Fast and Parallelizable Numerical Algorithms for Large Scale Conic Optimization Problems

by

MUHAMMAD ADIL

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

Dec 2021

To my parents for their unconditional love and support

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my PhD advisor, Ramtin Madani, for his endless support and valuable guidance throughout the journey. Ramtin spent countless hours discussing research ideas, helping in formulating problems and explaining the concepts in an understandable way. I am greatly indebted to Ramtin for inspiring me as a researcher and setting my future career directions. It was always incredible to watch him working, his hard work and dedication towards the job was truly inspiring. I am immensely thankful to him for being friendly and supportive.

I want to express my gratitude to Stephen Grikschat who served as a mentor during the summer internship at MathWorks. I truly enjoyed working with optimization toolbox team at MathWorks and it was not possible without the support of Steve. I found him a very generous and an open minded team lead. I am also thankful to Changrak Choi for believing in me and providing the opportunity to work for exciting projects at JPL NASA.

I had pleasure of working with Edward Quarm Jnr and Adnan Nasir as colleagues at UTA. I had learned so much from Mohsen Kheirandeshfard and Fariba Zohrizadeh and I am grateful for their valuable discussions.

This journey wouldn't have been possible without the love and support of my parents and siblings. My Mom never went to school but I found her a literate and educated woman with a kind heart. She always had a dream of sending us to best colleges in the country. I cannot be grateful enough to my brothers who sacrificed their lives for my success.

December 14, 2021

ABSTRACT

Fast and Parallelizable Numerical Algorithms for Large Scale Conic Optimization Problems

Muhammad Adil, Ph.D.

The University of Texas at Arlington, 2021

Supervising Professor: Ramtin Madani

Many real world problems from various application areas such as engineering, finance and operation research can be cast as optimization problems. Generally, the goal is to optimize an objective function under a set of constraints. Traditionally, convex optimization problems are solved by an interior point method (IPM). Interior point methods proved to achieve high accuracy for moderate size problems. However, the computation cost of iterations of these iterative algorithms grows non-linearly with the dimension of the problem. Although interior-point methods are robust and theoretically sound, they do not scale well for very large conic optimization programs. Computational cost, memory issues, and incompatibility with distributed platforms are among the major impediments for interior point methods in solving large-scale and practical conic optimization programs.

The rapid growth of problem size in application areas such as power systems, finance, signal processing and machine learning motivated researchers to develop computationally efficient optimization solvers. In recent years, first orders methods have received a particular attention for solving large convex optimization problems. Various optimization solvers based on first order numerical algorithms have been devel-

oped in the past decade. Although first order methods provide low accuracy solutions, but inexpensive iterations and low computational cost makes them attractive mathematical tools for handling large-scale problems.

One of the major shortcomings of first order methods to achieve a higher accuracy is their slow tail convergence behavior. The first part of this work is an attempt to remedy the problem of slow convergence for first-order numerical algorithms by proposing an adaptive conditioning heuristic policy. First, a parallelizable numerical algorithm is proposed that is capable of dealing with large-scale conic programs on distributed platforms such as graphics processing unit (GPU) with orders-of-magnitude time improvement. The mathematical proof for global convergence of proposed numerical algorithm is provided. In the past decade, several preconditioning methods have been applied to improve the condition number and convergence of first order methods. Diagonal preconditioning and matrix equilibration techniques are most commonly used for this purpose. In contrary to the existing techniques, in this work, it is argued that the condition number of the problem data is not a reliable predictor of convergence speed. In light of this observation, an adaptive conditioning heuristic is proposed which enables higher accuracy compared to other first-order numerical algorithms. A wide range of experiments are conducted on a variety of large-scale linear programming and second-order cone programming problems to demonstrate the scalability and computational advantages of the proposed algorithm compared to commercial and open-source solvers.

Solving the linear system is probably the most computationally expensive part in first order methods. The existing methods rely on direct and indirect methods for solving the linear systems of equations. Direct methods rely on factorization techniques which usually destroy the sparsity structure of original sparse problems and hence become computationally prohibitive. Alternatively, indirect methods are

iterative and various preconditioning variants of indirect or iterative methods have been studied in the literature to improve accuracy, but again the preconditioners do not necessarily retain the sparsity patterns of original problems. In the second part of this work, a matrix-free first order approach is proposed for solving large-scale sparse conic optimization problems. This method is based on an easy-to-compute decomposition of large sparse matrices into two factors. The proposed numerical algorithm is based on matrix-free decomposition and alternating direction method of multipliers. The iterations of the designed algorithm are computationally cheap, highly parallelizable and enjoy closed form solutions. The algorithm can easily be implemented on distributed platforms such as graphics processing units with orders-of-magnitude time improvements. The performance of the proposed algorithm is demonstrated on a variety of conic problems and the performance gain is compared with competing first-order solvers.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Page

CHAPTER 1

Introduction

Conic optimization is a subfield of convex optimization that encapsulate linear programming (LP), quadratic programming (QP), second-order cone programming (SOCP) and semidefinite programming (SDP) as special cases. Conic optimization can be thought as generalization of linear programming and minimizes a linear objective under the conic constraint $\mathcal{K}$, where $\mathcal{K}$ is a convex cone. The conic optimization problems can be written as follows:

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}} \qquad \boldsymbol{c}^\top \boldsymbol{x} \tag{1.1a}$$

$$\text{subject to} \qquad \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \in \mathcal{K} \tag{1.1b}$$

Conic programming is of practical interest in a wide variety of areas such as operation research, machine learning, signal processing, optimal control and portfolio management. The real-time operation of many real-world applications in these areas depends on solving various large-scale conic optimization problems very frequently. Recent advancements in terms of conic relaxations for optimal power flow problems opened new horizon for efficiently solving large scale conic optimization programs. A great deal of efforts have been made in the past few years by introducing conic relaxations for many non-convex problems appears in the area of machine learning. Conic optimization finds its applications in finance by efficiently solving portfolio

management and risk analysis problem. Solving large scale LMI problems for various applications in control systems can also be casted as conic optimization problems.

While conic optimization problems arise in a diverse set of fields efficiently solving the large scale optimization problems still remains an active area of research. In this dissertation, we propose efficient numerical algorithms for solving such large scale conic optimization problems.

1.1 First-Order Operator Splitting Algorithm

We introduce a first order method based on operator splitting technique for solving very large convex conic problems. The proposed algorithm uses only first-order information and iteratively compute the optimal solution of conic programs. The first order method distribute the objective function and then proximal operators are used iteratively to solve the cone programs. Solving the linear system of equation is of the important step of first order methods. In this algorithm, instead of solving the linear system in each iteration, we solve the linear system of equation in first iteration, store the factors and reuse these factors in subsequent iteration to increase the computational power of method. The iterative steps of the algorithm are computationally inexpensive and utilize less memory. This algorithm is parallelizable and can easily be implemented on graphics processing unit (GPU). This algorithm is highly scalable, parallelizable for GPU implementation and requires less computing memory. The simple arithmetic steps of the proposed algorithm enables the application of graphical processing units with an order-of-magnitude time improvement.

1.2 Rapid Convergence of First Order Methods via Adaptive Conditioning

First order methods have limitations in achieving higher accuracy within a reasonable number of iterations because of their sensitivity to the condition number of problem data matrices [11, 18, 24, 25]. Although first order operator splitting methods have been studied extensively in recent years for solving large scale conic programs for different applications but until the recent past, very few efforts are made to study the convergence rate [26, 27]. In past decade, the convergence analysis of first order algorithms remained of central importance. Despite the fact that devoted efforts have been made recently to accelerate the convergence behavior of operating splitting methods and make them more robust for practical applications, these techniques rely on the input problem data matrices, pre-conditioning, solution polishing, and step size parameter selection [15, 17, 25]. In conjunction with massively parallelizable and cheap iterative algorithm we propose a heuristic policy to scale the data matrices in such a way that the combined algorithm ensures the global convergence and achieves a high accuracy within a tens of iterations. In short, the proposed algorithm enjoys the benefits of first order algorithms such as low per iteration cost, scalability for very large problems, parallel and distributed implementations, and at the same time achieves the higher accuracy level of interior point methods.

1.3 A First-Order Numerical Algorithm without Matrix Operations

We pursue a matrix-free first order numerical algorithm to deal with very large-scale sparse conic optimization problems. The basic idea is to decompose the constraint matrix into sparse factors in such a way that iterative steps are division free and do not involve any matrix operations. The sparse factors are computed only in first iteration, and reused in subsequent iterations. In comparison to existing matrix

decomposition techniques such as LDL and QR decomposition, our approach is simple and computationally efficient to compute the factors. The notable property of the proposed decomposition is that computing the factors and steps of iterative loop are division free. The factors are easy-to-compute and require minimally possible memory storage. The matrix inversion lemma is utilized to make the operations division free. We reformulate the standard conic optimization problem by introducing auxiliary variables and then apply the proposed matrix-free algorithm in conjunction with well-known two-block ADMM [14, 19, 20]. The computational burden of solving the linear system in each iteration is being taken out of the iterative loop and factors are stored first, without any expensive factorization. The algorithm offers the benefits of matrix-free, division-free and inexpensive iterations. The proposed algorithm admitting parallel and large-scale implementation, and amenable to graphics processing unit (GPU) implementation. We demonstrate the performance gain and computational speedup of algorithm by conducting a wide range of experiments and compare the results with other first-order solvers.

CHAPTER 2

First-Order Operator Splitting Algorithm

2.1 Introduction

Conic optimization is of practical interest in a variety of application areas such as operation research, machine learning, signal processing and optimal control. Conic formulation can be visualized as a universal formulation of convex programs as any convex optimization problem can be cast in standard conic optimization formulation. In fact several convex optimization solvers and tools such as MOSEK [1], SDPT3 [2], CVX [3], GUROBI [4] and CPLEX [5] first transform the problems into a conic optimization problem. For this purpose, interior point-based algorithms perform very well and have become the standard method of solving conic optimization problems [6–8]. Various commercial and open-source solvers such as MOSEK [1], GUROBI [4], and SeDuMi [9] are based on interior point methods as their default algorithm. Although interior-point methods are robust and theoretically sound, they do not scale well for very large conic optimization programs. Computational cost, memory issues, and incompatibility with distributed platforms are among the major impediment for interior point methods in solving large-scale and practical conic optimization problems.

2.2 Related Work

The conic optimization problem can be cast as composite convex optimization problem and there exists a variety of first order operator splitting methods to solve such large scale problems. In recent years, operator splitting approaches such as Alternating Direction Method of Multipliers (ADMM) [14–20] and Douglas-Rachford

Splitting (DRS) [10–13] have received particular attention because of their potential for parallelization and ability to scale. These methods are interrelated in a sense that ADMM applied to primal is equivalent to DRS applied to the dual. First order methods are popular because the iterative steps are computationally cheap and easy to implement and thus ideal for large scale problems where high accuracy solutions are typically not required. Operator splitting techniques, on the other hand can lead to parallel and distributed implementation and provide moderate accuracy solutions to conic programs in a relatively lower computational time.

Motivated by the cheap per iteration cost and ability to handle large scale problems, several first order operator splitting algorithms have been proposed recently. Authors in [16], introduce a solver (SCS), a homogeneous self-dual embedding method based on ADMM to solve large convex cone programs and provide primal or dual infeasibility certificates when relevant. A MATLAB solver CDCS [17] extended the homogeneous self-dual embedding concept [16] and exploits the sparsity structure using chordal decomposition for solving large scale semidefinite programming problems. The ADMM algorithm introduced in [14] is improved by selecting the proximal parameter and pre-conditioning to introduce an open-source software package called POGS (Proximal Graph Solver) [15] and multiple practical problems are tested to evaluate the performance. Another application of operator splitting methods is provided in an open-source solver OSQP (operator splitting solver for quadratic programs) [18], where operator splitting technique is applied to solve quadratic programs. Open-source Julia implemented conic operator splitting method (COSMO) [21], solves the quadratic objective function under conic constraints. In [12], a Python package Anderson accelerated Douglas-Rachford splitting (A2DR) is introduced to solve large-scale non-smooth convex optimization problems. Although these solvers scale very well as the dimension of the problem increases in different practical areas but

suffers from slow convergence and do not perform well when the given problem is ill-conditioned [13, 22, 23].

2.2.1 Contributions

We propose a highly scalable, simple iterative, and parallelizable first order algorithm for solving large conic optimization programs. We propose a new operating splitting method where each iteration requires simple arithmetic operations, leads to parallel and distributive implementation, scales gracefully for very large cone programs and provides a very accurate solutions which is beyond the reach of other first order solvers. In short, we aim to propose algorithm which enjoys the benefits of first order algorithms such as low per iteration cost, scalability for very large problems, parallel and distributed implementations, and at the same time achieves the higher accuracy level of interior point methods.

The rest of the chapter is organized as follows. Some preliminaries of cone programming and definitions are presented in section 2.3. A brief introduction of commonly used first order algorithms is presented in 2.4. Section 2.5, discusses the proposed first order algorithm. Section 2.6 concludes the chapter.

2.2.2 Notations

Notations $\mathbb{R}$ and $\mathbb{N}$ denote the set of real and natural numbers, respectively. Matrices and vectors are represented by bold uppercase, and bold lowercase letters, respectively. Notation $\|\cdot\|_2$ refers to ℓ_2 norm of either matrix or vector depending on the context and $|\cdot|$ represents the absolute value. The symbol $(\cdot)^\top$ represent the transpose operators. The notations $\boldsymbol{I}_n$ refer to the $n \times n$ identity matrix. The symbol $\mathcal{K}$ is used to describe different types of cones used in this paper. The superscript

$(\cdot)^{\text{opt}}$ refers to the optimal solution of optimization problem. The notation $(\cdot)^{\dagger}$ denotes the Moore–Penrose pseudoinverse of transpose of a matrix. The symbol $\mathcal{L}$ represent the set of values to apply adaptive conditioning. The notation $\text{diag}\{\cdots\}$ represent the diagonal elements of a diagonal matrix. The symbols $\text{SK}, \text{AC}, \text{CS}$, are used to refer Sinkhorn-Knopp, adaptive conditioning and competing solver, respectively. The symbols ε^{abs} and ε^{rel} are used for absolute and relative tolerance, respectively.

2.3 Problem Formulation

In this chapter, we consider conic optimization problems with a linear objective, subject to a set of affine and second-order conic constraints. The primal formulation under study can be cast as:

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}} \quad & \boldsymbol{c}^\top \boldsymbol{x} && \text{(2.1a)} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && \text{(2.1b)} \\ & \boldsymbol{x} \in \mathcal{K} && \text{(2.1c)} \end{aligned}$$

where $\boldsymbol{c} \in \mathbb{R}^n, \boldsymbol{A} \in \mathbb{R}^{m\times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$ are given and $\boldsymbol{x} \in \mathbb{R}^n$ is the unknown optimization variable. Additionally, $\mathcal{K} \triangleq \mathcal{K}_{n_1} \times \mathcal{K}_{n_2} \times \cdots \times \mathcal{K}_{n_k} \subseteq \mathbb{R}^n$, where each $\mathcal{K}_{n_i} \subseteq \mathbb{R}^{n_i}$ is a Lorentz cone of size n_i, i.e.,

$$\mathcal{K}_{n_i} \triangleq \left\{\boldsymbol{w} \in \mathbb{R}^{n_i} \,|\, w_1 \geq \left\|[w_2, \ldots, w_{n_i}]\right\|_2\right\},$$

and $n_1 + n_2 + \ldots + n_k = n$.

The corresponding dual formulation of (2.1) is

$$\begin{aligned} \underset{\boldsymbol{y}\in\mathbb{R}^m,\boldsymbol{z}\in\mathbb{R}^n}{\text{maximize}} \quad & \boldsymbol{b}^\top \boldsymbol{y} && \text{(2.2a)} \\ \text{subject to} \quad & \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{z} = \boldsymbol{c} && \text{(2.2b)} \\ & \boldsymbol{z} \in \mathcal{K} && \text{(2.2c)} \end{aligned}$$

where $\boldsymbol{y}$ and $\boldsymbol{z}$ are dual variables associated with the constraints (2.1b) and (2.1c), respectively.

2.4 First Order Methods

In the past decade, several first order methods based solvers have been developed to solve large scale conic programs. Alternating Direction Method of Multipliers (ADMM) and Douglas-Rachford (DR) Splitting are the most common methods to target such large conic optimization problems. The key idea behind first order methods is to distribute the objective function by reformulating the original conic problem and then solve the distributed formulation. To this end, we breifly introduce DR splitting and ADMM methods.

2.4.1 Douglas-Rachford Splitting

In order to implement the DR splitting method, it is common practice to cast problem (2.1) as follows

$$\text{minimize} \qquad f(\boldsymbol{x}) + g(\boldsymbol{x}) \qquad \text{(2.3)}$$

where $f, g : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ are defined as

$$f(\boldsymbol{x}) \triangleq \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathcal{K} \\ \infty & \text{otherwise} \end{cases} \quad \text{and} \quad g(\boldsymbol{x}) \triangleq \begin{cases} \boldsymbol{c}^\top \boldsymbol{x} & \text{if } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ \infty & \text{otherwise} \end{cases}$$

leading to the following steps:

$$\boldsymbol{x} \leftarrow \text{prox}_f(\boldsymbol{z}) \tag{2.4a}$$

$$\boldsymbol{z} \leftarrow \boldsymbol{z} + \text{prox}_g(2\boldsymbol{x} - \boldsymbol{z}) - \boldsymbol{x}. \tag{2.4b}$$

2.4.2 Alternating Direction Method of Multipliers

A standard way of solving problem (2.1) via ADMM is through the formulation

$$\underset{\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n}{\text{minimize}} \qquad f(\boldsymbol{x}_1) + g(\boldsymbol{x}_2) \tag{2.5a}$$

$$\text{subject to} \qquad \boldsymbol{x}_1 = \boldsymbol{x}_2 \tag{2.5b}$$

leading to the steps

$$\boldsymbol{x}_1 \leftarrow \text{prox}_{\mu^{-1} f}(\boldsymbol{x}_2 - \mu^{-1}\boldsymbol{z}) \tag{2.6a}$$

$$\boldsymbol{x}_2 \leftarrow \text{prox}_{\mu^{-1} g}(\boldsymbol{x}_1 + \mu^{-1}\boldsymbol{z}) \tag{2.6b}$$

$$\boldsymbol{z} \leftarrow \boldsymbol{z} + \mu(\boldsymbol{x}_1 - \boldsymbol{x}_2). \tag{2.6c}$$

where μ is a fixed tuning parameter. In this paper, we pursue a proximal numerical method inspired by Douglas-Rachford splitting [10, 11] to solve the class of optimization problems of the form (2.1).

2.5 Proposed Operator Splitting Method

In this work, we propose a novel first order numerical algorithm which is inspired by Douglas-Rachford splitting to target large scale conic programs given in 2.1.

$$\text{minimize} \qquad f(\boldsymbol{x}) + g(\boldsymbol{x}) \tag{2.7}$$

where $f, g : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ are indicator functions and defined as

$$f(\boldsymbol{x}) \triangleq \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathcal{K} \\ \infty & \text{otherwise} \end{cases} \quad \text{and} \quad g(\boldsymbol{x}) \triangleq \begin{cases} \boldsymbol{c}^\top \boldsymbol{x} & \text{if } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ \infty & \text{otherwise} \end{cases}$$

Projection and absolute value operators serves the purpose of proximal parameters in this method. To this end, the projection and absolute value operators are defined as follows.

Definition 1. *For any proper cone $\mathcal{C} \in \mathbb{R}^n$, define the projection operator* $\text{proj}_{\mathcal{C}} : \mathbb{R}^n \to \mathcal{C}$ *as*

$$\text{proj}_{\mathcal{C}}(\boldsymbol{v}) \triangleq \text{argmin}_{\boldsymbol{u} \in \mathcal{C}} \ \|\boldsymbol{u} - \boldsymbol{v}\|_2.$$

Additionally, define the absolute value operator $\text{abs}_{\mathcal{C}} : \mathbb{R}^n \to \mathbb{R}^n$ *associated with* $\mathcal{C}$ *as*

$$\text{abs}_{\mathcal{C}}(\boldsymbol{x}_0) \triangleq 2\text{proj}_{\mathcal{C}}(\boldsymbol{x}_0) - \boldsymbol{x}_0.$$

Algorithm 1 details the proposed first-order numerical method for solving (2.1). The input of the algorithm are the data matrices $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, \mathcal{K})$ of original optimization problem as shown in 2.1.

Algorithm 1

Input: $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, \mathcal{K})$, fixed $\mu > 0$, and initial point $\boldsymbol{s} \in \mathbb{R}^n$

1: $\mathcal{A} := \text{range}\{\boldsymbol{A}^\top\}$

2: $\boldsymbol{d} := \boldsymbol{A}^\dagger \boldsymbol{b} + \dfrac{\mu}{2}\left(\text{abs}_{\mathcal{A}}(\boldsymbol{c}) - \boldsymbol{c}\right)$

3: **repeat**

4: $\quad \boldsymbol{p} \leftarrow \text{abs}_{\mathcal{K}}(\boldsymbol{s})$

5: $\quad \boldsymbol{r} \leftarrow \text{abs}_{\mathcal{A}}(\boldsymbol{p})$

6: $\quad \boldsymbol{s} \leftarrow \dfrac{\boldsymbol{s}}{2} - \dfrac{\boldsymbol{r}}{2} + \boldsymbol{d}$

7: **until** stopping criteria is met.

Output: $\boldsymbol{x} \leftarrow \dfrac{\boldsymbol{p}+\boldsymbol{s}}{2}, \quad \boldsymbol{z} \leftarrow \dfrac{\boldsymbol{p}-\boldsymbol{s}}{2\mu}$

- **Steps 1 - 2:** Solving linear system of equation in each iteration is one of the requirement of iterative algorithms. Instead of solving linear system in each iteration, we solve the system in this step and reuse the factors in subsequent iterations.
- **Steps 4-6:** These iterative steps of the algorithms includes cheap arithmetic operations and can be carried out in parallel. The projection of cones is performed and the structure of each cone is preserved.

Theorem 1. *Let* $\{\boldsymbol{s}^l\}_{l=0}^{\infty}$ *and* $\{\boldsymbol{p}^l\}_{l=0}^{\infty}$ *denote the sequence of vectors generated by Algorithm 1. Then we have*

$$\lim_{l\to\infty} \frac{\boldsymbol{p}^l + \boldsymbol{s}^l}{2} = \bar{\boldsymbol{x}} \qquad \text{and} \qquad \lim_{l\to\infty} \frac{\boldsymbol{p}^l - \boldsymbol{s}^l}{2\mu} = \bar{\boldsymbol{z}} \tag{2.8}$$

where $\bar{\boldsymbol{x}}$ *and* $\bar{\boldsymbol{z}}$ *are a pair of primal and dual solutions for problems* (2.1) *and* (2.2)*, respectively.*

Proof. Please see the Appendix for the proof. □

Despite its potential for massive parallelization for both CPU and GPU architectures, in and of itself, Algorithm (2) may not offer any advantages over the common-practice Alternating Direction Method of Multipliers (ADMM) and Douglas-Rachford (DR) splitting . However, as we will demonstrate next, Algorithm (2) enables us to perform adaptive conditioning to achieve much faster convergence speed in comparison with the standard pre-conditioning methods.

2.6 Conclusion

We introduced a first order numerical algorithm based on operator splitting method to solve large scale conic optimization problems. The proposed method is inspired from Douglas-Rachford splitting and distribute the objective function to solve problem in a distributed and parallelizable manner. The proposed approach is computationally inexpensive, requires less memory and can be implemented on distributed computing architecture such as GPU for better performance. The convergence proof of the algorithm is provided.

2.7 Proofs

In order to prove Theorem 1, we first give a few lemmas.

Lemma 1. *Define the notation* $|\cdot|_\star$ *as*

$$|\cdot|_\star \triangleq \mathrm{abs}_{\mathcal{A}}(\mathrm{abs}_{\mathcal{K}}(\cdot)). \tag{2.9}$$

Then for every $\boldsymbol{u} \in \mathbb{R}^n$*, we have*

$$\||\boldsymbol{u}|_\star\|_2 = \|\boldsymbol{u}\|_2 \tag{2.10}$$

and for every pair $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$*, we have*

$$|\boldsymbol{u}|_\star^\top |\boldsymbol{v}|_\star \geq \boldsymbol{u}^\top \boldsymbol{v}. \tag{2.11}$$

Proof. The proof follows directly from the definition of $\mathrm{abs}_{\mathcal{A}}$ and $\mathrm{abs}_{\mathcal{K}}$. □

Lemma 2. *Let* $\boldsymbol{x}^{\mathrm{opt}}$ *and* $\boldsymbol{z}^{\mathrm{opt}}$ *denote a pair of primal and dual solutions for problems* (2.1) *and* (2.2)*. Then*

$$\boldsymbol{s}^{\mathrm{opt}} \triangleq \boldsymbol{x}^{\mathrm{opt}} - \mu \boldsymbol{z}^{\mathrm{opt}} \tag{2.12}$$

is a fixed point of Algorithm (1) *and*

$$\boldsymbol{d} = \frac{\boldsymbol{s}^{\mathrm{opt}} + |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2}. \tag{2.13}$$

Proof. According to the Karush–Kuhn–Tucker (KKT) optimality conditions, there exists $\boldsymbol{y}^{\mathrm{opt}} \in \mathbb{R}^m$, for which

$$\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}} + \boldsymbol{A}^\top \boldsymbol{y}^{\mathrm{opt}} = 0 \tag{2.14a}$$

$$\boldsymbol{A}\boldsymbol{x}^{\mathrm{opt}} = \boldsymbol{b} \tag{2.14b}$$

$$(\boldsymbol{x}^{\mathrm{opt}})^\top \boldsymbol{z}^{\mathrm{opt}} = 0, \qquad \boldsymbol{x}^{\mathrm{opt}} \in \mathcal{K}, \qquad \boldsymbol{z}^{\mathrm{opt}} \in \mathcal{K}. \tag{2.14c}$$

Define

$$\boldsymbol{p}^{\mathrm{opt}} \triangleq \boldsymbol{x}^{\mathrm{opt}} + \mu \boldsymbol{z}^{\mathrm{opt}}, \tag{2.15}$$

then according to (2.14c), we have:

$$\boldsymbol{p}^{\mathrm{opt}} = \mathrm{abs}_{\mathcal{K}}(\boldsymbol{s}^{\mathrm{opt}}). \tag{2.16}$$

Hence

$$\boldsymbol{d} - \frac{\boldsymbol{s}^{\mathrm{opt}} + |\boldsymbol{s}^{\mathrm{opt}}|_{\star}}{2} = \tag{2.17a}$$

$$\boldsymbol{A}^{\dagger}\boldsymbol{b} + \frac{\mu(\mathrm{abs}_{\mathcal{A}}(\boldsymbol{c}) - \boldsymbol{c})}{2} - \frac{\boldsymbol{s}^{\mathrm{opt}} + |\boldsymbol{s}^{\mathrm{opt}}|_{\star}}{2} \overset{(2.16)}{=} \tag{2.17b}$$

$$\boldsymbol{A}^{\dagger}\boldsymbol{b} + \frac{\mu(\mathrm{abs}_{\mathcal{A}}(\boldsymbol{c}) - \boldsymbol{c})}{2} - \frac{\boldsymbol{s}^{\mathrm{opt}} + \mathrm{abs}_{\mathcal{A}}(\boldsymbol{p}^{\mathrm{opt}})}{2} = \tag{2.17c}$$

$$\boldsymbol{A}^{\dagger}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{\mathrm{opt}}) - \mu(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}} - \boldsymbol{A}^{\dagger}\boldsymbol{A}(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}})) \overset{(2.14b)}{=} \tag{2.17d}$$

$$\mu(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}} - \boldsymbol{A}^{\dagger}\boldsymbol{A}(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}})) \overset{(2.14a)}{=} \tag{2.17e}$$

$$\mu(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}} + \boldsymbol{A}^{\dagger}\boldsymbol{A}\boldsymbol{A}^{\top}\boldsymbol{y}^{\mathrm{opt}}) = \tag{2.17f}$$

$$\mu(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}} + \boldsymbol{A}^{\top}\boldsymbol{y}^{\mathrm{opt}}) \overset{(2.14a)}{=} \boldsymbol{0}_n, \tag{2.17g}$$

which concludes (2.13). Now according to the steps of Algorithm 1, one can immediately conclude that $\boldsymbol{s}^{\mathrm{opt}}$ is a fixed point. □

Lemma 3. *Let $\{\boldsymbol{s}^l\}_{l=0}^{\infty}$ be the sequence generated by Algorithm* (1) *and define $\boldsymbol{s}^{\mathrm{opt}} \triangleq \boldsymbol{x}^{\mathrm{opt}} - \mu\boldsymbol{z}^{\mathrm{opt}}$, where $\boldsymbol{x}^{\mathrm{opt}}$ and $\boldsymbol{z}^{\mathrm{opt}}$ denote an arbitrary pair of primal and dual solutions for problems* (2.1) *and* (2.2)*. Then,*

a) the sequence $\{\|\boldsymbol{s}^l - \boldsymbol{s}^{\mathrm{opt}}\|_2\}_{l=0}^{\infty}$ is convergent,

b) and the sequence $\{\|\boldsymbol{s}^{l+1} - \boldsymbol{s}^l\|_2\}_{l=0}^{\infty}$ converges to zero.

Proof. According to the steps of Algorithm 1, we have

$$\boldsymbol{s}^{l+1} = \frac{\boldsymbol{s}^l - |\boldsymbol{s}^l|_{\star}}{2} + \boldsymbol{d} \tag{2.18}$$

and due to (2.13):

$$\boldsymbol{s}^{l+1} = \frac{\boldsymbol{s}^l + \boldsymbol{s}^{\text{opt}}}{2} - \frac{|\boldsymbol{s}^l|_\star - |\boldsymbol{s}^{\text{opt}}|_\star}{2}. \tag{2.19}$$

Hence

$$\|\boldsymbol{s}^{l+1} - \boldsymbol{s}^l\|_2^2 + \|\boldsymbol{s}^{l+1} - \boldsymbol{s}^{\text{opt}}\|_2^2 - \|\boldsymbol{s}^l - \boldsymbol{s}^{\text{opt}}\|_2^2 \overset{(2.19)}{=} \tag{2.20a}$$

$$\|\frac{\boldsymbol{s}^l - \boldsymbol{s}^{\text{opt}}}{2} + \frac{|\boldsymbol{s}^l|_\star - |\boldsymbol{s}^{\text{opt}}|_\star}{2}\|_2^2 + \tag{2.20b}$$

$$\|\frac{\boldsymbol{s}^l - \boldsymbol{s}^{\text{opt}}}{2} - \frac{|\boldsymbol{s}^l|_\star - |\boldsymbol{s}^{\text{opt}}|_\star}{2}\|_2^2 - \|\boldsymbol{s}^l - \boldsymbol{s}^{\text{opt}}\|_2^2 = \tag{2.20c}$$

$$\frac{\||\boldsymbol{s}^l|_\star - |\boldsymbol{s}^{\text{opt}}|_\star\|_2^2}{2} - \frac{\|\boldsymbol{s}^l - \boldsymbol{s}^{\text{opt}}\|_2^2}{2} \overset{(2.10)}{=} \tag{2.20d}$$

$$(\boldsymbol{s}^l)^\top \boldsymbol{s}^{\text{opt}} - |\boldsymbol{s}^l|_\star^\top |\boldsymbol{s}^{\text{opt}}|_\star \overset{(2.11)}{\leq} 0, \tag{2.20e}$$

which concludes that $\{\|\boldsymbol{s}^l - \boldsymbol{s}^{\text{opt}}\|_2\}_{l=0}^\infty$ is nonincreasing and convergent. Additionally, (2.20) concludes that

$$\sum_{l=0}^{\infty} \|\boldsymbol{s}^{l+1} - \boldsymbol{s}^l\|_2^2 \;\leq\; \|\boldsymbol{s}^0 - \boldsymbol{s}^{\text{opt}}\|_2^2 \tag{2.21}$$

which means that $\{\|\boldsymbol{s}^{l+1} - \boldsymbol{s}^l\|_2\}_{l=0}^\infty$ converges to zero. □

Proof of theorem 1. According to the first part of Lemma 3, the sequence $\{\boldsymbol{s}^l\}_{l=0}^\infty$ is bounded and therefore, it has a convergent subsequence $\{\bar{\boldsymbol{s}}_l\}_{l=0}^\infty$, where

$$\lim_{l\to\infty} \bar{\boldsymbol{s}}_l = \bar{\boldsymbol{s}}. \tag{2.22}$$

Define

$$\bar{\boldsymbol{x}} \triangleq \frac{\mathrm{abs}_{\mathcal{K}}(\bar{\boldsymbol{s}}) + \bar{\boldsymbol{s}}}{2} \qquad \text{and} \qquad \bar{\boldsymbol{z}} \triangleq \frac{\mathrm{abs}_{\mathcal{K}}(\bar{\boldsymbol{s}}) - \bar{\boldsymbol{s}}}{2\mu}. \tag{2.23}$$

In order to show that $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{z}}$ are a pair of primal and dual solutions, we prove the following KKT optimality criteria:

$$\bar{\boldsymbol{z}} - \boldsymbol{c} \in \mathrm{range}\{\boldsymbol{A}^\top\} \tag{2.24a}$$

$$\boldsymbol{A}\bar{\boldsymbol{x}} = \boldsymbol{b} \tag{2.24b}$$

$$\bar{\boldsymbol{x}}^\top \bar{\boldsymbol{z}} = 0, \qquad \bar{\boldsymbol{x}} \in \mathcal{K}, \qquad \bar{\boldsymbol{z}} \in \mathcal{K}. \tag{2.24c}$$

Condition (2.24c) follows directly from the definition (2.23). Additionally, according to the second part of Lemma 3,

$$\mathbf{0}_n = \lim_{l\to\infty} \boldsymbol{s}^{l+1} - \boldsymbol{s}^l \tag{2.25a}$$

$$= \lim_{l\to\infty} \boldsymbol{d} - \frac{\boldsymbol{s}^l + |\boldsymbol{s}^l|_\star}{2} \tag{2.25b}$$

$$= \lim_{l\to\infty} \boldsymbol{d} - \frac{\bar{\boldsymbol{s}}^l + |\bar{\boldsymbol{s}}^l|_\star}{2} \overset{(2.22)}{=} \boldsymbol{d} - \frac{\bar{\boldsymbol{s}} + |\bar{\boldsymbol{s}}|_\star}{2} \tag{2.25c}$$

Hence,

$$\frac{\bar{\boldsymbol{s}} + |\bar{\boldsymbol{s}}|_\star}{2} = \boldsymbol{d} \overset{(2.13)}{=} \frac{\boldsymbol{s}^{\mathrm{opt}} + |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2} \tag{2.26}$$

Therefore,

$$\begin{aligned}
\mathbf{0}_n &= \frac{\bar{\boldsymbol{s}} + |\bar{\boldsymbol{s}}|_\star}{2} - \frac{\boldsymbol{s}^{\mathrm{opt}} + |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2} && \text{(2.27a)}\\
&= \frac{\bar{\boldsymbol{s}} - \boldsymbol{s}^{\mathrm{opt}} + (2\boldsymbol{A}^\dagger\boldsymbol{A} - \boldsymbol{I}_n)(\mathrm{abs}_{\mathcal{K}}(\bar{\boldsymbol{s}}) - \mathrm{abs}_{\mathcal{K}}(\boldsymbol{s}^{\mathrm{opt}}))}{2} && \text{(2.27b)}\\
&= \frac{(\bar{\boldsymbol{x}} - \boldsymbol{x}^{\mathrm{opt}}) - \mu(\bar{\boldsymbol{z}} - \boldsymbol{z}^{\mathrm{opt}})}{2}\\
&\quad + \frac{(2\boldsymbol{A}^\dagger\boldsymbol{A} - \boldsymbol{I}_n)[(\bar{\boldsymbol{x}} - \boldsymbol{x}^{\mathrm{opt}}) + \mu(\bar{\boldsymbol{z}} - \boldsymbol{z}^{\mathrm{opt}})]}{2} && \text{(2.27c)}\\
&= \boldsymbol{A}^\dagger(\boldsymbol{A}\bar{\boldsymbol{x}} - \boldsymbol{A}\boldsymbol{x}^{\mathrm{opt}}) - \mu(\boldsymbol{I}_n - \boldsymbol{A}^\dagger\boldsymbol{A})(\bar{\boldsymbol{z}} - \boldsymbol{z}^{\mathrm{opt}}). && \text{(2.27d)}
\end{aligned}$$

Now, pre-multiplication by $\boldsymbol{A}$ concludes that

$$\boldsymbol{A}\bar{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}^{\mathrm{opt}} = \boldsymbol{b}. \tag{2.28}$$

Similarly,

$$\begin{aligned}
\mathbf{0}_n &= \frac{\bar{\boldsymbol{s}} + |\bar{\boldsymbol{s}}|_\star}{2} - \boldsymbol{d} && \text{(2.29a)}\\
&= \boldsymbol{A}^\dagger(\boldsymbol{A}\bar{\boldsymbol{x}} - \boldsymbol{b}) - \mu(\boldsymbol{I}_n - \boldsymbol{A}^\dagger\boldsymbol{A})(\bar{\boldsymbol{z}} - \boldsymbol{c}) && \text{(2.29b)}\\
&= \mu(\boldsymbol{I}_n - \boldsymbol{A}^\dagger\boldsymbol{A})(\boldsymbol{c} - \bar{\boldsymbol{z}}) && \text{(2.29c)}
\end{aligned}$$

which concludes (2.24a). Therefore $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{z}}$ are primal and dual optimal, and according to the first part of Lemma 3, the following limit exists:

$$\lim_{l\to\infty} \|\boldsymbol{s}^l - \bar{\boldsymbol{s}}\|_2 = \lim_{l\to\infty} \|\bar{\boldsymbol{s}}^l - \bar{\boldsymbol{s}}\|_2 = 0 \tag{2.30}$$

which completes the proof. □

CHAPTER 3

Rapid Convergence of First Order Methods via Adaptive Conditioning

3.1 Introduction

First order methods are considered very sensitive to condition number of problem data and parameter selection, and consequently have limitations in achieving higher accuracy within a reasonable number of iterations [11, 18, 24, 25]. Although first order operator splitting methods have been studied extensively in recent years for solving large scale conic programs for different applications but until the recent past, very few efforts are made to study the convergence rate [26, 27]. As an attempt to solve the convergence rate issues, recently serious efforts have been made to make first order algorithms more robust and practical for real-world applications [22, 25, 28–33]. A line search method is proposed in [34] to accelerate convergence. In [35], a global linear convergence proof is given under strict convexity and Lipschitz gradient condition on one function. A global linear convergence approach and metric selection approach shown in [11] under strong convexity and smoothness conditions. Researchers have proposed several acceleration techniques to expedite the convergence speed of ADMM. Adaptive penalty scheme is introduced in [23, 36] to automatically tune the penalty parameter. In [37, 38], Anderson acceleration (AA) is applied to improve the convergence of local-global solver and ADMM with application to geometry optimization and physics simulation problems. The authors in [39], applied the type-I variant of Anderson acceleration [40] to splitting conic solver (SCS) [16] to solve conic optimization problems and improved the terminal convergence. A new framework known as SuperSCS is introduced in [13] by combining SCS solver with original type-II AA

to solve large cone problems and it is shown that the new approach performs better than the original SCS solver. Type-II Anderson acceleration Douglas-Rachford splitting (A2DR) algorithm is proposed in [12], to show the rapid convergence or provide infeasibility/unboundedness certificates. However, most of these techniques works reasonably well under limited scenarios, particular conditions, and for a very specific problem structures and yield no tangible benefits for any general class of problems. Improvements from these techniques are very limited and has very mild effect on the convergence due to the nature of accelerated algorithms. Moreover, these techniques fail to achieve a higher accuracy.

Operator splitting methods heavily rely on the input problem data matrices, pre-conditioning, solution polishing, and step size parameter selection [15, 17, 25]. Parameter selection for global convergence is still a challenge to be addressed [11, 18]. Despite the scalability and computational advantages, these methods suffer from slow terminal convergence, and are highly sensitive to problem condition number, hence, cannot be applied to many practical problems [12, 29, 33]. There is a dire need to develop a general purpose, and reliable first order algorithm that encapsulates the benefits of simple inexpensive iterations and scaling properties of first order algorithm, as well as providing the highly reliable and accurate solutions similar to that of interior point methods.

3.1.1 Contributions

In this work, we first show that the condition number of data matrices has no significant effect on convergence of general first order methods and this is the major impediment for achieving highly accurate results with operator splitting methods. Furthermore, we propose a new operating splitting method where each iteration requires simple arithmetic operations, leads to parallel and distributive implementation,

scales gracefully for very large cone programs and provides a very accurate solutions which is beyond the reach of other first order solvers. Moreover, in conjunction with massively parallelizable and cheap iterative algorithm we propose a heuristic policy to scale the data matrices in such a way that the combined algorithm ensures the global convergence and achieves a high accuracy within a tens of iterations. In short, the proposed algorithm enjoys the benefits of first order algorithms such as low per iteration cost, scalability for very large problems, parallel and distributed implementations, and at the same time achieves the higher accuracy level of interior point methods. The major contributions and novelty of this paper are as follows

1. We illustrate that a smaller condition number does not necessarily guarantee the faster convergence as the problem data matrices with a higher condition number can converge faster.
2. We propose a heuristic adaptive conditioning policy to obtain accurate solutions in comparison with other first order algorithms and a proof is provided to guarantee the convergence of algorithm.
3. We apply the proposed algorithm on graphics processing unit (GPU) to benefit the simple arithmetic operations in each iteration.
4. A wide range of tests are conducted on different conic programs and results are compared with several first order and interior point methods to justify the claims of scalability, efficiency and accuracy.

Rest of the chapter is organized as follows. Some preliminaries of cone programming and definitions are presented in section 3.2. The effect of preconditioning and the need for proposed adaptive conditioning is illustrated in section 3.3 by providing a numerical example. In section 3.4, we investigate the conditioning procedure to accelerate the convergence and provide an algorithm for adaptive conditioning. We compare the performance of proposed algorithm and adaptive conditioning in section

3.5, by solving a wide range of problems and comparing the results with commonly used solvers, and section 3.6 concludes the paper.

3.1.2 Notations

Notations $\mathbb{R}$ and $\mathbb{N}$ denote the set of real and natural numbers, respectively. Matrices and vectors are represented by bold uppercase, and bold lowercase letters, respectively. Notation $\|\cdot\|_2$ refers to ℓ_2 norm of either matrix or vector depending on the context and $|\cdot|$ represents the absolute value. The symbol $(\cdot)^\top$ represent the transpose operators. The notations $\boldsymbol{I}_n$ refer to the $n \times n$ identity matrix. The symbol $\mathcal{K}$ is used to describe different types of cones used in this paper. The superscript $(\cdot)^{\text{opt}}$ refers to the optimal solution of optimization problem. The notation $(\cdot)^\dagger$ denotes the Moore–Penrose pseudoinverse of transpose of a matrix. The symbol $\mathcal{L}$ represent the set of values to apply adaptive conditioning. The notation $\text{diag}\{\cdots\}$ represent the diagonal elements of a diagonal matrix. The symbols $\text{SK}, \text{AC}, \text{CS}$, are used to refer Sinkhorn-Knopp, adaptive conditioning and competing solver, respectively. The symbols ε^{abs} and ε^{rel} are used for absolute and relative tolerance, respectively.

3.2 Preliminaries

In this paper, we consider conic optimization problems with a linear objective, subject to a set of affine and second-order conic constraints. The primal formulation under study can be cast as:

$$\underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} \quad \boldsymbol{c}^\top \boldsymbol{x} \tag{3.1a}$$

$$\text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{3.1b}$$

$$\boldsymbol{x} \in \mathcal{K} \tag{3.1c}$$

where $\boldsymbol{c} \in \mathbb{R}^n, \boldsymbol{A} \in \mathbb{R}^{m \times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$ are given and $\boldsymbol{x} \in \mathbb{R}^n$ is the unknown optimization variable. Additionally, $\mathcal{K} \triangleq \mathcal{K}_{n_1} \times \mathcal{K}_{n_2} \times \cdots \times \mathcal{K}_{n_k} \subseteq \mathbb{R}^n$, where each $\mathcal{K}_{n_i} \subseteq \mathbb{R}^{n_i}$ is a Lorentz cone of size n_i, i.e.,

$$\mathcal{K}_{n_i} \triangleq \left\{ \boldsymbol{w} \in \mathbb{R}^{n_i} \,|\, w_1 \geq \left\| [w_2, \ldots, w_{n_i}] \right\|_2 \right\},$$

and $n_1 + n_2 + \ldots + n_k = n$.

The corresponding dual formulation of (3.1) is

$$\begin{aligned}
\underset{\boldsymbol{y} \in \mathbb{R}^m, \boldsymbol{z} \in \mathbb{R}^n}{\text{maximize}} \quad & \boldsymbol{b}^\top \boldsymbol{y} && \text{(3.2a)} \\
\text{subject to} \quad & \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{z} = \boldsymbol{c} && \text{(3.2b)} \\
& \boldsymbol{z} \in \mathcal{K} && \text{(3.2c)}
\end{aligned}$$

where $\boldsymbol{y}$ and $\boldsymbol{z}$ are dual variables associated with the constraints (3.1b) and (3.1c), respectively.

3.3 State of the Art Preconditioning Methods

One of the major drawbacks of first-order numerical methods is their sensitivity to the problem conditioning [15, 18, 41]. Hence, it is common practice to reformulate problem (3.1) with respect to new parameters

$$\hat{\boldsymbol{A}} \triangleq \boldsymbol{DAE}, \quad \hat{\boldsymbol{b}} \triangleq \boldsymbol{Db}, \quad \text{and} \quad \hat{\boldsymbol{c}} \triangleq \boldsymbol{Ec} \tag{3.3}$$

and new proxy variables

$$\hat{\boldsymbol{x}} = \boldsymbol{E}^{-1} \boldsymbol{x} \quad \text{and} \quad \hat{\boldsymbol{z}} = \boldsymbol{E}^\top \boldsymbol{z} \tag{3.4}$$

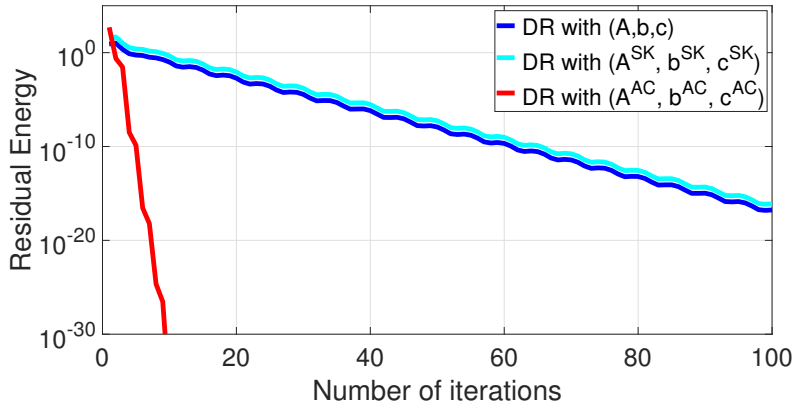
where $\boldsymbol{D} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{E} \in \mathbb{R}^{n\times n}$ are tuned to improve convergence speed. The process of finding an appropriate $\boldsymbol{D}$ and $\boldsymbol{E}$ to improve the performance of a first-order numerical algorithm is regarded as preconditioning of data.

Theoretical and practical evidence show that choices of $\boldsymbol{D}$ and $\boldsymbol{E}$ that result in smaller condition number for $\hat{\boldsymbol{A}}$ lead to better performance in both precision and convergence rate of first-order numerical algorithms [24, 41–43]. As a result, over the past decade, several research directions have pursued preconditioning methods such as heuristic diagonal scaling with the aim of reducing the condition number of $\hat{\boldsymbol{A}}$ [42, 44]. To this end, a number of matrix equilibration heuristics such as Sinkhorn-Knopp and Ruiz methods have been proposed in [15, 17, 18, 45] that indirectly influence the condition number of $\hat{\boldsymbol{A}}$ by equalizing ℓ_p norm for each row through diagonal choices of $\boldsymbol{D}$ and $\boldsymbol{E}$.

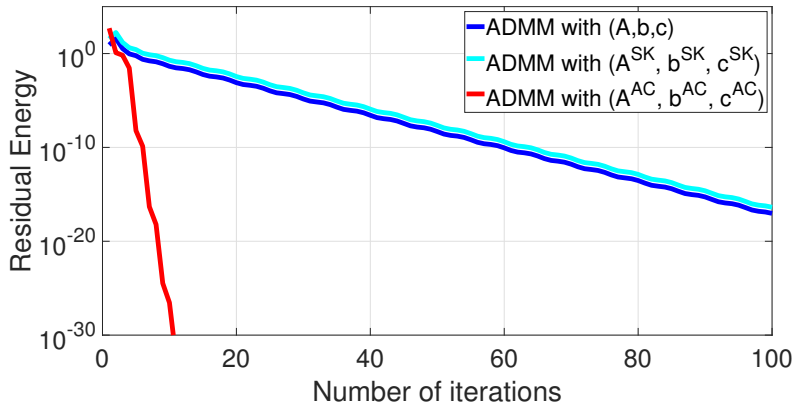
In this paper, we pursue an alternative approach. We argue that the condition number of $\hat{\boldsymbol{A}}$ is not a reliable indicator of convergence speed for first-order numerical methods and instead, we offer a new approach regarded as *adaptive conditioning*. Before elaborating the details of the proposed procedure, we first give a simple illustrative example through which it is shown that a smaller condition number for the data matrix $\hat{\boldsymbol{A}}$ does not necessarily result in better performance.

3.3.1 Example: The effect of condition number

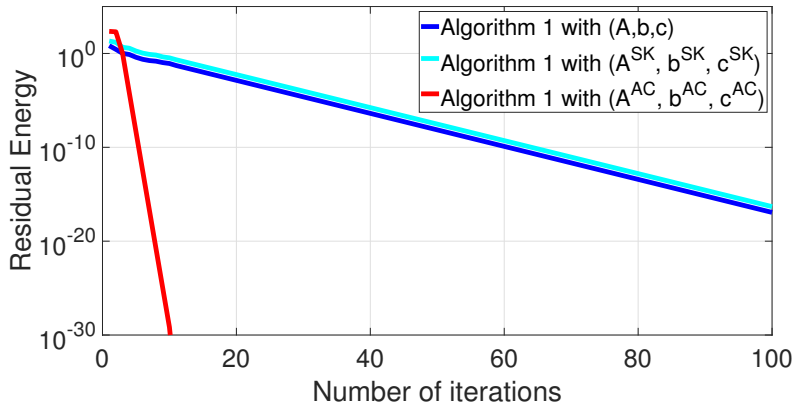
In this example, we provide simple data matrices and compare the effect of different pre-conditioning methods on the convergence of Algorithm (1), DR splitting, and ADMM. The goal is to demonstrate that the condition number of $\hat{\boldsymbol{A}}$ is not a reliable predictor of the convergence speed.



(a)



(b)



(c)

Figure 3-1: The effect of different pre-conditioning methods on the convergence of (a) Douglas-Rachford splitting, (b) Alternating Direction Method of Multipliers, and (c) Algorithm 1.

Consider the following data matrices:

$$\boldsymbol{A} := \begin{bmatrix} 3.57 & 3.45 & 3.33 & 64.24 & -72.76 \\ 3.45 & 3.33 & 3.23 & 95.14 & -23.34 \\ 3.33 & 3.23 & 3.13 & 93.53 & -17.43 \end{bmatrix},$$
$$\boldsymbol{b} := \begin{bmatrix} -10.44 & 20.65 & 22.94 \end{bmatrix}^\top,$$
$$\boldsymbol{c} := \begin{bmatrix} 0.37 & 1.93 & -0.12 & -0.38 & 1.01 \end{bmatrix}^\top,$$

and $\mathcal{K} := \mathbb{R}^5_+$. The corresponding diagonal matrices obtained from regularized Sinkhorn-Knopp algorithm [15] for ℓ_2 norms are

$$\boldsymbol{D}^{\mathrm{SK}} = \mathrm{diag}\{[0.0217, 0.0215, 0.0222]\},$$
$$\boldsymbol{E}^{\mathrm{SK}} = \mathrm{diag}\{[0.4722, 0.4722, 0.4722, 0.4722, 0.4722]\},$$

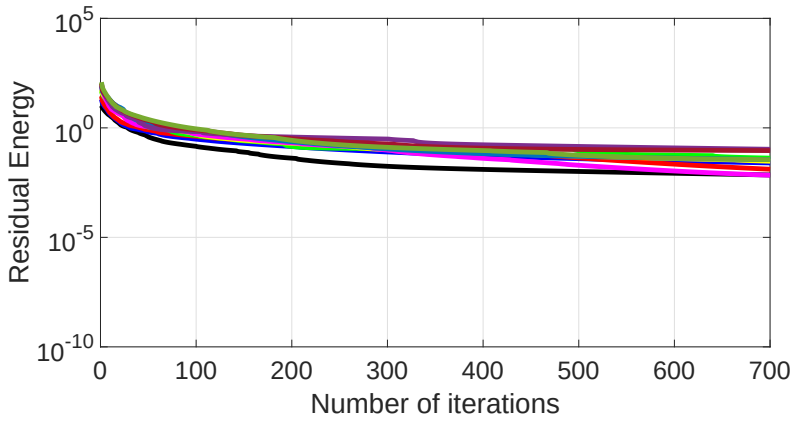
while the proposed heuristic adaptive conditioning results in the following matrices

$$\boldsymbol{D}^{\mathrm{AC}} = \boldsymbol{I}_{3\times 3},$$
$$\boldsymbol{E}^{\mathrm{AC}} = \mathrm{diag}\{[0.0792, 0.0884, 14.5484, 292.9524, 316.2179]\}.$$

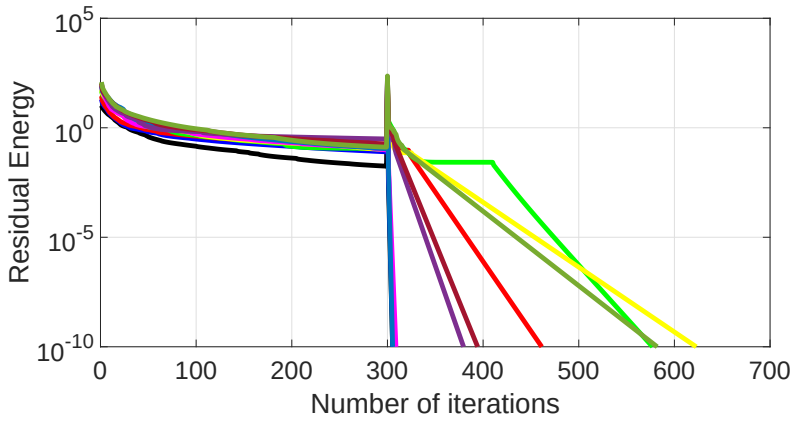
Define

$$\hat{\boldsymbol{A}}^{\mathrm{SK}} := \boldsymbol{D}^{\mathrm{SK}}\boldsymbol{A}\boldsymbol{E}^{\mathrm{SK}} \quad \text{and} \quad \hat{\boldsymbol{A}}^{\mathrm{AC}} := \boldsymbol{D}^{\mathrm{AC}}\boldsymbol{A}\boldsymbol{E}^{\mathrm{AC}}.$$

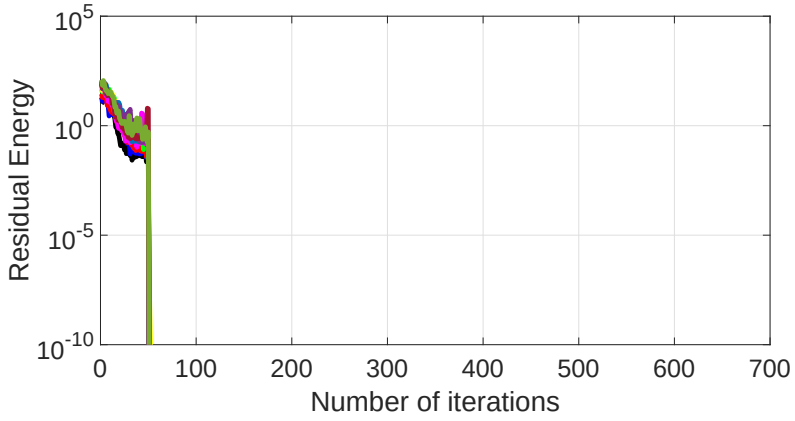
In this case, the condition numbers of $\boldsymbol{A}$, $\hat{\boldsymbol{A}}^{\mathrm{SK}}$, and $\hat{\boldsymbol{A}}^{\mathrm{AC}}$ are equal to 2046.4, 2044.38, and 72079.13, respectively.



(a)



(b)



(c)

Figure 3-2: Convergence of Algorithm (2) for 10 random linear programming instance (distinct color for each instance) with different conditioning steps: (a) No conditioning, i.e., $\mathcal{L} := \varnothing$, (b) One time conditioning at iteration $l = 300$, i.e., $\mathcal{L} := \{300\}$, and (c) Continuous conditioning at the first 50 iterations, i.e., $\mathcal{L} := \{1, 2, \ldots, 50\}$.

Figure (3-1) presents the outcome of DR splitting, ADMM, and Algorithm (1), respectively, with $\mu = 1$ and different pre-conditioning methods. The three cases of no preconditioning, Sinkhorn-Knopp preconditioning, and adaptive conditioning are illustrated in each figure. As demonstrated in Figure 3-1, a lower condition number for the data matrix does not necessarily result in a faster convergence. Motivated by this observation, the following section presents the proposed adaptive conditioning procedure.

3.4 Adaptive Conditioning

In this work, we rely on post multiplication of the data matrix $\boldsymbol{A}$ by a diagonal positive-definite matrix $\boldsymbol{O}$, to enhance the convergence speed of Algorithm 1. The primal problem (3.1) is reformulated as:

$$\begin{aligned} \underset{\hat{\boldsymbol{x}} \in \mathbb{R}^n}{\text{minimize}} \quad & (\boldsymbol{O}^\top \boldsymbol{c})^\top \hat{\boldsymbol{x}} && \text{(3.8a)} \\ \text{subject to} \quad & (\boldsymbol{A}\boldsymbol{O})\hat{\boldsymbol{x}} = \boldsymbol{b} && \text{(3.8b)} \\ & \hat{\boldsymbol{x}} \in \boldsymbol{O}^{-1}\mathcal{K} && \text{(3.8c)} \end{aligned}$$

and the dual problem (3.2) as:

$$\begin{aligned} \underset{\boldsymbol{y} \in \mathbb{R}^m, \hat{\boldsymbol{z}} \in \mathbb{R}^n}{\text{maximize}} \quad & \boldsymbol{b}^\top \boldsymbol{y} && \text{(3.9a)} \\ \text{subject to} \quad & (\boldsymbol{A}\boldsymbol{O})^\top \boldsymbol{y} + \hat{\boldsymbol{z}} = \boldsymbol{O}^\top \boldsymbol{c} && \text{(3.9b)} \\ & \hat{\boldsymbol{z}} \in \boldsymbol{O}^\top \mathcal{K}^* && \text{(3.9c)} \end{aligned}$$

where

$$\hat{\boldsymbol{x}} \triangleq \boldsymbol{O}^{-1}\boldsymbol{x} \quad \text{and} \quad \hat{\boldsymbol{z}} \triangleq \boldsymbol{O}^{\top}\boldsymbol{z} \tag{3.10}$$

are proxy variables.

In contrary to the existing practice that focuses on the condition number of the data matrix, we continuously update the matrix $\boldsymbol{O}$ according to a prespecified policy to improve the convergence speed. This heuristic procedure is detailed in Algorithm 2. As illustrated in Figure 3-3, the intuitive reason behind the proposed adaptive conditioning is to equalize the rate of convergence for elements of $\boldsymbol{s}$. The steps 4 to 15 of Algorithm 2 serve this purpose

- **Step 4:** Adaptive conditioning can be done based on a user-defined criteria or in the simplest case, at a set of user-defined iterations $\mathcal{L}$.
- **Step 5 and 11:** New coefficients are calculated for each cone to equalize the speed of convergence for the elements of $\boldsymbol{x}$ and $\boldsymbol{z}$. Note that since the vectors $\boldsymbol{x}$ and $\boldsymbol{z}$ are complementary at optimality, the elements of $\boldsymbol{o}$ can be very large or very small numbers and that is the motivation behind the normalization step 11.
- **Steps 12 and 13:** These two steps are concerned with the adjustments of the proximal operators and the vector $\boldsymbol{d}$, respectively.
- **Step 14:** This step casts the vector $\boldsymbol{s}$ into the new space so that current progress is continued.

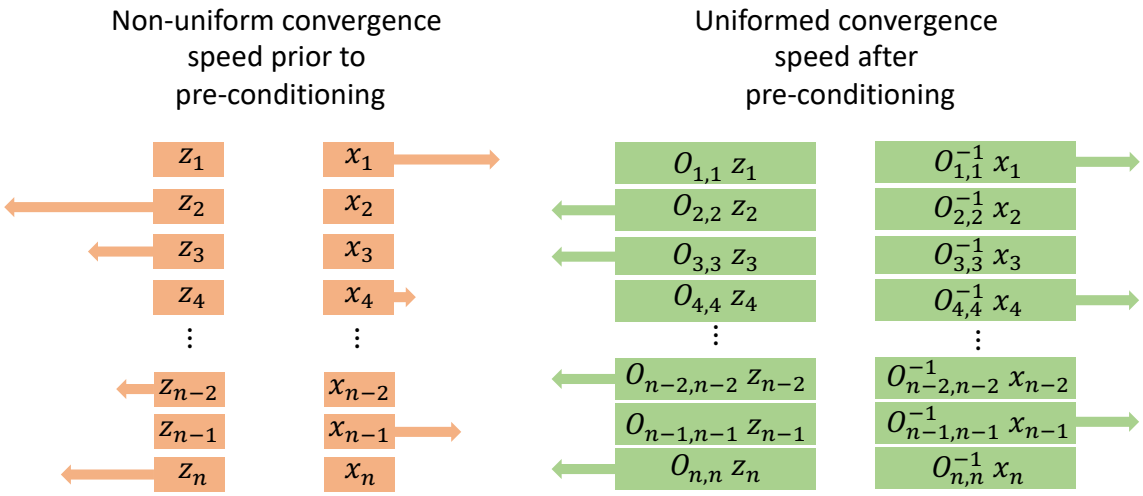


Figure 3-3: Intuitive reason behind adaptive conditioning to equalize the convergence speed

The next example demonstrates the effectiveness of the proposed adaptive conditioning approach on random instances of linear programming (LP).

3.4.1 Example: The choice of conditioning steps

This case study is concerned with the effect of conditioning steps on the convergence behavior of Algorithm (2). We consider three cases:

- No conditioning, i.e., $\mathcal{L} := \varnothing$,
- One time conditioning at iteration $l = 300$, i.e., $\mathcal{L} := \{300\}$,
- Continuous conditioning at the first 50 iterations, i.e., $\mathcal{L} := \{1, 2, \ldots, 50\}$.

We generated 10 random instances of linear programming with 100 variables and 80 linear constraints whose data are chosen such that:

- The elements of $\boldsymbol{A} \in \mathbb{R}^{80\times 100}$ have i.i.d standard normal distribution.
- $\boldsymbol{b} := \boldsymbol{A}\dot{\boldsymbol{x}}$ where the elements of $\dot{\boldsymbol{x}} \in \mathbb{R}^{100}$ have i.i.d uniform distribution from the interval $[0, 1]$.

- The elements of $\boldsymbol{c} \in \mathbb{R}^{100}$ have i.i.d standard normal distribution.
- And $\mathcal{K} = \mathbb{R}_{+}^{100}$.

The effect of adaptive conditioning proposed in Algorithm 2 for $t = 9.2$ is illustrated in Figure 3-2 for all 10 random instance. As demonstrated in the figure, even a one time adaptive conditioning results in significant improvement of convergence speed.

3.5 Numerical Experiments

We provide case studies to evaluate the performance of Algorithm 2 on both CPU and GPU platforms in comparison with the state-of-the-art commercial solvers MOSEK [1], GUROBI [4] as well as the open source software OSQP [18] and POGS [15]. Our case studies consist of randomly generated linear programming (LP) problems and second-order cone programming (SOCP) problems. We conduct experiments on problems with a wide range of variable and constraint numbers to assess both scalability and speed. Additionally, we consider different values for infeasiblity/gap tolerance, to assess the solution accuracy of Algorithm 2. The proposed algorithm and competing solvers are implemented in MATLABR2020a and all the simulations are conducted on a DGX station with 20 2.2 GHz cores, Intel Xeon E5-2698 v4 CPU, with NVIDIA Tesla V100-DGXS-32GB (128 GB total) GPU processor and 256 GB of RAM. The parallel nature of algorithm enables the implementation to take advantage of multi-core CPU processing. Note that our implementation of proposed algorithm in MATLAB utilizes only a single GPU and does not benefit from multiple GPU's of the platform. Moreover, all experiments reported in this paper are not bounded by RAM or GPU memory of DGX station. We used the MATLAB interface of OSQP v0.6.0, MOSEK v9.2.5 and GUROBI v9.0.

In all of the experiments, the stopping criteria of Algorithm 2 is when it exceeds both primal and dual feasibility of the solution produced by the competing solver. In other words, when the following two criteria are met:

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 < \|\boldsymbol{A}\boldsymbol{x}^{\mathrm{CS}} - \boldsymbol{b}\|_2 \tag{3.11a}$$

$$|(\boldsymbol{A}^{\dagger}\boldsymbol{b})^{\top}(\boldsymbol{c} - \boldsymbol{z}) - \boldsymbol{c}^{\top}\boldsymbol{x}| < |\boldsymbol{b}^{\top}\boldsymbol{y}^{\mathrm{CS}} - \boldsymbol{c}^{\top}\boldsymbol{x}^{\mathrm{CS}}| \tag{3.11b}$$

where $\boldsymbol{x}^{\mathrm{CS}}$ and $\boldsymbol{y}^{\mathrm{CS}}$ are primal and dual solutions produced by the competing solver under default settings. In each figure, the experiments are continued until the run time of the competing solver reached a maximum time of 1200 seconds. The maximum time is chosen in such a way that the experiments provide sufficient information to compare the computational time for all solvers.

3.5.1 Linear Programming

3.5.1.1 Comparisons with OSQP, Gurobi, and MOSEK

This cases study is concerned with the class of linear programming problems. The performance of Algorithm 2 is tested in comparison with the solvers, OSQP, Gurobi, and MOSEK. We have generated random LP instances with n ranging from 100 to 30000, and $m = \lfloor 0.8n \rfloor$. The number of nonzero elements of $\boldsymbol{A}$ ranges from 10^4 to 10^9. The data is generated as follows:

- The elements of $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ have i.i.d uniform distribution from the interval $[-1, 1]$.
- $\boldsymbol{b} := \boldsymbol{A}|\dot{\boldsymbol{x}}|$ where the elements of $\dot{\boldsymbol{x}} \in \mathbb{R}^n$ have i.i.d standard normal distribution.
- The elements of $\boldsymbol{c} \in \mathbb{R}^n$ have i.i.d standard normal distribution.
- And $\mathcal{K} = \mathbb{R}^n_+$.

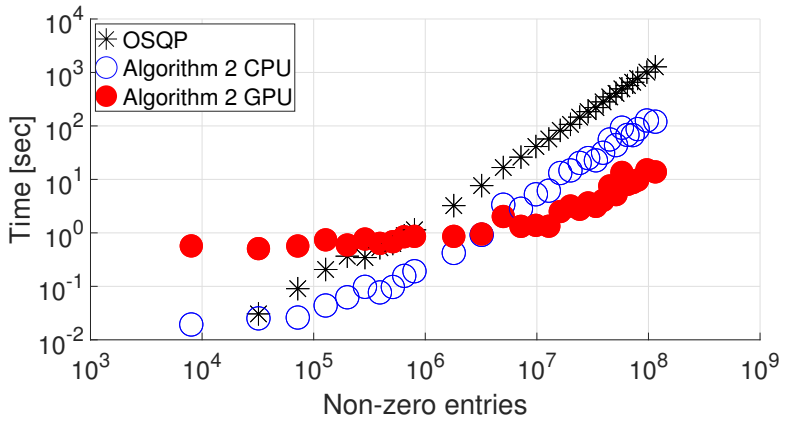
We start applying adaptive conditioning at iterations 300 and apply it once again in every 100 steps, i.e.,

$$\mathcal{L} = \{300, 400, 500, \ldots\}. \tag{3.12}$$

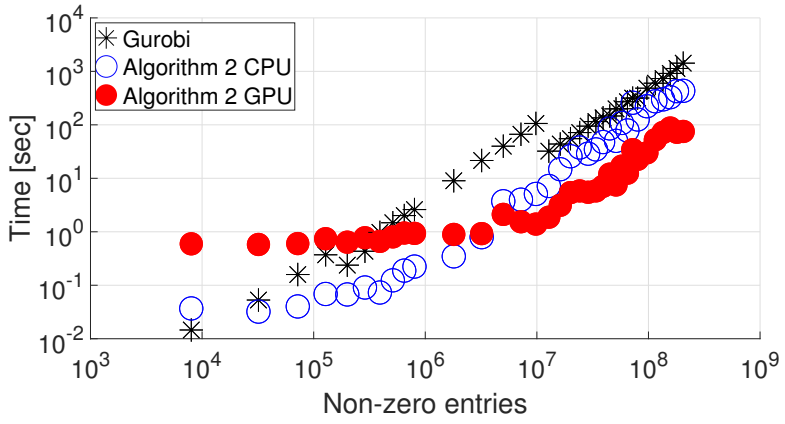
Parameters t and μ are set to 9.2 and 1, respectively. The advantage of using GPU can be seen for large scale problems, when the problem dimension increases, the GPU starts outperforming the other solvers significantly. The default settings are used for all three competing solvers and Algorithm 2 is terminated once a solution with better primal and dual feasibility is obtained, as defined in (3.11). As demonstrated in Figure 3-4, for large instances, we have achieved approximately 3.3 and 19 times improvements for CPU and GPU respectively, in comparison with Gurobi, and more than an order-of-magnitude time improvement in comparison with MOSEK and OSQP with their default settings.

3.5.2 Single vs Multi-core CPU Implementation

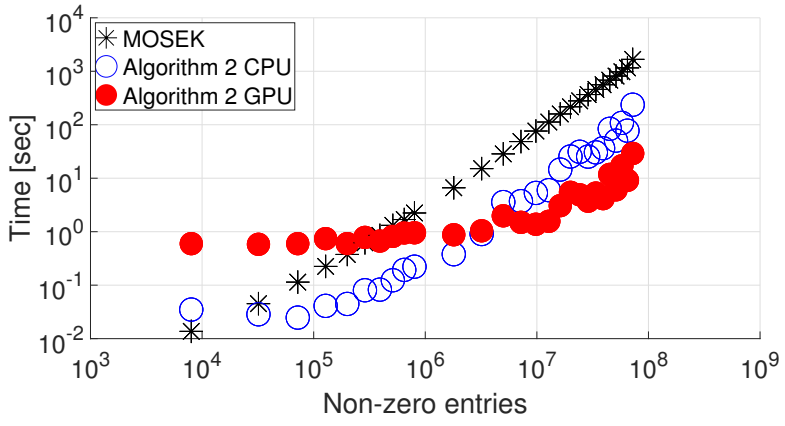
The iterative steps of Algorithms 1 and 2 are completely parallelizable. The parallel steps of algorithms are not only important for the graphics processing unit (GPU) implementation, but also provides the computational benefits for CPU implementation. We demonstrate the parallel processing strength of the proposed algorithm by solving the previous instances of linear programming on both single-core and multi-core (20 cores) CPU settings. Figure 3-5 shows that the multi-core CPU implementation is approximately 10 times faster than the single-core implementation.



(a)



(b)



(c)

Figure 3-4: The performance of Algorithm 2 for linear programming in comparison with (a) OSQP, (b) GUROBI, and (c) MOSEK.

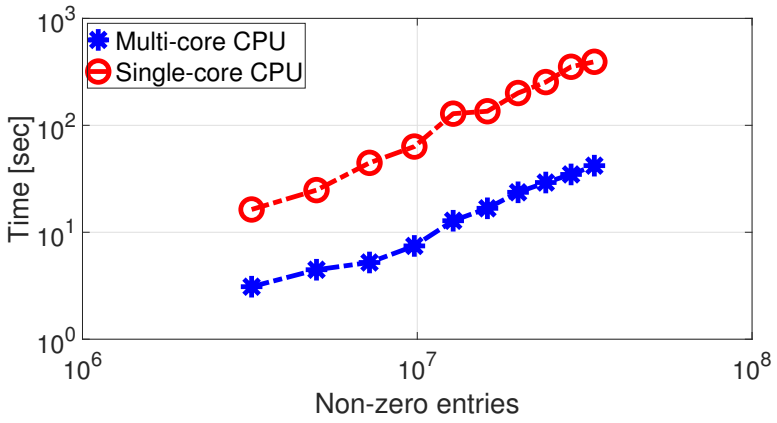


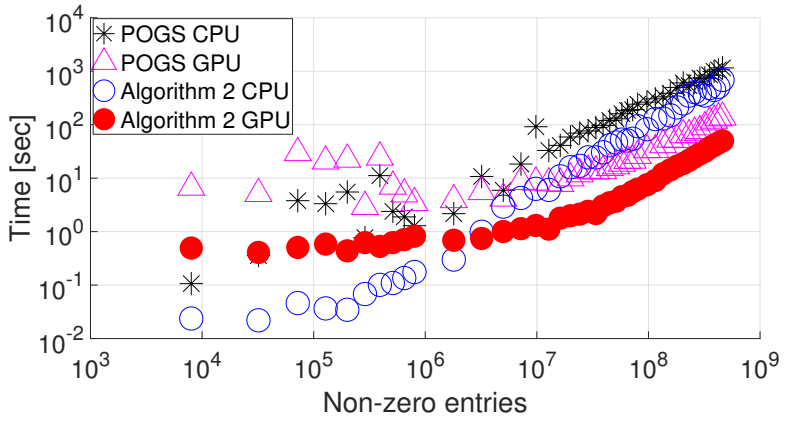
Figure 3-5: Single-core vs Multi-core CPU implementation

3.5.3 Comparisons with POGS

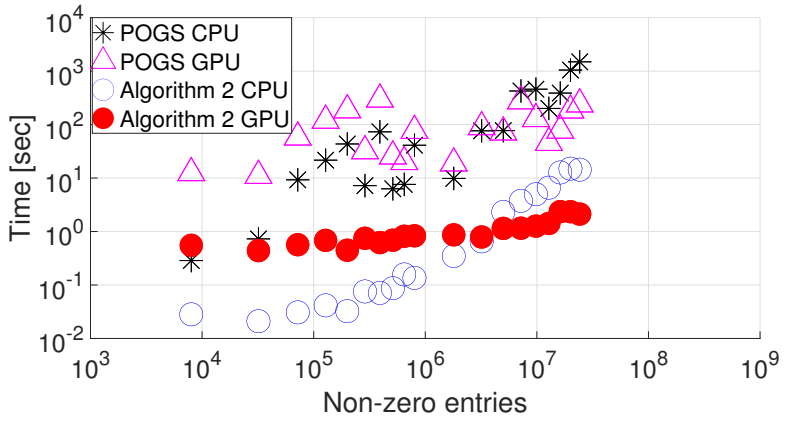
In this case study, we seek to demonstrate the ability of Algorithm 2 in finding very accurate solutions unlike competing first-order solvers that struggle with accuracy. In Figure 3-6, we perform comparisons between Algorithm 2 and the first-order solver POGS [15]. We use the default settings for POGS except for the absolute and relative tolerance values ε^{abs} and ε^{rel} for stopping criteria. Figure 3-6 demonstrates that Algorithm 2 comprehensively outperforms one of the prominent first order solver, particularly with lower tolerance values. Similar to the previous experiment, the stopping criteria of Algorithm (2) depends on the competing solver, as define in (3.11).

3.5.4 Second-Order Cone Programming

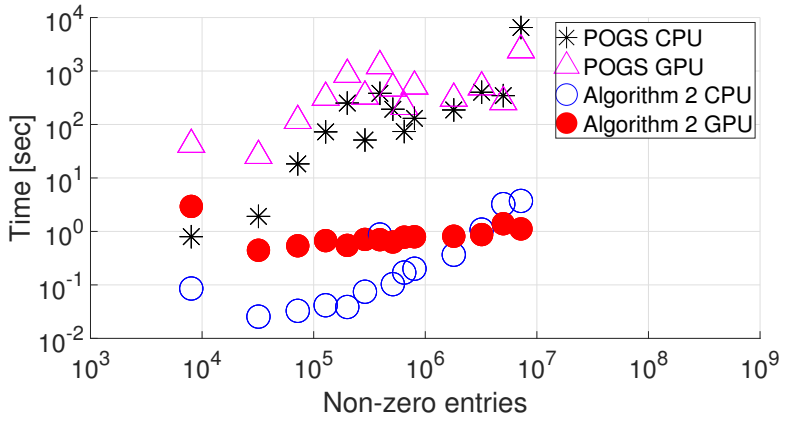
This case study is concerned with the class of second-order cone programming optimization problems. The performance of Algorithm 2 is tested in comparison with MOSEK on default settings. We have generated random SOCP instances with n ranging from 100 to 2900 (MOSEK takes the maximum time of 1200 seconds), and $m = \lfloor 0.8n \rfloor$:



(a)



(b)



(c)

Figure 3-6: The performance of Algorithm 2 for linear programming in comparison with POGS with the absolute and relative tolerances equal to (a) $\varepsilon^{\text{abs}} = 10^{-5}$, $\varepsilon^{\text{rel}} = 10^{-4}$, (b) $\varepsilon^{\text{abs}} = 10^{-6}$, $\varepsilon^{\text{rel}} = 10^{-5}$, and (c) $\varepsilon^{\text{abs}} = 10^{-7}$, $\varepsilon^{\text{rel}} = 10^{-6}$.

- The elements of $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ have i.i.d uniform distribution from the interval $[-1, 1]$.
- $\boldsymbol{b} := \boldsymbol{A} \times \text{abs}_{\mathcal{K}}(\dot{\boldsymbol{x}})$ where the elements of $\dot{\boldsymbol{x}} \in \mathbb{R}^n$ have i.i.d uniform distribution from the interval $[0, 1]$.
- The elements of $\boldsymbol{c} \in \mathbb{R}^n$ have i.i.d uniform distribution from the interval $[0, 1]$.
- And $\mathcal{K} = (\mathcal{K}_h)^{\frac{n}{h}}$, where $\mathcal{K}_h$ is the standard Lorentz cone of size h.

We start applying adaptive conditioning at iterations 200 and apply it once again in every 100 steps, i.e.,

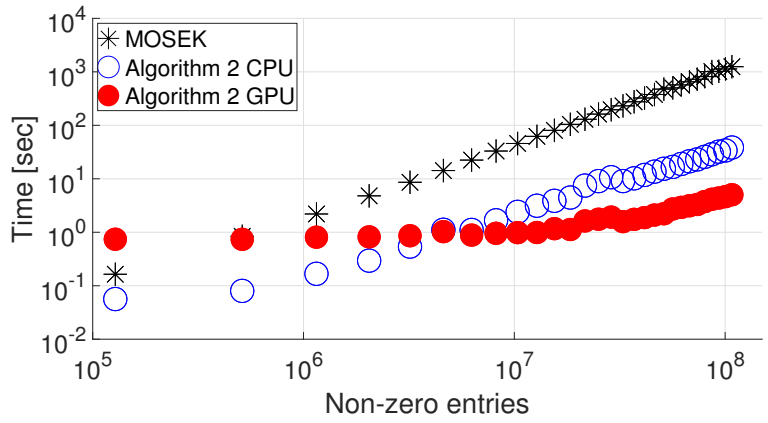
$$\mathcal{L} = \{200, 300, 400, \ldots\}. \tag{3.13}$$

Parameters t and μ are set to 1.7 and 1, respectively.

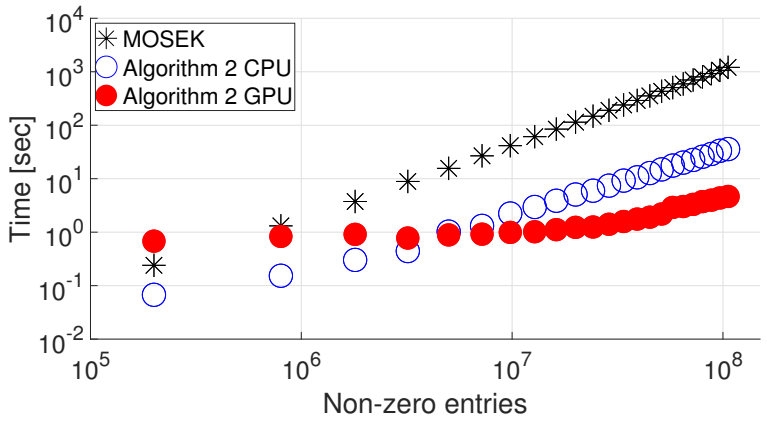
The comparison of computational time for Lorentz cones of size $h = 4$ and $h = 10$ are reported in Figure 3-7. It is clear from Figure 3-7 that Algorithm (2) outperforms MOSEK by a large margins as the size of problem grows, while MOSEK performs better for smaller size problems.

3.6 Conclusions

We proposed a proximal numerical method with potential for parallelization. Next, an adaptive conditioning heuristic was developed to speed up the convergence of the proposed method. We provided a numerical example to demonstrate the fact that existing acceleration, parameter tuning and preconditioning methods have very limited effect on convergence behavior of first order methods. Moreover, we showed that convergence rate can be improved, irrespective of the condition number of data matrices. The proposed algorithm is implemented on graphics processing unit with an order-of-magnitude time improvement. A wide range of numerical experiments



(a)



(b)

Figure 3-7: The performance of Algorithm 2 for second order cone programming in comparison with MOSEK with Lorentz cones of size (a) $h = 4$ and (b) $h = 10$

are conducted on large problems and results are compared with prominent first order solvers as well as the interior point method based solvers to demonstrate the claims made in this paper. We solved a variety of linear programs and second-order cone programs. It is evident from experimental results that the proposed algorithm outperforms the first order algorithms in terms of computational time and achieves the accuracy levels comparable to second-order state-of-the-art methods.

Algorithm 2

Input: $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, \mathcal{K})$, fixed $\mu > 0$, initial points $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{z} \in (\mathbb{R} \setminus \{0\})^n$, fixed $0 < t < 1$, and $\mathcal{L} \subseteq \mathbb{N}$

$l \leftarrow 0$
repeat
  $l \leftarrow l + 1$
  if $l \in \mathcal{L} \cup \{1\}$ **then**
    for $i = 1, \ldots, k$ **do**
      $h \leftarrow n_1 + \ldots + n_{i-1}$
      for $j = h + 1, \ldots, h + n_i$ **do**
        $\boldsymbol{o}_j \leftarrow \left| x_{h+1} - \|[x_{h+2}, \ldots, x_{h+n_i}]\|_2 \right| / |z_{h+1}|$
      end for
    end for
    $\boldsymbol{O} \leftarrow \operatorname{diag}\{|\boldsymbol{o}|\}^{\min\left\{1, \frac{t}{\log(\max\{\boldsymbol{o}\}) - \log(\min\{\boldsymbol{o}\})}\right\}}$
    $\hat{\mathcal{A}} \leftarrow \operatorname{range}\{(\boldsymbol{AO})^\top\}$
    $\boldsymbol{d} \leftarrow (\boldsymbol{AO})^\dagger \boldsymbol{b} + \dfrac{\mu}{2}\left(\operatorname{abs}_{\boldsymbol{O}^\top \mathcal{A}}(\boldsymbol{O}^\top \boldsymbol{c}) - \boldsymbol{O}^\top \boldsymbol{c}\right)$
    $\boldsymbol{s} \leftarrow \boldsymbol{O}^{-1}\boldsymbol{x} - \mu \boldsymbol{O}\boldsymbol{z}$
  end if
  $\boldsymbol{p} \leftarrow \operatorname{abs}_{\boldsymbol{O}^{-1}\mathcal{K}}(\boldsymbol{s})$
  $\boldsymbol{r} \leftarrow \operatorname{abs}_{\hat{\mathcal{A}}}(\boldsymbol{p})$
  $\boldsymbol{s} \leftarrow \dfrac{\boldsymbol{s}}{2} - \dfrac{\boldsymbol{r}}{2} + \boldsymbol{d}$
  $\boldsymbol{x} \leftarrow \dfrac{\boldsymbol{O}(\boldsymbol{p} + \boldsymbol{s})}{2}$
  $\boldsymbol{z} \leftarrow \dfrac{\boldsymbol{O}^{-1}(\boldsymbol{p} - \boldsymbol{s})}{2\mu}$
until stopping criteria is met.

Output: $\boldsymbol{x}$ and $\boldsymbol{z}$

CHAPTER 4

A First-Order Numerical Algorithm without Matrix Operations

4.1 Introduction

Conic formulation can be visualized as a universal formulation of convex programs as any convex optimization problem can be formulated in terms of conic optimization. In fact several convex optimization solvers and tools such as MOSEK [1], SDPT3 [2], CVX [3], GUROBI [4] and CPLEX [5] first transform the problems into a conic optimization problem. Several numerical algorithms have been developed in literature to solve conic optimization problems, the most commonly being interior point methods (IPMs) [6–8, 46–48]. These methods are known for reliability and efficiency to solve small to medium-sized problems. Interior point methods provide high accuracy solution in a few tens of iterations irrespective of problem condition number and structure. However, at each iteration IPMs rely on Newton method and are require to build and solve the linear systems corresponding to KKT matrix. Some of the common available algorithm for handling the KKT system in each iteration are Gauss-Jordan [49], Gaussian elimination [50], LU decomposition [51], Cholesky decomposition [52], QR decomposition and Monte Carlo Methods [53] for inverse. The computation of direct approaches are prohibitively expensive for large scale problems. Computation of KKT system in IPMs is the major bottleneck and impede their ability to target larger problems. Interior point methods are not used in practice for large scale problem because the computational cost and memory requirements are high in each iteration.

4.2 Related Work

Motivated by the requirement of solving linear system in each steps and computationally expensive direct methods, researchers explored matrix-free interior-point methods. Indirect or iterative numerical algorithms for solving KKT matrix have received a particular attention in recent past [54, 55]. Direct methods for solving linear systems of equations becomes impractical for large scale problems [54, 56]. Alternatively, iterative methods do not require to store Schur complement matrix [56], thus Krylove subspace based methods such as preconditioned conjugate gradient (PCG) becomes efficient. The conjugate gradient method was first introduced in [57] where an iterative technique was applied to solve the Newton step instead of factorizing Hessian matrix directly. Authors in [58] proposed a Lagrangian dual predictor-corrector algorithm and applied conjugate gradient method for handling linear system. In [59], an iterative algorithm is applied to modified barrier method for handling large-scale semidefinite optimization problems. The conjugate gradient method with a basic preconditioner is applied [60] to save the computation time for solving semidefinite programs. Inexact semi-smooth Newton conjugate gradient approach is adopted in [61] to improve the accuracy of handling semidefinite problems.

Unlike the direct methods, one of the notable disadvantage of iterative methods is to render the inexact or approximate solution of KKT system. The performance of iterative numerical algorithms is highly influenced by the spectral properties of linear system and depends upon the condition number of matrix involved [62]. The preconditioners in iterative methods are designed in way that the condition number should be as small as possible [63]. In order to ensure the accuracy of IPMs, a number of devoted attempts have been made to study the conditions in which inexact directions satisfy the high accuracy [55, 64]. A matrix-free numerical algorithm for equality

constraint nonlinear programming is proposed in [65] to remedy the ill-conditioning problem that may render from the rank-deficient Jacobian matrix. A matrix-free solver PDCO [66] uses LSMR to solver linear system. In [67] an attempt has been made to design a matrix-free IPM to work for quadratic programs, where the KKT system is regularized to bound the condition number and then the preconditioner is designed for the regularized system. This approach substantially decreased the computational cost and memory storage in each iteration in comparison to traditional IPMs at the trade of more iterations and lower accuracy. Despite these serious efforts and introduction of indirect methods, IPMs are inherently computationally expensive and thus not suitable for large-scale conic programs.

Alternatively, first order methods scales gracefully for handling huge cone programs at the cost of moderate accuracy [15–18]. The computational cost of each iteration is significantly lower than of IPMs and thus provide a reasonable alternative to deal with large problems. These methods are known to provide reasonable accuracy solutions in a few hundred of iterations because of computationally inexpensive iterations and are well suited for the applications such as large scale optimization, where high accuracy solutions are typically not required [13, 14]. The fast convergence of first-order method to achieve a reasonable accuracy in a limited number of iterations is an active research area and several dedicated attempts have been made to accelerate the convergence in past few years [11, 12, 25–27, 29, 32, 33, 44, 68].

Despite of the fact that first-order methods offers inexpensive iterative steps, low memory requirement and easy implementation, these methods are also required to solve KKT system in each iteration [18]. Thanks to *factorizing caching*, in practice, the linear system is solved in first iterations, factors are stored and then reused in subsequent iteration [13,15,18]. The most common factorizing caching approaches are LDL and QR decomposition [69]. For large-scale conic programs the direct methods

or factorizing caching becomes impractical, at which point matrix-free or indirect approaches become viable [15]. The first-order methods based solvers [15, 16, 18, 69], apply conjugate gradient method to solve the large scale problems.

First-order methods struggles to find high accuracy solutions and matrix-free indirect conjugate gradient approaches further hamper their ability to render high quality solutions. The computational complexity of direct methods and low accuracy solution of indirect methods to tackle large conic problems motivated us to propose a general purpose matrix-free approach to solve very large problems with a modest accuracy.

4.2.1 Contributions

In this work, we pursue a matrix-free first order numerical algorithm to deal with very large-scale sparse conic optimization problems. The basic idea is to decompose the constraint matrix into sparse factors in such a way that iterative steps are division free and do not involve any matrix operations. The sparse factors are computed only in first iteration, and reused in subsequent iterations. In comparison to existing matrix decomposition techniques such as LDL and QR decomposition, our approach is simple and computationally efficient to compute the factors. The notable property of the proposed decomposition is that computing the factors and steps of iterative loop are division free. The factors are easy-to-compute and require minimally possible memory storage. The matrix inversion lemma is utilized to make the operations division free. We reformulate the standard conic optimization problem by introducing auxiliary variables and then apply the proposed matrix-free algorithm in conjunction with well-known two-block ADMM [14, 19, 20]. The computational burden of solving the linear system in each iteration is being taken out of the iterative loop and factors are stored first, without any expensive factorization.

The algorithm offers the benefits of matrix-free, division-free and inexpensive iterations. The proposed algorithm admitting parallel and large-scale implementation, and amenable to graphics processing unit (GPU) implementation. We demonstrate the performance gain and computational speedup of algorithm by conducting a wide range of experiments and compare the results with other first-order solvers.

Rest of the chapter is organized as follows. We introduce the cone programming and a brief description of operator splitting numerical algorithms for solving such problems is presented in section 4.3. We analyze the existing direct and indirect methods for solving the linear system and provide a motivation for our numerical method in section 4.4. Section 4.5 presents the proposed algorithm. Numerical experiments and comparison with existing competing solvers are presented in 4.6. The conclusion is provided in section 4.7.

4.2.2 Notations

Notations $\mathbb{R}$ and $\mathbb{N}$ denote the set of real and natural numbers, respectively. Matrices and vectors are represented by bold uppercase, and bold lowercase letters, respectively. Notation $\|\cdot\|_2$ refers to ℓ_2 norm of either matrix or vector depending on the context and $|\cdot|$ represents the absolute value. The symbol $(\cdot)^\top$ represent the transpose operators. The notations $\boldsymbol{I}_n$ refer to the $n \times n$ identity matrix. The symbol $\mathcal{K}$ is used to describe different types of cones used in this paper. The superscript $(\cdot)^{\text{opt}}$ refers to the optimal solution of optimization problem. The notation o denotes the number of nonzero entries in matrix. The symbol $\mathcal{L}$ represent the augmented Lagrangian function. The notations "$\mathcal{P}_1$", "$\mathcal{P}_2$" and "$\mathcal{D}$" represent the primal and dual blocks of two-block ADMM. The symbols $\varepsilon_{\text{prim}}, \varepsilon_{\text{dual}}, \varepsilon_{\text{abs}}$ and ε_{rel} are used for primal, dual, absolute and relative tolerance, respectively, and notation ε_{gap} is the tolerance for difference between primal and dual objective values.

4.3 Preliminaries

We consider convex optimization programs with a linear objective function subject to a number of conic constraints in the form:

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}} \quad & \boldsymbol{c}^\top \boldsymbol{x} && \text{(4.1a)} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && \text{(4.1b)} \\ & \boldsymbol{x} \in \mathcal{K} && \text{(4.1c)} \end{aligned}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the primal decision variable of conic program and objective function is defined by vector $\boldsymbol{c} \in \mathbb{R}^n$. The conic constraints of primal problem are defined by sparse matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, vector $\boldsymbol{b} \in \mathbb{R}^m$ and a closed, convex and non-empty cone $\mathcal{K} \triangleq \mathcal{K}_{n_1} \times \mathcal{K}_{n_2} \times \cdots \times \mathcal{K}_{n_k} \subseteq \mathbb{R}^n$, where each $\mathcal{K}_{n_i} \subseteq \mathbb{R}^{n_i}$ is a Lorentz cone of size n_i, i.e.,

$$\mathcal{K}_{n_i} \triangleq \left\{ \boldsymbol{w} \in \mathbb{R}^{n_i} \,|\, w_1 \geq \left\| [w_2, \ldots, w_{n_i}] \right\|_2 \right\},$$

and $n_1 + n_2 + \ldots + n_k = n$.

In order to solve 4.1, various first order operator splitting methods have been implemented in past decade. First order methods are particularly interesting for the applications where iterative steps can be solved efficiently through closed form formula and large number of iterations can be executed in a short amount of time. The most common methods are Alternating Direction Method of Multipliers (ADMM) and Douglas-Rachford Splitting (DRS). To this end, we briefly introduce DRS and ADMM methods to solve problem 4.1.

4.3.1 Douglas-Rachford Splitting

DR splitting was fist introduced in [70] to find numerical solutions of differential equations for heat conduction problems, and has been used to solve separable convex optimization problems. Rather than operating on the whole problem directly, DRS works on a splitting scheme to address each component of problem separately. In order to implement the DR splitting method, the optimization problem (4.1) is casted in the form of

$$\text{minimize} \qquad f(\boldsymbol{x}) + g(\boldsymbol{x}) \tag{4.2}$$

where $f, g : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ are indicator functions and defined as

$$f(\boldsymbol{x}) \triangleq \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathcal{K} \\ \infty & \text{otherwise} \end{cases} \quad \text{and} \quad g(\boldsymbol{x}) \triangleq \begin{cases} \boldsymbol{c}^\top \boldsymbol{x} & \text{if } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ \infty & \text{otherwise} \end{cases}$$

leading to the following steps:

$$\boldsymbol{x} \leftarrow \text{prox}_f(\boldsymbol{z}) \tag{4.3a}$$

$$\boldsymbol{z} \leftarrow \boldsymbol{z} + \text{prox}_g(2\boldsymbol{x} - \boldsymbol{z}) - \boldsymbol{x}. \tag{4.3b}$$

Each iteration of DRS algorithm requires proximal operator evaluation and projection onto a linear subspace.

4.3.2 Alternating Direction Method of Multipliers

The alternating direction method of multipliers (ADMM) is one of the most commonly used first-order algorithm in the literature for handling large-scale opti-

mization problems. ADMM is efficient for large-scale problems owing to its simple implementation, computational cheapness, and distributive nature. This method can be analyzed as a special case of DRS as the former applied to primal is equivalent to applying DRS to the dual problem. A standard way of solving problem (4.1) via ADMM is through the formulation

$$\underset{\boldsymbol{x}_1,\boldsymbol{x}_2\in\mathbb{R}^n}{\text{minimize}} \qquad f(\boldsymbol{x}_1)+g(\boldsymbol{x}_2) \tag{4.4a}$$

$$\text{subject to} \qquad \boldsymbol{x}_1=\boldsymbol{x}_2 \tag{4.4b}$$

leading to the steps

$$\boldsymbol{x}_1 \leftarrow \text{prox}_{\mu^{-1}f}(\boldsymbol{x}_2-\mu^{-1}\boldsymbol{\lambda_1}) \tag{4.5a}$$

$$\boldsymbol{x}_2 \leftarrow \text{prox}_{\mu^{-1}g}(\boldsymbol{x}_1+\mu^{-1}\boldsymbol{\lambda_1}) \tag{4.5b}$$

$$\boldsymbol{\lambda_1} \leftarrow \boldsymbol{\lambda_1}+\mu(\boldsymbol{x}_1-\boldsymbol{x}_2). \tag{4.5c}$$

where $\boldsymbol{x}_1, \boldsymbol{x}_2$ are primal variables, $\boldsymbol{\lambda_1}$ is dual variable and μ is a fixed tuning parameter.

4.4 Solving Linear System

Although the first order numerical algorithms are considered highly efficient for solving large optimization problems with modest accuracy, but evaluating the projection operator in each step is probably one of the most computationally expensive part in these methods. One need to project the primal and dual variables onto the linear space and solve the linear system at each iterations. Depending on the problem dimension and structure of the constraint matrix $\boldsymbol{A}$, there are different ways to

find solution of the linear system. The common methods for solving linear system in iterative algorithms are classified as direct and indirect methods [15, 18].

4.4.1 Direct Methods

The solution of linear system can be computed by first factorizing the KKT matrix of optimization problem, and then performing forward or backward substitutions. Since the KKT matrix doesn't change throughout the iterations, the common practice is to compute the factors at the beginning in first iteration and then reuse in subsequent iterations. The process of computing the factors at first iteration and then reusing in rest of the iterations is known as *factorization caching*. The technique of factorization caching is computationally efficient and is frequently used by first order solvers [15, 16, 18, 69].

When $\boldsymbol{A}$ is dense, the common practice to solve the KKT matrix by performing Cholesky decomposition or QR decomposition at first iteration and then cache the factor for subsequent iterations. The complexity in both approaches is $O(mn^2)$ in first iterate and $O(mn)$ in subsequent iterations.

When the problem size of sparse matrix is not too big, the direct sparse $\boldsymbol{LDL}^\top$ decomposition is well defined for quasi-definite KKT matrix, i.e. it can be defined as a 2x2 block symmetric matrix where diagonal blocks are positive and negative definite, respectively. [71]. In such scenarios, direct method exactly solves the system by using $\boldsymbol{LDL}^\top$ factorization, with $\boldsymbol{L}$ being the lower triangular matrix and $\boldsymbol{D}$ is a diagonal matrix. By storing $\boldsymbol{D}^{-1}$ in first step, the solution of KKT matrix can be made division free. The computational complexity for sparse matrices is $O(o_{\boldsymbol{L}})$ and depends upon the nonzero entries of factor $\boldsymbol{L}$.

4.4.2 Indirect Methods

For very large problems factoring the linear system can be computationally expensive and direct methods becomes impractical. In these cases, indirect methods are preferred to approximately find the solution of linear system. Instead of solving the linear system by factorization, *indirect methods* applies an iterative scheme such as conjugate gradient CG [57, 72, 73] or LSQR [74, 75].

The indirect methods are required to evaluate the approximate projections in each step, but in practice, this method takes very few iterations to produce an adequate approximation. A more practical approach is to terminate the method when the system of equations is solved to a reasonable accuracy. Another advantage of indirect method is that the conjugate gradient technique can be warm-started by using the solution obtained in previous iteration as an initial point in each subsequent step. This heuristic approach results inaccurate projections at the beginning but start producing more accurate ones as the first order algorithms begins to converge. In contrast to direct methods, indirect methods offers low complexity and more freedom for adaptive parameter selection since there is no factorization. In each step of conjugate gradient, a vector is required to be multiplied by $\boldsymbol{A}$ and $\boldsymbol{A}^\top$, thus the computational cost of each iteration is $O(o_{\boldsymbol{A}})$.

4.5 Problem Formulation

Notice that problem 4.1 can be computationally expensive for large scale problems due to the requirement of solving linear system or factorization caching. In contrast to existing direct and indirect methods for solving linear system in literature, we propose a highly efficient matrix-free method to handle large scale sparse optimization problems. The basic idea is to decompose $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{V}$, such that $\boldsymbol{U} \in \mathbb{R}^{m \times o}$ has

exactly one non-zero value in each column and $\boldsymbol{V} \in \mathbb{R}^{o \times n}$ has exactly one non-zero entry in each row. Such decomposition can be readily constructed as follows.

Let $\{\boldsymbol{A}_{i_k,j_k}\}_{k=1}^{o}$ represent the non-zero elements of $\boldsymbol{A}$ in an arbitrary order.

$$\boldsymbol{U} \triangleq \sum_{k=1}^{o} \boldsymbol{A}_{i_k,j_k} \boldsymbol{e}_{i_k} \boldsymbol{f}_k^{\top} \tag{4.6a}$$

$$\boldsymbol{V} \triangleq \sum_{k=1}^{o} \boldsymbol{f}_k \boldsymbol{g}_{j_k}^{\top} \tag{4.6b}$$

where $\boldsymbol{e}_k{}_{k=1}^{m}$, $\boldsymbol{f}_k{}_{k=1}^{o}$, and $\boldsymbol{g}_k{}_{k=1}^{n}$ represent the standard basis for $\mathbb{R}^m$, $\mathbb{R}^o$, and $\mathbb{R}^n$, respectively. The factor $\boldsymbol{U}$ contains the nonzero elements of matrix $\boldsymbol{A}$ whereas $\boldsymbol{V}$ is the associated sparse permutation matrix. These factors are computed in such a way that the matrix $\boldsymbol{U}$ contains exactly one nonzero element in each column, whereas the matrix $\boldsymbol{V}$ contains exactly one nonzero entry in each row. We illustrate the idea of decomposition by providing a simple example as follows:

Example 1: We illustrate the decomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{V}$, where $\boldsymbol{A}$ is a sparse matrix with o nonzero entries, the corresponding $\boldsymbol{U}$ and $\boldsymbol{V}$ are

$$\boldsymbol{A} := \begin{bmatrix} 1 & 0 & 4 & 6 & 8 \\ 0 & 0 & 5 & 0 & 0 \\ 2 & 3 & 0 & 7 & 0 \end{bmatrix},$$

$$\boldsymbol{U} := \begin{bmatrix} 1 & 0 & 0 & 4 & 0 & 6 & 0 & 8 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 2 & 3 & 0 & 0 & 0 & 7 & 0 \end{bmatrix},$$

$$\boldsymbol{V} := \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^\top$$

To this end, we reformulate the problem 4.1 and apply matrix free decomposition

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}} \quad & \boldsymbol{c}^\top \boldsymbol{x} && \text{(4.8a)} \\ \text{subject to} \quad & \boldsymbol{U}\boldsymbol{y} = \boldsymbol{b} && \text{(4.8b)} \\ & \boldsymbol{y} = \boldsymbol{V}\boldsymbol{x} && \text{(4.8c)} \\ & \boldsymbol{z} = \boldsymbol{x} && \text{(4.8d)} \\ & \boldsymbol{z} \in \mathcal{K} && \text{(4.8e)} \end{aligned}$$

where $\boldsymbol{y} \in \mathbb{R}^n$ and $\boldsymbol{z} \in \mathbb{R}^n$ are auxiliary variables. The equality constraint 4.1b is reformulated by 4.8b and 4.8c to avoid the matrix inversion to solve linear sys-

tem, whereas 4.8d enable the formulation to be solved by ADMM. The augmented Lagrangian function for problem 4.8 is given as

$$\begin{aligned}\mathcal{L}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},\boldsymbol{\lambda},\boldsymbol{\gamma},\boldsymbol{\delta}) \triangleq \boldsymbol{c}^\top\boldsymbol{x} + \boldsymbol{\lambda}^\top(\boldsymbol{U}\boldsymbol{y}-b) + \frac{\mu}{2}\big\|\boldsymbol{U}\boldsymbol{y}-\boldsymbol{b}\big\|_2^2 \\ + \boldsymbol{\gamma}^\top(\boldsymbol{y}-\boldsymbol{V}\boldsymbol{x}) + \frac{\mu}{2}\big\|\boldsymbol{y}-\boldsymbol{V}\boldsymbol{x}\big\|_2^2 \\ + \boldsymbol{\delta}^\top(\boldsymbol{z}-\boldsymbol{x}) + \frac{\mu}{2}\big\|\boldsymbol{z}-\boldsymbol{x}\big\|_2^2\end{aligned} \tag{4.9}$$

where $\mu \in \mathbb{R}$ is a fixed parameter and $\boldsymbol{\lambda} \in \mathbb{R}^m, \boldsymbol{\gamma} \in \mathbb{R}^o$, and $\boldsymbol{\delta} \in \mathbb{R}^n$ are the Lagrange multipliers associated with constraint 4.8b,4.8c and 4.8d, respectively.

We perform two block ADMM and regroup the primal and dual variables as

$$\begin{aligned}&\text{(Block 1)} \qquad \mathcal{P}_1 = \{\boldsymbol{x}\} \\ &\text{(Block 2)} \qquad \mathcal{P}_2 = \{\boldsymbol{y},\boldsymbol{z}\} \\ &\text{(Dual)} \qquad \mathcal{D} = \{\boldsymbol{\lambda},\boldsymbol{\gamma},\boldsymbol{\delta}\}\end{aligned}$$

where "$\mathcal{P}_1$", "$\mathcal{P}_2$" and "$\mathcal{D}$" corresponds to $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and λ in standard formulation of ADMM. The iterations of Two block ADMM for problem 4.8 can be given as follows:

1. *Minimization in terms of $\boldsymbol{x}$:* This step consist of freezing the other variables at their previous values and minimizing the Lagrangian function with respect to $\boldsymbol{x}$, i.e.,

$$\boldsymbol{x}^{k+1} = \underset{\boldsymbol{x}\in\mathbb{R}^n}{\arg\min} \quad \mathcal{L}(\boldsymbol{x},\boldsymbol{y}^k,\boldsymbol{z}^k,\boldsymbol{\lambda}^k,\boldsymbol{\gamma}^k,\boldsymbol{\delta}^k)$$

The variable $\boldsymbol{x}$ update results in following closed form solution

$$\boldsymbol{x}^{k+1} = (\boldsymbol{I} + \boldsymbol{V}^\top \boldsymbol{V})^{-1}\Big[\boldsymbol{V}^\top\big(\boldsymbol{y}^k + \frac{\boldsymbol{\gamma}^k}{\mu}\big) + \boldsymbol{z}^k + \frac{\boldsymbol{\delta}^k}{2\mu} - \frac{\boldsymbol{c}}{\mu}\Big] \tag{4.10}$$

Observe that $\boldsymbol{V}^\top\boldsymbol{V}$ is diagonal and $(\boldsymbol{I}+\boldsymbol{V}^\top\boldsymbol{V})$ is a diagonal matrix with nonzero diagonal entries. Notice that once the factor $\boldsymbol{V}$ is known, calculating the solution of 4.10 can be made division-free by storing $(\boldsymbol{I} + \boldsymbol{V}^\top\boldsymbol{V})^{-1}$.

2. The subproblem 4.5b in terms of $\mathcal{P}_2$ consists of two parallel steps:

   (a) *Minimization in terms of* $\boldsymbol{y}$: Minimizing the Lagrangian function with respect to variable $\boldsymbol{y}$ and freezing the other variables at their latest values yields the following closed form formula for updating $\boldsymbol{y}$

$$\boldsymbol{y}^{k+1} = (\boldsymbol{I} + \boldsymbol{U}^\top\boldsymbol{U})^{-1}\Big[\boldsymbol{U}^\top\big(\boldsymbol{b} - \frac{\boldsymbol{\lambda}^k}{\mu}\big) + \boldsymbol{V}\boldsymbol{x}^{k+1} - \frac{\boldsymbol{\gamma}^k}{\mu}\Big] \tag{4.11}$$

   Notice that $\boldsymbol{U}\boldsymbol{U}^\top$ is diagonal and this step can be made matrix free by applying Lemma 1.

   (b) *Minimization in terms of* $\boldsymbol{z}$: This steps consists of projecting the variable onto associated Lorentz cones by using 4.16.

$$\boldsymbol{z}^{k+1} = \mathrm{proj}_{\mathcal{K}}\Big\{\boldsymbol{x}^{k+1} - \frac{\boldsymbol{\delta}^k}{\mu}\Big\} \tag{4.12}$$

   We project $\boldsymbol{w} = \boldsymbol{x}^{k+1} - \frac{\boldsymbol{\delta}^k}{\mu}$ onto cone $\mathcal{K}$ in each iteration. The projection onto each $\mathcal{K}_i$ can be done in parallel and with little computational cost. Projection onto Lorentz cone has the closed form solution is provided in Definition 2.

Algorithm 3

Input: $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, \mathcal{K})$, fixed $\mu > 0$, and initial points $\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\delta} \in \mathbb{R}^n, \boldsymbol{y}, \boldsymbol{\gamma} \in \mathbb{R}^o$, and $\boldsymbol{\lambda} \in \mathbb{R}^m$

$\boldsymbol{U} := \sum_{k=1}^{o} A_{i_k, j_k} \boldsymbol{e}_{i_k} \boldsymbol{f}_k^\top$

$\boldsymbol{V} := \sum_{k=1}^{o} \boldsymbol{f}_k \boldsymbol{g}_{j_k}^\top$

$\boldsymbol{F}_{\boldsymbol{U}} := \boldsymbol{I} - \boldsymbol{U}^\top \left(\boldsymbol{I} + \boldsymbol{U}\boldsymbol{U}^\top\right)^{-1} \boldsymbol{U}$

$\boldsymbol{F}_{\boldsymbol{V}} := \left(\boldsymbol{I} + \boldsymbol{V}^\top \boldsymbol{V}\right)^{-1}$

repeat

  $\boldsymbol{x} \leftarrow \boldsymbol{F}_{\boldsymbol{V}} \left(\boldsymbol{V}^\top \left(\boldsymbol{y} + \frac{\boldsymbol{\gamma}}{\mu}\right) + \boldsymbol{z} + \frac{\boldsymbol{\delta}}{\mu} - \frac{\boldsymbol{c}}{\mu} \right)$

  $\boldsymbol{y} \leftarrow \boldsymbol{F}_{\boldsymbol{U}} \left(\boldsymbol{U}^\top \left(\boldsymbol{b} - \frac{\boldsymbol{\lambda}}{\mu}\right) + \boldsymbol{V}\boldsymbol{x} - \frac{\boldsymbol{\gamma}}{\mu} \right)$

  $\boldsymbol{z} \leftarrow \mathrm{proj}_{\mathcal{K}} \left\{ \boldsymbol{x} - \frac{\boldsymbol{\delta}}{\mu} \right\}$

  $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \mu(\boldsymbol{U}\boldsymbol{y} - \boldsymbol{b})$

  $\boldsymbol{\gamma} \leftarrow \boldsymbol{\gamma} + \mu(\boldsymbol{y} - \boldsymbol{V}\boldsymbol{x})$

  $\boldsymbol{\delta} \leftarrow \boldsymbol{\delta} + \mu(\boldsymbol{z} - \boldsymbol{x})$

until stopping criteria is met.

Output: $\boldsymbol{x}^{\mathrm{opt}}, \boldsymbol{\lambda}^{\mathrm{opt}}$

3. *Dual variables update:* This steps involves the dual variable update given as follows:

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \mu(\boldsymbol{U}\boldsymbol{y}^{k+1} - \boldsymbol{b}) \tag{4.13}$$

$$\boldsymbol{\gamma}^{k+1} = \boldsymbol{\gamma}^k + \mu(\boldsymbol{y}^{k+1} - \boldsymbol{V}\boldsymbol{x}^{k+1}) \tag{4.14}$$

$$\boldsymbol{\delta}^{k+1} = \boldsymbol{\delta}^k + \mu(\boldsymbol{z}^{k+1} - \boldsymbol{x}^{k+1}) \tag{4.15}$$

In contrary to existing practices that focuses on either using standard matrix decompositions or apply an approximate solution by conjugate gradient method, we propose a matrix free algorithm which does not rely on standard matrix factorization. Instead, we store the nonzero entries in sparse matrices in such a way that we do not need to apply the standard matrix decomposition. The standard fast ADMM

iterations can be used to the reformulated problem. These steps are highlighted in Algorithm 3

- **Step 1 and 2:** The equality constraint matrix $\boldsymbol{A}$ is decomposed into $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{V}$ in such a manner that $\boldsymbol{U}$ has exactly one non-zero entry in each column while $\boldsymbol{V}$ has exactly one non-zero entry in each row.
- **Step 3 and 4:** In order to make iterative steps division free, we compute the multiplication factors in these steps. Note that these factors are easy-to-compute since $(\boldsymbol{I} + \boldsymbol{U}\boldsymbol{U}^\top)$ and $(\boldsymbol{I} + \boldsymbol{V}^\top\boldsymbol{V})$ are diagonal matrices with non-zero diagonal entries.
- **Step 6 to 11:** These steps are concerned with ADMM primal and dual variables update, whereas the projection onto specific cone is performed in Step 8.

Definition 2. *For any proper cone $\mathcal{K}_i \in \mathbb{R}^{n_i}$, define the projection operator* $\text{proj}_{\mathcal{K}_i} : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ *as*

$$\text{proj}_{\mathcal{K}_i}(\boldsymbol{w}) \triangleq \begin{cases} \boldsymbol{0}^{n_i} & \left\|[w_2, \ldots, w_{n_i}]\right\|_2 \leq -w_1 \\ \boldsymbol{w} & \left\|[w_2, \ldots, w_{n_i}]\right\|_2 \leq w_1 \\ \left[\alpha, \dfrac{\alpha[w_2, \ldots, w_{n_i}]}{\|[w_2, \ldots, w_{n_i}]\|_2}\right] & \text{otherwise} \end{cases} \tag{4.16}$$

where $\alpha = (w_1 + \|[w_2, \ldots, w_{n_i}]\|_2)/2$.

Lemma 4. *Let $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{U}$ be matrix of size $m \times o$, then the following identity holds:*

$$(\boldsymbol{I} + \boldsymbol{U}^\top\boldsymbol{U})^{-1} = \boldsymbol{I} - \boldsymbol{U}^\top(\boldsymbol{I} + \boldsymbol{U}\boldsymbol{U}^\top)^{-1}\boldsymbol{U} \tag{4.17}$$

where $\boldsymbol{U}\boldsymbol{U}^\top$ is diagonal and $\boldsymbol{I} + \boldsymbol{U}\boldsymbol{U}^\top$ is a diagonal matrix with nonzero elements.

Proof. Please refer to [76] for proof. □

4.6 Numerical Experiments

We highlight the computational strength and scalability of proposed algorithm by testing it on a variety of randomly generated large scale linear programming (LP) and second order cone programming (SOCP) sparse problems. We compare the results with other first order solvers POGS (Proximal Graph Solver) [15], OSQP (Operator Splitting Solver for Quadratic Programs) [18] and SCS (Splitting Conic Solver) [16]. Our algorithm's computational performance is a significant improvement over other standard first order solvers. For each experiment, we apply the termination criteria used by competing solver and compare the residual norms. In addition to termination criteria of competing solver, we also compare objective value and constraint violations to terminate the proposed algorithm. The computational gain of proposed approach is consistent across all problem instances and different hardware architectures.

The proposed algorithm and competing solvers are implemented in MATLABR2020a and all the experiments are counducted on a Linux based DGX station with 20 2.2 GHz cores, Intel Xeon E5-2698 v4 CPU, with NVIDIA Tesla V100-DGXS-32GB (128 GB total) GPU processor and 256 GB of RAM. The parallel nature of algorithm enables the implementation to take advantage of multi-core CPU processing. Note that our implementation of proposed algorithm in MATLAB utilizes only a single GPU and does not benefit from multiple GPU's of the platform. Moreover, all experiments reported in this paper are not bounded by RAM or GPU memory of DGX station. We used the MATLAB interface of POGS, OSQP v0.6.0 and SCS v2.1.2.

Problem instances: All data is generated in such a way that the all problems are bounded and feasible. The number of nonzero entries of $\boldsymbol{A}$ are in the range of 10^4 to 10^6.

- We generate $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ to be a random sparse matrix with 0.1%, 0.5% and 1.0% nonzero elements, which are drawn i.i.d. (independently and identically distributed) from $\mathcal{N}(0, 1)$.
- $\boldsymbol{b} := \boldsymbol{A} \times \text{proj}_{\mathcal{K}}(\dot{\boldsymbol{x}})$ where the elements of $\dot{\boldsymbol{x}} \in \mathbb{R}^n$ have i.i.d standard normal distribution.
- The elements of $\boldsymbol{c} \in \mathbb{R}^n$ have i.i.d standard normal distribution.
- For LP $\mathcal{K} = \mathbb{R}^n_+$ and $\mathcal{K} = (\mathcal{K}_h)^{\frac{n}{h}}$, where $\mathcal{K}_h$ is the standard Lorentz cone of size h for SOCP problems.

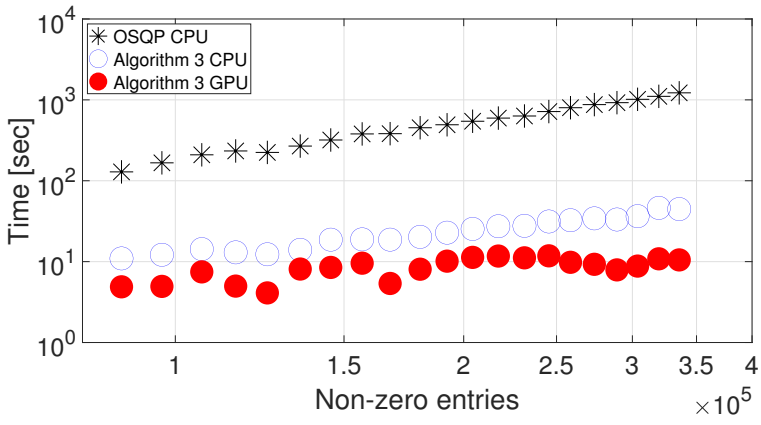
4.6.1 Linear Programming

We consider randomly generated linear programming problems and compare the result with first-order solvers. We compare the performance of algorithm 3 in comparison with POGS and OSQP on a variety of sparse problems.

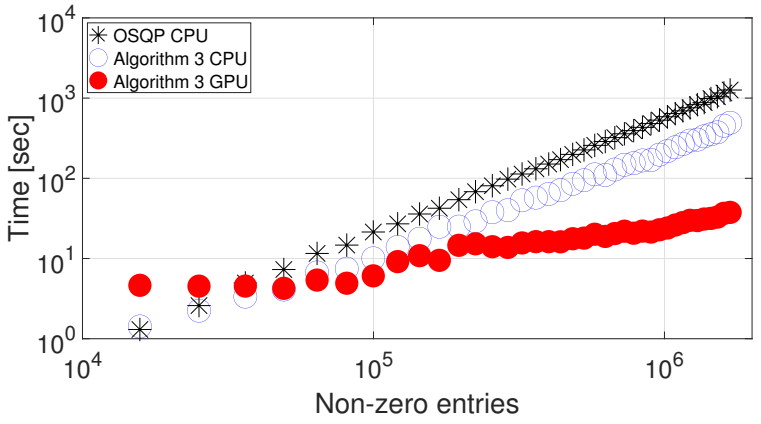
In each case study, the experiments are continued until the run time of the competing solver reaches a maximum time of 1200 seconds. The maximum time is chosen in such a way that the experiments provide sufficient information to compare the computational time for all solvers.

4.6.1.1 Comparison with OSQP

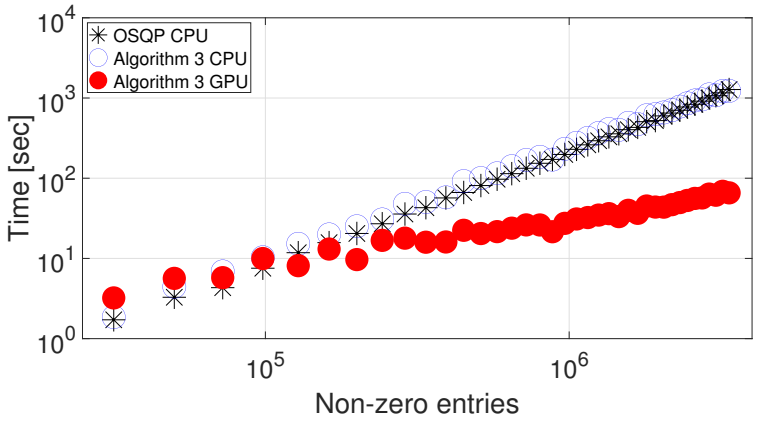
OSQP is a first order general purpose open-source solver based on alternating direction method on multiplies (ADMM). In figure 4-1, we demonstrate the computational performance of our algorithm in comparison with OSQP by running several



(a)



(b)



(c)

Figure 4-1: The performance of Algorithm 3 for linear programming in comparison with OSQP for (a) 0.1% (b) 0.5% and (c) 1.0% nonzero entries in matrix $\boldsymbol{A}$

sparse linear programming instances. For each instance, we solve the problem by using OSQP solver in its default parameter settings and then provide these parameters as input to our algorithm to achieve the same tolerance values.

Termination criteria: In all of the experiments, we use the following stopping criteria used by OSQP, where both primal and dual residuals are smaller than some predefined tolerance limits $\varepsilon_{\text{prim}} > 0$ and $\varepsilon_{\text{dual}} > 0$, i.e.,

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty < \varepsilon_{\text{prim}} \tag{4.18a}$$

$$\|\boldsymbol{A}^\top\boldsymbol{\lambda} + \boldsymbol{c}\|_\infty < \varepsilon_{\text{dual}} \tag{4.18b}$$

$$\varepsilon_{\text{prim}} = \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}} \max\{\|\boldsymbol{A}\boldsymbol{x}\|_\infty, \|\boldsymbol{b}\|_\infty\}$$

$$\varepsilon_{\text{dual}} = \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}} \max\{\|\boldsymbol{A}^\top\boldsymbol{\lambda}\|_\infty, \|\boldsymbol{c}\|_\infty\}$$

where $\varepsilon_{\text{abs}} = 10^{-4}$ and $\varepsilon_{\text{rel}} = 10^{-3}$ are default absolute and relative tolerance values, respectively. In addition to this stopping criteria, we also make sure to satisfy the following condition to terminate our algorithm

$$|\boldsymbol{c}^\top\boldsymbol{x} + \boldsymbol{b}^\top\boldsymbol{\lambda}| < |\boldsymbol{c}^\top\boldsymbol{x}^{\text{OSQP}} + \boldsymbol{b}^\top\boldsymbol{y}^{\text{OSQP}}|$$

Figure 4-1a, demonstrate that CPU implementation or proposed algorithm is at least 10 times faster than OSQP, whereas the GPU implementation shows two order or magnitude improvement. Similar results are depicted in 4-1b, where we used 0.2% nonzero values in constraint matrix $\boldsymbol{A}$. The CPU and GPU implementation

of proposed algorithm can easily achieve ten times and hundred times improvements respectively, while satisfying the same or even strict stopping criteria as compare to OSQP.

4.6.1.2 Comparison with POGS

POGS is an open-source implementation of graph projection splitting method to target the multi-core and GPU-based systems for solving optimization problems. In this case study, we compare the performance of proposed algorithm with POGS on both CPU and GPU system architectures in figure 4-2. In these experiments, POGS returns inaccurate solution in its default parameter settings, hence we made a slight change in default parameters by setting $\varepsilon_{\text{abs}} = 10^{-5}$ and $\varepsilon_{\text{rel}} = 10^{-4}$ to get reasonable accuracy. The same tolerance parameters along with primal and dual solutions returned by POGS are used as input parameters for proposed algorithm to meet the same stopping criteria.

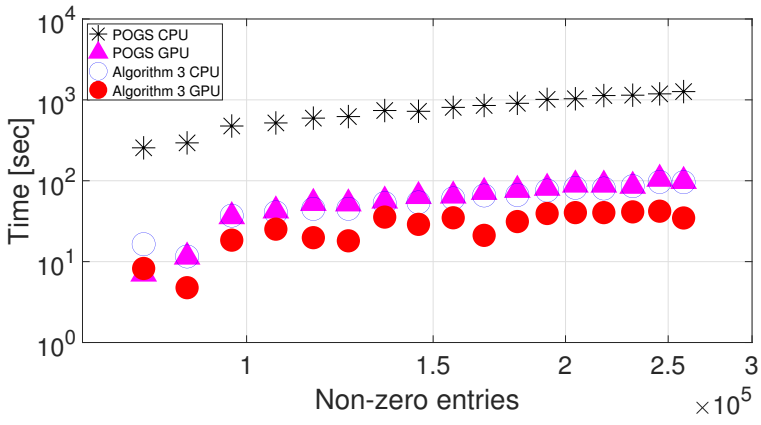
Termination criteria: In all of the experiments, the stopping criteria of Algorithm is when it exceeds both primal and dual feasibility of the solution produced by the competing solver. In other words, when the following two criteria are met:

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 < \|\boldsymbol{A}\boldsymbol{x}^{\text{POGS}} - \boldsymbol{b}\|_2 \tag{4.19a}$$

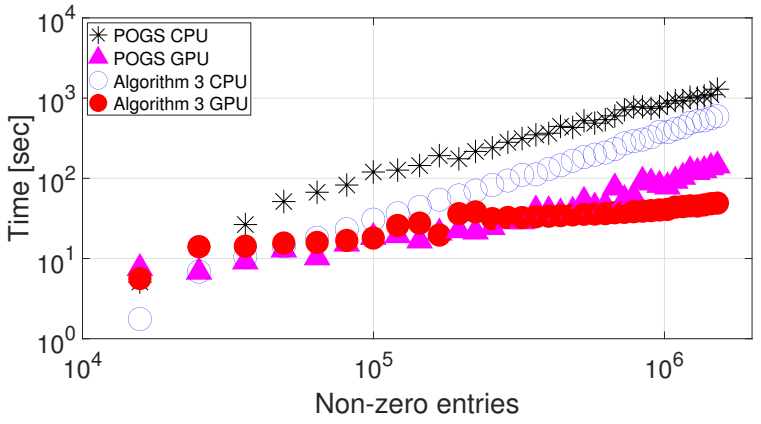
$$|\boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{\lambda}| < |\boldsymbol{c}^\top \boldsymbol{x}^{\text{POGS}} + \boldsymbol{b}^\top \boldsymbol{y}^{\text{POGS}}| \tag{4.19b}$$

where $\boldsymbol{x}^{\text{POGS}}$ and $\boldsymbol{y}^{\text{POGS}}$ are primal and dual solutions produced by the competing solver POGS under default settings.

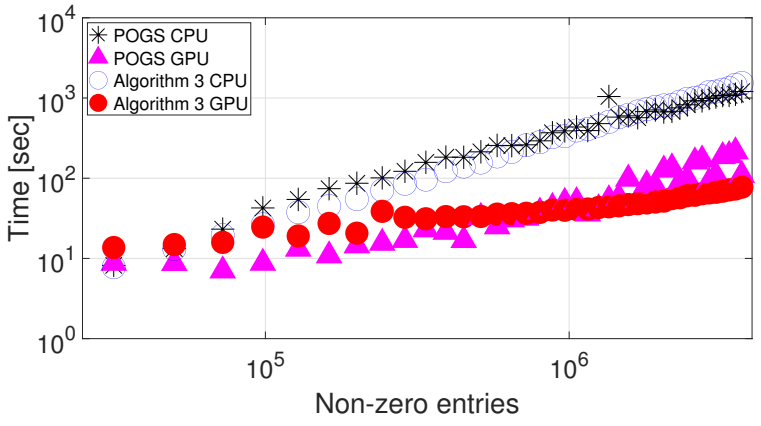
Figure 4-2a, and 4-2b, shows notable time improvements of proposed algorithm in both cases. Our CPU and GPU implementation of algorithm shows an order of



(a)



(b)



(c)

Figure 4-2: The performance of Algorithm 3 for linear programming in comparison with POGS for (a) 0.1% (b) 0.5% and (c) 1.0% nonzero entries in matrix $\boldsymbol{A}$

magnitude time improvements as compare to POGS, and this performance improvement keeps getting better as the problem becomes large.

4.6.2 Second-Order Cone Programming

In this case study, we consider the class of second-order cone programming optimization problems. We compare the performance of algorithm 3 in comparison with SCS on a variety of sparse conic problems.

4.6.2.1 Comparison with SCS

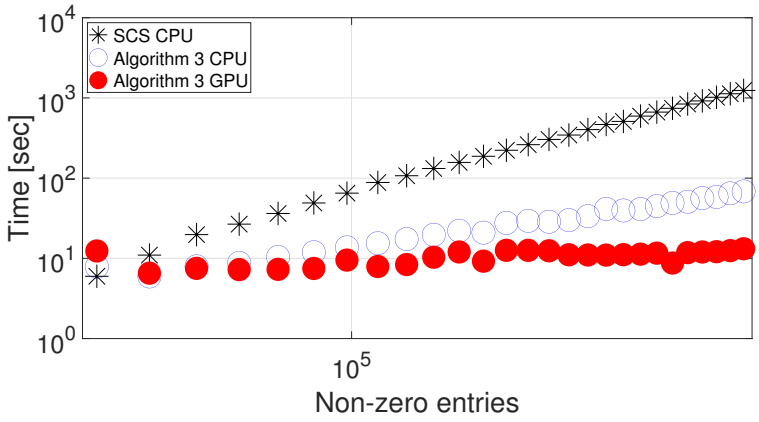
Splitting conic solver (SCS), primarily written in C, is a first-order numerical optimization solver for solving large-scale cone programs using ADMM. This solver returns both primal and dual solutions along with infeasibility certificate when applies.

Termination criteria: In all of the experiments, we use the following default stopping criteria of SCS solver.

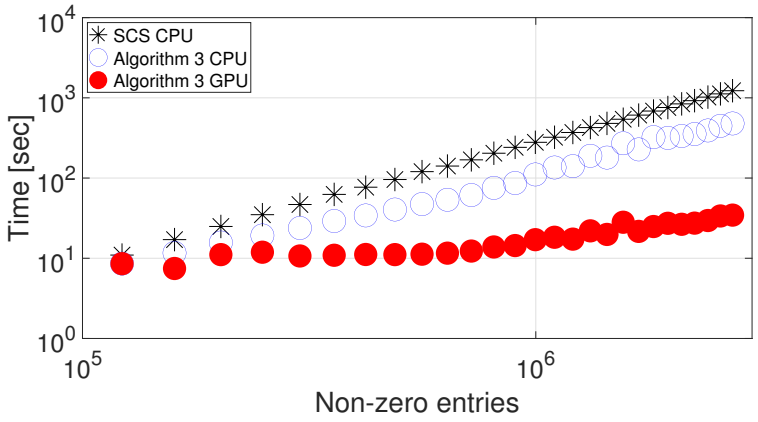
$$\frac{\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2}{(1+\|\boldsymbol{b}\|_2)} \leq \varepsilon_{\text{prim}}, \qquad \frac{\|\boldsymbol{A}^\top\boldsymbol{\lambda}+\boldsymbol{c}\|_2}{(1+\|\boldsymbol{c}\|_2)} \leq \varepsilon_{\text{dual}},$$
$$\frac{(\boldsymbol{c}^\top\boldsymbol{x}+\boldsymbol{b}^\top\boldsymbol{\lambda})}{1+|\boldsymbol{c}^\top\boldsymbol{x}|+|\boldsymbol{b}^\top\boldsymbol{\lambda}|} \leq \varepsilon_{\text{gap}}$$

where $\varepsilon_{\text{prim}} = \varepsilon_{\text{dual}} = \varepsilon_{\text{gap}} = 10^{-3}$.

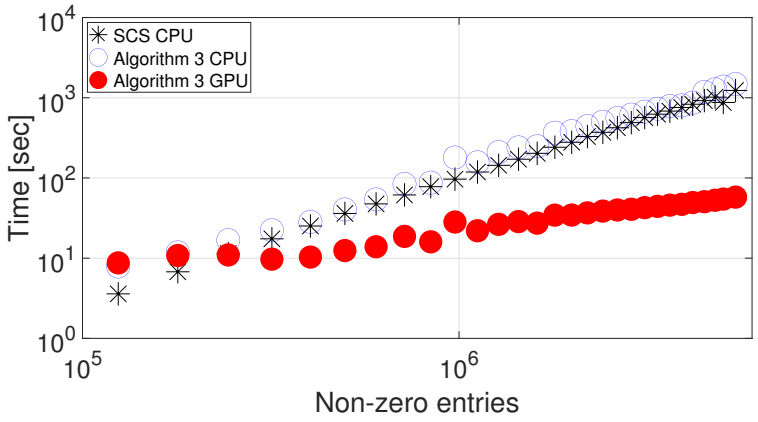
We solve multiple SOCP problems for different sparse density and satisfy the termination criteria of SCS solver and compare the computational time in figure 4-3. The proposed algorithms is approximately ten times faster than SCS.



(a)



(b)



(c)

Figure 4-3: The performance of Algorithm 3 for second order cone programming in comparison with SCS direct for (a) 0.1% (b) 0.5% and (c) 1.0% nonzero entries in matrix $\boldsymbol{A}$

4.7 Conclusion

Motivated by the requirement of solving linear systems in first-order methods and computational complexity of existing direct methods, we propose a computationally efficient and inexpensive matrix-free first order numerical algorithm for large sparse conic programs. The computational burden of solving the linear system is managed by decomposing the constraint into sparse factors in such a way that the decomposed factors are easy-to-compute. The matrix inverse lemma is used to make the algorithm division free. A highly parallelizable and inexpensive numerical algorithm is developed based on alternating direction method of multipliers. The iterative steps of the algorithm are division-free, can be computed by closed form solution, and involves only simple arithmetic operations. The parallel nature of algorithms enjoys the benefits of graphics processing unite implementation to speed up the computational gain of an order-of-magnitude time improvement. The proposed numerical algorithm is applied on a wide range of conic optimization programs. The numerical experiments shows that matrix-free algorithm is very competitive in comparison to other competing first-order solvers. Numerical experiments demonstrate that the proposed algorithm achieves notable time improvement in comparison to POGS, OSQP and SCS solvers.

CHAPTER 5

Conclusion

In this dissertation, we first introduced a proximal numerical method with potential for parallelization. This first order numerical algorithm is highly efficient and scalable for dealing with large problems, but in and of itself, the algorithm suffers from slow tail convergence similar to the existing first order numerical methods. Next, an adaptive conditioning heuristic was developed to accelerate the convergence of the proposed method. Most of the techniques available in literature are limited to a certain problem structures or conditions and cannot be applied to general cases of conic optimization. Moreover, we showed that convergence rate can be improved, irrespective of the condition number of data matrices. The proposed algorithm is implemented on graphics processing unit with an order-of-magnitude time improvement. A wide range of numerical experiments are conducted on large problems and results are compared with prominent first order solvers as well as the interior point method based solvers to demonstrate the performance of scalable and high accuracy algorithm. The numerical experiments show that the proposed algorithm outperforms the first order algorithms in terms of computational time and achieves the accuracy levels comparable to second-order interior point methods. We provide the proof of convergence of the algorithm.

Additionally, we introduced a matrix-free numerical algorithm for solving huge sparse conic programs. The first-order methods are required to solve the linear system of equations in each iteration. Several standard approaches (direct methods) exists in literature to solve the linear system when the problem size is not very large. The

direct methods are computationally expensive and thus not ideal for very large-scale problems. Alternatively, iterative approaches such as conjugate gradient methods (indirect methods) are applied for such problems, but these methods produces only the approximation and thus compromise on the accuracy of solution. High computational and memory cost of direct methods and low accuracy of indirect methods motivated us to design a numerical algorithm, which serves the purpose of computational efficiency and numerical accuracy at the same time. The original conic problem is reformulated based on a novel decomposition and a matrix free algorithm is developed for this new reformulated problem. The proposed algorithm is applied on a wide range of conic optimization problems. The numerical experiments show matrix-free algorithm outperforms other competing first-order solvers. Numerical experiments demonstrate that the proposed algorithm achieves approximately an order-of-magnitude time improvement in comparison to first order solvers POGS, OSQP and SCS solvers.

REFERENCES

[1] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019. [Online]. Available: http://docs.mosek.com/9.0/toolbox/index.html

[2] K. C. Toh, M. J. Todd, and R. H. Tütüncü, "SDPT3–A Matlab software package for semidefinite programming, Version 1.3," *Optimization Methods and Software*, vol. 11, no. 1-4, pp. 545–581, 1999. [Online]. Available: https://doi.org/10.1080/10556789908805762

[3] CVX Research Inc, "CVX: Matlab Software for Disciplined Convex Programming, version 2.0," http://cvxr.com/cvx, Aug. 2012.

[4] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[5] CPLEX IBM ILOG, "V12.1: User's Manual for CPLEX," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[6] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992. [Online]. Available: https://doi.org/10.1137/0802028

[7] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *2013 European Control Conference (ECC)*, 2013, pp. 3071–3076.

[8] R. H. Tütüncü, K. C. Toh, and M. J. Todd, "Solving semidefinite-quadratic-linear programs using SDPT3," *MATHEMATICAL PROGRAMMING*, vol. 95, pp. 189–217, 2003.

[9] J. F. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization methods and software*, vol. 11, no. 1-4, pp. 625–653, 1999.

[10] J. Eckstein and D. P. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 55, no. 1, pp. 293–318, Apr 1992. [Online]. Available: https://doi.org/10.1007/BF01581204

[11] P. Giselsson and S. Boyd, "Linear convergence and metric selection for Douglas-Rachford splitting and ADMM," *IRE Transactions on Automatic Control*, vol. 62, no. 2, pp. 532–544, 2 2017.

[12] A. Fu, J. Zhang, and S. P. Boyd, "Anderson accelerated Douglas-Rachford splitting," *arXiv: Optimization and Control*, 2019.

[13] P. Sopasakis, K. Menounou, and P. Patrinos, "SuperSCS: fast and accurate large-scale conic optimization," in *18th European Control Conference (ECC)*, 2019, pp. 1500–1505.

[14] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1561/2200000016

[15] C. Fougner and S. Boyd, *Parameter selection and preconditioning for a graph form solver*. Cham: Springer International Publishing, 2018, pp. 41–61. [Online]. Available: https://doi.org/10.1007/978-3-319-67068-3_4

[16] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, Jun 2016. [Online]. Available: https://doi.org/10.1007/s10957-016-0892-3

[17] Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn, "Chordal decomposition in operator-splitting methods for sparse semidefinite programs," *Mathematical Programming*, vol. 180, pp. 489–532, Mar. 2020.

[18] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *ArXiv e-prints*, Jan. 2018. [Online]. Available: https://arxiv.org/abs/1711.08013

[19] R. Madani, A. Kalbat, and J. Lavaei, "A low-complexity parallelizable numerical algorithm for sparse semidefinite programming," *IEEE Transactions on Control of Network Systems*, vol. 5, no. 4, pp. 1898–1909, 2018.

[20] ——, "ADMM for sparse semidefinite programming with applications to optimal power flow problem," in *2015 54th IEEE Conference on Decision and Control (CDC)*, 2015, pp. 5932–5939.

[21] M. Garstka, M. Cannon, and P. Goulart, "Cosmo: A conic operator splitting method for convex conic problems," 2020.

[22] A. Themelis and P. Patrinos, "SuperMann: A superlinearly convergent algorithm for finding fixed points of nonexpansive operators," *IEEE Transactions on Automatic Control*, vol. 64, no. 12, pp. 4875–4890, 2019.

[23] Z. Xu, G. Taylor, H. Li, M. A. T. Figueiredo, X. Yuan, and T. Goldstein, "Adaptive consensus ADMM for distributed optimization," ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 3841–3850.

[24] E. Ghadimi, A. Teixeira, I. Shames, and M. Johansson, "Optimal parameter selection for the alternating direction method of multipliers (ADMM): Quadratic problems," *IEEE Transactions on Automatic Control*, vol. 60, no. 3, pp. 644–658, 2015.

[25] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. Jordan, "A general analysis of the convergence of ADMM," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37. ICML'15, 2015, pp. 343–352.

[26] M. Hong and Z. Luo, "On the linear convergence of the alternating direction method of multipliers," *Mathematical Programming*, vol. 162, no. 1-2, pp. 165–199, 2017.

[27] W. Ouyang, Y. Peng, Y. Yao, J. Zhang, and B. Deng, "Anderson acceleration for nonconvex ADMM based on Douglas-Rachford splitting," *Computer Graphics Forum*, vol. 39, no. 5, pp. 221–239, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14081

[28] S. Wang and N. Shroff, "A new alternating direction method for linear programming," in *Advances in Neural Information Processing Systems*, vol. 30. NIPS, 2017, pp. 1480–1488. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/c4b31ce7d95c75ca70d50c19aef08bf1-Paper.pdf

[29] J. Eckstein and W. Yao, "Understanding the convergence of the alternating direction method of multipliers: Theoretical and computational perspectives," 2015.

[30] K. Guo, D. Han, and X. Yuan, "Convergence analysis of Douglas-Rachford splitting method for "strongly+weakly" convex programming," *SIAM Journal on Numerical Analysis*, vol. 55, no. 4, pp. 1549–1577, 2017. [Online]. Available: https://doi.org/10.1137/16M1078604

[31] L. Demanet and X. Zhang, "Eventual linear convergence of the Douglas-Rachford iteration for basis pursuit," *Math. Comput.*, vol. 85, pp. 209–238, 2016.

[32] G. Banjac and P. J. Goulart, "Global linear convergence in operator splitting methods," in *IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 233–238.

[33] G. Banjac and P. Goulart, "Tight global linear convergence rate bounds for operator splitting methods," *IEEE Transactions on Automatic Control*, vol. 63, pp. 4126–4139, 2018.

[34] P. Giselsson, M. Fält, and S. Boyd, "Line search for averaged operator iteration," in *IEEE 55th Conference on Decision and Control (CDC)*, Dec 2016, pp. 1015–1022.

[35] W. Deng and W. Yin, "On the global and linear convergence of the generalized alternating direction method of multipliers," *Journal of Scientific Computing*, vol. 66, no. 3, pp. 889–916, Mar 2016. [Online]. Available: https://doi.org/10.1007/s10915-015-0048-x

[36] C. Song, S. Yoon, and V. Pavlovic, "Fast ADMM algorithm for distributed optimization with adaptive penalty," in *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, ser. AAAI'16, 2016, p. 753–759.

[37] Y. Peng, B. , J. Zhang, F. Geng, W. Qin, and L. Liu, "Anderson acceleration for geometry optimization and physics simulation," *ACM Trans. Graph.*, vol. 37, no. 4, July 2018. [Online]. Available: https://doi.org/10.1145/3197517.3201290

[38] J. Zhang, Y. Peng, W. Ouyang, and B. Deng, "Accelerating ADMM for efficient simulation and optimization," vol. 38, no. 6, 2019. [Online]. Available: https://doi.org/10.1145/3355089.3356491

[39] J. Zhang, B. O'Donoghue, and S. P. Boyd, "Globally convergent type-I Anderson acceleration for non-smooth fixed-point iterations," *arXiv: Optimization and Control*, 2018.

[40] H. Fang and Y. Saad, "Two classes of multisecant methods for nonlinear acceleration," *Numerical Linear Algebra with Applications*, vol. 16, no. 3, pp. 197–221, Mar. 2009.

[41] P. Giselsson and S. Boyd, "Metric selection in fast dual forward–backward splitting," *Automatica*, vol. 62, pp. 1 – 10, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109815003611

[42] P. Giselsson and S. P. Boyd, "Diagonal scaling in Douglas-Rachford splitting and ADMM," *53rd IEEE Conference on Decision and Control*, pp. 5033–5039, 2014.

[43] P. Giselsson and S. Boyd, "Preconditioning in fast dual gradient methods," in *53rd IEEE Conference on Decision and Control*, Dec 2014, pp. 5040–5045.

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

[45] S. Diamond and S. Boyd, "Stochastic matrix-free equilibration," *Journal of Optimization Theory and Applications*, vol. 172, no. 2, pp. 436–454, Feb 2017. [Online]. Available: https://doi.org/10.1007/s10957-016-0990-2

[46] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[47] Y. Nesterov, *Lectures on Convex Optimization*, 2nd ed. Springer Publishing Company, Incorporated, 2018.

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

[49] S. C. Althoen and R. Mclaughlin, "Gauss-jordan reduction: A brief history," *The American Mathematical Monthly*, vol. 94, no. 2, pp. 130–142, 1987. [Online]. Available: https://doi.org/10.1080/00029890.1987.12000605

[50] P. S. Stanimirović and M. D. Petković, "Gauss–jordan elimination method for computing outer inverses," *Applied Mathematics and Computation*, vol. 219, no. 9, pp. 4667–4679, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0096300312010971

[51] R. Mittal and A. Al-Kurdi, "Lu-decomposition and numerical structure for solving large sparse nonsymmetric linear systems," *Computers & Mathematics with Applications*, vol. 43, no. 1, pp. 131–155, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0898122101002796

[52] I. P. Stanimirović and M. B. Tasić, "Computation of generalized inverses by using the ldl decomposition," *Applied Mathematics Letters*, vol. 25, no. 3, pp. 526–531, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893965911004630

[53] B. Fathi Vajargah, "A way to obtain monte carlo matrix inversion with minimal error," *Applied Mathematics and Computation*, vol. 191, no. 1, pp. 225–233, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0096300307002494

[54] S. Diamond and S. Boyd, "Matrix-free convex optimization modeling," in *Optimization and its applications in control and data sciences*. Springer, 2016, pp. 221–264.

[55] J. Al-Jeiroudi, G. Gondzio, "Convergence analysis of the inexact infeasible interior-point method for linear optimization," *Journal of Optimization Theory and Applications*, vol. 141, no. 2, pp. 231–247, 2009. [Online]. Available: https://doi.org/10.1007/s10957-008-9500-5

[56] K.-C. Toh, "Solving large scale semidefinite programs via an iterative solver on the augmented systems," *SIAM Journal on Optimization*, vol. 14, no. 3, pp. 670–698, 2004. [Online]. Available: https://doi.org/10.1137/S1052623402419819

[57] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of research of the National Bureau of Standards*, vol. 49, pp. 409–436, 1952.

[58] M. Fukuda, M. Kojima, and M. Shida, "Lagrangian dual interior-point methods for semidefinite programs," *SIAM Journal on Optimization*, vol. 12, no. 4, pp. 1007–1031, 2002. [Online]. Available: https://doi.org/10.1137/S1052623401387349

[59] M. KoăundefinedVara and M. Stingl, "On the solution of large-scale sdp problems by the modified barrier method using iterative solvers," *Math. Program.*, vol. 109, no. 2–3, p. 413–444, Mar. 2007.

[60] C. Choi and Y. Ye, "Solving sparse semidefinite programs using the dual scaling algorithm with an iterative solver," *Working paper, Department of Management Sciences, University of Iowa*, 2000.

[61] X.-Y. Zhao, D. Sun, and K.-C. Toh, "A newton-cg augmented lagrangian method for semidefinite programming," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1737–1765, 2010. [Online]. Available: https://doi.org/10.1137/080718206

[62] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, ser. Frontiers in Applied Mathematics. SIAM, 1995, no. 16. [Online]. Available: http://www.siam.org/books/textbooks/fr16_book.pdf

[63] M. DApuzzo, V. De Simone, and D. di Serafino, "On mutual impact of numerical linear algebra and large-scale optimization with focus on interior point methods," *Computational Optimization and Applications*, vol. 45, no. 2, pp. 283–310, 2010. [Online]. Available: https://doi.org/10.1007/s10589-008-9226-1

[64] Z. Lu, R. D. C. Monteiro, and J. W. O'Neal, "An iterative solver-based infeasible primal-dual path-following algorithm for convex quadratic programming," *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 287–310, 2006. [Online]. Available: https://doi.org/10.1137/04060771X

[65] F. E. Curtis, J. Nocedal, and A. Wächter, "A matrix-free algorithm for equality constrained optimization problems with rank-deficient jacobians," *SIAM J. on Optimization*, vol. 20, no. 3, p. 1224–1249, Sept. 2009.

[66] M. Saunders, B. Kim, C. Maes, A. Santiago, and M. Zahr, "PDCO: primal-dual interior method for convex Objectives," *http://www.stanford.edu/group/SOL/software/pdco.html*, 2018.

[67] J. Gondzio, "Matrix-free interior point method," *Comput. Optim. Appl.*, vol. 51, no. 2, p. 457–480, Mar. 2012. [Online]. Available: https://doi.org/10.1007/s10589-010-9361-3

[68] M. Adil, S. Tavakkol, and R. Madani, "Rapid convergence of first-order numerical algorithms via adaptive conditioning," 2021.

[69] E. Chu, B. O'Donoghue, N. Parikh, and S. P. Boyd, "A primal-dual operator splitting method for conic optimization," 2013.

[70] J. Douglas and H. H. Rachford, "On the numerical solution of heat conduction problems in two and three space variables," *Transactions of the American Mathematical Society*, vol. 82, no. 2, pp. 421–439, 1956. [Online]. Available: http://www.jstor.org/stable/1993056

[71] R. J. Vanderbei, "Symmetric quasidefinite matrices," *SIAM Journal on Optimization*, vol. 5, no. 1, pp. 100–113, 1995. [Online]. Available: https://doi.org/10.1137/0805005

[72] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)*. USA: Johns Hopkins University Press, 1996.

[73] J. Nocedal and S. Wright, *Numerical Optimization: Springer Series in Operations Research and Financial Engineering*. Springer, 2006.

[74] C. C. Paige and M. A. Saunders, "Lsqr: An algorithm for sparse linear equations and sparse least squares," *ACM Trans. Math. Software*, pp. 43–71, 1982.

[75] H. Huang, J. M. Dennis, L. Wang, and P. Chen, "A scalable parallel lsqr algorithm for solving large-scale linear system for tomographic problems: A case study in seismic tomography," *Procedia Computer Science*, vol. 18, pp. 581–590, 2013, 2013 International Conference on Computational Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050913003657

[76] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Society for Industrial and Applied Mathematics, 2002. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898718027

BIOGRAPHICAL STATEMENT

Muhammad Adil was born in a small village in northern Pakistan. He received his B.S. degree in Electrical Engineering from University of Engineering and Technology and his M.S. degree from Pakistan Institute of Engineering and Applied Sciences Islamabd in 2012 and 2014, respectively. He worked toward his PhD degree from 2017 to 2021 at University of Texas at Arlington. His research interested includes developing numerical algorithm for solving optimization problems appears in power systems, control systems and machine learning.