SuperSCS: fast and accurate large-scale conic optimization

Pantelis Sopasakis*, Krina Menounou† and Panagiotis Patrinos†

***Abstract*— We present SuperSCS: a fast and accurate method for solving large-scale convex conic problems. SuperSCS combines the SuperMann algorithmic framework with the Douglas-Rachford splitting which is applied on the homogeneous self-dual embedding of conic optimization problems: a model for conic optimization problems which simultaneously encodes the optimality conditions and infeasibility/unboundedness certificates for the original problem. SuperMann allows the use of fast quasi-Newtonian directions such as a modified restarted Broyden-type direction and Anderson's acceleration.**

I. Introduction

Conic optimization problems are of central importance in convex optimization as several solvers and parsers such as CVX [11], CVXPy [6], YALMIP [14] and MOSEK [17] transform given problems into a conic representation. Indeed, all convex optimization problems can be cast in the standard form of a conic optimization problem.

Various interior point methods have been proposed for solving conic optimization problems [23], [25], [8]. Interior point methods can achieve high accuracy, yet, do not scale well with the problem size. On the other hand, first-order methods have low per-iteration cost and minimum memory requirements, therefore, are better suited for large-scale problems [18]. Recent research has turned to first-order methods for large-scale conic problems such as SDPs [28]. However, their convergence rate is at most Q-linear with a Q-factor often close to one, especially for ill-conditioned problems.

The KKT conditions of a conic optimization problem together with conditions for the detection of infeasibility or unboundedness can be combined in a convex feasibility problem known as the homogeneous self-dual embedding (HSDE) [27]. The HSDE has been used in both interior point [23] and first-order methods [18].

In this paper we present a numerical optimization method for solving the HSDE. The HSDE is first cast as a variational inequality which can be equivalently seen as a monotone inclusion. We observe that the splitting cone solver (SCS) presented in [18] can be interpreted as the application of the Douglas-Rachford splitting (DRS) to that monotone inclusion. We then apply the reverse splitting to that monotone inclusion — which is a firmly nonexpansive operator — and employ the SuperMann scheme [24] which allows the use of quasi-Newtonian directions such as restarted Broyden directions and Anderson's acceleration [26], [9]. We call the resulting method *SuperSCS*. This way, SuperSCS can achieve a fast convergence rate while retaining a low per-iteration cost. In fact, SuperSCS uses exactly the same oracle as SCS.

* Queen's University Belfast, School of Electronics, Electrical Engineering and Computer Science, Centre for Intelligent Autonomous Manufacturing Systems, BT9 5AH, Northern Ireland, UK. Email: p.sopasakis@qub.ac.uk

† KU Leuven, Department of Electrical Engineering (ESAT), STADIUS, Kasteelpark 10, 3001 Leuven, Belgium.

This work is accompanied by the open-source (licensed with the MIT licence) free software `SuperSCS` which is available online at https://kul-forbes.github.io/scs/.

P. Sopasakis was supported by European Union's Horizon 2020 research and innovation programme (KIOS CoE) under Grant No. 739551. The work of the third author was supported by: FWO projects: No. G086318N; No. G086518N; Fonds de la Recherche Scientifique –FNRS, the Fonds Wetenschappelijk Onderzoek – Vlaanderen under EOS Project No. 30468160 (SeLMA) and Research Council KU Leuven C1 project No. C14/18/068.

II. Mathematical preliminaries

We denote by $\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{R}^n$ and $\mathbb{R}^{m\times n}$ the sets of real numbers, non-negative reals, n-dimensional real vectors and m-by-n real matrices respectively. We denote the transpose of a matrix A by $A^\top$. For two vectors $x, y \in \mathbb{R}^n$, we denote by $\langle x, y\rangle = x^\top y$ their standard inner product. Let E be a vector space in $\mathbb{R}^n$. We define the *orthogonal complement* of E in $\mathbb{R}^n$ to be the vector space $E^\perp = \{y \in \mathbb{R}^n \mid \langle y, x\rangle = 0, \forall x \in E\}$.

A set $\mathcal{K} \subseteq \mathbb{R}^n$ is called a *convex cone* if it is convex and $\lambda x \in \mathcal{K}$ for every $x \in \mathcal{K}$ and $\lambda > 0$. The binary relation $x \succcurlyeq_{\mathcal{K}} y$ is interpreted as $x - y \in \mathcal{K}$. The *dual cone* of $\mathcal{K}$ is defined as $\mathcal{K}^* = \{x^* \mid \langle x^*, x\rangle \geq 0, \forall x \in \mathcal{K}\}$.

A few examples of convex cones of interest are: (i) the zero cone $\mathcal{K}^{\mathrm{f}}_n = \{0\}^n$, (ii) the cone of symmetric positive semidefinite matrices $\mathcal{K}^{\mathrm{s}}_n = \{x \in \mathbb{R}^{n(n+1)/2} \mid \mathbf{mat}(x) : \text{pos. definite}\}$, where $\mathbf{mat} : \mathbb{R}^{n(n+1)/2} \to \mathbb{R}^{n\times n}$ is defined by

$$\mathbf{mat}(x) = \frac{1}{\sqrt{2}} \begin{bmatrix} \sqrt{2}x_1 & x_2 & \cdots & x_n \\ x_2 & \sqrt{2}x_{n+1} & \cdots & x_{2n-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_n & x_{2n-1} & \cdots & \sqrt{2}x_{n(n+1)/2} \end{bmatrix},$$

(iii) the second-order cone $\mathcal{K}^{\mathrm{q}}_n = \{z = (x, t) : x \in \mathbb{R}^{n-1}, t \in \mathbb{R} \mid \|x\|_2 \leq t\}$, (iv) the positive orthant $\mathcal{K}^{\mathrm{l}}_n = \{x \in \mathbb{R}^n \mid x \geq 0\}$ and (v) the three-dimensional exponential cone, $\mathcal{K}^{\mathrm{e}} = \mathbf{cl}\{(x_1, x_2, x_3) \mid x_1 \geq x_2 e^{x_3/x_2}, x_2 > 0\}$.

The *normal cone* of a nonempty closed convex set C is the set-valued mapping $N_C(x) = \{g \mid \langle g, y - x\rangle \leq 0, \forall y \in C\}$ for $x \in C$ and $N_C(x) = \emptyset$ for $x \notin C$. The *Euclidean projection* of x on C is denoted by Π_C.

III. Conic programs

A cone program is an optimization problem of the form

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\mathbf{minimize}} \quad & \langle c, x\rangle \\ \mathbf{subject\ to} \quad & b - Ax = s,\ s \in \mathcal{K}, \end{aligned} \tag{$\mathcal{P}$}$$

where $A \in \mathbb{R}^{m\times n}$ is a possibly sparse matrix and $\mathcal{K}$ is a nonempty closed convex cone.

The vast majority of convex optimization problems of practical interest can be represented in the above form [18], [3]. Indeed, cone programs can be thought of as a universal representation for all convex problems of practical interest and many convex optimization solvers first transform the given problem into this form.

The dual of ($\mathcal{P}$) is given by [3, Sec. 1.4.3]

$$\begin{array}{ll} \mathbf{minimize}_{y\in\mathbb{R}^m} & \langle b, y\rangle \\ \mathbf{subject\ to} & y \in \mathcal{K}^*,\ A^\top y + c = 0 \end{array} \tag{$\mathcal{D}$}$$

Let $p^\star$ be the optimal value of ($\mathcal{P}$) and $d^\star$ be the optimal value of ($\mathcal{D}$). Strong duality holds ($p^\star = -d^\star$) if the primal or the dual problem are strictly feasible [3, Thm. 1.4.2].

Whenever strong duality holds, the following KKT conditions are necessary and sufficient for optimality of $(x^\star, s^\star, y^\star) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m$:

$$\begin{aligned} Ax^\star + s^\star = b,\ s^\star \in \mathcal{K},\ y^\star \in \mathcal{K}^*, \\ A^\top y^\star + c = 0,\ \langle y^\star, s^\star\rangle = 0. \end{aligned} \tag{1}$$

Infeasibility and unboundedness conditions are provided by the so-called *theorems of the alternative*. The *weak* theorems of the alternative state that 1) Either primal feasibility holds, or there is a y with $A^\top y = 0$, $y \succeq_{\mathcal{K}^*} 0$ and $\langle b, y\rangle < 0$, and, similarly, 2) Either dual feasibility holds or there is a x so that $Ax \succeq_{\mathcal{K}} 0$ and $\langle c, x\rangle \leq 0$ [3, Sec. 1.4.7].

A. Homogeneous self-dual embedding

In this section we present a key result which is due to Ye *et al.* [27]: the HSDE, which is a feasibility problem which simultaneously describes the optimality, (in)feasibility and (un)boundedness conditions of a conic optimization problem. Solving the HSDE yields a solution of the original conic optimization problem, when one exists, or a certificate of infeasibility or unboundedness. We start by considering the following feasibility problem in $(\chi, \varsigma, \psi, \tau, \kappa)$

$$\begin{bmatrix} 0 \\ \varsigma \\ \kappa \end{bmatrix} = Q \begin{bmatrix} \chi \\ \psi \\ \tau \end{bmatrix},\ \varsigma \in \mathcal{K},\ \psi \in \mathcal{K}^*,\ \tau \geq 0,\ \kappa \geq 0, \tag{2a}$$

where

$$Q := \begin{bmatrix} 0 & A^* & c \\ -A & 0 & b \\ -c^* & -b^* & 0 \end{bmatrix} \tag{2b}$$

Note that for $\tau^\star = 1$ and $\kappa^\star = 0$, the above equations reduce to the primal-dual optimality conditions. As shown in [27], the solutions of (2a) satisfy $\kappa^\star\tau^\star = 0$, *i.e.*, at least one of $\kappa^\star$ and $\tau^\star$ must be zero. In particular, if $\kappa = 0$ and $\tau > 0$, then the triplet $(x^\star, y^\star, s^\star)$ with

$$x^\star = \chi^\star/\tau^\star,\ y^\star = \psi^\star/\tau^\star,\ s^\star = \varsigma^\star/\tau^\star,$$

is a primal-dual solution of ($\mathcal{P}$) and ($\mathcal{D}$). If instead $\tau^\star = 0$ and $\kappa > 0$, then the problem is either primal- or dual-infeasible. If $\tau = \kappa = 0$, no conclusion can be drawn.

We define $u = (\chi, \psi, \tau)$ and $v = (0, \varsigma, \kappa)$. The self-dual embedding reduces to the problem of determining u and v such that $Qu = v$ with $(u, v) \in \mathcal{C} \times \mathcal{C}^*$ where $\mathcal{C} := \mathbb{R}^n \times \mathcal{K}^* \times \mathbb{R}_+$. This is equivalent to the variational inequality

$$0 \in Qu + N_{\mathcal{C}}(u), \tag{3}$$

Indeed, for all $u \in \mathcal{C}$,

$$\begin{aligned} N_{\mathcal{C}}(u) &= \{y \mid \langle v - u, y\rangle \leq 0, \forall v \in \mathcal{C}\} \\ &= \{u\}^\perp \cap \{y \mid \langle v, y\rangle \leq 0, \forall v \in \mathcal{C}\} = \{u\}^\perp \cap (-\mathcal{C}^*), \end{aligned}$$

where the second equality follows by considering $v = {}^1\!/_2 u$ and $v = {}^3\!/_2 u$, which both belong to the cone $\mathcal{C}$ (see also [12, Ex. 5.2.6(a)]). From that and the fact that $Qu \in \{u\}^\perp$ since Q is skew-symmetric, the equivalence of the HSDE in (2a) and the variational inequality in (3) follows. Equation (3) is a monotone inclusion which can be solved using operator theory machinery as we discuss in the following section.

IV. SuperSCS

A. SCS and DRS

Since Q is a skew-symmetric linear operator, it is maximally monotone. Being the normal cone of a convex set, $N_{\mathcal{C}}$ is maximally monotone as well. Additionally, because of [2, Cor. 24.4(i)], $Q + N_{\mathcal{C}}$ is maximally monotone. Therefore, we may apply the Douglas-Rachford splitting on the monotone inclusion (3). The SCS algorithm [18] is precisely the application of the DRS to $N_{\mathcal{C}} + Q$ leading to the iterations discussed in [21, Sec. 7.3]. This observation furnishes a short and elegant interpretation of SCS. Here, on the other hand, we consider the reverse splitting, $Q + N_{\mathcal{C}}$, which leads to the following DRS iterations

$$\tilde{u}^\nu = (I + Q)^{-1}(u^\nu) \tag{4a}$$

$$\bar{u}^\nu = \Pi_{\mathcal{C}}(2\tilde{u}^\nu - u^\nu) \tag{4b}$$

$$u^{\nu+1} = u^\nu + \bar{u}^\nu - \tilde{u}^\nu. \tag{4c}$$

For any initial guess u^0, the iterates u^ν converge to a point $u^\star$ which satisfies the monotone inclusion (3) [2, Thm. 25.6(i), (iv)]. The linear system in (4a) can be either solved "directly" using a sparse LDL factorization or "indirectly" by means of the conjugate gradient method [18]. The projection on $\mathcal{C}$ in (4b) essentially requires that we be able to project on $\mathcal{K}^*$.

The iterative method (4) can be concisely written as

$$u^{\nu+1} = Tu^\nu, \tag{5}$$

where $T : \mathbb{R}^N \to \mathbb{R}^N$ is given by $Tu = u + \Pi_{\mathcal{C}}(2(I + Q)^{-1}u - u) - (I + Q)^{-1}u$ and is firmly nonexpansive [2, Chap. 26]. As such it fits the Krasnosel'skii-Mann framework [2, Sec. 5.2] leading to the relaxed iterations

$$u^{\nu+1} = (1 - \lambda)u^\nu + \lambda Tu^\nu, \tag{6}$$

with $\lambda \in (0, 2)$ and, as a result, it fits the SuperMann framework [24].

B. SuperSCS: SuperMann meets SCS

SuperMann considers the problem of finding a fixed-point $x^\star \in \mathbf{fix}\, T$ from the viewpoint of finding a zero of the residual operator

$$R = I - T. \tag{7}$$

SuperMann, instead of applying Krasnosel'skii-Mann-type updates of the form (6), takes extragradient-type updates of the general form

$$w^\nu = u^\nu + \alpha_\nu d^\nu, \tag{8a}$$
$$u^{\nu+1} = u^\nu - \zeta_\nu R w^\nu, \tag{8b}$$

where d^ν are fast, e.g., quasi-Newtonian, directions and scalar parameters α_ν and ζ_ν are appropriately chosen so as to guarantee global convergence.

At each step we perform backtracking line search on α_ν until we either trigger fast convergence (K1 steps) or ensure global convergence (K2 steps) as shown in Algorithm 1. The K2 step, cf. (8b), can be interpreted as a projection of the current iterate on a hyperplane generated by w^ν, separating the set of fixed points from u^ν, and thus guarantees that every iterate comes closer to fixed point set. Alongside, a sufficient decrease of the norm of the residual, $\|Ru^\nu\|$, may trigger a "blind update" (K0 steps) of the form $u^{\nu+1} = u^\nu + d^\nu$, where no line search iterations need to be executed.

Algorithm 1 SuperSCS algorithm

Input: $c_0, c_1, q \in [0,1)$, $\sigma \in (0,1)$, u^0, $\lambda \in (0,2)$ and $\epsilon > 0$
$\eta_0 \leftarrow \|Ru^0\|$, $r_{\text{safe}} \leftarrow \eta_0$
for $\nu = 0, 1, \ldots$ **do**
  Check termination with tolerance ϵ (Sec. IV-C)
  Choose direction d^ν (Sec. IV-D), let $\alpha_\nu \leftarrow 1$
  if $\|Ru^\nu\| \le c_0 \eta_\nu$ **then**
    (K0) $u^{\nu+1} \leftarrow w^\nu$, $\eta^{\nu+1} \leftarrow \|Ru^\nu\|$
  else
    $\eta_{\nu+1} \leftarrow \eta_\nu$
    loop
      $w^\nu \leftarrow u^\nu + \alpha_\nu d^\nu$ and $\rho_\nu \leftarrow \langle Rw^\nu, u^\nu - Tw^\nu \rangle$
      if $\|Ru^\nu\| \le r_{\text{safe}}$ and $\|Rw^\nu\| \le c_1 \|Ru^\nu\|$ **then**
        (K1) $u^{\nu+1} \leftarrow w^\nu$, $r_{\text{safe}} \leftarrow \|Rw^\nu\| + q^\nu \eta_0$, exit loop
      else if $\rho_\nu \ge \sigma \|Ru^\nu\| \|Rw^\nu\|$ **then**
        (K2) $u^{\nu+1} \leftarrow u^\nu - \lambda \frac{\rho_\nu}{\|Rw^\nu\|^2} Rw^\nu$ and exit loop
      else
        $\alpha_\nu \leftarrow \alpha_\nu / 2$

By exploiting the structure of T and, in particular, linearity of $(I+Q)^{-1}$, we may avoid evaluating linear system solves at every backtracking step. Instead, we only need to evaluate $\Pi_{\mathcal{C}}$ once in every backtracking iteration. In particular, for $w^\nu = u^\nu + \alpha d^\nu$, we have

$$\begin{aligned}\tilde{w}^\nu &= (I+Q)^{-1} w^\nu \\ &= (I+Q)^{-1} u^\nu + \alpha (I+Q)^{-1} d^\nu = \tilde{u}^\nu + \alpha \tilde{d}^\nu\end{aligned}$$

where $\tilde{u}^\nu$ has already been computed, since it is needed in the evaluation of Ru^ν, while $\tilde{d}^\nu$ solves $(I+Q)\tilde{d}^\nu = d^\nu$. The computation of $\tilde{d}^\nu$, which is the most costly operation, is performed only once, before the backtracking procedure takes place. The computation of the fixed-point residual of w is also easily computed by $Rw^\nu = \tilde{w}^\nu - \mathbf{\Pi}_{\mathcal{C}}(2\tilde{w}^\nu - w^\nu)$.

Overall, save the computation of the residuals, at every iteration of SuperSCS we need to solve the linear system (4a) twice and invoke $\Pi_{\mathcal{C}}$ exactly $1 + l_\nu$ times, where l_ν is the number of backtracks.

C. Termination

The algorithm is terminated when an approximate optimal solution is found based on its relative primal and dual residuals and relative duality gap, provided such a solution exists. At iteration ν let $u^\nu = (\chi^\nu, \psi^\nu, \tau^\nu)$, $\bar{u}^\nu = (\bar{\chi}^\nu, \bar{\psi}^\nu, \bar{\tau}^\nu)$ and $\tilde{u}^\nu = (\tilde{\chi}^\nu, \tilde{\psi}^\nu, \tilde{\tau}^\nu)$. We compute $\bar{\varsigma}^\nu = \bar{\psi}^\nu - 2\tilde{\psi}^\nu + \psi^\nu$. Let us also define the triplet $(\bar{x}^\nu, \bar{y}^\nu, \bar{s}^\nu) := (\bar{\chi}^\nu/\bar{\tau}^\nu, \bar{\psi}^\nu/\bar{\tau}^\nu, \bar{\varsigma}^\nu/\bar{\tau}^\nu)$, which serves as the candidate primal-dual solution at iteration ν. The relative primal residual is

$$\mathrm{pr}_\nu = \frac{\|A\bar{x}^\nu + \bar{s}^\nu - b\|}{1 + \|b\|} \tag{9a}$$

The relative dual residual is

$$\mathrm{dr}_\nu = \frac{\|A^\top \bar{y}^\nu + c\|}{1 + \|c\|} \tag{9b}$$

The relative duality gap is defined as

$$\mathrm{gap}_\nu = \frac{|\langle c, \bar{x}^\nu \rangle + \langle b, \bar{y}^\nu \rangle|}{1 + |\langle c, \bar{x}^\nu \rangle| + |\langle b, \bar{y}^\nu \rangle|} \tag{9c}$$

If pr_ν, dr_ν and gap_ν are all below a specified tolerance $\epsilon > 0$, then we conclude that ($\mathcal{P}$) is feasible, the algorithm is terminated and the triplet (x^ν, y^ν, s^ν) is an approximate solution.

The relative infeasibility certificate is defined as (note that $\bar{y}^\nu \in \mathcal{K}^*$ as a result of the projection step)

$$\mathrm{ic}_\nu = \begin{cases} \|b\| \, \|A^\top \bar{y}^\nu\| / \langle b, \bar{y}^\nu \rangle, & \text{if } \langle b, \bar{y}^\nu \rangle < 0 \\ +\infty, & \text{else} \end{cases} \tag{9d}$$

Likewise, the relative unboundedness certificate is defined as

$$\mathrm{uc}_\nu = \begin{cases} \|c\| \, \|A\bar{x}^\nu + \bar{s}^\nu\| / \langle c, \bar{x}^\nu \rangle, & \text{if } \langle c, \bar{x}^\nu \rangle < 0 \\ +\infty, & \text{else} \end{cases} \tag{9e}$$

Provided that $\bar{u}^\nu$ is not a feasible ϵ-optimal point, it is a certificate of unboundedness if $\mathrm{uc}_\nu < \epsilon$ and it is a certificate of infeasibility if $\mathrm{ic}_\nu < \epsilon$.

D. Quasi-Newtonian directions

Quasi-Newtonian directions d^ν can be computed according to the general rule

$$d^\nu = -B_\nu^{-1} R u^\nu = -H_\nu R u^\nu, \tag{10}$$

where invertible linear operators H_ν are updated according to certain low-rank updates so as to satisfy certain secant conditions starting from an initial operator H_0.

1) Restarted Broyden directions: Here we make use of Powell's trick to update linear operators B_ν in such a way so as to enforce nonsingularity using the recursive formula [20], [24]:

$$B_{\nu+1} = B_\nu + \tfrac{1}{\|z^\nu\|^2} (\tilde{\xi}^\nu - B_\nu z^\nu) z^{\nu\top} \tag{11}$$

where $z^\nu = w^\nu - u^\nu$, $\xi^\nu = Rw^\nu - Ru^\nu$ and for a fixed parameter $\bar{\vartheta} \in (0,1)$ and $\gamma_\nu = \langle H_\nu \xi^\nu, z^\nu \rangle / \|z^\nu\|^2$ we have $\tilde{\xi}^\nu = (1-\theta_\nu) B_\nu z^\nu + \theta_\nu \xi^\nu$ with

$$\theta_\nu = \begin{cases} 1 & \text{if } |\gamma_\nu| \ge \bar{\vartheta} \\ \frac{1 - \mathbf{sgn}(\gamma_\nu)\bar{\vartheta}}{1 - \gamma_\nu} & \text{otherwise} \end{cases} \tag{12}$$

and the convention $\mathbf{sgn}(0) = 1$. Using the Sherman-Morisson formula, operators H_ν are updated as follows:

$$H_{\nu+1} = H_\nu + \tfrac{1}{\langle H_\nu \tilde{\xi}^\nu, z^\nu \rangle}(z^\nu - H_\nu \tilde{\xi}^\nu)(z^{\nu\top} H_\nu), \qquad (13)$$

thus lifting the need to compute and store matrices B_ν.

The Broyden method requires that we store matrices of dimension $(m+n+1)\times(m+n+1)$. Here we employ a limited-memory restarted Broyden (RB) method which affords us a computationally favorable implementation using buffers of length mem, that is $Z_\nu = [z^\nu \; z^{\nu-1} \; \cdots \; z^{\nu-\mathrm{mem}+1}]$, and $\tilde{Z}_\nu = [\tilde{z}^\nu \; \tilde{z}^{\nu-1} \; \cdots \; \tilde{z}^{\nu-\mathrm{mem}+1}]$, where $\tilde{z}^i$ are the auxiliary variables $\tilde{z}^i := {z^i - H_i\tilde{\xi}^i}/{\langle s^i, H_i \tilde{\xi}^i \rangle}$. We have observed

Algorithm 2 Modified restarted Broyden method

Input: Old buffers $Z = Z_\nu$ and $\tilde{Z} = \tilde{Z}_\nu$, $\xi = \xi^\nu$, $r = Ru^\nu$, $z = z^\nu$, ϑ, mem
Output: Direction d, New buffers
  $d \leftarrow -r$, $\tilde{z} \leftarrow \xi$, $m' \leftarrow$ current cursor position
  for $i = 1, \ldots, m'$ **do**
    $\tilde{z} \leftarrow \tilde{z} + \langle z^i, \tilde{z} \rangle \tilde{z}^i$, and $d \leftarrow d + \langle z^i, d \rangle \tilde{z}^i$
  Compute θ as in (12) with $\gamma = {\langle \tilde{z}, z \rangle}/{\|z\|^2}$
  $\tilde{z} \leftarrow (1-\theta)z + \theta\tilde{z}$, $\tilde{z} \leftarrow {z - \tilde{z}}/{\langle z, \tilde{z} \rangle}$, and $d \leftarrow d + \langle z, d \rangle \tilde{z}$
  if $m' = \mathrm{mem}$ **then**
    Empty buffers Z and $\tilde{Z}$, $m' \leftarrow 1$
  else
    Append z to Z and $\tilde{z}$ to $\tilde{Z}$, $m' \leftarrow m' + 1$

that SuperSCS performs better when deactivating K0 steps or using a small value for c_0 (e.g., $c_0 = 0.1$) when using restarted Broyden directions.

2) Anderson's acceleration: Anderson's acceleration (AA) imposes a multi-secant condition [26], [9]. In particular, at every iteration ν we update a buffer of mem past values of z and ξ, that is we construct a buffer Z_ν as above and a buffer $\Xi_\nu = \begin{bmatrix}\xi^\nu & \xi^{\nu-1} & \cdots & \xi^{\nu-\mathrm{mem}+1}\end{bmatrix}$. Directions are computed as

$$d^\nu = -Ru^\nu - (Z_\nu - \Xi_\nu)t^\nu, \qquad (14)$$

where t^ν is a least-squares solution of the linear system $\Xi_\nu t^\nu = Ru^\nu$, that is t^ν solves

$$\underset{t^\nu}{\mathbf{minimize}} \; \|\Xi_\nu t^\nu - Ru^\nu\|^2, \qquad (15)$$

and can be solved using the singular value decomposition of Ξ_ν, or a QR factorization which may be updated at every iteration [26]. In practice Anderson's acceleration works well for short memory lengths, typically between 3 and 10, and, more often than not, outperforms the above restarted Broyden directions.

E. Convergence

The convergence properties of SuperSCS are inherited by those of the general SuperMann scheme [24]. In particular, under a weak boundedness assumption for the quasi-Newtonian directions d^ν, Ru^ν converges to zero and u^ν converges to a $u^\star$ which satisfies the HSDE (3). If, additionally, R is metrically subregular at $u^\star$ — a weak assumption — then, the convergence is R-linear. Under additional assumptions, SuperSCS with the full-memory counterpart of the above restarted Broyden scheme can be shown to converge superlinearly. The restarted Broyden directions of Algorithm 2 and Anderson's acceleration, though not proven superlinear directions, exhibit steep linear convergence as shown in the next section (see Fig. 2) and have low memory requirements.

V. Benchmarks and Results

A. Benchmarking methodology

In order to compare different solvers in a statistically meaningful way, we use the Dolan-Moré (DM) plot [7] and the shifted geometric mean [16]. The DM plot allows us to compare solvers in terms of their relative performance (e.g., computation time, flops, etc) and robustness, i.e., their ability to successful solve a given problem up to a certain tolerance.

Let P be a finite set of test problems and S a finite set of solvers we want to compare to one another. Let $t_{p,s}$ denote the computation time that solver s needs to solve problem p. We define the ratio between $t_{p,s}$ and the lowest observed cost to solve this problem using a solver from S as

$$r_{p,s} = \frac{t_{p,s}}{\min_{s' \in S} t_{p,s'}}. \qquad (16)$$

If s cannot solve p at all, we define $t_{p,s} = +\infty$ and $r_{p,s} = +\infty$. The cumulative distribution of the performance ratio is the DM performance profile plot. In particular, define

$$\rho_s(\tau) = {1}/{|P|} \cdot |\{p \in P : r_{p,s} \leq \tau\}|, \qquad (17)$$

for $\tau \geq 1$. The DM performance profile plot is the plot of $\rho_s(\tau)$ versus τ on a logarithmic x-axis. For every $s \in S$, the value $\rho_s(1)$ is the probability of solver s to solve a given problem faster than all other solvers, while $\mathbf{lim}_{\tau\to\infty} \rho_s(\tau)$ is the probability that solver s solves a given problem at all.

As demonstrated in [10], DM plots aim at comparing multiple solvers to the best one (cf. (16)), than to one another. Therefore, alongside we shall report the shifted geometric mean of computation times for each solver following [16]. For solver $s \in S$, define the vector $t^s \in \mathbb{R}^{|P|}$ with

$$t_p^s = \begin{cases} t_{p,s} & \text{if } t_{p,s} < \infty \\ 100 \, \mathbf{max}\{t_{p,s} \mid p \in P, t_{p,s} < \infty\} & \text{otherwise} \end{cases}$$

The shifted geometric mean of t^s with shifting parameter $\sigma \geq 0$ is defined as

$$\mathbf{sgm}_\sigma = \mathbf{exp}\left[\textstyle\sum_{p\in P} \mathbf{ln}\left(\mathbf{max}\{1, \sigma + t_p^s\}\right)\right] - \sigma. \qquad (18)$$

Hereafter we use $\sigma = 10\,\mathrm{s}$.

In what follows we compare SuperSCS with the quasi-Newtonian direction methods presented in Section IV-D against SCS [19], [18]. All tolerances are fixed to 10^{-4}. In order to allow for a fair comparison among algorithms with different per-iteration cost, we do not impose a maximum number of iterations; instead, we consider that an algorithm has failed to produce a solution — or a certificate of unboundedness/infeasibility — if it has not terminated after a certain (large) maximum time. All benchmarks were executed on a system with a quad-core i5-6200U CPU at 2.30 GHz and 12 GB RAM running Ubuntu 14.04.

B. Semidefinite programming problems

Let us consider the problem of sparse principal component analysis with an ℓ_1-regularizer which has the form [5].

$$\textbf{maximize } \textbf{trace}(SZ) - \lambda\|Z\|_1 \tag{19a}$$
$$\textbf{subject to } \textbf{trace}(Z) = 1,\ Z = Z^\top,\ Z \succeq 0 \tag{19b}$$

A total of 288 randomly generated problems was used for benchmarking with $d \in \{50, 120, 140, 180\}$ and $\lambda \in \{0.1, 2, 5\}$, where d is the dimension of Z. The DP plot in Fig. 1 shows that SuperSCS is consistently faster and more robust compared to SCS. In Table I we see that SuperSCS is faster than SCS by more than an order of magnitude; in particular, SuperSCS with AA directions and memory 5 was found to perform best.

TABLE I: Regularized PCA SDP: Solver Statistics

Method	$\mathbf{sgm}_{10}(t^s)$	Success
SCS	96.82	74.31%
RB (mem:50)	3.17	100%
RB (mem:100)	3.31	100%
AA (mem:5)	2.13	100%
AA (mem:10)	2.66	100%

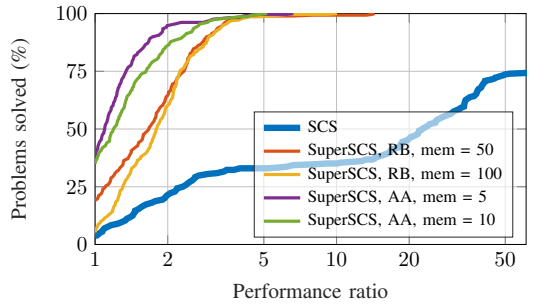

FIG. 1: DM performance plot on 288 ℓ_1-regularized PCA problems of the form (19).

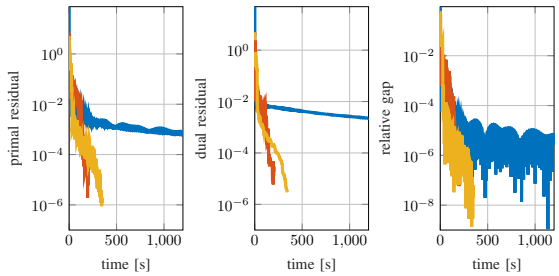

FIG. 2: Progress of SuperSCS (with RB and AA directions) and SCS versus time for a large-scale SDP of the form (19) with $d = 500$ (with $m = 625751$ and $n = 250501$). [— SCS; — SuperSCS RB with memory 50; — SuperSCS AA with memory 5].

Additionally, for this benchmark we tried the interior-point solvers of SDPT3 [25] and Sedumi [23]. Both exhibited similar performance; in particular, for problems of dimension $Z \in \mathbb{R}^{140\times140}$, SDPT3 and Sedumi required 6500 s to 7500 s and for problems of dimension $Z \in \mathbb{R}^{180\times180}$ they required 19000 s to 21500 s. Note that SuperSCS (with RB and memory 50) solves all problems in no more than 11.7 s. Additionally, interior-point methods have an immense memory footprint of several GB, in this example whereas SuperSCS with RB and memory 100 needs as little as 211.1 MB and with AA and memory 5 consumes just 46.2 MB.

C. LASSO problems

Regularized least-squares problems with the $\|\cdot\|_1$-regularizer, also known as LASSO problems, are optimization problems of the form

$$\underset{x\in\mathbb{R}^n}{\textbf{minimize}}\ 1/2\|Ax - b\|^2 + \mu\|x\|_1, \tag{20}$$

where $A \in \mathbb{R}^{m\times n}$ is a (sparse) matrix, and $\mu > 0$ is the regularization weight. LASSO problems find applications in statistics and compressed sensing [22]. LASSO problems are cast as second-order cone programs [13].

TABLE II: LASSO: Solver Statistics

Method	$\mathbf{sgm}_{10}(t^s)$	Success
SCS	5.97	100%
RB (mem:50)	2.88	100%
RB (mem:100)	2.61	100%
AA (mem:5)	3.38	100%
AA (mem:10)	3.87	100%

LASSO problems aim at finding a sparse vector x which minimizes $\|Ax - b\|^2$. The sparseness of the minimizer $x^\star$ can be controlled by $\mu \in \{0.01, 0.1, 1\}$. We tested 1152 randomly generated LASSO problems with $n \in \{631, 1000, 1585, 2512\}$, $m = \lceil n/5 \rceil$, and matrices A with condition numbers $\kappa_A \in \{10, 215, 4600, 10^5\}$. The DM plot in Fig. 3 and the statistics presented in Table II demonstrate that SuperSCS, both with RB and AA directions, outperforms SCS.

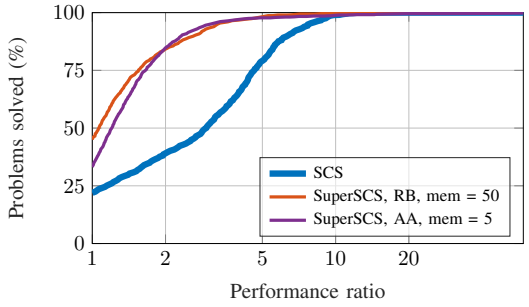

FIG. 3: DM performance plot on 1152 LASSO problems.

D. Sparse ℓ_1-regularized logistic regression

Logistic regression is a regression model where dependent variables are binary [4]. The ℓ_1 regularized variant of logistic regression aims at performing simultaneous regression and feature selection and amounts to solving an optimization problem of the following form [1]:

$$\underset{w\in\mathbb{R}^p}{\textbf{minimize}}\ \lambda\|w\|_1 - \sum_{i=1}^{q} \textbf{log}(1 + \textbf{exp}(a^\top w_i + b)). \tag{21}$$

Similar to LASSO (Sec. V-C), parameter $\lambda > 0$ controls the sparseness of the solution $w^\star$. Such problems can be cast as conic programs with the exponential cone, which is not self-dual. A total of 288 randomly generated problems was used for benchmarking

TABLE III: Sparse ℓ_1-regularized logistic regression: Solver Statistics

Method	$\mathbf{sgm}_{10}(t^s)$	Success
SCS	4.85	100%
RB (mem:50)	7.23	100%
RB (mem:100)	7.28	100%
AA (mem:5)	3.00	100%
AA (mem:10)	3.09	100%

with $p \in \{80, 100\}$, $q \in \{50, 100, 120\}$ and $\lambda \in \{10, 20, 50\}$. In Fig. 4 we observe that SuperSCS with RB directions is slower compared to SCS, however SuperSCS with AA is noticeably faster.

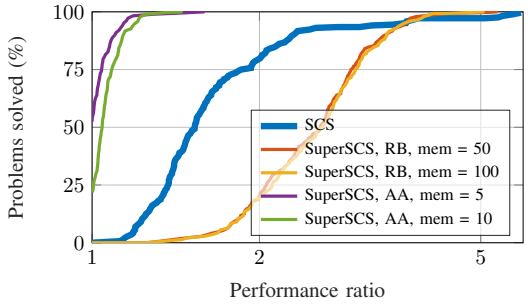

FIG. 4: DM performance plot on 288 logistic regression problems of the form (21).

E. Maros-Mészáros QP problems

TABLE IV: Maros-Mészáros QP problems: Solver Statistics

Method	$\mathbf{sgm}_{10}(t^s)$	Success
SCS	56.61	83.02%
RB (mem:50)	9.66	90.57%
RB (mem:100)	6.57	90.57%
AA (mem:5)	5.79	90.57%
AA (mem:10)	8.62	91.51%

Here we present performance of SCS and SuperSCS (with RB and AA directions) on this collection of problems on the Maros-Mészáros collection of problems [15]. As shown in Fig. 5 SuperSCS, both with RB directions and Anderson's acceleration, is faster and more robust compared to SCS. The two quasi-Newtonian directions appear to be on a par.

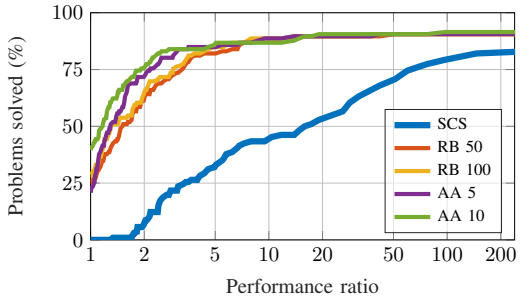

FIG. 5: DM performance plot on the problems of the Maros-Mészáros repository.

VI. Conclusions

In this work we introduced SuperSCS: a first-order method for large-scale conic optimization problems which combines the low iteration cost of SCS and the fast convergence of SuperMann. We have compared SuperSCS with SCS on a broad collection of conic optimization problems of practical interest. Using Dolan-Moré plots and runtime statistics, we demonstrated that SuperSCS with Anderson's acceleration is faster and more robust than SCS.

The C implementation of SuperSCS builds up on SCS and is a free open-source software. SuperSCS can be interfaced from MATLAB, Python, can be invoked via CVX, CVXPy and YALMIP and is also available as a Docker image (see https://kul-forbes.github.io/scs/).

References

[1] F. Abramovich and V. Grinshtein. High-dimensional classification by sparse logistic regression. *ArXiv e-prints 1706.08344v2*, 2017.

[2] H.H. Bauschke and P.L. Combettes. *Convex analysis and monotone operator theory in Hilbert spaces*. Springer, 2011.

[3] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization*. SIAM, Jan 2001.

[4] D.R. Cox. The regression analysis of binary sequences (with discussion). *J Roy Stat Soc B*, 20:215–242, 1958.

[5] A. d'Aspremont, L. El Ghaoui, M.I. Jordan, and G.R.G. Lanckriet. A direct formulation for sparse PCA using semidefinite programming. *SIAM Review*, 49(3):434–448, Jan 2007.

[6] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *J. Mach. Learn. Res.*, 17(83):1–5, 2016.

[7] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Math. Program., Ser. A*, 91:201–213, 2002.

[8] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *IEEE CDC*, pages 3071–3076, 2013.

[9] H. Fang and Y. Saad. Two classes of multisecant methods for nonlinear acceleration. *Numer. Lin. Alg. Applic.*, 16(3):197–221, Mar 2009.

[10] N. Gould and J. Scott. A note on performance profiles for benchmarking software. *ACM Trans. Math. Software*, 43(2):1–5, Aug 2016.

[11] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, Mar 2014.

[12] J.B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of convex analysis*. Springer, 2004.

[13] M. Sousa Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. In *ILAS Symp. Fast Alg. for Contr., Sign. Im. Proc.*, volume 284, pages 193–228, 1998.

[14] J. Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *CACSD Conference*, 2004.

[15] I. Maros and C. Mészáros. A repository of convex quadratic programming problems. *Optim. Meth. Soft.*, 11(1-4):671–681, Jan 1999.

[16] H. Mittelman. Decision tree for optimization problems: benchmarks for optimization software. http://plato.asu.edu/bench.html. [Online; accessed 1-June-2018].

[17] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 8.1.*, 2017.

[18] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *JOTA*, 169(3):1042–1068, Jun 2016.

[19] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, v2.0.2. https://github.com/cvxgrp/scs, Nov 2017.

[20] M. Powell. A hybrid method for nonlinear equations. *Numerical Methods for Nonlinear Algebraic Equations*, pages 87–144, 1970.

[21] E.K. Ryu and S. Boyd. A primer on monotone operator methods: a survey. *Appl. Comput. Math.*, 15(1):3–43, 2016.

[22] P. Sopasakis, N. Freris, and P. Patrinos. Accelerated reconstruction of a compressively sampled data stream. In *IEEE 24th European Signal Processing Conference EUSIPCO*, pages 1078–1082, Aug 2016.

[23] J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Meth. Soft.*, 11–12:625–653, 1999. Version 1.05 available at http://fewcal.kub.nl/sturm.

[24] A. Themelis and P. Patrinos. Supermann: a superlinearly convergent algorithm for finding fixed points of nonexpansive operators. *ArXiv e-prints 1609.06955*, 2016.

[25] R.H. Tütüncü, K. C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Math. Prog.*, 95(2):189–217, Feb 2003.

[26] H.F. Walker and P. Ni. Anderson acceleration for fixed-point iterations. *SIAM J. Numer. Anal.*, 49(4):1715–1735, Aug 2011.

[27] Y. Ye, M. Todd, and S. Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Math. Oper. Res.*, 19(1):53–67, 1994.

[28] Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn. Fast ADMM for homogeneous self-dual embedding of sparse SDPs. *IFAC-PapersOnLine*, 50(1):8411–8416, 2017.