A Practical GPU-Enhanced Matrix-Free Primal-Dual Method for Large-Scale Conic Programs

Zhenwei Lin* Zikai Xiong† Dongdong Ge‡ Yinyu Ye§

April 1, 2026

Abstract

In this paper, we introduce a practical GPU-enhanced matrix-free first-order method for solving large-scale conic programming problems, which we refer to as PDCS, standing for the **P**rimal-**D**ual **C**onic Programming **S**olver. Problems that it solves include linear programs, second-order cone programs, convex quadratic programs, and exponential cone programs. The method avoids matrix factorizations and leverages sparse matrix-vector multiplication as its core computational operation, which is both memory-efficient and well-suited for GPU acceleration. The method builds on the restarted primal-dual hybrid gradient method but further incorporates several enhancements. Additionally, it employs a bisection-based method to compute projections onto rescaled cones. Furthermore, cuPDCS is a GPU implementation of PDCS and it implements customized computational schemes that utilize different levels of GPU architecture to handle cones of different types and sizes. Numerical experiments demonstrate that cuPDCS is generally more efficient than state-of-the-art commercial solvers and other first-order methods on large-scale conic program applications, including Fisher market equilibrium problems, Lasso regression, and multi-period portfolio optimization. Furthermore, cuPDCS also exhibits better scalability, efficiency, and robustness compared to other first-order methods on the conic program benchmark dataset CBLIB. These advantages are more pronounced in large-scale, lower-accuracy settings.

Keywords: conic optimization, first-order methods, GPUs, second-order cone program, exponential cone program

1 Introduction

Conic programming seeks to find an optimal solution that minimizes (or maximizes) a linear objective function while remaining within the feasible region defined by the intersection of a linear

*School of Industrial Engineering, Purdue University, West Lafayette, IN 47907, USA. lin2193@purdue.edu

†H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA. zxiong84@gatech.edu (Corresponding author)

‡Antai School of Economics and Management, Shanghai Jiao Tong University, Shanghai, China. ddge@sjtu.edu.cn

§Department of Management Science and Engineering, Stanford University, Stanford, CA 94305, USA. yyye@stanford.edu

subspace and a convex cone. Many important real-world decision-making models can be formulated as conic programs. Typical examples are linear programs (LPs), second-order cone programs (SOCPs), semidefinite programs (SDPs), and exponential cone programs. Conic programs have extensive applications across numerous fields, including economics (see, e.g., [24, 56]), transportation (see, e.g., [10]), energy (see, e.g., [11]), healthcare (see, e.g., [45, 4]), finance (see, e.g., [55]), manufacturing (see, e.g., [7, 28]), computer science (see, e.g., [13]), and medicine (see, e.g., [59]), among many others.

Since the mid-20th century, the development of efficient methods for conic programs has been a central topic in the optimization community, with significant efforts on improving computational speed and scalability. Nearly all existing general-purpose solvers for conic programs are based on either the simplex method (mainly for LPs) or the barrier method (also known as the interior-point method, IPM). These methods are implemented in most commercial optimization solvers.

Despite their effectiveness for solving moderate-sized instances, both the simplex and barrier methods become impractical for large-scale conic programs. The primary bottleneck is their reliance on matrix factorizations, such as LU factorization for the simplex method and Cholesky factorization for the IPM, which are required to repeatedly solve linear systems at each iteration. The computational cost of matrix factorizations grows superlinearly with problem size, measured in terms of the number of decision variables, constraints, or nonzero entries in the problem data. This limitation arises from two main challenges. First, matrix factorizations are highly memory-intensive, and even sparse matrices can produce dense factors, making storage and computation prohibitive. Second, these factorization-based methods are inherently sequential, limiting their suitability for modern parallel and distributed computing architectures, such as graphics processing units (GPUs). Due to these challenges, early efforts to leverage GPUs in commercial solvers have largely been unsuccessful [23].

Meanwhile, outside of conic solver development, GPU-based parallel computing has emerged as a powerful tool for scaling up modern computational applications, most notably in the training of large-scale deep learning models. This contrast highlights the need for alternative conic program solvers that better align with modern high-performance computing architectures.

To better harness the power of GPUs and more efficiently solve large-scale conic programs, a matrix-free first-order method (FOM) is a promising choice. A matrix-free FOM addresses the two challenges mentioned above by eliminating the need for matrix factorizations and instead relying on more computationally efficient operations such as matrix-vector products. These operations are well-suited for GPU acceleration.

In the context of these developments, we develop a practical matrix-free first-order method for large-scale conic programs. We call the method PDCS, standing for "Primal-Dual Conic Programming Solver," because it is based on the primal-dual hybrid gradient method (PDHG). PDCS is designed to solve conic programs where the feasible region consists of Cartesian products of zero cones, nonnegative cones, second-order cones, and exponential cones. We also develop a GPU-enhanced implementation of PDCS that is called cuPDCS. As one might expect, for small-scale problems, the

IPMs in commercial solvers are often better than first-order methods (including PDCS) in efficiency and robustness. However, on the tested instances cuPDCS achieves the following advantages:

1. cuPDCS is generally more efficient than state-of-the-art commercial solvers and other existing first-order methods on large-scale conic program applications.
2. cuPDCS exhibits better scalability, efficiency, and robustness compared to other first-order methods on the small-scale conic program benchmark dataset CBLIB.

Furthermore, the above two advantages are more pronounced in large-scale, lower-accuracy settings. The PDHG algorithm and the associated practical enhancements used in PDCS do not require any matrix factorizations. As a result, the computational bottleneck of PDCS is performing (sparse) matrix-vector products when computing gradients. Thanks to recent advancements in GPU hardware and software optimization, sparse matrix-vector multiplication (SpMV) has been extensively optimized for GPUs, making cuPDCS orders of magnitude faster.

Recently, PDHG has shown promising progress in solving large-scale linear programs and exploiting GPU acceleration. PDHG can be applied to solve the saddle-point formulation of LP [3]. Several large-scale LP solvers have been developed based on PDHG in combination with effective heuristics. Notable implementations include PDLP for CPUs [2] and cuPDLP [41] along with its C-language version cuPDLP-C [43] for GPUs. [41, 43] have shown that GPU-based implementations of PDHG already outperform classical methods implemented in commercial solvers on a significant number of problem instances. As a result, PDHG has been integrated as a new algorithm for LP in COPT [21], Xpress [6], and Gurobi [54]. Additionally, it has been incorporated into Google OR-Tools [2], HiGHS [21], and NVIDIA cuOpt [18].

For general conic programs, [64] have established that PDHG (with restarts) can also solve conic programs beyond LPs. However, no practical PDHG-based method has been developed for conic programs, and it remains unknown whether GPUs can provide similar computational advantages for general-purpose conic programs.

To better unleash the potential of PDHG and leverage GPU acceleration, there are several enhancements implemented in PDCS (and cuPDCS). These enhancements go beyond a straightforward GPU implementation and involve algorithmic improvements, specialized projection methods, and customized parallelism strategies: (a) **Algorithmic improvements**: PDCS is based on a variant of PDHG that integrates several effective enhancements, including adaptive Halpern restarts, diagonal rescaling, and adaptive step-size selection. Some of these heuristics have been proven effective in the PDHG-based LP solver PDLP [2]. (b) **Specialized projection methods**: PDHG requires frequent Euclidean projections onto the underlying cones. While efficient projection methods exist for many common cones, such as nonnegative cones, second-order cones, and exponential cones, projections onto rescaled cones (after applying diagonal rescaling) can become nontrivial. PDCS addresses this by employing bisection-based algorithms. These methods maintain computational complexity comparable to that of projecting onto the original, unrescaled cones. (c) **Customized parallelism strategies**: Unlike LPs, which involve only nonnegative cones, the underlying cones in general conic programs are often more complex, consisting of Cartesian products of multiple cones of

different types and sizes. To better leverage GPU parallelism and accommodate the varying structure of the cones, cuPDCS implements customized computational schemes that utilize different levels of GPU architecture to handle different cones of different types and sizes.

It should be mentioned that PDCS also applies to general semidefinite programming (SDP) problems, but it requires performing a Euclidean projection onto the semidefinite cone in each iteration, whose cost grows superlinearly with respect to the problem dimension. For these SDP problems, recent works [27, 26] propose a less expensive GPU-based first-order solver cuLoRADS that is based on the alternating direction method of multipliers (ADMM) and the Burer-Monteiro method. It does not require projections onto the cone and exhibits better performance than commercial solvers in solving many large-scale SDP problem instances. In contrast, our PDCS focuses on cones other than semidefinite cones.

We conduct computational experiments to evaluate the efficiency, stability, and scalability of cuPDCS. On a benchmark dataset (CBLIB) comprising over $2,000$ small-scale conic optimization instances, our method solves the vast majority of problems, demonstrating better numerical stability compared with other methods. To assess scalability, we conduct experiments on three families of large-scale conic program problems: Fisher market equilibrium, Lasso regression, and multi-period portfolio optimization. The experiments demonstrate that cuPDCS scales more effectively to large-scale instances. Across scenarios involving varying cone counts, cuPDCS consistently produced high-quality solutions within reasonable computational time.

1.1 Related literature

The primal-dual hybrid gradient method (PDHG). The primal-dual hybrid gradient (PDHG) method was introduced in [17, 53] for solving general convex-concave saddle-point problems, of which the saddle-point formulation of conic programs is a special case. PDHG (with restarts) exhibits linear convergence on LPs, and its performance, including linear convergence rates and fast local convergence, has been extensively studied in [3, 29, 62, 63, 39, 60]. [40] introduce a Halpern restart scheme that achieves a complexity improvement of a constant order over the standard average-iteration restart scheme used in PDLP [2]. [64] establish the convergence of restarted PDHG for general conic programs, showing that its performance is closely linked to the local geometry of sublevel sets near optimal solutions. [61] uses probabilistic analysis to establish a high-probability polynomial-time complexity result for the PDHG used on LPs. [42, 31] use PDHG to solve large-scale convex quadratic programs, which can also be reformulated as conic programs.

Other first-order methods for conic programs. Several other first-order methods have been developed for general conic programs. ABIP [36, 14] solves conic programs using an ADMM-based IPM applied to the homogeneous self-dual embedding of the conic program. SCS [51, 50] employs a similar ADMM-based approach to solve the homogeneous self-dual embedding. These ADMM-based methods require solving a linear equation of similar form in each iteration, which is typically handled either through matrix factorization or the conjugate gradient method. However, matrix factorizations suffer from scalability limitations and are not well-suited for parallel computing

on GPUs. The conjugate gradient method, while avoiding explicit factorizations, requires multiple matrix-vector products per iteration, making the choice of tolerance for solving these linear systems a critical heuristic. We will present a comparison between PDCS and these methods in Section 5. Beyond general-purpose conic programs, ADMM has also been used to solve semidefinite programs and quadratic programs (which can be equivalently formulated as SOCPs); see [33] and [57].

GPU acceleration for conic programs. The simplex method and IPMs generally do not benefit much from GPU due to their reliance on solving linear systems [58, 23]. Recently, NVIDIA developed the first GPU-accelerated direct sparse solver (cuDSS) [49]. Based on cuDSS, [12] introduced CuClarabel, a GPU-accelerated IPM. The ADMM-based solver SCS [51, 50] also has a GPU version that leverages the conjugate gradient method for solving linear systems. However, these algorithms still face scalability issues for large-scale problems, as demonstrated in our comparison in Section 5. During the final preparation of this manuscript, we became aware (by private communication) of a concurrent working project by Haihao Lu, Zedong Peng, and Jinwen Yang that proposes a GPU-implemented PDHG-based solver to solve large-scale SOCPs.

1.2 Outline

The paper is organized as follows. Section 1.3 introduces the notation used throughout the paper. Section 2 provides the conic program formulations considered in this paper and the PDHG iterations used as the base algorithm. Section 3 presents the practical enhancements used in PDCS. Section 4 describes the customized computational schemes that leverage different levels of GPU architecture to efficiently handle various cone types. Finally, Section 5 presents numerical experiments comparing PDCS against other first-order methods and commercial solvers. Omitted proofs and additional details on experiments are provided in the Appendix.

1.3 Notation

Throughout the paper, we use the following notations. Let $[n] := \{1, \dots, n\}$ for integer n. We use bold letters like $\mathbf{v}$ to represent vectors and bold capital letters like $\mathbf{A}$ to represent matrices. In general, we use subscripts to denote the coordinates of a vector without additional clarification. For matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, $\|\mathbf{M}\|_\infty$ denotes $\max_{i \in [m]} \sum_{j \in [n]} |\mathbf{M}_{ij}|$. Denote the identity matrix as $\mathbf{I}$. Let $\mathbf{v}^+ = [(\mathbf{v}_1^+, \dots, \mathbf{v}_n^+)]$, where $\mathbf{v}_i$ is the i-th entry of $\mathbf{v}$ and $\mathbf{v}_i^+ = \max\{\mathbf{v}_i, 0\}$ is the coordinate-wise positive part of a vector. Similar notation for the coordinate-wise negative part $\mathbf{v}^-$ is defined by using $\mathbf{v}_i^- = -\min\{\mathbf{v}_i, 0\}$. We denote the scaled cone as $\mathbf{D}\mathcal{K}$, which implies that $\mathbf{v} \in \mathbf{D}\mathcal{K}$ is equivalent to $\mathbf{D}^{-1}\mathbf{v} \in \mathcal{K}$. We write $\text{proj}_{\mathcal{S}}(\mathbf{x})$ for the Euclidean projection of $\mathbf{x}$ onto $\mathcal{S}$. Finally, we use $\text{diag}(x_1, \cdots, x_n)$ to denote the diagonal matrix with diagonal elements $x_1, \cdots, x_n$.

We use the following notations for commonly used cones: $\mathbf{0}^d$ represents the zero cone, $\mathbb{R}_+^d$ represents the non-negative cone, $\mathcal{K}_{\text{soc}}^{d+1} = \left\{(t, \mathbf{x}) \mid \mathbf{x} \in \mathbb{R}^d, t \in \mathbb{R}, \|\mathbf{x}\| \le t\right\}$ denotes the second-order cone, $\mathcal{K}_{\text{rsoc}}^{d+2} = \{(x, y, \mathbf{z}) \mid x, y \in \mathbb{R}_+, \mathbf{z} \in \mathbb{R}^d, \|\mathbf{z}\|^2 \le 2xy\}$ denotes the rotated second-order cone, $\mathcal{K}_{\text{exp}} := \left\{(r, s, t) \in \mathbb{R}^3 \mid s > 0, t \ge s \cdot \exp\left(\frac{r}{s}\right)\right\} \cup \{(r, s, t) \in \mathbb{R}^3 \mid s = 0, t \ge 0, r \le 0\}$ denotes the exponential cone, and $\mathbb{S}_+^{d \times d}$ denotes the semidefinite cone, defined as $\mathbb{S}_+^{d \times d} =$

$\left\{\mathbf{A} \in \mathbb{R}^{d\times d} : \mathbf{A} = \mathbf{A}^\top, \mathbf{x}^\top \mathbf{A}\mathbf{x} \geq 0, \forall \mathbf{x} \in \mathbb{R}^d \backslash \{\mathbf{0}\}\right\}$. The dual cone $\mathcal{K}^*$ of $\mathcal{K}$ is the set of non-negative dot products of $\mathbf{y} \in \mathbb{R}^d$ and $\mathbf{x} \in \mathcal{K}$, which is defined as $\mathcal{K}^* = \{\mathbf{y} \in \mathbb{R}^d : \langle \mathbf{y}, \mathbf{x}\rangle \geq 0, \forall \mathbf{x} \in \mathcal{K}\}$.

2 Preliminaries of Conic Optimization and PDHG

In this section, we present the formulation of the conic optimization problems (conic programs), along with their corresponding dual problems and equivalent saddle-point formulations. In Section 2.1, we outline the iterations of the basic PDHG method for these conic programs.

We consider the following conic program:

$$\min_{\mathbf{x}=(\mathbf{x}_1,\mathbf{x}_2):\mathbf{x}_1\in\mathbb{R}^{n_1},\mathbf{x}_2\in\mathbb{R}^{n_2}} \langle \mathbf{c}, \mathbf{x}\rangle \quad \text{s.t. } \mathbf{G}\mathbf{x} - \mathbf{h} \in \mathcal{K}_{\mathrm{d}}^* \ , \ \mathbf{l} \leq \mathbf{x}_1 \leq \mathbf{u} \ , \ \mathbf{x}_2 \in \mathcal{K}_{\mathrm{p}} \ , \tag{1}$$

where $\mathbf{G} \in \mathbb{R}^{m\times n}$ is the constraint matrix, and $\mathbf{l}$ and $\mathbf{u}$ are elementwise lower and upper bounds on $\mathbf{x}_1$. The set $\mathcal{K}_{\mathrm{p}}$ denotes the primal cone, which is a Cartesian product of nonempty closed smaller cones: $\mathcal{K}_1 \times \mathcal{K}_2 \times \ldots \times \mathcal{K}_{l_{\mathrm{p}}}$, where l_{p} denotes the number of the smaller cones. These smaller cones include the zero cone $\mathbf{0}^d$, the nonnegative orthant $\mathbb{R}^d_+$, the second-order cone $\mathcal{K}^{d+1}_{\mathrm{soc}}$, the exponential cone $\mathcal{K}_{\mathrm{exp}}$, the rotated second-order cone $\mathcal{K}^{d+2}_{\mathrm{rsoc}}$ and the dual exponential cone $\mathcal{K}^*_{\mathrm{exp}}$, and we call them *disciplined cones*. Our algorithm also applies to the semidefinite cone ($\mathbb{S}^{d\times d}_+$). Here, $\mathcal{K}^*_{\mathrm{d}}$ denotes the dual cone of $\mathcal{K}_{\mathrm{d}}$. We use $\mathcal{K}^*_{\mathrm{d}}$ in the primal so that the dual feasibility condition can be written using $\mathcal{K}_{\mathrm{d}}$ as follows:

$$\max_{\substack{\mathbf{y},\ \boldsymbol{\lambda}=(\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2):\\ \mathbf{y}\in\mathbb{R}^m,\ \boldsymbol{\lambda}_1\in\mathbb{R}^{n_1},\ \boldsymbol{\lambda}_2\in\mathbb{R}^{n_2}}} \langle \mathbf{y}, \mathbf{h}\rangle + \mathbf{l}^\top \boldsymbol{\lambda}_1^+ - \mathbf{u}^\top \boldsymbol{\lambda}_1^- \ , \quad \text{s.t. } \mathbf{c} - \mathbf{G}^\top \mathbf{y} = \boldsymbol{\lambda} \ , \ \mathbf{y} \in \mathcal{K}_{\mathrm{d}}, \ \boldsymbol{\lambda}_1 \in \Lambda \ , \ \boldsymbol{\lambda}_2 \in \mathcal{K}^*_{\mathrm{p}} \ , \tag{2}$$

$$\text{where } \Lambda := \Lambda_1 \times \Lambda_2 \times \cdots \times \Lambda_{n_1} \quad \text{and} \quad \Lambda_i := \begin{cases} \{0\}, & \text{if } \mathbf{l}_i = -\infty, \mathbf{u}_i = +\infty \\ \mathbb{R}^-, & \text{if } \mathbf{l}_i = -\infty, \mathbf{u}_i \in \mathbb{R} \\ \mathbb{R}^+, & \text{if } \mathbf{l}_i \in \mathbb{R}, \mathbf{u}_i = +\infty \\ \mathbb{R}, & \text{if otherwise} \end{cases} \quad \text{for } i = 1, 2, \ldots, n_1 \ . \tag{3}$$

Similarly, the $\mathcal{K}_{\mathrm{d}}$ can be expressed as a Cartesian product of smaller disciplined cones: $\mathcal{K}_{\mathrm{d}} := \mathcal{K}_1 \times \mathcal{K}_2 \times \ldots \times \mathcal{K}_{l_{\mathrm{d}}}$. Moreover, its dual cone $\mathcal{K}^*_{\mathrm{d}}$ is correspondingly given by $\mathcal{K}^*_1 \times \mathcal{K}^*_2 \times \ldots \times \mathcal{K}^*_{l_{\mathrm{d}}}$.

The problem (1) has an equivalent primal-dual formulation, expressed as the following saddle-point problem on the Lagrangian $\mathcal{L}(\mathbf{x}, \mathbf{y})$:

$$\min_{\mathbf{x}\in\mathcal{X}} \max_{\mathbf{y}\in\mathcal{Y}} \ \mathcal{L}(\mathbf{x}, \mathbf{y}) := \langle \mathbf{c}, \mathbf{x}\rangle - \langle \mathbf{y}, \mathbf{G}\mathbf{x} - \mathbf{h}\rangle \ , \tag{4}$$

where $\mathcal{X} := [\mathbf{l}, \mathbf{u}] \times \mathcal{K}_{\mathrm{p}}$ and $\mathcal{Y} := \mathcal{K}_{\mathrm{d}}$. A saddle point $(\mathbf{x}, \mathbf{y})$ of (4) corresponds to a solution satisfying the Karush-Kuhn-Tucker conditions, which in turn corresponds to an optimal primal solution $\mathbf{x}$ for (1) and an optimal dual solution $(\mathbf{y}, \mathbf{c} - \mathbf{G}^\top \mathbf{y})$ for (2).

2.1 PDHG for conic programs

PDCS builds on PDHG. Let τ and σ be the primal and dual step sizes, respectively. One iteration of PDHG for solving (4) at iterate $\mathbf{z} = (\mathbf{x}, \mathbf{y})$ (denoted by OnePDHG($\mathbf{z}$)) is given by:

$$\hat{\mathbf{z}} = \texttt{OnePDHG}(\mathbf{z}) := \begin{cases} \hat{\mathbf{x}} = \text{Proj}_{[\mathbf{l},\mathbf{u}]\times\mathcal{K}_\text{p}} \left\{\mathbf{x} - \tau(\mathbf{c} - \mathbf{G}^\top \mathbf{y})\right\} \\ \hat{\mathbf{y}} = \text{Proj}_{\mathcal{K}_\text{d}} \left\{\mathbf{y} + \sigma(\mathbf{h} - \mathbf{G}(2\hat{\mathbf{x}} - \mathbf{x}))\right\} \end{cases} . \tag{5}$$

It has been shown in [9, 38, 64] that the iterates generated by (5) globally converge to the saddle point of (4) and achieve a linear convergence rate for LP problems, provided that $\tau\sigma$ is sufficiently small.

The single PDHG iteration (5) does not require any matrix factorization and its core operations are sparse matrix–vector multiplications and projections onto $[\mathbf{l}, \mathbf{u}] \times \mathcal{K}_\text{p}$ and $\mathcal{K}_\text{d}$. Matrix-vector multiplications are typically cheaper than matrix factorizations in both memory usage and computational complexity. Moreover, projections onto the Cartesian product of disciplined cones can be parallelized. Specifically, each component of the vector (corresponding to different disciplined cones) can be projected independently, i.e.,

$$\text{Proj}_{\mathcal{K}_1\times\ldots\times\mathcal{K}_l}(\mathbf{x}) = [\text{Proj}_{\mathcal{K}_1}(\mathbf{x}_{[1]})^\top, \ldots, \text{Proj}_{\mathcal{K}_l}(\mathbf{x}_{[l]})^\top]^\top , \tag{6}$$

where $\mathbf{x}_{[i]}$ denotes the component of $\mathbf{x}$ corresponding to the i-th disciplined cone $\mathcal{K}_i$. Detailed discussions on projections onto disciplined cones can be found in [52, 20]. In particular, projection onto $\mathbf{0}^d$, $\mathbb{R}^d_+$, $\mathcal{K}^{d+1}_\text{soc}$ and $\mathcal{K}_\text{exp}$ are computationally straightforward. The only "expensive" projection above is the projection onto the $\mathbb{S}^{d\times d}_+$, which used to require an eigendecomposition, but a recent work [32] proposes a matrix-free projection method that can achieve significant speedups via GPU implementation. Notably, the $\mathcal{K}^{d+2}_\text{rsoc}$ is equivalent to the second-order cone via reformulation, and the projection onto the $\mathcal{K}^*_\text{exp}$ closely resembles that of the exponential cone, which is shown in Appendix A.2. As a result, PDHG remains a fully matrix-free method, making it scalable for most large-scale conic program problems.

3 A Practical Matrix-Free Primal-Dual Method for Conic Programs

Built upon the basic PDHG framework, PDCS integrates several techniques, including adaptive step-size selection, adaptive reflected Halpern iteration, adaptive restart, and primal weight updates. The overall algorithmic framework is summarized in Algorithm 1.

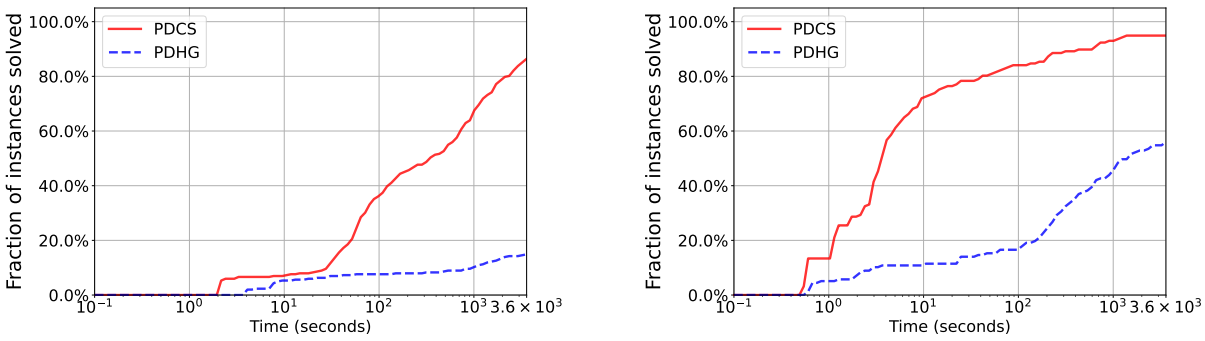
To briefly summarize the main components in Algorithm 1: the function `AdaptiveStepPDHG` in Line 4 performs a line search to determine an appropriate step size, instead of using the conservative theoretical step sizes. Lines 5, 6, and 7 together implement an adaptive reflected Halpern iteration, which is a variant of the reflected Halpern iteration that was first used by [40] to accelerate PDHG

Algorithm 1 PDCS: Primal-Dual Conic Programming Solver (without preconditioning)

Require: Initial iterate $\bar{\mathbf{z}}^{0,0} = \mathbf{z}^{0,0} = (\mathbf{x}^{0,0}, \mathbf{y}^{0,0})$, initial step-size $\eta^{0,0}$, initial primal weight ω^0, $t \leftarrow 0$

1: **repeat**
2: $\quad k \leftarrow 0$
3: $\quad$ **while** the restart condition is not triggered **do**
4: $\quad\quad \hat{\mathbf{z}}^{t,k+1}, \eta^{t,k+1} \leftarrow \texttt{AdaptiveStepPDHG}(\mathbf{z}^{t,k}, \omega^t, \eta^{t,k})$
5: $\quad\quad \beta^{t,k} \leftarrow \texttt{AdaptiveReflectionParameter}(\hat{\mathbf{z}}^{t,k+1})$
6: $\quad\quad \mathbf{z}^{t,k+1} \leftarrow \texttt{ReflectedHalpern}(\hat{\mathbf{z}}^{t,k+1}, \mathbf{z}^{t,k}, \mathbf{z}^{t,0}, \beta^{t,k}) := \frac{k+1}{k+2}((1+\beta^{t,k})\hat{\mathbf{z}}^{t,k+1} - \beta^{t,k}\mathbf{z}^{t,k}) + \frac{1}{k+2}\mathbf{z}^{t,0}$
7: $\quad\quad \bar{\mathbf{z}}^{t,k+1} \leftarrow \sum_{i=1}^{k+1} \eta^{t,i}\mathbf{z}^{t,i} / \sum_{i=1}^{k+1} \eta^{t,i}$
8: $\quad\quad \mathbf{z}_c^{t,k+1} \leftarrow \texttt{GetRestartCandidate}(\mathbf{z}^{t,k+1}, \bar{\mathbf{z}}^{t,k+1})$
9: $\quad\quad k \leftarrow k+1$
10: $\quad \mathbf{z}^{t+1,0} \leftarrow \mathbf{z}_c^{t,k}$, $\omega^{t+1} \leftarrow \texttt{PrimalWeightUpdate}(\mathbf{z}^{t+1,0}, \mathbf{z}^{t,0}, \omega^t)$, $t \leftarrow t+1$
11: **until** termination criteria hold
12: **Return** $\mathbf{z}^{t,0}$.

for LPs. The function `GetRestartCandidate` in Line 8 selects a candidate for restarting based on a measure of optimality (either the normalized duality gap or KKT error). At each outer iteration, the primal weight ω is updated based on the progress of primal and dual iterates, and this weight plays a central role in balancing the primal and dual step sizes via the function `AdaptiveStepPDHG`. Most of the above heuristics have been common in PDHG-based LP solvers, including [2, 40, 41] and others. While many of these heuristics originate in PDHG-based LP solvers, extending them to general conic programs is nontrivial and requires additional design. For example, a direct implementation of the Halpern iteration does not yield a clear benefit, so we propose a function `AdaptiveReflectionParameter` in Line 5 to compute the reflection coefficient and dampen oscillations while preserving fast local convergence. Overall, all of these heuristics preserve the matrix-free structure of PDHG. They use only matrix-vector multiplications and projections onto the disciplined cones, and thus do not change the matrix-free nature of the base PDHG method. Details of the algorithm and implementation are presented in the technical report [37].



(a) Problems with second-order cone constraints

(b) Problems with exponential and second-order cone constraints

Figure 1: Performance of PDCS and PDHG in terms of the running time.

We compare the standard PDHG method with PDCS. The performance of both methods is evaluated on two subsets of the `CBLIB` dataset. We say an instance is solved if termination tolerances

on feasibility and optimality fall below 10^{-6}. Specific experiment details are reported in Appendix D. Figure 1 plots the fraction of instances solved versus wall-clock time. Figure 1a shows the results on the dataset where all constraints are second-order cones, while Figure 1b shows results on the dataset with exponential and second-order cone constraints. On both datasets, PDCS significantly outperforms PDHG, solving substantially more instances within the time limit.

Algorithm 1 presents the PDCS on the original instance without preconditioning. In the practical version of PDCS, we also apply a diagonal rescaling to improve conditioning of the problem instance. This preprocessing yields an equivalent conic program in which the PDHG iteration is unchanged in scaled variables, but requires Euclidean projections onto diagonally rescaled cones of the form $\mathbf{D}\mathcal{K} := \{(d_1 x_1, \ldots, d_n x_n) \mid (x_1, \ldots, x_n) \in \mathcal{K}\}$ for a diagonal matrix $\mathbf{D} = \operatorname{diag}(d_1, \ldots, d_n)$. For LP cones this is trivial; for second-order cones and exponential cones we use the following characterizations to compute the projections.

Theorem 1. Projection onto $\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1}$. *The projection of $(t, \mathbf{x})$ onto rescaled cone $\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1}$ is given as follows. For $\mathbf{D}$, let $\hat{\mathbf{D}}$ denote $\operatorname{diag}(\hat{d}_2, \ldots, \hat{d}_{n+1})$ with $\hat{d}_i = d_i/d_1, \forall i \geq 2$. If $t \leq 0$ and $\|\hat{\mathbf{D}}\mathbf{x}\| \leq -t$, then $\operatorname{Proj}_{\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1}}\{(t, \mathbf{x})\} = (0, \mathbf{0})$. If $\left\|\hat{\mathbf{D}}^{-1}\mathbf{x}\right\| \leq t$, then $\operatorname{Proj}_{\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1}}\{(t, \mathbf{x})\} = (t, \mathbf{x})$. If $t = 0$ and $\mathbf{x} \neq \mathbf{0}$, then $\operatorname{Proj}_{\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1}}\{(t, \mathbf{x})\} = \left(\|(\hat{\mathbf{D}} + \hat{\mathbf{D}}^{-1})^{-1}\mathbf{x}\|, (\mathbf{I}_{n\times n} + \hat{\mathbf{D}}^{-2})^{-1}\mathbf{x}\right)$. Otherwise, if $\left\|\hat{\mathbf{D}}^{-1}\mathbf{x}\right\| > t$, then let $\lambda > 0$ be the solution of the following univariate equation of λ:*

$$\sum_{i=1}^{n} \left(\frac{\hat{d}_{i+1}^{-1}}{1 + 2\lambda \hat{d}_{i+1}^{-2}} \mathbf{x}_i \right)^2 - \frac{t^2}{(1-2\lambda)^2} = 0\,, \quad \frac{t}{(1-2\lambda)} > 0. \tag{7}$$

Then the projection is given by $\operatorname{Proj}_{\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1}}\{(t, \mathbf{x})\} = \left((1-2\lambda)^{-1}t, (\mathbf{I} + 2\lambda\hat{\mathbf{D}}^{-2})^{-1}\mathbf{x}\right)$.

In practice, the fourth case is handled using a root-finding method (such as a bisection method) because when regarded as a function, the left-hand side of the equality in (7) is continuous and has values of opposite sign at the endpoints.

Let $\mathbf{D} = \operatorname{diag}(d_r, d_s, d_t) \succ 0$ and $v_0 = (r_0, s_0, t_0) \in \mathbb{R}^3$. The projection onto the rescaled exponential cone $\mathbf{D}\mathcal{K}_{\text{exp}}$ can also be computed by case analysis. See an informal statement of the result below. The proof of Theorem 1 is given in Appendix A.1. A full statement and proof for the diagonally scaled exponential cone projection are provided in Appendix A.2.

Theorem 2. Projection onto $\mathbf{D}\mathcal{K}_{\text{exp}}$ (Informal). *Let $\mathbf{D} = \operatorname{diag}(d_r, d_s, d_t) \succ 0$. For any $v_0 = (r_0, s_0, t_0)$, the projection $\operatorname{Proj}_{\mathbf{D}\mathcal{K}_{\text{exp}}}(v_0)$ is given as follows: If v_0 is already in $\mathbf{D}\mathcal{K}_{\text{exp}}$, then $\operatorname{Proj}_{\mathbf{D}\mathcal{K}_{\text{exp}}}(v_0) = v_0$. If v_0 is in the polar cone $-\mathbf{D}^{-1}\mathcal{K}_{\text{exp}}^*$, then $\operatorname{Proj}_{\mathbf{D}\mathcal{K}_{\text{exp}}}(v_0) = \mathbf{0}$. If $r_0 \leq 0$ and $s_0 \leq 0$, then $\operatorname{Proj}_{\mathbf{D}\mathcal{K}_{\text{exp}}}(v_0) = (r_0, 0, t_0^+)$. Otherwise, $\operatorname{Proj}_{\mathbf{D}\mathcal{K}_{\text{exp}}}(v_0)$ is determined by a scalar ρ that is the root of a continuous univariate function $h(\rho)$ on an explicitly computable interval. Once ρ is obtained (via solving the root-finding problem), $\operatorname{Proj}_{\mathbf{D}\mathcal{K}_{\text{exp}}}(v_0)$ is given by closed-form expressions in ρ.*

4 cuPDCS: A PDCS Implementation with GPU Enhancements

In this section, we present the design of PDCS optimized for efficiently exploiting GPU architectures. We call this GPU implementation cuPDCS. It is available at https://github.com/ZikaiXiong/PDCS. It is designed to be easy to use, even for users without deep expertise in conic optimization. It integrates with mathematical programming modeling platforms, JuMP [44] and CVXPY [15].

As demonstrated in previous sections, the primary computational bottlenecks of PDCS are matrix-vector multiplications and projections onto cones. While matrix-vector multiplications are already well-optimized on GPUs (see, e.g., [5]), projections onto cones remain comparatively under-optimized. Consequently, developing an efficient projection strategy is essential for achieving overall performance gains in our cuPDCS.

We first briefly review the hierarchical organization of GPU parallel computing, which comprises three levels of execution granularity: grid, block, and thread (Figure 2). Upon dispatch to the GPU, each workload is organized into a single grid consisting of multiple blocks; each block contains numerous threads that share block-level (shared) memory and can synchronize their execution. Accordingly, it is crucial to allocate computational resources properly. Moreover, empirical evidence indicates that frequent data transfers between the CPU and GPU result in significant overhead. This is a primary reason why existing GPU solvers (e.g., [41, 43]) almost all emphasize executing the most computations on the GPU. Consequently, a fundamental design principle for projection operators is to reduce data transfers as much as possible between the CPU and GPU while maximizing the proportion of computations executed on the GPU. In the case of multi-cone projection, it is also essential to allocate computational resources in a manner that enables all cone projections to be executed in parallel. Recall that, as indicated in (6), cone projections can be carried out by separately projecting onto rescaled disciplined cones. As discussed in Section 3, projections onto second-order and exponential cones can be reformulated as root-finding problems that typically involve a few vector inner products and scalar multiplications.

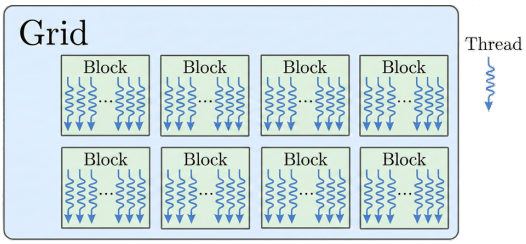


Figure 2: Illustration of the GPU architecture.

In our GPU implementation of PDCS, we employ three parallel computing strategies. **Grid-wise**: Assign the entire grid to execute each cone projection sequentially. Vector inner products are computed using the cuBLAS library, which delivers high-performance capability for individual operations. However, each cuBLAS call also incurs significant launch and resource-allocation overhead. **Block-wise**: Assign one disciplined cone projection to each CUDA block. This approach amortizes the kernel-launch overhead by simultaneously processing many cones within a single kernel invocation. **Thread-wise**: Assign one disciplined cone projection to each CUDA thread. This strategy removes nearly all synchronization and kernel-launch overhead, allowing simultaneous processing of a large number of tasks by distributing them across individual threads.

A distinguishing feature of the projection phase is that the number, dimension, and type of individual disciplined cones can vary substantially. Consequently, no single parallelization strategy can be expected to perform optimally across all cone types or configurations. For instance, consider the task of projecting a randomly generated vector in $\mathbb{R}^{m(d+1)}$ onto a multi-block second-order cone $\mathcal{K} := \mathcal{K}_{\text{soc}}^{d+1} \times \mathcal{K}_{\text{soc}}^{d+1} \times \cdots \times \mathcal{K}_{\text{soc}}^{d+1}$ where there are m second-order cone blocks. In our experiments, we fix the dimension d of each cone as $d = \lceil 1.2 \times 10^9/m \rceil$, thus ensuring that the total dimensionality of the cone remains at least 1.2×10^9. Figure 3 compares the performance of the three proposed strategies—grid-wise (one grid per projection), block-wise (one block per projection), and thread-wise (one thread per projection)—under this configuration. In this test, a time limit of 15 seconds is imposed, and we report the average runtime along with the standard deviation (indicated by the shaded region) over 10 independent trials. Note that the CPU implementation of these projections always requires more than 15 seconds; hence, its data points are represented by a horizontal dotted line at 15 seconds.

Intuitively, the grid-wise strategy benefits from the highly optimized norm computations provided by the cuBLAS library, rendering it particularly effective for a small number of high-dimensional tasks. However, this strategy suffers when addressing a large number of small-dimension tasks, as each cuBLAS function call incurs non-negligible overhead that accumulates significantly across many projections. In contrast, the block-wise approach, which effectively utilizes shared memory to perform in-block reductions, is ideally suited for a moderate number of tasks with moderate dimensions. Finally, the thread-wise method may become the best when the workload consists of a very large number of very low-dimensional tasks, as each small cone projection can be efficiently handled by an individual thread, and the overall massive parallelism is fully exploited.

When m (the number of cones) is very small (i.e., each cone is high-dimensional), the grid-wise strategy performs best. As m grows and d becomes moderate, the block-wise strategy becomes more efficient. Ultimately, when m is extremely large (yielding many small cones), the thread-wise approach is the best, since a single thread is sufficient for handling the small cone projection and the large number of threads provide high parallelism. Note that we have a time limit of 15 seconds for the projection task, and both the thread-wise and block-wise strategies exceed the threshold for a small number of high-dimensional cones, whereas the grid-wise strategy exceeds the threshold as the number of cones increases.

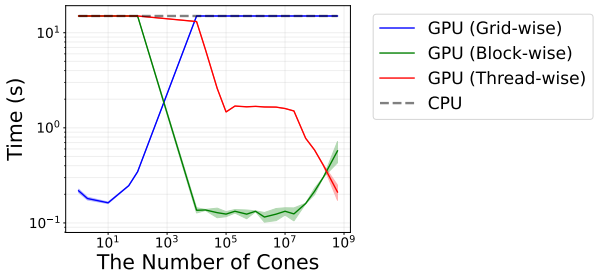


Figure 3: Runtime of projections onto second-order cones using different parallelization strategies.

Another key factor in resource allocation is the computational cost associated with different cone types. For the zero and nonnegative cones, the projection requires only simple elementwise comparisons (and, for the nonnegative cone, scalar clamping). These inexpensive operations have

negligible synchronization or overhead. Projection onto the exponential cone, by solving a root-finding problem, likewise consists primarily of elementwise comparisons and scalar multiplications, making it well-suited for thread-level parallelism. Consequently, for these three cone types, we assign one GPU thread per cone, thereby minimizing synchronization overhead.

In contrast, projection onto a second-order cone with diagonal rescaling (Theorem 1) reduces to solving a univariate root-finding problem that involves both elementwise products and a Euclidean norm calculation. While the elementwise products can be handled in parallel, the norm computation necessitates a reduction (i.e., summation), which requires synchronization. To balance throughput and synchronization costs, we implement reduction at three distinct levels. At the grid level, we directly leverage the cuBLAS library; at the block level, reduction is conducted using shared memory; and at the thread level, we perform serial addition. These three strategies correspond to the methods we previously proposed and are aimed at ensuring efficiency for second-order cone projections under varying scenarios.

Table 1: Projection strategies for different cone types.

Cone Type	Thread-wise	Block-wise	Grid-wise
Zero Cone	✓	✗	✗
Nonnegative Cone	✓	✗	✗
SOC	✓	✓	✓
Exponential Cone	✓	✗	✗

Both the block-wise and thread-wise strategies avoid repeated CPU-GPU data transfers and the overhead associated with multiple cuBLAS calls by performing the entire projection operation within a single kernel launch. By selecting the number of blocks (in the block-wise strategy) or threads (in the thread-wise approach) based on available GPU resources, we can execute projections for all cones in a multi-cone structure concurrently. This approach increases hardware utilization, and leads to more efficient projections for large-scale instances. Table 1 summarizes the projection strategies for different cone types.

Finally, we examine the effectiveness of these strategies on exponential-cone projections in Figure 4 by projecting a randomly generated vector to a multi-block exponential cone. We compare the GPU-based, thread-wise strategy to a CPU-based implementation as the number of cones varies. Similar to Figure 3, Figure 4 displays the mean projection time over ten runs, with the shaded region representing the corresponding standard deviation. The results demonstrate that the massive parallelism of the GPU provides significant speedups, and the performance gap between the GPU- and CPU-based implementations widens with increasing numbers of cones.

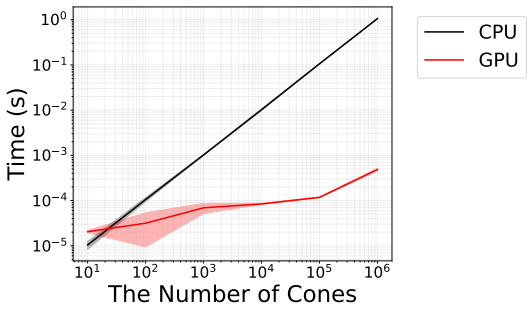


Figure 4: Projections onto exponential cones using different parallelization strategies.

5 Numerical experiments

We evaluate our method's performance against several state-of-the-art conic solvers:

- **SCS** [51], **ABIP** [14]: ADMM-based solvers. SCS provides CPU-direct, CPU-indirect, and GPU-indirect variants. Here, "direct" denotes the use of matrix factorization, while "indirect" refers to iterative methods for solving inner linear systems.
- **CuClarabel** [12]: An open-source solver implementing an IPM. It uses the cuDSS library to solve the linear systems on the GPU via matrix factorizations.
- **MOSEK** [48], **COPT** [22]: Commercial IPM-based solvers.

The methods that do not require matrix factorization are classified as *matrix-free first-order methods*. They include SCS(indirect) and PDCS (including its GPU implementation cuPDCS). The methods that require a single matrix factorization are classified as *non-matrix-free first-order methods*. They include SCS(direct) and ABIP. The other solvers are essentially based on IPMs. They are CuClarabel, MOSEK and COPT. For certain problem instances, the presolve capabilities of the commercial solvers can detect structural properties, thereby significantly reducing problem complexity and potentially offering a computational advantage. We denote versions of COPT and MOSEK with presolve enabled as COPT* and MOSEK*, respectively, and the versions without presolve as COPT and MOSEK.

We organize the experiments as follows. Section 5.1 evaluates performance on the CBLIB, a small-scale classical conic program dataset. In Section 5.2, we consider the Fisher market equilibrium problem, a conic programming problem involving the exponential cone. Section 5.3 focuses on solving large-scale Lasso problems. Finally, in Section 5.4, we consider multi-period portfolio optimization, a conic program with multiple second-order cones. The conic program formulations of these problems and supplementary details are provided in Appendix B. All GPU experiments are conducted on an NVIDIA H100 with 80 GB of VRAM, running on a cluster equipped with an Intel Xeon Platinum 8468 CPU. All CPU benchmarks are performed on a Mac mini M2 Pro with 32 GB of RAM.

Because each method adopts different formulations and methodologies, their actual built-in termination criteria can vary even when we set the same tolerance. Moreover, some methods keep their termination criteria private. There is no uniform convergence standard across the methods we evaluated, but we attempt to document and summarize the known termination criteria for these methods in Appendix C. For PDCS and cuPDCS, we assess primal and dual feasibility, as well as primal-dual optimality:

$$\begin{aligned}
&\mathrm{err_p}(\mathbf{x}) := \left\|(\mathbf{G}\mathbf{x}-\mathbf{h}) - \mathrm{Proj}_{\mathcal{K}_\mathrm{d}^*}\{\mathbf{G}\mathbf{x}-\mathbf{h}\}\right\|_\infty / \left(1+\max\left\{\|\mathbf{h}\|_\infty, \|\mathbf{G}\mathbf{x}\|_\infty, \left\|\mathrm{Proj}_{\mathcal{K}_\mathrm{d}^*}\{\mathbf{G}\mathbf{x}-\mathbf{h}\}\right\|_\infty\right\}\right),\\
&\mathrm{err_d}(\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2) := \max\left\{\|\boldsymbol{\lambda}_1 - \mathrm{Proj}_\Lambda\{\boldsymbol{\lambda}_1\}\|_\infty, \left\|\boldsymbol{\lambda}_2 - \mathrm{Proj}_{\mathcal{K}_\mathrm{p}^*}\{\boldsymbol{\lambda}_2\}\right\|_\infty\right\} / \left(1+\max\left\{\|\mathbf{c}\|_\infty, \left\|\mathbf{G}^\top\mathbf{y}\right\|_\infty\right\}\right),\\
&\mathrm{err_{gap}}(\mathbf{x},\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2) := \left|\langle\mathbf{c},\mathbf{x}\rangle - \left(\mathbf{y}^\top\mathbf{h} + \mathbf{l}^\top\boldsymbol{\lambda}_1^+ - \mathbf{u}^\top\boldsymbol{\lambda}_1^-\right)\right| / \left(1+\max\left\{|\langle\mathbf{c},\mathbf{x}\rangle|, \left|\mathbf{y}^\top\mathbf{h} + \mathbf{l}^\top\boldsymbol{\lambda}_1^+ - \mathbf{u}^\top\boldsymbol{\lambda}_1^-\right|\right\}\right).
\end{aligned} \tag{8}$$

Our method terminates once all three of these criteria fall below a specified tolerance, which is usually either 10^{-3} or 10^{-6}. According to the Karush-Kuhn-Tucker conditions, when these three criteria are all zero, the solution is optimal.

In order to compare the runtime of different methods across multiple instances, we use the shifted geometric mean (SGM): $\text{SGM}(k) := \left[\prod_{i=1}^{N}(t_i + k)\right]^{1/N} - k$ where t_i is the time (in seconds) for the method to run on problem i, and k is a shift parameter that is often set as 10 [46]. SGM mitigates the potentially significant influence of tiny runtimes of only a few instances on the overall geometric mean. If a method fails to solve a given instance, we assign t_i to be the maximum allowable time (e.g., the time limit).

5.1 Relaxation of Problems from the CBLIB Dataset

CBLIB is a public dataset of mixed-integer conic programming problems collected from real-world applications [19]. We run our experiments on the instances with LP cones, second-order cones, and exponential cones. In line with standard practice for solving such problems, we first use COPT to apply presolve and subsequently relax all integer constraints, thereby producing two refined subsets: (1) 1,943 problems that do not involve exponential cones, and (2) 157 problems that include exponential cones.

For the subset without exponential cones, we categorize problem instances as small-, medium-, and large-scale based on the number of nonzeros in the constraint matrix, using thresholds of 50,000 and 500,000. As a result, the dataset comprises 1,641 small-scale, 220 medium-scale and 82 large-scale problem instances. It should be noted that almost all of these problems are relatively small in scale so the IPM-based solvers and the SCS(direct) are unlikely to reach computational bottlenecks and can easily perform the required matrix factorizations. We start with this dataset to give an overview of the performance of PDCS on small-scale problems. All experiments in this benchmark are executed with a one-hour time limit and a termination tolerance of 10^{-6}.

Table 2: Conic Program Problems without Exponential Cones

		Small(1641)		Medium(220)		Large(82)		Total(1943)	
		SGM(10)	solved	SGM(10)	solved	SGM(10)	solved	SGM(10)	solved
Matrix-free first-order methods	SCS(indirect)	3.31	1638	231.68	188	1856.58	46	12.77	1872
	PDCS	2.78	1640	160.64	208	1602.14	53	11.02	1901
	cuPDCS	2.91	1640	**44.95**	211	**312.65**	57	7.43	1908
Non-matrix-free first-order methods	ABIP	3.66	1622	1342.77	84	3458.00	2	19.04	1708
	SCS(direct)	**0.59**	1636	54.98	213	463.21	58	**5.26**	1907
Interior-point methods	CuClarabel	0.15	1640	**1.32**	219	46.17	61	1.05	1920
	COPT*	0.08	1639	2.03	215	53.76	58	1.12	1912
	MOSEK*	**0.05**	1640	2.34	214	**6.21**	81	**0.49**	1935

Tables 2 and 3 summarize the results for these two categories of problems. Among the first-order methods, SCS(indirect) is the fastest for small-scale problems and the problems with exponential cones. Our cuPDCS is faster for the medium-scale and large-scale problems without exponential cones. Since these problems are still relatively small, interior-point methods all perform far better

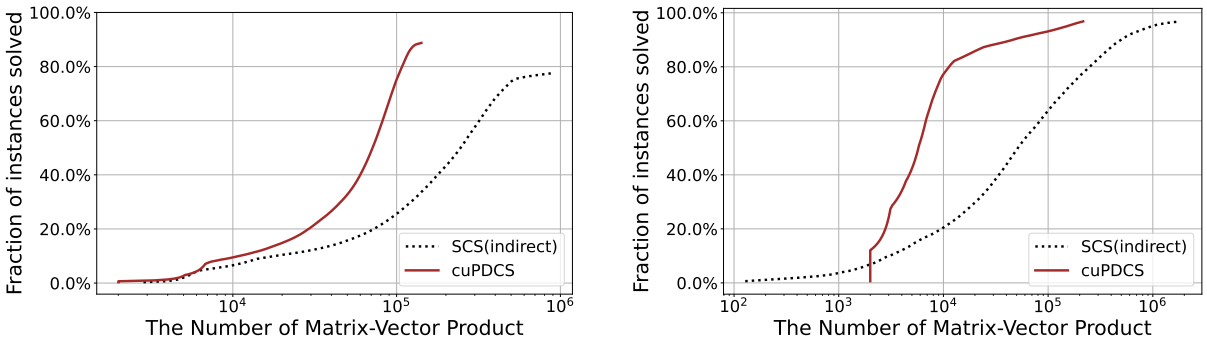
than first-order methods on these problems.

Both the CPU-based version of our method (denoted by PDCS) and its GPU-enhanced implementation (denoted by cuPDCS) exhibit strong numerical stability, solving nearly as many instances as the commercial solver COPT. However, the CPU version occasionally struggles with more challenging instances, primarily due to the limited parallelism in projection operations and matrix-vector multiplications. In contrast, the GPU implementation, cuPDCS, enables more iterations within the allotted time and successfully solves some instances that are otherwise too difficult for the CPU version.

Table 3: Conic Program Problems with Exponential Cones

	First-order methods				Interior-point methods		
	SCS(direct)	SCS(indirect)	PDCS	cuPDCS	CuClarabel	COPT*	MOSEK*
SGM(10)	**6.07**	19.98	18.52	12.04	0.87	**0.13**	5.19
solved	152	152	149	155	156	157	146

The computational bottleneck of matrix-free methods lies in the sparse matrix-vector products, so we also compare the number of matrix-vector products required for cuPDCS and SCS(indirect) to solve a particular number of problems. Figure 5 reports the fraction of instances solved versus the number of sparse matrix–vector products. Notably, when we compare the number of matrix-vector products needed to solve 60% of the problem instances, we see that cuPDCS requires only about one-fifth to one-tenth as many matrix-vector products as the indirect SCS. Since cuPDCS follows the PDCS algorithm, we can conclude that, even if not implemented on a GPU, PDCS still requires fewer iterations than the other matrix-free solver, SCS(indirect). This advantage is not clearly reflected in the runtime reported in Table 2 because the actual runtime significantly depends on implementation details and programming language. In particular, for small-scale problems, a substantial amount of time is consumed by fixed overheads.



(a) Medium- and large-scale conic program problems without exponential cones

(b) Conic program problems with exponential cones

Figure 5: Performance of cuPDCS and SCS(indirect) in terms of the number of matrix-vector products.

Among the interior-point methods, MOSEK* consistently demonstrates advantages over other methods for conic programs without exponential cones. However, for problems involving expo-

nential cones, MOSEK* frequently fails to meet the necessary solution tolerance, resulting in a "SLOW_PROGRESS" status. All such instances are regarded as failures, regardless of whether the time limit is reached. This makes the SGM of MOSEK* for problems with exponential cones appear worse than the other interior-point methods.

It should be emphasized that the problems in the CBLIB dataset are relatively small in scale. For these problems, solving the matrix factorizations and the corresponding linear systems remains accessible. Hence, the SCS with direct factorizations implemented on CPUs appears to be the best first-order method, while all IPMs significantly outperform first-order methods. However, as the dimension of the problem instance increases, SCS(direct) and all IPMs face greater challenges and become considerably slower. In the following, we present the experiments on larger-scale problem.

5.2 Fisher Market Equilibrium Problem

We next consider Fisher market equilibrium instances [16, 65], an important optimization problem arising in economics. We formulate it as a standard conic program with exponential cones (see the specific formulation in Appendix B) and compare various methods (excluding ABIP as it is not applicable) under two levels of solution accuracy: low accuracy (10^{-3}) and high accuracy (10^{-6}). From now on, the time limits are set to 2 hours for the low-accuracy tests and 5 hours for the high-accuracy tests. The results are summarized in Table 4. The first three columns m, n and nnz of Table 4 denote the number of rows, the number of columns, and the number of nonzero entries of $\mathbf{U}$, respectively (see (28) in Appendix B for more details).

Table 4: Experiments on Fisher market equilibrium problems.

m	n	nnz	Without Presolve					With Presolve	
			cuPDCS	SCS(GPU)	CuClarabel	COPT	MOSEK	COPT*	MOSEK*
					10^{-3}				
1.0E+02	5.0E+03	1.0E+05	1.5E+01	f	1.0E+01	2.6E+00	1.0E+00	2.8E+00	**4.2E-01**
1.0E+05	1.0E+03	2.0E+07	4.6E+02	f	f	f	2.5E+03	f	**1.5E+02**
1.5E+05	1.0E+03	3.0E+07	4.3E+02	f	f	f	f	f	**2.5E+02**
2.0E+05	1.0E+03	4.0E+07	**1.1E+03**	f	f	f	f	f	3.9E+03
2.5E+05	1.0E+03	5.0E+07	**1.4E+03**	f	f	f	f	f	6.4E+03
2.8E+05	1.0E+03	5.5E+07	**1.6E+03**	f	f	f	f	f	f
					10^{-6}				
1.0E+02	5.0E+03	1.0E+05	3.7E+01	1.2E+04	1.2E+01	2.8E+00	1.1E+00	3.0E+00	**4.2E-01**
1.0E+05	1.0E+03	2.0E+07	1.8E+03	f	f	f	2.7E+03	f	**1.8E+02**
1.5E+05	1.0E+03	3.0E+07	2.8E+03	f	f	f	f	f	**2.9E+02**
2.0E+05	1.0E+03	4.0E+07	4.2E+03	f	f	f	f	f	**4.0E+03**
2.5E+05	1.0E+03	5.0E+07	**5.7E+03**	f	f	f	f	f	6.5E+03
2.8E+05	1.0E+03	5.5E+07	**6.2E+03**	f	f	f	f	f	7.7E+03

Note. Entries marked "f" indicate failure due to exceeding the time limit or returning errors. Bold entries indicate the best runtime among all methods for each row. Shaded cells highlight the best-performing method on the original problem (without presolve).

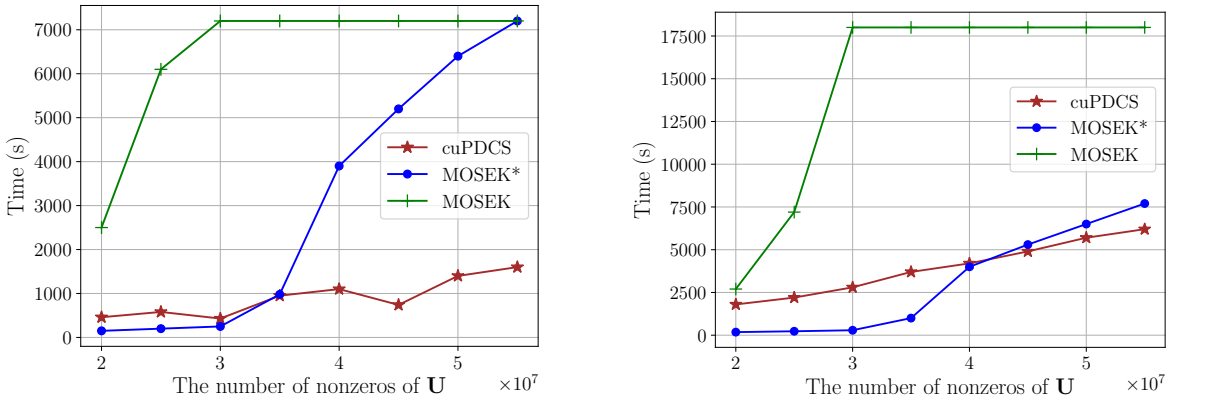
As shown in Table 4, cuPDCS consistently outperforms the other matrix-free GPU-enhanced solver, SCS(GPU). Although both methods utilize GPU, SCS(GPU) is significantly slower. This is primarily because SCS(GPU) is a GPU implementation of SCS(indirect), which typically requires

more iterations to converge than cuPDCS. Moreover, SCS(GPU) uses the GPU only to accelerate the iterative method for solving the linear system, while the rest of the algorithm remains implemented on CPU. This design results in substantial data transfer between the CPU and GPU, which incurs significant overhead and slows down the overall performance. Nevertheless, SCS(GPU) is still faster than its two CPU versions, SCS(indirect) and SCS(direct).

Even when compared to IPMs such as COPT and MOSEK, cuPDCS performs competitively, often outperforming them on problem instances where $\mathbf{U}$ contains more than 2×10^7 nonzero elements. Although CuClarabel is a GPU-accelerated interior-point solver, its performance is generally closer to that of COPT and MOSEK than to cuPDCS, suggesting that classic IPMs benefit less from GPU than first-order methods. It is also worth noting that cuPDCS is more sensitive to accuracy requirements than interior-point methods. When the target tolerance tightens from 10^{-3} to 10^{-6}, the runtime of cuPDCS increases significantly, while the runtimes of IPMs remain relatively stable. This indicates that computing high-accuracy solutions is more challenging for cuPDCS.

The presolve function in commercial solvers can significantly affect performance. Presolve may exploit matrix structure and reduce the cost of matrix factorizations, though the benefits are not always guaranteed. For MOSEK, presolve leads to improved robustness and faster runtimes. MOSEK* performs substantially better than its non-presolved counterpart. Conversely, for COPT, presolve leads to a slight performance degradation in our experiments. When only comparing algorithm performance on the original problems (without presolve), cuPDCS performs the best on all problems but the smallest-scale problem, as indicated by the shaded cells.

To further illustrate the scalability of cuPDCS, Figure 6 compares its performance against MOSEK and MOSEK*. Runtimes are plotted against problem size, defined by the non-zero count of $\mathbf{U}$. In the low-accuracy setting, cuPDCS enjoys a substantial speed advantage over MOSEK, and this advantage becomes more pronounced as the problem size increases. Moreover, regardless of the target tolerance, when the number of nonzeros exceeds approximately 4×10^7, the runtime of MOSEK* grows sharply, whereas cuPDCS maintains better scalability, showcasing the benefits of GPU-accelerated first-order methods for solving large-scale conic programs.



(a) Target tolerance 10^{-3}.

(b) Target tolerance 10^{-6} (Time limit: 5 hours).

Figure 6: Performance of cuPDCS, MOSEK, and MOSEK* on problems of different scales.

5.3 Lasso Problem

The Lasso problem is a classic optimization model in statistical learning that integrates variable selection and regularization to improve both prediction accuracy and model interpretability: $\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2 + \lambda \|\mathbf{x}\|_1$, where $\mathbf{A}$ is the data matrix, $\mathbf{b}$ is the response vector, and $\lambda > 0$ is the regularization parameter.

We generate a family of synthetic Lasso problem instances following the experimental setting used for ABIP in [14]. By decomposing the decision variable $\mathbf{x}$ into its positive and negative parts, the problem can be equivalently reformulated as an SOCP. We again compare the methods under the two levels of solution accuracy. The results are summarized in Table 5. The first three columns m, n and nnz denote the number of rows, the number of columns, and the number of nonzero entries of $\mathbf{A}$.

Table 5: Experiments on Lasso problems.

m	n	nnz		Without Presolve					With Presolve	
			cuPDCS	SCS(GPU)	ABIP	CuClarabel	COPT	MOSEK	COPT*	MOSEK*
					10^{-3}					
1.0E+04	1.0E+05	1.0E+05	**7.1E-02**	5.5E+02	7.5E+00	4.1E+01	2.4E+02	2.0E+00	6.7E+01	1.7E+00
7.0E+04	7.0E+05	4.9E+06	**2.9E-01**	f	f	4.3E+02	f	1.7E+03	f	3.1E+03
4.0E+05	7.0E+06	2.8E+08	**1.2E+02**	f	f	f	f	f	f	f
7.0E+05	7.0E+06	4.9E+08	**3.2E+02**	f	f	f	f	f	f	f
7.5E+05	7.5E+06	5.6E+08	**4.7E+02**	f	f	f	f	f	f	f
					10^{-6}					
1.0E+04	1.0E+05	1.0E+05	**1.1E-01**	1.1E+03	1.6E+01	4.3E+01	2.5E+02	3.6E+00	6.7E+01	2.1E+00
7.0E+04	7.0E+05	4.9E+06	**8.4E-01**	f	f	f	f	2.5E+03	f	3.5E+03
4.0E+05	7.0E+06	2.8E+08	**2.6E+02**	f	f	f	f	f	f	f
7.0E+05	7.0E+06	4.9E+08	**5.2E+02**	f	f	f	f	f	f	f
7.5E+05	7.5E+06	5.6E+08	**6.0E+02**	f	f	f	f	f	f	f

Note. See Table 4 for definitions of "f", bold entries, and shaded cells.

It is observed from Table 5 that, for Lasso problems, cuPDCS consistently achieves the best performance among all tested methods, including both first-order and IPMs, whether or not presolve is applied. To evaluate scalability, we examined instances with matrix dimensions $m = 7.5\times10^5$ and $n = 7.5 \times 10^6$. Even at this extreme scale, cuPDCS successfully computed high-accuracy solutions within a practical time frame. Moreover, cuPDCS shows only a modest increase in runtime when transitioning from a target tolerance of 10^{-3} to 10^{-6}. This near-constant runtime indicates that the underlying PDCS algorithm converges faster (potentially at a linear rate) on Lasso problems compared to Fisher market equilibrium problems. This conjecture is further supported by the empirical convergence behavior illustrated in Figure 8 in Section E.

While presolve continues to benefit IPM-based solvers like COPT and MOSEK, the improvement is less significant than in the Fisher market experiments. In contrast, GPU acceleration offers greater advantages. Specifically, CuClarabel demonstrates performance comparable to that of both COPT* and MOSEK*. However, its applicability is confined to moderate problem sizes, as it struggles to solve very large-scale instances and is slower than commercial solvers on small-scale instances.

5.4 Multi-period Portfolio Optimization Problems

In this section, we study Multi-Period Portfolio Optimization (MPO) problems (see the specific formulation in Appendix B) , which include a mix of multiple cone types and increasingly higher problem dimensions as the number of time periods grows.

We select all securities from five major indices (S&P 500, NASDAQ-100, DOW 30, FTSE 100, and Nikkei 225) resulting in 844 assets in total. The historical data used to construct the problem was retrieved using the `cvxportfolio` package [8], which sources prices from Yahoo Finance. The data from the year 2024 were used to estimate the returns, variances, and prices for the first period. For subsequent periods, the predicted data were generated by introducing a controlled level of random noise to the first period's estimates. The parameters are set as $\gamma_{1\tau} = \left\|\hat{\mathbf{\Sigma}}_{\tau+1}^{1/2}(\mathbf{w}_{1/n} - \mathbf{w}_b)\right\|$, $\gamma_{2\tau,i} = 0.05$, and $\gamma_{3\tau} = 0.05$, where $\mathbf{w}_{1/n}$ denotes the equally weighted strategy.

Table 6: Experiments on MPO problems.

T	cuPDCS	Without Presolve					With Presolve	
		SCS(GPU)	ABIP	CuClarabel	COPT	MOSEK	COPT*	MOSEK*
				10^{-3}				
3	1.9E+00	3.7E+00	1.9E+00	1.5E+00	8.3E-01	8.1E-01	8.6E-01	**4.0E-02**
48	7.2E+00	3.5E+02	4.0E+01	9.9E+01	8.8E+00	1.7E+01	9.3E+00	**8.6E-01**
96	1.1E+01	1.8E+03	9.1E+01	6.3E+02	1.9E+01	3.3E+01	2.0E+01	**1.7E+00**
360	5.1E+01	f	1.9E+02	f	1.2E+02	1.2E+02	1.2E+02	**7.2E+00**
1440	4.9E+02	f	9.0E+02	f	f	f	f	**4.8E+01**
2160	**5.8E+02**	f	f	f	f	f	f	f
3600	**1.1E+03**	f	f	f	f	f	f	f
				10^{-6}				
3	7.5E+00	f	f	1.6E+00	8.3E-01	1.3E+00	8.4E-01	**5.0E-02**
48	1.8E+01	f	f	1.1E+02	8.8E+00	2.7E+01	8.8E+00	**9.9E-01**
96	5.7E+01	f	f	6.1E+02	1.9E+01	6.0E+01	1.9E+01	**1.9E+00**
360	4.2E+02	f	f	9.2E+03	1.2E+02	2.4E+02	1.1E+02	**8.2E+00**
1440	3.4E+03	f	f	f	f	f	f	**5.6E+01**
2160	**1.0E+03**	f	f	f	f	f	f	f
3600	**9.0E+03**	f	f	f	f	f	f	f

Note. See Table 4 for definitions of "f", bold entries, and shaded cells.

We again evaluate performance on MPO problems under two levels of solution accuracy: low accuracy (10^{-3}) and high accuracy (10^{-6}). Results are presented in Table 6. As the number of time periods T increases, the number of constraints, variables, and cones grows proportionally.

From Table 6, we observe that cuPDCS consistently outperforms the other first-order methods, SCS(GPU) and ABIP. Notably, ABIP is available only as a CPU implementation and still relies on at least one matrix factorization per iteration, making it less suitable for large-scale problems. Among the methods that work directly on the original problem formulation (i.e., without presolve), cuPDCS demonstrates a clear advantage on large-scale instances.

However, as the desired accuracy increases, the runtime of cuPDCS grows more rapidly than that of IPMs. Presolve can significantly enhance MOSEK's performance, making it more robust and quicker on moderate-sized instances. Yet, even MOSEK* struggles to handle large-scale MPO

problems due to its considerable memory requirements. For cases with $T \geq 2160$, MOSEK* runs out of memory, whereas cuPDCS continues to perform effectively. These results show the scalability and robustness of cuPDCS for solving large-scale conic optimization problems involving mixed cone types.

In summary, cuPDCS demonstrates consistently strong performance across a wide range of large-scale conic optimization problems. It outperforms other first-order methods, such as SCS and ABIP, particularly on high-dimensional problems, where GPU acceleration is more effective. Moreover, even in comparison with commercial interior-point method solvers like MOSEK and COPT (both with and without presolve), cuPDCS remains highly competitive. Its advantages are especially pronounced in large-scale, lower-accuracy settings. It would be interesting to study the presolve for first-order methods like cuPDCS, as it could potentially lead to another significant speedup, similar to the presolve for MOSEK.

References

[1] Jonas Adler, Holger Köhr, and Ozan Öktem. Operator discretization library (ODL), 2017.

[2] David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O'Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. In *Advances in Neural Information Processing Systems*, volume 34, pages 20243–20257, 2021.

[3] David Applegate, Oliver Hinder, Haihao Lu, and Miles Lubin. Faster first-order primal-dual methods for linear programming using restarts and sharpness. *Mathematical Programming*, 201(1):133–184, 2023.

[4] Chaithanya Bandi, Nikolaos Trichakis, and Phebe Vayanos. Robust multiclass queuing theory for wait time estimation in resource allocation systems. *Management Science*, 65(1):152–187, 2019.

[5] Nathan Bell and Michael Garland. Efficient sparse matrix-vector multiplication on CUDA. Technical report, NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, 2008.

[6] Alexander Biele and Dinakar Gade. FICO® xpress solver 9.4, 2024. Accessed: 2025-10-07.

[7] Edward H Bowman. Production scheduling by the transportation method of linear programming. *Operations Research*, 4(1):100–103, 1956.

[8] Stephen Boyd, Enzo Busseti, Steven Diamond, Ron Kahn, Peter Nystrup, and Jan Speth. Multi–period trading via convex optimization. *Foundations and Trends in Optimization*, 3(1):1–76, 2017.

[9] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40:120–145, 2011.

[10] Abraham Charnes and William W Cooper. The stepping stone method of explaining linear programming calculations in transportation problems. *Management Science*, 1(1):49–69, 1954.

[11] Li Chen, Long He, and Yangfang Zhou. An exponential cone programming approach for managing electric vehicle charging. *Operations Research*, 72(5):2215–2240, 2024.

[12] Yuwen Chen, Danny Tse, Parth Nobel, Paul Goulart, and Stephen Boyd. CuClarabel: GPU acceleration for a conic optimization solver. *arXiv preprint arXiv:2412.19027*, 2024.

[13] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 4th edition, 2022.

[14] Qi Deng, Qing Feng, Wenzhi Gao, Dongdong Ge, Bo Jiang, Yuntian Jiang, Jingsong Liu, Tianhao Liu, Chenyu Xue, Yinyu Ye, and Chuwen Zhang. An enhanced alternating direction method of multipliers-based interior point method for linear and conic optimization. *INFORMS Journal on Computing*, 37(2):338–359, 2025.

[15] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[16] Edmund Eisenberg and David Gale. Consensus of subjective probabilities: The pari-mutuel method. *The Annals of Mathematical Statistics*, 30(1):165–168, 1959.

[17] Ernie Esser, Xiaoqun Zhang, and Tony F Chan. A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. *SIAM Journal on Imaging Sciences*, 3(4):1015–1046, 2010.

[18] Alex Fender. Advances in optimization AI, 2024. Accessed: 2025-10-07.

[19] Henrik A Friberg. CBLIB 2014: A benchmark library for conic mixed-integer and continuous optimization. *Mathematical Programming Computation*, 8:191–214, 2016.

[20] Henrik A Friberg. Projection onto the exponential cone: A univariate root-finding problem. *Optimization Methods and Software*, 38(3):457–473, 2023.

[21] Dongdong Ge, Haodong Hu, Qi Huangfu, Jinsong Liu, Tianhao Liu, Haihao Lu, Jinwen Yang, Yinyu Ye, and Chuwen Zhang. cuPDLP-C, 2024. Accessed: 2025-10-07.

[22] Dongdong Ge, Qi Huangfu, Zizhuo Wang, Jian Wu, and Yinyu Ye. Cardinal Optimizer (COPT) user guide. *arXiv preprint arXiv:2208.14314*, 2022.

[23] Gregory Glockner. Parallel and distributed optimization with Gurobi, 2023. Accessed: 2025-10-07.

[24] William H Greene. *Econometric analysis*. Pearson, 7th edition, 2017.

[25] Richard C. Grinold and Ronald N. Kahn. *Active Portfolio Management: A Quantitative Approach for Producing Superior Returns and Controlling Risk*. McGraw-Hill, 2nd edition, 1999.

[26] Qiushi Han, Chenxi Li, Zhenwei Lin, Caihua Chen, Qi Deng, Dongdong Ge, Huikang Liu, and Yinyu Ye. A low-rank ADMM splitting approach for semidefinite programming. *INFORMS Journal on Computing*, 2025.

[27] Qiushi Han, Zhenwei Lin, Hanwen Liu, Caihua Chen, Qi Deng, Dongdong Ge, and Yinyu Ye. Accelerating low-rank factorization-based semidefinite programming algorithms on GPU. *arXiv preprint arXiv:2407.15049*, 2024.

[28] Fred Hanssmann and Sidney W Hess. A linear programming approach to production and employment scheduling. *Management Science*, 1(1):46–51, 1960.

[29] Oliver Hinder. Worst-case analysis of restarted primal-dual hybrid gradient on totally unimodular linear programs. *Operations Research Letters*, 57:107199, 2024.

[30] Michael Ho, Zheng Sun, and Jack Xin. Weighted elastic net penalized mean-variance portfolio design and computation. *SIAM Journal on Financial Mathematics*, 6(1):1220–1244, 2015.

[31] Yicheng Huang, Wanyu Zhang, Hongpei Li, Dongdong Ge, Huikang Liu, and Yinyu Ye. Restarted primal-dual hybrid conjugate gradient method for large-scale quadratic programming. *INFORMS Journal on Computing*, 2025.

[32] Shucheng Kang, Haoyu Han, Antoine Groudiev, and Heng Yang. Factorization-free orthogonal projection onto the positive semidefinite cone with composite polynomial filtering. *arXiv preprint arXiv:2507.09165*, 2025.

[33] Shucheng Kang, Xin Jiang, and Heng Yang. Local linear convergence of the alternating direction method of multipliers for semidefinite programming under strict complementarity. *arXiv preprint arXiv:2503.20142*, 2025.

[34] Duan Li and Wan-Lung Ng. Optimal dynamic portfolio selection: Multiperiod mean-variance formulation. *Mathematical finance*, 10(3):387–406, 2000.

[35] Jiahan Li. Sparse and stable portfolio selection with parameter uncertainty. *Journal of Business & Economic Statistics*, 33(3):381–392, 2015.

[36] Tianyi Lin, Shiqian Ma, Yinyu Ye, and Shuzhong Zhang. An ADMM-based interior-point method for large-scale linear programming. *Optimization Methods and Software*, 36(2-3):389–424, 2021.

[37] Zhenwei Lin, Zikai Xiong, Dongdong Ge, and Yinyu Ye. A technical note on the implementation and use of PDCS. *arXiv preprint arXiv:2603.15504*, 2026.

[38] Haihao Lu and Jinwen Yang. On the infimal sub-differential size of primal-dual hybrid gradient method and beyond. *arXiv preprint arXiv:2206.12061*, 2022.

[39] Haihao Lu and Jinwen Yang. On the geometry and refined rate of primal–dual hybrid gradient for linear programming. *Mathematical Programming*, pages 1–39, 2024.

[40] Haihao Lu and Jinwen Yang. Restarted Halpern PDHG for Linear Programming. *arXiv preprint arXiv:2407.16144*, 2024.

[41] Haihao Lu and Jinwen Yang. cuPDLP. jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in Julia. *Operations Research*, 2025.

[42] Haihao Lu and Jinwen Yang. A practical and optimal first-order method for large-scale convex quadratic programming. *Mathematical Programming*, 215:771–808, 2026.

[43] Haihao Lu, Jinwen Yang, Haodong Hu, Qi Huangfu, Jinsong Liu, Tianhao Liu, Yinyu Ye, Chuwen Zhang, and Dongdong Ge. cuPDLP-C: A strengthened implementation of cuPDLP for linear programming by C language. *arXiv preprint arXiv:2312.14832*, 2023.

[44] Miles Lubin and Iain Dunning. Computing in operations research using Julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

[45] Ho-Yin Mak, Ying Rong, and Jiawei Zhang. Appointment scheduling with limited distributional information. *Management Science*, 61(2):316–334, 2015.

[46] Hans D Mittelmann. Benchmarks for Optimization Software, 2025. "https://plato.asu.edu/". Accessed: 2025-10-07.

[47] Jean Jacques Moreau. Décomposition orthogonale d'un espace hilbertien selon deux cônes mutuellement polaires. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 255:238–240, 1962.

[48] MOSEK ApS. MOSEK Modeling Cookbook, 2024. https://docs.mosek.com/MOSEKModelingCookbook-a4paper.pdf. Accessed: 2025-10-07.

[49] NVIDIA Corporation. NVIDIA cuDSS, 2025. Accessed: 2025-10-07.

[50] Brendan O'Donoghue. Operator splitting for a homogeneous embedding of the linear complementarity problem. *SIAM Journal on Optimization*, 31(3):1999–2023, 2021.

[51] Brendan O'donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169:1042–1068, 2016.

[52] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):127–239, 2014.

[53] Thomas Pock, Daniel Cremers, Horst Bischof, and Antonin Chambolle. An algorithm for minimizing the Mumford–Shah functional. In *Proceedings of the 2009 International Conference on Computer Vision*, pages 1133–1140. IEEE, 2009.

[54] Edward Rothberg. New options for solving giant LPs, 2024. https://www.gurobi.com/events/new-options-for-solving-giant-lps/. Accessed: 2024-10-07.

[55] Napat Rujeerapaiboon, Daniel Kuhn, and Wolfram Wiesemann. Robust growth-optimal portfolios. *Management Science*, 62(7):2090–2109, 2016.

[56] Vadim I Shmyrev. An algorithm for finding equilibrium in the linear exchange model with fixed budgets. *Journal of Applied and Industrial Mathematics*, 3:505–518, 2009.

[57] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[58] Kasia Swirydowicz, Eric Darve, Wesley Jones, Jonathan Maack, Shaked Regev, Michael A Saunders, Stephen J Thomas, and Slaven Peles. Linear solvers for power grid optimization problems: A review of GPU-accelerated linear solvers. *Parallel Computing*, 111:102870, 2022.

[59] Michael Wagner, Jaroslaw Meller, and Ron Elber. Large-scale linear programming techniques for the design of protein folding potentials. *Mathematical Programming*, 101:301–318, 2004.

[60] Zikai Xiong. Accessible theoretical complexity of the restarted primal-dual hybrid gradient method for linear programs with unique optima. *arXiv preprint arXiv:2410.04043*, 2024.

[61] Zikai Xiong. High-probability polynomial-time complexity of restarted PDHG for linear programming. *arXiv preprint arXiv:2501.00728*, 2025.

[62] Zikai Xiong and Robert M Freund. Computational guarantees for restarted PDHG for LP based on "limiting error ratios" and LP sharpness. *arXiv preprint arXiv:2312.14774*, 2023.

[63] Zikai Xiong and Robert M Freund. On the relation between LP sharpness and limiting error ratio and complexity implications for restarted PDHG. *arXiv preprint arXiv:2312.13773*, 2023.

[64] Zikai Xiong and Robert M Freund. The role of level-set geometry on the performance of PDHG for conic linear optimization. *arXiv preprint arXiv:2406.01942*, 2024.

[65] Yinyu Ye. A path to the Arrow–Debreu competitive market equilibrium. *Mathematical Programming*, 111(1):315–348, 2008.

Appendices

A Projection onto rescaled second-order cone and exponential cone

In this section, we provide the proofs of Theorems 1 and 2, about the projection onto the rescaled second-order cone $\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1}$ and the rescaled exponential cone $\mathbf{D}\mathcal{K}_{\text{exp}}$.

A.1 Projection onto rescaled second-order cone

Proof of Theorem 1. Let $\mathbf{D}$ be the diagonal rescaling matrix, the projection onto $\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1}$ is the solution of the following problem:

$$\text{Proj}_{\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1}}\{(t,\mathbf{x})\} = \underset{(s,\mathbf{y})\in\mathbb{R}^{n+1}}{\text{argmin}} \frac{1}{2}\|(t,\mathbf{x})-(s,\mathbf{y})\|^2, \quad \text{s.t.} \left\|\hat{\mathbf{D}}^{-1}\mathbf{y}\right\|^2 \le s^2, \ s\ge 0\ , \tag{9}$$

where $(t,\mathbf{x})\in\mathbb{R}^{n+1}$, $\hat{\mathbf{D}} = \text{diag}(\hat{\mathbf{d}}_2,\cdots,\hat{\mathbf{d}}_{n+1})$ with $\hat{\mathbf{d}}_i = \mathbf{d}_i/\mathbf{d}_1$ and $\mathbf{d}_i$ is the i-th entry in the diagonal of $\mathbf{D}$.

We first consider the simplest case: when $(s,\mathbf{y})$ is the degenerate solution $(0,\mathbf{0})$. It should be noted that the projection onto the convex cone $\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1}$ is $(0,\mathbf{0})$ if and only if $-(t,\mathbf{x})$ lies in $(\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1})^*$. Because $(\mathbf{D}\mathcal{K}_{\text{soc}}^{n+1})^* = \mathbf{D}^{-1}\mathcal{K}_{\text{soc}}^{n+1}$, it implies that when $\|\hat{\mathbf{D}}\mathbf{x}\|\le -t$, the projection $(s,\mathbf{y})$ is $(0,\mathbf{0})$. This proves the first case.

Now we introduce the multipliers $\lambda\ge 0$ and $\mu\ge 0$ for the two constraints of (9), then the Lagrangian is $\mathcal{L}(\mathbf{y},s;\lambda,\mu) = \frac{1}{2}\|\mathbf{y}-\mathbf{x}\|^2 + \frac{1}{2}(s-t)^2 + \lambda\big(\|\hat{\mathbf{D}}^{-1}\mathbf{y}\|^2 - s^2\big) + \mu(-s)$. Thus, the KKT conditions of (9) are as follows:

$$\begin{aligned} &\|\hat{\mathbf{D}}^{-1}\mathbf{y}\|^2 - s^2 \le 0, \mathbf{y}-\mathbf{x}+2\lambda\hat{\mathbf{D}}^{-2}\mathbf{y} = 0, (s-t)-2\lambda s-\mu = 0,\\ &\lambda\left(\|\hat{\mathbf{D}}^{-1}\mathbf{y}\|^2 - s^2\right) = 0, \mu s = 0, s\ge 0, \lambda\ge 0, \mu\ge 0, \end{aligned} \tag{10}$$

It should be noted that Slater's condition holds for (9) as there exist strictly feasible solutions, so any nonzero $(s,\mathbf{y})$ satisfying (10) is an optimal solution pair for (9).

If $(t,\mathbf{x})$ is already in the rescaled cone, i.e., $\|\hat{\mathbf{D}}^{-1}\mathbf{x}\|^2\le t^2$, then $(s,\mathbf{y}) = (t,\mathbf{x})$ is an optimal solution for (9) because it is feasible and has the smallest possible objective value 0. This proves the second case.

If $\|\hat{\mathbf{D}}^{-1}\mathbf{x}\|^2 > t^2$, since $(t,\mathbf{x})$ is not in the rescaled cone, the projected solution $(s,\mathbf{y})$ must lie in the boundary of the cone. In other words, $\|\hat{\mathbf{D}}^{-1}\mathbf{y}\| = s$. If $t=0$, then the optimality conditions (10) become

$$(\mathbf{y}-\mathbf{x})+2\lambda\hat{\mathbf{D}}^{-2}\mathbf{y} = 0\ ,\ s-2\lambda s = \mu\ ,\ s\mu = 0\ ,\ \left\|\hat{\mathbf{D}}^{-1}\mathbf{y}\right\|^2 = s^2,\ s\ge 0\ ,\ \lambda\ge 0\ ,\ \mu\ge 0\ . \tag{11}$$

$$\mathbf{y} = (\mathbf{I}_{n\times n}+\hat{\mathbf{D}}^{-2})^{-1}\mathbf{x}\ ,\ s = \left\|(\hat{\mathbf{D}}+\hat{\mathbf{D}}^{-1})^{-1}\mathbf{x}\right\|\ ,\ \lambda = \frac{1}{2}\ ,\ \mu = 0 \tag{12}$$

satisfy the above system. Therefore, when $t = 0$ and $\left\|\hat{\mathbf{D}}^{-1}\mathbf{x}\right\| > 0$, (12) gives the solution s and $\mathbf{y}$ for (9). This proves the third case.

If $t > 0$, let $\mu = 0$ and then the optimality conditions (10) become

$$(\mathbf{y} - \mathbf{x}) + 2\lambda\hat{\mathbf{D}}^{-2}\mathbf{y} = 0 \ , \ (1 - 2\lambda)s = t \ , \ \left\|\hat{\mathbf{D}}^{-1}\mathbf{y}\right\|^2 = s^2 \ , \ s \geq 0, \ \lambda \geq 0 \ . \tag{13}$$

If there exist $\mathbf{y}$, s and $\lambda \in (0, \frac{1}{2})$ satisfying (13), then $\mathbf{y}$ and s can be computed by the first and second equations:

$$\mathbf{y} = (\mathbf{I} + 2\lambda\hat{\mathbf{D}}^{-2})^{-1}\mathbf{x} \ , \quad s = (1 - 2\lambda)^{-1}t \geq 0 \tag{14}$$

Substituting them into the third equation of (13) yields:

$$f(\lambda) := \|(\hat{\mathbf{D}} + 2\lambda\hat{\mathbf{D}}^{-1})^{-1}\mathbf{x}\|^2 - (1 - 2\lambda)^{-2}t^2 = 0 \ . \tag{15}$$

Note that there must exist a root $\lambda \in (0, \frac{1}{2})$ of the above equation $f(\lambda) = 0$ because f is continuous on $(0, \frac{1}{2})$, while $\lim_{\lambda \to 0+} f(\lambda) > 0$ (since $\|\hat{\mathbf{D}}^{-1}\mathbf{x}\|^2 > t^2$) and $\lim_{\lambda \to \frac{1}{2}} f(\lambda) < 0$ (since $t > 0$). Therefore, let λ be a root of (15) and then (14) gives the solution $\mathbf{y}$ and s.

Similarly, if $t < 0$ and $\|\hat{\mathbf{D}}\mathbf{x}\| > -t$ (the case $\|\hat{\mathbf{D}}\mathbf{x}\| \leq -t$ has been discussed in the beginning), then there exists $\lambda > \frac{1}{2}$ that is a root of $f(\lambda) = 0$. The existence of such a root is due to $\|\hat{\mathbf{D}}\mathbf{x}\| > -t$. Then (14) gives the solution $\mathbf{y}$ and s based on the root λ. Now the fourth case is proven. □

A.2 Projection onto rescaled exponential cone

To compute the projection of $\mathbf{v}_0$ onto the diagonally rescaled exponential cone $\mathbf{D}\mathcal{K}_{\text{exp}}$ and its dual cone $\mathbf{D}^{-1}\mathcal{K}^*_{\text{exp}}$, we first recall the definitions of $\mathcal{K}_{\text{exp}}$ and $\mathcal{K}^*_{\text{exp}}$:

$$\mathcal{K}_{\text{exp}} = \left\{(r, s, t) \in \mathbb{R}^3 \mid s > 0, \ t \geq s \cdot \exp\left(\frac{r}{s}\right)\right\} \bigcup \left\{(r, s, t) \in \mathbb{R}^3 \mid s = 0, \ t \geq 0, \ r \leq 0\right\} \tag{16}$$

$$\mathcal{K}^*_{\text{exp}} = \left\{(r, s, t) \in \mathbb{R}^3 \mid r < 0, \ e \cdot t \geq -r \cdot \exp\left(\frac{s}{r}\right)\right\} \bigcup \left\{(r, s, t) \in \mathbb{R}^3 \mid r = 0, \ s \geq 0, \ t \geq 0\right\} \tag{17}$$

and then we present the formal statement and proof of Theorem 2. To compute the projection onto the rescaled cone, we need to use the following Moreau's decomposition theorem [47]:

Lemma 1. *Let $\mathcal{K}$ be a closed convex cone in $\mathbb{R}^n$ and $\mathcal{K}^*$ be its dual cone. For any $\mathbf{v}_p$ and $\mathbf{v}_d \in \mathbb{R}^n$, the following statements are equivalent. (a) $\mathbf{v}_p \in \mathcal{K}$, $\mathbf{v}_d \in -\mathcal{K}^*$ and $\langle \mathbf{v}_p, \mathbf{v}_d \rangle = 0$. (b) $\mathbf{v}_p = \text{Proj}_{\mathcal{K}}(\mathbf{v}_p + \mathbf{v}_d)$ and $\mathbf{v}_d = \text{Proj}_{-\mathcal{K}^*}(\mathbf{v}_p + \mathbf{v}_d)$.*

Therefore, the projection problem essentially reduces to computing $\mathbf{v}_p$ and $\mathbf{v}_d$ such that

$$\mathbf{v}_0 = \mathbf{v}_\text{p} + \mathbf{v}_\text{d} \ , \ \mathbf{v}_\text{p} \in \mathbf{D}\mathcal{K}_{\text{exp}} \ , \ \mathbf{v}_\text{d} \in -\mathbf{D}^{-1}\mathcal{K}^*_{\text{exp}} \ , \ \langle \mathbf{v}_\text{p}, \mathbf{v}_\text{d} \rangle = 0 \ . \tag{18}$$

Since the projection of $\mathbf{v}_0$ onto the rescaled cone is unique, the corresponding $\mathbf{v}_p$ and $\mathbf{v}_d$ satisfying (18) are also unique. Now, we are ready to present the formal statement of Theorem 2:

Theorem 3. *For any* $\mathbf{v}_0 = (r_0, s_0, t_0) \in \mathbb{R}^3$, *the corresponding* $\mathbf{v}_p$ *and* $\mathbf{v}_d$ *that satisfy* (18) *can be computed as follows:*

1. *If* $\mathbf{v}_0 \in \mathbf{D}\mathcal{K}_{\exp}$, *then* $\mathbf{v}_p = \mathbf{v}_0$ *and* $\mathbf{v}_d = \mathbf{0}_{3\times 1}$.
2. *If* $\mathbf{v}_0 \in -\mathbf{D}^{-1}\mathcal{K}_{\exp}^*$, *then* $\mathbf{v}_p = \mathbf{0}_{3\times 1}$ *and* $\mathbf{v}_d = \mathbf{v}_0$.
3. *If* $r_0 \le 0$ *and* $s_0 \le 0$, *then* $\mathbf{v}_p = \left(r_0, 0, t_0^+\right)$ *and* $\mathbf{v}_d = \left(0, s_0, -t_0^-\right)$.
4. *Otherwise, solve the following root-finding problem:*

$$h(\rho) := d_t \exp(\rho) s_p(\rho) - d_t^{-1} \exp(-\rho) r_d(\rho) - t_0 = 0 \ , \tag{19}$$

$$s_p(\rho) = \frac{s_0 d_r^{-2} d_s - d_r^{-1} r_0 (1-\rho)}{d_s^2 d_r^{-2} + \rho(\rho-1)} \ \text{and} \ r_d(\rho) = \frac{d_s^2 d_r^{-1}\left(r_0 - d_s^{-1} d_r \rho s_0\right)}{d_s^2 d_r^{-2} + \rho(\rho-1)}. \tag{20}$$

for ρ *in interval* (l_ρ, u_ρ), *where*

$$(l_\rho, u_\rho) = \begin{cases} (\min\{a_3, a_4\}, \max\{a_3, a_4\}), & r_0 > 0, \ s_0 > 0 \quad (21a) \\ (-\infty, a_3), & r_0 \le 0, \ s_0 > 0 \quad (21b) \\ (a_4, +\infty), & r_0 > 0, \ s_0 \le 0 \quad (21c) \end{cases}$$

and

$$a_3 = \frac{r_0 d_s}{s_0 d_r} \ , \ a_4 = 1 - \frac{s_0 d_s}{r_0 d_r} \ . \tag{22}$$

Then the solution $(\mathbf{v}_p, \mathbf{v}_d)$ *to system* (18) *is given by*

$$\mathbf{v}_p = \left(d_r \rho, d_s, d_t \exp(\rho)\right) s_p(\rho) \ \text{ and } \ \mathbf{v}_d = \left(d_r^{-1}, (1-\rho) d_s^{-1}, -d_t^{-1} \exp(-\rho)\right) r_d(\rho) \ . \tag{23}$$

Theorem 3 provides the solution to (18). The first three cases of Theorem 3 are simple cases. In practice, we first check the first three cases, followed by solving the root-finding problem of the nonlinear equation (19). Our strategy is built upon the projection algorithm for the unscaled exponential cone [20], yet the diagonal scaling matrix introduces a non-trivial coupling that complicates the root-finding analysis.

Remark 1. Projecting onto the rescaled dual exponential cone ($\text{Proj}_{\mathbf{D}\mathcal{K}_{\exp}^*}(\mathbf{v}_0)$) is equivalent to the following problem:

$$\mathbf{v}_{\mathrm{d}} = \underset{\mathbf{v}}{\text{argmin}} \ \|\mathbf{v} - \mathbf{v}_0\|^2 \quad \text{s. t.} \ \ \mathbf{v} \in \mathbf{D}\mathcal{K}_{\exp}^* \ . \tag{24}$$

This task can be completed by first projecting onto the primal cone, specifically, $\mathbf{v}_{\mathrm{p}} = \text{Proj}_{\mathbf{D}^{-1}\mathcal{K}_{\exp}}(-\mathbf{v}_0)$, and the solution can then be recovered via $\mathbf{v}_{\mathrm{d}} = \mathbf{v}_0 + \mathbf{v}_{\mathrm{p}}$.

Before proving Theorem 3, we first present two preparatory lemmas.

Lemma 2. *Let* $\nu \in (0, \frac{1}{2}], \beta > 0$ *and denote*

$$a_1 = \left(1 - \sqrt{1-4\nu^2}\right)/2, a_2 = \left(1 + \sqrt{1-4\nu^2}\right)/2, a_3 = \beta\nu, a_4 = 1 - \nu/\beta, I = (\min\{a_3, a_4\}, \max\{a_3, a_4\}).$$

If $a_3 > a_4$, then $I \subseteq (-\infty, a_1) \cup (a_2, \infty)$. If $a_4 > a_3$, then $I \subseteq (a_1, a_2)$.

Proof. Recall $a_1 + a_2 = 1$ and $a_1 a_2 = \nu^2$. Consider $a_4 - a_3 = 1 - \frac{\nu}{\beta} - \beta\nu = -\frac{1}{\beta}\big(\nu\beta^2 - \beta + \nu\big)$. The quadratic function $q(\beta) := \nu\beta^2 - \beta + \nu$ has two positive roots $\beta_1 := \frac{a_1}{\nu}$ and $\beta_2 := \frac{a_2}{\nu}$, since $a_1 + a_2 = 1$ and $a_1 a_2 = \nu^2$. Because $\beta > 0$, $a_3 > a_4$ is equivalent to $\beta \in (0, \beta_1) \cup (\beta_2, \infty)$ and $a_4 > a_3$ is equivalent to $\beta \in (\beta_1, \beta_2)$.

Case 1: $a_3 > a_4$. If $\beta \le \beta_1$, then $a_3 = \beta\nu \le a_1$ and $a_4 = 1 - \frac{\nu}{\beta} < 1 - \frac{\nu}{\beta_1} = 1 - \frac{\nu}{a_1/\nu} = 1 - \frac{\nu^2}{a_1} = 1 - (1 - a_1) = a_1$, so $I = (a_4, a_3) \subset (-\infty, a_1)$. If $\beta \ge \beta_2$, then $a_3 = \beta\nu \ge a_2$ and $a_4 = 1 - \frac{\nu}{\beta} \ge 1 - \frac{\nu}{\beta_2} = 1 - \frac{\nu}{a_2/\nu} = 1 - \frac{\nu^2}{a_2} = 1 - a_1 = a_2$, so $I = (a_4, a_3) \subset [a_2, \infty)$. In either subcase $I \nsubseteq (a_1, a_2)$, and in particular $a_1, a_2 \notin I$.

Case 2: $a_4 > a_3$. Here $\beta \in (\beta_1, \beta_2)$, so $a_1 < \beta\nu = a_3$ and $a_4 = 1 - \frac{\nu}{\beta} < 1 - \frac{\nu}{\beta_2} = 1 - a_1 = a_2$. Thus $a_1 < a_3 < a_4 < a_2$, hence $I = (a_3, a_4) \subset (a_1, a_2)$ (and therefore $a_1, a_2 \notin I$). When $\nu = \frac{1}{2}$, the interval (a_1, a_2) collapses to $\{1/2\}$ and the strict branch $a_4 > a_3$ is empty; the stated inclusion remains valid. □

Lemma 3. *Suppose $r_0 > 0$ or $s_0 > 0$ and $(l_\rho, u_\rho) \neq \varnothing$. Then for every $\rho \in (l_\rho, u_\rho)$ (defined in (21)), the quantities $s_p(\rho)$ and $r_d(\rho)$ (defined in (20)) satisfy $s_p(\rho) > 0$ and $r_d(\rho) > 0$.*

Proof. For simplicity, set $\tau := d_s/d_r > 0$. Then $a_3 = \frac{r_0}{s_0}\tau$ and $a_4 = 1 - \frac{s_0}{r_0}\tau$, and introduce

$$\tilde{d}(\rho) = (\rho - \tfrac{1}{2})^2 + (\tau^2 - \tfrac{1}{4}), \quad \tilde{s}(\rho) = s_0\tau - r_0 + r_0\rho, \quad \tilde{r}(\rho) = \tau r_0 - \rho s_0. \tag{25}$$

A direct simplification of (20) gives $s_{\mathrm{p}}(\rho) = d_r^{-1}\frac{\tilde{s}(\rho)}{\tilde{d}(\rho)}$ and $r_{\mathrm{d}}(\rho) = d_s\frac{\tilde{r}(\rho)}{\tilde{d}(\rho)}$. Since the d_r^{-1} and d_s are positive, it suffices to show that for $\rho \in (l_\rho, u_\rho)$, the fractions $\tilde{s}(\rho)/\tilde{d}(\rho)$ and $\tilde{r}(\rho)/\tilde{d}(\rho)$ are positive.
Case 1. $r_0 = 0$, $s_0 > 0$. Here $\tilde{s}(\rho) = s_0\tau > 0$ and $\tilde{r}(\rho) = -\rho s_0$. For $\rho < a_3 = 0$ we have $\tilde{r}(\rho) > 0$, and since $\rho < 0$, $\tilde{d}(\rho) = (\rho - \frac{1}{2})^2 + (\tau^2 - \frac{1}{4}) > \tau^2 - 1/4 + 1/4 = \tau^2 > 0$. Thus all signs match and positivity of the fractions $\tilde{s}(\rho)/\tilde{d}(\rho)$ and $\tilde{r}(\rho)/\tilde{d}(\rho)$ holds.
Case 2. $r_0 > 0$, $s_0 = 0$. Here $\tilde{r}(\rho) = \tau r_0 > 0$ and $\tilde{s}(\rho) = r_0(\rho - 1)$. For $\rho > a_4 = 1$, we have $\tilde{s}(\rho) > 0$. Also $\tilde{d}(\rho) > 0$ for $\rho > 1$ since $(\rho - \frac{1}{2})^2 \ge 1/4$. Thus positivity holds.
Case 3. $r_0 < 0$, $s_0 > 0$. For $\rho < a_3 = (r_0/s_0)\tau < 0$, we have $\tilde{r}(\rho) = \tau r_0 - \rho s_0 > \tau r_0 - a_3 s_0 = 0$, and also $\tilde{s}(\rho) = s_0\tau - r_0 + r_0\rho > s_0\tau - r_0 > 0$ since $r_0 < 0 < s_0$. Furthermore $\rho < 0$ implies $\tilde{d}(\rho) > 0$. Thus positivity holds.
Case 4. $r_0 > 0$, $s_0 < 0$. For $\rho > a_4 = 1 - (s_0/r_0)\tau > 1$, we compute $\tilde{s}(\rho) = s_0\tau - r_0 + r_0\rho > s_0\tau - r_0 + r_0(1 - (s_0/r_0)\tau) = 0$, and $\tilde{r}(\rho) = \tau r_0 - \rho s_0 > \tau r_0 - s_0 > 0$, since $s_0 < 0 < r_0$. Also $\rho > 1$ guarantees $\tilde{d}(\rho) > 0$. Thus positivity holds.
Case 5. $r_0 > 0$, $s_0 > 0$. This is the most complicated case. We will discuss based on the value of τ.

If $\tau > 1/2$, then $\tilde{d}(\rho) > 0$. Moreover, $a_3 - a_4 = \tau\left(\frac{r_0}{s_0} + \frac{s_0}{r_0}\right) - 1 \ge 2\tau - 1 > 0$, so $(l_\rho, u_\rho) = (a_4, a_3)$. Hence $\tilde{s}(\rho) > 0$, $\tilde{r}(\rho) > 0$, and $\tilde{d}(\rho) > 0$ on (l_ρ, u_ρ), which gives $s_p(\rho) > 0$ and $r_d(\rho) > 0$. If $\tau = 1/2$, then $\tilde{d}(\rho) \ge 0$, with equality only at $\rho = \frac{1}{2}$. By Lemma 2 (with $\nu = \frac{1}{2}$, $\beta = r_0/s_0$), if $a_3 > a_4$ then $(a_4, a_3) \subset (-\infty, \frac{1}{2}) \cup (\frac{1}{2}, \infty)$, so $\frac{1}{2} \notin (l_\rho, u_\rho)$ and $\tilde{d}(\rho) > 0$ on (l_ρ, u_ρ); if $a_4 \ge a_3$, then (l_ρ, u_ρ) would be empty, contradicting the assumptions. Thus we must be

in the former situation, where again $\tilde{s}(\rho), \tilde{r}(\rho) > 0$ and positivity holds. Finally, if $\tau \in (0, \frac{1}{2})$, let $a_1 = \frac{1-\sqrt{1-4\tau^2}}{2}$ and $a_2 = \frac{1+\sqrt{1-4\tau^2}}{2}$, so $\tilde{d}(\rho) < 0$ exactly for $\rho \in (a_1, a_2)$ and $\tilde{d}(\rho) > 0$ otherwise. By Lemma 2, if $a_3 > a_4$, then $(l_\rho, u_\rho) = (a_4, a_3) \subset (-\infty, a_1) \cup (a_2, \infty)$, where $\tilde{d}(\rho) > 0$. On this interval $\tilde{s}(\rho), \tilde{r}(\rho) > 0$, so the fractions $\tilde{s}(\rho)/\tilde{d}(\rho)$ and $\tilde{r}(\rho)/\tilde{d}(\rho)$ are positive. Otherwise, if $a_4 > a_3$, then $(l_\rho, u_\rho) = (a_3, a_4) \subset (a_1, a_2)$, where $\tilde{d}(\rho) < 0$. On this interval $\tilde{s}(\rho), \tilde{r}(\rho) < 0$, so the fractions $\tilde{s}(\rho)/\tilde{d}(\rho)$ and $\tilde{r}(\rho)/\tilde{d}(\rho)$ are again positive.

In all subcases we obtain $\tilde{s}(\rho)/\tilde{d}(\rho) > 0$ and $\tilde{r}(\rho)/\tilde{d}(\rho) > 0$ for $\rho \in (l_\rho, u_\rho)$. Since d_r^{-1} and d_s^2 are positive, this proves $s_p(\rho) > 0$ and $r_d(\rho) > 0$. □

Now, we are ready to prove Theorem 3.

Proof of Theorem 3. By Lemma 1, the desired pair $(\mathbf{v}_\mathrm{p}, \mathbf{v}_\mathrm{d})$ is the unique solution of (18). We use the convention $x^+ = \max\{x, 0\}$ and $x^- = -\min\{x, 0\}$, so $x = x^+ - x^-$ and $x^+ x^- = 0$.

Cases 1–2. These follow directly from Lemma 1 and uniqueness of projection.

Case 3 ($r_0 \le 0$ and $s_0 \le 0$) under the present convention. Set $\mathbf{v}_\mathrm{p} = (r_0,\, 0,\, t_0^+)$ and $\mathbf{v}_\mathrm{d} = (0,\, s_0,\, -t_0^-)$. Then $\mathbf{v}_\mathrm{p} + \mathbf{v}_\mathrm{d} = \mathbf{v}_0$ because $t_0 = t_0^+ - t_0^-$. Moreover $\mathbf{D}^{-1}\mathbf{v}_\mathrm{p} = (r_0/d_r,\, 0,\, t_0^+/d_t) \in \mathcal{K}_\mathrm{exp}$ (the $s = 0$ branch with $r \le 0$ and $t \ge 0$), and $-\mathbf{D}\,\mathbf{v}_\mathrm{d} = (0,\, -d_s s_0,\, d_t t_0^-) \in \mathcal{K}_\mathrm{exp}^*$ (the $r = 0$ branch with $s \ge 0$ and $t \ge 0$, since $s_0 \le 0$ and $-t_0^- \le 0$). Finally, $\langle \mathbf{v}_\mathrm{p}, \mathbf{v}_\mathrm{d} \rangle = r_0 \cdot 0 + 0 \cdot s_0 - t_0^+\, t_0^- = 0$, because $t_0^+ t_0^- = 0$. Hence (18) holds.

Case 4 (general case). Here $\mathbf{v}_0 \notin \mathbf{D}\mathcal{K}_\mathrm{exp}$ and $\mathbf{v}_0 \notin -\mathbf{D}^{-1}\mathcal{K}_\mathrm{exp}^*$, and at least one of $r_0 > 0$ or $s_0 > 0$ holds (the complement of Case 3). We show that the formulas (20) and (23) with $\rho \in (l_\rho, u_\rho)$ yield the unique Moreau pair.

Step 1 (membership and orthogonality). For any $\rho \in \mathbb{R}$ and positive scalars $s_\mathrm{p}, r_\mathrm{d}$ define $\mathbf{v}_\mathrm{p} = \big(d_r\rho,\, d_s,\, d_t e^\rho\big)\, s_\mathrm{p}$ and $\mathbf{v}_\mathrm{d} = \big(d_r^{-1},\, (1-\rho) d_s^{-1},\, -d_t^{-1} e^{-\rho}\big)\, r_\mathrm{d}$. Then $\mathbf{D}^{-1}\mathbf{v}_\mathrm{p} = s_\mathrm{p}\,(\rho, 1, e^\rho) \in \mathcal{K}_\mathrm{exp}$ and $-\mathbf{D}\,\mathbf{v}_\mathrm{d} = r_\mathrm{d}\,(-1, \rho - 1, e^{-\rho}) \in \mathcal{K}_\mathrm{exp}^*$ whenever $s_\mathrm{p} > 0$ and $r_\mathrm{d} > 0$, each with equality in the defining inequalities. Moreover, $\langle \mathbf{v}_\mathrm{p}, \mathbf{v}_\mathrm{d} \rangle = s_\mathrm{p} r_\mathrm{d}\,\big(\rho + (1-\rho) - 1\big) = 0$, so orthogonality is automatic, independently of ρ, s_p, and r_d. By Lemma 3, for every $\rho \in (l_\rho, u_\rho)$ we have $s_\mathrm{p}(\rho) > 0$ and $r_\mathrm{d}(\rho) > 0$. Therefore $\mathbf{v}_\mathrm{p} \in \mathbf{D}\mathcal{K}_\mathrm{exp}$ and $\mathbf{v}_\mathrm{d} \in -\mathbf{D}^{-1}\mathcal{K}_\mathrm{exp}^*$ for all $\rho \in (l_\rho, u_\rho)$, and $\langle \mathbf{v}_\mathrm{p}, \mathbf{v}_\mathrm{d} \rangle = 0$.

Step 2 (matching the r- and s-coordinates). A direct substitution into the r- and s-components of $\mathbf{v}_\mathrm{p} + \mathbf{v}_\mathrm{d} = \mathbf{v}_0$ gives the equations: $\underbrace{d_r \rho\, s_\mathrm{p}(\rho)}_{r \text{ from } \mathbf{v}_\mathrm{p}} + \underbrace{d_r^{-1}\, r_\mathrm{d}(\rho)}_{r \text{ from } \mathbf{v}_\mathrm{d}} = r_0$, and $\underbrace{d_s\, s_\mathrm{p}(\rho)}_{s \text{ from } \mathbf{v}_\mathrm{p}} + \underbrace{d_s^{-1}(1-\rho)\, r_\mathrm{d}(\rho)}_{s \text{ from } \mathbf{v}_\mathrm{d}} = s_0$, Furthermore, with s_p and $r_\mathrm{d}(\rho)$ as in (20), it can be checked that $\mathbf{v}_\mathrm{p}$ and $\mathbf{v}_\mathrm{d}$ match $\mathbf{v}_0$ in the r- and s-coordinates.

Step 3 (matching the t-coordinate). Furthermore, the remaining t-coordinate equation $\big(\mathbf{v}_\mathrm{p} + \mathbf{v}_\mathrm{d}\big)_t = t_0$ is precisely equivalent to the scalar equation (19). The only possible singularities of $h(\rho)$ occur when the denominator is zero, i.e.,

$$d_s^2 d_r^{-2} + \rho(\rho - 1) = \big(\rho - \tfrac{1}{2}\big)^2 + \big((d_s/d_r)^2 - \tfrac{1}{4}\big) = 0,$$

i.e., at $\rho = a_1, a_2 \in (0,1)$ (defined in the above proof of Lemma 3) when $d_s/d_r < \frac{1}{2}$. By the proof of Lemma 3, the open interval (l_ρ, u_ρ) used in (21) never contains a_1 or a_2. Hence h is continuous on (l_ρ, u_ρ). We then show that $h(\rho)$ attains opposite signs at the endpoints (or limits) of (l_ρ, u_ρ) in each branch of (21).

(i) $r_0 > 0$, $s_0 > 0$. Here $(l_\rho, u_\rho) = (\min\{a_3, a_4\}, \max\{a_3, a_4\})$. A direct simplification of (20) shows $r_{\mathrm{d}}(a_3) = 0$ and $s_{\mathrm{p}}(a_4) = 0$, and consequently

$$h(a_3) = \frac{s_0 d_{\mathrm{r}}^{-1} d_{\mathrm{s}} - r_0(1-a_3)}{d_{\mathrm{s}}^2 d_{\mathrm{r}}^{-1} + d_{\mathrm{r}} a_3(a_3-1)} d_t \exp(a_3) - t_0 = \frac{s_0 d_{\mathrm{r}}^{-1} d_{\mathrm{s}} - r_0(1-\frac{r_0 d_{\mathrm{s}}}{s_0 d_{\mathrm{r}}})}{d_{\mathrm{s}}^2 d_{\mathrm{r}}^{-1} + d_{\mathrm{r}} \frac{r_0 d_{\mathrm{s}}}{s_0 d_{\mathrm{r}}}(\frac{r_0 d_{\mathrm{s}}}{s_0 d_{\mathrm{r}}} - 1)} d_t \exp(a_3) - t_0 = \frac{s_0}{d_{\mathrm{s}}} d_t e^{a_3} - t_0 \tag{26}$$

and

$$\begin{aligned} h(a_4) &= -\frac{r_0 - d_{\mathrm{s}}^{-1} d_{\mathrm{r}} \left(1 - \frac{s_0 d_{\mathrm{s}}}{r_0 d_{\mathrm{r}}}\right) s_0}{d_{\mathrm{r}}^{-1} + d_{\mathrm{s}}^{-2} d_{\mathrm{r}} \left(1 - \frac{s_0 d_{\mathrm{s}}}{r_0 d_{\mathrm{r}}}\right)(-\frac{s_0 d_{\mathrm{s}}}{r_0 d_{\mathrm{r}}})} d_t^{-1} \exp\left(\frac{s_0 d_{\mathrm{s}}}{r_0 d_{\mathrm{r}}} - 1\right) - t_0 \\ &= \frac{d_{\mathrm{s}}^{-1} d_{\mathrm{r}} \left(1 - \frac{s_0 d_{\mathrm{s}}}{r_0 d_{\mathrm{r}}}\right) s_0 - r_0}{d_{\mathrm{r}}^{-1} + d_{\mathrm{s}}^{-1} \left(1 - \frac{s_0 d_{\mathrm{s}}}{r_0 d_{\mathrm{r}}}\right)\left(-\frac{s_0}{r_0}\right)} d_t^{-1} \exp\left(\frac{s_0 d_{\mathrm{s}}}{r_0 d_{\mathrm{r}}} - 1\right) - t_0 = -d_{\mathrm{r}} r_0 d_t^{-1} \exp\left(\frac{s_0 d_{\mathrm{s}}}{r_0 d_{\mathrm{r}}} - 1\right) - t_0 \,. \end{aligned} \tag{27}$$

Since $\mathbf{v}_0 \notin \mathbf{D}\mathcal{K}_{\exp}$ with $s_0 > 0$, the membership test for $(r_0/d_r, s_0/d_s, t_0/d_t)$ in $\mathcal{K}_{\exp}$ fails, i.e., $t_0 < (s_0/d_s)\, d_t\, e^{a_3}$, so $h(a_3) > 0$. Likewise, since $\mathbf{v}_0 \notin -\mathbf{D}^{-1}\mathcal{K}_{\exp}^*$ with $r_0 > 0$, the test for $(-d_r r_0, -d_s s_0, -d_t t_0) \in \mathcal{K}_{\exp}^*$ fails on the $r < 0$ branch, i.e., $t_0 > -d_r r_0\, d_t^{-1} \exp(\frac{d_s s_0}{d_r r_0} - 1)$, so $h(a_4) < 0$. In particular $a_3 \neq a_4$, hence (l_ρ, u_ρ) is a nonempty open interval and $h(\rho)$ changes sign across it.

(ii) $r_0 \leq 0$, $s_0 > 0$. Here $(l_\rho, u_\rho) = (-\infty, a_3)$ (with $a_3 \leq 0$). As $\rho \to -\infty$, the term $-d_t^{-1} e^{-\rho} r_{\mathrm{d}}(\rho)$ dominates with negative sign and $h(\rho) \to -\infty$. At $\rho = a_3$, the same calculation as in (26) yields $h(a_3) = (s_0/d_s)\, d_t\, e^{a_3} - t_0 > 0$ because $\mathbf{v}_0 \notin \mathbf{D}\mathcal{K}_{\exp}$ with $s_0 > 0$. Thus a sign change occurs on $(-\infty, a_3)$.

(iii) $r_0 > 0$, $s_0 \leq 0$. Here $(l_\rho, u_\rho) = (a_4, \infty)$ (with $a_4 \geq 1$). As $\rho \to +\infty$, the term $d_t e^{\rho} s_{\mathrm{p}}(\rho)$ dominates positively and $h(\rho) \to +\infty$. At $\rho = a_4$, the same dual-side evaluation as in (27) yields $h(a_4) < 0$. Hence a sign change occurs on (a_4, ∞).

By the intermediate value theorem, in each branch there exists $\rho^\star \in (l_\rho, u_\rho)$ with $h(\rho^\star) = 0$.

Step 4. With this unique $\rho^\star \in (l_\rho, u_\rho)$ and $s_{\mathrm{p}} = s_{\mathrm{p}}(\rho^\star)$, $r_{\mathrm{d}} = r_{\mathrm{d}}(\rho^\star)$ from (20), the definitions (23) give $\mathbf{v}_{\mathrm{p}} \in \mathbf{D}\mathcal{K}_{\exp}$, $\mathbf{v}_{\mathrm{d}} \in -\mathbf{D}^{-1}\mathcal{K}_{\exp}^*$, $\mathbf{v}_{\mathrm{p}} + \mathbf{v}_{\mathrm{d}} = \mathbf{v}_0$, and $\langle \mathbf{v}_{\mathrm{p}}, \mathbf{v}_{\mathrm{d}} \rangle = 0$. By Lemma 1, $\mathbf{v}_{\mathrm{p}} = \mathrm{Proj}_{\mathbf{D}\mathcal{K}_{\exp}}(\mathbf{v}_0)$ and $\mathbf{v}_{\mathrm{d}} = \mathrm{Proj}_{-\mathbf{D}^{-1}\mathcal{K}_{\exp}^*}(\mathbf{v}_0)$, completing the proof. □

B Supplementary details of experiments

In this section, we present the additional details for the experiments in Sections 5.2, 5.3, and 5.4, including the conic program reformulations of the original problems and the details of the data generation.

B.1 Fisher market equilibrium problem (Section 5.2)

Fisher market equilibrium instances [16, 65] can be written as follows:

$$\min_{\mathbf{X}\in\mathbb{R}^{m\times n}} -\sum_{i\in[m]} \mathbf{w}_i \left(\log\left(\sum_{j\in[n]} \mathbf{U}_{ij}\mathbf{X}_{ij}\right)\right) \quad \text{s.t.} \sum_{i\in[m]} \mathbf{X}_{ij} = \mathbf{b}_j \ , \quad \mathbf{X}_{ij} \geq 0 \ , \tag{28}$$

where m is the number of buyers, n is the number of goods, $\mathbf{w} \in \mathbb{R}^m, \mathbf{U} \in \mathbb{R}^{m\times n}$. The vector $\mathbf{w} \in \mathbb{R}^m$ represents the monetary endowment of each buyer, $\mathbf{U} \in \mathbb{R}^{m\times n}$ denotes the utility of each buyer for each good and the j-th entry of $\mathbf{b}$ denotes the overall amount of goods j. In this formulation, each buyer i allocates their budget $\mathbf{w}_i$ to maximize their utility, while each seller j offers a good for sale. An equilibrium is reached when goods are priced so that every buyer acquires an optimal bundle and the market clears, meaning all budgets are spent and all goods are sold.

The problem (28) is equivalently written as the following:

$$\min_{\mathbf{p}\in\mathbb{R}^m \ , \ \mathbf{X}\in\mathbb{R}^{m\times n}, \ \mathbf{t}\in\mathbb{R}^m} -\sum_{i\in[m]} \mathbf{w}_i\mathbf{p}_i \ , \text{s.t.} \quad \begin{array}{l} \sum\limits_{i\in[m]} \mathbf{X}_{ij} = \mathbf{b}_j \ , \ \forall \ j \in [n] \ , \mathbf{t}_i = \sum\limits_{j\in[n]} \mathbf{U}_{ij}\mathbf{X}_{ij} \ , \ \forall \ i \in [m] \ , \\ \mathbf{p}_i \leq \log\left(\mathbf{t}_i\right), \forall \ i \in [m] \ , \mathbf{X}_{ij} \geq 0 \ , \ \forall i \in [m] \ , \ j \in [n] \ , \end{array} \tag{29}$$

where $\mathbf{w} \in \mathbb{R}^m$ is the vector of buyer budgets and $\mathbf{U} \in \mathbb{R}^{m\times n}$ is the utility matrix.

We now derive an equivalent conic formulation. Let the decision variables be stacked as $\mathbf{y} \in \mathbb{R}^{m(n+2)}$: $\mathbf{y} = \left[\mathbf{X}_{1,:}^\top, \ldots, \mathbf{X}_{m,:}^\top, (\mathbf{p}_1, \mathbf{t}_1), \ldots, (\mathbf{p}_m, \mathbf{t}_m)\right]^\top$. Let the problem parameters be stacked as follows in $\mathbf{c}, \mathbf{A}, \tilde{\mathbf{b}}, \mathbf{Q}, \mathbf{d}$: $\mathbf{c} = \left[\mathbf{0}_{mn\times 1}^\top, (-\mathbf{w}_1, 0), \ldots, (-\mathbf{w}_m, 0)\right]^\top$,

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_{n\times n} & \cdots & \mathbf{I}_{n\times n} & & & \\ \mathbf{U}_{1,:} & & & (0,-1) & & \\ & \cdots & & & \cdots & \\ & & \mathbf{U}_{m,:} & & & (0,-1) \end{bmatrix}, \tilde{\mathbf{b}} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0}_m \end{bmatrix},$$

$$\mathbf{Q} = \left[\mathbf{0}_{3m\times mn} \ \ \mathbf{I}_m \otimes \left[\mathbf{e}_{3,1} \ \mathbf{e}_{3,3}\right]\right], \mathbf{d} = \left[\mathbf{1}_{m\times 1} \otimes (-\mathbf{e}_{3,2})\right],$$

where $\mathbf{e}_{3,i} \in \mathbb{R}^3$ is the i-th standard basis vector of $\mathbb{R}^3$.

Then (29) is equivalent to the following conic program problem:

$$\min_{\mathbf{y}\in\mathbb{R}^{m(n+2)}} \ \mathbf{c}^\top\mathbf{y} \ , \ \text{s.t.} \ \mathbf{A}\mathbf{y} = \tilde{\mathbf{b}} \ , \ \mathbf{Q}\mathbf{y} - \mathbf{d} \in \mathcal{K}_{\text{exp}}^m, \ \mathbf{y} \geq \mathbf{l} \tag{30}$$

where $\mathcal{K}_{\text{exp}}^m := \underbrace{\mathcal{K}_{\text{exp}} \times \cdots \times \mathcal{K}_{\text{exp}}}_{m \text{ items}}$ and $\mathbf{l} := \left[\underbrace{0, \ldots, 0}_{mn \text{ items}}, \underbrace{-\infty, \ldots, -\infty}_{2m \text{ items}}\right]$.

In the experiments, we consider some different dimension choices of m and n for matrix $\mathbf{U}$ while maintaining a constant sparsity level of 0.2. Each nonzero entry of $\mathbf{w}$ and $\mathbf{U}$ is sampled independently from the uniform distribution $U[0,1]$, and we set $\mathbf{b}_j = 0.25$ for all $j \in [n]$.

B.2 Lasso Problem (Section 5.3)

In this subsection, we present the SOCP reformulation of the Lasso problem following the approach of [14]. The original Lasso problem is as follows $\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2 + \lambda\|\mathbf{x}\|_1$, where $\mathbf{A}\in\mathbb{R}^{m\times n}$. Let $\mathbf{x}=\mathbf{x}_1-\mathbf{x}_2, \mathbf{x}_1\in\mathbb{R}^n_+, \mathbf{x}_2\in\mathbb{R}^n_+$ and $\mathbf{y}=\mathbf{A}\mathbf{x}-\mathbf{b}$, then the above problem can be rewritten as

$$\min_{\mathbf{x}_1\in\mathbb{R}^n_+,\mathbf{x}_2\in\mathbb{R}^n_+,\mathbf{y}\in\mathbb{R}^m,r\in\mathbb{R}} 2r+\lambda\cdot\mathbf{1}_{n\times 1}^\top(\mathbf{x}_1+\mathbf{x}_2)\ \text{ s.t. }\ \mathbf{y}=\mathbf{A}(\mathbf{x}_1-\mathbf{x}_2)-\mathbf{b}\ ,\|\mathbf{y}\|^2\le 2r.$$

It should be noted that the constraint $\|\mathbf{y}\|^2\le 2r$ can be reformulated as a second-order cone constraint $\begin{bmatrix}\frac{1+r}{\sqrt{2}} & \frac{1-r}{\sqrt{2}} & \mathbf{y}^\top\end{bmatrix}^\top\in\mathcal{K}^{m+2}_{\text{soc}}$ because it is essentially $\left(\frac{1+r}{\sqrt{2}}\right)^2-\left(\frac{1-r}{\sqrt{2}}\right)^2-\|\mathbf{y}\|^2\ge 0$, which reduces to $2r-\|\mathbf{y}\|^2\ge 0$.

Since the rotated second-order cone is essentially a linearly transformed second-order cone, the SOCP reformulation of the Lasso problem is given by

$$\begin{aligned}\min_{\substack{w\in\mathbb{R},r\in\mathbb{R},\mathbf{y}\in\mathbb{R}^m\\ \mathbf{x}_1\in\mathbb{R}^n_+,\mathbf{x}_2\in\mathbb{R}^n_+}}\quad & \left\langle\begin{bmatrix}0 & 2 & (\mathbf{0}_{m\times 1})^\top & \lambda\cdot(\mathbf{1}_{2n\times 1})^\top\end{bmatrix}^\top,\begin{bmatrix}w & r & \mathbf{y}^\top & \mathbf{x}_1^\top & \mathbf{x}_2^\top\end{bmatrix}^\top\right\rangle\\ \text{s.t.}\quad & \begin{bmatrix}1 & 0 & 0 & \cdots & 0\\ \mathbf{0}_{m\times 1} & \mathbf{0}_{m\times 1} & \mathbf{I}_{m\times m} & -\mathbf{A} & \mathbf{A}\end{bmatrix}\begin{bmatrix}w & r & \mathbf{y}^\top & \mathbf{x}_1^\top & \mathbf{x}_2^\top\end{bmatrix}^\top=\begin{bmatrix}1\\ -\mathbf{b}\end{bmatrix},\\ & \begin{bmatrix}\frac{w+r}{\sqrt{2}} & \frac{w-r}{\sqrt{2}} & \mathbf{y}^\top\end{bmatrix}^\top\in\mathcal{K}^{m+2}_{\text{soc}}\end{aligned}.$$

We generate a family of synthetic Lasso problem instances following the experimental setting used for ABIP in [14]. Specifically, the matrix $\mathbf{A}\in\mathbb{R}^{m\times n}$ is generated with a fixed sparsity level of 10^{-4}, where each nonzero entry is drawn independently from the uniform distribution, i.e., $\mathbf{A}_{ij}\sim U[0,1]$ for all i and j. The label vector $\mathbf{b}$ is then generated by $\mathbf{b}=\mathbf{A}\tilde{\mathbf{x}}+10^{-6}\cdot\mathbf{1}$, where $\tilde{\mathbf{x}}\in\mathbb{R}^n$ has entries drawn independently from the standard normal distribution, and half of its components are randomly set to zero. The vector $\mathbf{1}$ denotes the all-ones vector. We set $\lambda=\left\|\mathbf{A}^\top\mathbf{b}\right\|_\infty$, as in [14].

B.3 Multi-period Portfolio Optimization (Section 5.4)

In this subsection, we present the multi-period portfolio optimization (MPO) problem and its conic program reformulation. [8] present a penalty-based formulation of the MPO problem as follows:

$$\begin{aligned}\max_{\substack{\mathbf{w}_{\tau+1},\mathbf{z}_\tau\in\mathbb{R}^{n+1}:\\ \tau=0,\ldots,T-1}}\ & \sum_{\tau=0}^{T-1}\left\{\hat{\mathbf{r}}_{\tau+1}^\top\mathbf{w}_{\tau+1}-\gamma_\tau^{\text{risk}}\hat{\psi}_\tau(\mathbf{w}_{\tau+1})-\gamma_\tau^{\text{hold}}\hat{\phi}_\tau^{\text{hold}}(\mathbf{w}_{\tau+1})-\gamma_\tau^{\text{trade}}\hat{\phi}_\tau^{\text{trade}}(\mathbf{z}_\tau)\right\}\\ \text{s.t.}\quad & \mathbf{1}^\top\mathbf{z}_\tau=0,\ \ \mathbf{z}_\tau\in\mathcal{Z}_\tau,\ \mathbf{w}_\tau+\mathbf{z}_\tau\in\mathcal{W}_\tau,\ \mathbf{w}_{\tau+1}=\mathbf{w}_\tau+\mathbf{z}_\tau,\ \text{for all }\tau=0,\ldots,T-1,\end{aligned}\tag{31}$$

where the decision variable $\mathbf{w}_{\tau+1}\in\mathbb{R}^{n+1}$ represents the weights of assets for the period $\tau+1$, in which the first n components denote n assets and the $(n+1)$-th component denotes the weight of cash. The other decision variable $\mathbf{z}_\tau$ denotes the transaction amount in the period τ.

In this formulation, the expected return $\hat{\mathbf{r}}_{\tau+1} \in \mathbb{R}^{n+1}$ is predicted beforehand using other statistical or machine learning methods. The function $\hat{\psi}_\tau(\cdot)$ quantifies risk, which is typically defined as $\hat{\psi}_\tau(\mathbf{w}_{\tau+1}) := \|\hat{\mathbf{\Sigma}}_\tau^{1/2}[\mathbf{w}_{\tau+1} - \mathbf{w}_b]_{[n]}\|^2 + \kappa(\sum_{i=1}^n \hat{\mathbf{\Sigma}}_{\tau,ii}^{1/2} |\mathbf{w}_{\tau+1} - \mathbf{w}_b|_{(i)})^2$. In this formulation, $\mathbf{w}_b$ denotes the benchmark asset weight and $\hat{\mathbf{\Sigma}}$ is an estimate of the covariance matrix of the stochastic returns, and the term $(\sum_{i=1}^n \hat{\mathbf{\Sigma}}_{\tau,ii}^{1/2} |\mathbf{w}_{\tau+1} - \mathbf{w}_b|_{(i)})^2$ is a measure of covariance forecast error risk and κ is a hyper-parameter [30, 35]. The term $\hat{\phi}_\tau^{\text{hold}}$ denotes the holding cost, which may include factors such as borrowing fees. The term $\hat{\phi}_\tau^{\text{trade}}(\mathbf{z}_\tau)$ denotes the transaction cost, which is typically modeled as a linear and quadratic function of the transaction amount $\mathbf{z}_\tau$, i.e., $\hat{\phi}_\tau^{\text{trade}}([\mathbf{z}_\tau]_i) = a_{\tau,i}|[\mathbf{z}_\tau]_i| + b_{\tau,i}|[\mathbf{z}_\tau]_{(i)}|^2$ where a_τ and b_τ are coefficients [25] that can be predicted using other statistical and machine learning methods. The set $\mathcal{Z}_\tau$ contains the constraints for the transaction amounts $\mathbf{z}_\tau$, such as limiting the transaction ratio in each period. The set $\mathcal{W}_\tau$ contains the constraints on the asset weights $\mathbf{w}_{\tau+1}$, such as the requirement for market neutrality, $(\mathbf{w}_\tau^m)_{[n]}^\top \hat{\mathbf{\Sigma}}_\tau (\mathbf{w}_{\tau+1})_{[n]} = 0$ where $\mathbf{w}_\tau^m$ is the asset value in period τ.

Hence, a specific example of (31) is

$$\begin{aligned}
\max_{\substack{\mathbf{w}_{\tau+1}\in\mathbb{R}^{n+1}:\\ \tau=0,\dots,T-1}} \quad & \sum_{\tau=0}^{T-1} \Bigg[\hat{\mathbf{r}}_{\tau+1}^\top \mathbf{w}_{\tau+1} - \gamma_\tau^{\text{risk}} \underbrace{\left(\left\| \hat{\mathbf{\Sigma}}_\tau^{1/2} [\mathbf{w}_{\tau+1} - \mathbf{w}_b]_{[n]} \right\|^2 + \kappa \left(\sum_{i=1}^n \hat{\mathbf{\Sigma}}_{\tau,ii}^{1/2} |[\mathbf{w}_{\tau+1} - \mathbf{w}_b]_i| \right)^2 \right)}_{\hat{\psi}_\tau^{\text{risk}}(\mathbf{w}_{\tau+1})} \\
& - \gamma_\tau^{\text{trade}} \cdot \underbrace{\left(\sum_{i=1}^n a_{\tau,i} |\mathbf{w}_{\tau+1} - \mathbf{w}_\tau|_i + b_{\tau,i} (\mathbf{w}_{\tau+1} - \mathbf{w}_\tau)_i^2 \right)}_{\hat{\phi}_\tau^{\text{trade}}(\mathbf{w}_{\tau+1})} - \gamma_\tau^{\text{hold}} \cdot \underbrace{0}_{\hat{\phi}_\tau^{\text{hold}}(\mathbf{w}_{\tau+1})} \Bigg] \\
\text{s.t.} \quad & \mathbf{1}^\top (\mathbf{w}_{\tau+1} - \mathbf{w}_\tau) = 0 \ , \quad \mathbf{w}_{\tau+1} \geq 0 \ , \quad (\mathbf{w}_\tau^m)_{[n]}^\top \hat{\mathbf{\Sigma}}_\tau (\mathbf{w}_{\tau+1})_{[n]} = 0 \ , \tau = 0, \dots, T-1 \ .
\end{aligned} \tag{32}$$

This specific example considers the case with no borrowing fee, long holding, and no constraint on transactions. In this formulation, $\gamma_\tau^{\text{risk}}, \kappa, \gamma_\tau^{\text{trade}}, a_{\tau,i}, b_{\tau,i}\, \gamma_\tau^{\text{hold}}$ are hyper-parameters.

The above problem models the risk-control tasks in the portfolio optimization problem as regularization terms added on the objective function, but how to choose the proper hyper-parameters to balance different objectives would be difficult in practice and require heavy parameter tuning. Given that, another commonly seen strategy is to model the other objectives other than profit within the constraints. In this way, the hyper-parameters directly correspond to the maximum allowed holding costs, transaction cost, or risk. See, for example, [34]. Such a constraint-based formulation of (32)

is as follows, a SOCP problem:

$$\max_{\substack{\mathbf{u}_\tau\in\mathbb{R}^n,\mathbf{w}_{\tau+1}\in\mathbb{R}^{n+1}:\\ \tau=0,\ldots,T-1}} \sum_{\tau=0}^{T-1} \hat{\mathbf{r}}_{\tau+1}^\top \mathbf{w}_{\tau+1}$$
$$\text{s.t.} \begin{cases} \mathbf{1}^\top(\mathbf{w}_{\tau+1}-\mathbf{w}_\tau)=0, \mathbf{w}_{\tau+1}\geq 0, (\mathbf{w}_\tau^m)_{[n]}^\top \hat{\mathbf{\Sigma}}_\tau (\mathbf{w}_{\tau+1})_{[n]}=0, \\ \mathbf{u}_{\tau,i}\geq (\mathbf{w}_{\tau+1}-\mathbf{w}_b)_i, \mathbf{u}_{\tau,i}\geq -(\mathbf{w}_{\tau+1}-\mathbf{w}_b)_i, \sum_{i=1}^n \left(\hat{\mathbf{\Sigma}}_\tau^{1/2}\right)_{ii} \mathbf{u}_{\tau,i}\leq \gamma_{3\tau} \\ \left\|\hat{\mathbf{\Sigma}}_\tau^{1/2}(\mathbf{w}_{\tau+1}-\mathbf{w}_b)_{[n]}\right\|\leq \gamma_{1\tau}\,, \\ -\gamma_{2\tau,i}\leq (\mathbf{w}_{\tau+1}-\mathbf{w}_\tau)_i\leq \gamma_{2\tau,i}, \quad i=1,\ldots,n+1 \end{cases} \forall \tau=0,\ldots,T-1. \tag{33}$$

The above problem (33) is the problem that the various solvers directly address in the experiments in Section 5.4.

C Optimality Termination Criteria

This section describes the termination criteria of the solvers in our experiments. Generally, these criteria comprise three types of errors: primal infeasibility (err_p), dual infeasibility (err_d), and the duality gap (err_gap).

ABIP addresses the following primal and dual problems:

$$\min_{\mathbf{x}} \quad \mathbf{c}^\top\mathbf{x} \quad \text{s.t.}\, \mathbf{A}\mathbf{x}=\mathbf{b}\,, \quad \mathbf{x}\in\mathcal{K} \quad , \quad \max_{\mathbf{y}} \quad \mathbf{b}^\top\mathbf{y} \quad \text{s.t.}\, \mathbf{A}^\top\mathbf{y}+\mathbf{s}=\mathbf{c}\,, \quad \mathbf{s}\in\mathcal{K}^*\,. \tag{34}$$

If the given optimization problem does not satisfy this formulation, ABIP reformulates the problem to (34) first and then solves it. It should be noted that for all iterates of ABIP, $\mathbf{x}$ and $\mathbf{s}$ always lie in the cones $\mathcal{K}$ and $\mathcal{K}^*$ because ABIP uses an ADMM-based interior-point method, which ensures all iterates are always in the interior of the cones. On the formulation (34), ABIP determines its termination criteria based on the following three types of errors:

$$\text{err}_\text{p} := \frac{\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_\infty}{1+\max\{\|\mathbf{A}\mathbf{x}\|_\infty, \|\mathbf{b}\|_\infty\}}, \ \text{err}_\text{d} := \frac{\left\|\mathbf{c}-\mathbf{A}^\top\mathbf{y}-\mathbf{s}\right\|_\infty}{1+\|\mathbf{c}\|_\infty}, \ \text{err}_\text{gap} := \frac{|\mathbf{c}^\top\mathbf{x}-\mathbf{b}^\top\mathbf{y}|}{1+\max\{|\mathbf{c}^\top\mathbf{x}|, |\mathbf{b}^\top\mathbf{y}|\}}.$$

In our experiments, we say a solution satisfies the ε_rel tolerance if $\max\{\text{err}_\text{p}, \text{err}_\text{d}, \text{err}_\text{gap}\}\leq\varepsilon_\text{rel}$ for this solution.

SCS solves the conic program problems in the following formulation:

$$\min_{\mathbf{x}} \quad \mathbf{c}^\top\mathbf{x} \quad \text{s.t.}\, \mathbf{A}\mathbf{x}+\mathbf{s}=\mathbf{b}, \quad \mathbf{s}\in\mathcal{K}, \quad \max_{\mathbf{y}} \quad -\mathbf{b}^\top\mathbf{y} \quad \text{s.t.}\, \mathbf{A}^\top\mathbf{y}+\mathbf{c}=0, \quad \mathbf{y}\in\mathcal{K}^*. \tag{35}$$

If the problem is not in this formulation, we first reformulate it into this formulation and then use SCS to solve it. All iterates of SCS satisfy the conic constraints $s\in\mathcal{K}$ and $y\in\mathcal{K}^*$ as the solver does projections onto the cones in each iteration. On the formulation (35), SCS determines its

termination criteria based on the following residual thresholds:

$$\begin{aligned}
\mathrm{err_p} &= \|\mathbf{A}\mathbf{x}+\mathbf{s}-\mathbf{b}\|_\infty \le \varepsilon_{\mathrm{abs}} + \varepsilon_{\mathrm{rel}} \cdot \max\left\{\|\mathbf{A}\mathbf{x}\|_\infty, \|\mathbf{s}\|_\infty, \|\mathbf{b}\|_\infty\right\}, \\
\mathrm{err_d} &= \left\|\mathbf{A}^\top\mathbf{y}+\mathbf{c}\right\|_\infty \le \varepsilon_{\mathrm{abs}} + \varepsilon_{\mathrm{rel}} \cdot \max\left\{\left\|\mathbf{A}^\top\mathbf{y}\right\|_\infty, \|\mathbf{c}\|_\infty\right\}, \\
\mathrm{err_{gap}} &= |\mathbf{c}^\top\mathbf{x}-\mathbf{b}^\top\mathbf{y}| \le \varepsilon_{\mathrm{abs}} + \varepsilon_{\mathrm{rel}} \cdot \max\left\{|\mathbf{c}^\top\mathbf{x}|, |\mathbf{b}^\top\mathbf{y}|\right\}.
\end{aligned}$$

In our experiments, we say a solution satisfies the ε tolerance if the above conditions all hold with $\varepsilon_{\mathrm{rel}} = \varepsilon_{\mathrm{abs}} = \varepsilon$.

MOSEK uses an interior-point method to solve the conic program problem that is similarly structured to (34). It addresses the self-dual homogeneous model below:

$$\mathbf{A}\mathbf{x}-\mathbf{b}\tau = 0,\ \mathbf{A}^\top\mathbf{y}+\mathbf{s}-\mathbf{c}\tau = 0,\ \ -\mathbf{c}^\top\mathbf{x}+\mathbf{b}^\top\mathbf{y}-\kappa = 0,\ \mathbf{x}\in\mathcal{K},\ \ \mathbf{s}\in\mathcal{K}^*,\ \ \tau,\kappa\ge 0\ .$$

A solution is regarded as satisfying the $\varepsilon_{\mathrm{rel}}$ tolerance in MOSEK if the following conditions hold:

$$\begin{aligned}
\mathrm{err_p} = \frac{\left\|\mathbf{A}\frac{\mathbf{x}}{\tau}-\mathbf{b}\right\|_\infty}{1+\|\mathbf{b}\|_\infty} \le \varepsilon_{\mathrm{rel}}\ , \quad \mathrm{err_d} = \frac{\left\|\mathbf{A}^\top\frac{\mathbf{y}}{\tau}+\frac{\mathbf{s}}{\tau}-\mathbf{c}\right\|_\infty}{1+\|\mathbf{c}\|_\infty} \le \varepsilon_{\mathrm{rel}}, \\
\mathrm{err_{gap}} = \max\left\{\frac{\mathbf{x}^\top\mathbf{s}}{\tau^2}, \left|\frac{\mathbf{c}^\top\mathbf{x}}{\tau}-\frac{\mathbf{b}^\top\mathbf{y}}{\tau}\right|\right\} \ \Big/\ \max\left\{1, \frac{\min\left\{|\mathbf{c}^\top\mathbf{x}|, |\mathbf{b}^\top\mathbf{y}|\right\}}{\tau}\right\} \le \varepsilon_{\mathrm{rel}}\ .
\end{aligned}$$

The problem that CuClarabel directly addresses is similar to that of SCS, i.e., (35). However, CuClarabel introduces two slack variables $\tau, \kappa \ge 0$ and considers the following optimization problem:

$$\min\ \ \mathbf{s}^\top\mathbf{y}+\tau\kappa\ \ \text{s.t.}\ \ \mathbf{c}^\top\mathbf{x}+\mathbf{b}^\top\mathbf{y} = -\kappa\ ,\ \mathbf{A}^\top\mathbf{y}+\mathbf{c}\tau = 0, \mathbf{A}\mathbf{x}+\mathbf{s}-\mathbf{b}\tau = 0,\ \ (\mathbf{s},\mathbf{y},\tau,\kappa)\in\mathcal{K}\times\mathcal{K}^*\times\mathbb{R}_+\times\mathbb{R}_+.$$

For the above formulation, a solution is regarded as satisfying the $\varepsilon_{\mathrm{rel}}$ tolerance if the following conditions hold:

$$\begin{aligned}
\mathrm{err_p} &= \frac{\left\|\mathbf{A}\frac{\mathbf{x}}{\tau}+\frac{\mathbf{s}}{\tau}-\mathbf{b}\right\|_\infty}{\max\{1, \|\mathbf{b}\|_\infty + \|\mathbf{x}/\tau\|_\infty + \|\mathbf{s}/\tau\|_\infty\}} \le \varepsilon_{\mathrm{rel}}\ , \\
\mathrm{err_d} &= \frac{\left\|\mathbf{A}^\top\frac{\mathbf{y}}{\tau}+\mathbf{c}\right\|_\infty}{\max\{1, \|\mathbf{c}\|_\infty + \|\mathbf{x}/\tau\|_\infty + \|\mathbf{y}/\tau\|_\infty\}} \le \varepsilon_{\mathrm{rel}} \\
\mathrm{err_{gap}} &= \frac{\left|\mathbf{c}^\top\mathbf{x}+\mathbf{b}^\top\mathbf{y}\right|}{\max\{1, \min\{|\mathbf{c}^\top\mathbf{x}|, |\mathbf{b}^\top\mathbf{y}|\}\}} \le \varepsilon_{\mathrm{rel}}\ .
\end{aligned}$$

Since CuClarabel also uses an interior-point method, all iterates of CuClarabel are within the cones.

COPT uses an interior-point method to solve conic programs, but the definition of tolerance in its optimality criteria is not publicly available.

D Comparison of PDHG and PDCS

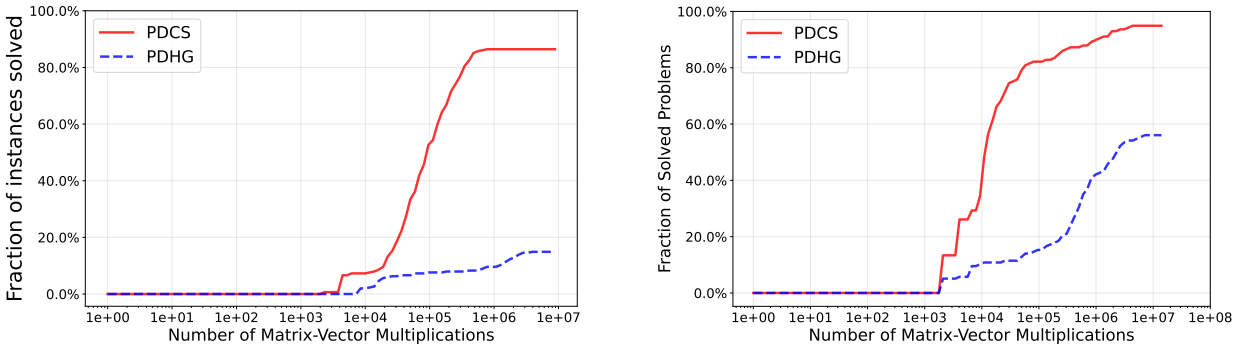
In this section, we provide additional details about the comparison between the standard PDHG and PDCS presented in Figure 1 in Section 3.

We evaluate PDHG and PDCS, two CPU-based implementations, on the CBLIB dataset, which

Table 7: SGM(10) Comparison between PDHG and PDCS

	Small SOCP (1641)		Medium SOCP (220)		Large SOCP (82)		Exponential Cone Problems (157)	
	PDCS	PDHG	PDCS	PDHG	PDCS	PDHG	PDCS	PDHG
SGM(10)	2.78	16.65	160.64	1862.50	1602.14	3600.00	18.52	675.84
count	1640	1552	208	45	53	0	149	88

is divided into small-, medium-, and large-scale subsets for the second-order cone and exponential cone problems. The classification criteria have been described in Section 5. For the baseline PDHG used in our comparisons, we select a simple choice where the primal step size and the dual step size are both set to $0.9/\|G\|_2$, which is similar to the default parameters in the standard PDHG implementation in [1, 2]. The baseline PDHG does not employ any additional enhancement techniques described in Section 3. Figure 1 in Section 3 shows the experimental results on the medium- and large-scale subset of the second-order cone and exponential cone problems. The time limit is set to 3600 seconds for both methods. The complete SGM(10) results are shown in Table 7. On



(a) Medium and Large Second-order Cone Dataset

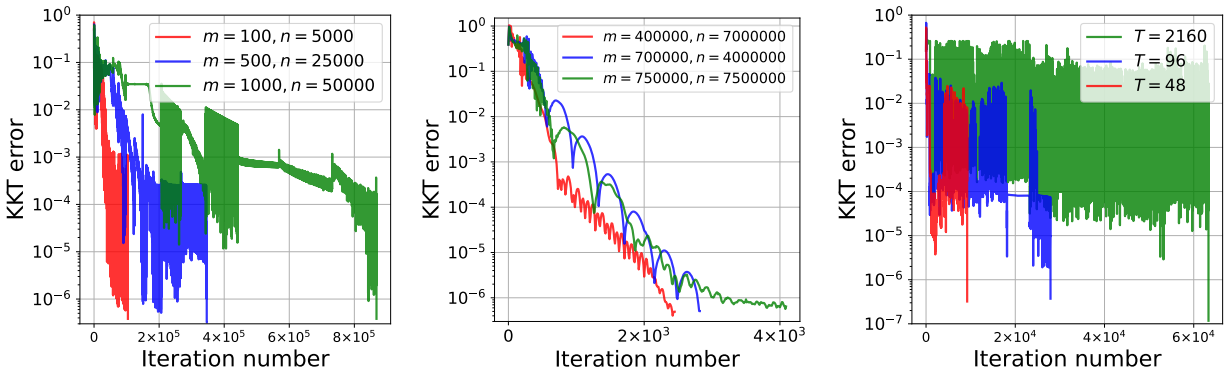
(b) Exponential & Second-order Cone Dataset

Figure 7: Performance of PDCS and PDHG in terms of the number of matrix-vector multiplications.

small- and medium-scale SOCP datasets, PDCS achieves a performance advantage of roughly 6-12 times over PDHG. PDCS also shows good scalability, as evidenced by its ability to solve large-scale SOCP problems that remain unsolved by PDHG. For the problems with both exponential and second-order cones, PDCS demonstrates more improvements, delivering speedups of up to 36 times compared with PDHG. These results demonstrate that the effectiveness of PDCS is largely attributable to the proposed enhancement. Figure 7 presents the results, where the y-axis indicates the fraction of solved instances and the x-axis denotes the number of matrix-vector multiplications. Figure 7a corresponds to the dataset in which all constraints are second-order cone, while Figure 7b corresponds to the dataset with exponential and second-order cone constraints. In both datasets, PDCS significantly outperforms PDHG, solving substantially more instances within the same iteration limit, which demonstrates that the enhancement techniques in Section 3 considerably improve the performance of PDCS.

E Additional remarks on the experiments in Section 5

To provide further insight into the algorithmic behavior of PDCS, we illustrate its convergence on several representative problem instances in Figure 8. Each plot shows the evolution of the "KKT error" across iterations of PDCS. Here, the "KKT error" is defined as the average of the three error metrics introduced in (8).



(a) Fisher market equilibrium problem (Section 5.2) (b) LASSO problem (Section 5.3) (c) MPO problem (Section 5.4)

Figure 8: Convergence behavior of PDCS.

Figures 8a and 8c illustrate the convergence behavior for three Fisher market equilibrium problems and three MPO problems, respectively. In both cases, we observe an initial phase of rapid progress followed by a plateau with slower KKT error reduction. This trend helps explain why PDCS is particularly effective in computing low-accuracy solutions. Interestingly, during the final iterations, PDCS often exhibits a phase of accelerated convergence, which differs from the typical sublinear behavior expected from first-order methods. Figure 8b shows the convergence of PDCS on three Lasso problem instances. In this case, the algorithm exhibits nearly linear convergence, enabling it to achieve high-accuracy solutions in a relatively short time. This observation is consistent with the results reported in Section 5.3 and suggests that the Lasso problem is particularly well-suited for PDCS. It would be interesting to explore what properties of conic problems (such as Lasso problems) could make them particularly suitable for PDCS.