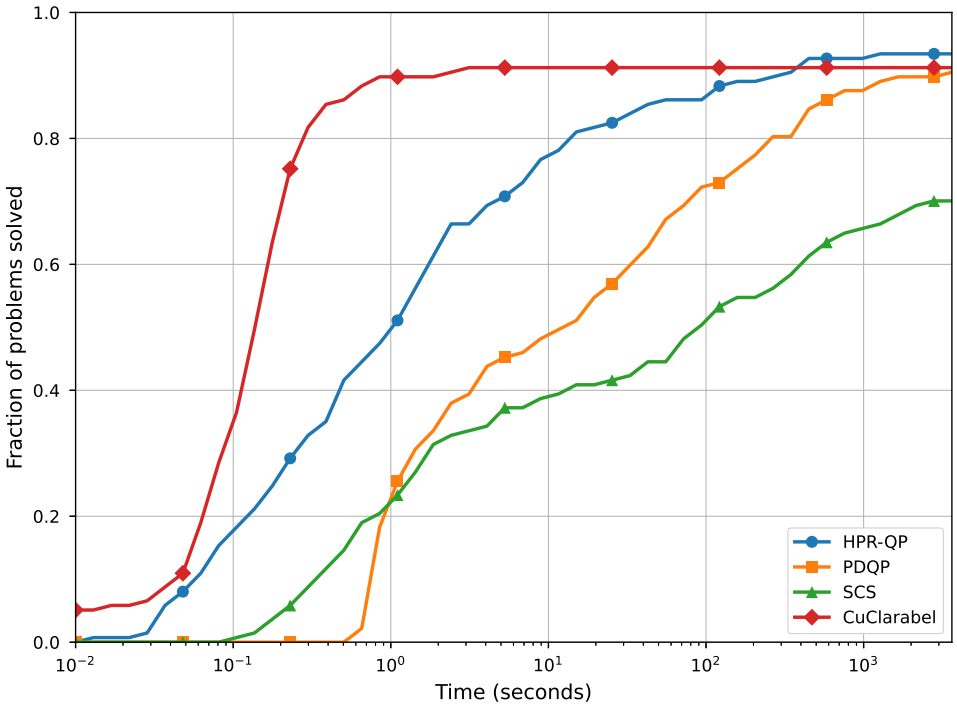
Table 5.2: Summary of the tested solvers.

Solver	**Method**	**Source**	**Hardware**
HPR-QP	ADMM/LADMM	Julia	CPU/GPU
PDQP	LADMM	Julia	GPU
CuClarabel	IPM	Julia/Rust	CPU/GPU
SCS	ADMM	C	Hybrid
OSQP	ADMM	C	CPU/GPU
Gurobi	IPM	C	CPU

We test the solvers, HPR-QP, SCS (direct linear solver), Gurobi, OSQP, on 137 instances of the Maros–Mészáros dataset (excluding the instance VALUES, because Gurobi reports it to be non-convex) on CPU, and the solvers, HPR-QP, SCS (iterative linear solver), CuClarabel, PDQP, on GPU. Figures 5.13a and 5.13b show that HPR-QP is the best among first-order solvers on both CPU and GPU, and the interior point solvers Gurobi and CuClarabel are faster than first order solvers. Table 5.3 summarizes the numerical performance of GPU solvers on the Maros–Mészáros dataset under tolerances 10^{-4}, 10^{-6}, and 10^{-8}. It indicates that for 10^{-8} accuracy, HPR-QP is 3.4x faster than PDQP and 11.6x faster than SCS in terms of SGM10, and it solves more problems than CuClarabel, though with a slightly higher SGM10.



(a) CPU solvers.



(b) GPU solvers.

Figure 5.13: Absolute performance profiles of solvers on the Maros–Mészáros instances.

Table 5.3: Numerical performance of GPU solvers for 137 Maros–Mészáros instances under varying tolerances.

Tolerance	Metric	HPR-QP	PDQP	SCS	CuClarabel
10^{-4}	SGM10	7.4	11.3	32.8	0.3
	Solved	130	133	126	137
10^{-6}	SGM10	10.5	33.1	111.5	3.1
	Solved	129	125	106	131
10^{-8}	SGM10	12.6	42.5	146.6	7.0
	Solved	128	124	96	125

Table 5.4: Benchmark results. GPU solvers (left block) and CPU solvers (right block). Times are in seconds. T = Timeout. Values like "< 0.01" denote less than 10^{-2}.

instance	**GPU solvers**				**CPU solvers**			
	HPR-QP	**SCS (iterative)**	**CuClarabel**	**PDQP**	**HPR-QP**	**SCS (direct)**	**Gurobi**	**OSQP**
AUG2D	0.39	0.99	0.01	1.89	1.63	0.08	0.04	0.04
AUG2DC	0.78	0.86	0.01	1.49	1.80	0.10	0.11	0.07
AUG2DCQP	1.91	76.4	0.19	2.35	3.27	2.45	0.04	0.79
AUG2DQP	0.41	77.4	0.21	3.26	3.08	2.61	0.12	0.81
AUG3D	0.2	0.36	0.02	0.7	0.06	0.07	0.01	0.01
AUG3DC	0.11	0.36	0.01	0.72	0.03	0.02	0.03	0.01
AUG3DCQP	0.17	1.56	0.12	0.8	0.07	0.09	0.01	0.03
AUG3DQP	0.09	1.51	0.14	0.91	0.06	0.12	0.03	0.02
BOYD1	20.64	256	T	397.88	348.20	11.50	0.50	836.94
BOYD2	418.42	T	T	393.56	T	T	5.89	T
CONT-050	3.69	81.1	0.13	16.28	4.19	2.48	0.04	0.97
CONT-100	8.14	906	0.24	47.94	49.60	42.30	0.17	12.41
CONT-101	41.71	1060	0.26	242.35	253.43	76.60	0.17	35.59
CONT-200	46.7	T	0.62	234.24	1228.11	680.00	0.67	827.03
CONT-201	118.94	T	0.66	1268.57	2991.11	2320.00	0.66	736.72
CONT-300	245.68	T	0.72	3013.98	T	T	1.55	T
CVXQP1_L	12.22	115	0.54	54.15	64.65	26.30	F	135.52
CVXQP1_M	0.83	8.31	0.15	5.19	0.36	0.18	0.14	0.19

continued on next page

instance	GPU solvers				CPU solvers			
	HPR-QP	**SCS (iterative)**	**CuClarabel**	**PDQP**	**HPR-QP**	**SCS (direct)**	**Gurobi**	**OSQP**
CVXQP1_S	0.21	4.7	0.06	1.54	0.01	0.04	< 0.01	< 0.01
CVXQP2_L	0.49	14.5	0.27	1.58	1.00	7.86	2.19	6.93
CVXQP2_M	0.37	3.18	0.12	0.94	0.04	0.09	0.08	0.06
CVXQP2_S	0.11	0.55	0.06	0.77	< 0.01	0.02	< 0.01	< 0.01
CVXQP3_L	5.43	319	0.38	37.4	32.06	59.00	43.93	213.33
CVXQP3_M	1.93	63.8	0.21	13.89	1.16	1.51	0.23	0.55
CVXQP3_S	0.22	1.65	0.12	2.44	< 0.01	0.08	0.01	< 0.01
DPKLO1	0.07	0.54	< 0.01	0.79	0.00	0.02	0.00	< 0.01
DTOC3	7.62	102	< 0.01	52.41	53.30	0.13	0.04	0.86
DUAL1	0.08	2.39	0.08	0.86	0.02	0.02	0.01	< 0.01
DUAL2	0.06	1.71	0.07	0.74	0.02	0.01	0.01	< 0.01
DUAL3	0.06	1.52	0.09	0.77	0.02	0.01	0.01	< 0.01
DUAL4	0.04	0.27	0.08	0.75	0.01	0.06	< 0.01	< 0.01
DUALC1	0.11	0.62	0.07	3.47	0.01	0.02	0.02	< 0.01
DUALC2	0.08	1.01	0.07	1.93	0.01	0.02	0.02	< 0.01
DUALC5	0.1	4.78	0.06	1.39	0.01	0.02	0.02	< 0.01
DUALC8	0.61	2.03	0.06	1.28	0.02	0.02	0.03	< 0.01
EXDATA	11.66	26.2	0.37	90.66	180.34	4.29	1.44	6.84
GENHS28	0.23	0.18	< 0.01	0.69	< 0.01	0.07	< 0.01	< 0.01
GOULDQP2	1.53	2110	0.11	5.46	0.15	0.78	0.01	0.08
GOULDQP3	0.18	1.2	0.06	0.9	0.02	0.03	0.01	< 0.01

continued on next page

instance	GPU solvers				CPU solvers			
	HPR-QP	**SCS (iterative)**	**CuClarabel**	**PDQP**	**HPR-QP**	**SCS (direct)**	**Gurobi**	**OSQP**
HS118	0.08	1.48	0.07	0.9	< 0.01	0.01	< 0.01	< 0.01
HS21	0.03	0.49	0.05	0.52	< 0.01	0.09	< 0.01	< 0.01
HS268	1.05	0.6	0.06	410.76	0.01	0.07	< 0.01	< 0.01
HS35	0.03	0.19	0.03	0.71	< 0.01	0.01	< 0.01	< 0.01
HS35MOD	0.03	0.17	0.06	0.71	< 0.01	0.01	< 0.01	< 0.01
HS51	0.03	0.11	< 0.01	0.81	< 0.01	0.05	< 0.01	< 0.01
HS52	0.03	0.17	< 0.01	0.72	< 0.01	0.07	< 0.01	< 0.01
HS53	0.05	0.14	0.04	0.71	< 0.01	0.03	< 0.01	< 0.01
HS76	0.03	0.24	0.03	0.71	< 0.01	0.01	< 0.01	< 0.01
HUES-MC	0.13	0.303	0.2	0.83	0.10	0.09	0.01	0.48
HUESTIS	0.15	0.204	0.06	0.87	0.11	0.06	0.01	4.96
KSIP	0.93	T	0.08	22.66	0.15	482.00	0.02	3.35
LASER	0.5	2.68	0.11	0.77	0.05	0.15	0.03	0.02
LISWET1	T	T	T	T	T	T	0.74	T
LISWET10	T	T	T	T	T	T	0.24	T
LISWET11	T	T	T	T	T	T	2.32	T
LISWET12	T	T	T	T	T	T	0.16	T
LISWET2	T	T	0.13	T	T	T	0.11	2893.91
LISWET3	17.04	T	0.18	183.34	96.34	T	0.11	1244.35
LISWET4	111.28	T	0.2	348.18	632.99	T	0.11	2151.58
LISWET5	42.68	T	0.1	242.85	233.92	2000.00	0.10	806.09

continued on next page

instance	GPU solvers				CPU solvers			
	HPR-QP	**SCS (iterative)**	**CuClarabel**	**PDQP**	**HPR-QP**	**SCS (direct)**	**Gurobi**	**OSQP**
LISWET6	105.33	T	0.15	606.32	573.14	T	0.10	1878.52
LISWET7	T	T	T	T	T	1910.00	0.14	T
LISWET8	T	T	T	T	T	T	1.03	T
LISWET9	T	T	T	T	T	T	2.03	T
LOTSCHD	0.07	1.16	0.06	0.72	< 0.01	0.22	< 0.01	< 0.01
MOSARQP1	0.38	1.36	0.2	1.06	0.08	0.86	0.02	0.12
MOSARQP2	0.45	0.41	0.07	0.72	0.02	0.29	0.01	0.02
POWELL20	7.57	99.3	0.139	84.46	31.70	0.25	0.47	323.75
PRIMAL1	0.05	0.46	0.06	0.95	0.02	0.18	0.01	< 0.01
PRIMAL2	0.05	0.4	0.05	0.78	0.02	0.16	0.01	< 0.01
PRIMAL3	0.06	0.55	0.06	1.21	0.04	0.13	0.02	0.01
PRIMAL4	0.06	0.34	0.08	2.02	0.04	0.07	0.01	0.01
PRIMALC1	0.58	239	0.13	T	0.08	T	< 0.01	T
PRIMALC2	0.28	64.7	0.08	3.68	0.03	1.61	< 0.01	T
PRIMALC5	0.16	0.9	0.07	1.11	0.02	0.55	< 0.01	T
PRIMALC8	0.42	4.49	0.07	1.36	0.03	0.63	< 0.01	T
Q25FV47	32.36	T	0.46	245.64	62.62	279.00	0.15	36.44
QADLITTL	0.24	40.8	0.11	1.28	0.01	3.12	< 0.01	0.01
QAFIRO	0.09	1.39	0.08	1.02	< 0.01	0.36	< 0.01	< 0.01
QBANDM	1.44	1530	0.15	37.6	0.28	66.40	0.01	1.29
QBEACONF	0.79	2580	0.12	59.41	0.06	6.04	0.01	0.10

continued on next page

instance	GPU solvers				CPU solvers			
	HPR-QP	**SCS (iterative)**	**CuClarabel**	**PDQP**	**HPR-QP**	**SCS (direct)**	**Gurobi**	**OSQP**
QBORE3D	8.75	T	0.24	154.75	1.35	442.00	0.01	T
QBRANDY	2.3	437	0.19	24.61	0.39	11.00	0.01	1.41
QCAPRI	30.03	T	0.27	524.47	8.07	T	0.02	55.97
QE226	1.31	T	0.18	43.15	0.32	T	0.01	T
QETAMACR	2.32	T	0.2	29.34	1.26	768.00	0.05	10.08
QFFFFF80	11.26	T	0.31	198.04	5.54	377.00	0.12	11.68
QFORPLAN	12.65	T	0.18	502.31	3.27	541.00	0.01	15.43
QGFRDXPN	7.1	T	0.17	398.83	2.91	824.00	0.02	29.65
QGROW15	1.65	T	0.26	127.94	0.66	325.00	0.01	T
QGROW22	2.19	T	0.3	27.55	1.11	444.00	0.02	1864.26
QGROW7	1.58	T	0.18	83.65	0.34	130.00	< 0.01	350.02
QISRAEL	1.26	531	0.16	30.32	0.21	23.10	0.01	1.29
QPCBLEND	0.53	461	0.11	6.97	0.02	2.13	< 0.01	0.01
QPCBOEI1	0.48	320	0.15	4.02	0.20	5.58	0.01	0.57
QPCBOEI2	5.3	T	0.14	64.76	0.58	302.00	0.01	20.17
QPCSTAIR	0.64	62.3	0.19	4.2	0.15	2.00	0.02	0.11
QPILOTNO	143.46	T	T	T	84.82	T	0.08	T
QPTEST	0.04	0.24	0.04	0.71	< 0.01	0.02	< 0.01	< 0.01
QRECIPE	0.25	1.18	0.12	1.08	0.01	0.07	< 0.01	< 0.01
QSC205	1.26	12.5	0.13	16.1	0.07	0.09	< 0.01	< 0.01
QSCAGR25	1.05	2050	0.16	17.46	0.23	8.15	0.01	T

continued on next page

instance	GPU solvers				CPU solvers			
	HPR-QP	**SCS (iterative)**	**CuClarabel**	**PDQP**	**HPR-QP**	**SCS (direct)**	**Gurobi**	**OSQP**
QSCAGR7	0.92	1390	0.12	36.54	0.05	9.60	< 0.01	T
QSCFXM1	4.18	T	0.23	51.12	1.28	146.00	0.01	T
QSCFXM2	10.61	T	0.29	129.63	4.85	437.00	0.03	53.95
QSCFXM3	12.71	T	0.34	119.68	6.82	1710.00	0.04	101.04
QSCORPIO	2.02	70.8	0.09	29.43	0.59	0.67	0.01	T
QSCRS8	4.07	T	0.33	54.4	1.72	138.00	0.02	1281.18
QSCSD1	0.16	10.5	0.08	1.61	0.03	0.13	0.01	0.11
QSCSD6	0.28	315	0.11	3.85	0.13	7.28	0.01	T
QSCSD8	1.31	68.9	0.1	14.92	1.39	2.24	0.02	0.16
QSCTAP1	0.24	22.4	0.16	3.67	0.04	2.18	0.01	T
QSCTAP2	0.2	5.27	0.15	1.89	0.08	0.50	0.03	T
QSCTAP3	0.73	8.57	0.14	1.86	0.13	0.82	0.04	T
QSEBA	6.01	T	0.28	166.81	2.68	32.10	0.07	5.61
QSHARE1B	3.48	T	0.23	81.64	0.45	5.25	< 0.01	27.23
QSHARE2B	3.46	513	0.11	629.62	0.19	24.80	< 0.01	T
QSHELL	1.68	605	0.54	15.38	3.01	5.48	0.19	1.13
QSHIP04L	1.31	386	0.1	23.31	0.66	10.10	0.01	1.68
QSHIP04S	1.75	40.2	0.1	10.87	0.46	0.83	0.01	0.22
QSHIP08L	2.11	434	0.2	7.27	3.74	19.60	0.09	5.38
QSHIP08S	1.22	122	0.15	8.57	1.18	6.54	0.06	0.55
QSHIP12L	4.49	143	0.21	38.42	20.32	8.91	0.15	2.62

continued on next page

instance	GPU solvers				CPU solvers			
	HPR-QP	**SCS (iterative)**	**CuClarabel**	**PDQP**	**HPR-QP**	**SCS (direct)**	**Gurobi**	**OSQP**
QSHIP12S	1.79	35.6	0.14	17.4	3.50	0.99	0.06	0.69
QSIERRA	0.46	689	T	64.41	0.34	15.10	0.02	1.17
QSTAIR	1.38	390	0.16	9.53	0.46	4.57	0.02	6.13
QSTANDAT	0.38	109	0.14	2.79	0.02	1.01	0.01	0.50
S268	1.06	0.54	0.06	395.58	0.01	0.10	< 0.01	< 0.01
STADAT1	1005.33	T	0.08	T	2187.98	T	F	T
STADAT2	293.6	T	0.18	1006.87	561.55	3140.00	0.03	T
STADAT3	426.29	T	0.23	1567.37	1622.14	T	0.09	T
STCQP1	0.07	0.68	0.11	0.74	0.11	0.12	0.16	0.02
STCQP2	0.16	0.7	0.13	1.27	0.34	0.23	0.25	0.06
TAME	0.01	0.1	0.03	0.64	< 0.01	0.06	< 0.01	< 0.01
UBH1	442.78	T	0.14	T	T	10.10	0.08	6.76
YAO	T	T	T	T	T	T	0.03	T
ZECEVIC2	0.43	0.25	0.04	0.62	< 0.01	0.03	< 0.01	< 0.01

The scale of Maros–Mészáros dataset is relatively small, which may not fully capture the performance of GPU solvers. Following Section 8 of Stellato et al. (2020), we generated 30 large-scale synthetic CQP instances across six problem classes: random QP, equality constrained QP, optimal control, portfolio optimization, and huber fitting. The dimension of the instances can be up to 10^7, and the number of nonzeros in A and Q is up to 10^8. Details can be found in Table 5.6. The performance of GPU solvers and Gurobi is listed in Table 5.5 and visualized in Figure 5.14. HPR-QP successfully solves all instances within one hour. Its SGM10 in runtime is approximately 2.2× and 3.3× faster than CuClarabel and PDQP, respectively. As shown in Figure 5.14, HPR-QP solves about 85% of the instances in approximately 100 seconds, whereas CuClarabel solves about 75% of the instances in the same time frame, and PDQP solves about 60%. These results clearly demonstrate the superior efficiency and scalability of HPR-QP. The performance of CuClarabel compared to Gurobi also partially proves that the interior point method on GPU can have a clear speedup over the CPU version. We list the runtime of the solvers for synthetic instances with 10^{-8} accuracy in Table 5.7.

Table 5.5: Numerical performance of different solvers for 30 synthetic QP instances under varying tolerances.

Tolerance	Metric	HPR-QP	PDQP	SCS	CuClarabel	Gurobi
10^{-4}	SGM10	8.6	42.6	648.9	39.1	230.4
	Solved	30	28	13	25	20
10^{-6}	SGM10	14.3	51.9	781.5	41.4	238.2
	Solved	30	28	13	25	19
10^{-8}	SGM10	19.6	63.8	847.7	43.1	242.8
	Solved	30	27	12	25	19

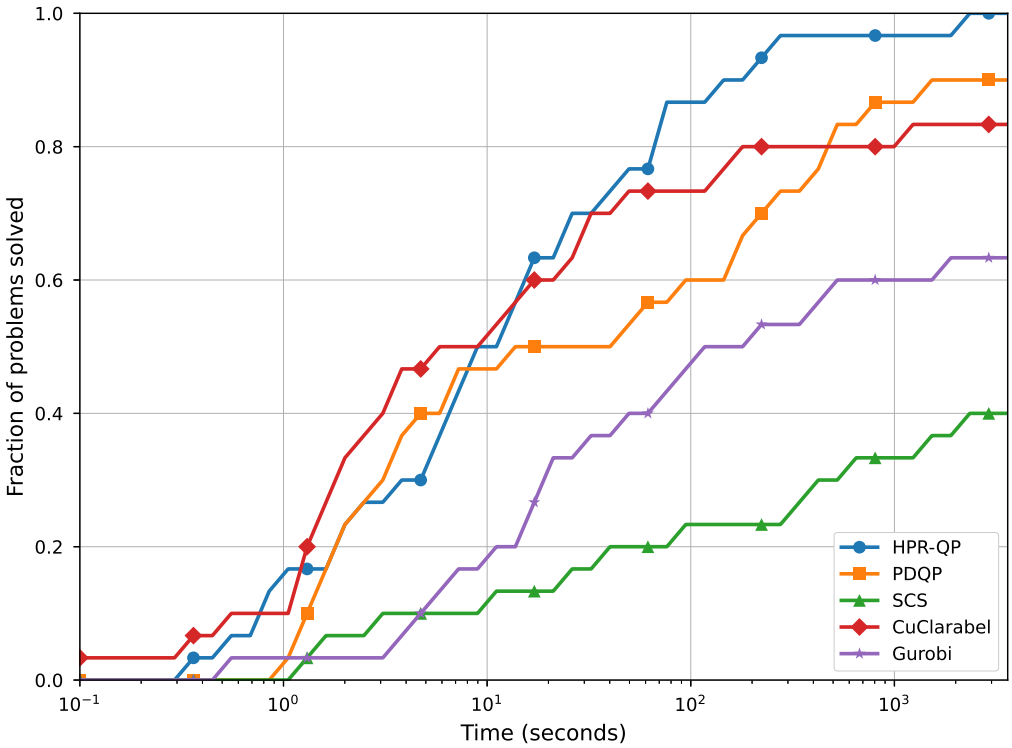


Figure 5.14: Absolute performance profiles of solvers on the synthetic dataset.

Table 5.6: Problem dimensions and sparsity of matrices A and Q in the synthetic CQP instances.

Instance	Rows	Cols	nnz(A)	nnz(Q)
random_1	5.00×10^5	5.00×10^4	1.00×10^6	2.50×10^5
random_2	1.00×10^6	1.00×10^5	2.00×10^6	5.01×10^5
random_3	5.00×10^6	5.00×10^5	1.00×10^7	2.50×10^6
random_4	1.00×10^7	1.00×10^6	2.00×10^7	5.00×10^6
random_5	5.00×10^7	5.00×10^6	1.00×10^8	2.50×10^7
equality_1	5.00×10^3	5.00×10^3	1.99×10^5	6.80×10^6
equality_2	5.00×10^3	1.00×10^4	2.00×10^5	1.47×10^7
equality_3	1.00×10^4	2.00×10^4	4.00×10^5	3.07×10^7
equality_4	2.50×10^4	5.00×10^4	1.00×10^6	7.87×10^7
equality_5	5.00×10^4	1.00×10^5	2.00×10^6	1.59×10^8
control_1	2.20×10^3	3.20×10^3	6.02×10^5	4.24×10^4
control_2	5.50×10^3	8.00×10^3	3.76×10^6	2.56×10^5
control_3	1.10×10^4	1.60×10^4	1.50×10^7	1.01×10^6
control_4	1.65×10^4	2.40×10^4	3.38×10^7	2.27×10^6
control_5	2.20×10^4	3.20×10^4	6.00×10^7	4.02×10^6
portfolio_1	4.01×10^2	4.04×10^4	8.04×10^6	4.04×10^4
portfolio_2	5.01×10^2	5.05×10^4	1.26×10^7	5.05×10^4
portfolio_3	6.01×10^2	6.06×10^4	1.81×10^7	6.06×10^4
portfolio_4	7.01×10^2	7.07×10^4	2.46×10^7	7.07×10^4
portfolio_5	8.01×10^2	8.08×10^4	3.21×10^7	8.08×10^4
huber_1	5.00×10^5	1.51×10^6	1.60×10^6	5.00×10^5
huber_2	1.00×10^6	3.01×10^6	3.20×10^6	1.00×10^6
huber_3	2.00×10^6	6.02×10^6	6.40×10^6	2.00×10^6
huber_4	5.00×10^6	1.51×10^7	1.60×10^7	5.00×10^6
huber_5	1.00×10^7	3.01×10^7	3.20×10^7	1.00×10^7
svm_1	1.00×10^6	1.01×10^6	1.40×10^6	1.00×10^4
svm_2	2.00×10^6	2.02×10^6	2.80×10^6	2.00×10^4
svm_3	5.00×10^6	5.05×10^6	7.00×10^6	5.00×10^4
svm_4	1.00×10^7	1.01×10^7	1.40×10^7	1.00×10^5
svm_5	2.00×10^7	2.02×10^7	2.80×10^7	2.00×10^5

Table 5.7: Comparison of solver runtimes (in seconds) for synthetic QP instances. T = Timeout, M = Memory out.

Instance	**HPR-QP**	**PDQP**	**SCS**	**CuClarabel**	**Gurobi**
random_1	6.5	342.7	T	144.0	T
random_2	11.2	1442.2	T	1010.0	M
random_3	73.7	T	T	M	M
random_4	198.6	T	T	M	M
random_5	2173.4	T	T	M	M
equality_1	0.3	1.3	1.1	0.1	16.6
equality_2	0.5	1.5	1.6	0.3	71.8
equality_3	1.0	1.9	3.0	1.9	449.2
equality_4	8.6	3.4	9.3	32.1	T
equality_5	5.5	5.9	21.7	M	M
control_1	14.5	46.5	32.7	0.5	0.5
control_2	22.4	77.8	84.1	1.9	4.7
control_3	66.1	233.4	367.0	11.9	29.9
control_4	131.5	433.6	577.0	9.1	92.1
control_5	270.5	655.6	1470.0	30.2	215.9
portfolio_1	12.2	56.8	T	1.3	3.6
portfolio_2	22.2	190.3	T	1.5	5.9
portfolio_3	33.3	179.5	T	1.5	9.7
portfolio_4	45.8	179.2	T	2.2	13.9
portfolio_5	72.3	458.3	T	3.2	20.1
huber_1	2.0	2.0	T	1.3	5.6
huber_2	1.6	2.2	T	2.7	17.6
huber_3	3.7	3.1	T	5.0	1732.2
huber_4	7.2	6.5	T	14.0	M
huber_5	14.3	12.1	T	41.8	M
svm_1	0.7	0.9	300.0	1.1	42.6
svm_2	0.8	1.1	2190.0	3.2	116.8
svm_3	1.8	1.6	T	23.9	398.7
svm_4	5.6	2.6	T	158.0	M
svm_5	8.3	4.2	T	M	M

Given matrices $\widehat{A}, \widehat{B} \in \mathbb{S}^d$, a good lower bound for the QAP (Anstreicher and

Brixius, 2001) can be obtained by solving the following CQP:

$$\min_{\mathrm{vec}(X)\in\mathbb{R}^{d^2}} \left\{ \langle \mathrm{vec}(X), \widehat{Q}\,\mathrm{vec}(X)\rangle \;\middle|\; (e^\top \otimes I_d)\,\mathrm{vec}(X) = e,\ (I_d \otimes e^\top)\,\mathrm{vec}(X) = e,\ \mathrm{vec}(X) \geq 0 \right\},$$

where matrix $\widehat{Q} \in \mathbb{S}^{d^2}$ is given by $\widehat{Q} = (\widehat{B} \otimes \widehat{A} - I \otimes S - T \otimes I)$, with $S, T \in \mathbb{S}^d$, which corresponds to the self-adjoint positive semidefinite linear operator $\widehat{\mathcal{Q}}$ defined by $\widehat{\mathcal{Q}}(X) := (\widehat{A}X\widehat{B} - SX - XT)\,\forall X \in \mathbb{R}^{d\times d}$. HPR-QP uses the operator form in practice.

Computation of the matrices S and T. Let $\widehat{A} = V_A D_A V_A^\top$ and $\widehat{B} = V_B D_B V_B^\top$ be the eigenvalue decompositions of $\widehat{A}$ and $\widehat{B}$, where $D_A = \mathrm{diag}(\alpha_1, \ldots, \alpha_d)$, $D_B = \mathrm{diag}(\beta_1, \ldots, \beta_d)$, with $\alpha_1 \geq \cdots \geq \alpha_d$ and $\beta_1 \leq \cdots \leq \beta_d$. Let $(\bar{s}, \bar{t})$ be the solution to the linear program: $\max\{\langle e, s+t\rangle \mid s_i + t_j \leq \alpha_i\beta_j,\ \forall i, j = 1, \ldots, d\}$, which admits a closed-form solution as described in Anstreicher and Brixius (2001). Then, $S = V_A\,\mathrm{diag}(\bar{s})V_A^\top, T = V_B\,\mathrm{diag}(\bar{t})V_B^\top$.

We tested 36 instances with $50 \leq d \leq 256$ from the QAPLIB benchmark set (Burkard et al., 1997) (excluding instance `sko90`, which Gurobi reports as non-convex). The numerical results are summarized in Table 5.8. It shows that HPR-QP significantly outperforms SCS, the second-best first-order solver in this category, achieving a speedup of approximately 18.3× at 10^{-8} tolerance and 6.3× at 10^{-6} tolerance. Table 5.8 demonstrates that HPR-QP outperforms SCS approximately 18.3×, and Gurobi approximately 5.7× at 10^{-8} tolerance. We list the runtime of the solvers for QAP instances from QAPLIB with 10^{-8} accuracy in Table 5.9.

Table 5.8: Numerical performance of different solvers for 36 QAP relaxation instances under varying tolerances.

Tolerance	Metric	HPR-QP	PDQP	SCS	CuClarabel	Gurobi
10^{-4}	SGM10	0.4	1.3	0.5	4.2	22.8
	Solved	36	36	36	36	36
10^{-6}	SGM10	1.8	124.1	11.3	13.6	24.8
	Solved	36	23	36	33	36
10^{-8}	SGM10	4.7	149.4	86.0	114.9	26.8
	Solved	36	23	36	22	36

Table 5.9: Comparison of solver runtimes (in seconds) for QAP instances from QAPLIB. T = Timeout, F = Failure.

Instance	**HPR-QP**	**PDQP**	**SCS**	**CuClarabel**	**Gurobi**
esc128	1.9	0.7	0.3	F	1.5
esc64a	0.1	0.7	0.2	0.2	0.2
lipa50a	0.8	5.8	9.4	0.5	2.7
lipa50b	1.2	5.7	4.5	0.6	2.7
lipa60a	0.9	14.3	17.7	0.8	7.5
lipa60b	1.5	4.3	8.3	0.9	7.6
lipa70a	1.1	5.1	19.6	1.5	18.2
lipa70b	1.3	6.2	10.8	1.4	19.0
lipa80a	1.2	31.6	14.5	2.0	40.3
lipa80b	1.6	12.4	15.7	2.3	40.5
lipa90a	1.0	11.7	26.3	3.0	84.5
lipa90b	1.7	11.4	26.9	3.5	88.4
sko100a	8.1	T	703.0	F	30.7
sko100b	5.8	T	501.0	F	30.2
sko100c	16.0	T	906.0	F	32.8
sko100d	11.4	T	1150.0	F	32.8
sko100e	56.9	T	1510.0	F	34.3
sko100f	30.1	T	761.0	F	32.1
sko56	2.5	754.4	102.0	F	2.1
sko64	8.2	T	249.0	F	4.3
sko72	6.5	T	267.0	F	6.8
sko81	6.6	T	421.0	F	12.1
tai100a	0.5	38.2	14.4	4.9	149.5
tai100b	2.4	145.1	455.0	16.0	204.4
tai150b	1.9	277.2	562.0	152.0	3525.4
tai256c	0.1	2.5	3.0	55.9	108.2
tai50a	0.5	2.5	2.3	0.5	2.6
tai50b	2.0	62.5	125.0	0.8	2.9
tai60a	0.7	2.6	2.7	0.8	7.1
tai60b	2.7	21.2	192.0	1.4	7.6
tai64c	0.1	0.7	0.2	0.2	0.1
tai80a	0.6	3.5	6.6	2.1	41.4
tai80b	2.3	T	257.0	4.4	48.8
tho150	31.8	T	2570.0	F	494.0
wil100	18.1	T	1760.0	F	36.9
wil50	15.7	T	250.0	F	1.6

We generate synthetic QAP instances with size $d \geq 256$ for testing the scalability of HPR-QP, and the advantage of the implicit matrix representation of Q.

Generation of $\widehat{A}$ and $\widehat{B}$ for synthetic QAP. We uniformly sample d points in the unit square $[0,1)^2$, and define the matrix $\widehat{A}$ as their pairwise Euclidean distance matrix, i.e., $\widehat{A}_{ij} = \|(x_i, y_i) - (x_j, y_j)\|$. The matrix $\widehat{B}$ is constructed as a symmetric random matrix with entries drawn uniformly from $[0,1)$ and zero diagonal.

We report the results for HPR-QP in Table 5.10, highlighting its crucial advantage of operating without an explicit matrix form of Q. Notably, HPR-QP successfully solves QAP relaxation instances up to $d = 8192$, demonstrating its scalability and efficiency in matrix-free settings.

Table 5.10: The runtimes of HPR-QP on extremely large-scale QAP relaxations.

d	10^{-4}	10^{-6}	10^{-8}
256	0.8	3.0	6.0
512	1.3	4.0	9.3
1024	3.6	26.3	73.3
2048	73.0	291.0	700.0
4096	32.0	3640.0	9580.7
8192	188.6	61905.6	126547.9

Consider the following LASSO problem:

$$\min_{x \in \mathbb{R}^q} \left\{ \frac{1}{2}\|\widehat{A}x - \widehat{b}\|^2 + \lambda \|x\|_1 \right\}, \tag{5.3}$$

where $\widehat{A} \in \mathbb{R}^{p \times q}$, $\widehat{b} \in \mathbb{R}^p$, and $\lambda > 0$ is the regularization parameter. This problem can be equivalently reformulated as the following CQP:

$$\min_{(x,s,t) \in \mathbb{R}^q \times \mathbb{R}^p \times \mathbb{R}^q} \left\{ \frac{1}{2}\|s\|^2 + \lambda \sum_{i=1}^{q} t_i \;\middle|\; s = \widehat{A}x - \widehat{b},\ -t_i \leq x_i \leq t_i,\ i = 1, \ldots, q \right\}. \tag{5.4}$$

We evaluate performance on 11 LASSO instances from the UCI Machine Learning Repository (as used in Li et al. (2018a)), originally sourced from the LIBSVM

datasets (Chang and Lin, 2011). Table 5.11 summarizes the dimensions and number of nonzeros in $\widehat{A}$. The parameter $\lambda = 10^{-3}\|\widehat{A}^\top \widehat{b}\|_\infty$. We apply HPR-QP directly to the original LASSO problem (5.3) without forming the explicit matrix representation of $Q = \widehat{A}^\top \widehat{A}$, while other solvers solve the equivalent CQP reformulation (5.4). We list the runtime of the solvers for UCI LASSO instances with 10^{-8} accuracy in Table 5.12.

Table 5.11: Dimensions and number of nonzeros in $\widehat{A}$ for UCI LASSO instances.

Instance	p	q	nnz($\widehat{A}$)
abalone7	4 177	6 435	22 873 422
bodyfat7	252	116 280	29 302 560
E2006.test	3 308	150 358	4 559 533
E2006.train	16 087	150 360	19 971 015
housing7	506	77 520	39 225 120
log1p.E2006.test	3 308	4 272 226	22 474 250
log1p.E2006.train	16 087	4 272 227	96 731 839
mpg7	392	3 432	1 174 932
pyrim5	74	201 376	8 054 057
space_ga9	3 107	5 005	15 550 535
triazines4	186	635 376	77 638 169

Table 5.12: Solver runtime comparison (in seconds) for UCI LASSO instances. T = Timeout.

Name	**HPR-QP**	**PDQP**	**SCS**	**CuClarabel**	**Gurobi**
abalone7	10.5	372.5	T	24.4	127.3
bodyfat7	1.2	33.3	T	2.2	30.8
E2006.test	0.2	1.3	T	15.4	9.0
E2006.train	0.7	1.9	T	116.0	277.8
housing7	22.6	123.3	T	5.7	125.9
log1p.E2006.test	7.0	1416.9	T	196.0	137.0
log1p.E2006.train	17.3	2983.2	T	361.0	878.8
mpg7	0.6	18.1	2000.0	0.3	1.2
pyrim5	49.1	410.6	T	3.5	35.9
space_ga9	0.6	62.7	1210.0	6.7	38.1
triazines4	401.3	3533.3	T	26.0	843.1

To further evaluate the scalability of the proposed method, we generated 5 larger

LASSO instances following Appendix A.5 of Stellato et al. (2020) with $p = 2\times10^5, 4\times10^5, 6\times10^5, 8\times10^5, 10^6$ and $q = 10^7$. The parameter $\lambda = \frac{1}{5}\|A^\top b\|_\infty$. Numerical results are summarized in Figure 5.15. HPR-QP consistently outperforms PDQP on these instances and is 5x faster on average. PDQP cannot solve the largest instance due to the memory limit.

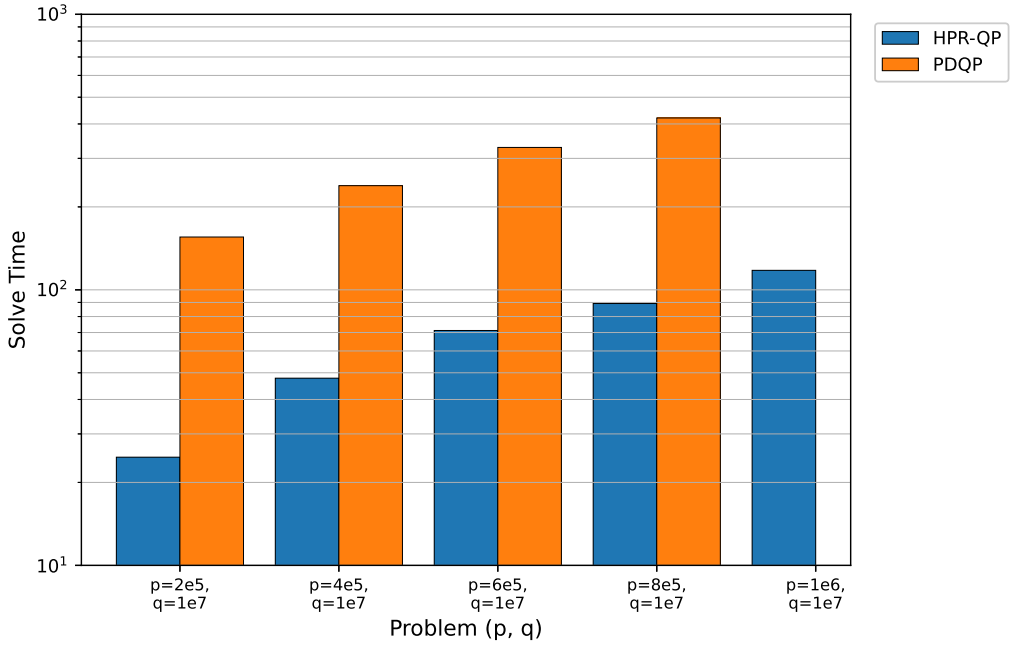


Figure 5.15: The performance of HPR-QP and PDQP on the synthetic LASSO instances.

Chapter 6

Conclusions and future work

6.1 Conclusions

In this thesis, we developed a GPU-oriented solver for convex composite quadratic programming based on the Halpern Peaceman–Rachford (HPR) method. By integrating the restricted Wolfe dual formulation and symmetric Gauss–Seidel decomposition, we established an efficient and robust algorithm with an $O(1/k)$ convergence rate in terms of the KKT residual. Numerical enhancements, including adaptive restart and penalty parameter tuning, were shown to further improve practical performance. On the implementation side, the solver was carefully designed to exploit GPU parallelism without any linear system solver or preconditioner, relying only on matrix-vector multiplications and vector-level operations. This design enables scalability to very large problem instances and applicability to important domains such as quadratic assignment and LASSO regression. Extensive experiments confirmed that the proposed solver consistently outperforms state-of-the-art methods in both speed and scalability, demonstrating its potential as a practical tool for large-scale optimization.

6.2 Future work

Looking ahead, several directions appear promising. First, extending the HPR framework to more general constrained optimization problems, such as quadratically constrained quadratic programming and structured nonconvex formulations, could broaden its scope. Second, incorporating presolve routines and mixed-precision computation may enhance robustness and GPU efficiency. Third, adaptive parameter selection strategies, informed by machine learning or problem-specific heuristics, offer potential for automated solver tuning. At last, integrating the solver into broader optimization toolkits and testing it on large-scale real-world applications in machine learning, finance, control, and combinatorial optimization would both validate its effectiveness and provide avenues for further refinement.

Bibliography

Anstreicher, K. M. and Brixius, N. W. (2001), "A new bound for the quadratic assignment problem based on convex quadratic programming," *Mathematical Programming*, 89, 341–357.

Applegate, D., Díaz, M., Hinder, O., Lu, H., Lubin, M., O'Donoghue, B., and Schudy, W. (2021), "Practical large-scale linear programming using primal-dual hybrid gradient," *Advances in Neural Information Processing Systems*, 34, 20243–20257.

Applegate, D., Hinder, O., Lu, H., and Lubin, M. (2023), "Faster first-order primal-dual methods for linear programming using restarts and sharpness," *Mathematical Programming*, 201, 133–184.

Bambade, A., Schramm, F., El-Kazdadi, S., Caron, S., Taylor, A., and Carpentier, J. (2025), "ProxQP: An efficient and versatile quadratic programming solver for real-time robotics applications and beyond," *IEEE Transactions on Robotics*.

Benidis, K., Feng, Y., and Palomar, D. P. (2017), "Sparse portfolios for high-dimensional financial index tracking," *IEEE Transactions on Signal Processing*, 66, 155–170.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017), "Julia: A fresh approach to numerical computing," *SIAM review*, 59, 65–98.

Borrelli, F., Bemporad, A., and Morari, M. (2017), *Predictive control for linear and hybrid systems*, Cambridge University Press.

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

Bredies, K., Chenchene, E., Lorenz, D. A., and Naldi, E. (2022), "Degenerate preconditioned proximal point algorithms," *SIAM Journal on Optimization*, 32, 2376–2401.

Brodie, J., Daubechies, I., De Mol, C., Giannone, D., and Loris, I. (2009), "Sparse and stable Markowitz portfolios," *Proceedings of the National Academy of Sciences*, 106, 12267–12272.

Bühlmann, P. and Van De Geer, S. (2011), *Statistics for high-dimensional data: methods, theory and applications*, Springer Science & Business Media.

Burkard, R. E., Karisch, S. E., and Rendl, F. (1997), "QAPLIB–a quadratic assignment problem library," *Journal of Global Optimization*, 10, 391–403.

Cela, E. (2013), *The quadratic assignment problem: theory and algorithms*, vol. 1, Springer Science & Business Media.

Chambolle, A. and Pock, T. (2011), "A first-order primal-dual algorithm for convex problems with applications to imaging," *Journal of Mathematical Imaging and Vision*, 40, 120–145.

Chang, C.-C. and Lin, C.-J. (2011), "LIBSVM: A library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, 2, 1–27.

Chen, K., Sun, D., Yuan, Y., Zhang, G., and Zhao, X. (2024a), "HPR-LP: An implementation of an HPR method for solving linear programming," *arXiv preprint arXiv:2408.12179*.

Chen, Y., Tse, D., Nobel, P., Goulart, P., and Boyd, S. (2024b), "CuClarabel: GPU acceleration for a conic optimization solver," *arXiv preprint arXiv:2412.19027*.

Chen, Z., Mi, C. C., Xiong, R., Xu, J., and You, C. (2014), "Energy management of a power-split plug-in hybrid electric vehicle based on genetic algorithm and quadratic programming," *Journal of Power Sources*, 248, 416–426.

Cortes, C. and Vapnik, V. (1995), "Support-vector networks," *Machine Learning*, 20, 273–297.

Cottle, R. W. (1963), "Symmetric dual quadratic programs," *Quarterly of Applied Mathematics*, 21, 237–243.

Cottle, R. W. (1964), "Note on a fundamental theorem in quadratic programming," *Journal of the Society for Industrial and Applied Mathematics*, 12, 663–665.

Cui, Y., Li, X., Sun, D., and Toh, K.-C. (2016), "On the convergence properties of a majorized alternating direction method of multipliers for linearly constrained convex optimization problems with coupled objective functions," *Journal of Optimization Theory and Applications*, 169, 1013–1041.

Dantzig, G. B. (1961), *Quadratic Programming: A Variant of the Wolfe–Markowitz Algorithms*, Operations Research Center, Institute of Engineering Research, University of California, Berkeley.

Davis, D. and Yin, W. (2016), "Convergence rate analysis of several splitting schemes," in *Splitting Methods in Communication, Imaging, Science, and Engineering*, pp. 115–163, Springer.

Donoho, D. L. (2006), "Compressed sensing," *IEEE Transactions on Information Theory*, 52, 1289–1306.

Dorn, W. S. (1960), "Duality in quadratic programming," *Quarterly of Applied Mathematics*, 18, 155–162.

Eckstein, J. and Bertsekas, D. P. (1992), "On the Douglas—Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, 55, 293–318.

Esser, E., Zhang, X., and Chan, T. F. (2010), "A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science," *SIAM Journal on Imaging Sciences*, 3, 1015–1046.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008), "LIBLINEAR: A library for large linear classification," *Journal of Machine Learning Research*, 9, 1871–1874.

Fazel, M., Pong, T. K., Sun, D., and Tseng, P. (2013), "Hankel matrix rank minimization with applications to system identification and realization," *SIAM Journal on Matrix Analysis and Applications*, 34, 946–977.

Fletcher, R. (1971), "A general quadratic programming algorithm," *IMA Journal of Applied Mathematics*, 7, 76–91.

Gabay, D. and Mercier, B. (1976), "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Computers & mathematics with applications*, 2, 17–40.

Garcia, C. E., Prett, D. M., and Morari, M. (1989), "Model predictive control: Theory and practice—A survey," *Automatica*, 25, 335–348.

Gerald, C. F. (2004), *Applied Numerical Analysis*, Pearson Education India.

Gertz, E. M. and Wright, S. J. (2003), "Object-oriented software for quadratic programming," *ACM Transactions on Mathematical Software*, 29, 58–81.

Gill, P., Murray, W., and Wright, M. (1981), *Practical optimization*, Academic Press.

Glowinski, R. and Marroco, A. (1975), "Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de Dirichlet non linéaires," *Revue Française D'automatique, Informatique, Recherche Opérationnelle. Analyse Numérique*, 9, 41–76.

Golub, G. H. and Van Loan, C. F. (2013), *Matrix Computations*, JHU press.

Goulart, P. J. and Chen, Y. (2024), "Clarabel: An interior-point solver for conic programs with quadratic objectives," *arXiv preprint arXiv:2405.12762*.

Gould, N. I. and Toint, P. L. (2000), "A quadratic programming bibliography," *Numerical Analysis Group Internal Report*, 1, 1.

Gould, N. I., Hribar, M. E., and Nocedal, J. (2001), "On the solution of equality constrained quadratic programming problems arising in optimization," *SIAM Journal on Scientific Computing*, 23, 1376–1395.

Gurobi Optimization, LLC (2025), "Gurobi Optimizer Reference Manual," Accessed Aug. 24, 2025.

Halpern, B. (1967), "Fixed points of nonexpanding maps," *Bulletin of the American Mathematical Society*, 73, 957–961.

Han, D., Sun, D., and Zhang, L. (2018), "Linear rate convergence of the alternating direction method of multipliers for convex composite programming," *Mathematics of Operations Research*, 43, 622–637.

Hermans, B., Themelis, A., and Patrinos, P. (2019), "QPALM: a Newton-type proximal augmented Lagrangian method for quadratic programs," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 4325–4330, IEEE.

Hestenes, M. R. (1969), "Multiplier and gradient methods," *Journal of Optimization Theory and Applications*, 4, 303–320.

IBM ILOG CPLEX (2022), "CPLEX Optimizer User's Manual," Accessed Aug. 24, 2025.

Karmarkar, N. (1984), "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of Computing*, pp. 302–311.

Karush, W. (1939), "Minima of functions of several variables with inequalities as side constraints," *M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago*.

Kojima, M., Mizuno, S., and Yoshise, A. (1989), "A primal-dual interior point algorithm for linear programming," in *Progress in Mathematical Programming: Interior-Point and Related Methods*, ed. N. Megiddo, pp. 29–47, Springer.

Koopmans, T. C. and Beckmann, M. (1957), "Assignment problems and the location of economic activities," *Econometrica: journal of the Econometric Society*, pp. 53–76.

Kuhn, H. W. and Tucker, A. W. (1951), "Nonlinear Programming," in *Second Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492.

Lavei, J., Rantzer, A., and Low, S. (2011), "Power flow optimization using positive quadratic programming," *IFAC Proceedings Volumes*, 44, 10481–10486.

Li, M., Sun, D., and Toh, K.-C. (2015), "A convergent 3-block semi-proximal ADMM for convex minimization problems with one strongly convex block," *Asia-Pacific Journal of Operational Research*, 32, 1550024.

Li, X., Sun, D., and Toh, K.-C. (2016), "A Schur complement based semi-proximal ADMM for convex quadratic conic programming and extensions," *Mathematical Programming*, 155, 333–373.

Li, X., Sun, D., and Toh, K.-C. (2018a), "A highly efficient semismooth Newton augmented Lagrangian method for solving Lasso problems," *SIAM Journal on Optimization*, 28, 433–458.

Li, X., Sun, D., and Toh, K.-C. (2018b), "QSDPNAL: A two-phase augmented Lagrangian method for convex quadratic semidefinite programming," *Mathematical Programming Computation*, 10, 703–743.

Li, X., Sun, D., and Toh, K.-C. (2019), "A block symmetric Gauss–Seidel decomposition theorem for convex composite quadratic programming and its applications," *Mathematical Programming*, 175, 395–418.

Liang, L., Li, X., Sun, D., and Toh, K.-C. (2022), "QPPAL: A two-phase proximal augmented Lagrangian method for high-dimensional convex quadratic programming problems," *ACM Transactions on Mathematical Software (TOMS)*, 48, 1–27.

Lieder, F. (2021), "On the convergence rate of the Halpern-iteration," *Optimization Letters*, 15, 405–418.

Lin, Z., Xiong, Z., Ge, D., and Ye, Y. (2025), "PDCS: A primal-dual large-scale conic programming solver with GPU enhancements," *arXiv preprint arXiv:2505.00311*.

Lions, P.-L. and Mercier, B. (1979), "Splitting algorithms for the sum of two nonlinear operators," *SIAM Journal on Numerical Analysis*, 16, 964–979.

Loiola, E. M., De Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., and Querido, T. (2007), "A survey for the quadratic assignment problem," *European Journal of Operational Research*, 176, 657–690.

Lu, H. and Yang, J. (2023), "cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in Julia," *arXiv preprint arXiv:2311.12180*.

Lu, H. and Yang, J. (2025), "A practical and optimal first-order method for large-scale convex quadratic programming," *Mathematical Programming*, pp. 1–38.

Lu, H., Yang, J., Hu, H., Huangfu, Q., Liu, J., Liu, T., Ye, Y., Zhang, C., and Ge, D. (2023), "cuPDLP-C: A strengthened implementation of cuPDLP for linear programming by C language," *arXiv preprint arXiv:2312.14832*.

Markowitz, H. (1952), "Portfolio selection," *Journal of Finance*, 7, 77–91.

Maros, I. and Mészáros, C. (1999), "A repository of convex quadratic programming problems," *Optimization Methods and Software*, 11, 671–681.

Monteiro, R. D. and Svaiter, B. F. (2013), "Iteration-complexity of block-decomposition algorithms and the alternating direction method of multipliers," *SIAM Journal on Optimization*, 23, 475–507.

MOSEK ApS (2025), "MOSEK Optimization Toolbox for MATLAB — Version 11.0.27 Documentation," Accessed Aug. 24, 2025.

Muthukrishnan, R. and Rohini, R. (2016), "LASSO: A feature selection technique in predictive modeling for machine learning," in *2016 IEEE international conference on advances in computer applications (ICACA)*, pp. 18–20, Ieee.

Nesterov, Y. and Nemirovskii, A. (1994), *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM.

Nocedal, J. and Wright, S. J. (2006), *Numerical Optimization*, Springer, 2nd edn.

NVIDIA (2024), "Accelerate large linear programming problems with NVIDIA cuOpt," `https://developer.nvidia.com/blog/accelerate-large-linear-programming-problems-with-nvidia-cuopt/`, Accessed Aug. 25, 2025.

NVIDIA (2025a), "Accelerate decision optimization using open source NVIDIA cuOpt," `https://developer.nvidia.com/blog/accelerate-decision-optimization-using-open-source-nvidia-cuopt/`, Accessed Aug. 25, 2025.

NVIDIA (2025b), "NVIDIA cuOpt User Guide (v25.05)," `https://docs.nvidia.com/cuopt/user-guide/latest/introduction.html`, Accessed Aug. 25, 2025.

O'Donoghue, B. (2021), "Operator splitting for a homogeneous embedding of the linear complementarity problem," *SIAM Journal on Optimization*, 31, 1999–2023.

O'donoghue, B., Chu, E., Parikh, N., and Boyd, S. (2016), "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, 169, 1042–1068.

Pock, T. and Chambolle, A. (2011), "Diagonal preconditioning for first order primal-dual algorithms in convex optimization," in *2011 International Conference on Computer Vision*, pp. 1762–1769, IEEE.

Powell, M. J. (1969), "A method for nonlinear constraints in minimization problems," *Optimization*, pp. 283–298.

Rawlings, J. B., Mayne, D. Q., and Diehl, M. (2020), *Model predictive control: theory, computation, and design*, vol. 2, Madison, WI: Nob Hill Publishing.

Rockafellar, R. T. (1970), *Convex Analysis*, Princeton University Press.

Rockafellar, R. T. (1973), "The multiplier method of Hestenes and Powell applied to convex programming," *Journal of Optimization Theory and Applications*, 12, 555–562.

Rockafellar, R. T. (1976), "Augmented Lagrangians and applications of the proximal point algorithm in convex programming," *Mathematics of Operations Research*, 1, 97–116.

Rockafellar, R. T., Uryasev, S., et al. (2000), "Optimization of conditional value-at-risk," *Journal of risk*, 2, 21–42.

Ruiz, D. (2001), "A scaling algorithm to equilibrate both rows and columns norms in matrices," Tech. rep., Rutherford Appleton Laboratory.

Sabach, S. and Shtern, S. (2017), "A first order method for solving convex bilevel optimization problems," *SIAM Journal on Optimization*, 27, 640–660.

Schubiger, M., Banjac, G., and Lygeros, J. (2020), "GPU acceleration of ADMM for large-scale quadratic programming," *Journal of Parallel and Distributed Computing*, 144, 55–67.

Shin, S., Anitescu, M., and Pacaud, F. (2024), "Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods," *Electric Power Systems Research*, 236, 110651.

Skutella, M. (2001), "Convex quadratic and semidefinite programming relaxations in scheduling," *Journal of the ACM (JACM)*, 48, 206–242.

Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S. (2020), "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, 12, 637–672.

Sun, D. (2020), "Convex quadratic programming: Restricted Wolfe dual and symmetric Gauss–Seidel decomposition theorem," Presentation slides, The Fourth Youth Forum of China Operations Research, Department of Applied Mathematics, Shandong University, June 28, 2020.

Sun, D., Yuan, Y., Zhang, G., and Zhao, X. (2025), "Accelerating preconditioned ADMM via degenerate proximal point mappings," *SIAM Journal on Optimization*, 35, 1165–1193.

Tibshirani, R. (1996), "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B*, 58, 267–288.

Vanderbei, R. J. (1999), "LOQO: An interior point code for quadratic programming," *Optimization methods and software*, 11, 451–484.

Wolfe, P. (1959), "The simplex method for quadratic programming," *Econometrica: Journal of the Econometric Society*, pp. 382–398.

Wolfe, P. (1961), "A duality theorem for non-linear programming," *Quart. Appl. Math.*, 19, 239–244.

Wright, S. J. (1997), *Primal-Dual Interior-Point Methods*, SIAM.

Xiao, Y., Chen, L., and Li, D. (2018), "A generalized alternating direction method of multipliers with semi-proximal terms for convex composite conic programming," *Mathematical Programming Computation*, 10, 533–555.

Zhang, G., Yuan, Y., and Sun, D. (2022), "An efficient HPR algorithm for the Wasserstein barycenter problem with $O(\mathrm{Dim}(\mathrm{P})/\varepsilon)$ computational complexity," *arXiv preprint arXiv:2211.14881*.