J Optim Theory Appl (2016) 169:1042–1068
DOI 10.1007/s10957-016-0892-3

Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding

Brendan O'Donoghue[1] · **Eric Chu**[1] · **Neal Parikh**[2] · **Stephen Boyd**[1]

Received: 24 February 2015 / Accepted: 2 February 2016 / Published online: 22 February 2016

Abstract We introduce a first-order method for solving very large convex cone programs. The method uses an operator splitting method, the alternating directions method of multipliers, to solve the homogeneous self-dual embedding, an equivalent feasibility problem involving finding a nonzero point in the intersection of a subspace and a cone. This approach has several favorable properties. Compared to interior-point methods, first-order methods scale to very large problems, at the cost of requiring more time to reach very high accuracy. Compared to other first-order methods for cone programs, our approach finds both primal and dual solutions when available or a certificate of infeasibility or unboundedness otherwise, is parameter free, and the per-iteration cost of the method is the same as applying a splitting method to the primal or dual alone. We discuss efficient implementation of the method in detail, including direct and indirect methods for computing projection onto the subspace, scaling the original problem data, and stopping criteria. We describe an open-source implementation, which handles the usual (symmetric) nonnegative, second-order, and semidefinite cones as well as the (non-self-dual) exponential and power cones and their duals. We report numerical results that show speedups over interior-point cone solvers for large problems, and scaling to very large general cone programs.

Keywords Optimization · Cone programming · Operator splitting · First-order methods

Mathematics Subject Classification 90C25 · 90C06 · 49M29 · 49M05

✉ Brendan O'Donoghue
bodonoghue85@gmail.com

1 Department of Electrical Engineering, Stanford University, Stanford, CA, USA

2 Department of Computer Science, Stanford University, Stanford, CA, USA

1 Introduction

In this paper we develop a method for solving convex cone optimization problems that can (a) provide primal or dual certificates of infeasibility when relevant and (b) scale to large problem sizes. The general idea is to use a first-order method to solve the homogeneous self-dual embedding of the primal–dual pair; the homogeneous self-dual embedding provides the necessary certificates, and first-order methods scale well to large problem sizes.

The homogeneous self-dual embedding is a single convex feasibility problem that encodes the primal–dual pair of optimization problems. Solving the embedded problem involves finding a nonzero point in the intersection of two convex sets, a convex cone and a subspace. If the original pair is solvable, then a solution can be recovered from any nonzero solution to the embedding; otherwise, a certificate of infeasibility is generated that proves that the primal or dual is infeasible (and the other one unbounded). The homogeneous self-dual embedding has been widely used with interior-point methods [1–3].

We solve the embedded problem with an operator splitting method known as the *alternating direction method of multipliers* (ADMM) [4–7]; see [8] for a recent survey. It can be viewed as a simple variation of the classical alternating projections algorithm for finding a point in the intersection of two convex sets. Roughly speaking, ADMM adds a dual-state variable to the basic method, which can substantially improve convergence. The overall method can reliably provide solutions to modest accuracy after a relatively small number of iterations and can solve large problems far more quickly than interior-point methods. (It may not be suitable if high accuracy is required, due to the slow 'tail convergence' of first-order methods in general, and ADMM in particular [9]). To the best of our knowledge, this is the first application of a first-order method to solving such embeddings. The approach described in this paper combines a number of different ideas that are well established in the literature, such as cone programming and operator splitting methods. We highlight various dimensions along which our method can be compared to others.

Some methods for solving cone programs only return primal solutions, while others can return primal–dual pairs. In addition, some methods can only handle feasible problems, while other methods can also return certificates of infeasibility or unboundedness. The original idea of the homogeneous self-dual embedding is due to Ye et al. [10,11]. Self-dual embeddings have generally been solved via interior-point methods [12], while the literature on other algorithms has generally yielded methods that cannot return certificates of infeasibility; see, e.g., [13–15].

Our approach involves converting a primal–dual pair into a convex feasibility problem involving finding a point in the intersection of two convex sets. There are many projection algorithms that could be used to solve this kind of problem, such as the classical alternating directions method or Dykstra's alternating projections method [16,17], among others [18,19]. For a further discussion of these and many other projection methods, see Bauschke and Koch [20]. Any of these methods could be used to solve the problem in homogeneous self-dual embedding form.

Operator splitting techniques go back to the 1950s; ADMM itself was developed in the mid-1970s [4,5]. Since then a rich literature has developed around ADMM and related methods [6,7,21–28]. Many equivalences exist between ADMM and other operator splitting methods. It was shown in [6] that ADMM is equivalent to the variant of Douglas–Rachford splitting presented in [26] (the original, more restrictive, form of Douglas–Rachford splitting was presented in [29]) applied to the dual problem, which itself is equivalent to Rockafellar's proximal point algorithm [21,30].

Douglas–Rachford splitting is also equivalent to Spingarn's 'method of partial inverses' [31–33] when one of the operators is the normal cone map of a linear subspace [7,34]. In this paper we apply ADMM to a problem where one of the functions is the indicator of a linear subspace, so our algorithm can also be viewed as an application of Spingarn's method. Another closely related technique is the 'split-feasibility problem,' which seeks two points related by a linear mapping, each of which is constrained to be in a convex set [19,35–37].

In [7] and [38] it was shown that equivalences exist between ADMM applied to the primal problem, the dual problem, and a saddle point formulation of the problem; in other words, ADMM is (in a sense) itself self-dual.

These techniques have been used in a broad range of applications including imaging [39–41], control [42–46], estimation [47], signal processing [48–51], finance [52], and distributed optimization [53,54].

There are several different ways to apply ADMM to solve cone programs [8,13]. In some cases, these are applied to the original cone program (or its dual) and yield methods that can return primal–dual pairs, but cannot handle infeasible or unbounded problems.

The indirect version of our method interacts with the data solely by multiplication by the data matrix or its adjoint, which we can informally refer to as a 'scientific computing' style algorithm; it is also called a 'matrix-free method.' There are several other methods that share similar characteristics, such as [55–62], as well as some techniques for solving the split-feasibility problem [35]. See Esser et al. [63] for a detailed discussion of various first-order methods and the relationships between them, and Parikh and Boyd [64] for a survey of proximal algorithms in particular.

Outline In Sect. 2 we review convex cone optimization, conditions for optimality, and the homogeneous self-dual embedding. In Sect. 3, we derive an algorithm (1) that solves convex cone programs using ADMM applied to the homogeneous self-dual embedding. In Sect. 4, we discuss how to perform the substeps of the procedure efficiently. In Sect. 5 we introduce a scaling procedure that greatly improves convergence in practice. We conclude with some numerical examples in Sect. 6, including (when applicable, i.e., the problems are small enough and involve only symmetric cones) a comparison of our approach with state-of-the-art interior-point methods, both in quality of solution and solution time.

2 Conic Optimization

Consider the *primal–dual pair* of (convex) cone optimization problems

$$\begin{array}{ll} \text{minimize } c^{\mathrm{T}}x & \text{maximize } -b^{\mathrm{T}}y \\ \text{s.t. } \; Ax+s=b & \text{s.t. } \; -A^{\mathrm{T}}y+r=c \\ \quad (x,s)\in\mathbb{R}^n\times\mathcal{K}, & \quad (r,y)\in\{0\}^n\times\mathcal{K}^*. \end{array} \tag{1}$$

Here $x \in \mathbb{R}^n$ and $s \in \mathbb{R}^m$ (with $n \leq m$) are the primal variables, and $r \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ are the dual variables. We refer to x as the primal variable, s as the primal slack variable, y as the dual variable, and r as the dual residual. The set $\mathcal{K}$ is a nonempty, closed, convex cone with dual cone $\mathcal{K}^*$, and $\{0\}^n$ is the dual cone of $\mathbb{R}^n$, so the cones $\mathbb{R}^n \times \mathcal{K}$ and $\{0\}^n \times \mathcal{K}^*$ are duals of each other. The problem data are $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and the cone $\mathcal{K}$. (We consider all vectors to be column vectors).

The primal and dual optimal values are denoted $p^\star$ and $d^\star$, respectively; we allow the cases when these are infinite: $p^\star = +\infty$ $(-\infty)$ indicates primal infeasibility (unboundedness), and $d^\star = -\infty$ $(+\infty)$ indicates dual infeasibility (unboundedness). It is easy to show weak duality, i.e., $d^\star \leq p^\star$, with no assumptions on the data. We will assume that strong duality holds, i.e., $p^\star = d^\star$, including the cases when they are infinite.

2.1 Optimality Conditions

When strong duality holds, the KKT (Karush–Kuhn–Tucker) conditions are necessary and sufficient for optimality. Explicitly, $(x^\star, s^\star, r^\star, y^\star)$ satisfies the KKT conditions and so is primal–dual optimal, when

$$Ax^\star + s^\star = b, \quad s^\star \in \mathcal{K}, \quad A^{\mathrm{T}}y^\star + c = r^\star, \quad r^\star = 0, \quad y^\star \in \mathcal{K}^*, \quad (y^\star)^{\mathrm{T}}s^\star = 0,$$

i.e., when $(x^\star, s^\star)$ is primal feasible, $(r^\star, y^\star)$ is dual feasible, and the complementary slackness condition $(y^\star)^{\mathrm{T}}s^\star = 0$ holds. The complementary slackness condition can equivalently be replaced by the condition

$$c^{\mathrm{T}}x^\star + b^{\mathrm{T}}y^\star = 0,$$

which explicitly forces the *duality gap*, $c^{\mathrm{T}}x + b^{\mathrm{T}}y$, to be zero.

2.2 Certificates of Infeasibility

If strong duality holds, then exactly one of the sets

$$\mathcal{P} = \{(x,s) : Ax + s = b, \; s \in \mathcal{K}\}, \tag{2}$$

$$\mathcal{D} = \{y : A^{\mathrm{T}}y = 0, \; y \in \mathcal{K}^*, \; b^{\mathrm{T}}y < 0\}, \tag{3}$$

is nonempty, a result known as a *theorem of strong alternatives* [65, Sect. 5.8]. Since the set $\mathcal{P}$ encodes primal feasibility, this implies that any dual variable $y \in \mathcal{D}$ serves as a *proof* or *certificate* that the set $\mathcal{P}$ is empty, i.e., that the problem is primal infeasible.

Intuitively, the set $\mathcal{D}$ encodes the requirements for the dual problem to be feasible but unbounded.

Similarly, exactly one of the following two sets is nonempty:

$$\tilde{\mathcal{P}} = \{x : -Ax \in \mathcal{K},\ c^{\mathrm{T}}x < 0\}, \tag{4}$$

$$\tilde{\mathcal{D}} = \{y : A^{\mathrm{T}}y = -c,\ y \in \mathcal{K}^*\}. \tag{5}$$

Any primal variable $x \in \tilde{\mathcal{P}}$ is a certificate of dual infeasibility.

2.3 Homogeneous Self-Dual Embedding

The original pair of problems (1) can be converted into a single feasibility problem by embedding the KKT conditions into a single system of equations and inclusions that the primal and dual optimal points must jointly satisfy. The embedding is as follows:

$$\begin{bmatrix} r \\ s \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & A^{\mathrm{T}} \\ -A & 0 \\ c^{\mathrm{T}} & b^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ b \\ 0 \end{bmatrix}, \quad (x, s, r, y) \in \mathbb{R}^n \times \mathcal{K} \times \{0\}^n \times \mathcal{K}^*. \tag{6}$$

Any $(x^\star, s^\star, r^\star, y^\star)$ that satisfies (6) is optimal for (1). However, if (1) is primal or dual infeasible, then (6) has no solution.

The homogeneous self-dual embedding [10] addresses this shortcoming:

$$\begin{bmatrix} r \\ s \\ \kappa \end{bmatrix} = \begin{bmatrix} 0 & A^{\mathrm{T}} & c \\ -A & 0 & b \\ -c^{\mathrm{T}} & -b^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \tau \end{bmatrix}, \quad (x, s, r, y, \tau, \kappa) \in \mathbb{R}^n \times \mathcal{K} \times \{0\}^n \times \mathcal{K}^* \times \mathbb{R}_+ \times \mathbb{R}_+. \tag{7}$$

This embedding introduces two new variables, τ and κ, that are nonnegative and complementary, i.e., at most one is nonzero. To see complementarity, note that the inner product between (x, y, τ) and (r, s, κ) at any solution must be zero due to the skew symmetry of the matrix in (7), and the individual components $x^{\mathrm{T}}r$, $y^{\mathrm{T}}s$, and $\tau\kappa$ must each be nonnegative by the definition of dual cones.

The reason for using this embedding is that the different possible values of τ and κ encode the different possible outcomes. If τ is nonzero at the solution, then it serves as a scaling factor that can be used to recover the solutions to (1); otherwise, if κ is nonzero, then the original problem is primal or dual infeasible. In particular, if $\tau = 1$ and $\kappa = 0$, then the self-dual embedding reduces to the simpler embedding (6).

Any solution of the self-dual embedding $(x, s, r, y, \tau, \kappa)$ falls into one of three cases:

1. $\tau > 0$ and $\kappa = 0$. The point

$$(\hat{x}, \hat{y}, \hat{s}) = (x/\tau, y/\tau, s/\tau)$$

satisfies the KKT conditions of (1) and so is a primal–dual solution.

2. $\tau = 0$ and $\kappa > 0$. This implies that the gap $c^T x + b^T y$ is negative, which immediately tells us that the problem is either primal or dual infeasible.
   - If $b^T y < 0$, then $\hat{y} = y/(b^T y)$ is a certificate of primal infeasibility (i.e., $\mathcal{D}$ is nonempty) since

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

   - If $c^T x < 0$, then $\hat{x} = x/(-c^T x)$ is a certificate of dual infeasibility (i.e., $\tilde{\mathcal{P}}$ is nonempty) since

$$-A\hat{x} \in \mathcal{K}, \quad c^T \hat{x} = -1.$$

   - If both $c^T x < 0$ and $b^T y < 0$, then the problem is both primal and dual infeasible (but the strong duality assumption is violated).
3. $\tau = \kappa = 0$. If one of $c^T x$ or $b^T y$ is negative, then it can be used to derive a certificate of primal or dual infeasibility. Otherwise, nothing can be concluded about the original problem. Note that zero is always a solution to (7), but steps can be taken to avoid it, as we discuss in Sect. 3.4.

The system (7) is homogeneous because if $(x, s, r, y, \tau, \kappa)$ is a solution to the embedding, then so is $(tx, ts, tr, ty, t\tau, t\kappa)$ for any $t \geq 0$, and when $t > 0$ this scaled value yields the same primal–dual solution or certificates for (1). The embedding is also self-dual, which we show below.

Notation To simplify the subsequent discussion, let

$$u = \begin{bmatrix} x \\ y \\ \tau \end{bmatrix}, \quad v = \begin{bmatrix} r \\ s \\ \kappa \end{bmatrix}, \quad Q = \begin{bmatrix} 0 & A^T & c \\ -A & 0 & b \\ -c^T & -b^T & 0 \end{bmatrix}.$$

The homogeneous self-dual embedding (7) can then be expressed as

$$\begin{array}{ll} \text{find} & (u, v) \\ \text{s.t.} & v = Qu \\ & (u, v) \in \mathcal{C} \times \mathcal{C}^*, \end{array} \tag{8}$$

where $\mathcal{C} = \mathbb{R}^n \times \mathcal{K}^* \times \mathbb{R}_+$ is a cone with dual cone $\mathcal{C}^* = \{0\}^n \times \mathcal{K} \times \mathbb{R}_+$. We are interested in finding a nonzero solution of the homogeneous self-dual embedding (8). In the sequel, u_x, u_y, u_τ and v_r, v_s, v_κ will denote the entries of u and v that correspond to x, y, τ and r, s, κ, respectively.

Self-Dual Property Let us show that the feasibility problem (8) is self-dual. The Lagrangian has the form

$$L(u, v, \nu, \lambda, \mu) = \nu^T (Qu - v) - \lambda^T u - \mu^T v,$$

where the dual variables are ν, λ, μ, with $\lambda \in \mathcal{C}^*$, $\mu \in \mathcal{C}$. Minimizing over the primal variables u, v, we conclude that

$$Q^T \nu - \lambda = 0, \quad -\nu - \mu = 0.$$

Eliminating $\nu = -\mu$ and using $Q^T = -Q$ we can write the dual problem as

$$\begin{array}{ll} \text{find} & (\mu, \lambda) \\ \text{s.t.} & \lambda = Q\mu \\ & (\mu, \lambda) \in \mathcal{C} \times \mathcal{C}^*, \end{array}$$

with variables μ, λ. This is identical to (8).

3 Operator Splitting Method

The convex feasibility problem (8) can be solved by many methods, ranging from simple alternating projections to sophisticated interior-point methods. We are interested in methods that scale to very large problems, so we will use an operator splitting method, the alternating direction method of multipliers (ADMM). There are many operator splitting methods (some of which are equivalent to ADMM) that could be used to solve the convex feasibility problem, such as Douglas–Rachford iteration, split-feasibility methods, Spingarn's method of partial inverses, Dykstra's method, and others. While we have not tried these other methods, we suspect that many of them would yield comparable results to ADMM. Moreover, much of our discussion below, on simplifying the iterations and efficiently carrying out the required steps, would also apply to (some) other operator splitting methods.

3.1 Basic Method

ADMM is an operator splitting method that can solve convex problems of the form

$$\text{minimize } [f(x) + g(z)] \;\; \text{s.t.} \;\; x = z. \tag{9}$$

(ADMM can also solve problems where x and z are affinely related; see [8] and the references therein). Here, f and g may be nonsmooth or take on infinite values to encode implicit constraints. The basic ADMM algorithm is

$$\begin{aligned} x^{k+1} &= \operatorname*{argmin}_x \left(f(x) + (\rho/2)\|x - z^k - \lambda^k\|_2^2 \right) \\ z^{k+1} &= \operatorname*{argmin}_z \left(g(z) + (\rho/2)\|x^{k+1} - z - \lambda^k\|_2^2 \right) \\ \lambda^{k+1} &= \lambda^k - x^{k+1} + z^{k+1}, \end{aligned}$$

where $\rho > 0$ is a step-size parameter and λ is the (scaled) dual variable associated with the constraint $x = z$, and the superscript k denotes iteration number. The initial points z^0 and λ^0 are arbitrary, but are usually taken to be zero. Under some very mild conditions [8, Sect. 3.2], ADMM converges to a solution, in the following sense: $f(x^k)+g(z^k)$ converges to the optimal value, λ^k converges to an optimal dual variable, and $x^k - z^k$, the equality constraint residual, converges to zero. Additionally, for the restricted form we consider in (9), we have the stronger guarantee that x^k and z^k converge to a common value [21, Sect. 5]. We will mention later some variations on this basic ADMM algorithm with similar convergence guarantees.

To apply ADMM, we transform the embedding (8) to ADMM form (9):

$$\text{minimize } \left[I_{\mathcal{C}\times\mathcal{C}^*}(u,v) + I_{Qu=v}(\tilde{u},\tilde{v})\right] \text{ s.t. } (u,v) = (\tilde{u},\tilde{v}), \tag{10}$$

where $I_{\mathcal{S}}$ denotes the indicator function [66, Sect. 4] of the set $\mathcal{S}$. A direct application of ADMM to the self-dual embedding, written as (10), yields the following algorithm:

$$\begin{aligned}
(\tilde{u}^{k+1}, \tilde{v}^{k+1}) &= \Pi_{Qu=v}(u^k + \lambda^k, v^k + \mu^k) \\
u^{k+1} &= \Pi_{\mathcal{C}}(\tilde{u}^{k+1} - \lambda^k) \\
v^{k+1} &= \Pi_{\mathcal{C}^*}(\tilde{v}^{k+1} - \mu^k) \\
\lambda^{k+1} &= \lambda^k - \tilde{u}^{k+1} + u^{k+1} \\
\mu^{k+1} &= \mu^k - \tilde{v}^{k+1} + v^{k+1},
\end{aligned} \tag{11}$$

where $\Pi_{\mathcal{S}}(x)$ denotes the Euclidean projection of x onto the set $\mathcal{S}$. Here, λ and μ are dual variables for the equality constraints on u and v, respectively.

3.2 Simplified Method

In this section we show that the basic ADMM algorithm (11) given above can be simplified using properties of our specific problem.

3.2.1 Eliminating Dual Variables

If we initialize $\lambda^0 = v^0$ and $\mu^0 = u^0$, then $\lambda^k = v^k$ and $\mu^k = u^k$ for all subsequent iterations. This result allows us to eliminate the dual variable sequences above. This will also simplify the linear system in the first step and remove one of the cone projections.

Proof The proof is by induction. The base case holds because we can initialize the variables accordingly. Assuming that $\lambda^k = v^k$ and $\mu^k = u^k$, the first step of the algorithm becomes

$$(\tilde{u}^{k+1}, \tilde{v}^{k+1}) = \Pi_{\mathcal{Q}}\left(u^k + \lambda^k, v^k + \mu^k\right) = \Pi_{\mathcal{Q}}\left(u^k + v^k, u^k + v^k\right), \tag{12}$$

where $\mathcal{Q} = \{(u,v) : Qu = v\}$.

The orthogonal complement of $\mathcal{Q}$ is $\mathcal{Q}^\perp = \{(v,u) : Qu = v\}$ because Q is skew-symmetric. It follows that if $(u,v) = \Pi_{\mathcal{Q}}(z,z)$, then $(v,u) = \Pi_{\mathcal{Q}^\perp}(z,z)$ for any z, since the two projection problems are identical save for reversed output arguments. This implies that

$$(\tilde{v}^{k+1}, \tilde{u}^{k+1}) = \Pi_{\mathcal{Q}^\perp}\left(u^k + v^k, u^k + v^k\right). \tag{13}$$

Recall that $z = \Pi_{\mathcal{Q}}(z) + \Pi_{\mathcal{Q}^\perp}(z)$ for any z. With (12) and (13), this gives

$$u^k + v^k = \tilde{u}^{k+1} + \tilde{v}^{k+1}. \tag{14}$$

The *Moreau decomposition* [64, Sect. 2.5] of x with respect to a nonempty, closed, convex cone $\mathcal{C}$ is given by

$$x = \Pi_{\mathcal{C}}(x) + \Pi_{-\mathcal{C}^*}(x), \tag{15}$$

and moreover, the two terms on the right-hand side are orthogonal. It can be written equivalently as $x = \Pi_{\mathcal{C}}(x) - \Pi_{\mathcal{C}^*}(-x)$. Combining this with (14) gives

$$\begin{aligned} u^{k+1} &= \Pi_{\mathcal{C}}(\tilde{u}^{k+1} - v^k) \\ &= \Pi_{\mathcal{C}}(u^k - \tilde{v}^{k+1}) \\ &= u^k - \tilde{v}^{k+1} + \Pi_{\mathcal{C}^*}(\tilde{v}^{k+1} - u^k) \\ &= u^k - \tilde{v}^{k+1} + v^{k+1} \\ &= \mu^{k+1}. \end{aligned}$$

A similar derivation yields $\lambda^{k+1} = v^{k+1}$, which completes the proof. This lets us eliminate the sequences λ^k and μ^k. □

Once the value $u^{k+1} = \Pi_{\mathcal{C}}(\tilde{u}^{k+1} - v^k)$ has been calculated, the step that projects onto the dual cone $\mathcal{C}^*$ can be replaced with

$$v^{k+1} = v^k - \tilde{u}^{k+1} + u^{k+1}.$$

This follows from the λ^k update, which is typically cheaper than a projection step. Now no sequence depends on $\tilde{v}^k$, so it too can be eliminated.

3.2.2 Projection Onto Affine Set

Each iteration, the algorithm (11) computes a projection onto $\mathcal{Q}$ by solving

$$\text{minimize } \left[(1/2)\|u - u^k - v^k\|_2^2 + (1/2)\|v - u^k - v^k\|_2^2\right] \text{ s.t. } v = Qu,$$

with variables u and v. The KKT conditions for this problem are

$$\begin{bmatrix} I & Q^{\mathrm{T}} \\ Q & -I \end{bmatrix} \begin{bmatrix} u \\ \mu \end{bmatrix} = \begin{bmatrix} u^k + v^k \\ u^k + v^k \end{bmatrix}, \tag{16}$$

where $\mu \in \mathbb{R}^{m+n+1}$ is the dual variable associated with the equality constraint $Qu - v = 0$. By eliminating μ, we obtain

$$\tilde{u}^{k+1} = (I + Q^{\mathrm{T}} Q)^{-1}(I - Q)(u^k + v^k).$$

The matrix Q is skew-symmetric, so this simplifies to

$$\tilde{u}^{k+1} = (I + Q)^{-1}(u^k + v^k).$$

(The matrix $I + Q$ is guaranteed to be invertible since Q is skew-symmetric).

3.2.3 Final Algorithm

Combining the simplifications of the previous sections, the final algorithm is

$$\begin{aligned} \tilde{u}^{k+1} &= (I + Q)^{-1}(u^k + v^k) \\ u^{k+1} &= \Pi_{\mathcal{C}}\left(\tilde{u}^{k+1} - v^k\right) \\ v^{k+1} &= v^k - \tilde{u}^{k+1} + u^{k+1}. \end{aligned} \tag{17}$$

The algorithm consists of three steps. The first step is projection onto a subspace, which involves solving a linear system with coefficient matrix $I + Q$; this is discussed in more detail in Sect. 4.1. The second step is projection onto a cone, a standard operation discussed in detail in [64, Sect. 6.3].

The last step is computationally trivial and has a simple interpretation: As the algorithm runs, the vectors u^k and $\tilde{u}^k$ converge to each other, so $u^{k+1} - \tilde{u}^{k+1}$ can be viewed as the error at iteration $k + 1$. The last step shows that v^{k+1} is exactly the running sum of the errors. Roughly speaking, this running sum of errors is used to drive the error to zero, exactly as in integral control [67].

We can also interpret the second and third steps as a combined Moreau decomposition of the point $\tilde{u}^{k+1} - v^k$ into its projection onto $\mathcal{C}$ (which gives u^{k+1}) and its projection onto $-\mathcal{C}^*$ (which gives v^{k+1}).

The algorithm is homogeneous: If we scale the initial points by some factor $\gamma > 0$, then all subsequent iterates are also scaled by γ and the overall algorithm will give the same primal–dual solution or certificates for (1), since the system being solved is also homogeneous.

A straightforward application of ADMM directly to the primal or dual problem in (1) obtains an algorithm which requires one linear system solve involving $A^{\mathrm{T}}A$ and one projection onto the cone $\mathcal{K}$, which has the same per-iteration cost as (17); see, e.g., [13] for details.

3.3 Variations

There are many variants on the basic ADMM algorithm (17) described above, and any of them can be employed with the homogeneous self-dual embedding. We briefly describe two important variations that we use in our reference implementation.

Over-Relaxation In the u- and v-updates, replace all occurrences of $\tilde{u}^{k+1}$ with

$$\alpha\tilde{u}^{k+1} + (1 - \alpha)u^k,$$

where $\alpha \in]0, 2[$ is a relaxation parameter [21,68]. When $\alpha = 1$, this reduces to the basic algorithm given above. When $\alpha > 1$, this is known as *over-relaxation*; when $\alpha < 1$, this is *under-relaxation*. Some numerical experiments suggest that values of α around 1.5 can improve convergence [44,69].

Approximate Projection Another variation replaces the subspace projection update with a suitable approximation [21,30,68]. We replace $\tilde{u}^{k+1}$ in the first line of (17) with any $\tilde{u}^{k+1}$ that satisfies

$$\|\tilde{u}^{k+1} - (I + Q)^{-1}(u^k + v^k)\|_2 \leq \zeta^k, \tag{18}$$

where $\zeta^k > 0$ satisfy $\sum_k \zeta^k < \infty$. This variation is particularly useful when an iterative method is used to compute $\tilde{u}^{k+1}$.

Note that (18) is implied by the (more easily verified) inequality

$$\|(Q + I)\tilde{u}^{k+1} - (u^k + v^k)\|_2 \leq \zeta^k. \tag{19}$$

This follows from the fact that $\|(I + Q)^{-1}\|_2 \leq 1$, which holds since Q is skew-symmetric. The left-hand side of (19) is the norm of the residual in the equations that define $\tilde{u}^{k+1}$ in the basic algorithm.

3.4 Convergence

Algorithm Convergence We show that the algorithm converges, in the sense that it eventually produces a point for which the optimality conditions almost hold. For the basic algorithm (17) and the variant with over-relaxation and approximate projection, for all iterations $k > 0$ we have

$$u^k \in \mathcal{C}, \quad v^k \in \mathcal{C}^*, \quad (u^k)^{\mathrm{T}} v^k = 0. \tag{20}$$

These follow from the last two steps of (17), and hold for any values of v^{k-1} and $\tilde{u}^k$. Since u^{k+1} is a projection onto $\mathcal{C}$, $u^k \in \mathcal{C}$ follows immediately. The condition $v^k \in \mathcal{C}^*$ holds since the last step can be rewritten as $v^{k+1} = \Pi_{\mathcal{C}^*}(v^k - \tilde{u}^{k+1})$, as observed above. The last condition, $(u^k)^{\mathrm{T}} v^k = 0$, holds by our observation that these two points are the (orthogonal) Moreau decomposition of the same point.

In addition to the three conditions in (20), only one more condition must hold for (u^k, v^k) to be optimal: $Qu^k = v^k$. This equality constraint holds asymptotically, i.e., we have, as $k \to \infty$,

$$Qu^k - v^k \to 0. \tag{21}$$

(We show this from the convergence result for ADMM below). Thus, the iterates (u^k, v^k) satisfy three of the four optimality conditions (20) at every step, and the fourth one (21) is satisfied in the limit.

To show that the equality constraint holds asymptotically we use general ADMM convergence theory; see, e.g., [8, Sect. 3.4.3], or [21] for the case of approximate projections. This convergence theory tells us that

$$\tilde{u}^k \to u^k, \quad \tilde{v}^k \to v^k \tag{22}$$

as $k \to \infty$, even with over-relaxation and approximate projection. From the last step in (17) we conclude that $v^{k+1} - v^k \to 0$. From (14), (22), and $v^{k+1} - v^k \to 0$, we obtain $u^{k+1} - u^k \to 0$.

Expanding (19), we have

$$Q\tilde{u}^{k+1} + \tilde{u}^{k+1} - u^k - v^k \to 0,$$

and using (22) we get

$$Qu^{k+1} + u^{k+1} - u^k - v^k \to 0.$$

From $u^{k+1} - u^k \to 0$ and $v^{k+1} - v^k \to 0$ we conclude

$$Qu^k - v^k \to 0,$$

which is what we wanted to show.

Eliminating Convergence to Zero We can guarantee that the algorithm will not converge to zero if a nonzero solution exists, by proper selection of the initial point (u^0, v^0), at least in the case of exact projection.

Denote by $(u^\star, v^\star)$ any nonzero solution to (8), which we assume satisfies either $u^\star_\tau > 0$ or $v^\star_\kappa > 0$, i.e., we can use it to derive an optimal point or a certificate for (1). If we choose initial point (u^0, v^0) with $u^0_\tau = 1$ and $v^0_\kappa = 1$, and all other entries zero, then we have

$$(u^\star, v^\star)^{\mathrm{T}}(u^0, v^0) > 0.$$

Let ϕ denote the mapping that consists of one iteration of algorithm (17), i.e., $(u^{k+1}, v^{k+1}) = \phi(u^k, v^k)$. We show in the appendix that the mapping ϕ is nonexpansive, i.e., for any (u, v) and $(\hat{u}, \hat{v})$ we have that

$$\|\phi(u, v) - \phi(\hat{u}, \hat{v})\|_2 \le \|(u, v) - (\hat{u}, \hat{v})\|_2. \tag{23}$$

(Nonexpansivity holds for ADMM more generally; see, e.g., [6,21,28] for details). Since $(u^\star, v^\star)$ is a solution to (8), it is a fixed point of ϕ, i.e.,

$$\phi(u^\star, v^\star) = (u^\star, v^\star). \tag{24}$$

Since the problem is homogeneous, the point $\gamma(u^\star, v^\star)$ is also a solution for any positive γ, and is also a fixed point of ϕ. Combining this with (23), we have at iteration k

$$\|(u^k, v^k) - \gamma(u^\star, v^\star)\|_2^2 \le \|(u^0, v^0) - \gamma(u^\star, v^\star)\|_2^2, \tag{25}$$

for any $\gamma > 0$. Expanding (25) and setting

$$\gamma = \|(u^0, v^0)\|_2^2/(u^\star, v^\star)^{\mathrm{T}}(u^0, v^0),$$

which is positive by our choice of (u^0, v^0), we obtain

$$2(u^\star, v^\star)^T(u^k, v^k) \geq (u^\star, v^\star)^T(u^0, v^0)(1 + \|(u^k, v^k)\|_2^2/\|(u^0, v^0)\|_2^2),$$

which implies that

$$(u^\star, v^\star)^T(u^k, v^k) \geq (u^\star, v^\star)^T(u^0, v^0)/2,$$

and applying Cauchy–Schwarz yields

$$\|(u^k, v^k)\|_2 \geq (u^\star, v^\star)^T(u^0, v^0)/2\|(u^\star, v^\star)\|_2 > 0. \tag{26}$$

Thus, for $k = 1, 2, \ldots$, the iterates are bounded away from zero.

Normalization The vector given by

$$(\hat{u}^k, \hat{v}^k) = (u^k, v^k)/\|(u^k, v^k)\|_2$$

satisfies the conditions given in (20) for all iterations, and by combining (21) with (26) we have that

$$Q\hat{u}^k - \hat{v}^k \to 0,$$

in the exact projection case at least. In other words, the unit vector $(\hat{u}^k, \hat{v}^k)$ eventually satisfies the optimality conditions for the homogeneous self-dual embedding to any desired accuracy.

3.5 Termination Criteria

In view of the discussion of the previous section, a stopping criterion of the form

$$\|Qu^k - v^k\|_2 \leq \epsilon$$

for some tolerance ϵ, or alternatively a normalized criterion

$$\|Qu^k - v^k\|_2 \leq \epsilon\|(u^k, v^k)\|_2,$$

will work, i.e., the algorithm eventually stops. Here, we propose a different scheme that handles the components of u and v corresponding to primal and dual variables separately. This yields stopping criteria that are consistent with ones traditionally used for cone programming.

We terminate the algorithm when it finds a primal–dual optimal solution or a certificate of primal or dual infeasibility, up to some tolerances. If $u_\tau^k > 0$, then let

$$x^k = u_x^k/u_\tau^k, \quad s^k = v_s^k/u_\tau^k, \quad y^k = u_y^k/u_\tau^k$$

be the candidate solution. This candidate is guaranteed to satisfy the cone constraints and complementary slackness condition by (20). It thus suffices to check that the residuals

$$p^k = Ax^k + s^k - b, \quad d^k = A^T y^k + c, \quad g^k = c^T x^k + b^T y^k,$$

are small. Explicitly, we terminate if

$$\|p^k\|_2 \le \epsilon_{\text{pri}}(1 + \|b\|_2), \quad \|d^k\|_2 \le \epsilon_{\text{dual}}(1 + \|c\|_2), \quad |g^k| \le \epsilon_{\text{gap}}(1 + |c^T x| + |b^T y|)$$

and emit (x^k, s^k, y^k) as (approximately) primal–dual optimal. Here, quantities ϵ_{pri}, ϵ_{dual}, ϵ_{gap} are the primal residual, dual residual, and duality gap tolerances, respectively.

On the other hand, if the current iterates satisfy

$$\left\| Au_x^k + v_s^k \right\|_2 \le \left(-c^T u_x^k / \|c\|_2\right) \epsilon_{\text{unbdd}},$$

then $u_x^k/(-c^T u_x^k)$ is an approximate certificate of unboundedness with tolerance ϵ_{unbdd}, or if they satisfy

$$\left\| A^T u_y^k \right\|_2 \le \left(-b^T u_y^k / \|b\|_2\right) \epsilon_{\text{infeas}},$$

then $u_y^k/(-b^T u_y^k)$ is an approximate certificate of infeasibility with tolerance ϵ_{infeas}.

These stopping criteria are identical to those used by many other cone solvers and similar to those used by DIMACS [70,71] and the Sedumi solver [2].

4 Efficient Subspace Projection

In this section we discuss how to efficiently compute the projection onto the subspace $\mathcal{Q}$, exactly and also approximately (for the approximate variation).

4.1 Solving the Linear System

The first step in (17) is to solve the linear system $(I + Q)\tilde{u}^k = w$ for some w:

$$\begin{bmatrix} I & A^T & c \\ -A & I & b \\ -c^T & -b^T & 1 \end{bmatrix} \begin{bmatrix} \tilde{u}_x \\ \tilde{u}_y \\ \tilde{u}_\tau \end{bmatrix} = \begin{bmatrix} w_x \\ w_y \\ w_\tau \end{bmatrix}. \tag{27}$$

To lighten notation, let

$$M = \begin{bmatrix} I & A^T \\ -A & I \end{bmatrix}, \quad h = \begin{bmatrix} c \\ b \end{bmatrix},$$

so

$$I + Q = \begin{bmatrix} M & h \\ -h^T & 1 \end{bmatrix}.$$

It follows that

$$\begin{bmatrix} \tilde{u}_x \\ \tilde{u}_y \end{bmatrix} = (M + hh^T)^{-1} \left(\begin{bmatrix} w_x \\ w_y \end{bmatrix} - w_\tau h \right),$$

where $M + hh^T$ is the Schur complement of the lower right block 1 in $I + Q$. Applying the Sherman–Morrison–Woodbury formula [72, p. 50] to $(M + hh^T)^{-1}$ yields

$$\begin{bmatrix} \tilde{u}_x \\ \tilde{u}_y \end{bmatrix} = \left(M^{-1} - \frac{M^{-1}hh^T M^{-1}}{(1 + h^T M^{-1} h)} \right) \left(\begin{bmatrix} w_x \\ w_y \end{bmatrix} - w_\tau h \right)$$

and

$$\tilde{u}_\tau = w_\tau + c^T \tilde{u}_x + b^T \tilde{u}_y.$$

Thus, in the first iteration, we compute and cache $M^{-1}h$. To solve (27) in subsequent iterations, it is only necessary to compute $M^{-1}(w_x, w_y)$, which will require the bulk of the computational effort, and then to perform some simple vector operations using cached quantities.

There are two main ways to solve linear equations of the form

$$\begin{bmatrix} I & -A^T \\ -A & -I \end{bmatrix} \begin{bmatrix} z_x \\ -z_y \end{bmatrix} = \begin{bmatrix} w_x \\ w_y \end{bmatrix}, \tag{28}$$

the system that needs to be solved once per iteration. The first method, a *direct method* that exactly solves the system, is to solve (28) by computing a sparse permuted LDL^T factorization [73] of the matrix in (28) before the first iteration and then to use this cached factorization to solve the system in subsequent steps. This technique, called factorization caching, is very effective in the common case when the factorization cost is substantially higher than the subsequent solve cost, so all iterations after the first one can be carried out quickly. Because the matrix is quasi-definite, the factorization is guaranteed to exist for any symmetric permutation [74].

The second method, an *indirect method* that we use to approximately solve the system, involves first rewriting (28) as

$$z_x = (I + A^T A)^{-1}(w_x - A^T w_y), \quad z_y = w_y + A z_x,$$

by elimination. This system is then solved with the conjugate gradient method (CG) [72,75,76]. Each iteration of conjugate gradient requires multiplying once by A and once by A^T, each of which can be parallelized. If A is very sparse, then these multiplications can be performed especially quickly; when A is dense, it may be better to first form $G = I + A^T A$ in the setup phase. We warm-start CG by initializing each subsequent call with the solution obtained by the previous call. We terminate the CG iterations when the residual satisfies (19) for some appropriate sequence ζ^k.

4.2 Repeated Solves

If the cone problem must be solved more than once, then computation from the first solve can be re-used in subsequent solves by warm-starting: We set the initial point to $u^0 = (x^\star, y^\star, 1)$, $v^0 = (0, s^\star, 0)$, where $x^\star, s^\star, y^\star$ are the optimal primal–dual variables from the previous solve. If the data matrix A does not change and a direct method is being used, then the sparse permuted LDL^T factorization can also be reused across solves for additional savings. This arises in many practical situations, such as in control, statistics, and sequential convex programming.

5 Scaling Problem Data

Though the algorithm in (17) has no explicit parameters, the relative scaling of the problem data can greatly affect the convergence. This suggests a preprocessing step where we scale the data to (hopefully) improve the convergence.

In particular, consider scaling vectors b and c by positive scalars σ and ρ, respectively, and scaling the primal and dual equality constraints by diagonal positive definite matrices D and E, respectively. This yields the following scaled primal–dual problem pair:

$$\begin{array}{ll} \text{minimize } \rho(Ec)^T \hat{x} & \text{maximize } -\sigma(Db)^T \hat{y} \\ \text{s.t. } \; DAE\hat{x} + \hat{s} = \sigma Db & \text{s.t. } \; -EA^T D\hat{y} + \hat{r} = \rho Ec \\ \qquad (\hat{x}, \hat{s}) \in \mathbb{R}^n \times \mathcal{K}, & \qquad (\hat{r}, \hat{y}) \in \{0\}^n \times \mathcal{K}^*, \end{array}$$

with variables $\hat{x}$, $\hat{y}$, $\hat{r}$, and $\hat{s}$. After solving this new cone program with problem data $\hat{A} = DAE$, $\hat{b} = \sigma Db$, and $\hat{c} = \rho Ec$, the solution to the original problem (1) can be recovered from the scaled solution via

$$x^\star = E\hat{x}^\star/\sigma, \quad s^\star = D^{-1}\hat{s}^\star/\sigma, \quad y^\star = D\hat{y}^\star/\rho.$$

Transformation by the matrix D must preserve membership of the cone $\mathcal{K}$, to ensure that if $s \in \mathcal{K}$, then $D^{-1}s \in \mathcal{K}$ (the same is not required of E). If $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_q$, where $\mathcal{K}_i \subset \mathbb{R}^{m_i}$, then we could use, for example,

$$D = \mathbf{diag}(\pi_1 I_{m_1}, \ldots, \pi_q I_{m_q}),$$

where each $\pi_i > 0$.

We have observed that in practice, data which has been *equilibrated*, i.e., scaled to have better conditioning, admits better convergence [77–80]. We have found that if the columns of A and b all have Euclidean norm close to one and the rows of A and c have similar norms, then the algorithm (17) typically performs well. The scaling parameters E, D, σ, and ρ can be chosen to (approximately) achieve this [80–82], though the question of whether there is an optimal scaling remains open. There has recently been much work devoted to the question of choosing an optimal, or at least good diagonal scaling; see [83,84].

Scaled Termination Criteria When the algorithm is applied to the scaled problem, it is still desirable to terminate the procedure when the residuals for the *original* problem satisfy the stopping criteria defined in Sect. 3.5.

The original residuals can be expressed in terms of the scaled data as

$$
\begin{aligned}
p^k &= (1/\sigma)D^{-1}(\hat{A}\hat{x}^k + \hat{s}^k - \hat{b}),\\
d^k &= (1/\rho)E^{-1}(\hat{A}^{\mathrm{T}}\hat{y}^k + \hat{c}),\\
g^k &= (1/\rho\sigma)(\hat{c}^{\mathrm{T}}\hat{x}^k + \hat{b}^{\mathrm{T}}\hat{y}^k),
\end{aligned}
$$

and the convergence checks can be applied as before. The stopping criteria for unboundedness and infeasibility then become

$$
\begin{aligned}
\left\| D^{-1}\left(\hat{A}\hat{u}_x^k + \hat{v}_s^k\right)\right\|_2 &\le \left(-\hat{c}^{\mathrm{T}}\hat{u}_x^k/\|E^{-1}\hat{c}\|_2\right)\epsilon_{\mathrm{unbdd}},\\
\left\| E^{-1}\left(\hat{A}^{\mathrm{T}}\hat{u}_y^k\right)\right\|_2 &\le \left(-\hat{b}^{\mathrm{T}}\hat{u}_y^k/\|D^{-1}\hat{b}\|_2\right)\epsilon_{\mathrm{infeas}}.
\end{aligned}
$$

6 Numerical Experiments

In this section we present numerical results for SCS, our implementation of the algorithm described above. We show results on two application problems, in each case instances that are small, medium, and large. We compare the results to SDPT3 [85] and Sedumi [2], state-of-the-art interior-point solvers. We use this comparison for several purposes. First, the solution computed by these solvers is high accuracy, so we can use it to assess the quality of the solution found by SCS. Second, we can compare the computing times. Run-time comparison is not completely fair, since an interior-point method reliably computes a high-accuracy solution, whereas SCS is meant only to compute a solution of modest accuracy and may take longer than an interior-point method if high accuracy is required. Third, Sedumi targets the same homogeneous self-dual embedding (7) as SCS, so we can compare a first-order and a second-order method on the same embedding.

6.1 SCS

Our implementation, which we call SCS for 'splitting conic solver,' is written in C and can solve cone programs involving any combination of nonnegative, second-order, semidefinite, exponential, and power cones (and dual exponential and power cones) [86]. It has multi-threaded and single-threaded versions, and computes the (approximate) projections onto the subspace using either a direct method or an iterative method. SCS is available online at https://github.com/cvxgrp/scs.

along with the code to run the numerical examples. SCS can be used in other C, C++, Python, MATLAB, R, Julia, Java, and Scala programs and is a supported solver in parser-solvers CVX [87], CVXPY [88], Convex.jl [89], and YALMIP [90]. It is now the default solver for CVXPY and Convex.jl for problems that cannot be expressed using the standard symmetric cones.

The direct implementation uses a single-threaded sparse permuted LDL^T decomposition from the SuiteSparse package [73,91,92]. The sparse indirect implementation, which uses conjugate gradient, can perform the matrix multiplications on the CPU or on the GPU. The CPU version uses a basic sparse multiplication routine parallelized using OpenMP [93]. By default, the GPU version uses the sparse CUDA BLAS library [94]. By default, the indirect solver uses $\zeta^k = (1/k)^{1.5}$ as the termination tolerance at iteration k, where the tolerance is defined in (19).

SCS handles the usual nonnegative, second-order, and semidefinite cones, as well as the exponential cone and its dual [64, Sect. 6.3.4],

$$\begin{aligned} K_{\text{exp}} &= \{(x,y,z) : y > 0,\ ye^{x/y} \le z\} \cup \{(x,y,z) : x \le 0,\ y = 0,\ z \ge 0\}, \\ K^*_{\text{exp}} &= \{(u,v,w) : u < 0,\ -ue^{v/u} \le ew\} \cup \{(0,v,w) : v \ge 0,\ w \ge 0\}, \end{aligned}$$

and the power cone and its dual [95–97], defined as

$$\begin{aligned} K^a_{\text{pwr}} &= \{(x,y,z) : x^a y^{(1-a)} \ge |z|,\ x \ge 0,\ y \ge 0\}, \\ \left(K^a_{\text{pwr}}\right)^* &= \{(u,v,w) : (u/a)^a (v/(1-a))^{(1-a)} \ge |w|,\ u \ge 0,\ v \ge 0\}, \end{aligned}$$

for any $a \in [0, 1]$. Projections onto the semidefinite cone are performed using the LAPACK `dsyevr` method for computing the eigendecomposition; projections onto the other cones are implemented in C. The multi-threaded version computes the projections onto the cones in parallel.

In the experiments reported below, we use the termination criteria described in Sects. 3.5 and 5, with the default values

$$\epsilon_{\text{pri}} = \epsilon_{\text{dual}} = \epsilon_{\text{gap}} = \epsilon_{\text{unbdd}} = \epsilon_{\text{infeas}} = 10^{-3}.$$

The objective value reported for SCS in the experiments below is the average of the primal and dual objectives at termination. The time required to do any preprocessing (such as the matrix factorization) and to carry out and undo the scaling are included in the total solve times.

All the experiments were carried out on a system with 32 2.2 GHz cores and 512 GB of RAM, running Linux. (The single-threaded versions, of course, do not make use of the multiple cores). The GPU used was a Geforce GTX Titan X with 12 GB of memory.

6.2 Lasso

Consider the following optimization problem:

$$\text{minimize } (1/2)\|Fz - g\|_2^2 + \mu\|z\|_1, \tag{29}$$

over $z \in \mathbb{R}^p$, where $F \in \mathbb{R}^{q\times p}$, $g \in \mathbb{R}^q$ and $\mu \in \mathbb{R}_+$ are data. This problem, known as the *lasso* [98], is widely studied in high-dimensional statistics, machine learning, and compressed sensing. Roughly speaking, (29) seeks a sparse vector z such that $Fz \approx g$, and the parameter μ trades off between quality of fit and sparsity. It has been observed that first-order methods can perform very well on lasso-type problems when the solution is sparse [99,100].

The lasso problem can be formulated as the SOCP [101]

$$\begin{array}{ll} \text{minimize} & (1/2)w + \mu\mathbf{1}^{\mathrm{T}}t \\ \text{s.t.} & -t \le z \le t, \quad \left\| \begin{array}{c} 1-w \\ 2(Fz-g) \end{array} \right\|_2 \le 1+w \end{array}$$

with variables $z \in \mathbb{R}^p$, $t \in \mathbb{R}^p$ and $w \in \mathbb{R}$. This formulation is easily transformed in turn into the standard form (1).

Problem Instances We generated data for the numerical instances as follows. First, the entries of F were sampled independently from a standard normal distribution. We randomly generated a sparse vector $\hat{z}$ with p entries, only $p/10$ of which were nonzero. We then set $g = F\hat{z} + w$, where the entries in w were sampled independently and identically from $\mathcal{N}(0, 0.1)$. We chose $\mu = 0.1\mu^{\max}$ for all instances, where $\mu^{\max} = \|F^{\mathrm{T}}g\|_\infty$ is the smallest value of μ for which the solution to (29) is zero.

Results The results are summarized in Table 1. For the small, medium, and large instances, the fastest implementation of SCS, indirect on the GPU, provides a speedup of roughly 30×, 190×, and 1000×, respectively over SDPT3 and Sedumi. In the largest case, SCS takes <4 min compared to nearly 3 days for SDPT3 and Sedumi. In other words, not only is the degree of speedup dramatic in each case, but it also continues to increase as the problem size gets larger; this is consistent with our goal of solving problems outside the ability of traditional interior-point methods.

SCS is meant to provide solutions of modest, not high, accuracy. However, we see that the solutions returned attain an objective value within 0.01 % of the optimal value attained by SDPT3 and Sedumi, a negligible difference in applications.

If we compare the direct and indirect CPU implementations of SCS, we see that for small problems the direct version of SCS is faster, but for larger problems the multi-threaded indirect method dominates. The sparsity pattern in this problem lends itself to an efficient multi-threaded matrix multiply since the columns in the data matrix A have a similar number of nonzeros. This speedup is even more pronounced when the matrix multiplications are performed on the GPU.

Table 1 Results for the lasso example

	Small	Medium	Large
Variables p	10,000	30,000	1,00,000
Measurements q	2000	6000	20,000
Standard form variables n	2001	6001	20,001
Standard form constraints m	22,002	66,002	220,002
Nonzeros in A	3.8×10^6	3.4×10^7	3.9×10^8
SDPT3			
Total solve time (s)	196.5	4.2×10^3	2.3×10^5
Objective	682.2	2088.0	6802.6
Sedumi			
Total solve time (s)	138.0	5.6×10^3	2.5×10^5
Objective	682.2	2088.0	6802.6
SCS direct			
Total solve time (s)	21.9	3.6×10^2	6.6×10^3
Factorization time (s)	5.5	1.1×10^2	4.2×10^3
Iterations	400	540	500
Objective	682.2	2088.1	6803.5
SCS indirect			
Total solve time (s)	31.6	1.2×10^2	7.5×10^2
Average CG iterations	5.9	5.9	5.9
Iterations	400	540	500
Objective	682.2	2088.1	6803.6
SCS indirect GPU			
Total solve time (s)	4.6	22.0	2.1×10^2

6.3 Portfolio Optimization

Consider a simple long-only portfolio optimization problem [52,102], [65, Sect. 4.4.1], in which we choose the relative weights of assets to maximize the expected risk-adjusted return of a portfolio:

$$\text{maximize } \left[\mu^T z - \gamma(z^T \Sigma z)\right] \text{ s.t. } \mathbf{1}^T z = 1, \quad z \geq 0,$$

where the variable $z \in \mathbb{R}^p$ represents the portfolio of p assets, $\mu \in \mathbb{R}^p$ is the vector of expected returns, $\gamma > 0$ is the *risk aversion parameter*, and $\Sigma \in \mathbb{R}^{p \times p}$ is the asset return covariance matrix, also known as the *risk model*. The risk model is expressed in *factor model form*

$$\Sigma = FF^T + D,$$

where $F \in \mathbb{R}^{p \times q}$ is the *factor loading matrix* and $D \in \mathbb{R}^{p \times p}$ is a diagonal matrix representing 'idiosyncratic' or asset-specific risk. The number of risk factors q is typically much less than the number of assets p. (The factor model form is widely used in practice).

This problem can be converted in the standard way into an SOCP:

$$\begin{array}{ll} \text{maximize} & \mu^T z - \gamma(t+s) \\ \text{s.t.} & \mathbf{1}^T z = 1, \quad z \geq 0, \quad \|D^{1/2} z\|_2 \leq u, \quad \|F^T z\|_2 \leq v \\ & \|(1-t, 2u)\|_2 \leq 1+t, \quad \|(1-s, 2v)\|_2 \leq 1+s, \end{array} \tag{30}$$

with variables $z \in \mathbb{R}^p$, $t \in \mathbb{R}$, $s \in \mathbb{R}$, $u \in \mathbb{R}$, and $v \in \mathbb{R}$. This can be transformed into standard form (1) in turn.

Problem Instances The vector of log-returns, $\log(\mu)$, was sampled from a standard normal distribution, yielding lognormally distributed returns. The entries in F were sampled independently from $\mathcal{N}(0, 0.1)$, and the diagonal entries of D were sampled independently from a uniform distribution on $[0, 0.1]$. For all problems, we chose $\gamma = 1$.

Results The results are summarized in Table 2. In all cases the objective value attained by SCS was within 0.5 % of the optimal value. The worst budget constraint violation of the solution returned by SCS in any instance was only 0.002 and the worst nonnegativity constraint violation was only 5×10^{-7}. SCS direct is more than seven times faster than SDPT3 on the largest instance, and much faster than Sedumi, which did not manage to solve the largest instance after a week of computation.

Unlike the previous example, the direct solver is faster than the indirect solver on the CPU for all instances. This is due to imbalance in the number of nonzeros per column which, for the simple multi-threaded matrix multiply we're using, leads to some threads handling much more data than others, and so the speedup provided by parallelization is modest. The indirect method on the GPU is fastest for the medium sized example. For the small example the cost of transferring the data to the GPU outweighs the benefits of performing the computation on the GPU, and the large example could not fit into the GPU memory.

7 Conclusions

We presented an algorithm that can return primal and dual optimal points for convex cone programs when possible, and certificates of primal or dual infeasibility otherwise. The technique involves applying an operator splitting method, the alternating direction method of multipliers, to the homogeneous self-dual embedding of the original optimization problem. This embedding is a feasibility problem that involves finding a point in the intersection of an affine set and a convex cone, and each iteration of our method solves a system of linear equations and projects a point onto the cone. We showed how these individual steps can be implemented efficiently and are often amenable to parallelization. We discuss methods for automatic problem scaling, a critical step in making the method robust.

Table 2 Results for the portfolio optimization example

	Small	Medium	Large
Assets p	1,00,000	5,00,000	25,00,000
Factors q	100	500	2500
Standard form variables n	100,103	500,503	2,502,503
Standard form constraints m	200,104	1,000,504	5,002,504
Nonzeros in A	1.3×10^6	2.5×10^7	5.1×10^8
SDPT3			
Total solve time (s)	70.7	1.6×10^3	6.3×10^4
Objective	0.0388	0.0364	0.0369
Sedumi			
Total solve time (s)	100.6	7.9×10^3	$>6.1 \times 10^5$
Objective	0.0388	0.0364	?
SCS direct			
Total solve time (s)	13.0	190	9.6×10^3
Factorization time (s)	0.6	19.2	913
Iterations	500	440	980
Objective	0.388	0.0365	0.0367
SCS indirect			
Total solve time (s)	27.6	313	2.5×10^4
Average CG iterations	3.0	3.0	3.0
Iterations	500	440	980
Objective	0.0388	0.0365	0.0367
SCS indirect GPU			
Total solve time (s)	27.8	184	OOM

We provide a reference implementation of our algorithm in C, which we call SCS. We show that this solver can solve large instances of cone problems to modest accuracy quickly and is particularly well suited to solving large cone problems outside of the reach of standard interior-point methods.

Acknowledgments This research was supported by DARPA's XDATA program under grant FA8750-12-2-0306. N. Parikh was supported by a NSF Graduate Research Fellowship under grant DGE-0645962. The authors thank Wotao Yin for extensive comments and suggestions on an earlier version of this manuscript, and Lieven Vandenberghe for fruitful discussions early on. We would also like to thank the anonymous reviewers for their constructive feedback.

Conflict of interest The authors declare that they have no conflict of interest.

Appendix: Nonexpansivity

In this appendix we show that the mapping consisting of one iteration of the algorithm (17) is nonexpansive, i.e., if we denote the mapping by ϕ, then we shall show that

$$\|\phi(u, v) - \phi(\hat{u}, \hat{v})\|_2 \le \|(u, v) - (\hat{u}, \hat{v})\|_2,$$

for any (u, v) and $(\hat{u}, \hat{v})$.

From (17) we can write the mapping as the composition of two operators, $\phi = P \circ L$, where

$$P(x) = (\Pi_{\mathcal{C}}(x), -\Pi_{-\mathcal{C}^*}(x)),$$

and

$$L(u, v) = (I + Q)^{-1}(u + v) - v.$$

To show that ϕ is nonexpansive, we only need to show that both P and L are nonexpansive.

To show that P is nonexpansive, we proceed as follows

$$\begin{aligned}
\|x - \hat{x}\|_2^2 &= \|\Pi_{\mathcal{C}}(x) + \Pi_{-\mathcal{C}^*}(x) - \Pi_{\mathcal{C}}(\hat{x}) - \Pi_{-\mathcal{C}^*}(\hat{x})\|_2^2 \\
&= \|\Pi_{\mathcal{C}}(x) - \Pi_{\mathcal{C}}(\hat{x})\|_2^2 + \|\Pi_{-\mathcal{C}^*}(x) - \Pi_{-\mathcal{C}^*}(\hat{x})\|_2^2 \\
&\quad - 2\Pi_{\mathcal{C}}(\hat{x})^{\mathrm{T}}\Pi_{-\mathcal{C}^*}(x) - 2\Pi_{\mathcal{C}}(x)^{\mathrm{T}}\Pi_{-\mathcal{C}^*}(\hat{x}) \\
&\ge \|\Pi_{\mathcal{C}}(x) - \Pi_{\mathcal{C}}(\hat{x})\|_2^2 + \|\Pi_{-\mathcal{C}^*}(x) - \Pi_{-\mathcal{C}^*}(\hat{x})\|_2^2 \\
&= \|(\Pi_{\mathcal{C}}(x) - \Pi_{\mathcal{C}}(\hat{x})), -(\Pi_{-\mathcal{C}^*}(x) - \Pi_{-\mathcal{C}^*}(\hat{x}))\|_2^2 \\
&= \|P(x) - P(\hat{x})\|_2^2,
\end{aligned}$$

where the first equality is from the Moreau decompositions of x and $\hat{x}$ with respect to the cone $\mathcal{C}$, the second follows by expanding the norm squared and the fact that $\Pi_{\mathcal{C}}(x) \perp \Pi_{-\mathcal{C}^*}(x)$ for any x, and the inequality follows from $\Pi_{\mathcal{C}}(\hat{x})^{\mathrm{T}}\Pi_{-\mathcal{C}^*}(x) \le 0$ by the definition of dual cones.

Similarly for L we have

$$\begin{aligned}
\|L(u, v) - L(\hat{u}, \hat{v})\|_2 &= \left\|(I + Q)^{-1}(u - \hat{u} + v - \hat{v}) - v + \hat{v}\right\|_2 \\
&= \left\|\left[(I + Q)^{-1} \; -(I - (I + Q)^{-1})\right](u - \hat{u}, v - \hat{v})\right\|_2 \\
&\le \|(u - \hat{u}, v - \hat{v})\|_2 = \|(u, v) - (\hat{u}, \hat{v})\|_2,
\end{aligned}$$

where the inequality can be seen from the fact that

$$\left[(I + Q)^{-1} \; -(I - (I + Q)^{-1})\right]\left[(I + Q)^{-1} \; -(I - (I + Q)^{-1})\right]^{\mathrm{T}} = I$$

by the skew symmetry of Q, and so $\left\|\left[(I + Q)^{-1} \; -(I - (I + Q)^{-1})\right]\right\|_2 = 1$.

References

1. Ye, Y.: Interior Point Algorithms: Theory and Analysis. Wiley, London (2011)
2. Sturm, J.: Using SeDuMi 1.02, a Matlab toolbox for optimization over symmetric cones. Optim. Methods Softw. **11**(1), 625–653 (1999)
3. Skajaa, A., Ye, Y.: A homogeneous interior-point algorithm for nonsymmetric convex conic optimization. http://www.stanford.edu/yyye/nonsymmhsdimp.pdf (2012)
4. Glowinski, R., Marrocco, A.: Sur l'approximation, par elements finis d'ordre un, et la resolution, par penalisation-dualité, d'une classe de problems de Dirichlet non lineares. Rev. Fr. d'Autom. Inf. Rech. Opér. **9**, 41–76 (1975)
5. Gabay, D., Mercier, B.: A dual algorithm for the solution of nonlinear variational problems via finite element approximations. Comput. Math. Appl. **2**, 17–40 (1976)
6. Gabay, D.: Applications of the method of multipliers to variational inequalities. In: Fortin, M., Glowinski, R. (eds.) Augmented Lagrangian Methods: Applications to Numerical Solution of Boundary-Value Problems, pp. 299–331. North-Holland, Amsterdam (1983)
7. Eckstein, J.: Splitting methods for monotone operators with applications to parallel optimization. Ph.D. thesis, Massachusetts Institute of Technology (1989)
8. 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**, 1–122 (2011)
9. He, B., Yuan, X.: On the $O(1/n)$ convergence rate of the Douglas–Rachford alternating direction method. SIAM J. Numer. Anal. **50**(2), 700–709 (2012)
10. 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)
11. Xu, X., Hung, P., Ye, Y.: A simplified homogeneous and self-dual linear programming algorithm and its implementation. Ann. Oper. Res. **62**, 151–171 (1996)
12. Nesterov, Y., Nemirovski, A.: Interior-Point Polynomial Methods in Convex Programming. SIAM, Philadelphia (1994)
13. Wen, Z., Goldfarb, D., Yin, W.: Alternating direction augmented Lagrangian methods for semidefinite programming. Math. Program. Comput. **2**(3–4), 203–230 (2010)
14. Lan, G., Lu, Z., Monteiro, R.: Primal–dual first-order methods with $\mathcal{O}(1/\epsilon)$ iteration-complexity for cone programming. Math. Program. **126**(1), 1–29 (2011)
15. Aybat, N., Iyengar, G.: An augmented Lagrangian method for conic convex programming. Preprint (2013). arXiv:1302.6322v1
16. Boyle, J., Dykstra, R.: A method for finding projections onto the intersection of convex sets in Hilbert spaces. In: Dykstra, R., Robertson, T., Wright, F. (eds.) Advances in Order Restricted Statistical Inference. Lecture Notes in Statistics, vol. 37, pp. 28–47. Springer, New York (1986)
17. Bauschke, H., Borwein, J.: Dykstra's alternating projection algorithm for two sets. J. Approx. Theory **79**(3), 418–443 (1994)
18. Censor, Y., Chen, W., Combettes, P., Davidi, R., Herman, G.: On the effectiveness of projection methods for convex feasibility problems with linear inequality constraints. Comput. Optim. Appl. **51**(3), 1065–1088 (2012)
19. Censor, Y., Elfving, T.: A multiprojection algorithm using Bregman projections in a product space. Numer. Algorithms **8**, 221–239 (1994)
20. Bauschke, H., Koch, V.: Projection methods: Swiss army knives for solving feasibility and best approximation problems with halfspaces. arXiv:1301.4506 (2013)
21. Eckstein, J., Bertsekas, D.: On the Douglas–Rachford splitting method and the proximal point algorithm for maximal monotone operators. Math. Program. **55**, 293–318 (1992)
22. Combettes, P., Pesquet, J.: Primal–dual splitting algorithm for solving inclusions with mixtures of composite, Lipschitzian, and parallel-sum type monotone operators. Set-Valued Var. Anal. **20**(2), 307–330 (2012)
23. Combettes, P.: Systems of structured monotone inclusions: duality, algorithms, and applications. SIAM J. Optim. **23**(4), 2420–2447 (2013)
24. Komodakis, N., Pesquet, J.: Playing with duality: an overview of recent primal–dual approaches for solving large-scale optimization problems. arXiv:1406.5429 (2014)
25. Glowinski, R., Le Tallec, P.: Augmented Lagrangian and Operator-Splitting Methods in Nonlinear Mechanics. SIAM, Philadelphia (1989)

26. Lions, P., Mercier, B.: Splitting algorithms for the sum of two nonlinear operators. SIAM J. Numer. Anal. **16**(6), 964–979 (1979)
27. Glowinski, R.: Numerical Methods for Nonlinear Variational Problems. Springer, Berlin (1984)
28. Fortin, M., Glowinski, R.: Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems. North-Holland, Amsterdam (1983)
29. Douglas, J., Rachford, H.: On the numerical solution of the heat conduction problem in 2 and 3 space variables. Trans. Am. Math. Soc. **82**, 421–439 (1956)
30. Rockafellar, R.: Monotone operators and the proximal point algorithm. SIAM J. Control Optim. **14**(5), 877–898 (1976)
31. Spingarn, J.: Partial inverse of a monotone operator. Appl. Math. Optim. **10**, 247–265 (1983)
32. Spingarn, J.: Applications of the method of partial inverses to convex programming: decomposition. Math. Program. **32**, 199–223 (1985)
33. Spingarn, J.: A primal–dual projection method for solving systems of linear inequalities. Linear Algebra Appl. **65**, 45–62 (1985)
34. Eckstein, J.: The Lions–Mercier splitting algorithm and the alternating direction method are instances of the proximal point algorithm. Tech. Rep. LIDS-P-1769, Massachusetts Institute of Technology (1989)
35. Byrne, C.: Iterative oblique projection onto convex sets and the split feasibility problem. Inverse Probab. **18**(2), 441 (2002)
36. Censor, Y., Motova, A., Segal, A.: Perturbed projections and subgradient projections for the multiple-sets split feasibility problem. J. Math. Anal. Appl. **327**(2), 1244–1256 (2007)
37. Censor, T.: Sequential and parallel projection algorithms for feasibility and optimization. In: Multispectral Image Processing and Pattern Recognition, pp. 1–9. Bellingham: International Society for Optics and Photonics (2001)
38. Yan, M., Yin, W.: Self equivalence of the alternating direction method of multipliers. arXiv:1407.7400 (2014)
39. Combettes, P.: The convex feasibility problem in image recovery. Adv. Imaging Electron Phys. **95**, 155–270 (1996)
40. Goldstein, T., Osher, S.: The split Bregman method for L1-regularized problems. SIAM J. Imaging Sci. **2**(2), 323–343 (2009)
41. O'Connor, D., Vandenberghe, L.: Image deblurring by primal–dual operator splitting. SIAM J. Imaging Sci. **7**(3), 1724–1754 (2014)
42. Lin, F., Fardad, M., Jovanovic, M.: Design of optimal sparse feedback gains via the alternating direction method of multipliers. In: Proceedings of the 2012 American Control Conference, pp. 4765–4770 (2012)
43. Annergren, M., Hansson, A., Wahlberg, B.: An ADMM algorithm for solving ℓ_1 regularized MPC (2012)
44. O'Donoghue, B., Stathopoulos, G., Boyd, S.: A splitting method for optimal control. IEEE Trans. Control Syst. Technol. **21**(6), 2432–2442 (2013)
45. Mota, J., Xavier, J., Aguiar, P., Puschel, M.: Distributed ADMM for model predictive control and congestion control. In: 2012 IEEE 51st Annual Conference on Decision and Control (CDC), pp. 5110–5115 (2012)
46. O'Donoghue, B.: Suboptimal control policies via convex optimization. Ph.D. thesis, Stanford University (2012)
47. Wahlberg, B., Boyd, S., Annergren, M., Wang, Y.: An ADMM algorithm for a class of total variation regularized estimation problems. In: Proceedings 16th IFAC Symposium on System Identification (to appear) (2012)
48. Combettes, P., Wajs, V.: Signal recovery by proximal forward–backward splitting. Multiscale Model. Simul. **4**(4), 1168–1200 (2006)
49. Combettes, P., Pesquet, J.: A Douglas–Rachford splitting approach to nonsmooth convex variational signal recovery. IEEE J. Sel. Top. Sign. Proces. **1**(4), 564–574 (2007)
50. Combettes, P., Pesquet, J.: Proximal splitting methods in signal processing. In: Fixed-Point Algorithms for Inverse Problems in Science and Engineering, pp. 185–212. Springer, Berlin (2011)
51. Yang, J., Zhang, Y.: Alternating direction algorithms for ℓ_1-problems in compressive sensing. SIAM J. Sci. Comput. **33**(1), 250–278 (2011)
52. Boyd, S., Mueller, M., O'Donoghue, B., Wang, Y.: Performance bounds and suboptimal policies for multi-period investment. Found. Trends Optim. **1**(1), 1–69 (2013)

53. Parikh, N., Boyd, S.: Block splitting for distributed optimization. Math. Program. Comput. **6**(1), 77–102 (2013)
54. Kraning, M., Chu, E., Lavaei, J., Boyd, S.: Dynamic network energy management via proximal message passing. Found. Trends Optim. **1**(2), 70–122 (2014)
55. Chambolle, A., Pock, T.: A first-order primal–dual algorithm for convex problems with applications to imaging. J. Math. Imaging Vis. **40**(1), 120–145 (2011)
56. Becker, S., Candès, E., Grant, M.: Templates for convex cone problems with applications to sparse signal recovery. Math. Program. Comput. **3**(3), 1–54 (2010)
57. Gondzio, J.: Matrix-free interior point method. Comput. Optim. Appl. **51**(2), 457–480 (2012)
58. Monteiro, R., Ortiz, C., Svaiter, B.: An inexact block-decomposition method for extra large-scale conic semidefinite programming. Optimization-online preprint **4158**, 1–21 (2013)
59. Monteiro, R., Ortiz, C., Svaiter, B.: Implementation of a block-decomposition algorithm for solving large-scale conic semidefinite programming problems. Comput. Optim. Appl. **57**, 45–69 (2014)
60. Monteiro, R., Ortiz, C., Svaiter, B.: A first-order block-decomposition method for solving two-easy-block structured semidefinite programs. Math. Program. Comput. **6**, 103–150 (2014)
61. Zhao, X., Sun, D., Toh, K.: A Newton-CG augmented Lagrangian method for semidefinite programming. SIAM J. Optim. **20**, 1737–1765 (2010)
62. O'Donoghue, B., Candès, E.: Adaptive restart for accelerated gradient schemes. Found. Comput. Math. **15**(3), 715–732 (2015)
63. Esser, E., Zhang, X., Chan, T.: A general framework for a class of first order primal–dual algorithms for convex optimization in imaging science. SIAM J. Imaging Sci. **3**(4), 1015–1046 (2010)
64. Parikh, N., Boyd, S.: Proximal algorithms. Found. Trends Optim. **1**(3), 123–231 (2014)
65. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
66. Rockafellar, R.: Convex Analysis. Princeton University Press, Princeton (1970)
67. Franklin, G., Powell, J., Emami-Naeini, A.: Feedback Control of Dynamic Systems, vol. 3. Addison-Wesley, Reading, MA (1994)
68. Gol'shtein, E., Tret'yakov, N.: Modified Lagrangians in convex programming and their generalizations. Point-to-Set Maps Math. Program. **10**, 86–97 (1979)
69. Eckstein, J.: Parallel alternating direction multiplier decomposition of convex programs. J. Optim. Theory Appl. **80**(1), 39–62 (1994)
70. Pataki, G., Schmieta, S.: The DIMACS library of mixed semidefinite-quadratic-linear programs. dimacs.rutgers.edu/Challenges/Seventh/Instances
71. Mittelmann, H.: An independent benchmarking of SDP and SOCP solvers. Math. Program. (Ser. B) **95**, 407–430 (2003)
72. Golub, G., Van Loan, C.: Matrix Computations, 3rd edn. Johns Hopkins University Press, Baltimore (1996)
73. Davis, T.: Direct Methods for Sparse Linear Systems. SIAM Fundamentals of Algorithms. SIAM, Philadelphia (2006)
74. Vanderbei, R.: Symmetric quasi-definite matrices. SIAM J. Optim. **5**(1), 100–113 (1995)
75. Nocedal, J., Wright, S.: Numerical Optimization. Springer, Berlin (2006)
76. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia (2003)
77. Bauer, F.: Optimally scaled matrices. Numer. Math. **5**(1), 73–87 (1963)
78. Bauer, F.: Remarks on optimally scaled matrices. Numer. Math. **13**(1), 1–3 (1969)
79. Van Der Sluis, A.: Condition numbers and equilibration of matrices. Numer. Math. **14**(1), 14–23 (1969)
80. Ruiz, D.: A scaling algorithm to equilibrate both rows and columns norms in matrices. Tech. Rep., Rutherford Appleton Laboratories (2001)
81. Osborne, E.: On pre-conditioning of matrices. JACM **7**(4), 338–345 (1960)
82. Pock, T., Chambolle, A.: Diagonal preconditioning for first order primal–dual algorithms in convex optimization. In: Proceedings of the 2011 IEEE International Conference on Computer Vision (ICCV), pp. 1762–1769. IEEE (2011)
83. Giselsson, P., Boyd, S.: Diagonal scaling in Douglas–Rachford splitting and ADMM. In: Proceedings of the 54th IEEE Conference on Decision and Control, pp. 5033–5039 (2014)
84. Giselsson, P., Boyd, S.: Metric selection in fast dual forward backward splitting. Automatica **62**, 1–10 (2015)
85. Toh, K., Todd, M., Tütüncü, R.: SDPT3: A Matlab software package for semidefinite programming. Optim. Methods Softw. **11**(12), 545–581 (1999)

86. SCS: Splitting conic solver v1.1.0. https://github.com/cvxgrp/scs (2015)
87. Grant, M., Boyd, S.: CVX: Matlab software for disciplined convex programming, version 2.0 beta. http://cvxr.com/cvx (2013)
88. Diamond, S., Boyd, S.: CVXPY: A python-embedded modeling language for convex optimization. http://web.stanford.edu/boyd/papers/cvxpy_paper.html (2015)
89. Udell, M., Mohan, K., Zeng, D., Hong, J., Diamond, S., Boyd, S.: Convex optimization in Julia. SC14 Workshop on High Performance Technical Computing in Dynamic Languages (2014)
90. Lofberg, J.: YALMIP: A toolbox for modeling and optimization in MATLAB. In: IEEE International Symposium on Computed Aided Control Systems Design, pp. 294–289 (2004)
91. Davis, T.: Algorithm 849: a concise sparse Cholesky factorization package. ACM Trans. Math. Softw. **31**(4), 587–591 (2005)
92. Amestoy, P., Davis, T., Duff, I.: Algorithm 837: AMD, an approximate minimum degree ordering algorithm. ACM Trans. Math. Softw. **30**(3), 381–388 (2004)
93. OpenMP Architecture Review Board: OpenMP application program interface version 3.0. http://www.openmp.org/mp-documents/spec30.pdf (2008)
94. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA. Queue **6**(2), 40–53 (2008)
95. Nesterov, Y.: Towards nonsymmetric conic optimization. http://www.optimization-online.org/DB_FILE/2006/03/1355.pdf (2006). CORE discussion paper
96. Skajaa, A., Ye, Y.: A homogeneous interior-point algorithm for nonsymmetric convex conic optimization. Math. Program. **150**(2), 391–422 (2015)
97. Khanh Hien, L.: Differential properties of Euclidean projection onto power cone. http://www.optimization-online.org/DB_FILE/2014/08/4502.pdf (2014)
98. Tibshirani, R.: Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B **58**(1), 267–288 (1996)
99. Daubechies, I., Defrise, M., De Mol, C.: An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. Commun. Pure Appl. Math. **57**(11), 1413–1457 (2004)
100. Demanet, L., Zhang, X.: Eventual linear convergence of the Douglas–Rachford iteration for basis pursuit. arXiv preprint arXiv:1301.0542 (2013)
101. Lobo, M., Vandenberghe, L., Boyd, S., Lebret, H.: Applications of second-order cone programming. Linear Algebra Appl. **284**, 193–228 (1998)
102. Markowitz, H.: Portfolio selection. J. Finance **7**(1), 77–91 (1952)