Journal of Optimization Theory and Applications (2022) 192:292–320
https://doi.org/10.1007/s10957-021-01971-3

Efficient Semidefinite Programming with Approximate ADMM

Nikitas Rontsis[1] · Paul Goulart[1] · Yuji Nakatsukasa[2]

Received: 23 July 2020 / Accepted: 31 October 2021 / Published online: 27 November 2021

Abstract
Tenfold improvements in computation speed can be brought to the alternating direction method of multipliers (ADMM) for Semidefinite Programming with virtually no decrease in robustness and provable convergence simply by projecting approximately to the Semidefinite cone. Instead of computing the projections via "exact" eigendecompositions that scale cubically with the matrix size and cannot be warm-started, we suggest using state-of-the-art factorization-free, approximate eigensolvers, thus achieving almost quadratic scaling and the crucial ability of warm-starting. Using a recent result from Goulart et al. (Linear Algebra Appl 594:177–192, 2020. https://doi.org/10.1016/j.laa.2020.02.014), we are able to circumvent the numerical instability of the eigendecomposition and thus maintain tight control on the projection accuracy. This in turn guarantees convergence, either to a solution or a certificate of infeasibility, of the ADMM algorithm. To achieve this, we extend recent results from Banjac et al. (J Optim Theory Appl 183(2):490–519, 2019. https://doi.org/10.1007/s10957-019-01575-y) to prove that reliable infeasibility detection can be performed with ADMM even in the presence of approximation errors. In all of the considered problems of SDPLIB that "exact" ADMM can solve in a few thousand iterations, our approach brings a significant, up to 20x, speedup without a noticeable increase in ADMM's iterations.

Keywords Semidefinite programming · Iterative eigensolvers · ADMM

Communicated by Amir Beck.

✉ Nikitas Rontsis
nrontsis@gmail.com

Paul Goulart
paul.goulart@eng.ox.ac.uk

Yuji Nakatsukasa
yuji.nakatsukasa@maths.ox.ac.uk

1 Department of Engineering Science, University of Oxford, Oxford, UK

2 Mathematical Institute, University of Oxford, Oxford, UK

Mathematics Subject Classification 90C22 · 65F15

1 Introduction

Semidefinite Programming is of central importance in many scientific fields. Areas as diverse as kernel-based learning [20], dimensionality reduction [9] analysis and synthesis of state feedback policies of linear dynamical systems [6], sum of squares programming [33], optimal power flow problems [21] and fluid mechanics [15] rely on Semidefinite Programming as a crucial enabling technology.

The wide adoption of Semidefinite Programming was facilitated by reliable algorithms that can solve semidefinite problems with polynomial worst-case complexity [6]. For small to medium-sized problems, it is widely accepted that primal–dual Interior Point methods are efficient and robust and are therefore often the method of choice. Several open-source solvers, like `SDPT3` [40] and `SDPA` [42], as well as the commercial solver `MOSEK` [25] exist that follow this approach. However, the limitations of interior point methods become evident in large-scale problems, since each iteration requires factorizations of large Hessian matrices. First-order methods avoid this bottleneck and thereby scale better to large problems, with the ability to provide modest-accuracy solutions for many large-scale problems of practical interest.

We will focus on the Alternating Directions Method of Multipliers (ADMM), a popular first-order algorithm that has been the method of choice for several popular optimization solvers both for Semidefinite Programming [12,28,43] and other types of convex optimization problems such as Quadratic Programming (QP) [38]. Following an initial factorization of an $m \times m$ matrix, every iteration of ADMM entails the solution of a linear system via forward/backward substitution and a projection to the Semidefinite Cone. For SDPs, this projection operation typically takes the majority of the solution time, sometimes 90% or more. Thus, reducing the per-iteration time of ADMM is directly linked to computing conic projections in a time-efficient manner.

The projection of a symmetric matrix $n \times n$ matrix A to the Semidefinite Cone is defined as

$$\Pi_{\mathbb{S}_+}(A) := \arg\min_X \|A - X\|_F$$

and can be computed in "closed form" as a function of the eigendecomposition of X. Indeed, assuming

$$\begin{bmatrix} V_+ & V_- \end{bmatrix} \begin{bmatrix} \Lambda_+ & \\ & \Lambda_- \end{bmatrix} \begin{bmatrix} V_+ & V_- \end{bmatrix}^T := X \tag{1}$$

where V_+ (respectively, V_-) is an orthonormal matrix containing the positive (nonpositive) eigenvectors, and Λ_+ (Λ_-) is a diagonal matrix that contains the respective positive (nonnegative) eigenvalues of A, then

$$\Pi_{\mathbb{S}_+}(A) = V_+ \Lambda_+ V_+^T = A - V_- \Lambda_- V_-^T. \tag{2}$$

The computation of $\Pi_{\mathbb{S}_+}$ therefore entails the (partial) eigendecomposition of A followed by a scaled matrix–matrix product.

The majority of optimization solvers, e.g., `SCS` [28] and `COSMO.jl` [12], calculate $\Pi_{\mathbb{S}_+}$ by computing the full eigendecomposition using `LAPACK`'s `syevr` routine.[1] There are two important limitations associated with computing full eigendecompositions. Namely, eigendecomposition has cubic complexity with respect to the matrix size n [14, §8], and it cannot be warm started. This has prompted research on methods for the approximate computation of a few eigenpairs in an iterative fashion [10,31,35], and the associated development of relevant software tools such as the widely used `ARPACK` [22], and the more recent `BLOPEX` [19] and `PRIMME` [37]. The reader can find surveys of relevant software in [17] and [37, §2]

However, the use of iterative eigensolvers in the Semidefinite optimization community has been very limited. To the best of our knowledge, the use of approximate eigensolvers has been limited in widely available ADMM implementations. In a related work, [24] considered the use of polynomial subspace extraction to avoid expensive eigenvalue decompositions required by first order-methods (including ADMM) and showed improved performance in problems of low-rank structure, while still maintaining convergence guarantees. In the wider area of first-order methods, [41, §3.1] considered `ARPACK` but disregarded it on the basis that it does not allow efficient warm starting, suggesting that it should only be used when the problem is known a priori to have low rank. Wen's suggestion of using `ARPACK` for SDPs whose solution is expected to be low rank has been demonstrated recently by [36]. At every iteration, [36] uses `ARPACK` to compute the r largest eigenvalues/vectors and then uses the approximate projection $\tilde{\Pi}(A) = \sum_{i=1}^{r} \max(\lambda_i, 0) v_i v_i^T$. The projection error can then be bounded by

$$\left\| \Pi(A) - \tilde{\Pi}(A) \right\|_F^2 = \left\| \sum_{i=r+1}^{n} \max(\lambda_i, 0) v_i v_i^T \right\|_F^2 \leq (n-r) \max(\lambda_r, 0)^2.$$

The parameter r is chosen such that it decreases with increasing iteration count so that the projection errors are summable. The summability of the projection errors is important, as it has been shown to ensure convergence of averaged non-expansive operators [4, Proposition 5.34] and for ADMM in particular [11, Theorem 8].

However, the analysis of [36] depends on the assumption that the iterative eigensolver will indeed compute the r largest eigenpairs "exactly." This is both practically and theoretically problematic; the computation of eigenvectors is numerically unstable since it depends inverse-proportionally on the spectral gap (defined as the distance between the corresponding eigenvalue and its nearest neighboring eigenvalue; refer to Sect. 5.2 for a concise definition), and therefore, no useful bounds can be given when repeated eigenvalues exist.

In contrast, our approach relies on a novel bound that characterizes the projection accuracy independently of the spectral gaps, depending only on the residual norms. The derived bounds do not require that the eigenpairs have been computed "exactly,"

[1] Detailed in https://software.intel.com/mkl-developer-reference-c-syevr.

but hold for *any* set of approximate eigenpairs obtained via the Rayleigh–Ritz process. This allows us to compute the eigenpairs with a relatively loose tolerance while still retaining convergence guarantees. Furthermore, unlike [36], our approach has the ability of warm-starting of the eigensolver, which typically results in improving computational efficiency.

On the theoretical side, we extend recent results regarding the detection of primal or dual infeasibility. It is well known that if an SDP problem is infeasible, then the iterates of ADMM will diverge [11]. This is true even when the iterates of ADMM are computed approximately with summable approximation errors. Hence, infeasibility can be detected in principle by stopping the ADMM algorithm when the iterates exceed a certain bound. However, this is unreliable both in practice, because it depends on the choice of the bound, and in theory, because it does not provide certificates of infeasibility [8]. Recently, [3] has shown that the successive differences of ADMM's iterates, which always converge regardless of feasibility, can be used to reliably detect infeasibility and construct infeasibility certificates. This approach has been used successfully in the optimization solver `OSQP` [38]. We extend Banjac's results to show that they hold even when ADMM's iterates are computed approximately, under the assumption that the approximation errors are summable.

Notation used: Let $\mathcal{H}$ denote a real Hilbert space equipped with an inner-product-induced norm $\|\cdot\| = \sqrt{\langle \cdot, \cdot \rangle}$ and $\operatorname{Cont}(\mathcal{D})$ the set of nonexpansive operators in $\mathcal{D} \subseteq \mathcal{H}$. $\operatorname{cl} \mathcal{D}$ denotes the closure of $\mathcal{D}$, $\operatorname{conv} \mathcal{D}$ the convex hull of $\mathcal{D}$, and $\mathcal{R}(T)$ the range of T. Id denotes the identity operator on $\mathcal{H}$, while I denotes an identity matrix of appropriate dimensions. For any scalar, nonnegative ϵ, let $x \approx_\epsilon y$ denote the following relation between x and y: $\|x - y\| \leq \epsilon$. $\mathbb{S}_+$ denotes the set of positive semidefinite matrices with a dimension that will be obvious from the context. Finally, define $\Pi_\mathcal{C}$ the projection, $\mathcal{C}^\infty$ the recession cone, and $S_\mathcal{C}$ the support function associated with a set $\mathcal{C}$.

2 Approximate ADMM

Although the focus of this paper is on Semidefinite Programming, our analysis holds for more general convex optimization problems that allow for combinations of semidefinite Problems, Linear Programs (LPs), Quadratic Programs (QPs), Second-Order Cone Programs (SOCPs) among others.[2] In particular, the problem form we consider is defined as

$$\begin{aligned} \text{minimize} \;\; & \tfrac{1}{2}x^T P x + q^T x \\ \text{subject to} \;\; & Ax = z \\ & z \in \mathcal{C}, \end{aligned} \tag{$\mathcal{P}$}$$

[2] Note that any problem of the form ($\mathcal{P}$) can be converted to an SDP by noting that the positive orthant and the second order cone can be expressed as a semidefinite cone, and by considering the epigraph form of ($\mathcal{P}$) [8, §4.1.3].

where $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^m$ are the decision variables, $P \in \mathbb{S}^n_+$, $q \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$ and $\mathcal{C}$ is a translated composition of the positive orthant, second order and/or semidefinite cones.

We suggest solving ($\mathcal{P}$), i.e., finding a solution $(\bar{x}, \bar{z}, \bar{y})$ where $\bar{y}$ is a Lagrange multiplier for the equality constraint of ($\mathcal{P}$), with the approximate version of ADMM described in Algorithm 1. As is common in the case in ADMM methods, our Algorithm

Algorithm 1: Solving ($\mathcal{P}$) with approximate ADMM

```
1 given initial values $x^0, y^0, z^0$, parameters $\rho > 0, \sigma > 0, \alpha \in (0, 2)$, the summable sequences
  $(\mu^k)_{k\in\mathbb{N}}, (\nu^k)_{k\in\mathbb{N}}$ and $x \approx_\epsilon y$ denoting that the vectors $x, y$ satisfy $\|x - y\| \le \epsilon$; for $k = 0, \ldots$
  until convergence do
2     $\begin{bmatrix}\tilde{x}^{k+1}\\ \tilde{z}^{k+1}\end{bmatrix} \approx_{\mu^k} \begin{bmatrix}P+\sigma I & \rho A^T\\ \rho A & -\rho I\end{bmatrix} \Big\backslash \left(\begin{bmatrix}\sigma I & \rho A^T\\ 0 & 0\end{bmatrix}\begin{bmatrix}x^k\\ z^k - y^k/\rho\end{bmatrix} - \begin{bmatrix}q\\ 0\end{bmatrix}\right)$
      $x^{k+1} = \alpha\tilde{x}^{k+1} + (1-\alpha)x^k \quad z^{k+1} \approx_{\nu^k} \Pi_{\mathcal{C}}(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k + y^k/\rho)$
      $y^{k+1} = y^k + \rho(\alpha\tilde{x}^{k+1} + (1-\alpha)z^k - z^{k+1})$
3 end
```

consists of repeated solutions of linear systems (line 2) and projections to $\mathcal{C}$ (line 4). These steps are the primary drivers of efficiency of ADMM and are typically computed to machine precision via matrix factorizations. Indeed, Algorithm 1 was first introduced by [38] and [3] in the absence of approximation errors. However, "exact" computations can be prohibitively expensive for large problems (and indeed impossible in finite-precision arithmetic), and the practitioner may have to rely on approximate methods for their computation. For example, [7, §4.3] suggests using the Conjugate Gradient method for approximately solving the linear systems embedded in ADMM. In Sect. 5, we suggest specific methods for the approximation computation of ADMM steps with a focus in the operation of line 4. Before moving into particular methods, we first discuss the convergence properties of Algorithm 1.

Our analysis explicitly accounts for approximation errors and provides convergence guarantees, either to solutions or certificates of infeasibility, in their presence. In general when ADMM's steps are computed approximately, ADMM might lose its convergence properties. Indeed, when the approximation errors are not controlled appropriately, the Fejér monotonicity [4] of the iterates and any convergence rates of ADMM can be lost. In the worst case, the iterates could diverge. However, the following Theorem, which constitutes the main theoretical result of this paper, shows that Algorithm 1 converges either to a solution or to a certificate of infeasibility of ($\mathcal{P}$) due to the requirement that the approximation errors are summable across the Algorithm's iterations.

Theorem 2.1 *Consider the iterates x^k, z^k, and y^k of Algorithm 1. If a KKT point exists for ($\mathcal{P}$), then (x^k, z^k, y^k) converges to a KKT point, i.e., a solution of ($\mathcal{P}$), when $k \mapsto \infty$. Otherwise, the successive differences*

$$\delta x := \lim_{k\to\infty} x^{k+1} - x^k, \quad \textit{and} \quad \delta y := \lim_{k\to\infty} y^{k+1} - y^k.$$

still converge and can be used to detect infeasibility as follows:

(i) If $\delta y \neq 0$*, then* ($\mathcal{P}$) *is primal infeasible and* δy *is a certificate of primal infeasibility* [3, Proposition 3.1] *in that it satisfies*

$$A^T \delta y = 0 \quad \textit{and} \quad S_{\mathcal{C}}(\delta y) < 0. \tag{3}$$

(ii) If $\delta x \neq 0$*, then* ($\mathcal{P}$) *is dual infeasible and* δx *is a certificate of dual infeasibility [3, Proposition 3.1] in that it satisfies*

$$P\delta x = 0, \quad A\delta x \in \mathcal{C}^\infty, \quad \textit{and} \quad q^T \delta x < 0. \tag{4}$$

(iii) If both $\delta x \neq 0$ *and* $\delta y \neq 0$*, then* ($\mathcal{P}$) *is both primal and dual infeasible and* $(\delta x, \delta y)$ *are certificates of primal and dual infeasibility as above.*

In order to prove Theorem 2.1, we must first discuss some key properties of ADMM. This will provide the theoretical background that will allow us to present the proof in Sect. 4. Then, in Sect. 5 we will discuss particular methods for the approximate computation of ADMM's steps that can lead to significant speedups.

3 The Asymptotic Behavior of Approximate ADMM

In this section, we present ADMM in a general setting, express it as an iteration over an averaged operator and then consider its convergence when this operator is computed only approximately.

ADMM is used to solve split optimization problems of the following form

$$\begin{aligned} &\text{minimize} \;\; f(\chi) + g(\psi) \\ &\text{subject to} \;\; \chi = \psi \end{aligned} \tag{$\mathcal{S}$}$$

where χ, ψ denote the decision variables on $\mathbb{R}^\ell$ which is equipped with an inner product induced norm $\|\cdot\| = \langle\cdot,\cdot\rangle$. The functions $f : \mathbb{R}^\ell \to [-\infty, +\infty]$, $g : \mathbb{R}^\ell \to [-\infty, +\infty]$ are proper, lower-semicontinuous, and convex.

ADMM works by alternately minimizing the augmented Lagrangian of ($\mathcal{S}$), defined as[3]

$$L(\chi, \psi, \omega) := f(\chi) + g(\psi) + \langle \omega, \chi - \psi \rangle + \frac{1}{2}\|\chi - \psi\|^2 \tag{5}$$

over χ and ψ. That is, ADMM consists of the following iterations

[3] Note that, following [3], this definition can account for the penalty parameters ρ and σ of Algorithm 1 via an appropriate definition for $\|\cdot\|$, as done later in (13).

$$\chi^{k+1} = \arg\min_{\chi} L(\chi, \psi^k, \omega^k) \tag{ADMM$_1$}$$

$$\psi^{k+1} = \arg\min_{\psi} L(\bar{\chi}^{k+1}, \psi, \omega^k) \tag{ADMM$_2$}$$

$$\omega^{k+1} = \omega^k + (\bar{\chi}^{k+1} - \psi^{k+1}) \tag{ADMM$_3$}$$

where $\bar{\chi}^{k+1}$ is a relaxation of χ^{k+1} with $\bar{\chi}^{k+1} = \alpha\chi^{k+1} + (1-\alpha)\psi^k$ for some relaxation parameter $\alpha \in (0, 2)$.

Although (ADMM$_1$)–(ADMM$_3$) are useful for implementing ADMM, theoretical analyses of the algorithm typically consider ADMM as an iteration over an averaged operator. To express ADMM in operator form, note that (ADMM$_1$) and (ADMM$_2$) can be expressed in terms of the *proximity operator* [4, §24]

$$\mathrm{prox}_f(\phi) := \arg\min_{\chi} \left(f(\chi) + \frac{1}{2}\|\chi - \phi\|^2 \right), \tag{6}$$

and the similarly defined prox_g, as

$$\chi^{k+1} = \mathrm{prox}_f(\psi^k - \omega^k), \quad \psi^{k+1} = \mathrm{prox}_g(\bar{\chi}^{k+1} + \omega^k),$$

respectively. Now, using the *reflections* of prox_f and prox_g, i.e., $R_f := 2\mathrm{prox}_f - \mathrm{Id}$ and $R_g := 2\mathrm{prox}_g - \mathrm{Id}$, we can express ADMM as an iteration over the $\frac{1}{2}\alpha$-averaged operator

$$T := \left(1 - \frac{1}{2}\alpha\right)\mathrm{Id} + \frac{1}{2}\alpha R_f R_g \tag{7}$$

on the variable $\phi^k := \bar{\chi}^k + \omega^{k-1}$ (see [34, §3.A] or [13, Appendix B] for details). The variables ψ, χ, ω of (ADMM$_1$)–(ADMM$_3$) can then be obtained from ϕ as

$$\chi^{k+1} = \mathrm{prox}_f R_g \phi^k, \quad \psi^k = \mathrm{prox}_g \phi^k, \quad \text{and} \quad \omega^k = (\mathrm{Id} - \mathrm{prox}_g)\phi^k. \tag{8}$$

We are interested in the convergence properties of ADMM when the operators prox_f, prox_g, and thus T, are computed inexactly. In particular, we suppose that the iterates are generated as

$$(\forall k \in \mathbb{N}) \quad \phi^{k+1} = \left(1 - \frac{1}{2}\alpha\right)\phi_k + \frac{1}{2}\alpha\left(R_f\left(R_g\phi^k + \epsilon_g^k\right) + \epsilon_f^k\right) \tag{9}$$

for some error sequences $\epsilon_f^k, \epsilon_g^k \in \mathbb{R}^\ell$. Our convergence results will depend on the assumption that $\left\|\epsilon_f^k\right\|$ and $\left\|\epsilon_g^k\right\|$ are summable. This implies that ϕ^k can be considered as an approximate iteration over T, i.e.,

$$\phi^{k+1} \approx_{\epsilon^k} T\phi^k, \tag{10}$$

for some summable error sequence (ϵ^k). Indeed, since R_g and R_f are nonexpansive, we have

$$\begin{aligned} \left\| R_f\left(R_g\phi^k + \epsilon_g^k\right) + \epsilon_f^k - R_f R_g \phi^k \right\| &\le \left\| R_g\phi^k + \epsilon_g^k - R_g\phi^k \right\| + \left\| \epsilon_f^k \right\| \\ &\le \left\| \epsilon_f^k \right\| + \left\| \epsilon_g^k \right\|, \end{aligned}$$

or $\left\|\phi^{k+1} - T\phi^k\right\| \le \alpha \left\|\epsilon_f^k\right\|/2 + \alpha\left\|\epsilon_g^k\right\|/2$, from which the summability of $\left\|\epsilon^k\right\|$ follows.

It is well known that, when $\left\|\epsilon_f^k\right\|, \left\|\epsilon_g^k\right\|$ are summable, (9) converges to a solution of ($\mathcal{S}$), obtained by ϕ according to (8), provided that ($\mathcal{S}$) has a KKT point [11, Theorem 8]. We will show that, under the summability assumption, $\delta\phi = \lim \phi^{k+1} - \phi^k$ always converges, *regardless* of whether ($\mathcal{S}$) has a KKT point:

Theorem 3.1 *The successive differences* $lim_{k\to\infty}(\phi^{k+1} - \phi^k)$ *of* (9) *converge to the unique minimum-norm element of* $\operatorname{cl}\mathcal{R}(\mathrm{Id} - T)$ *provided that* $\sum\left\|\epsilon_f^k\right\| < \infty$ *and* $\sum\left\|\epsilon_g^k\right\| < \infty$.

Proof This is a special case of Proposition A.1 of Appendix 1. □

Theorem 3.1 will prove useful in detecting infeasibility, as we will show in the following section.

4 Proof of Theorem 2.1

We now turn our attention to proving Theorem 2.1. To this end, note that ($\mathcal{P}$) can be regarded as a special case of ($\mathcal{S}$) [3,38]. This becomes clear if we set $\chi = (\tilde{x}, \tilde{z})$, $\psi = (x, z)$, and define

$$f(\tilde{x}, \tilde{z}) := \frac{1}{2}\tilde{x}^T P \tilde{x} + q^T \tilde{x} + \mathcal{I}_{A\tilde{x}=\tilde{z}}(\tilde{x}, \tilde{z}), \tag{11}$$

$$g(x, z) := \mathcal{I}_{\mathcal{C}}(z), \tag{12}$$

where $\mathcal{I}_{\mathcal{C}}(z)$ denotes the indicator function of $\mathcal{C}$. Furthermore, using the analysis of the previous section and defining the norm

$$\|(x, z)\| = \sqrt{\sigma\,\|x\|_2^2 + \rho\,\|z\|_2^2}. \tag{13}$$

we find that Algorithm 1 is equivalent to iteration (9).

First, we show that if ($\mathcal{P}$) has a KKT point, then Algorithm 1 converges to its primal–dual solution. Due to (11)–(13), every KKT point $(\bar{x}, \bar{z}, \bar{y})$ of ($\mathcal{P}$) produces a KKT point

$$(\bar{\chi}, \bar{\psi}, \bar{\omega}) = ((\bar{x}, \bar{z}), (\bar{x}, \bar{z}), (0, \bar{y}/\rho)) \tag{14}$$

for ($\mathcal{S}$). Likewise, every KKT point of ($\mathcal{S}$) is in the form of (14) (right) and gives a KKT point $(\bar{x}, \bar{z}, \bar{y})$ for ($\mathcal{P}$). Thus, according to [11, Theorem 8], Algorithm 1 converges to a KKT point of ($\mathcal{P}$), assuming that a KKT point exists.

It remains to show points $(i)-(iii)$ of Theorem 2.1. These are a direct consequence of [3, Theorem 5.1] and the following proposition:

Proposition 4.1 *The following limits*

$$\delta x := \lim_{k\to\infty} x^{k+1} - x^k, \quad \delta y := \lim_{k\to\infty} y^{k+1} - y^k,$$

defined by the iterates of Algorithm 1, converge to the respective limits defined by the iterates of Algorithm 1 with $\mu^k = \nu^k = 0\, \forall k \in \mathbb{N}$.

Proof According to [34, §3.A], we can rewrite Algorithm 1 as follows

$$z^k \approx_{\nu^{k-1}} \Pi_{\mathcal{C}}(\upsilon^k) \tag{15a}$$

$$(\tilde{x}^{k+1}, \tilde{z}^{k+1}) \approx_{\mu^k} \operatorname{prox}_f((x^k, 2z^k - \upsilon^k)) \tag{15b}$$

$$x^{k+1} = x^k - \alpha(\tilde{x}^{k+1} - x^k) \tag{15c}$$

$$\upsilon^{k+1} = \upsilon^k + \alpha(\tilde{z}^{k+1} - z^k) \tag{15d}$$

where $(x^k, \upsilon^k) := \phi^k$ and y^k can be obtained as $y^k = \rho(\upsilon^k - z^k)$.

Define $\delta x^k := x^{k+1} - x^k$, $\forall k \in \mathbb{N}$ and $\delta z^k, \delta\upsilon^k, \delta\tilde{x}^k, \delta\tilde{z}^k$ in a similar manner. Due to Theorem 3.1 and [3, Lemma 5.1], we conclude that $\lim_{k\to\infty} \delta x^k$ and $\lim_{k\to\infty} \delta\upsilon^k$, defined by the iterates of Algorithm 1, converge to the respective limits defined by the iterates of Algorithm 1 with $\mu^k = \nu^k = 0\, \forall k \in \mathbb{N}$.

To show the same result for δy, first recall that $y^k = \rho(\upsilon^k - z^k)$. It then suffices to show the desired result for $\lim_{k\to\infty} \delta z^k$. We show this using arguments similar to [3, Proposition 5.1 (iv)]. Indeed, note that due to (15c)–(15d) we have

$$-(\delta x^{k+1} - \delta x^k)/\alpha = \delta x^k - \delta\tilde{x}^{k+1}$$
$$-(\delta\upsilon^{k+1} - \delta\upsilon^k)/\alpha = \delta z^k - \delta\tilde{z}^{k+1}$$

and thus, $\lim_{k\to\infty} \delta x^k = \lim_{k\to\infty} \delta\tilde{x}^k$ and $\lim_{k\to\infty} \delta z^k = \lim_{k\to\infty} \delta\tilde{z}^k$. Furthermore, due to (11) we have $A\tilde{x}^{k+1} - \tilde{z}^{k+1} = e^k$ for some sequence (e^k) with summable norms, thus

$$\lim_{k\to\infty} \delta\tilde{z}^k = A \lim_{k\to\infty} \delta\tilde{x}^k = A \lim_{k\to\infty} \delta x^k$$

and the claim follows due to [3, Proposition 5.1 (i) and (iv)]. □

5 Krylov-Subspace Methods for ADMM

In this Section, we suggest suitable methods for calculating the individual steps of Algorithm 1. We will focus on Semidefinite Programming, i.e., when $\mathcal{C}$ is the semidefinite cone. After an initial presentation of state-of-the-art methods used for solving linear systems approximately, we will describe (in Sect. 5.1) LOBPCG, the suggested method for projecting onto the semidefinite cone. Note that some of our presentation recalls established linear algebra techniques that we include for the sake of completeness.

We begin with a discussion of the Conjugate Gradient method, a widely used method for the solution of the linear systems embedded in Algorithm 1. Through CG's presentation, we will introduce the Krylov Subspace which is a critical component of LOBPCG. Finally, we will show how we can assure that the approximation errors are summable across ADMM iterations, thus guaranteeing convergence of the algorithm.

The linear systems embedded in Algorithm 1 are in the following form

$$\underbrace{\begin{bmatrix} P+\sigma I & \rho A^T \\ \rho A & -\rho I \end{bmatrix}}_{:=Q} \begin{bmatrix} \tilde{x}^{k+1} \\ \tilde{z}^{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} \sigma I & \rho A^T \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x^k \\ z^k - y^k/\rho \end{bmatrix} - \begin{bmatrix} q \\ 0 \end{bmatrix}}_{:=b^k}. \tag{16}$$

The linear system (16) belongs to the widely studied class of symmetric quasidefinite systems [5,29]. Standard scientific software packages, such as the Intel Math Kernel Library and the Pardiso Linear Solver, implement methods that can solve (16) approximately. Since the approximate solution (16) can be considered standard in the Linear Algebra community, we will only discuss the popular class of Krylov Subspace methods, which includes the celebrated Conjugate Gradient method.[4] Although CG has been used in ADMM extensively [§4.3.4] [7,28], its presentation will be useful for introducing some basic concepts that are shared with the main focus of this section, i.e., the approximate projection to the semidefinite cone.

From an optimization perspective, Krylov subspace algorithms for solving linear systems can be considered an improvement of gradient methods. Indeed, solving $Ax = b$, where $A \in \mathbb{S}^n_{++}$ via gradient descent on the objective function $c(x) := \frac{1}{2}x^T Ax - x^T b$ amounts to the following iteration

$$(\forall k \in \mathbb{N}) \quad x^{k+1} = x^k - \beta^k \nabla c(x) = x^k - \beta^k \underbrace{(Ax^k - b)}_{:=r^k} \tag{17}$$

where β^k is the step size at iteration k. Note that

$$x^{k+1} \in x_0 + \underbrace{\operatorname{span}(r_0, Ar_0, \dots A^k r_0)}_{:=\mathcal{K}_k(A,r^0)},$$

[4] The Conjugate Gradient Method is only suitable for Positive Definite Linear Systems. However, (16) can be solved with CG via a variable reduction which yields a smaller positive linear system [29, §1].

where $\mathcal{K}_k(A, r_0)$ is known as the *Krylov Subspace*. As a result, the following algorithm

$$(\forall k \in \mathbb{N}) \quad x^{k+1} = \arg\min_{x \in x^0 + \mathcal{K}_k(A, r^0)} \frac{1}{2} x^T A x - x^T b \tag{CG}$$

is guaranteed to yield results that are no worse than gradient descent. What is remarkable is that (CG) can be implemented efficiently in the form of two-term recurrences, resulting in the *Conjugate Gradient* (CG) Algorithm [14, §11.3].

We now turn our attention to the projection to the Semidefinite cone, which we have already defined in the introduction. Recalling (2), we note that the projection to the semidefinite cone can be computed via either the positive or the negative eigenpairs of A. As we will see, the cost of approximating eigenpairs of a matrix depends on their cardinality, thus computing $\Pi_{\mathbb{S}_+}(A)$ with the positive eigenpairs of A is preferable when A has mostly nonpositive eigenvalues, and vice versa. In the following discussion, we will focus on methods that compute the positive eigenpairs of A, thus assuming that A has mostly nonpositive eigenvalues. The opposite case can be easily handled by considering $-A$.

Similarly to CG, the class of Krylov Subspace methods is very popular for the computation of "extreme" eigenvectors of an $n \times n$ symmetric matrix A and can be considered as an improvement to gradient methods. In the subsequent analysis, we will make frequent use of the real eigenvalues of A, which we denote with $\lambda_1 \geq \cdots \geq \lambda_n$ and a set of corresponding orthogonal eigenvectors $\upsilon_1, \ldots \upsilon_n$. The objective to be maximized in this case is the Rayleigh Quotient,

$$r(x) := \frac{x^T A x}{x^T x}. \tag{18}$$

due to the fact that the maximum and the minimum values of $r(x)$ are λ_1 and λ_n, respectively, with υ_1 and υ_n as corresponding maximizers [14, Theorem 8.1.2]. Thus, we end up with the following gradient ascent iteration

$$\begin{aligned}(\forall k \in \mathbb{N}) \quad x^{k+1} &= \alpha^k x^k - \beta^k \nabla r(x^k) \\ &= \alpha^k x^k - 2\beta^k \left(A x^k - r(x^k) x^k \right)\end{aligned} \tag{19}$$

where the "stepsizes" α^k and β^k and the initial point x^0 are chosen so that all the iterates lie on the unit sphere. Although $r(x)$ is nonconvex, (19) can be shown to converge when appropriate stepsizes are used. For example, if we choose $\alpha_k = -2\beta^k r(x^k) \Rightarrow x^{k+1} \propto A x^k \ \forall k \in \mathbb{N}$, then (19) is simply the *Power Method*, which is known to converge linearly to an eigenvector associated with $\max |\lambda_i|$. Other stepsize choices can also assure convergence to an eigenvector associated with $\max \lambda_i$ [10, 11.3.4], [1, Theorem 3].

Algorithm 2: The Rayleigh-Ritz Procedure

```
1 given $A \in \mathbb{S}^n$ and an $n \times m$ thin matrix $S$ that spans the trial subspace orthonormalize $S$ $(\tilde{\Lambda}, \tilde{W}) \leftarrow$ Eigendecomposition of $S^T AS$ with $\tilde{\Lambda}_{(1,1)} \leq \cdots \leq \tilde{\Lambda}_{(m,m)}$ return the Ritz vectors $S\tilde{W}$ and Ritz values $\tilde{\Lambda}$ of $A$ on span$(S)$
```

Similarly to the gradient descent method for linear systems, the iterates of (19) lie in the Krylov subspace $\mathcal{K}_k(A, x_0)$. As a result, the following Algorithm

$$(\forall k \in \mathbb{N}) \quad x^{k+1} = \begin{array}{ll} \operatorname{argmax} & r(x) \\ \text{subject to} & x \in \mathcal{K}_k(A, x_0) \\ & \|x\|_2 = 1, \end{array} \tag{20}$$

is guaranteed to yield no worse results than any variant of (19) in finding an eigenvector associated with max λ_i, and in practice the difference is often remarkable. But how can the Rayleigh Quotient be maximized over a subspace? This can be achieved with the *Rayleigh–Ritz* Procedure, defined in Algorithm 2, which computes approximate eigenvalues/vectors (called *Ritz values/vectors*) that are restricted to lie on a certain subspace and are, under several notions, optimal [31, 11.4] (see discussion after Theorem 5.1). Indeed, every iterate x^{k+1} of (20) coincides with the last column of X^{k+1}, i.e., the largest Ritz vector, of the following Algorithm [31, Theorem 11.4.1]

$$(\forall k \in \mathbb{N}) \quad (\Lambda^{k+1}, X^{k+1}) = \text{Rayleigh Ritz of } A \text{ on the trial subspace } [x^0 \; Ax^0 \; \ldots \; A^k x^0]. \tag{21}$$

Note that unlike (20), Algorithm (21) provides approximations to not only one, but k eigenpairs, with the extremum ones exhibiting a faster rate of convergence.

Remarkably, similarly to the Conjugate Gradient algorithm, (21) and (20) also admit an efficient implementation, in the form of three-term recurrences known as the *Lanczos Algorithm* [14, §10.1]. In fact, the Lanczos Algorithm produces a sequence of orthonormal vectors that tridiagonalize A. Given this sequence of vectors, the computation of the associated Ritz pairs is inexpensive [14, 8.4]. The Lanczos Algorithm is usually the method of choice for computing a few extreme eigenpairs for a symmetric matrix. However, although the Lanczos Algorithm is computationally efficient, the Lanczos process can suffer from lack of orthogonality, with the issue becoming particularly obvious when a Ritz pair is close to converging to some (usually extremal) eigenpair [30]. Occasional re-orthogonalizations, with a cost of $O(n^2 l^k)$ where l^k is the dimension of the kth trial subspace, are required to mitigate the effects of the numerical instability. To avoid such a computational cost, the Krylov subspace is restarted or shrunk so that l^k, and thus, the computational costs of re-othogonalizations are bounded by an acceptable amount. The Lanczos Algorithm with occasional restarts is the approach employed by the popular eigensolver `ARPACK` [22] for symmetric matrices.

However, there are two limitations of the Lanczos Algorithm. Namely, it does not allow for efficient warm starting of multiple eigenvectors since its starting point is a single eigenvector, and it cannot detect the multiplicity of the approximated eigenvalues as it normally provides a single approximate eigenvector for every invariant subspace of A.

Block Lanczos addresses both of these issues. Similarly to the standard Lanczos Algorithm, Block Lanczos computes Ritz pairs on the trial block Krylov Subspace $\mathcal{K}_k(A, X_0) := \text{span}(X^0, AX^0, \dots, A^k X^0)$ where X_0 is an $n \times m$ matrix that contains a set of initial eigenvector guesses. Thus, Block Lanczos readily allows for the warm starting of multiple Ritz pairs. Furthermore, block methods handle clustered and multiple eigenvectors (of multiplicity up to m) well. However, these benefits come at the cost of higher computational costs, as the associated subspace is increased by m at every iteration. This, in turn, requires more frequent restarts, particularly for the case where m is comparable to n.

In our experiments, we observed that a single block iteration often provides Ritz pairs that give good enough projections for Algorithm 1. This remarkably good performance motivated us to use the *Locally Optimal Block Preconditioned Conjugate Gradient Method* (LOBPCG), presented in the following subsection.

5.1 LOBPCG: The Suggested Eigensolver

LOBPCG [19] is a block Krylov method that, after the first iteration, uses the trial subspace $\text{span}(X^k, AX^k, \Delta X^k)$, where $\Delta X^k := X^k - X^{k-1}$, and sets X^{k+1} to Ritz vectors corresponding to the m largest eigenvalues. Thus, the size of the trial subspace is fixed to $3m$. As a result, LOBPCG keeps its computational costs bounded and is particularly suitable for obtaining Ritz pairs of modest accuracy, as it not guaranteed to exhibit the super-linear convergence of Block Lanczos [10] which might only be observed after a large number of iterations. Algorithm 3 presents LOBPCG for computing the positive eigenpairs of a symmetric matrix.[5] Note that the original LOBPCG Algorithm [19, Algorithm 5.1] is more general in the sense that it allows for the solution of generalized eigenproblems and supports preconditioning. We do not discuss these features of LOBPCG as they are not directly relevant to Algorithm 1. On the other hand, [19] assumes that the number of desired eigenpairs is known a priori. However, this is not the case for $\Pi_{\mathbb{S}_+}$, where the computation of all *positive* eigenpairs is required.

In order to allow the computation of all the positive eigenpairs, X^k is expanded when more than m

positive eigenpairs are detected in the Rayleigh–Ritz Procedure in Line 6 of Algorithm 3. Note that the Rayleigh–Ritz Procedure produces $3m$ Ritz pairs, of which usually n are approximate eigenpairs, (or $2m$ in the first iteration of LOBPCG) and the number of positive Ritz values is always no more than the positive eigenvalues of A [31, 10.1.1], thus the subspace X^k must be expanded when more than m positive Ritz values are found.

[5] Note that Algorithm 3 performs Rayleigh–Ritz on the subspace spanned by $[X^k \; AX^k - X^k\Lambda^k \; \Delta X^k]$. Since Λ^k is diagonal, this is mathematically the same as using $[X^k \; AX^k \; \Delta X^k]$ but using $AX^k - X^k\Lambda^k$ improves the conditioning of the Algorithm.

It might appear compelling to expand the subspace to include *all* the positive Ritz pairs computed by Rayleigh–Ritz. However, this can lead to ill-conditioning, as we proceed to show. Indeed, consider the case where we perform LOBPCG starting from an initial matrix X^0. In the first iteration, Rayleigh–Ritz is performed on span(X^0, AX^0). Suppose that all the Rayleigh values are positive and we thus decide to include all of the Ritz vectors in X^1, setting $X^1 = [X^0\ AX^0]W$ for some nonsingular W. In the next iteration, we perform Rayleigh–Ritz on the subspace spanned by

$$\left[X^1\ AX^1\ \Delta X^1\right] = \left[X^0\ AX^0\ AX^0\ A^2X^0\ \Delta X^1\right] \begin{bmatrix} W & & \\ & W & \\ & & I \end{bmatrix}.$$

The problem is that the above matrix is rank deficient. Thus, one has to rely on a numerically stable Algorithm, like Householder QR, for its orthonormalization (required by the Rayleigh–Ritz Procedure) instead of the more efficient Cholesky QR algorithm [39, page 251]. Although, for this example, one can easily reduce columns from the matrix so that it becomes full column rank, the situation becomes more complicated when not all of the Rayleigh values are positive. In order to avoid this numerical instability, and thus be able to use Cholesky QR for othonormalizations, we expand X^k whenever necessary by a fixed size (equal to a small percentage of n (the size of A), e.g., $n/50$) with a set of randomly generated vectors. When is projecting to $\mathbb{S}_+$ with

Algorithm 3: The LOBPCG Algorithm for Computing the Positive Eigenpairs of a Symmetric Matrix

```
1 given $A \in \mathbb{S}^n$ and the $n \times m$ thin matrix $X^0$ that spans the initial trial subspace $(\Lambda^0, X^0) \leftarrow$
  Rayleigh–Ritz for $A$ on the trial subspace span$(X^0)$ $\Delta X^0 \leftarrow$ empty $n \times 0$ matrix;
2 for $k = 0, \ldots$ until convergence do
3     $R^k \leftarrow AX^k - X^k\Lambda^k$;
4     $(\Lambda^{k+1}, X^{k+1}) \leftarrow$ Apply Rayleigh–Ritz for $A$ on the trial subspace span$(X^k, R^k, \Delta X^k)$ and
      return the $m$ largest eigenpairs;
5     $\Delta X^{k+1} \leftarrow X^{k+1} - X^k$;
6     Expand $\Lambda^{k+1}, X^{k+1}$ with randomly generated elements and set
      $m = $ size$(X^{k+1}, 2) = $ size$(\Lambda^{k+1}, 2)$ if the positive Ritz values of line 6 were more than $m$.
7 end
8 return $X^k, \Lambda^k$ containing $m$ Ritz pairs that approximate the positive eigenpairs of $A$
```

LOBPCG most efficient? Recall that there exist two ways to project a matrix A into the semidefinite cone. The first is to compute all the positive eigenpairs Λ_+, V_+ of A and set $\Pi_{\mathbb{S}_+}(A) = V_+\Lambda_+V_+^T$. The opposite approach is to compute all the *negative* eigenpairs Λ_-, V_- of A and set $\Pi_{\mathbb{S}_+}(A) = I - V_-\Lambda_-V_-^T$. The per-iteration cost of LOBPCG is $O(n^2m)$ where m is the number of computed eigenpairs. Thus, when most of the eigenvalues are nonpositive, then the positive eigenpairs should be approximated, and vice versa.

As a result, LOBPCG is most efficient when the eigenvalues of the matrix under projection are either almost all nonnegative or almost all nonpositive, in which case

LOBPCG exhibits an almost quadratic complexity, instead of the cubic complexity of the full eigendecomposition. This is the case when ADMM converges to a low rank primal or dual solution of ($\mathcal{P}$). Fortunately, low rank solutions are often present or desirable in practical problems [23]. On the other hand, the worst case scenario is when half of the eigenpairs are nonpositive and half nonnegative, in which case LOBPCG exhibits worse complexity than the full eigendecomposition and thus, the latter should be preferred.

5.2 Error Analysis and Stopping Criteria

Algorithm 1 requires that the approximation errors in lines 3 and 5 are bounded by a summable sequence. As a result, bounds on the accuracy of the computed solutions are necessary to assess when the approximate algorithms (CG and LOBPCG) can be stopped.

For the approximate solution of the Linear System (16) one can easily devise such bounds. Indeed, note that the left hand matrix of (16) is fixed across iterations and is full rank. We can check if an approximate solution $[\bar{x}^{k+1};\ \bar{z}^{k+1}]$ satisfies the condition

$$\left\| \begin{bmatrix} \tilde{x}^{k+1} \\ \tilde{z}^{k+1} \end{bmatrix} - \begin{bmatrix} \bar{x}^{k+1} \\ \bar{z}^{k+1} \end{bmatrix} \right\|_2 \leq \mu^k \tag{22}$$

of Algorithm 1 easily, since (recalling Q is the KKT matrix defined in 16)

$$\left\| \begin{bmatrix} \tilde{x}^{k+1} \\ \tilde{z}^{k+1} \end{bmatrix} - \begin{bmatrix} \bar{x}^{k+1} \\ \bar{z}^{k+1} \end{bmatrix} \right\|_2 \leq \left\| Q^{-1} \right\| \underbrace{\left\| b^k - Q \begin{bmatrix} \bar{x}^{k+1} \\ \bar{z}^{k+1} \end{bmatrix} \right\|_2}_{:=r^k}. \tag{23}$$

Since $\left\| Q^{-1} \right\|$ is constant across iterations, it can be ignored when considering the summability of the approximation errors (22). Thus, we can terminate CG (or any other iterative linear system solver employed) when the *residual* r^k of the approximate solution $[\bar{x}^{k+1};\ \bar{z}^{k+1}]$ becomes less than a summable sequence, e.g., $1/k^2$.

On the other hand, controlling the accuracy of the projection to the Semidefinite Cone requires a closer examination. Recall that, given a symmetric matrix A that is to be projected,[6] our approach uses LOBCPG to compute a set of positive Ritz pairs $\tilde{V}$, $\tilde{\Lambda}$ approximating V_+, Λ_+ of (1) which we then use to approximate $\Pi_{\mathbb{S}_+^n}(A) = V_+ \Lambda_+ V_+^T$ as[7] $\tilde{V}\tilde{\Lambda}\tilde{V}^T$. A straightforward approach would be to quantify the projection's accuracy with respect to the accuracy of the Ritz pairs. Indeed, if we assume that our approximate positive eigenspace is "sufficiently rich" in the sense that $\lambda_{\max}(\tilde{V}_\perp A \tilde{V}_\perp) \leq 0$, then we get $m = \tilde{m}$ [31, Theorem 10.1.1], thus we can define $\Delta\Lambda = \Lambda_+ - \tilde{\Lambda}$, $\Delta V = V_+ - \tilde{V}$

6 Note that the matrices under projection depend on the iteration number of ADMM. We do not make this dependence explicit in order to keep the notation uncluttered.

7 When LOBPCG approximates the negative eigenspace (because the matrix under projection is believed to be almost positive definite), then all of the results of this section hold mutatis mutandis. Refer to for more details.

which then gives the following bound

$$\left\| V_+ \Lambda_+ V_+^T - \tilde{V} \tilde{\Lambda} \tilde{V}^T \right\| \leq 2 \left\| \Delta V \Lambda_+ V_+^T \right\| + \left\| V_+ \Delta \Lambda V_+^T \right\| + O(\|\Delta\|^2) \quad (24)$$

with $\|\Delta\| := \max(\|\Delta V\|, \|\Delta\Lambda\|)$. Standard results of eigenvalue perturbation theory can be used to bound the error in the computation of the eigenvalues, i.e., by $\|\Delta\Lambda\|_F^2 \leq 2\|R\|_F^2$ [31, Theorem 11.5.2][8] where

$$R := A\tilde{V} - \tilde{V}\tilde{\Lambda}.$$

In contrast, $\|\Delta V\|$ is ill-conditioned, as the eigenvectors are not uniquely defined in the presence of multiple (i.e., clustered) eigenvalues. At best, eigenvalue perturbation theory can give $\|\Delta V\| \lessapprox \|R\| / \text{gap}$ [26, Theorem 3.1 and Remark 3.1] where

$$\text{gap} := \min_{i,j} \left(\tilde{\Lambda}_{(i,i)} - \Lambda_{-(j,j)} \right).$$

This implies that the projection accuracy depends on the separation of the spectrum and can be very poor in the presence of small eigenvalues. Note that unlike R that is readily computable from $(\tilde{V}, \tilde{\Lambda})$, "gap" is, in general, unknown and non-trivial to compute, thus further complicating the analysis.

To overcome these issues, we employ a novel bound that shows that, although the accuracy of the Ritz pairs depends on the separation of eigenvalues, the approximate projection does not:

Theorem 5.1 *Assume that $\tilde{V}$ and $\tilde{\Lambda}$ are such that $\tilde{\Lambda} = \tilde{V}^T A \tilde{V}$. Then,*

$$\left\| \tilde{V}\tilde{\Lambda}\tilde{V}^T - \Pi_{\mathbb{S}_+}(A) \right\|_F^2 \leq 2\|R\|_F^2 + \left\| \Pi_{\mathbb{S}_+}(\tilde{V}_\perp^T A \tilde{V}_\perp) \right\|_F^2$$

Proof This is a restatement of [16, Corollary 2.1]. □

Note that the above result does not depend on the assumption that $\lambda_{\max}(\tilde{V}_\perp A \tilde{V}_\perp)$ is nonpositive or that $m = \tilde{m}$. Nevertheless, with a block Krylov subspace method it is often expected that $\lambda_{\max}(\tilde{V}_\perp A \tilde{V}_\perp)$ will be either small or negative, thus the bound of Theorem 5.1 will be dominated by $\|R\|$. The assumption $\tilde{\Lambda} = \tilde{V}^T A \tilde{V}$ is satisfied when $\tilde{V}$ and $\tilde{\Lambda}$ are generated with the Rayleigh–Ritz Procedure and thus holds for Algorithm 3. In fact, the use of the Rayleigh–Ritz, which is employed by Algorithm 3, is strongly suggested by Theorem 5.1 as it minimizes $\|R\|_F$ [31, Theorem 11.4.2].

We suggest terminating Algorithm 3 when every positive Ritz pair has a residual with norm bounded by a sequence that is summable across ADMM's iterations. Then, excluding the effect of $\left\| \Pi_{\mathbb{S}_+}(\tilde{V}_\perp^T A \tilde{V}_\perp) \right\|_F^2$, which appears to be negligible according to the results of the next section, Theorem 5.1 implies that the summability requirements of Algorithm 3 will be satisfied. Either way, the term $\left\| \Pi_{\mathbb{S}_+}(\tilde{V}_\perp^T A \tilde{V}_\perp) \right\|_F^2$ can

[8] Note that following [31, Theorem 11.5.1], $\lambda_{\max}(\tilde{V}_\perp A \tilde{V}_\perp) \leq 0$ implies that the indices of α can coincide with the indices of θ in [31, Theorem 11.5.2].

be bounded by $(n - \tilde{m})\lambda^2_{\max}(\tilde{V}_\perp A \tilde{V}_\perp)$, where $\lambda^2_{\max}(\tilde{V}_\perp A \tilde{V}_\perp)$ can be estimated with a projected Lanczos methods.

6 Experiments and Software

In this section, we provide numerical results for Semidefinite Programming with Algorithm 1, where the projection to the Semidefinite Cone is performed with Algorithm 3. Our implementation is essentially a modification of the optimization solver `COSMO.jl`. `COSMO.jl` is an open-source Julia implementation of Algorithm 1 which allows for the solution of problems in the form ($\mathcal{P}$) for which $\mathcal{C}$ is a composition of translated cones $\{\mathcal{K}_i + b_i\}$. Normally, `COSMO.jl` computes ADMM's steps to machine precision and supports any cone $\mathcal{K}_i$ for which a method to calculate its projection is provided.[9] `COSMO.jl` provides default implementations for various cones, including the Semidefinite cone, where `LAPACK`'s `syevr` function is used for its projection. The modified solver used in the experiments of this section can be found online:

https://github.com/nrontsis/ApproximateCOSMO.jl.

Code reproducing the results of this section is also publicly available.[10]

We compared the default version of `COSMO.jl` with a version where the operation `syevr` for the Semidefinite Cone is replaced with Algorithm 3. We have reimplemented `BLOPEX`, the original `MATLAB` implementation of LOBPCG [19], in Julia. For the purposes of simplicity, our implementation supports only symmetric standard eigenproblems without preconditioning. For these problems, our implementation was tested against `BLOPEX` to assure that exactly the same results (up to machine precision) are returned for identical problems. Furthermore, according to Sect. 5.1 we provide the option to compute all eigenvalues that are larger or smaller than a given bound.

At every iteration k of Algorithm 1, we compute approximate eigenpairs of every matrix that is to be projected onto the semidefinite cone. If, at the previous iteration of ADMM, a given matrix was estimated to have less than a third of its eigenvectors positive, then LOBPCG is used to compute its *positive* eigenpairs, according to (2) (middle). If it had less than a third of its eigenvectors negative, then LOBPCG computes its negative eigenpairs according to (2) (right). Otherwise, a full eigendecomposition is used.

In every case, LOBPCG is terminated when all of the Ritz pairs have a residual with norm less than $10/k^{1.01}$. According to Sect. 5.2, this implies that the projection errors are summable across ADMM's iterations, assuming that the rightmost term of Theorem 5.1 is negligible. Indeed, in our experiments, these terms were found to converge to zero very quickly, and we therefore ignored them. A more theoretically rigorous approach would require the consideration of these terms, a bound of which can obtained using, e.g., a projected Lanczos algorithm, as discussed in Sect. 5.2.

[9] Operations for testing if a vector belongs to $\mathcal{K}_i$, its polar and its recession must be provided. These operations might be used to check for termination of the Algorithm, which, by default, is checked every 40 iterations. For the Semidefinite Cone, both of these tests can be implemented via the Cholesky factorization.

[10] For Sects. 6.1 and 6.2 at https://github.com/nrontsis/SDPExamples.jl.

The linear systems of Algorithm 1 are solved to machine precision via an LDL factorization [27, §16.2]. We did not rely on an approximate method for the solution of the linear system because, in the problems that we considered, the projection to the Semidefinite Cone required the majority of the total time of Algorithm 1. Nevertheless, the analysis of presented in Sects. 2–4 allows for the presence of approximation errors in the solution of the linear systems.

6.1 Results for the SDPLIB Collection

We first consider problems of the SDPLIB collection, in their dual form, i.e.,

$$\begin{array}{ll} \text{maximize} & \langle F_0, Y\rangle \\ \text{subject to} & \langle F_i, Y\rangle = c_i, \quad Y \in \mathbb{S}^n_+. \end{array} \tag{25}$$

The problems are stored in the sparse SDPA form, which was designed to efficiently represent SDP problems in which the matrices $F_i, i = 0, \dots m$ are block diagonal with sparse blocks. If the matrices F_i consist of ℓ diagonal blocks, then the solution of (25) can be obtained by solving

$$\begin{array}{ll} \text{maximize} & \sum_{j=1}^{\ell}\langle F_{0,j}, Y_j\rangle \\ \text{subject to} & \sum_{j=1}^{\ell}\langle F_{i,j}, Y_j\rangle = c_i, \quad Y_j \in \mathbb{S}^{n_j}_+ \quad j = 1, \dots, \ell. \end{array} \tag{26}$$

where $F_{i,j}$ denotes the jth diagonal block of F_i and Y_j the respective block of Y. Note that (26) has more but smaller semidefinite variables than (25); thus, it is typically solved by solvers like COSMO.jl more efficiently than (25). As a result, our results refer to the solution of problems in the form (26).

Table 1 shows the results on all the problems of SDPLIB problems for which the largest semidefinite variable is of size at least 50. We observe that our approach can lead to a significant speedup of up to 20x. At the same time, the robustness of the solver is not affected, in the sense that the number of iterations to reach convergence is not, on average, increased by using approximate projections. It is remarkable that for every problem that the original `COSMO.jl` implementation converges within 2500 iterations (i.e., the default maximum iteration limit), our approach also converges with a faster overall solution time.

6.2 Infeasible Problems

Next, we demonstrate the asymptotic behavior of Algorithm 1 on the problem `infd1` of the SDPLIB collection. This problem can be expressed in the form ($\mathcal{P}$) with $\mathcal{C} = \left\{\mathrm{vec}_u(X) \;\middle|\; X \in \mathbb{S}^{30}\right\}$ (the set of vectorized 30×30 positive semidefinite matrices), and $x \in \mathbb{R}^{10}$.

As the name suggests, `infd1` is dual infeasible. Following [3, §5.2], `COSMO` detects dual infeasibility in conic problems when the certificate (4) holds approximately, that

Table 1 Results for the SDPLIB collection (§6.1). $n_{\max}$ denotes the dimensions of the largest semidefinite variable at each problem, "rank" the maximum number of computed ritz pairs by LOBPCG in the last iteration of Algorithm 3, while t^{exact}, $t^{\text{exact}}_{\text{proj}}$, $\text{Iter}^{\text{exact}}$, f^{exact} the solution time (in seconds), the time spent in projecting to the semidefinite cone, the ADMM iterations and the resulting objective when computing the projections exactly. Iter and f denote identical metrics when computing projections approximately

Name	$n_{\max}$	rank	t^{exact}	Speedup	$t^{\text{exact}}_{\text{proj}}$	$\text{Speedup}_{\text{proj}}$	$\text{Iter}^{\text{exact}}$	Iter	f^{exact}	f	f^*
arch0	161	5	2.14×10^1	1.00	1.98×10^1	0.99	2500	2500	2.35×10^5	2.35×10^5	5.67×10^{-1}
arch2	161	4	2.24×10^1	0.83	2.08×10^1	0.82	2500	2500	8.49×10^4	7.88×10^4	6.72×10^{-1}
arch4	161	5	2.24×10^1	0.65	2.05×10^1	0.62	2500	2500	3.30×10^5	3.30×10^5	9.73×10^{-1}
arch8	161	6	2.23×10^1	0.64	2.07×10^1	0.62	2500	2500	1.49×10^6	1.49×10^6	7.06
control10	100	26	3.61×10^1	0.98	1.05×10^1	1.00	2500	2500	7.31×10^2	7.51×10^2	3.85×10^1
control11	110	17	5.23×10^1	0.93	1.25×10^1	0.96	2500	2500	9.51×10^2	8.28×10^2	3.20×10^1
control6	60	16	7.92	0.76	4.37	0.65	2500	2500	6.44×10^2	6.44×10^2	3.73×10^1
control7	70	15	1.10×10^1	0.72	5.60	0.59	2500	2500	6.79×10^2	6.79×10^2	2.06×10^1
control8	80	14	1.63×10^1	0.77	6.75	0.62	2500	2500	6.97×10^2	7.08×10^2	2.03×10^1
control9	90	14	2.35×10^1	0.74	8.07	0.57	2500	2500	7.00×10^2	8.48×10^2	1.47×10^1
equalG11	801	104	3.49×10^1	3.11	3.03×10^1	3.80	160	120	6.25×10^2	6.25×10^2	6.29×10^2
equalG51	1001	258	1.05×10^2	2.14	9.27×10^1	2.17	280	160	3.99×10^3	3.98×10^3	4.01×10^3
gpp100	100	8	1.27	3.15	1.17	4.08	360	440	-4.49×10^1	-4.49×10^1	-4.49×10^1
gpp124-1	124	8	3.00	4.56	2.76	6.18	520	520	−7.28	−7.28	−7.34
gpp124-2	124	7	1.72	2.86	1.57	3.34	320	320	-4.68×10^1	-4.68×10^1	-4.69×10^1
gpp124-3	124	9	1.14	3.04	9.36×10^{-1}	3.74	200	280	-1.53×10^2	-1.53×10^2	-1.53×10^2
gpp124-4	124	9	2.24	7.23	2.09	10.44	360	240	-4.19×10^2	-4.18×10^2	-4.19×10^2

Table 1 continued

Name	$n_{\max}$	rank	t^{exact}	Speedup	$t_{\text{proj}}^{\text{exact}}$	Speedup$_{\text{proj}}$	Iter$^{\text{exact}}$	Iter	f^{exact}	f	f^*
gpp250-1	250	10	1.37×10^1	5.03	1.23×10^1	8.22	720	720	-1.54×10^1	-1.54×10^1	-1.54×10^1
gpp250-2	250	12	1.88×10^1	6.69	1.72×10^1	13.21	840	800	-8.19×10^1	-8.18×10^1	-8.19×10^1
gpp250-3	250	13	1.96×10^1	8.90	1.80×10^1	15.11	840	600	-3.03×10^2	-3.03×10^2	-3.04×10^2
gpp250-4	250	13	8.03	6.35	7.37	11.64	360	360	-7.47×10^2	-7.47×10^2	-7.47×10^2
gpp500-1	500	20	4.67×10^1	3.48	4.24×10^1	4.92	560	680	-2.61×10^1	-2.60×10^1	-2.53×10^1
gpp500-2	500	19	5.46×10^1	2.70	4.96×10^1	5.66	640	1720	-1.56×10^2	-1.56×10^2	-1.56×10^2
gpp500-3	500	21	8.17×10^1	14.45	7.44×10^1	29.97	960	440	-5.13×10^2	-5.13×10^2	-5.13×10^2
gpp500-4	500	22	2.71×10^1	5.50	2.42×10^1	10.77	320	360	-1.57×10^3	-1.57×10^3	-1.57×10^3
maxGll	800	43	1.31×10^2	6.94	1.18×10^2	13.21	640	600	6.29×10^2	6.29×10^2	6.29×10^2
maxG32	2000	70	1.23×10^3	9.01	1.11×10^3	22.47	760	760	1.57×10^3	1.57×10^3	1.57×10^3
maxG51	1000	54	7.22×10^1	3.17	6.43×10^1	5.25	200	360	4.00×10^3	4.00×10^3	4.00×10^3
maxG55	5000	105	1.27×10^4	8.97	1.15×10^4	17.65	840	960	1.28×10^4	1.28×10^4	10.00×10^3
maxG60	7000	178	4.67×10^4	10.75	4.31×10^4	15.71	880	920	1.52×10^4	1.51×10^4	1.52×10^4
mcpl00	100	8	1.37	3.25	1.25	3.98	280	320	2.26×10^2	2.26×10^2	2.26×10^2
mcp124-1	124	20	2.88	2.29	2.63	2.37	400	400	1.42×10^2	1.42×10^2	1.42×10^2
mcp124-2	124	8	1.51	3.13	1.38	4.26	240	240	2.70×10^2	2.70×10^2	2.70×10^2
mcp124-3	124	9	1.49	3.21	1.35	4.06	240	400	4.68×10^2	4.68×10^2	4.68×10^2
mcp124-4	124	8	1.60	4.62	1.47	5.93	200	240	8.64×10^2	8.64×10^2	8.64×10^2
mcp250-1	250	30	1.68×10^1	2.69	1.52×10^1	2.88	680	680	3.17×10^2	3.17×10^2	3.17×10^2
mcp250-2	250	14	1.17×10^1	6.73	1.05×10^1	10.60	560	520	5.32×10^2	5.32×10^2	5.32×10^2
mcp250-3	250	13	9.16	5.14	8.20	6.99	440	400	9.81×10^2	9.81×10^2	9.81×10^2

Table 1 continued

Name	$n_{\max}$	rank	t^{exact}	Speedup	$t^{\text{exact}}_{\text{proj}}$	Speedup$_{\text{proj}}$	Iter$^{\text{exact}}$	Iter	f^{exact}	f	f^*
mcp250-4	250	14	4.39	2.90	3.93	4.43	200	360	1.68×10^{3}	1.68×10^{3}	1.68×10^{3}
mcp500-1	500	72	9.89×10^{1}	1.26	9.11×10^{1}	1.50	920	2400	5.98×10^{2}	5.98×10^{2}	5.98×10^{2}
mcp500-2	500	27	1.20×10^{2}	12.40	1.09×10^{2}	22.08	1160	600	1.07×10^{3}	1.07×10^{3}	1.07×10^{3}
mcp500-3	500	21	4.78×10^{1}	4.10	4.25×10^{1}	7.26	400	760	1.85×10^{3}	1.85×10^{3}	1.85×10^{3}
mcp500-4	500	22	7.31×10^{1}	9.82	6.67×10^{1}	17.94	680	480	3.57×10^{3}	3.57×10^{3}	3.57×10^{3}
qap10	101	29	1.21	2.07	1.07	2.09	280	160	-1.08×10^{3}	-1.08×10^{3}	-1.09×10^{1}
qap8	65	20	3.70×10^{-1}	1.13	3.04×10^{-1}	1.06	160	160	-7.42×10^{2}	-7.42×10^{2}	-7.57×10^{2}
qap9	82	24	4.84×10^{-1}	1.16	4.10×10^{-1}	1.10	160	160	-1.39×10^{3}	-1.39×10^{3}	-1.41×10^{3}
qpG11	1600	38	7.10×10^{2}	5.71	6.11×10^{2}	13.14	920	880	2.45×10^{3}	2.45×10^{3}	2.45×10^{3}
qpG51	2000	20	3.64×10^{3}	10.13	3.20×10^{3}	38.71	2500	2500	1.18×10^{4}	1.19×10^{4}	1.18×10^{3}
ss30	294	8	9.70×10^{1}	0.95	8.87×10^{1}	0.92	2500	2500	1.94×10^{6}	1.94×10^{6}	2.02×10^{1}
theta2	100	18	1.04×10^{1}	2.67	9.56	2.82	2240	2240	3.29×10^{1}	3.29×10^{1}	3.29×10^{1}
theta3	150	26	5.05	3.12	4.60	3.85	440	560	4.23×10^{1}	4.22×10^{1}	4.22×10^{1}
theta4	200	34	1.13×10^{1}	3.99	1.01×10^{1}	4.58	720	680	5.05×10^{1}	5.04×10^{1}	5.03×10^{1}
theta5	250	17	1.16×10^{1}	0.88	1.03×10^{1}	1.09	480	2500	5.74×10^{1}	9.43×10^{1}	5.72×10^{1}
theta6	300	13	1.68×10^{1}	1.02	1.46×10^{1}	1.33	480	2500	6.36×10^{1}	-2.39	6.35×10^{1}
thetaG11	801	26	7.71×10^{2}	20.79	7.37×10^{2}	47.91	1280	1280	3.99×10^{2}	3.99×10^{2}	4.00×10^{2}
thetaG51	1001	116	1.31×10^{3}	5.40	1.19×10^{3}	7.00	2500	2500	3.40×10^{2}	3.43×10^{2}	3.49×10^{2}

“Speedup” and “Speedup$_{\text{proj}}$” denote the speedups achieved for the full ADMM algorithm and only the projection part when using LOBPCG, and f^* the optimal objective of each problem. Hardware used: Intel Gold 5120 with 192GB of memory

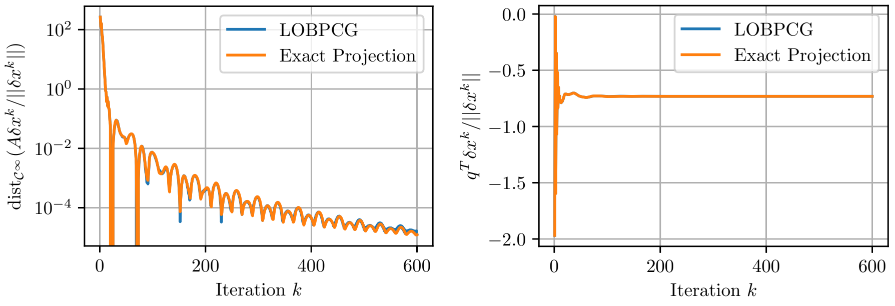

Fig. 1 Convergence of $\left(\delta x^k / \left\|\delta x^k\right\|\right)_{k\in\mathbb{N}}$ to a certificate of dual infeasibility for problem `infd1` from the SDPLIB. A fixed value of $\rho = 10^{-3}$ is used in Algorithm 1

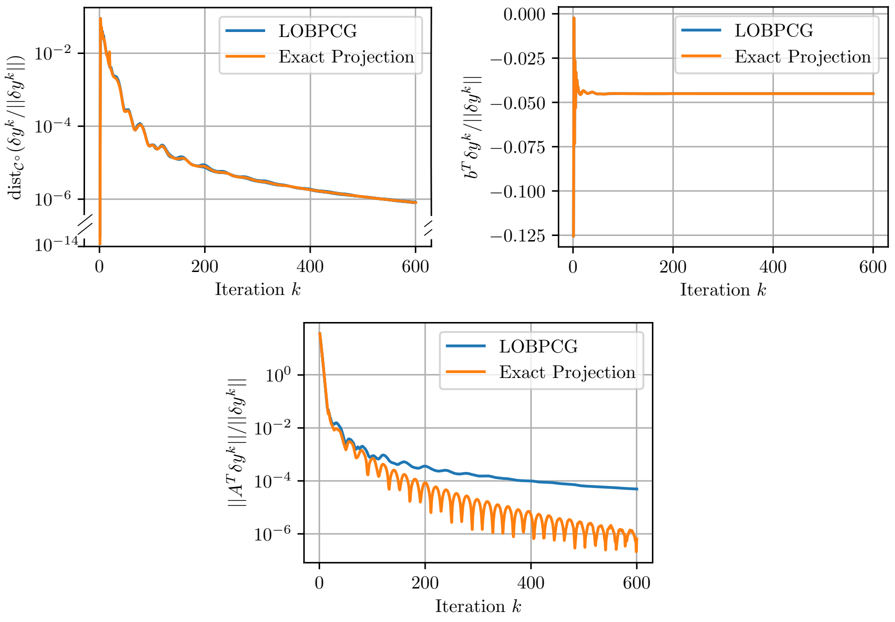

Fig. 2 Convergence of $\left(\delta y^k / \left\|\delta y^k\right\|\right)_{k\in\mathbb{N}}$ to a certificate of primal infeasibility for the dual of the problem `infd1` from the SDPLIB. A fixed value of $\rho = 10^{3}$ is used in Algorithm 1

is when $\delta x^k \neq 0$ and

$$\mathrm{dist}_{C^\infty}\left(A\bar{x}^k\right) < \epsilon_{\mathrm{dinf}}, \quad \text{and} \quad q^T\bar{x}^k < \epsilon_{\mathrm{dinf}},$$

where $\bar{x}^k := \delta x^k/\|\delta x^k\|$, for a positive tolerance ϵ_{dinf}. Figure 1 depicts the convergence of these quantities both for the case where the projection to the semidefinite cone is computed approximately and when LOBPCG is used. The convergence of the successive differences to a certificate of dual infeasibility is practically identical.

To demonstrate the detection of primal infeasibility, we consider the dual of `infd1`. Following [3, §5.2], `COSMO` detects primal infeasibility in conic problems when the certificate (3) is satisfied approximately, that is when $\delta y^k \neq 0$ and

$$\left\| P \bar{y}^k \right\| < \epsilon_{\text{pinf}}, \quad \left\| A^T \bar{y}^k \right\| < \epsilon_{\text{pinf}}, \quad \text{dist}_{\mathcal{C}^\circ} \left(\bar{y}^k \right) < \epsilon_{\text{pinf}}, \quad b^T \bar{y}^k < \epsilon_{\text{pinf}},$$

where $\bar{y}^k := \delta y^k / ||\delta y^k||$, for a positive tolerance ϵ_{pinf}. Note that, for the case of the dual of `infd1`, the first condition is trivial since $P = 0$. Figure 2 compares the convergence of our approach, against standard COSMO, to a certificate of infeasibility. LOBPCG yields practically identical convergence as the exact projection for all of the quantities except $\left\| A^T \bar{y}^k \right\|$, where slower convergence is observed.

Note that SDPLIB also contains two instances of primal infeasible problems: `infp1` and `infp2`. However, in these problems, there is a single positive semidefinite variable of size 30 and, in ADMM, the matrices projected to the semidefinite cone have rank $15 = 30/2$ across all the iterations (except for the very first few). Thus, according to Sect. 5.1 LOBPCG yields identical results to the exact projection; hence, a comparison would be of little value.

7 Conclusions

We have shown that state-of-the art approximate eigensolvers can bring significant speedups to ADMM for the case of Semidefinite Programming. We have extended the results of [3] to show that infeasibility can be detected even in the presence of appropriately controlled projection errors, thus ensuring the same overall asymptotic behavior as an exact ADMM method. Future research directions include exploring the performance of other state-of-the-art eigensolvers from the Linear Algebra community [37].

Acknowledgements This work was supported by the EPSRC AIMS CDT Grant EP/L015987/1 and Schlumberger. We would also like to acknowledge the support of the NII International Internship Program, which funded the first author's visit to NII, during which this collaboration was launched.

Appendix A: Convergence of Approximate Iterations of Nonexpansive Operators

In this section, we provide a proof for Theorem 3.1. We achieve this by generalizing some of the results of [2,32] and [18] to account for sequences generated by approximate evaluation of averaged operators T for which $\operatorname{cl}\mathcal{R}(\mathrm{Id}-T)$ has the *minimum property* defined below:

Definition A.1 (*minimum property*) Let $K \subseteq \mathcal{H}$ be closed and let ℓ be the minimum-norm element of $\operatorname{cl}\operatorname{conv} K$. The set K has the minimum property if $\ell \in K$.

Note that $\operatorname{cl}\mathcal{R}(\mathrm{Id}-T)$ has the minimum property when T is defined as (7) because the domain of (7) is convex [32, Lemma 5]. Thus, Theorem 3.1 follows from the following result:

Proposition A.1 *Consider some $\mathcal{D} \subseteq \mathcal{H}$ that is closed, an averaged $T : \mathcal{D} \to \mathcal{D}$ and assume that $\operatorname{cl}\mathcal{R}(\mathrm{Id}-T)$ has the minimum property. For any sequence defined as*

$$(\forall k \in \mathbb{N}) \quad x^{k+1} \approx_{\epsilon^k} Tx^k,$$

for some $x^0 \in \mathcal{D}$ and a summable nonnegative sequence $(\epsilon^k)_{k\in\mathbb{N}}$, we have

$$\lim_{k\to\infty}(x^{k+1}-x^k) = \lim_{k\to\infty} x^k/k = -\ell,$$

where ℓ is the unique element of minimum norm in $\operatorname{cl}\mathcal{R}(\mathrm{Id}-T)$.

To prove Proposition A.1, we will need the following Lemma:

Lemma A.1 *For any approximate iteration over an averaged operator, defined as:*

$$x^{k+1} = (1-t)x^k + t\tilde{T}x^k + \delta^k,$$

where $x_0 \in \mathcal{D} \subseteq \mathcal{H}$, $\tilde{T} : \mathcal{D} \mapsto \mathcal{D}$ is a nonexpansive operator, $t \in [0, 1)$ and $(\|\delta^k\|)_{k\in\mathbb{N}}$ is some real, summable sequence representing approximation errors in the iteration, we have:

$$\frac{1}{N}\lim_{k\to\infty}\left\|x^{k+N}-x^k\right\| = \lim_{k\to\infty}\left\|x^{k+1}-x^k\right\|. \tag{27}$$

Proof When $(\|\delta^k\|)_{k\in\mathbb{N}}$ is zero, the proof for (27) is mentioned as "a straightforward modification of Ishikawa's argument" in [2, Proof of Theorem 2.1]; we show it in detail and for any summable $(\|\delta^k\|)$, indeed by modifying [18, Proof of Lemma 2].

We will first show that $\lim_{k\to\infty}\|x^{k+1}-x^k\|$ and $\lim_{k\to\infty}\|x^{k+N}-x^k\|$ exist and are bounded. To this end, consider the sequence

$$q^k := \left\|x^{k+1}-x^k\right\| + \sum_{i=k}^{\infty}\left\|\delta^i-\delta^{i-1}\right\|.$$

Note that $\sum_{i=k}^{\infty}\|\delta^i-\delta^{i-1}\|$ converges to a finite value for every k, as it is the limit $n\to\infty$ of the nondecreasing sequence $(\sum_{i=k}^{n}\|\delta^i-\delta^{i-1}\|)$ that is bounded above because $(\|\delta^i\|)$ is summable. Since $\|x^k-x^{k-1}\| \le \|x^{k+1}-x^k\| + \|\delta^k-\delta^{k-1}\|$, we conclude that (q^k) is nonincreasing. Since (q^k) is also bounded below by zero, we conclude that

$\lim_{k\to\infty} q^k$ exists and is bounded. Finally, because $\lim_{k\to\infty} \sum_{i=k}^{\infty} \left\|\delta^i - \delta^{i-1}\right\| = 0$, we conclude that $\lim_{k\to\infty} \left\|x^{k+1} - x^k\right\|$ also exists and is bounded. Using similar arguments, we can show the same for $\lim_{k\to\infty} \left\|x^{k+N} - x^k\right\|$.

Since $x^{k+1} - x^k = t(\tilde{T}x^k - x^k) + \delta^k \ \forall k \in \mathbb{N}$, it suffices to show

$$\frac{1}{N} \lim_{k\to\infty} \left\|x^{k+N} - x^k\right\| = t \lim_{k\to\infty} \left\|x^k - \tilde{T}x^k\right\| \tag{28}$$

instead of (27), because $\left\|\delta^k\right\| \to 0$. Note that both limits in the above equation exist and are bounded, as discussed above.

We will show (28) by showing that

$$tr \le \frac{1}{N} \lim_{k\to\infty} \left\|x^{k+N+1} - x^{k+1}\right\| \le tr.$$

where $r := \left\|x^k - \tilde{T}x^k\right\|$.

The proof for both of these bounds depends on the following equality:

$$\begin{aligned} x^{k+N+1} - x^{k+1} &= \sum_{i=1}^{N} [(1-t)x^{k+i} + t\tilde{T}x^{k+i} + \delta^{k+i}] - x^{k+i} \\ &= \sum_{i=1}^{N} t(\tilde{T}x^{k+i} - x^{k+i}) + \delta^{k+i}. \end{aligned} \tag{29}$$

The upper bound follows easily from the triangular inequality:

$$\left\|x^{k+N+1} - x^{k+1}\right\| \le \sum_{i=1}^{N} \left(t \left\|\tilde{T}x^{k+i} - x^{k+i}\right\| + \left\|\delta^{k+i}\right\| \right) \Rightarrow \tag{30}$$

$$\lim_{k\to\infty} \frac{1}{N} \left\|x^{k+N+1} - x^{k+1}\right\| \le t \lim_{k\to\infty} \left\|x^k - \tilde{T}x^k\right\|, \tag{31}$$

since $\left\|\delta^{k+i}\right\| \to 0$.

To get the lower bound, define $u^i := \tilde{T}x^i - x^i$ and $s := 1 - t$, and note that for all $i \in \mathbb{N}$:

$$\begin{aligned} \left\|u^{i+1} - su^i\right\| &= \left\|\tilde{T}x^{i+1} - x^{i+1} - (1-t)(\tilde{T}x^i - x^i)\right\| \\ &\le \left\|\tilde{T}x^{i+1} - ((1-t)x^i + t\tilde{T}x^i)) - (1-t)(\tilde{T}x^i - x^i)\right\| + \|\delta_i\| \\ &= \left\|\tilde{T}((1-t)x^i + t\tilde{T}x^i + \delta^i) - \tilde{T}x^i\right\| + \left\|\delta^i\right\| \\ &\le \left\|((1-t)x^i + t\tilde{T}x^i) - x^i\right\| + 2\left\|\delta^i\right\| \\ &= t\left\|\tilde{T}x^i - x^i\right\| + 2\left\|\delta^i\right\| \\ &= (1-s)\left\|u^i\right\| + 2\left\|\delta^i\right\|. \end{aligned} \tag{32}$$

Thus, using (29), and since $\left\|\delta^{k+i}\right\| \to 0$ we have

$$\lim_{k\to\infty} s^{N-1}\left\|x^{k+N+1}-x^{k+1}\right\| = \lim_{k\to\infty}\left\|s^{N-1}\sum_{i=1}^{N}(1-s)u^{k+i}\right\|$$
$$\geq \limsup_{k\to\infty}\left[(1-s^N)\left\|u^{k+N}\right\| - \sum_{i=1}^{N-1}s^{N-1-i}(1-s^i)\left\|u^{k+i+1}-su^{k+i}\right\|\right],$$

where [18, Lemma 1] was used above,

$$\geq (1-s^N)\liminf_{k\to\infty}\left\|u^{k+N}\right\| - \sum_{i=1}^{N-1}s^{N-1-i}(1-s^i)\liminf_{k\to\infty}\left\|u^{k+i+1}-su^{k+i}\right\|$$
$$\geq (1-s^N)r - \sum_{i=1}^{N-1}s^{N-1-i}(1-s^i)(1-s)r,$$

because of (32) and because

$$2\left\|\delta^i\right\| \to 0, = \left[1-s^N-\sum_{i=1}^{N-1}s^{N-1-i}(1-s^i)(1-s)\right]r = rs^{N-1}\sum_{i=1}^{N}(1-s).$$

where [18, Lemma 1] was used in the last equality above. Thus,

$$\lim_{k\to\infty} s^{N-1}\left\|x^{k+N+1}-x^{k+1}\right\| \geq rs^{N-1}\sum_{i=1}^{N}(1-s)$$

or, since $s > 0$, we get the desired lower bound that concludes the proof

$$\frac{1}{N}\lim_{k\to\infty}\left\|x^{k+N+1}-x^{k+1}\right\| \geq t\lim_{k\to\infty}\left\|x^k-\tilde{T}x^k\right\|.$$

□

We can now proceed with the Proof of Proposition A.1. We first show $\lim_{k\to\infty} x^k/k = -\ell$. The nonexpansiveness of T gives

$$\left\|x^n - T^n x^0\right\| \leq \left\|Tx^{n-1} - TT^{n-1}x^0\right\| + \epsilon_n, \quad \forall n \in \mathbb{N}$$
$$\Rightarrow \frac{1}{n}\left\|x^n - T^n x^0\right\| \leq \frac{1}{n}\sum_{i=1}^{n}\epsilon_i \Rightarrow \lim_{n\to\infty}\left\|\frac{x^n}{n}-\frac{T^n x^0}{n}\right\| = 0,$$

where the summability of $(\epsilon_i)_{i\in\mathbb{N}}$ was used in the last implication. Thus, the claim follows from [32, Theorem 2].

It remains to show that $\lim_{k\to\infty} x^{k+1} - x^k$ also converges to $-\ell$. To this end, note that due to Lemma A.1, we have $\lim_{k\to\infty} \left\|x^{k+N} - x^k\right\| / N = \lim_{k\to\infty} \left\|x^{k+1} - x^k\right\|$. Furthermore, the non-expansiveness of T gives for all $N \geq 1$:

$$\lim_{k\to\infty} \left\|x^{k+N} - x^k\right\| / N \leq \frac{1}{N}\left(\left\|x^N - x^0\right\| + 2\sum_{i=0}^{\infty} \epsilon^i\right).$$

Noting also that $\lim\limits_{N\to\infty} \left\|x^N - x^0\right\| / N$ exists due to the first part of the proof, we get:

$$\begin{aligned}\lim_{k\to\infty} \left\|x^{k+1} - x^k\right\| &= \lim_{k\to\infty} \left\|x^{k+N} - x^k\right\| / N \quad \forall N \geq 1 \\ &\leq \lim_{N\to\infty} \frac{1}{N} \left\|x^N - x^0\right\| \leq \limsup_{N\to\infty} \frac{1}{N} \sum_{i=0}^{N-1} \left\|x^{i+1} - x^i\right\| \\ &= \lim_{k\to\infty} \left\|x^{k+1} - x^k\right\|,\end{aligned} \tag{33}$$

where the above chain of equations follows [2, Proof of Theorem 2.1] and the properties of the Cesàro summation for the last equation.

Hence, $\lim_{k\to\infty} \left\|x^{k+1} - x^k\right\| = \lim_{k\to\infty} \left\|x^k - x^0\right\| / k$, which is equal to $\|\ell\|$, as we have shown in the first part of this proof. As a result, we also have $\lim_{k\to\infty} \left\|Tx^k - x^k\right\| = \|\ell\|$ because $\epsilon^k \to 0$. We conclude that $\lim_{k\to\infty} Tx^k - x^k = -\ell$ due to [32, Lemma 2]. The desired $\lim_{k\to\infty} x^{k+1} - x^k = -\ell$ then follows because $\epsilon^k \to 0$.

References

1. Aishima, K.: Global convergence of the restarted Lanczos and Jacobi-Davidson methods for symmetric eigenvalue problems. Numer. Math. **131**(3), 405–423 (2015). https://doi.org/10.1007/s00211-015-0699-4
2. Baillon, J.B., Bruck, R.E., Reich, S.: On the asymptotic behavior of nonexpansive mappings and semigroups in Banach spaces. Houst. J. Math. **4**(1), 1–9 (1978)
3. Banjac, G., Goulart, P.J., Stellato, B., Boyd, S.: Infeasibility detection in the alternating direction method of multipliers for convex optimization. J. Optim. Theory Appl. **183**(2), 490–519 (2019). https://doi.org/10.1007/s10957-019-01575-y
4. Bauschke, H., Combettes, P.L.: Convex Analysis and Monotone Operator Theory in Hilbert Spaces, 2nd edn. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-48311-5
5. Benzi, M., Golub, G., Liesen, J.: Numerical solution of saddle point problems. Acta Numer. **14**, 1–137 (2005). https://doi.org/10.1017/s0962492904000212
6. Boyd, S., El Ghaoui, L., Feron, E., Balakrishnan, V.: Linear Matrix Inequalities in System and Control Theory. SIAM, Philadelphia (1994). https://doi.org/10.1137/1.9781611970777
7. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1), 1–122 (2011). https://doi.org/10.1561/2200000016
8. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
9. d'Aspremont, A., El Ghaoui, L., Jordan, M.I., Lanckriet, G.R.G.: A direct formulation for sparse PCA using semidefinite programming. SIAM Rev. **49**(3), 434–448 (2007). https://doi.org/10.1137/050645506

10. Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H.: Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM, Philadelphia (2000). https://doi.org/10.1137/1.9780898719581
11. Eckstein, J., Bertsekas, D.P.: On the Douglas–Rachford splitting method and the proximal point algorithm for maximal monotone operators. Math. Program. **55**(1), 293–318 (1992). https://doi.org/10.1007/BF01581204
12. Garstka, M., Cannon, M., Goulart, P.J.: COSMO: a conic operator splitting method for convex conic problems. J. Optim. Theory Appl. (2021). https://doi.org/10.1007/s10957-021-01896-x
13. Giselsson, P., Fält, M., Boyd, S.: Line search for averaged operator iteration. ArXiv e-prints arXiv:1603.06772 (2016)
14. Golub, G.H., Van Loan, C.F.: Matrix Computations, 4th edn. Johns Hopkins University Press, Baltimore (2013)
15. Goulart, P.J., Chernyshenko, S.: Global stability analysis of fluid flows using sum-of-squares. Physica D **241**(6), 692–704 (2012). https://doi.org/10.1016/j.physd.2011.12.008
16. Goulart, P.J., Nakatsukasa, Y., Rontsis, N.: Accuracy of approximate projection to the semidefinite cone. Linear Algebra Appl. **594**, 177–192 (2020). https://doi.org/10.1016/j.laa.2020.02.014
17. Hernandez, V., Roman, J.E., Tomas, A., Vidal, V.: A survey of software for sparse eigenvalue problems. Tech. Rep. STR-6, Universitat Politècnica de València (2009). http://slepc.upv.es
18. Ishikawa, S.: Fixed points and iteration of a nonexpansive mapping in a Banach space. Proc. Am. Math. Soc. **59**(1), 65–71 (1976). https://doi.org/10.1090/s0002-9939-1976-0412909-x
19. Knyazev, A.V.: Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method. SIAM J. Sci. Comput. **23**(2), 517–541 (2001). https://doi.org/10.1137/S1064827500366124
20. Lanckriet, G.R.G., Cristianini, N., Bartlett, P., El Ghaoui, L., Jordan, M.I.: Learning the kernel matrix with semidefinite programming. J. Mach. Learn. Res. **5**, 27–72 (2004)
21. Lavaei, J., Low, S.H.: Zero duality gap in optimal power flow problem. IEEE Trans. Power Syst. **27**(1), 92–107 (2012). https://doi.org/10.1109/TPWRS.2011.2160974
22. Lehoucq, R., Sorensen, D., Yang, C.: ARPACK Users' Guide. SIAM, Philadelphia (1998). https://doi.org/10.1137/1.9780898719628
23. Lemon, A., So, A.M.C., Ye, Y.: Low-rank semidefinite programming: theory and applications. Found. Trends Optim. **2**(1–2), 1–156 (2016). https://doi.org/10.1561/2400000009
24. Li, Y., Liu, H., Wen, Z., Yuan, Y.: Low-rank matrix iteration using polynomial-filtered subspace extraction. SIAM J. Sci. Comput. **42**(3), A1686–A1713 (2020). https://doi.org/10.1137/19M1259444
25. MOSEK ApS: The MOSEK optimization toolbox for Python manual. http://www.mosek.com/
26. Nakatsukasa, Y.: Sharp error bounds for Ritz vectors and approximate singular vectors. Math. Comput. **89**(324), 1843–1866 (2020). https://doi.org/10.1090/mcom/3519
27. Nocedal, J., Wright, S.J.: Numerical Optimization, 2nd edn. Springer, New York (2006)
28. O'Donoghue, B., Chu, E., Parikh, N., Boyd, S.: Conic optimization via operator splitting and homogeneous self-dual embedding. J. Optim. Theory Appl. **169**(3), 1042–1068 (2016). https://doi.org/10.1007/s10957-016-0892-3
29. Orban, D., Arioli, M.: In: Iterative solution of symmetric quasi-definite linear systems, USA (2017). https://doi.org/10.1137/1.9781611974737
30. Paige, C.C.: Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. Linear Algebra Appl. **34**, 235–258 (1980). https://doi.org/10.1016/0024-3795(80)90167-6
31. Parlett, B.N.: The Symmetric Eigenvalue Problem. SIAM, Philadelphia (1998). https://doi.org/10.1137/1.9781611971163
32. Pazy, A.: Asymptotic behavior of contractions in Hilbert space. Isr. J. Math. **9**(2), 235–240 (1971). https://doi.org/10.1007/BF02771588
33. Prajna, S., Papachristodoulou, A., Parrilo, P.A.: Introducing SOSTOOLS: a general purpose sum of squares programming solver. In: Proceedings of the 41st IEEE Conference on Decision and Control, vol. 1, pp. 741–746 (2002). https://doi.org/10.1109/CDC.2002.1184594
34. Rontsis, N.: Numerical optimization with applications to machine learning. Ph.D. thesis, University of Oxford (2019). https://ora.ox.ac.uk/objects/uuid:1d153c5b-deef-4082-bcea-cf2deb824997
35. Saad, Y.: Numerical Methods for Large Eigenvalue Problems. SIAM, Philadelphia (2011). https://doi.org/10.1137/1.9781611970739
36. Souto, M., Garcia, J.D., Veiga, Á.: Exploiting low-rank structure in semidefinite programming by approximate operator splitting. Optimization (2020). https://doi.org/10.1080/02331934.2020.1823387

37. Stathopoulos, A., McCombs, J.R.: PRIMME: preconditioned iterative multimethod eigensolver: methods and software description. ACM Trans. Math. Softw. **37**(2), 21:1-21:30 (2010). https://doi.org/10.1145/1731022.1731031
38. Stellato, B., Banjac, G., Goulart, P.J., Bemporad, A., Boyd, S.: OSQP: an operator splitting solver for quadratic programs. Math. Program. Comput. **12**(4), 637–672 (2020). https://doi.org/10.1007/s12532-020-00179-2
39. Stewart, G.W.: Matrix Algorithms: Volume 1, Basic Decompositions. SIAM, Philadelphia (1998). https://doi.org/10.1137/1.9781611971408
40. Toh, K.C., Todd, M.J., Tütüncü, R.H.: SDPT3—a MATLAB software package for semidefinite programming. Optim. Methods Softw. **11**, 545–581 (1998). https://doi.org/10.1080/10556789908805762
41. Wen, Z., Goldfarb, D., Yin, W.: Alternating direction augmented Lagrangian methods for semidefinite programming. Math. Program. Comput. **2**(3), 203–230 (2010). https://doi.org/10.1007/s12532-010-0017-1
42. Yamashita, M., Fujisawa, K., Kojima, M.: Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0). Optim. Methods Softw. **18**(4), 491–505 (2003). https://doi.org/10.1080/1055678031000118482
43. Zheng, Y., Fantuzzi, G., Papachristodoulou, A., Goulart, P.J., Wynn, A.: Chordal decomposition in operator-splitting methods for sparse semidefinite programs. Math. Program. **180**(1), 489–532 (2020). https://doi.org/10.1007/s10107-019-01366-3