Computational Optimization and Applications
https://doi.org/10.1007/s10589-019-00122-9

Solution refinement at regular points of conic problems

Enzo Busseti[1] · Walaa M. Moursi[1,2] · Stephen Boyd[1]

Received: 5 March 2019

Abstract
Many numerical methods for conic problems use the homogenous primal–dual embedding, which yields a primal–dual solution or a certificate establishing primal or dual infeasibility. Following Themelis and Patrinos (IEEE Trans Autom Control, 2019), we express the embedding as the problem of finding a zero of a mapping containing a skew-symmetric linear function and projections onto cones and their duals. We focus on the special case when this mapping is regular, i.e., differentiable with nonsingular derivative matrix, at a solution point. While this is not always the case, it is a very common occurrence in practice. In this paper we do not aim for new theorerical results. We rather propose a simple method that uses LSQR, a variant of conjugate gradients for least squares problems, and the derivative of the residual mapping to refine an approximate solution, i.e., to increase its accuracy. LSQR is a matrix-free method, i.e., requires only the evaluation of the derivative mapping and its adjoint, and so avoids forming or storing large matrices, which makes it efficient even for cone problems in which the data matrices are given and dense, and also allows the method to extend to cone programs in which the data are given as abstract linear operators. Numerical examples show that the method improves an approximate solution of a conic program, and often dramatically, at a computational cost that is typically small compared to the cost of obtaining the original approximate solution. For completeness we describe methods for computing the derivative of the projection onto the cones commonly used in practice: nonnegative, second-order, semidefinite, and exponential cones. The paper is accompanied by an open source implementation.

✉ Walaa M. Moursi
wmoursi@stanford.edu

Enzo Busseti
ebusseti@stanford.edu

Stephen Boyd
boyd@stanford.edu

1 Department of Electrical Engineering, Stanford University, 350 Serra Mall, Stanford, CA 94305, USA

2 Mathematics Department, Faculty of Science, Mansoura University, Mansoura 35516, Egypt

Published online: 26 August 2019

Keywords Conic programming · Homogenous self-dual embedding · Projection operator · Residual map

1 Conic problem and homogeneous primal–dual embedding

We consider a conic optimization problem in its primal (P) and dual (D) forms (see, e.g., [7, §4.6.1] and [6]):

$$\begin{array}{ll} \text{(P) minimize } c^T x & \text{(D) minimize } b^T y \\ \quad \text{subject to } Ax + s = b & \quad \text{subject to } A^T y + c = 0 \\ \qquad s \in \mathcal{K} & \qquad y \in \mathcal{K}^*. \end{array} \tag{1}$$

Here $x \in \mathbf{R}^n$ is the *primal* variable, $y \in \mathbf{R}^m$ is the *dual* variable, and $s \in \mathbf{R}^m$ is the primal *slack* variable. The set $\mathcal{K} \subseteq \mathbf{R}^m$ is a nonempty closed convex cone and the set $\mathcal{K}^* \subseteq \mathbf{R}^m$ is its *dual cone*, $\mathcal{K}^* = \left\{ y \in \mathbf{R}^m \;\middle|\; \inf_{k \in \mathcal{K}} y^T k \geq 0 \right\}$. The *problem data* are the matrix $A \in \mathbf{R}^{m \times n}$, the vectors $b \in \mathbf{R}^m$, $c \in \mathbf{R}^n$ and the cone $\mathcal{K}$.

Applications of conic problems Conic problems are widely used in practice. Any convex optimization problem can be formulated as a conic problem [6]. Popular convex optimization solvers, such as `ecos` [11], `scs` [29,39], `mosek` [24], `sedumi` [41], solve problems formulated as conic problems. Convex optimization frameworks, such as `yalmip` [21], `cvx` [15,16], `cvxpy` [10], `Convex.jl` [45], and `cvxr` [14], let the user formulate a convex optimization problem in high level mathematical language and then transform it to a conic problem. Classes of problems of practical importance, such as financial portfolio optimization [2,4] and power grid management [43], are increasingly specified, and solved, as conic problems.

Optimality conditions Let $(x, y, s) \in \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R}^m$. If (x, y, s) satisfies the optimality or Karush–Kuhn–Tucker (KKT) conditions

$$Ax + s = b, \quad A^T y + c = 0, \quad s \in \mathcal{K}, \quad y \in \mathcal{K}^*, \quad s^T y = 0, \tag{2}$$

then (x, s) is primal optimal, y is dual optimal, and we say that (x, y, s) is a solution of the primal–dual pair (1).

Primal and dual infeasibility If y satisfies

$$A^T y = 0, \quad y \in \mathcal{K}^*, \quad b^T y = -1, \tag{3}$$

then y serves as a proof or certificate that the primal problem is infeasible (equivalently, the dual problem is unbounded). If (x, s) satisfies

$$Ax + s = 0, \quad s \in \mathcal{K}, \quad c^T x = -1, \tag{4}$$

then the pair (x, s) serves as a proof or certificate that the primal problem is unbounded (equivalently, the dual problem is infeasible).

Solving a conic program It is easy to show that (2) and (3) are mutually exclusive, and that (2) and (4) are mutually exclusive. For non-degenerate conic programs, (3) and (4) are also mutually exclusive. (There exist degenerate conic programs for which both (3) and (4) are infeasible, but such problems do not arise in applications.) By *solving* the conic program (1), we mean finding a solution of (2), (3), or (4).

Homogenous self-dual embedding The homogenous self-dual embedding of (1), introduced by Ye and others (see, e.g., [47], [29] and [32]), can be used to solve a conic problem. The embedding is as follows:

$$Qu = v, \quad u \in \mathcal{K}, \quad v \in \mathcal{K}^*, \quad u_{m+n+1} + v_{m+n+1} > 0, \tag{5}$$

where

$$\mathcal{K} = \mathbf{R}^n \times \mathcal{K}^* \times \mathbf{R}_+, \quad \mathcal{K}^* = \{0\}^n \times \mathcal{K} \times \mathbf{R}_+,$$

and Q is the skew-symmetric matrix

$$Q = \begin{bmatrix} 0 & A^T & c \\ -A & 0 & b \\ -c^T & -b^T & 0 \end{bmatrix}.$$

The homogeneous self-dual embedding (5) is evidently positive homogeneous.

Constructing a solution of the conic problem (1), from a solution of the homogeneous embedding (5) proceeds as follows. We partition u as $u = (u_1, u_2, \tau)$, and v as $v = (v_1, v_2, \kappa)$, with

$$u_1 \in \mathbf{R}^n, \quad u_2 \in \mathcal{K}^*, \quad \tau \geq 0, \quad v_1 = 0 \in \mathbf{R}^n, \quad v_2 \in \mathcal{K}, \quad \kappa \geq 0.$$

If (u, v) satisfies (5) then the skew symmetry of Q implies that

$$u^T v = u_1^T v_1 + u_2^T v_2 + \tau\kappa = u_2^T v_2 + \tau\kappa = 0,$$

from which we conclude that $u_2^T v_2 = 0$ and $\tau\kappa = 0$. Thus in view of (5), because we do not consider the trivial solution $\tau = \kappa = 0$, one of τ and κ is positive, and the other is zero. We distinguish three cases.

- $\tau > 0$. Then $x = u_1/\tau$, $y = u_2/\tau$, $s = v_2/\tau$ is a primal–dual solution of the conic program, i.e., they satisfy (2).
- $\kappa > 0$ and $b^T u_2 < 0$. Then $y = u_2/(b^T u_2)$ is a certificate of primal infeasibility, i.e., satisfies (3).
- $\kappa > 0$ and $c^T u_1 < 0$. Then $x = u_1/(c^T u_1)$, $s = v_2/(c^T u_1)$ is a certificate of dual infeasibility, i.e., satisfies (4).

Any solution of the homogenous self-dual embedding (5) must fall in one of the above three cases. To see this, suppose $\kappa > 0$. Then $\tau = 0$ and the last equation in $v = Qu$ becomes $-c^T u_1 - b^T u_2 = \kappa > 0$, which implies that at least one of $b^T u_2$ and $c^T u_1$ is negative. These correspond to the second and third cases. (If both are negative, the conic program is degenerate.)

A converse also holds.

- If (x, y, s) satisfies (2), then $u = (x, y, 1)$ and $v = (0, s, 0)$ satisfy (5).
- If y satisfies (3), then $u = (0, y, 0)$ and $v = (0, 0, 1)$ satisfy (5).
- If x, s satisfy (4), then $u = (x, 0, 0)$, $v = (0, s, 1)$ satisfy (5).

2 The residual map

The conic complementarity set We define the *conic complementarity set* as

$$\mathcal{C} = \left\{(u, v) \in \mathcal{K} \times \mathcal{K}^* \;\middle|\; u^T v = 0\right\}.$$

$\mathcal{C}$ is evidently a closed cone, but not necessarily convex. We let Π denote Euclidean projection on $\mathcal{K}$, and Π^* denote Euclidean projection on $-\mathcal{K}^*$. We observe that (see, e.g., [23])

$$\Pi^* = I - \Pi,$$

where I denotes the identity operator.

Minty's parametrization of the complementarity set The mapping $M : \mathbf{R}^{m+n+1} \to \mathcal{C}$ given by

$$M(z) = (\Pi z, -\Pi^* z)$$

is called the Minty parametrization of $\mathcal{C}$. It is a bijection, with inverse $M^{-1} : \mathcal{C} \to \mathbf{R}^{m+n+1}$

$$M^{-1}(u, v) = u - v.$$

(See, e.g., [36, Corollary 31.5.1] or [3, Remark 23.23(i)].) Since Π is (firmly) nonexpansive, we conclude that M is Lipschitz continuous with constant 1.

Using this parametrization of $\mathcal{C}$, we can express the self-dual embedded conditions (5) in terms of z as

$$-\Pi^* z = Q\Pi z, \quad z_{m+n+1} \neq 0. \tag{6}$$

We slightly stretch our notation and say that $z \in \mathbf{R}^{m+n+1}$ is a solution of the homogenous self-dual embedding (5) if $M(z)$ is a solution.

Residual map We define the *residual map* $\mathcal{R} : \mathbf{R}^{m+n+1} \to \mathbf{R}^{m+n+1}$ by

$$\mathcal{R}(z) = Q\Pi z + \Pi^* z = ((Q - I)\Pi + I)z. \tag{7}$$

The map $\mathcal{R}$ is positively homogenous and differentiable almost everywhere (see "Appendix A").

Normalized residual The *normalized residual* map $\mathcal{N} : \{z \in \mathbf{R}^{m+n+1} \mid z_{m+n+1} \neq 0\} \to \mathbf{R}^{m+n+1}$ is given by

$$\mathcal{N}(z) = \mathcal{R}(z/|w|) = \mathcal{R}(z)/|w|, \tag{8}$$

where we use w to denote z_{m+n+1} to lighten the notation. The second equality follows from positive homogeneity of $\mathcal{R}$. Note that by (6), if $z \in \mathbf{R}^{m+n+1}$ satisfies the self-dual embedding (5), then $\mathcal{N}(z) = 0$. Conversely, if $\mathcal{N}(z) = 0$, then z satisfies the self-dual embedding (5). The idea of formulating the homogeneous self-dual embedding problem as finding a zero of a mapping has been used in other work, e.g., [44].

For $z \in \mathbf{R}^{m+n+1}$, with $w = z_{m+n+1} \neq 0$, we use $\|\mathcal{N}(z)\|_2$ as a practical measure of the suboptimality of z, i.e., how far z deviates from being a solution of (5). We refer to $\|\mathcal{N}(z)\|_2$ as the *normalized residual norm* of a candidate z.

Derivatives of residual and normalized residual maps Let $z \in \mathbf{R}^{m+n+1}$ be such that Π is differentiable at z. Then $\mathcal{R}$ is differentiable at z, with derivative

$$\mathsf{D}\mathcal{R}(z) = (Q - I)\mathsf{D}\Pi(z) + I. \tag{9}$$

Now suppose $z \in \mathbf{R}^{m+n+1}$, with $w \neq 0$. In view of (8) and (9), $\mathcal{N}$ is differentiable at z, with derivative

$$\begin{aligned}\mathsf{D}\mathcal{N}(z) = \frac{\mathsf{D}\mathcal{R}(z)}{|w|} - \mathbf{sign}(w)\frac{\mathcal{R}(z)}{w^2}e^T = \frac{(Q - I)\mathsf{D}\Pi(z) + I}{|w|} \\ - \mathbf{sign}(w)\frac{((Q - I)\Pi + I)z}{w^2}e^T,\end{aligned}$$

where $e = (0, \ldots, 0, 1) \in \mathbf{R}^{m+n+1}$, and we remind the reader that $w = z_{m+n+1}$.

3 Cone projections and matrix-free derivative evaluations

Here we consider some of the standard cones used in practice, and for each one, describe the projection, and also how to evaluate its derivative mapping and its adjoint efficiently. Many of these results have appeared in other works [1,20,25,31]. We give them here for completeness, to put them in a common notation, and to point out how the derivative mappings can be efficiently evaluated.

Cartesian product of cones In many cases of interest the cone $\mathcal{K}$ is a Cartesian product of simpler closed convex cones, i.e., $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_p$. The projection Π onto $\mathcal{K}$ is evidently the Cartesian product of the projections, $\Pi = \Pi_{\mathcal{K}_1} \times \cdots \times \Pi_{\mathcal{K}_p}$. It is clear that Π is differentiable at $x = (x_1, \ldots, x_p)$ if and only if each $\Pi_{\mathcal{K}_i}$ is differentiable at x_i, and its derivative is

$$\mathsf{D}\Pi = \mathsf{D}\Pi_{\mathcal{K}_1} \times \cdots \times \mathsf{D}\Pi_{\mathcal{K}_p}.$$

(When the derivative is represented as a matrix, the Cartesian product here is a block diagonal matrix.)

So it suffices to discuss the simpler cones, where for simplicity of notation, we drop the subscript and refer to the (smaller) cones as $\mathcal{K}$.

The computational cost of the projection, and evaluating its derivative, on a Cartesian product of cones is the sum of the costs of the operations carried out on each of the individual cones. Also, these operations can be easily parallelized.

Zero and free cones For $\mathcal{K} = \{0\}$, we have $\Pi x = 0$; Π is differentiable everywhere and, for every $x \in \mathbf{R}$, $\mathsf{D}\Pi(x) = 0$. For $\mathcal{K} = \mathbf{R}$, $\Pi x = x$; Π is differentiable everywhere and, for every $x \in \mathbf{R}$, $\mathsf{D}\Pi(x) = 1$.

Nonnegative cone For the nonnegative cone $\mathbf{R}_+$ the projection is given by $\Pi x = \max(x, 0)$. It is differentiable for $x \neq 0$, with derivative $\mathsf{D}\Pi(x) = \frac{1}{2}(\mathbf{sign}(x) + 1)$.

Second-order cone For the second-order cone (also known as the Lorentz cone)

$$\mathcal{K} = \{(t, x) \in \mathbf{R}_+ \times \mathbf{R}^n \mid \|x\|_2 \leq t\},$$

we have

$$\Pi(t, y) = \begin{cases} (t, y), & \text{if } \|y\|_2 \leq t \\ (0, 0), & \text{if } \|y\|_2 \leq -t \\ \frac{t+\|y\|_2}{2}\big(1, y/\|y\|_2\big), & \text{otherwise.} \end{cases}$$

It follows from [20, Lemma 2.5] that Π is differentiable at (t, x) whenever $\|x\|_2 \neq |t|$, in which case we have

$$\mathsf{D}\Pi(t, x) = \begin{cases} I, & \text{if } \|x\|_2 < t \\ 0, & \text{if } \|x\|_2 < -t \\ \frac{1}{2\|x\|_2}\begin{bmatrix} \|x\|_2 & x^T \\ x & (t + \|x\|_2)I - t\frac{x}{\|x\|_2}\frac{x^T}{\|x\|_2} \end{bmatrix}, & \text{otherwise.} \end{cases} \tag{10}$$

(Note that $\mathsf{D}\Pi$ is symmetric, a consequence of $\mathcal{K}$ being self-dual.) Evaluating $\mathsf{D}\Pi(t, x)(\tilde{t}, \tilde{x})$ efficiently is easy in the first two cases. In the third case one does not form the matrix, but rather evaluates it by computing $x^T\tilde{x}$, and then forming the resulting vector using vector operations. This requires a number of flops (floating point operations) that is linear in the dimension of the cone, as opposed to quadratic.

Semidefinite cone The semidefinite cone $\mathbf{S}^n_+$ is the cone of $n \times n$ positive semidefinite matrices. Let $X \in \mathbf{S}^n$ (the set of symmetric $n \times n$ matrices) and let

$$X = U\mathbf{diag}(\lambda)U^T$$

be its eigendecomposition, with $U^T U = I$ and λ is the vector of eigenvalues of X. The projection of X onto $\mathbf{S}^n_+$ is given by

$$\Pi X = U\mathbf{diag}(\lambda_+)U^T, \tag{11}$$

where $\lambda_+ = \max(\lambda, 0)$ (elementwise). It is known that (see, e.g., [25, Theorem 2.7]) Π is differentiable at X whenever $\det X \neq 0$. For $\det X \neq 0$, let $\mathsf{D}\Pi(X) : \mathbf{S}^n \to \mathbf{S}^n$ be the derivative of Π at X, and let $\widetilde{X} \in \mathbf{S}^n$. Then (see [25, Theorem 2.7], [9, Proposition 4.2] and also "Appendix B" below)

$$\mathsf{D}\Pi(X)(\widetilde{X}) = U(B \circ (U^T \widetilde{X} U))U^T, \tag{12}$$

where $\circ$ denotes the Hadamard (i.e., entrywise) product, and the symmetric matrix B is given by

$$B_{ij} = \begin{cases} 0, & \text{if } i \leq k,\ j \leq k; \\ \frac{(\lambda_+)_i}{(\lambda_-)_j + (\lambda_+)_i}, & \text{if } i > k,\ j \leq k; \\ \frac{(\lambda_+)_j}{(\lambda_-)_i + (\lambda_+)_j}, & \text{if } i \leq k,\ j > k; \\ 1, & \text{if } i > k,\ j > k, \end{cases}$$

where k is the number of negative eigenvalues of X. (In passing, we point out that (10) and (12) can be obtained as special cases of [40, Theorem 3.2].) This derivative is symmetric (self-adjoint) and is readily evaluated at $\tilde{X}$ using matrix–matrix products, which cost order n^3 flops. If we represent $X \in \mathbf{S}^n$ as a vector $x \in \mathbf{R}^m$, with $m = n(n+1)/2$, the cost is order $m^{3/2}$.

The exponential cone The exponential cone is given by

$$\mathcal{K} = \{(x, y, z) \in \mathbf{R}^3 \mid y\, e^{x/y} \leq z,\ y > 0\} \ \cup \ \mathbf{R}_- \times \{0\} \times \mathbf{R}_+,$$

and its dual cone is given by

$$\mathcal{K}^* = \{(u, v, w) \in \mathbf{R}^3 \mid u < 0,\ -ue^{v/u} \leq ew\} \ \cup \ \{0\} \times \mathbf{R}_+ \times \mathbf{R}_+.$$

We have four cases (see [31, §6.3.4]):

- Case 1: $(x, y, z) \in \mathcal{K}$. Then $\Pi(x, y, z) = (x, y, z)$.
- Case 2: $(x, y, z) \in -\mathcal{K}^* \setminus \{(0, 0, 0)\}$. Then $\Pi(x, y, z) = (0, 0, 0)$.
- Case 3: $x < 0$ and $y < 0$. Then $\Pi(x, y, z) = (x, 0, \max(z, 0))$.
- Case 4: Otherwise, we have $\Pi(x, y, z) = (x^*, y^*, z^*)$, where (x^*, y^*, z^*) is the unique solution of

$$\begin{array}{l} \text{minimize } \|(x, y, z) - (\overline{x}, \overline{y}, \overline{z})\|_2^2 \\ \text{subject to } \overline{z} = \overline{y} e^{\overline{x}/\overline{y}},\ \overline{y} > 0. \end{array} \tag{13}$$

The optimization problem (13) can be solved by a primal–dual Newton method (see [31, §6.3.4]). (The existence and uniqueness of (x^*, y^*, z^*) follow from the fact that $\mathcal{K}$ is closed and convex.) The derivative $\mathsf{D}\Pi$ is given in the following cases:

- Case 1: $(x, y, z) \in \mathbf{int}\, \mathcal{K}$. Then $\mathsf{D}\Pi(x, y, z) = (x, y, z)$.
- Case 2: $(x, y, z) \in -\mathbf{int}\, \mathcal{K}^*$. Then $\mathsf{D}\Pi(x, y, z) = (0, 0, 0)$.
- Case 3: $x < 0$, $y < 0$ and $z \neq 0$. Then $\mathsf{D}\Pi(x, y, z) = (1, 0, \frac{1}{2}(1 + \mathrm{sign}(z)))$.
- Case 4: $(x, y, z) \in \mathbf{int}\, (\mathbf{R}^3 \setminus (\mathcal{K} \cup \mathcal{K}^* \cup \mathbf{R}_- \times \mathbf{R}_- \times \mathbf{R}))$. See the system of equations (26) below.

4 Refinement

Suppose that $\hat{z} \in \mathbf{R}^{m+n+1}$, $\hat{z}_{m+n+1} \neq 0$, is a given approximate solution of the self-dual embedding (5), by which we mean that it has a small positive normalized residual norm $\|\mathcal{N}(\hat{z})\|_2$. Our goal is to refine the approximate solution, i.e., to produce a vector δ for which

$$\|\mathcal{N}(\hat{z} + \delta)\|_2 < \|\mathcal{N}(\hat{z})\|_2 \tag{14}$$

(and, implicitly, $\hat{z}_{m+n+1} + \delta_{m+n+1} \neq 0$). We refer to $z = \hat{z} + \delta$ as the refined (approximate) solution.

Related work Refinement of an approximate solution of an optimization problem is a very old idea; in linear programming, for example, it relies on guessing the active set and then solving a set of linear equations, see, e.g., [27, §16.5]. In more general conic problems, it was introduced in, e.g., [13]. Refinement is used in numerical solvers, such as the quadratic programming solver OSQP [38], where it is called polishing.

Refinement approach Our approach to refinement requires the assumptions that Π is differentiable at $\hat{z}$, hence $\mathcal{N}$ is differentiable at $\hat{z}$, and that $\mathsf{D}\mathcal{N}(\hat{z})$ is invertible. In other words, the normalized residual mapping is regular at the point $\hat{z}$. While this condition need not hold, it does in many practical cases.

With our assumption, $\|\mathcal{N}(z)\|_2^2$ is differentiable at $\hat{z}$. The condition (14) holds for $\|\delta\|$ sufficiently small, whenever δ is a *descent direction* of $\|\mathcal{N}(z)\|_2^2$ at $\hat{z}$, i.e.,

$$\delta^T (\mathsf{D}\mathcal{N}(\hat{z}))^T \mathcal{N}(\hat{z}) < 0. \tag{15}$$

Such a descent direction always exists, by our assumption that $\hat{z}$ is not a solution (i.e., $\mathcal{N}(\hat{z}) \neq 0$) and $\mathsf{D}\mathcal{N}(\hat{z})$ is invertible; for example, $\delta = -t(\mathsf{D}\mathcal{N}(\hat{z}))^T \mathcal{N}(\hat{z})$, with $t > 0$ sufficiently small. (This is the negative gradient descent direction.) There are many other ways to choose δ for which the descent condition (15) holds. We now propose a specific method to find a descent direction δ.

Levenberg–Marquardt refinement The Levenberg–Marquardt nonlinear least squares method uses the direction δ that minimizes

$$\|\mathcal{N}(\hat{z}) + \mathsf{D}\mathcal{N}(\hat{z})\delta\|_2^2 + \lambda\|\delta\|_2^2, \tag{16}$$

where $\lambda > 0$ is a regularization parameter (see, e.g., [8,26,46]). Many other methods for constructing a descent method could be used, for example Newton-type methods; see, e.g., [18,27,34,35] and the references therein.

Levenberg–Marquardt refinement with truncated LSQR Finding a δ that minimizes (16) is a least squares problem; we propose to use the iterative algorithm LSQR [33], a variant of the conjugate gradient method, to find an approximate minimizer. Specifically, we run the LSQR algorithm for some number of steps, and use the resulting δ as our descent direction. Because $(\mathsf{D}\mathcal{N}(\hat{z}))^T\mathcal{N}(\hat{z}) \neq 0$, it can be shown that δ obtained using this truncated LSQR method is a descent direction, i.e., (15) holds; see [22].

We have two motivations for using LSQR to compute δ. First, LSQR does not require forming the matrix $\mathsf{D}\mathcal{N}(\hat{z})$; it simply requires us to multiply a vector by it, and its transpose. This gives us all the computational advantages described in the previous section. The second reason is that with relatively few iterations of LSQR, a quite good descent direction is typically found.

Line search We take as our refined approximate solution $\hat{z}+t\delta$, where the step-size $t > 0$ is obtained via *backtracking line search* [7, §9.2]. Specifically, we choose $t = 2^{-p}$ where p is the smallest nonnegative integer, not exceeding a given maximum $K > 0$, for which $\|\mathcal{N}(\hat{z}+t\delta)\|_2 < \|\mathcal{N}(\hat{z})\|_2$ holds (and, implicitly, $\hat{z}_{m+n+1} + t\delta_{m+n+1} \neq 0$). When $\|\mathcal{N}(\hat{z}+t\delta)\|_2 < \|\mathcal{N}(\hat{z})\|_2$ fails to hold for $t = 2^{-K}$, we exit the refinement process, using $t = 0$, i.e., $z = \hat{z}$. We refer to this as a case of failed refinement. We find that $K = 10$ is a good choice in practice; in almost all cases, far fewer backtracking steps are needed to produce a refined point.

Iterated refinement The refinement method described above can be iterated, provided that at each of the points produced, $\mathcal{N}$ is regular. One might even imagine that iterated refinement could be used to solve a conic problem, by starting from some arbitrary point with large normalized residual, and iteratively refining it. But we note that the regularity condition on the iterates need not hold, and even when they do, the method can fail to converge to a solution, and even when they converge to a solution, the convergence can be very slow. For these reasons we cannot recommend iterated refinement as a general method for solving a conic problem. We propose refinement, and iterated refinement, as nothing more than a method that can, and often does, produce a more accurate approximate solution, given an approximate solution produced by another method.

Refinement algorithm parameters Our refinement method has only a few parameters: the number of LSQR iterations to carry out to determine the descent direction δ, the maximum number of backtracking steps in the line search, the regularization parameter λ, and the number of steps of refinement. Default values such as 30 LSQR iterations, 10 backtracking steps, $\lambda = 10^{-8}$, and 2 steps of iterated refinement seem to provide very good results across a variety of problem instances.

Computational complexity Here we summarize the computational complexity of our refinement method. Each LSQR iteration requires one matrix vector multiplication by Q, one by its transpose, one by $\mathsf{D}\mathcal{N}$, one by its transpose, and a few vector operations. Each evaluation of the residual function requires a multiplication by Q, one evaluation

of $\Pi_{\mathcal{K}}$, and a few vector operations. These operations have costs that depend on the format of A (e.g., dense or sparse) and the size and types of the cones that form $\mathcal{K}$. Using the default parameter values of 30 LSQR iterations and 2 steps of iterated refinement, we have 60 (or fewer) LSQR iterations and between 3 and 20 evaluations of the normalized residual function.

More specific estimates depend on the structure of A and $\mathcal{K}$. If A is sparse with $\mathbf{nnz}(A)$ nonzero entries, multiplications by Q and Q^T cost roughly $\approx \mathbf{nnz}(A) + n + m$ flops. Perhaps the most expensive evaluations of the normalized residual, and its derivative, occur with $\mathcal{K}$ a single semidefinite cone. In this case, the projection $\Pi_{\mathcal{K}}$ and multiplications by $\mathrm{D}\mathcal{N}$ and its transpose require a number of flops proportional to $m^{3/2}$.

Reference implementation This paper is accompanied by a reference implementation, written in Python and available at

https://github.com/cvxgrp/cone_prog_refine.

The code implements the refinement method described in this paper, using Numpy [30], Scipy [19], and Numba [28], a just-in-time compiler, to run faster. Refining an approximate solution requires a single function call, with the problem data expressed in the same way used by the open source solvers ECOS [11] and SCS [29]: A is a sparse compressed-column matrix, b and c are Numpy arrays. The cone $\mathcal{K}$ is the Cartesian product of a sequence of zero, nonnegative, second order, semi-definite, and primal or dual exponential cones. We represent it with a Python dictionary containing the dimensions of the component cones.

5 Numerical experiments

We test the refinement algorithm on a variety of randomly generated problems, including feasible, infeasible, and unbounded problems. We report results obtained on an (Apple) laptop with a 2.7 GHz quad-core processor and 16 Gb of RAM. The Python environment used is the Anaconda distribution (with Python 3.7, Numpy 1.15, Scipy 1.1, and Numba 0.36). All the experiments can be reproduced by running the `experiments.ipynb` notebook (in the `examples` folder of the software repository).

Problem generation The random problem instances are generated as follows.

- The cone $\mathcal{K}$ is the Cartesian product of a zero cone of size random uniform in $\{10, \ldots, 50\}$, a nonnegative cone of size random uniform in $\{20, \ldots, 100\}$, a number of Lorentz cones random uniform in $\{2, \ldots, 100\}$ where each has size random uniform in $\{5, \ldots, 20\}$, a number of semi-definite cones random uniform in $\{5, \ldots, 20\}$ where each has size random uniform in $\{2, \ldots, 10\}$, and a number of primal, and dual, exponential cones both random uniform in $\{2, \ldots, 10\}$. This determines the value of m, and we choose n random uniform in $\{1, \ldots, m\}$. The (empirical) 10th and 90th quantiles of the distribution of n and m are about (100, 1000) and (500, 1500), respectively.

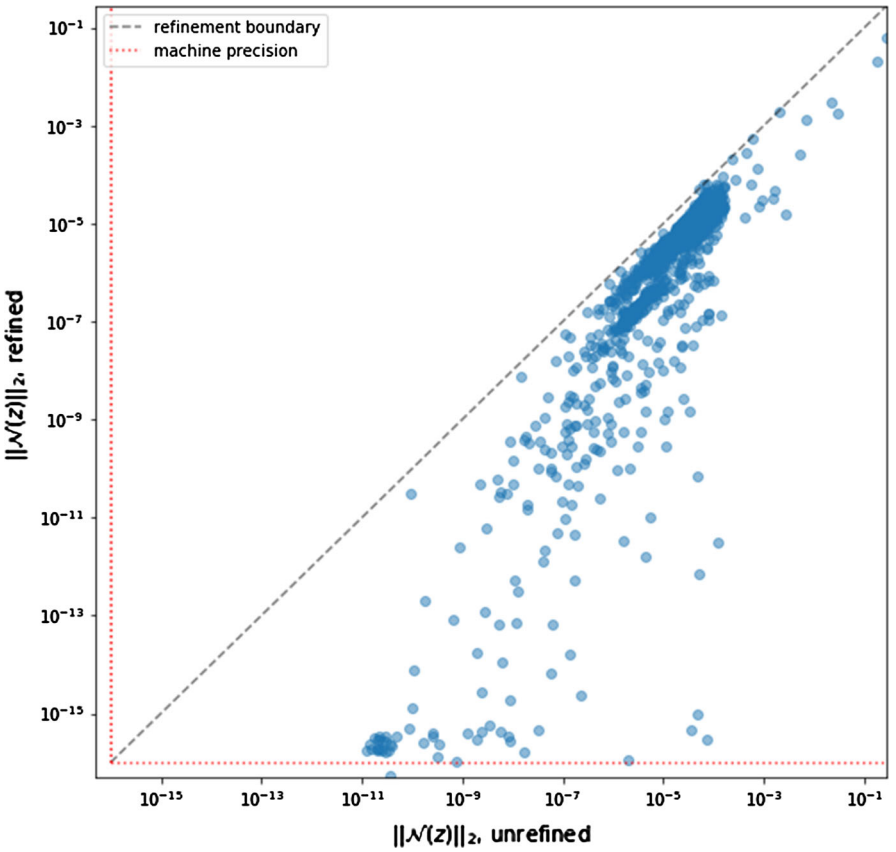

Fig. 1 Residual norm of unrefined and refined approximate solutions, for the experiments of Sect. 5

- The matrix A has a random sparsity pattern with density chosen random uniformly in $[0.1, 0.3]$. Its entries, and the entries of the optimal x and $r = (s - y)$, are chosen random uniformly in $[-1, 1]$. A is then divided by its Frobenius norm, so that $\|A\|_F = 1$. We compute $s = \Pi(r)$, and $y = s - r$. Then, we choose randomly between a feasible, infeasible, or unbounded problem, with probabilities 0.8, 0.1, and 0.1, respectively, and proceed as follows:
  - For a feasible problem instance, we set $b = Ax + s$, and $c = -A^T y$.
  - For an infeasible problem instance, for each column j of A we pick the first nonzero element, if there are any, say A_{ij}. We check that $y_i \neq 0$, and if not choose the next nonzero A_{ij}, and substitute A_{ij} with $A_{ij} - (A^T y)_j / y_i$, so now $A^T y = 0$. We then set $b = -y/\|y\|_2^2$, and c with random uniform entries in $[-1, 1]$.
  - For an unbounded problem instance, for any element x_j of the solution x that is zero, i.e., $x_j = 0$, we set $x_j = 1$. Then, for each row i of A, we pick the first nonzero element, say A_{ij}, or, if there are none, A_{i1}. We substitute A_{ij} with

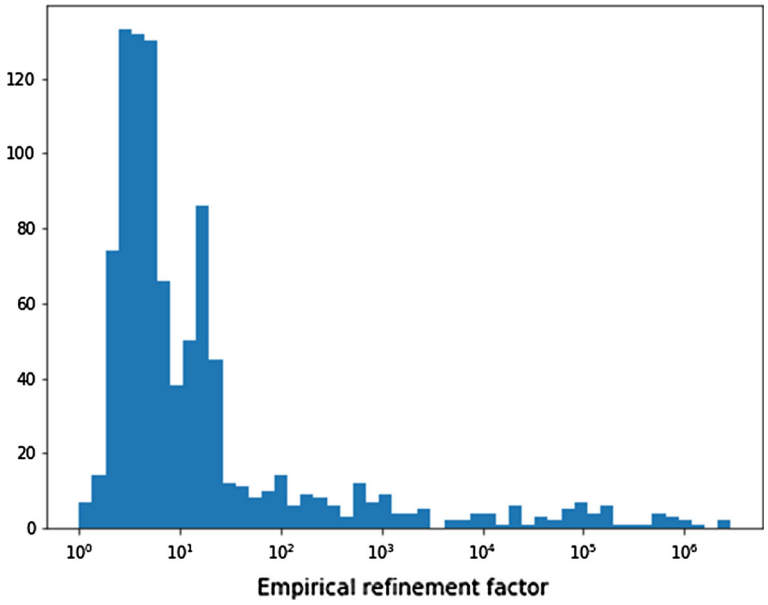

Fig. 2 Distribution of refinement factor $\|\mathcal{N}(z_{\text{unref}})\|_2/\|\mathcal{N}(z_{\text{ref}})\|_2$ over the experiments of Sect. 5

$A_{ij} - (Ax + s)_i/x_j$, so now $Ax + s = 0$. We then set $c = -x/\|x\|_2^2$, and b with random uniform entries in $[-1, 1]$.

Experiments We generate 1000 such problems (783 are solvable, 106 are infeasible, and 111 are unbounded), and for each we obtain an approximate solution (or a certificate of infeasibility or unboundedness) with the numerical solver SCS (if the cone includes semi-definite or exponential cones), or ECOS (otherwise). We then pass the approximate solution returned by the solver to the refinement algorithm, and obtain a refined approximate solution. We use default parameters for both the solvers and the refinement algorithm.

Results Figure 1 shows the scatter plot, for each problem, of $\|\mathcal{N}(z_{\text{unref}})\|_2$ against $\|\mathcal{N}(z_{\text{ref}})\|_2$, where z_{unref} is the approximate solution returned by the solver and z_{ref} is the refined solution returned by the refinement algorithm. We see that the refined solutions always have smaller residual norm than the unrefined ones, and sometimes significantly so. (More details can be seen in the `experiments.ipynb` notebook in the software repository.) Figure 2 shows the distribution of the *refinement factor* $\|\mathcal{N}(z_{\text{unref}})\|_2/\|\mathcal{N}(z_{\text{ref}})\|_2$, i.e., the change in solution quality before and after refinement. We see that in most cases the refinement algorithm improves the solution quality by about an order of magnitude, and sometimes by a few orders of magnitude. The geometric mean of the refinement factor over our 1000 problems is around 30. The three classes of problems (solvable, infeasible, and unbounded) have similar distributions.

Timing Using the default parameters, the time required for refinement of the 1000 example problems is either insignificant compared to the original solver, or comparable in some cases. For very small problems, for which the base solvers ECOS and SCS are very fast (in the few millisecond range), our current Python implementation of the refinement method requires (relatively) more time, but a C implementation of refinement would rectify this.

Acknowledgements The authors thank Yinyu Ye, Micheal Saunders, Nicholas Moehle, and Steven Diamond for useful discussions.

Appendix A

Differentiability properties of the residual map Let C be a nonempty closed convex subset of $\mathbf{R}^n$. It is well known that the projection Π_C onto C is (firmly) nonexpansive (see, e.g., [5, Proposition 2]), hence it is Lipschitz continuous with a Lipschitz constant at most 1. Consequently, if $A : \mathbf{R}^n \to \mathbf{R}^m$ is linear then the composition $A \circ \Pi_C$ is also Lipschitz continuous. Therefore, by the Rademacher Theorem (see, e.g., [37, Theorem 9.60] or [12, Theorem 3.2]) both Π_C and $A \circ \Pi_C$ are differentiable almost everywhere. This allows us to conclude that the residual map (7) is differentiable almost everywhere. Moreover, let $z \in \mathbf{R}^{m+n+1}$. Clearly $\mathcal{R}$ is differentiable at z if Π is differentiable at z.

Appendix B

Semi-definite cone projection derivative

Let $X \in \mathbf{S}^n$, let $X = U\mathbf{diag}(\lambda)U^T$ be an eigendecomposition of X and suppose that $\det(X) \neq 0$. Without loss of generality, we can and do assume that the entries of λ are in an increasing order. That is, there exists $k \in \{1, \ldots, n\}$ such that

$$\lambda_1 \leq \cdots \leq \lambda_k < 0 < \lambda_{k+1} \leq \cdots \leq \lambda_n. \tag{17}$$

We also note that

$$\Pi X - X = U\mathbf{diag}(\lambda_-)U^T, \tag{18}$$

where $\lambda_- = -\min(\lambda, 0)$. It follows from (11), (18), and the orthogonality of U that

$$U^T \Pi X U = \mathbf{diag}(\lambda_+), \quad U^T(\Pi X - X)U = \mathbf{diag}(\lambda_-). \tag{19}$$

Note that

$$\Pi X(\Pi X - X) = U\mathbf{diag}(\lambda_+)\mathbf{diag}(\lambda_-)U^T = 0. \tag{20}$$

Let $\mathsf{D}\Pi(X) : \mathbf{S}^n \to \mathbf{S}^n$ be the derivative of Π at X, and let $\widetilde{X} \in \mathbf{S}^n$. We now show that (12) holds.

Indeed, using the first order Taylor approximation of Π around X, for $\Delta X \in \mathbf{S}^n$ such that $\|\Delta X\|_F$ is sufficiently small (here $\|\cdot\|_F$ denotes the Frobenius norm) we

have

$$\Pi(X + \Delta X) \approx \Pi X + \mathsf{D}\Pi(X)(\Delta X). \tag{21}$$

To simplify the notation, we set $\Delta Y = \mathsf{D}\Pi(X)(\Delta X)$. Now

$$\begin{aligned}
0 &= \Pi(X + \Delta X)(\Pi(X + \Delta X) - X - \Delta X) && \text{(22a)}\\
&\approx (\Pi X + \Delta Y)(\Pi X + \Delta Y - X - \Delta X) && \text{(22b)}\\
&= \Pi X(\Pi X - X) + \Delta Y(\Pi X - X) + \Pi X(\Delta Y - \Delta X) + \Delta Y(\Delta Y - \Delta X)\\
&\approx \Pi X(\Delta Y - \Delta X) + \Delta Y(\Pi X - X) && \text{(22c)}\\
&\approx U^T \Pi X(\Delta Y - \Delta X)U + U^T \Delta Y(\Pi X - X)U && \text{(22d)}\\
&= (U^T \Pi X U)U^T(\Delta Y - \Delta X)U + U^T \Delta Y U(U^T(\Pi X - X)U) && \text{(22e)}\\
&= \mathbf{diag}(\lambda_+)U^T(\Delta Y - \Delta X)U + U^T \Delta Y U(\mathbf{diag}(\lambda_-)). && \text{(22f)}
\end{aligned}$$

Here, (22a) follows from applying (20) with X replaced by $X + \Delta X$, (22b) follows from combining (22a) and (21), (22c) follows from (20) by neglecting second order terms, (22d) follows from multiplying (22c) from the left by U^T and from the right by U, (22e) follows from the fact that $UU^T = I$ and finally (22f) follows from (19). We rewrite the Sylvester [17,42] Eq. (22f) as

$$\mathbf{diag}(\lambda_+)U^T \Delta Y U + U^T \Delta Y U \mathbf{diag}(\lambda_-) \approx \mathbf{diag}(\lambda_+)U^T \Delta X U. \tag{23}$$

Using (23), we learn that for any $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, n\}$, we have

$$((\lambda_-)_j + (\lambda_+)_i)(U^T \Delta Y U)_{ij} \approx (\lambda_+)_i (U^T \Delta X U)_{ij}.$$

Recalling (17), if $i \leq k$, $j > k$ we have $(\lambda_-)_j = (\lambda_+)_i = 0$. Otherwise, $(\lambda_-)_j + (\lambda_+)_i \neq 0$ and

$$(U^T \Delta Y U)_{ij} \approx \underbrace{\frac{(\lambda_+)_i}{(\lambda_-)_j + (\lambda_+)_i}}_{=B_{ij}} (U^T \Delta X U)_{ij}. \tag{24}$$

Proceeding by cases in view of (17), and using that ΔY is symmetric (so is $U^T \Delta Y U$), we conclude that

$$B_{ij} = \begin{cases} 0, & \text{if } i \leq k,\ j \leq k;\\ \frac{(\lambda_+)_i}{(\lambda_-)_j + (\lambda_+)_i}, & \text{if } i > k,\ j \leq k;\\ \frac{(\lambda_+)_j}{(\lambda_-)_i + (\lambda_+)_j}, & \text{if } i \leq k,\ j > k;\\ 1, & \text{if } i > k,\ j > k. \end{cases}$$

Therefore, combining with (24) we obtain

$$U^T \Delta Y U \approx B \circ (U^T \Delta X U),$$

where "$\circ$" denotes the Hadamard (i.e., entrywise) product. Recalling the definition of ΔY and using that $UU^T = I$ we conclude that

$$\mathsf{D}\Pi(X)(\Delta X) \approx U(B \circ (U^T \Delta X U))U^T.$$

Letting $\|\Delta X\|_F \to 0$ and applying the implicit function theorem, we conclude that (12) holds.

Appendix C

Exponential cone projection derivative The Lagrangian of the constrained optimization problem (13) is

$$\tfrac{1}{2}\|(x, y, z) - (\overline{x}, \overline{y}, \overline{z})\|^2 + \mu(\overline{y}e^{\overline{x}/\overline{y}} - \overline{z}),$$

where $\mu \in \mathbf{R}$ is the dual variable. The KKT conditions at a solution (x^*, y^*, z^*, μ^*) are

$$\begin{aligned} x^* - x + \mu^* e^{x^*/y^*} &= 0 \\ y^* - y + \mu^* e^{x^*/y^*}\left(1 - \tfrac{x^*}{y^*}\right) &= 0 \\ z^* - z - \mu^* &= 0 \\ y^* e^{x^*/y^*} - z^* &= 0. \end{aligned} \tag{25}$$

Considering the differentials dx, dy, dz and $dx^*, dy^*, dz^*, d\mu^*$ of the KKT conditions in (25), the authors of [1, Lemma 3.6] obtain the system of equations

$$\underbrace{\begin{bmatrix} 1 + \frac{\mu^* e^{x^*/y^*}}{y^*} & -\frac{\mu^* x^* e^{x^*/y^*}}{y^{*2}} & 0 & e^{x^*/y^*} \\ -\frac{\mu^* x^* e^{x^*/y^*}}{y^{*2}} & 1 + \frac{\mu^* x^{*2} e^{x^*/y^*}}{y^{*3}} & 0 & (1 - x^*/y^*)e^{x^*/y^*} \\ 0 & 0 & 1 & -1 \\ e^{x^*/y^*} & (1 - x^*/y^*)e^{x^*/y^*} & -1 & 0 \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} dx^* \\ dy^* \\ dz^* \\ d\mu^* \end{bmatrix}}_{du^*} = \underbrace{\begin{bmatrix} dx \\ dy \\ dz \\ 0 \end{bmatrix}}_{du}. \tag{26}$$

Note that, since (13) is feasible, D is invertible. Therefore, $du^* = D^{-1}(du)$. Consequently, the upper left 3×3 block matrix of D^{-1} is the Jacobian of the projection at (x, y, z) in Case 4.

References

1. Ali, A., Wong, E., Kolter, J.: A semismooth newton method for fast, generic convex programming. In: Proceedings of the 34th International Conference on Machine Learning, pp. 272–279 (2018)

2. Boyd, S., Busseti, E., Diamond, S., Kahn, R., Koh, K., Nystrup, P., Speth, J.: Multi-period trading via convex optimization. Found. Trends Optim. **3**(1), 1–76 (2017)
3. Bauschke, H., Combettes, P.: Convex Analysis and Monotone Operator Theory in Hilbert Spaces, 2nd edn. Springer, Berlin (2017)
4. Busseti, E., Ryu, E., Boyd, S.: Risk-constrained Kelly gambling. J. Invest. **25**(3), 118–134 (2016)
5. Browder, F.: Convergence theorems for sequences of nonlinear operators in Banach spaces. Math. Z. **100**(3), 201–225 (1967)
6. Ben-Tal, A., Nemirovski, A.: Lectures on Modern Convex Optimization. SIAM, Philadelphia (2001)
7. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
8. Boyd, S., Vandenberghe, L.: Introduction to Applied Linear Algebra - Vectors, Matrices, and Least Squares. Cambridge University Press, Cambridge (2018)
9. Chen, X., Qi, H.D., Tseng, P.: Analysis of nonsmooth symmetric-matrix-valued functions with applications to semidefinite complementarity problems. SIAM J. Optim. **13**(4), 960–985 (2003)
10. Diamond, S., Boyd, S.: CVXPY: a Python-embedded modeling language for convex optimization. J. Mach. Learn. Res. **16**(83), 1–5 (2016)
11. Domahidi, A., Chu, E., Boyd, S.: ECOS: an SOCP solver for embedded systems. In: 2013 European Control Conference, pp. 3071–3076. IEEE (2013)
12. Evans, L., Gariepy, R.: Measure Theory and Fine Properties of Functions. CRC Press, Boca Raton (1992)
13. El Ghaoui, L., Lebret, H.: Robust solutions to least-squares problems with uncertain data. SIAM J. Matrix Anal. Appl. **18**(4), 1035–1064 (1997)
14. Fu, A., Narasimhan, B., Boyd, S.: CVXR: an R package for disciplined convex optimization. J. Stat. Softw. (2019) (to appear)
15. Grant, M., Boyd, S.: Graph implementations for nonsmooth convex programs. In: Recent Advances in Learning and Control, Lecture Notes in Control and Information Sciences, pp. 95–110. Springer (2008)
16. Grant, M., Boyd, S.: CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx (2014)
17. Gardiner, J., Laub, A., Amato, J., Moler, C.: Solution of the Sylvester matrix equation $AXB^T + CXD^T = E$. ACM Trans. Math. Softw. **18**(2), 223–231 (1992)
18. Jiang, H.: Global convergence analysis of the generalized Newton and Gauss–Newton methods of the Fischer–Burmeister equation for the complementarity problem. Math. Oper. Res. **24**(3), 529–543 (1999)
19. Jones, E., Oliphant, T., Peterson, P., Others: SciPy: Open source scientific tools for Python. http://www.scipy.org/ (2001). Accessed 4 Mar 2019
20. Kanzow, C., Ferenczi, I., Fukushima, M.: On the local convergence of semismooth Newton methods for linear and nonlinear second-order cone programs without strict complementarity. SIAM J. Optim. **20**(1), 297–320 (2009)
21. Löfberg, J.: YALMIP: A toolbox for modeling and optimization in MATLAB. In: Proceedings of the IEEE International Symposium on Computer Aided Control Systems Design, pp. 284–289 (2004)
22. Lasdon, L., Mitter, S., Waren, A.: The conjugate gradient method for optimal control problems. IEEE Trans. Autom. Control **12**(2), 132–138 (1967)
23. Moreau, J.-J.: Décomposition orthogonale d'un espace hilbertien selon deux cônes mutuellement polaires. Bulletin de la Société Mathématique de France **93**, 273–299 (1965)
24. MOSEK ApS. The MOSEK optimization toolbox for MATLAB manual, version 8.0 (revision 57) (2017)
25. Malick, J., Sendov, H.: Clarke generalized Jacobian of the projection onto the cone of positive semidefinite matrices. Set-Valued Anal. **14**(3), 273–293 (2006)
26. Nash, S.: A survey of truncated-Newton methods. J. Comput. Appl. Math. **124**(1–2), 45–59 (2000)
27. Nocedal, J., Wright, S.: Numerical Optimization. Springer Series in Operations Research and Financial Engineering, 2nd edn. Springer, Berlin (2006)
28. Numba Development Team. Numba. http://numba.pydata.org (2015). Accessed 4 Mar 2019
29. 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)
30. Oliphant, T.: A Guide to NumPy, vol. 1. Trelgol Publishing, Spanish Fork (2006)
31. Parikh, N., Boyd, S.: Proximal algorithms. Found. Trends Optim. **1**(3), 123–231 (2014)

32. Permenter, F., Friberg, H.A., Andersen, E.D.: Solving conic optimization problems via self-dual embedding and facial reduction: a unified approach. SIAM J. Optim, **27**(3), 1257–1282 (2017)
33. Paige, C., Saunders, M.: LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Softw. **8**(1), 43–71 (1982)
34. Qi, L., Sun, J.: A nonsmooth version of Newton's method. Math. Program. **58**(3, Ser. A), 353–367 (1993)
35. Qi, L., Sun.: A survey of some nonsmooth equations and smoothing Newton methods. In: Progress in Optimization, volume 30 of Applied Optimization, pp. 121–146. Kluwer (1999)
36. Rockafellar, R.: Convex Analysis. Princeton University Press, Princeton (1970)
37. Rockafellar, R., Wets, R.: Variational Analysis. Springer, Berlin (1998)
38. Stellato, B., Banjac, G., Goulart, P., Bemporad, A., Boyd, S.: OSQP: an operator splitting solver for quadratic programs. *ArXiv e-prints* (2017)
39. SCS. Splitting conic solve, version 1.1.0. https://github.com/cvxgrp/scs (2015)
40. Sun, D., Sun, J.: Löwner's operator and spectral functions in Euclidean Jordan algebras. Math. Oper. Res. **33**(2), 421–445 (2008)
41. Sturm, J.: Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Optim. Methods Softw. **11**(1–4), 625–653 (1999)
42. Sylvester, J.: Sur l'équation linéaire trinôme en matrices d'un ordre quelconque. Comptes Rendus de l'Académie des Sciences **99**, 527–529 (1884)
43. Taylor, J.: Convex Optim. Power Syst. Cambridge University Press, Cambridge (2015)
44. Themelis, A., Patrinos, P.: SuperMann: a superlinearly convergent algorithm for finding fixed points of nonexpansive operators. IEEE Trans. Autom. Control (2019). https://doi.org/10.1109/TAC.2019.2906393
45. Udell, M., Mohan, K., Zeng, D., Hong, J., Diamond, S., Boyd, S.: Convex optimization in Julia. In: SC14 Workshop on High Performance Technical Computing in Dynamic Languages (2014)
46. Wright, S., Holt, J.: An inexact Levenberg–Marquardt method for large sparse nonlinear least squares. ANZIAM J. **26**(4), 387–403 (1985)
47. Ye, Y., Todd, M., Mizuno, S.: An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. Math. Oper. Res. **19**(1), 53–67 (1994)