Rapid Convergence of First-Order Numerical Algorithms via Adaptive Conditioning

Muhammad Adil, Sasan Tavakkol, and Ramtin Madani

***Abstract*—This paper is an attempt to remedy the problem of slow convergence for first-order numerical algorithms by proposing an adaptive conditioning heuristic. First, we propose a parallelizable numerical algorithm that is capable of solving large-scale conic optimization problems on distributed platforms such as graphics processing unit with orders-of-magnitude time improvement. Proof of global convergence is provided for the proposed algorithm. We argue that on the contrary to common belief, the condition number of the data matrix 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. Numerical experiments on a wide range of large-scale linear programming and second-order cone programming problems demonstrate the scalability and computational advantages of the proposed algorithm compared to commercial and open-source state-of-the-art solvers.**

I. Introduction

Conic optimization is of practical interest in a wide variety of areas such as operation research, machine learning, signal processing and optimal control. For this purpose, interior point-based algorithms perform very well and have become the standard method of solving conic optimization problems [1]–[3]. Various commercial and open-source solvers such as MOSEK [4], GUROBI [5], and SeDuMi [6] 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.

In recent years, operator splitting methods such as Douglas-Rachford Splitting (DRS) [7]–[10] and Alternating Direction Method of Multipliers (ADMM) [11]–[17] have received particular attention because of their potential for parallelization and ability to scale. 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 [13], 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 [14] extended the homogeneous self-dual embedding concept [13] and exploits the sparsity structure using chordal decomposition for solving large scale semidefinite programming problems. The ADMM algorithm introduced in [11] is improved by selecting the proximal parameter and pre-conditioning to introduce an open-source software package called POGS (Proximal Graph Solver) [12] 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) [15], where operator splitting technique is applied to solve quadratic programs. Open-source Julia implemented conic operator splitting method (COSMO) [18], solves the quadratic objective function under conic constraints. In [9], 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 [10], [19], [20].

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 [8], [15], [21], [22]. 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 [23], [24]. 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 [19], [22], [25]–[30]. A line search method is proposed in [31] to accelerate convergence. In [32], 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 [8] 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 [20], [33] to automatically tune the penalty parameter. In [34], [35], 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 [36], applied the

Muhammad Adil and Ramtin Madani are with the University of Texas at Arlington. Sasan Tavakkol is with Google Research. This work is funded, in part, by the Office of Naval Research under award N00014-18-1-2186, and approved for public release under DCN# 43-7474-20.

type-I variant of Anderson acceleration [37] to splitting conic solver (SCS) [13] to solve conic optimization problems and improved the terminal convergence. A new framework known as SuperSCS is introduced in [10] 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 [9], 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 [12], [14], [22]. Parameter selection for global convergence is still a challenge to be addressed [8], [15]. 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 [9], [26], [30]. 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.

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 propose a highly scalable, simple iterative, and parallelizable first order algorithm for solving large conic optimization programs.
2) 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.
3) 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.
4) We apply the proposed algorithm on graphics processing unit (GPU) to benefit the simple arithmetic operations in each iteration.
5) 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.

The organization of the rest of this paper is as follows. Some preliminaries of cone programming and definitions are presented in section II. The effect of preconditioning and the need for proposed adaptive conditioning is illustrated in section III by providing a numerical example. In section IV, 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 V, by solving a wide range of problems and comparing the results with commonly used solvers, and section VI concludes the paper.

A. Notations

Symbols $\mathbb{R}$ 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.

II. Preliminaries

In this paper, we consider the class of convex 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:

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

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.,

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

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

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

The corresponding dual formulation of (1) is

$$\begin{aligned} \underset{\boldsymbol{y}\in\mathbb{R}^m, \boldsymbol{z}\in\mathbb{R}^n}{\text{maximize}} \quad & \boldsymbol{b}^\top \boldsymbol{y} && \text{(2a)} \\ \text{subject to} \quad & \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{z} = \boldsymbol{c} && \text{(2b)} \\ & \boldsymbol{z} \in \mathcal{K} && \text{(2c)} \end{aligned}$$

where $\boldsymbol{y}$ and $\boldsymbol{z}$ are dual variables associated with the constraints (1b) and (1c), respectively.

In this paper, we pursue a proximal numerical method inspired by Douglas-Rachford splitting [7], [8] to solve the class of optimization problems of the form (1). 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 (1).

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*

$$\lim_{l\to\infty} \frac{\boldsymbol{p}^l + \boldsymbol{s}^l}{2} = \bar{\boldsymbol{x}} \qquad \text{and} \qquad \lim_{l\to\infty} \frac{\boldsymbol{p}^l - \boldsymbol{s}^l}{2\mu} = \bar{\boldsymbol{z}} \quad \text{(3)}$$

where $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{z}}$ are a pair of primal and dual solutions for problems (1) *and* (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 (1) may not offer any advantages over the common-practice Douglas-Rachford splitting (DR) and the Alternating Direction Method of Multipliers (ADMM). However, as we will demonstrate next, Algorithm (1) enables us to perform adaptive conditioning to achieve much faster convergence speed in comparison with the state-of-the-art pre-conditioning methods.

III. State of the Art Preconditioning Methods

One of the major drawbacks of first-order numerical methods is their sensitivity to the problem conditioning [12], [15], [38]. Hence, it is common practice to reformulate problem (1) with respect to new parameters

$$\hat{\boldsymbol{A}} \triangleq \boldsymbol{D}\boldsymbol{A}\boldsymbol{E}, \quad \hat{\boldsymbol{b}} \triangleq \boldsymbol{D}\boldsymbol{b}, \quad \text{and} \quad \hat{\boldsymbol{c}} \triangleq \boldsymbol{E}\boldsymbol{c} \qquad \text{(4)}$$

and new proxy variables

$$\hat{\boldsymbol{x}} = \boldsymbol{E}^{-1}\boldsymbol{x} \quad \text{and} \quad \hat{\boldsymbol{z}} = \boldsymbol{E}^\top \boldsymbol{z} \qquad \text{(5)}$$

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 [21], [38]–[40]. 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 $\hat{\boldsymbol{A}}$ [39], [41]. To this end, a number of matrix equilibration heuristics such as Sinkhorn-Knopp and Ruiz methods have been proposed in [12], [14], [15], [42] 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.

A. Example: The effect of condition number

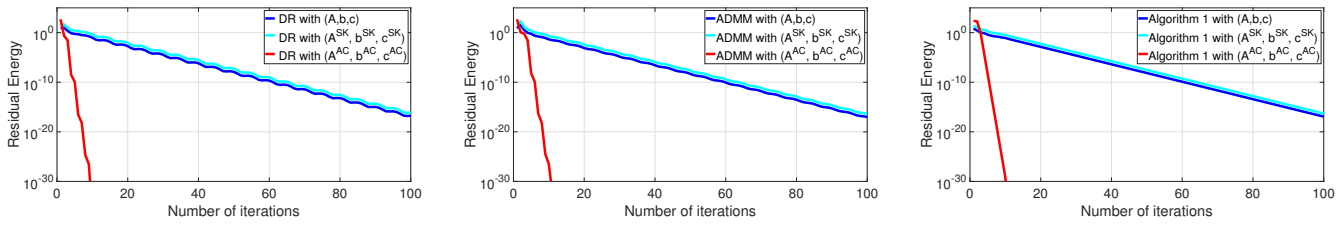
In this example, we provide simple data matrices and compare the effect of different pre-conditioning methods on the convergence of Algorithm (1), DR splitting, and ADMM. The goal is to demonstrate that the condition number of $\hat{\boldsymbol{A}}$ is not a reliable predictor of the convergence speed.

Consider the following data matrices:

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

and $\mathcal{K} := \mathbb{R}^5_+$. The corresponding diagonal matrices obtained from regularized Sinkhorn-Knopp algorithm [12] for ℓ_2 norms are

$$\begin{aligned} \boldsymbol{D}^{\text{SK}} &= \text{diag}\{[0.0217, 0.0215, 0.0222]\}, \\ \boldsymbol{E}^{\text{SK}} &= \text{diag}\{[0.4722, 0.4722, 0.4722, 0.4722, 0.4722]\}, \end{aligned}$$



(a) (b) (c)

Fig. 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.

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.

1) Douglas-Rachford Splitting: In order to implement the DR splitting method, it is common practice to cast problem (1) in the form of

$$\text{minimize} \quad f(\boldsymbol{x}) + g(\boldsymbol{x}) \tag{9}$$

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:

$$\boldsymbol{x} \leftarrow \mathrm{prox}_f(\boldsymbol{z}) \tag{10a}$$
$$\boldsymbol{z} \leftarrow \boldsymbol{z} + \mathrm{prox}_g(2\boldsymbol{x} - \boldsymbol{z}) - \boldsymbol{x}. \tag{10b}$$

2) Alternating Direction Method of Multipliers: A standard way of solving problem (1) via ADMM is through the formulation

$$\underset{\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n}{\text{minimize}} \quad f(\boldsymbol{x}_1) + g(\boldsymbol{x}_2) \tag{11a}$$
$$\text{subject to} \quad \boldsymbol{x}_1 = \boldsymbol{x}_2 \tag{11b}$$

leading to the steps

$$\boldsymbol{x}_1 \leftarrow \mathrm{prox}_{\mu^{-1} f}(\boldsymbol{x}_2 - \mu^{-1}\boldsymbol{z}) \tag{12a}$$
$$\boldsymbol{x}_2 \leftarrow \mathrm{prox}_{\mu^{-1} g}(\boldsymbol{x}_1 + \mu^{-1}\boldsymbol{z}) \tag{12b}$$
$$\boldsymbol{z} \leftarrow \boldsymbol{z} + \mu(\boldsymbol{x}_1 - \boldsymbol{x}_2). \tag{12c}$$

where μ is a fixed tuning parameter.

Figure (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 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.

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 $\mathcal{L} \subseteq \mathbb{N}$

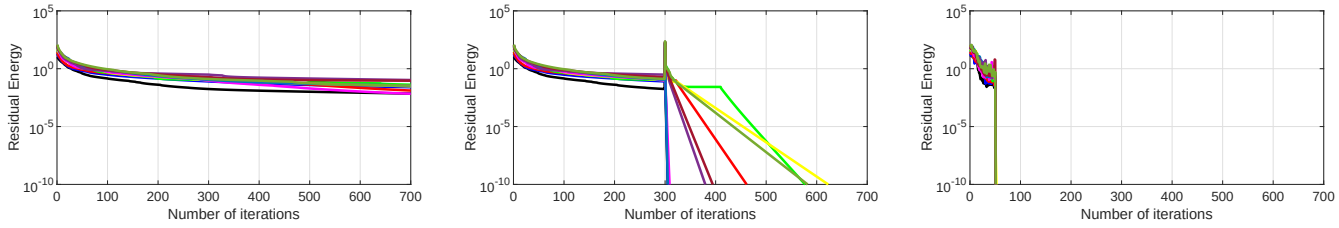
$l \leftarrow 0$
repeat
  $l \leftarrow l + 1$
  if $l \in \mathcal{L} \cup \{1\}$ **then**
    for $i = 1, \ldots, k$ **do**
      $h \leftarrow n_1 + \ldots + n_{i-1}$
      for $j = h+1, \ldots, h+n_i$ **do**
        $\boldsymbol{o}_j \leftarrow \left|x_{h+1} - \|[x_{h+2}, \ldots, x_{h+n_i}]\|_2\right|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

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

IV. 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 (1) is reformulated as:

$$\underset{\hat{\boldsymbol{x}} \in \mathbb{R}^n}{\text{minimize}} \quad (\boldsymbol{O}^\top \boldsymbol{c})^\top \hat{\boldsymbol{x}} \tag{13a}$$
$$\text{subject to} \quad (\boldsymbol{A}\boldsymbol{O})\hat{\boldsymbol{x}} = \boldsymbol{b} \tag{13b}$$
$$\hat{\boldsymbol{x}} \in \boldsymbol{O}^{-1}\mathcal{K} \tag{13c}$$



(a) (b) (c)

Fig. 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., $\mathcal{L} := \{1, 2, \ldots, 50\}$.

and the dual problem (2) as:

$$\begin{aligned} \underset{\boldsymbol{y}\in\mathbb{R}^m, \hat{\boldsymbol{z}}\in\mathbb{R}^n}{\text{maximize}} \quad & \boldsymbol{b}^\top \boldsymbol{y} && \text{(14a)} \\ \text{subject to} \quad & (\boldsymbol{AO})^\top \boldsymbol{y} + \hat{\boldsymbol{z}} = \boldsymbol{O}^\top \boldsymbol{c} && \text{(14b)} \\ & \hat{\boldsymbol{z}} \in \boldsymbol{O}^\top \mathcal{K}^* && \text{(14c)} \end{aligned}$$

where

$$\hat{\boldsymbol{x}} \triangleq \boldsymbol{O}^{-1}\boldsymbol{x} \quad \text{and} \quad \hat{\boldsymbol{z}} \triangleq \boldsymbol{O}^\top \boldsymbol{z} \tag{15}$$

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, 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.

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

A. 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., $\mathcal{L} := \varnothing$,

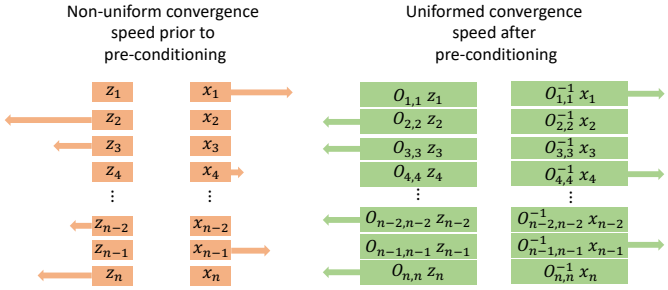


Fig. 3: Intuitive reason behind adaptive conditioning to equalize the convergence speed

- 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 2 for all 10 random instance. As demonstrated in the figure, even a one time adaptive conditioning results in significant improvement of convergence speed.

V. Numerical Experiments

In this section 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 [4], GUROBI [5] as well as the open source software OSQP [15] and POGS [12]. Our case studies consist of randomly generated linear programming (LP) 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:

$$\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2 < \|\boldsymbol{A}\boldsymbol{x}^{\text{CS}}-\boldsymbol{b}\|_2 \tag{16a}$$

$$|(\boldsymbol{A}^\dagger \boldsymbol{b})^\top(\boldsymbol{c}-\boldsymbol{z})-\boldsymbol{c}^\top\boldsymbol{x}| < |\boldsymbol{b}^\top\boldsymbol{y}^{\text{CS}}-\boldsymbol{c}^\top\boldsymbol{x}^{\text{CS}}| \tag{16b}$$

where $\boldsymbol{x}^{\text{CS}}$ and $\boldsymbol{y}^{\text{CS}}$ 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.

A. Linear Programming

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.,

$$\mathcal{L} = \{300, 400, 500, \ldots\}. \tag{17}$$

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 size becomes larger, 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 (16). As demonstrated in Figure 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.

B. 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 6 shows that the multi-core CPU implementation is approximately 10 times faster than the single-core implementation.

C. 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 5, we perform comparisons between Algorithm 2 and the first-order solver POGS [12]. We use the default settings for POGS except for the absolute and relative tolerance values ε^{abs} and ε^{rel} for stopping criteria. Figure 5 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 (16).

D. 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 $m = \lfloor 0.8n \rfloor$:

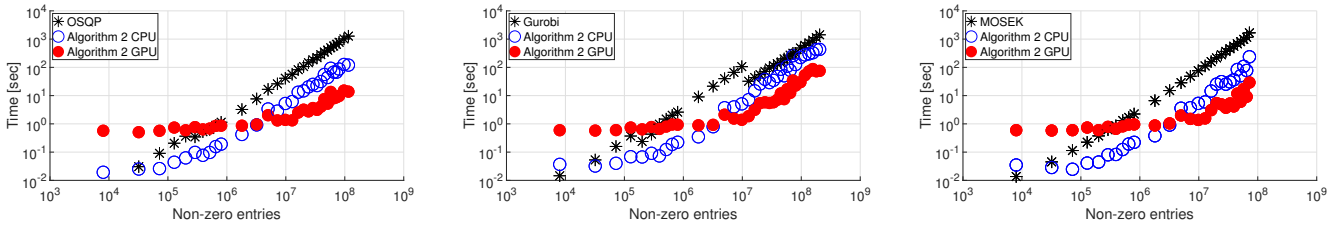
- The elements of $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ have i.i.d uniform distribution from the interval $[-1, 1]$.
- $\boldsymbol{b} := \boldsymbol{A} \times \text{abs}_{\mathcal{K}}(\dot{\boldsymbol{x}})$ where the elements of $\dot{\boldsymbol{x}} \in \mathbb{R}^n$ have i.i.d uniform distribution 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.,

$$\mathcal{L} = \{200, 300, 400, \ldots\}. \tag{18}$$

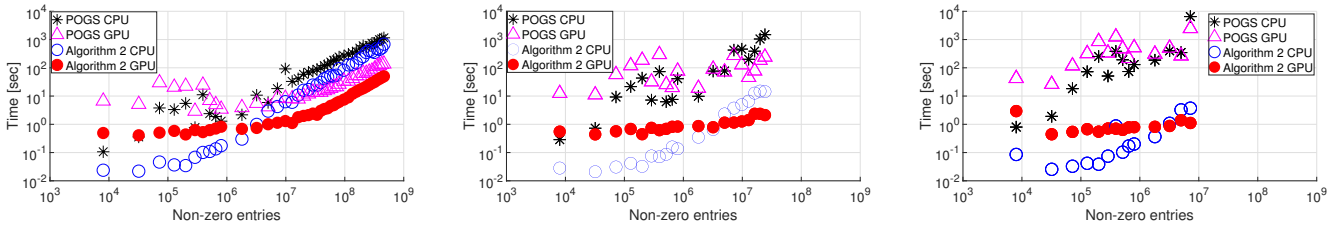
Parameters t and μ are set to 1.7 and 1, respectively.

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



(a) (b) (c)

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



(a) (b) (c)

Fig. 5: 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) $\varepsilon^{\text{abs}} = 10^{-7}$, $\varepsilon^{\text{rel}} = 10^{-6}$.

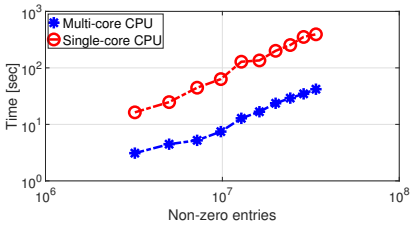


Fig. 6: Single-core vs Multi-core CPU implementation

VI. 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 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. The experimental results show 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.

References

[1] 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

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

[3] 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.

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

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

[6] 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.

[7] 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

[8] 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.

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

[10] 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.

[11] 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

[12] 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

[13] 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.

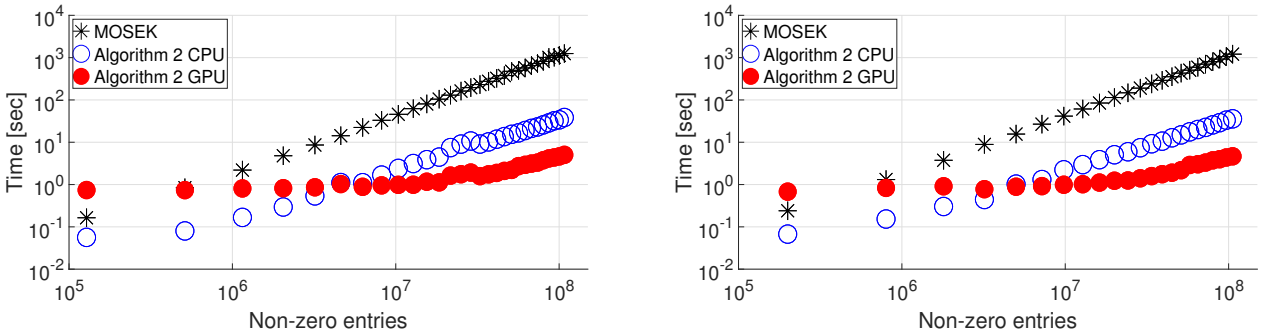


Fig. 7: The performance of Algorithm 2 for second order cone programming in comparison with MOSEK with Lorentz cones of size (a) $h = 4$ and (b) $h = 10$

1042–1068, Jun 2016. [Online]. Available: https://doi.org/10.1007/s10957-016-0892-3

[14] 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.

[15] 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

[16] 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.

[17] ——, "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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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

[25] 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

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

[27] 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

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

[29] 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.

[30] 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.

[31] 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.

[32] 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

[33] 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.

[34] 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, Jul. 2018. [Online]. Available: https://doi.org/10.1145/3197517.3201290

[35] 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

[36] 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.

[37] 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.

[38] 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

[39] 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.

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

[41] 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.

[42] 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

APPENDIX

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

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

$$|\cdot|_\star \triangleq \mathrm{abs}_{\mathcal{A}}(\mathrm{abs}_{\mathcal{K}}(\cdot)). \tag{19}$$

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

$$\||\boldsymbol{u}|_\star\|_2 = \|\boldsymbol{u}\|_2 \tag{20}$$

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

$$|\boldsymbol{u}|_\star^\top |\boldsymbol{v}|_\star \geq \boldsymbol{u}^\top \boldsymbol{v}. \tag{21}$$

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* (1) *and* (2)*. Then*

$$\boldsymbol{s}^{\mathrm{opt}} \triangleq \boldsymbol{x}^{\mathrm{opt}} - \mu \boldsymbol{z}^{\mathrm{opt}} \tag{22}$$

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

$$\boldsymbol{d} = \frac{\boldsymbol{s}^{\mathrm{opt}} + |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2}. \tag{23}$$

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

$$\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}} + \boldsymbol{A}^\top \boldsymbol{y}^{\mathrm{opt}} = 0 \tag{24a}$$
$$\boldsymbol{A}\boldsymbol{x}^{\mathrm{opt}} = \boldsymbol{b} \tag{24b}$$
$$(\boldsymbol{x}^{\mathrm{opt}})^\top \boldsymbol{z}^{\mathrm{opt}} = 0, \qquad \boldsymbol{x}^{\mathrm{opt}} \in \mathcal{K}, \qquad \boldsymbol{z}^{\mathrm{opt}} \in \mathcal{K}. \tag{24c}$$

Define

$$\boldsymbol{p}^{\mathrm{opt}} \triangleq \boldsymbol{x}^{\mathrm{opt}} + \mu \boldsymbol{z}^{\mathrm{opt}}, \tag{25}$$

then according to (24c), we have:

$$\boldsymbol{p}^{\mathrm{opt}} = \mathrm{abs}_{\mathcal{K}}(\boldsymbol{s}^{\mathrm{opt}}). \tag{26}$$

Hence

$$\boldsymbol{d} - \frac{\boldsymbol{s}^{\mathrm{opt}} + |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2} = \tag{27a}$$
$$\boldsymbol{A}^\dagger \boldsymbol{b} + \frac{\mu(\mathrm{abs}_{\mathcal{A}}(\boldsymbol{c}) - \boldsymbol{c})}{2} - \frac{\boldsymbol{s}^{\mathrm{opt}} + |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2} \overset{(26)}{=} \tag{27b}$$
$$\boldsymbol{A}^\dagger \boldsymbol{b} + \frac{\mu(\mathrm{abs}_{\mathcal{A}}(\boldsymbol{c}) - \boldsymbol{c})}{2} - \frac{\boldsymbol{s}^{\mathrm{opt}} + \mathrm{abs}_{\mathcal{A}}(\boldsymbol{p}^{\mathrm{opt}})}{2} = \tag{27c}$$
$$\boldsymbol{A}^\dagger(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{\mathrm{opt}}) - \mu(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}} - \boldsymbol{A}^\dagger \boldsymbol{A}(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}})) \overset{(24b)}{=} \tag{27d}$$
$$\mu(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}} - \boldsymbol{A}^\dagger \boldsymbol{A}(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}})) \overset{(24a)}{=} \tag{27e}$$
$$\mu(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}} + \boldsymbol{A}^\dagger \boldsymbol{A}\boldsymbol{A}^\top \boldsymbol{y}^{\mathrm{opt}}) = \tag{27f}$$
$$\mu(\boldsymbol{c} - \boldsymbol{z}^{\mathrm{opt}} + \boldsymbol{A}^\top \boldsymbol{y}^{\mathrm{opt}}) \overset{(24a)}{=} \boldsymbol{0}_n, \tag{27g}$$

which concludes (23). 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* (1) *and* (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

$$\boldsymbol{s}^{l+1} = \frac{\boldsymbol{s}^l - |\boldsymbol{s}^l|_\star}{2} + \boldsymbol{d} \tag{28}$$

and due to (23):

$$\boldsymbol{s}^{l+1} = \frac{\boldsymbol{s}^l + \boldsymbol{s}^{\mathrm{opt}}}{2} - \frac{|\boldsymbol{s}^l|_\star - |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2}. \tag{29}$$

Hence

$$\|\boldsymbol{s}^{l+1} - \boldsymbol{s}^l\|_2^2 + \|\boldsymbol{s}^{l+1} - \boldsymbol{s}^{\mathrm{opt}}\|_2^2 - \|\boldsymbol{s}^l - \boldsymbol{s}^{\mathrm{opt}}\|_2^2 \overset{(29)}{=} \tag{30a}$$
$$\|\frac{\boldsymbol{s}^l - \boldsymbol{s}^{\mathrm{opt}}}{2} + \frac{|\boldsymbol{s}^l|_\star - |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2}\|_2^2 + \tag{30b}$$
$$\|\frac{\boldsymbol{s}^l - \boldsymbol{s}^{\mathrm{opt}}}{2} - \frac{|\boldsymbol{s}^l|_\star - |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2}\|_2^2 - \|\boldsymbol{s}^l - \boldsymbol{s}^{\mathrm{opt}}\|_2^2 = \tag{30c}$$
$$\frac{\||\boldsymbol{s}^l|_\star - |\boldsymbol{s}^{\mathrm{opt}}|_\star\|_2^2}{2} - \frac{\|\boldsymbol{s}^l - \boldsymbol{s}^{\mathrm{opt}}\|_2^2}{2} \overset{(20)}{=} \tag{30d}$$
$$(\boldsymbol{s}^l)^\top \boldsymbol{s}^{\mathrm{opt}} - |\boldsymbol{s}^l|_\star^\top |\boldsymbol{s}^{\mathrm{opt}}|_\star \overset{(21)}{\leq} 0, \tag{30e}$$

which concludes that $\{\|\boldsymbol{s}^l - \boldsymbol{s}^{\mathrm{opt}}\|_2\}_{l=0}^\infty$ is nonincreasing and convergent. Additionally, (30) concludes that

$$\sum_{l=0}^{\infty} \|\boldsymbol{s}^{l+1} - \boldsymbol{s}^l\|_2^2 \ \leq \ \|\boldsymbol{s}^0 - \boldsymbol{s}^{\mathrm{opt}}\|_2^2 \tag{31}$$

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

$$\lim_{l \to \infty} \bar{\boldsymbol{s}}_l = \bar{\boldsymbol{s}}. \tag{32}$$

Define

$$\bar{\boldsymbol{x}} \triangleq \frac{\mathrm{abs}_{\mathcal{K}}(\bar{\boldsymbol{s}}) + \bar{\boldsymbol{s}}}{2} \qquad \text{and} \qquad \bar{\boldsymbol{z}} \triangleq \frac{\mathrm{abs}_{\mathcal{K}}(\bar{\boldsymbol{s}}) - \bar{\boldsymbol{s}}}{2\mu}. \tag{33}$$

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:

$$\bar{\boldsymbol{z}} - \boldsymbol{c} \in \mathrm{range}\{\boldsymbol{A}^\top\} \tag{34a}$$
$$\boldsymbol{A}\bar{\boldsymbol{x}} = \boldsymbol{b} \tag{34b}$$
$$\bar{\boldsymbol{x}}^\top \bar{\boldsymbol{z}} = 0, \qquad \bar{\boldsymbol{x}} \in \mathcal{K}, \qquad \bar{\boldsymbol{z}} \in \mathcal{K}. \tag{34c}$$

Condition (34c) follows directly from the definition (33). Additionally, according to the second part of Lemma 3,

$$\boldsymbol{0}_n = \lim_{l \to \infty} \boldsymbol{s}^{l+1} - \boldsymbol{s}^l \tag{35a}$$
$$= \lim_{l \to \infty} \boldsymbol{d} - \frac{\boldsymbol{s}^l + |\boldsymbol{s}^l|_\star}{2} \tag{35b}$$
$$= \lim_{l \to \infty} \boldsymbol{d} - \frac{\bar{\boldsymbol{s}}^l + |\bar{\boldsymbol{s}}^l|_\star}{2} \overset{(32)}{=} \boldsymbol{d} - \frac{\bar{\boldsymbol{s}} + |\bar{\boldsymbol{s}}|_\star}{2} \tag{35c}$$

Hence,

$$\frac{\bar{\boldsymbol{s}} + |\bar{\boldsymbol{s}}|_\star}{2} = \boldsymbol{d} \overset{(23)}{=} \frac{\boldsymbol{s}^{\mathrm{opt}} + |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2} \tag{36}$$

Therefore,

$$\mathbf{0}_n = \frac{\bar{\boldsymbol{s}} + |\bar{\boldsymbol{s}}|_\star}{2} - \frac{\boldsymbol{s}^{\mathrm{opt}} + |\boldsymbol{s}^{\mathrm{opt}}|_\star}{2} \tag{37a}$$

$$= \frac{\bar{\boldsymbol{s}} - \boldsymbol{s}^{\mathrm{opt}} + (2\boldsymbol{A}^\dagger\boldsymbol{A} - \boldsymbol{I}_n)(\mathrm{abs}_{\mathcal{K}}(\bar{\boldsymbol{s}}) - \mathrm{abs}_{\mathcal{K}}(\boldsymbol{s}^{\mathrm{opt}}))}{2} \tag{37b}$$

$$= \frac{(\bar{\boldsymbol{x}} - \boldsymbol{x}^{\mathrm{opt}}) - \mu(\bar{\boldsymbol{z}} - \boldsymbol{z}^{\mathrm{opt}})}{2} + \frac{(2\boldsymbol{A}^\dagger\boldsymbol{A} - \boldsymbol{I}_n)[(\bar{\boldsymbol{x}} - \boldsymbol{x}^{\mathrm{opt}}) + \mu(\bar{\boldsymbol{z}} - \boldsymbol{z}^{\mathrm{opt}})]}{2} \tag{37c}$$

$$= \boldsymbol{A}^\dagger(\boldsymbol{A}\bar{\boldsymbol{x}} - \boldsymbol{A}\boldsymbol{x}^{\mathrm{opt}}) - \mu(\boldsymbol{I}_n - \boldsymbol{A}^\dagger\boldsymbol{A})(\bar{\boldsymbol{z}} - \boldsymbol{z}^{\mathrm{opt}}). \tag{37d}$$

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

$$\boldsymbol{A}\bar{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}^{\mathrm{opt}} = \boldsymbol{b}. \tag{38}$$

Similarly,

$$\mathbf{0}_n = \frac{\bar{\boldsymbol{s}} + |\bar{\boldsymbol{s}}|_\star}{2} - \boldsymbol{d} \tag{39a}$$

$$= \boldsymbol{A}^\dagger(\boldsymbol{A}\bar{\boldsymbol{x}} - \boldsymbol{b}) - \mu(\boldsymbol{I}_n - \boldsymbol{A}^\dagger\boldsymbol{A})(\bar{\boldsymbol{z}} - \boldsymbol{c}) \tag{39b}$$

$$= \mu(\boldsymbol{I}_n - \boldsymbol{A}^\dagger\boldsymbol{A})(\boldsymbol{c} - \bar{\boldsymbol{z}}) \tag{39c}$$

which concludes (34a). 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:

$$\lim_{l\to\infty} \|\boldsymbol{s}^l - \bar{\boldsymbol{s}}\|_2 = \lim_{l\to\infty} \|\bar{\boldsymbol{s}}^l - \bar{\boldsymbol{s}}\|_2 = 0 \tag{40}$$

which completes the proof. □