SDPT3 — a MATLAB software package for semidefinite programming, version 2.1

K. C. Toh [*], M. J. Todd [†], and R. H. Tütüncü [‡]

September 28, 1999

Abstract

This software package is a MATLAB implementation of infeasible path-following algorithms for solving standard semidefinite programming (SDP) problems. Mehrotra-type predictor-corrector variants are included. Analogous algorithms for the homogeneous formulation of the standard SDP problem are also implemented. Four types of search directions are available, namely, the AHO, HKM, NT, and GT directions. A few classes of SDP problems are included as well. Numerical results for these classes show that our algorithms are fairly efficient and robust on problems with dimensions of the order of a few hundreds.

[*]Department of Mathematics, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260. (`mattohkc@math.nus.edu.sg`). Research supported in part by National University of Singapore Academic Research Grant RP972685.

[†]School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York 14853, USA (`miketodd@cs.cornell.edu`). Research supported in part by NSF through grants DMS-9505155 and DMS-9805602 and ONR through grant N00014-96-1-0050.

[‡]Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213, USA (`reha+@andrew.cmu.edu`). Research supported in part by NSF through grant DMS-9706950.

1 Introduction

This is a software package for solving the standard SDP problem:

$$
\begin{array}{rrcl}
(P) & & \min_X \; C \bullet X & \\
 & A_k \bullet X & = & b_k, \quad k = 1, \ldots, m \\
 & X & \succeq & 0,
\end{array}
\qquad (1)
$$

where $A_k \in \mathcal{H}^n$, $C \in \mathcal{H}^n$ and $b \in I\!R^m$ are given data, and $X \in \mathcal{H}^n$ is the variable, possibly complex. Here $\mathcal{H}^n$ denotes the space of $n \times n$ Hermitian matrices, $P \bullet Q$ denotes the inner product $\mathrm{Tr}(P^* Q)$, and $X \succeq 0$ means that X is positive semidefinite. We assume that the set $\{A_1, \ldots, A_k\}$ is linearly independent. (Linearly dependent constraints are allowed; these are detected and removed automatically. However, if this set is nearly dependent, transformation to a better-conditioned basis may be advisable for numerical stability.) The software also solves the dual problem associated with (P):

$$
\begin{array}{rrcl}
(D) & & \max_{y,Z} \; b^T y & \\
 & \sum_{k=1}^m y_k A_k + Z & = & C \\
 & Z \succeq 0, & &
\end{array}
\qquad (2)
$$

where $y \in I\!R^m$ and $Z \in \mathcal{H}^n$ are the variables.

This package is written in MATLAB version 5.0. It is available from the internet sites:

`http://www.math.nus.edu.sg/~mattohkc/index.html`

`http://www.math.cmu.edu/~reha/sdpt3.html`

The purpose of this software package is to provide researchers in SDP with a collection of reasonably efficient and robust algorithms that can solve general SDPs with matrices of dimensions of the order of a hundred. If your problem is large-scale, you should probably use an implementation that exploits problem structure. The only structures we exploit in this package are sparsity and block-diagonal structure, where MATLAB cell arrays are used to handle dense and sparse blocks separately. We hope that researchers in SDP may benefit from the algorithmic and computational framework provided by our software in developing their own algorithms. We also hope that the computational results provided here will be useful for benchmarking. To facilitate other authors in evaluating the performance of their own algorithms, we include a few classes of SDP problems in this software package as well.

Our software is designed for general SDPs, where we do not exploit any special structures present in the data $A_1, \cdots, A_m$ and C except for sparsity and block diagonal structures as mentioned above. For SDP problems where the data has additional structure, such as that arising from the SDP relaxations of the maximum cut or graph partitioning problems, specialized algorithms such as the dual-scaling algorithm proposed by Benson et al. [5] and the nonsmooth methods proposed by Helmberg and

Rendl [13] and Burer and Monteiro [10] can certainly outperform a general purpose code like ours by orders of magnitude to achieve moderately accurate solutions. However, for problems with sparsity and structure that is not well understood (and so a specialized code is not available), our generic approach exploiting sparsity is worth trying.

A special feature that distinguishes this SDP software from others (e.g., [4],[8],[7],[12],[31]) is that complex data are allowed, a feature shared by the SeDuMi code of Sturm [25]. But note that b and y must be real. Another feature of our package, also shared by the software of [7],[12] and [25], is that the sparsity of matrices A_k is fully exploited in the computation of the Schur complement matrix required at each iteration of our SDP algorithms. Lastly, we calculate the step-lengths required for the iterates in each interior-point iteration via the Lanczos iteration [24]. This method is cheaper compared to the backtracking scheme with Cholesky factorization and the QR algorithm currently employed in all the SDP softwares mentioned in this paper.

Part of the codes for real symmetric matrices is originally based on those by Alizadeh, Haeberly, and Overton, whose help we gratefully acknowledge.

Section 2 discusses our infeasible-interior-point algorithms, while our homogeneous self-dual methods are described in Section 4. Initialization is detailed in Section 4, and Section 5 outlines how the package is called and input and output arguments. In Section 6 we give some examples, while Section 7 contains some sample runs. Section 8 describes some specialized routines for computing the Schur complement matrix, and we conclude in Section 8 with some numerical results.

2 Infeasible-interior-point algorithms

2.1 The search direction

For later discussion, let us first introduce the operator $\mathcal{A}$ defined by

$$\begin{aligned} \mathcal{A} : \mathcal{H}^n &\longrightarrow I\!R^m, \\ \mathcal{A}X &= \begin{pmatrix} A_1 \bullet X \\ \vdots \\ A_m \bullet X \end{pmatrix}. \end{aligned} \tag{3}$$

The adjoint of $\mathcal{A}$ with respect to the standard inner products in $\mathcal{H}^n$ and $I\!R^m$ is the operator

$$\begin{aligned} \mathcal{A}^* : I\!R^m &\longrightarrow \mathcal{H}^n, \\ \mathcal{A}^* y &= \textstyle\sum_{k=1}^m y_k A_k. \end{aligned} \tag{4}$$

The main step at each iteration of our algorithms is the computation of the search direction $(\Delta X, \Delta y, \Delta Z)$ from the *symmetrized Newton equation* (with respect to an invertible matrix P which is usually chosen as a function of the current iterate X, Z) given below.

$$\boxed{\begin{array}{lllllll} & \mathcal{A}^*\Delta y & + & \Delta Z & = R_d & := & C - Z - \mathcal{A}^* y \\ \mathcal{A}\Delta X & & & & = r_p & := & b - \mathcal{A}X \\ \mathcal{E}\Delta X & & + & \mathcal{F}\Delta Z & = R_c & := & \sigma\mu I - H_P(XZ), \end{array}} \tag{5}$$

where $\mu = X \bullet Z/n$ and σ is the centering parameter. Here H_P is the symmetrization operator defined by

$$\begin{aligned} H_P : \mathbb{C}^{n\times n} &\longrightarrow \mathcal{H}^n \\ H_P(U) &= \tfrac{1}{2}\left[PUP^{-1} + P^{-*}U^*P^*\right], \end{aligned} \tag{6}$$

and $\mathcal{E}$ and $\mathcal{F}$ are the linear operators

$$\mathcal{E} = P \circledast P^{-*}Z, \qquad \mathcal{F} = PX \circledast P^{-*}, \tag{7}$$

where $R \circledast T$ denotes the linear operator defined by

$$\begin{aligned} R \circledast T : \mathcal{H}^n &\longrightarrow \mathcal{H}^n \\ R \circledast T(U) &= \tfrac{1}{2}\left[RUT^* + TUR^*\right]. \end{aligned} \tag{8}$$

Assuming that $m = \mathcal{O}(n)$, we compute the search direction via a Schur complement equation as follows (the reader is referred to [3] and [26] for details). First compute Δy from the Schur complement equation

$$M\Delta y = h, \tag{9}$$

where

$$M = \mathcal{A}\mathcal{E}^{-1}\mathcal{F}\mathcal{A}^*, \tag{10}$$

$$h = r_p + \mathcal{A}\mathcal{E}^{-1}\mathcal{F}(R_d) - \mathcal{A}\mathcal{E}^{-1}(R_c). \tag{11}$$

Then compute ΔX and ΔZ from the equations

$$\Delta Z = R_d - \mathcal{A}^*\Delta y \tag{12}$$

$$\Delta X = \mathcal{E}^{-1}R_c - \mathcal{E}^{-1}\mathcal{F}(\Delta Z). \tag{13}$$

If $m \gg n$, solving (9) by a direct method is overwhelmingly expensive; in this case, the user should consider using an implementation that solves (9) by an iterative method such as the conjugate gradient or quasi-minimal residual method [23]. In our package, (9) is solved by a direct method such as LU or Cholesky decomposition with the implicit assumption that $m = \mathcal{O}(n)$ and m is at most a few hundred. If the SDP data is dense, we recommend that n is no more than about 200 so that the SDP problem can be comfortably solved on a fast workstation with, say, 200MHz speed.

2.2 Computation of specific search directions

In this package, the user has four choices of symmetrizations resulting in four different search directions, namely,

(1) the AHO direction [3], corresponding to $P = I$;

(2) the HKM direction [14],[16],[21], corresponding to $P = Z^{1/2}$;

(3) the NT direction [26], corresponding to $P = N^{-1}$, where N is the unique matrix such that $D := N^*ZN = N^{-1}XN^{-*}$ is a diagonal matrix (then $W := NN^*$ is the NT scaling matrix with $WZW = X$); and

(4) the GT direction [27], corresponding to $P = \bar{D}^{1/2}\bar{G}^{-*}$, where the matrices $\bar{D}$ and $\bar{G}$ are defined as follows. Suppose $X = G^*G$ and $Z = H^*H$ are the Cholesky factorizations of X and Z respectively. Let the SVD of GH^* be $GH^* = U\Sigma V^*$. Let Ψ and Φ be positive diagonal matrices such that the equalities $U^*G = \Psi\bar{G}$ and $V^*H = \Phi\bar{H}$ hold, with all the rows of $\bar{G}$ and $\bar{H}$ having unit norms. Then $\bar{D} = \Sigma(\Psi\Phi)^{-1}$.

To describe our implementation SDPT3, a discussion on the efficient computation of the Schur complement matrix M is necessary, since this is the most expensive step in each iteration of our algorithms where usually at least 50% to 80% of the total CPU time is spent. From equation (10), it is easily shown that the (i,j) element of M is given by

$$M_{ij} \;=\; A_i \bullet \mathcal{E}^{-1}\mathcal{F}(A_j). \tag{14}$$

Thus for a fixed j, computing first the matrix $\mathcal{E}^{-1}\mathcal{F}(A_j)$ and then taking its inner product with each A_i, $i = 1,\ldots,m$, give the jth column of M.

However, the computation of M for the four search directions mentioned above can also be arranged in a different way. The operator $\mathcal{E}^{-1}\mathcal{F}$ corresponding to these four directions can be decomposed generically as

$$\mathcal{E}^{-1}\mathcal{F}(A_j) \;=\; (R^* \circledast T^*)(\boldsymbol{D}_1 \circ [(\boldsymbol{D}_2 \circledast I)(R \circledast T(A_j))]), \tag{15}$$

where $\circ$ denotes the Hadamard (elementwise) product and the matrices R, T, $\boldsymbol{D}_1$, and $\boldsymbol{D}_2$ depend only on X and Z. (Note that for the HKM direction, $R \circledast T$ should be replaced by the linear map defining the Kronecker product $R \otimes T$ in (15).) Thus the (i,j) element of M in (14) can be written equivalently as

$$M_{ij} \;=\; (R \circledast T(A_i)) \bullet (\boldsymbol{D}_1 \circ [(\boldsymbol{D}_2 \circledast I)(R \circledast T(A_j))]). \tag{16}$$

Therefore the Schur complement matrix M can also be formed by first computing and storing $R \circledast T(A_j)$ for each $j = 1,\ldots,m$, and then taking inner products as in (16).

Computing M via different formulas, (14) or (16), will result in different computational complexities. Roughly speaking, if most of the matrices A_k are dense, then it is cheaper to use (16). However, if most of the matrices A_k are sparse, then using (14) will be cheaper because the sparsity of the matrices A_k can be exploited in (14) when taking inner products. For the sake of completeness, in Table 1, we give an upper bound on the complexity of computing M for the above mentioned search directions when computed via (14) and (16). (Here we have assumed that all A_k's are dense; if they are block diagonal with dense blocks, each term n^3 or n^2 should be replaced by a sum of the cubes or squares of the block dimensions.)

directions	upper bound on complexity using (16)	upper bound on complexity using (14)
AHO	$4mn^3 + m^2n^2$	$6\frac{1}{3}mn^3 + m^2n^2$
HKM	$2mn^3 + m^2n^2$	$4mn^3 + 0.5m^2n^2$
NT	$mn^3 + 0.5m^2n^2$	$2mn^3 + 0.5m^2n^2$
GT	$2mn^3 + 0.5m^2n^2$	$4\frac{1}{3}mn^3 + 0.5m^2n^2$

Table 1: *Upper bounds on the complexities of computing M (for real SDP data) for various search directions. We count one addition and one multiplication each as one flop. Note that all directions other than the HKM direction require an eigenvalue decomposition of a symmetric matrix in the computation of M.*

The reader is referred to [3], [26], and [27] for more computational details, such as the actual formation of M and h, involved in computing the above search directions. The derivation of the upper bounds on the computational complexities of M computed via (14) and (16) is given in [27]. The issue of exploiting the sparsity of the matrices A_k is discussed in full detail in [11] for the HKM and NT directions, whereas an analogous discussion for the AHO and GT directions can be found in [27].

Let `NZ` be the total number of nonzero elements of $A_1, \cdots, A_m$. In our implementation, we consider the following two cases in exploiting possible sparsity in the SDP data:

if `NZ` exceeds a certain fraction of mn^2,

> we decide on the formula to use for computing M based on the CPU time taken during the third and fourth iteration to compute M via (16) and (14), respectively. We do not base our decision on the first two iterations for two reasons. Firstly, if the initial iterates X^0 and Z^0 are diagonal matrices, then the CPU time taken to compute M during these two iterations would not be an accurate estimate of the time required for subsequent iterations. Secondly, there are overheads incurred when variables are first loaded into MATLAB workspace.

else

> we use (14) throughout.

We should mention that when (14) is used, we further exploit the sparsity of the matrices $A_1, \cdots, A_m$ based on the ideas proposed in [12]. The reader is referred to

[12] for the details.

2.3 The primal-dual path-following algorithm

The algorithmic framework of our primal-dual path-following algorithm is as follows. We note that most of our implementation choices here and in our other algorithms are based on minimizing either the number of iterations or the CPU time of the linear algebra involved in computing the Schur complement. In the computation of the eigenvalues necessary in our choice of step-size, we used an iterative solver, the Lanczos iteration. If we assume that the Cholesky factors of X and Z are given, this method requires only $\mathcal{O}(n^2)$ flops instead of $\mathcal{O}(n^3)$ flops as required by Cholesky factorization or the QR algorithm to compute the step-size. A detailed discussion on computing the step-size via the Lanczos iteration is given in [29].

Algorithm IPF. Suppose we are given an initial iterate (X^0, y^0, Z^0) with X^0, Z^0 positive definite. Decide on the symmetrization operator $H_P(\cdot)$ to use. Set $\gamma^0 = 0.9$ and $\sigma^0 = 0.5$.

For $k = 0, 1, \ldots$

(Let the current and the next iterate be (X, y, Z) and (X^+, y^+, Z^+) respectively. Also, let the current and the next step-length (centering) parameter be denoted by γ and γ^+ (σ and σ^+) respectively.)

- Set $\mu = X \bullet Z/n$ and

  $$\phi \;=\; \max\left(\frac{\|r_p\|}{\max(1,\|b\|)}, \frac{\|R_d\|_F}{\max(1,\|C\|_F)}\right). \tag{17}$$

  Stop the iteration if the infeasibility measure ϕ and the duality gap $X \bullet Z$ are sufficiently small.

- Compute the search direction $(\Delta X, \Delta y, \Delta Z)$ from the equations (9), (12) and (13).

- Update (X, y, Z) to (X^+, y^+, Z^+) by

  $$X^+ \;=\; X + \alpha\,\Delta X, \quad y^+ \;=\; y + \beta\,\Delta y, \quad Z^+ \;=\; Z + \beta\,\Delta Z, \tag{18}$$

  where

  $$\alpha \;=\; \min\left(1, \frac{-\gamma}{\lambda_{\min}(X^{-1}\Delta X)}\right), \quad \beta \;=\; \min\left(1, \frac{-\gamma}{\lambda_{\min}(Z^{-1}\Delta Z)}\right). \tag{19}$$

  (Here $\lambda_{\min}(U)$ denotes the minimum eigenvalue of U; if the minimum eigenvalue in either expression is positive, we ignore the corresponding term.)

- Update the step-length parameter by

  $$\gamma^+ \;=\; 0.9 + 0.09\min(\alpha, \beta), \tag{20}$$

  and the centering parameter by $\sigma^+ \;=\; 1 - 0.9\min(\alpha, \beta)$.

Remarks.

(a) Note that, if $\alpha < 1$ in (19), then the step is γ times that making X^+ positive semidefinite but not positive definite, and similarly for β. Hence the step is a constant (γ) multiple of the longest feasible step, or the full Newton-like step. The adaptive choice of the step-length parameter γ in (20) is used as the default in our implementation, but the user has the option of fixing the value of γ. The motivation for using adaptive step-length and centering parameters is as follows. If large step sizes for α and β have been taken, this indicates that good progress was made in the previous iteration, so more aggressive values for the step-length parameter γ^+ and centering parameter σ^+ can be chosen for the next iteration so as to get as much reduction in the total complementarity $X \bullet Z$ (we often call this the duality gap; it is equal if both iterates are feasible) and infeasibilities as possible. On the other hand, if either α or β is small, this indicates that (X^+, y^+, Z^+) is close to the boundary of $\mathcal{H}^n_+$: in this case, we would want to concentrate more on centering in the next iteration by using less aggressive values for γ^+ and σ^+.

(b) If ΔX and ΔZ are orthogonal (which certainly holds if both iterates are feasible) and equal steps are taken in both primal and dual, then the reduction in the infeasibilities is exactly by the factor $(1-\alpha)$, and that in the total complementarity is exactly by the factor $(1-\alpha(1-\sigma))$; thus we expect the infeasibilities to decrease faster than the total complementarity. We often observed this behavior in practice and do not otherwise ensure that infeasibilities decrease faster than the total complementarity.

(c) It is known that as the parameter μ decreases to 0, the norm $\|r_p\|$ will tend to increase, even if the initial iterate is primal feasible, due to increasing numerical instability of the Schur complement approach. In our implementation of the algorithms, the user has the option of correcting for the loss in primal feasibility by projecting ΔX onto the null space of the operator $\mathcal{A}$. That is, before updating to X^+, we replace ΔX by

$$\Delta X - \mathcal{A}^* \mathcal{D}^{-1} \mathcal{A}\,(\Delta X),$$

where $\mathcal{D} = \mathcal{A}\mathcal{A}^*$. Note that this step is inexpensive. The $m \times m$ matrix $\mathcal{D}$ only needs to be formed once at the beginning of the algorithm. Typically, this option might lose one order of magnitude in the duality gap achieved, but gain sometimes two or three orders of magnitude in the final primal feasibility.

(d) We only described termination when approximately optimal solutions are at hand. Nevertheless, our codes stop when any of the following indications arise:

- The step-length taken in either primal or dual spaces becomes very small, in which case the message `‘‘steps in predictor too short’’` will be displayed.
- If μ and ϕ are both less than 10^{-8}, we terminate if the reduction in the total complementarity is significantly worse than that for the previous few iterations, in which case the message `‘‘lack of progress in the predictor step’’` will be displayed.

- We also stop if we get an indication of infeasibility, as follows. If the current iterate has $b^T y$ much larger than $\|\mathcal{A}^* y + Z\|$, then a suitable scaling is an approximate solution of $b^T y = 1$, $\mathcal{A}^* y + Z = 0$, $Z \succeq 0$, which is a certificate that the primal problem is infeasible. Similarly, if $-C \bullet X$ is much larger than $\|\mathcal{A}X\|$, we have an indication of dual infeasibility: a scaled iterate is then an approximate solution of $C \bullet X = -1$, $\mathcal{A}X = 0$, $X \succeq 0$, which is a certificate that the dual problem is infeasible. In either case, we return the appropriate scaled iterate suggesting primal or dual infeasibility.
- Termination also occurs if either X or Z is numerically not positive definite, if the Schur complement matrix M becomes singular or too ill-conditioned for satisfactory progress, or if the iteration limit is reached. Our other algorithms described in the following pages also terminate if one of these situations occurs.

2.4 The Mehrotra-type predictor-corrector algorithm

The algorithmic framework of the Mehrotra-type predictor-corrector [19] variant of the previous algorithm is as follows.

Algorithm IPC. Suppose we are given an initial iterate (X^0, y^0, Z^0) with X^0, Z^0 positive definite. Decide on the type of symmetrization operator $H_P(\cdot)$ to use. Set $\gamma^0 = 0.9$. Choose a value for the parameter *expon* used in the exponent e.

For $k = 0, 1, \ldots$

(Let the current and the next iterate be (X, y, Z) and (X^+, y^+, Z^+) respectively, and similarly for γ and γ^+.)

- Set $\mu = X \bullet Z/n$ and ϕ as in (17). Stop the iteration if the infeasibility measure ϕ and the duality gap $X \bullet Z$ are sufficiently small.
- (Predictor step)
  Solve the linear system (9), (12) and (13) with $\sigma = 0$, i.e., with $R_c = -H_P(XZ)$. Denote the solution by $(\delta X, \delta y, \delta Z)$. Let α_p and β_p be the step-lengths defined as in (19) with $\Delta X, \Delta Z$ replaced by $\delta X, \delta Z$, respectively.
- Take σ to be

$$\sigma \;=\; \min\left(1, \left[\frac{(X + \alpha_p\,\delta X) \bullet (Z + \beta_p\,\delta Z)}{X \bullet Z}\right]^e\right), \tag{21}$$

  where the exponent e is chosen as follows:

$$e = \begin{cases} 1 & \text{if } \mu > 10^{-6} \text{ and } \min(\alpha_p, \beta_p) < 1/\sqrt{3}, \\ \max[\mathit{expon}, 3\min(\alpha_p, \beta_p)^2] & \text{if } \mu > 10^{-6} \text{ and } \min(\alpha_p, \beta_p) \geq 1/\sqrt{3}, \\ \max[1, \min[\mathit{expon}, 3\min(\alpha_p, \beta_p)^2]] & \text{if } \mu \leq 10^{-6}. \end{cases} \tag{22}$$

- (Corrector step)
  Compute the search direction $(\Delta X, \Delta y, \Delta Z)$ from the same linear system (9), (12) and (13) but with R_c replaced by

$$R_q \;=\; \sigma\mu I - H_P(XZ) - H_P(\delta X \delta Z).$$

- Update (X, y, Z) to (X^+, y^+, Z^+) as in (18), where α and β are computed as in (19) with γ chosen to be

$$\gamma \;=\; 0.9 + 0.09\min(\alpha_p, \beta_p).$$

- Update γ to γ^+ as in (20).

Remarks.

(a) The default choices of *expon* for the AHO, HKM, NT, and GT directions are *expon* $= 3, 1, 1, 1$, respectively. We observed experimentally that using *expon* $=$ 2 for the HKM and NT directions seems to be too aggressive, and usually results in slightly poorer numerical stability when μ is small than the choice *expon* $= 1$. We should mention that the choice of the exponent e in Algorithm IPC above is only a rough guide. The user might want to explore other possibilities.

(b) In our implementation, the user has the option to switch from Algorithm IPF to Algorithm IPC once the infeasibility measure ϕ is below a certain threshold specified by the variable `sw2PC_tol`. This option is for the convenience of the user; such a strategy was recommended in an earlier version of [3].

(c) Once again, we also terminate if the primal or dual step-length is too small, in which case the message `‘‘steps in predictor too short’’` or `‘‘steps in corrector too short’’` will be displayed, or we get an indication of primal or dual infeasibility. If μ and ϕ are both less than 10^{-8}, we terminate if the reduction in the total complementarity in the predictor step is significantly worse than that for the previous few iterations. Similar termination also applies to the corrector step with displayed message `‘‘lack of progress in the corrector step’’`, unless the reduction in the total complementarity corresponding to the predictor point $(X + \alpha_p\, \delta X, y + \beta_p \delta y, Z + \beta_p\, \delta Z)$ is satisfactory. In the latter case, the iterate (X^+, y^+, Z^+) is taken to be this good predictor point and a corresponding message `‘‘update to good predictor point’’` is displayed.

3 Homogeneous and self-dual algorithms

Homogeneous embedding of an SDP in a self-dual problem was first developed by Potra and Sheng [22], and subsequently extended independently by Luo et al. [17] and de Klerk et al. [15]. The implementation of such homogeneous and self-dual algorithms for SDP first appeared in [9], where they are based on those appearing in [32] for linear programming (LP). Our algorithms are also the SDP extensions of those appearing in [32] for LP. However, we use a different criterion for choosing equal versus unequal primal and dual step-lengths in the iteration process.

3.1 The search direction

Let $\hat{\mathcal{A}}$ be the operator

$$\begin{aligned} \hat{\mathcal{A}} : \mathcal{H}^n &\longrightarrow I\!R^{m+1}, \\ \hat{\mathcal{A}} X &= \begin{pmatrix} A_1 \bullet X \\ \vdots \\ A_m \bullet X \\ -C \bullet X \end{pmatrix}. \end{aligned}$$

Then the adjoint of $\hat{\mathcal{A}}$ with respect to the standard inner products in $\mathcal{H}^n$ and $I\!R^{m+1}$ is the operator

$$\begin{aligned} \hat{\mathcal{A}}^* : I\!R^{m+1} &\longrightarrow \mathcal{H}^n, \\ \hat{\mathcal{A}}^* \begin{pmatrix} y \\ \tau \end{pmatrix} &= \textstyle\sum_{k=1}^m y_k A_k \,-\, \tau C. \end{aligned}$$

Our homogeneous and self-dual linear feasibility model for SDP based on that appearing in [32] for LP has the following form:

$$\begin{aligned} \hat{\mathcal{A}}^* \begin{pmatrix} y \\ \tau \end{pmatrix} + Z &= 0 \\ \hat{\mathcal{A}} X + \begin{pmatrix} 0 & -b \\ b^T & 0 \end{pmatrix} \begin{pmatrix} y \\ \tau \end{pmatrix} - \begin{pmatrix} 0 \\ \kappa \end{pmatrix} &= 0 \\ XZ = 0, \quad \tau\kappa &= 0, \end{aligned} \tag{23}$$

where X, Z, τ, κ are positive semidefinite. A solution to this system with $\tau + \kappa$ positive either gives optimal solutions to the SDP and its dual or gives a certificate of primal or dual infeasibility.

At each iteration of our algorithms, we apply a variant of Newton's method to a perturbation of equation (23), namely,

$$\begin{aligned} \hat{\mathcal{A}}^* \begin{pmatrix} y \\ \tau \end{pmatrix} + Z &= 0 \\ \hat{\mathcal{A}} X + \begin{pmatrix} 0 & -b \\ b^T & 0 \end{pmatrix} \begin{pmatrix} y \\ \tau \end{pmatrix} - \begin{pmatrix} 0 \\ \kappa \end{pmatrix} &= 0 \\ XZ = \sigma\mu I, \quad \tau\kappa &= \sigma\mu, \end{aligned} \tag{24}$$

where σ is a parameter.

Similarly to the case of infeasible path-following methods, the search direction $(\Delta X, \Delta y, \Delta Z, \Delta\tau, \Delta\kappa)$ at each iteration of our homogeneous algorithms is computed from a symmetrized Newton equation derived from the perturbed equation (24), but now with an extra perturbation added, controlled by the parameter η:

$$\boxed{\begin{array}{llll} & \hat{\mathcal{A}}^* \begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} & + \;\; \Delta Z & = \eta\, \hat{R}_d \\ \hat{\mathcal{A}}\Delta X \;+ & \begin{pmatrix} 0 & -b \\ b^T & \kappa/\tau \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} & & = \eta\, \hat{r}_p + \begin{pmatrix} 0 \\ r_c/\tau \end{pmatrix} \\ \mathcal{E}\Delta X & & + \;\; \mathcal{F}\Delta Z & = R_c \\ & & \tau\Delta\kappa \;+\; \kappa\Delta\tau & = r_c \end{array}} \tag{25}$$

where $H_P(XZ)$ and $\mathcal{E}$, $\mathcal{F}$ are defined as in (6) and (7), respectively, and

$$\hat{R}_d \;:=\; -\hat{\mathcal{A}}^* \begin{pmatrix} y \\ \tau \end{pmatrix} - Z, \tag{26}$$

$$\hat{r}_p := \begin{pmatrix} 0 \\ \kappa \end{pmatrix} - \begin{pmatrix} 0 & -b \\ b^T & 0 \end{pmatrix} \begin{pmatrix} y \\ \tau \end{pmatrix} - \hat{\mathcal{A}}X, \tag{27}$$

$$\mu := \frac{X \bullet Z + \tau\kappa}{n+1}, \tag{28}$$

$$r_c := \sigma\mu - \tau\kappa,$$

$$R_c := \sigma\mu I - H_P(XZ).$$

In fact, we choose the parameter η to be $1-\sigma$. This ensures (with any of our choices of P) that $\Delta X \bullet \Delta Z + \Delta\tau\Delta\kappa = 0$, so that equal step sizes of α in both primal and dual will give new iterates with infeasibilities and total complementarity reduced by the same factor $1-\alpha(1-\sigma)$. This aids convergence. Also in this case, the search directions coincide with those derived from the semidefinite extension of the homogeneous self-dual programming approach of Ye, Todd, and Mizuno [33].

We compute the search direction via a Schur complement equation as follows. First compute $\Delta y, \Delta\tau$ from the equation

$$\left[\hat{M} + \begin{pmatrix} 0 & -b \\ b^T & \kappa/\tau \end{pmatrix}\right] \begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} = \hat{h}, \tag{29}$$

where

$$\hat{M} = \hat{\mathcal{A}}\mathcal{E}^{-1}\mathcal{F}\hat{\mathcal{A}}^*, \tag{30}$$

$$\hat{h} = \eta\,\hat{r}_p + \begin{pmatrix} 0 \\ r_c/\tau \end{pmatrix} + \eta\,\hat{\mathcal{A}}\mathcal{E}^{-1}\mathcal{F}\hat{R}_d - \hat{\mathcal{A}}\mathcal{E}^{-1}R_c. \tag{31}$$

Then compute ΔX, ΔZ, and $\Delta\kappa$ from the equations

$$\begin{aligned} \Delta Z &= \eta\,\hat{R}_d - \hat{\mathcal{A}}^*\begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} \\ \Delta X &= \mathcal{E}^{-1}R_c - \mathcal{E}^{-1}\mathcal{F}\Delta Z \\ \Delta\kappa &= (r_c - \kappa\Delta\tau)/\tau. \end{aligned} \tag{32}$$

3.2 Primal and dual step-lengths

As in the case of infeasible path-following algorithms, using different step-lengths in the primal and dual updates under appropriate conditions can enhance the performance of homogeneous algorithms. Our purpose now is to establish one such condition.

Suppose we are given the search direction $(\Delta X, \Delta y, \Delta Z, \Delta\tau, \Delta\kappa)$. Let

$$\tau_p = \tau + \alpha\Delta\tau, \qquad \tau_d = \tau + \beta\Delta\tau. \tag{33}$$

Suppose (X, y, Z, τ, κ) is updated to $(X^+, y^+, Z^+, \tau^+, \kappa^+)$ by

$$\begin{gathered} \tau^+ = \min(\tau_p, \tau_d), \qquad \kappa^+ = \begin{cases} \kappa + \alpha\,\Delta\kappa, & \text{if } \tau^+ = \tau_p \\ \kappa + \beta\,\Delta\kappa, & \text{if } \tau^+ = \tau_d. \end{cases} \\ X^+ = \tfrac{\tau^+}{\tau_p}(X + \alpha\,\Delta X), \quad y^+ = \tfrac{\tau^+}{\tau_d}(y + \beta\,\Delta y), \quad Z^+ = \tfrac{\tau^+}{\tau_d}(Z + \beta\,\Delta Z). \end{gathered} \tag{34}$$

Then it can be shown that the scaled primal and dual infeasibilities, defined respectively by

$$r_p^+(\alpha) \;=\; b - \mathcal{A}(X^+/\tau^+), \qquad R_d^+(\beta) \;=\; C - Z^+/\tau^+ - \mathcal{A}^*(y^+/\tau^+), \tag{35}$$

satisfy the relation

$$r_p^+(\alpha) \;=\; \frac{1-\alpha\eta}{1+\alpha\Delta\tau/\tau}\, r_p, \qquad R_d^+(\beta) \;=\; \frac{1-\beta\eta}{1+\beta\Delta\tau/\tau}\, R_d, \tag{36}$$

where

$$r_p = b - \mathcal{A}(X/\tau), \qquad R_d = C - Z/\tau - \mathcal{A}^*(y/\tau). \tag{37}$$

Our condition is basically determined by considering reductions in the norms of the scaled infeasibilities. To determine this condition, let us note that the function $f(t) := (1-t\eta)/(1+t\Delta\tau/\tau)$, $t \in [0,1]$, is decreasing if $\Delta\tau \geq -\eta\tau$ and increasing otherwise. Using this fact, we see that the norms of the scaled infeasibilities $r_p^+(\alpha), R_d^+(\beta)$ are decreasing functions of the step-lengths if $\Delta\tau \geq -\eta\tau$ and they are increasing functions of the step-lengths otherwise.

Let $\hat{\alpha}$ and $\hat{\beta}$ be γ times the maximum possible primal and dual step-lengths that can be taken for the primal and dual updates, respectively (where $0 < \gamma < 1$). To keep the possible amplifications in the norms of the scaled infeasibilities to a minimum, we set α and β to be $\min(\hat{\alpha}, \hat{\beta})$ when $\Delta\tau < -\eta\tau$; otherwise, it is beneficial to take the maximum possible primal and dual step-lengths so as to get the maximum possible reductions in the scaled infeasibilities. For the latter case, we take different step-lengths, $\alpha = \hat{\alpha}$ and $\beta = \hat{\beta}$, provided that the resulting scaled total complementarity is also reduced, that is, if

$$\frac{X^+ \bullet Z^+ + \tau^+\kappa^+}{(\tau^+)^2} \;\leq\; \frac{X \bullet Z + \tau\kappa}{\tau^2}, \tag{38}$$

when we substitute $\alpha = \hat{\alpha}$ and $\beta = \hat{\beta}$ into (33) and (34).

To summarize, we take different step-lengths, $\alpha = \hat{\alpha}$ and $\beta = \hat{\beta}$, for the primal and dual updates only when $\Delta\tau \geq -\eta\tau$ and (38) holds; otherwise, we take the same step-length $\min(\hat{\alpha}, \hat{\beta})$ for α and β.

3.3 The homogeneous path-following algorithm

Our homogeneous self-dual path-following algorithms and their Mehrotra-type predictor-corrector variants are modeled after those proposed in [32] for LP and the infeasible path-following algorithms discussed in Section 2.

The algorithmic framework of these homogeneous path-following algorithm is as follows.

Algorithm HPF. Suppose we are given an initial iterate $(X^0, y^0, Z^0, \tau^0, \kappa^0)$ with $X^0, Z^0, \tau^0, \kappa^0$ positive definite. Decide on the symmetrization operator $H_P(\cdot)$ to use. Set $\gamma^0 = 0.9$, $\sigma^0 = 0.1$, and $\eta^0 = 0.9$.

For $k = 0, 1, \dots$

(Let the current and the next iterate be (X, y, Z, τ, κ) and $(X^+, y^+, S^+, \tau^+, \kappa^+)$ respectively. Also, let the current and the next step-length (centering) parameter be denoted by γ and γ^+ (σ and σ^+) respectively.)

- Set $\mu = (X \bullet Z + \tau\kappa)/(n+1)$ and

$$\phi = \max\left(\frac{\|\tau b - \mathcal{A}X\|}{\tau \max(1, \|b\|)}, \frac{\|\tau C - Z - \mathcal{A}^* y\|_F}{\tau \max(1, \|C\|_F)}\right). \tag{39}$$

  Stop the iteration if either of the following occurs:

  (a) The infeasibility measure ϕ and the duality gap $X \bullet Z/\tau^2$ are sufficiently small. In this case, $(X/\tau, y/\tau, Z/\tau)$ is an approximately optimal solution of the given SDP and its dual.

  (b)

$$\max\left(\frac{\mu}{\mu_0}, \frac{\tau/\kappa}{\tau_0/\kappa_0}\right) \quad \text{is sufficiently small.}$$

  In this case, either the primal or the dual problem (or both) is suspected to be infeasible.

- Compute the search direction $(\Delta X, \Delta y, \Delta Z, \Delta\tau, \Delta\kappa)$ from the equations (29)–(32) using $\eta = 1 - \sigma$.

- Let

$$\begin{aligned} \hat{\alpha} &= \min\left(1, \frac{-\gamma}{\lambda_{\min}(X^{-1}\Delta X)}, \frac{-\gamma}{\tau^{-1}\Delta\tau}, \frac{-\gamma}{\kappa^{-1}\Delta\kappa}\right), \\ \hat{\beta} &= \min\left(1, \frac{-\gamma}{\lambda_{\min}(Z^{-1}\Delta Z)}, \frac{-\gamma}{\tau^{-1}\Delta\tau}, \frac{-\gamma}{\kappa^{-1}\Delta\kappa}\right). \end{aligned} \tag{40}$$

  (If any of the denominators in either expression is positive, we ignore the corresponding term.)

  If $\Delta\tau \geq -\eta\tau$ and (38) holds, set $\alpha = \hat{\alpha}$ and $\beta = \hat{\beta}$; otherwise, set $\alpha = \beta = \min(\hat{\alpha}, \hat{\beta})$.

- Let

$$\tau_p \;=\; \tau + \alpha\Delta\tau, \quad \tau_d \;=\; \tau + \beta\Delta\tau.$$

Update (X, y, Z, τ, κ) to $(X^+, y^+, Z^+, \tau^+, \kappa^+)$ by

$$\begin{aligned} \tau^+ \;&=\; \min(\tau_p, \tau_d), \qquad \kappa^+ \;=\; \begin{cases} \kappa + \alpha\,\Delta\kappa, & \text{if } \tau^+ = \tau_p \\ \kappa + \beta\,\Delta\kappa, & \text{if } \tau^+ = \tau_d, \end{cases} \\ X^+ \;&=\; \tfrac{\tau^+}{\tau_p}(X + \alpha\,\Delta X), \quad y^+ \;=\; \tfrac{\tau^+}{\tau_d}(y + \beta\,\Delta y), \quad Z^+ \;=\; \tfrac{\tau^+}{\tau_d}(Z + \beta\,\Delta Z). \end{aligned} \tag{41}$$

- Update the step-length parameter by

$$\gamma^+ \;=\; 0.9 + 0.09\min(\alpha, \beta), \tag{42}$$

and let

$$\sigma^+ = 1 - 0.9\min(\alpha, \beta).$$

3.4 The homogeneous predictor-corrector algorithm

The Mehrotra-type predictor-corrector variant of the homogeneous path-following algorithm is as follows.

Algorithm HPC. Suppose we are given an initial iterate $(X^0, y^0, Z^0, \tau^0, \kappa^0)$ with $X^0, Z^0, \tau^0, \kappa^0$ positive definite. Decide on the type of symmetrization operator $H_P(\cdot)$ to use. Set $\gamma^0 = 0.9$. Choose a value for the parameter *expon* used in the exponent e.

For $k = 0, 1, \ldots$

(Let the current and the next iterate be (X, y, Z, τ, κ) and $(X^+, y^+, Z^+, \tau^+, \kappa^+)$ respectively. Also, let the current and the next step-length parameter be denoted by γ and γ^+ respectively.)

- Set $\mu = (X \bullet Z + \tau\kappa)/(n + 1)$ and ϕ as in (39). Stop the iteration if either of the following occurs:

  (a) The infeasibility measure ϕ and the duality gap $X \bullet Z/\tau^2$ are sufficiently small.

  (b)

$$\max\left(\frac{\mu}{\mu_0}, \frac{\tau/\kappa}{\tau_0/\kappa_0}\right) \quad \text{is sufficiently small.}$$

- (Predictor step)
  Solve the linear system (29)–(32), with $\sigma = 0$ and $\eta = 1$, i.e., with $R_c = -H_P(XZ)$. Denote the solution by $(\delta X, \delta y, \delta Z, \delta\tau, \delta\kappa)$. Let α_p and β_p be defined as in (40) with $\Delta X, \Delta Z, \Delta\tau, \Delta\kappa$ replaced by $\delta X, \delta Z, \delta\tau, \delta\kappa$, respectively.

- Take σ to be

$$\sigma = \min\left(1, \left[\frac{(X+\alpha_p\,\delta X)\bullet(Z+\beta_p\,\delta Z) + (\tau+\alpha_p\,\delta\tau)(\kappa+\beta_p\,\delta\kappa)}{X\bullet Z + \tau\kappa}\right]^e\right),$$

  where the exponent e is chosen as in (22). Set $\eta = 1-\sigma$.

- (Corrector step)

  Compute the search direction $(\Delta X, \Delta y, \Delta Z, \Delta\tau, \Delta\kappa)$ from the same linear system (29)–(32) but with R_c, r_c replaced, respectively, by

$$\begin{aligned} R_q &= \sigma\mu I - H_P(XZ) - H_P(\delta X\delta Z) \\ r_q &= \sigma\mu - \tau\kappa - \delta\tau\,\delta\kappa. \end{aligned}$$

- Update (X, y, Z, τ, κ) to $(X^+, y^+, Z^+, \tau^+, \kappa^+)$ as in (41), where α and β are computed as in (40) with γ chosen to be

$$\gamma = 0.9 + 0.09\min(\alpha_p, \beta_p).$$

- Update γ to γ^+ as in (42).

Remarks for Algorithms HPF and HPC.

(a) The numerical instability of the Schur complement equation (29) arising from the homogeneous algorithms appears to be much more severe than that of the infeasible path-following algorithms as μ decreases to zero. We overcome this difficulty by first setting $\Delta\tau = 0$ and then computing Δy in (29) when μ is smaller than 10^{-8}. This amounts to switching to the infeasible path-following algorithms where the Schur complement equation (9) is numerically more stable.

(b) For the homogeneous algorithms, it seems not desirable to correct for the primal infeasibility so as keep it below a certain small level once that level has been reached via the projection step mentioned in Remark (c) of Section 2.3. The effect of such a correction can be quite erratic, in contrast to the case of the infeasible path-following algorithms described in Section 2.

(c) Once again, there are other termination criteria: lack of positive definiteness, lack of progress, or short step-lengths. We also test possible infeasibility in the same way we did for our infeasible-interior-point algorithms, because it gives explicit infeasibility certificates, and possibly gives an earlier indication than the specific termination criterion (item (b) in the algorithm descriptions above) to detect infeasibility.

(Remarks (a) and (b) are based on computational experience rather than our having any explanation at this time.)

4 Initial iterates

Our algorithms can start with an infeasible starting point. However, the performance of these algorithms is quite sensitive to the choice of the initial iterate. As observed in [12], it is desirable to choose an initial iterate that at least has the same order of magnitude as an optimal solution of the SDP. Suppose the matrices A_k and C are block diagonal with the same structure, each consisting of L diagonal blocks of square matrices of dimensions $n_1, n_2, \ldots, n_L$. Let $A_k^{(i)}$ and $C^{(i)}$ denote the ith block of A_k and C, respectively. If a feasible starting point is not known, we recommend that the following initial iterate be used:

$$X^0 \; = \; \mathrm{Diag}(\xi_i \, I_{n_i}), \quad y^0 \; = \; 0, \quad Z^0 \; = \; \mathrm{Diag}(\eta_i \, I_{n_i}), \tag{43}$$

where $i = 1, \ldots, L$, I_{n_i} is the identity matrix of order n_i, and

$$\xi_i \; = \; n_i \max_{1 \le k \le m} \frac{1 + |b_k|}{1 + \|A_k^{(i)}\|_F}, \qquad \eta_i \; = \; \frac{1 + \max[\max_k \{\|A_k^{(i)}\|_F\}, \, \|C^{(i)}\|_F]}{\sqrt{n_i}}.$$

If we multiply the identity matrix I_{n_i} by the factors ξ_i and η_i for each i, the initial iterate has a better chance of having the same order of magnitude as an optimal solution of the SDP.

The initial iterate above is set by calling `infeaspt.m`, with initial line

```
function  [X0,y0,Z0] = infeaspt(blk,A,C,b,options,scalefac),
```

where `options = 1 (default)` corresponds to the initial iterate just described, and `options = 2` corresponds to the choice where `X0`, `Z0` are `scalefac` times identity matrices and `y0` is a zero vector.

5 The main routine

The main routine that corresponds to the infeasible path-following algorithms described in Section 2 is `sdp.m`:

```
[obj,X,y,Z,gaphist,infeashist,info,Xiter,yiter,Ziter] =
sdp(blk,A,C,b,X0,y0,Z0,OPTIONS),
```

whereas the corresponding routine for the homogeneous self-dual algorithms described in Section 3 is `sdphlf.m`:

```
[obj,X,y,Z,gaphist,infeashist,info,Xiter,yiter,Ziter,
tau,kap] = sdphlf(blk,A,C,b,X0,y0,Z0,tau0,kap0,OPTIONS).
```

Functions used.

`sdp.m` calls the following function files during its execution:

`AHOpred.m`	`HKMpred.m`	`NTpred.m`	`GTpred.m`
`AHOcorr.m`	`HKMcorr.m`	`NTcorr.m`	`GTcorr.m`
`ops.m`	`blktrace.m`	`blkchol.m`	`blkeig.m`
`Asum.m`	`Prod2.m`	`Prod3.m`	`nonzerolist.m`
`steplength.m`	`validate.m`	`scaling.m`	`preprocess.m.`
`aasen.m`	`Atriu.m`	`corrprim.m`	

`sdphlf.m` calls the same set of function files except that the first two rows in the list above are replaced by

`AHOpredhlf.m`	`HKMpredhlf.m`	`NTpredhlf.m`	`GTpredhlf.m`
`AHOcorrhlf.m`	`HKMcorrhlf.m`	`NTcorrhlf.m`	`GTcorrhlf.m.`

In addition, `sdphlf.m` calls the function file `schurhlf.m`.

Input arguments.

`blk`: a cell array describing the block structure of the A_k's and C (see below).

`A` : a cell array with m columns such that the kth column corresponds to the matrix A_k.

`C`, `b`: given data of the SDP.

`X0`, `y0`, `Z0`: an initial iterate.

`tau0`, `kap0`: initial values for τ and κ (sdphlf.m only).

`OPTIONS`: a structure array of parameters — see below.

If the input argument `OPTIONS` is omitted, default values are used. For `sdphlf.m`, if also input arguments `tau0` and `kap0` are omitted, default values of 1 are used.

Output arguments.

`obj = [C` $\bullet$ `X  b`^{T}`y ]`.

`X,y,Z`: an approximately optimal solution (with normal termination; it could alternatively give an approximate certificate of infeasibility – see below).

`gaphist`: a row vector that records the total complementarity $X \bullet Z$ at each iteration.

`infeashist`: a row vector that records the infeasibility measure ϕ at each iteration.

`info`: a 1×5 vector that contains performance information:
`info(1)` = termination code,
`info(2)` = number of iterations taken,
`info(3)` = final duality gap,
`info(4)` = final infeasibility measure, and
`info(5)` = total CPU time taken.

`info(1)` takes on ten possible integral values depending on the termination conditions:
`info(1) = 0` for normal termination;
`info(1) = -1` for lack of progress in either the predictor or corrector step;
`info(1) = -2` if primal or dual step-lengths are too short;
`info(1) = -3` if the primal or dual iterates lose positive definiteness;
`info(1) = -4` if the Schur complement matrix becomes singular;
`info(1) = -5` if the Schur complement matrix becomes too ill-conditioned for further progress;
`info(1) = -6` if the iteration limit is reached;
`info(1) = -10` for incorrect input;
`info(1) = 1` if there is an indication of primal infeasibility; and
`info(1) = 2` if there is an indication of dual infeasibility.

If `info(1)` is positive, the output variables `X,y,Z` have a different meaning: if `info(1) = 1` then `y,Z` gives an indication of primal infeasibility: $\mathtt{b}^T\mathtt{y} = 1$ and $\mathcal{A}^*\mathtt{y} + \mathtt{Z}$ is small, while if `info(1) = 2` then `X` gives an indication of dual infeasibility: $\mathtt{C}\bullet\mathtt{X} = -1$ and $\mathcal{A}\mathtt{X}$ is small.

`Xiter,yiter,Ziter`: the last iterate of `sdp.m` or `sdphlf.m`. If desired, the user can continue the iteration process with this as the initial iterate.

`tau,kap`: the last values of τ and κ, for `sdphlf.m` only.

Note that the user can omit the last few output arguments if they are of no interest to him/her.

A structure array for parameters.

`sdp.m` and `sdphlf.m` use a number of parameters which are specified in a MATLAB structure array called `OPTIONS` in the M-file `parameters.m`. If desired, the user can change the values of these parameters. The meaning of the specified fields in `OPTIONS` are as follows.

`vers`: type of search direction to be used, where
`vers = 1` corresponds to the AHO direction,
`vers = 2` corresponds to the HKM direction,
`vers = 3` corresponds to the NT direction, and
`vers = 4` corresponds to the GT direction.
The default is `vers = 2`.

`gam`: step-length parameter. To use the default, set `gam = 0`; otherwise, set `gam` to the desired fixed value, say `gam = 0.98`.

`predcorr`: a 0–1 flag indicating whether to use the Mehrotra-type predictor-corrector. The default is `1`.

`expon`: a 1×4 vector specifying the lower bound for the exponent to be used in updating the centering parameter σ in the predictor-corrector algorithm, where

`expon(1)`: for the AHO direction,
`expon(2)`: for the HKM direction,
`expon(3)`: for the NT direction, and
`expon(4)`: for the GT direction.
The default is `expon = [3 1 1 1]`.

`steptol`: the step-length threshold below which the iteration is terminated.
The default is `1e-6`.

`gaptol`: the required relative accuracy in the duality gap, i.e., $X \bullet Z/[1+\max(C \bullet X, b^T y)]$.
The default is `1e-8`.

`inftol`: the tolerance for $\|\mathcal{A}^T \mathtt{y} + \mathtt{Z}\|$ (with $b^T \mathtt{y} = 1$) or $\|\mathcal{A}\mathtt{X}\|$ (with $C \bullet \mathtt{X} = -1$) in order to terminate with an indication of infeasibility. Also used as the tolerance in termination criterion (b) in the homogeneous algorithms.
The default is `1e-8`.

`maxit`: maximum number of iterations allowed. The default is `50`.

`sw2PC_tol`: the infeasibility measure threshold below which the predictor-corrector step is applied. The default is `sw2PC_tol = Inf`.

`use_corrprim`: a 0–1 flag indicating whether to correct for primal infeasibility.
The default is 0.

`printyes`: a 0–1 flag indicating whether to display the result of each iteration.
The default is `1`.

`scale_data`: a 0–1 flag indicating whether to scale the SDP data.
The default is `0`.

`schurfun`: a string containing the initial line of user supplied function file for computing the Schur complement matrix.
The default is `[ ]`.

`schurfun_parms`: a cell array containing the external parameters needed in user supplied Schur routine described in `schurfun`.
The default is `[ ]`.

`randnstate`: the initial seed used for the random vector used to initiate the Arnoldi iteration.
The default is `0`.

C Mex files used.

The computation of the Schur complement matrix M requires repeated computation of matrix products involving either matrices that are triangular or products that are known a priori to be Hermitian. We compute these matrix products in a C Mex routine generated from a C program `mexProd2.c` written to take advantage of the structures of the matrix products. A C Mex routine generated from the C program `mexProd3nz.c` computes certain elements of the products of three sparse matrices, while another, generated from the C program `mexschur.c`, computes the Schur complement efficiently. Likewise, computation of the inner product between two matrices is done in a C Mex routine generated from the C program `mextrace.c` written to take advantage of possible sparsity in either matrix. Another C Mex routine, generated from the C program `mexAsum.c`, computes the result of applying the adjoint of A to a vector. Finally, a C Mex routine that is used in our package is generated from the C program `mexaasen.c` written to compute the Aasen decomposition of a Hermitian matrix [1]. To summarize, here are the C programs used in our package:

```
mextrace.c    mexProd2.c    mexProd3nz.c    mexAsum.c

mexschur.c    mexaasen.c
```

In addition to the source codes of these routines, corresponding binary files for a number of platforms (including Solaris, Linux, Alpha, SGI, and Windows) are available from the internet sites mentioned in the introduction.

Cell array representation for problem data.

Our implementation SDPT3 exploits the block-diagonal structure of the given data, A_k and C. Suppose the matrices A_k and C are block-diagonal of the same structure. If the initial iterate (X^0, Z^0) is chosen to have the same block-diagonal structure, then this structure is preserved for all the subsequent iterates (X, Z). For reasons that will be explained later, if there are numerous small blocks each of dimension less than say 10, it is advisable to group them together as a single sparse block-diagonal matrix instead of considering them as individual blocks. Suppose now that each of the matrices A_k and C consists of L diagonal blocks of square matrices of dimensions $n_1, n_2, \ldots, n_L$. We can classify each of these blocks into one of the following three types:

1. a dense or sparse matrix of dimension greater than or equal to 10;
2. a sparse block-diagonal matrix consisting of numerous sub-blocks each of dimension less than 10; or
3. a diagonal matrix.

For each SDP problem, the block-diagonal structure of A_k and C is described by an $L \times 2$ cell array named `blk` where the content of each of its elements is given as follows. If the ith block of each A_k and C is a dense or sparse matrix of dimension greater than or equal to 10, then

$$\texttt{blk\{i,1\} = 'nondiag'} \quad \texttt{blk\{i,2\}} = \mathtt{n_i}$$
$$\texttt{A\{i,k\}, C\{i\}} = [\mathtt{n_i x n_i} \texttt{ double] or [}\mathtt{n_i x n_i}\texttt{ sparse].}$$

(It is possible for some A_k's to have a dense ith block and some to have a sparse ith block, and similarly the ith block of C can be either dense or sparse.) If the ith block of each A_k and C is a sparse matrix consisting of numerous small sub-blocks, say t of them, of dimensions $n_i^{(1)}, n_i^{(2)}, \ldots, n_i^{(t)}$ such that $\sum_{l=1}^{t} n_i^{(l)} = n_i$, then

$$\texttt{blk\{i,1\} = 'nondiag'} \quad \texttt{blk\{i,2\}} = [\mathtt{n_i^{(1)}}\ \mathtt{n_i^{(2)}}\ \cdots\ \mathtt{n_i^{(t)}}]$$
$$\texttt{A\{i,k\}, C\{i\}} = [\mathtt{n_i x n_i}\texttt{ sparse].}$$

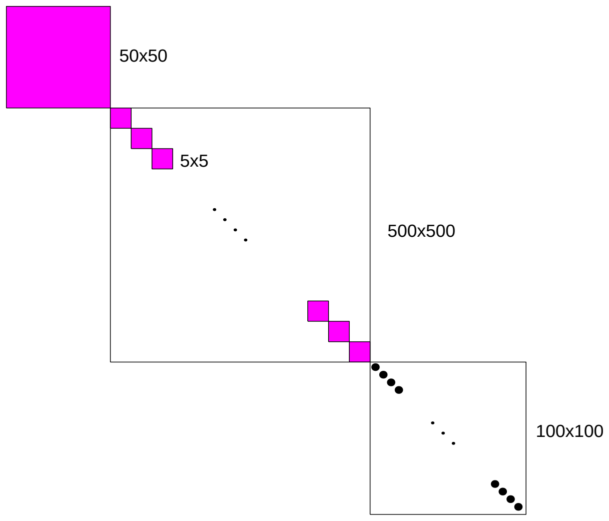

Figure 1: *An example of a block-diagonal matrix.*

If the ith block of each A_k and C is a diagonal matrix, then

```
blk{i,1} = 'diag'       blk{i,2} = n_i
A{i,k}, C{i} = [n_ix1 double].
```

As an example, suppose each of the A_k's and C has block structure as shown in Figure 1; then we have

```
blk{1,1} = 'nondiag'    blk{1,2} = 50
blk{2,1} = 'nondiag'    blk{2,2} = [5 5 ··· 5]
blk{3,1} = 'diag'       blk{3,2} = 100
```

and the matrices A_k and C are stored in cell arrays as

```
A{1,k}, C{1} = [50x50 double]
A{2,k}, C{2} = [500x500 sparse]
A{3,k}, C{3} = [100x1 double].
```

Notice that when the block is a diagonal matrix, only the diagonal elements are stored, and they are stored as a column vector.

Recall that when a block is a sparse block-diagonal matrix consisting of t sub-blocks of dimensions $n_i^{(1)}, n_i^{(2)}, \ldots, n_i^{(t)}$, we can actually view it as t individual blocks, in which case there will be t cell array elements associated with the t blocks rather than just one single cell array element originally associated with the sparse block-diagonal matrix. The reason for using the sparse matrix representation to handle the case when we have numerous small diagonal blocks is that it is less efficient for MATLAB to work with a large number of cell array elements compared to working with a single cell array element consisting of a large sparse block-diagonal matrix.

Technically, no problem will arise if one chooses to store the small blocks individually instead of grouping them together as a sparse block-diagonal matrix.

We should also mention the function file `ops.m` used in our package. The purpose of this file is to facilitate arithmetic operations on the contents of any two cell arrays with constituents that are matrices of corresponding dimensions.

For the usage of MATLAB cell arrays, we refer to [18].

Complex data.

Complex SDP data are allowed in our package. The user does not have to make any declaration even when the data is complex. Our codes will automatically detect whether this is the case.

Caveats.

The user should be aware that semidefinite programming is more complicated than linear programming. For example, it is possible that both primal and dual problems are feasible, but their optimal values are not equal. Also, either problem may be infeasible without there being a certificate of that fact (so-called weak infeasibility). In such cases, our software package is likely to terminate after some iterations with an indication of short step-length or lack of progress. Also, even if there is a certificate of infeasibility, our infeasible-interior-point methods may not find it. Our homogeneous self-dual methods may also fail to detect infeasibility, but they are practical variants of theoretical methods that are guaranteed to obtain certificates of infeasibility if such exist. In our very limited testing on strongly infeasible problems, most of our algorithms have been quite successful in detecting infeasibility.

6 Example files

To solve a given SDP, the user needs to express it in the standard form (1) and (2), and then write a function file, say `problem.m`, to compute the input data `blk,A,C,b,X0,y0,Z0` for the solvers `sdp.m` or `sdphlf.m`. This function file may take the form

```
[blk,A,C,b,X0,y0,Z0] = problem(input arguments).
```

Alternatively, one can provide the data in the input format described in [6], which is based on that of [12], and execute the function (based on a routine written by Brian Borchers)

```
[blk,A,C,b,X0,y0,Z0] = read_sdpa('filename').
```

The user can easily learn how to use this software package by reading the script file `demo.m`, which illustrates how the solvers `sdp.m` and `sdphlf.m` can be used to solve a few SDP examples. The next section shows how `sdp.m` and `sdphlf.m` can be used to solve random problems generated by `randsdp.m`, `graph.m`, and `maxcut.m`, and the resulting output, for several of our algorithms.

This software package also includes example files for the following classes of SDPs. In these files, unless otherwise stated, the input variables `feas` and `solve` are used as follows:

$$\texttt{feas} = \begin{cases} \texttt{0} & \text{corresponds to the initial iterate given in (43),} \\ \texttt{1} & \text{corresponds to a feasible initial iterate;} \end{cases}$$

$$\texttt{solve} = \begin{cases} \texttt{0} & \text{only gives the input data } \texttt{blk,A,C,b,X0,y0,Z0} \text{ for } \texttt{sdp.m} \text{ or } \texttt{sdphlf.m}, \\ \texttt{1} & \text{solves the given problem by an infeasible path-following algorithm,} \\ \texttt{2} & \text{solves the given problem by a homogeneous self-dual algorithm.} \end{cases}$$

If `solve` is positive, the output variable `objval` is the objective value of the associated optimization problem, and the output variables after `objval` give approximately optimal solutions to the original problem and its dual (or possibly indications of infeasibility).

Here are our examples.

(1) Random SDP: The associated M-file is `randsdp.m`, with initial line

```
[blk,A,C,b,X0,y0,Z0,objval,X,y,Z] = randsdp(de,sp,di,m,feas,solve),
```

where the input parameters describe a particular block diagonal structure for each A_k and C. Specifically, the vector `de` is a list of dimensions of dense blocks; the vector `sp` is a list of dimensions of (small) subblocks in a single sparse block; and the scalar `di` is the size of the diagonal block. The scalar `m` is the number of equality constraints.

There is an alternative function `randinfsdp.m` that generates primal or dual infeasible problems. The associated M-file has the initial line

```
[blk,A,C,b,X0,y0,Z0,objval,X,y,Z] = randinfsdp(de,sp,di,m,infeas,solve).
```

The input variables `de, sp, di,` and `solve` all have the same meaning as in `randsdp.m`, but the variable `infeas` is used as follows:

$$\texttt{infeas} = \begin{cases} \texttt{1} & \text{if want primal infeasible pair of problems,} \\ \texttt{2} & \text{if want dual infeasible pair of problems.} \end{cases}$$

(2) Norm minimization problem [30]:

$$\min_{x \in \mathbb{R}^m} \|B_0 + \sum_{k=1}^{m} x_k B_k\|,$$

where the B_k, $k = 0, \ldots, m$, are $p \times q$ matrices (possibly complex, in which case x ranges over $\mathbb{C}^m$) and the norm is the matrix 2-norm. The associated M-file is `norm_min.m`, with initial line

```
[blk,A,C,b,X0,y0,Z0,objval,x] = norm_min(B,feas,solve),
```

where `B` is a cell array with `B{k + 1}` = `B_k`, `k = 0,...,m`.

(3) Chebyshev approximation problem for a matrix [28]:

$$\min_p \|p(B)\|,$$

where the minimization is over the class of monic polynomials of degree m, B is a square matrix (possibly complex) and the norm is the matrix 2-norm. The associated M-file is `chebymat.m`, with initial line

```
[blk,A,C,b,X0,y0,Z0,objval,p] = chebymat(B,m,feas,solve).
```

See also `igmres.m`, which solves an analogous problem with p normalized such that $p(0) = 1$.

(4) Max-Cut problem [14]:

$$\begin{array}{rl} \min_X & L \bullet X \\ \text{s.t.} & \operatorname{diag}(X) = e/4, \quad X \succeq 0, \end{array}$$

where $L = B - \operatorname{Diag}(Be)$, e is the vector of all ones and B is the weighted adjacency matrix of a graph [14]. The associated M-file is `maxcut.m`, with initial line

```
[blk,A,C,b,X0,y0,Z0,objval,X] = maxcut(B,feas,solve).
```

See also `graph.m`, from which the user can generate a weighted adjacency matrix B of a random graph.

(5) ETP (Educational testing problem) [30]:

$$\begin{array}{rl} \max_{d \in I\!R^N} & e^T d \\ \text{s.t.} & B - \operatorname{Diag}(d) \succeq 0, \quad d \geq 0, \end{array}$$

where B is a real $N \times N$ positive definite matrix and e is again the vector of all ones. The associated M-file is `etp.m`, with initial line

```
[blk,A,C,b,X0,y0,Z0,objval,d] = etp(B,feas,solve).
```

(6) Lovász θ function for a graph [2]:

$$\begin{array}{rll} & \min_X \; C \bullet X & \\ \text{s.t.} & A_1 \bullet X = 1, & \\ & A_k \bullet X = 0, & k = 2, \ldots, m, \\ & X \succeq 0, & \end{array}$$

where C is the matrix of all minus ones, $A_1 = I$, and $A_k = e_i e_j^T + e_j e_i^T$, where the $(k-1)$st edge of the given graph (with $m-1$ edges) is from vertex i to vertex j. Here e_i denotes the ith unit vector. The associated M-file is `theta.m`, with initial line

```
[blk,A,C,b,X0,y0,Z0,objval,X] = theta(B,feas,solve),
```

where `B` is the adjacency matrix of the graph.

(7) Logarithmic Chebyshev approximation problem [30]**:**

$$\min_{x\in\mathbb{R}^m} \max_{1\le k\le N} |\log(b_k^T x) - \log(f_k)|,$$

where $B = [b_1\ b_2\ \cdots\ b_N]^T$ is a real $N \times m$ matrix and f is a real N-vector. The associated M-file is `logcheby.m`, with initial line

```
[blk,A,C,b,X0,y0,Z0,objval,x] = logcheby(B,f,feas,solve).
```

(8) Chebyshev approximation problem in the complex plane [28]**:**

$$\min_{p} \max_{1\le k\le N} |p(d_k)|,$$

where the minimization is over the class of monic polynomials of degree m and $\{d_1, \dots, d_N\}$ is a given set of points in the complex plane. The associated M-file is `chebyinf.m`, with initial line

```
[blk,A,C,b,X0,y0,Z0,objval,p] = chebyinf(d,m,feas,solve),
```

where $\mathtt{d} = [\mathtt{d}_1\ \ \mathtt{d}_2\ \ \dots\ \ \mathtt{d}_\mathtt{N}]$.

See also `cheby0.m`, which solves an analogous problem with p normalized such that $p(0) = 1$.

(9) Control and system problem [30]**:**

$$\begin{array}{rl} & \max_{t,P}\ t \\ \text{s.t.} & -B_k^T P - PB_k \succeq 0, \quad k = 1, \dots, L \\ & P \succeq tI, \quad I \succeq P, \quad P = P^T, \end{array}$$

where B_k, $k = 1, \dots, L$, are square real matrices of the same dimension. The associated M-file is `control.m`, with initial line

```
[blk,A,C,b,X0,y0,Z0,objval,P] = control(B,solve),
```

where `B` is a cell array with $\mathtt{B}\{\mathtt{k}\} = \mathtt{B}_\mathtt{k}$, `k = 1,...,L`.

7 Sample Runs

Assuming that the current directory is `SDPT3-2.1`, we will now generate some sample runs to illustrate how our package might be used.

```
>> randn('seed',0) % reset random generator to its initial seed.
>> rand('seed',0)  %
>> startup         % set up default parameters in the OPTIONS structure,
>>                 % set paths
>>
>> %%  random SDP  %%
>>
>> de=[20]; sp=[]; di=[]; % one 20X20 dense block, no sparse/diag blocks
>> m=20;                  % 20 equality constraints
>> feas=1;                % feasible initial iterate
>> solve=0;               % do not solve the problem, just generate data.
>> [blk,A,C,b,X0,y0,Z0] = randsdp(de,sp,di,m,feas,solve);
>>
>> OPTIONS.gaptol=1e-12;  % use a non-default relative accuracy tolerance
>> OPTIONS.vers=1;        % use the AHO direction
>>                        % solve using IPC
>> [obj,X,y,Z] = sdp(blk,A,C,b,X0,y0,Z0,OPTIONS);

*******************************************************************
         Infeasible path-following algorithms
*******************************************************************
 version  predcorr  gam  expon  use_corrprim  sw2PC_tol  scale_data
   1        1      0.000   3         0          Inf           0

it  pstep dstep p_infeas d_infeas  gap        obj       sigma
-------------------------------------------------------------------
 0  0.000 0.000 1.8e-16 8.9e-17  8.2e+03  2.545530e+03
 1  0.002 0.029 5.8e-16 8.7e-17  8.2e+03  2.569574e+03  0.998
 .    .     .      .       .        .         .          .
 .    .     .      .       .        .         .          .
12  0.988 0.988 1.3e-13 1.5e-16  1.5e-09 -1.058223e+03  0.000
13  0.967 0.974 8.8e-14 1.4e-16  8.7e-11 -1.058223e+03  0.028

 Stop: max(relative gap, infeasibilities) < 1.00e-12
---------------------------------------------------
 number of iterations   = 13
 gap                    = 8.73e-11
 relative gap           = 8.25e-14
 infeasibilities        = 8.76e-14
 Total CPU time         = 4.9
 CPU time per iteration = 0.4
 termination code       =  0
---------------------------------------------------
  Percentage of CPU time spent in various parts
---------------------------------------------------
  chol    pred    steplen    corr    steplen    misc
  0.7    71.9   4.5       13.7   5.5       3.8
----------------------------------------------------
```

An explanation for the notations used in the iteration output above is in order:

- `it`: the iteration number.
- `pstep, dstep`: denote the step-lengths α and β, respectively.
- `p_infeas, d_infeas`: denote the relative primal infeasibility $\|r_p\|/\max(1,\|b\|)$ and dual infeasibility $\|R_d\|_F/\max(1,\|C\|_F)$, respectively.
- `gap`: the duality gap $X \bullet Z$.
- `obj`: the mean objective value $(C \bullet X + b^T y)/2$.
- `sigma`: the value used for the centering parameter σ.

To give the reader an idea of the amount of CPU time spent in various steps of our algorithms, we give the breakdowns of the CPU time spent in algorithm IPC, and this is reported in the last line of the summary table above.

```
>> randn('seed',0);  rand('seed',0);
>>          % next, generate new data with a different block structure
>> feas=0;  % and use the (infeasible) initial iterate given in (42)
>> [blk,A,C,b,X0,y0,Z0] = randsdp([20 15],[4 3 3],5,30,feas,solve);
>>
>> OPTIONS.vers=4;  % use the GT direction
>>                  % solve using HPC
>> [obj,X,y,Z,tau,kap] = sdphlf(blk,A,C,b,X0,y0,Z0,1,1,OPTIONS);

*******************************************************************
         Homogeneous self-dual algorithms
*******************************************************************
 version  predcorr  gam  expon  use_corrprim  sw2PC_tol  scale_data
   4         1     0.000    1         0          Inf          0

it pstep dstep p_infeas d_infeas  gap         obj      sigma
-------------------------------------------------------------------
 0  0.000 0.000 1.8e+01 9.1e-01 5.4e+05  1.539414e+05
 1  0.865 1.000 7.6e+00 3.1e-01 1.9e+05  6.344746e+04  0.376
 2  1.000 1.000 2.3e+00 9.5e-02 5.6e+04  1.925616e+04  0.321
 .    .     .     .       .       .         .           .
 .    .     .     .       .       .         .           .
14  1.000 1.000 2.0e-11 4.2e-15 3.0e-09  3.067248e+02  0.019
15  0.998 1.000 9.1e-12 1.8e-16 4.6e-11  3.067248e+02  0.014

 Stop: relative gap < 0.2*infeasibility
------------------------------------
 number of iterations   = 15
 gap                    = 4.61e-11
 relative gap           = 1.50e-13
 infeasibilities        = 9.08e-12
 Total CPU time         = 20.6
 CPU time per iteration = 1.4
 termination code       =  0
---------------------------------------------------
  Percentage of CPU time spent in various parts
---------------------------------------------------
  chol   pred   steplen   corr   steplen   misc
  0.5    74.2   4.1       14.4   4.1       2.7
---------------------------------------------------

>> %%%%% MAXCUT PROBLEM %%%%%
>>
>> randn('seed',0);  rand('seed',0);
>> B = graph(50,0.3); % generate an adjacency matrix of a 50 node graph
>>                    % where each edge is present with probability 0.3
>> feas=1;    % use a feasible initial iterate;
>> solve=1;   % generate data, then solve the problem using IPC
>>            % with default parameters set up for the OPTIONS structure
>>            % in parameters.m but with the NT direction
>>            % next solve the maxcut problem defined on the given graph
>>
>> [blk,A,C,b,X0,y0,Z0,objval,X] = maxcut(B,feas,solve);

*******************************************************************
         Infeasible path-following algorithms
*******************************************************************
 version  predcorr  gam  expon  use_corrprim  sw2PC_tol  scale_data
   2        1      0.000   1         0          Inf          0

it  pstep dstep p_infeas d_infeas  gap        obj       sigma
-------------------------------------------------------------------
 0  0.000 0.000 0.0e+00 0.0e+00  2.2e+02 -2.952000e+02
 1  1.000 1.000 1.1e-15 5.2e-17  7.1e+01 -2.383341e+02  0.321
 2  0.699 1.000 3.9e-16 7.3e-17  4.6e+01 -2.542980e+02  0.475
 .    .     .      .       .        .           .           .
 .    .     .      .       .        .           .           .
 9  1.000 1.000 1.9e-13 5.4e-17  3.8e-06 -2.527228e+02  0.097
10  1.000 1.000 2.8e-13 9.3e-17  1.8e-07 -2.527228e+02  0.048

 Stop: max(relative gap, infeasibilities) < 1.00e-08
---------------------------------------------------
 number of iterations   = 10
 gap                    = 1.82e-07
 relative gap           = 7.21e-10
 infeasibilities        = 2.78e-13
 Total CPU time         = 2.0
 CPU time per iteration = 0.2
 termination code       =  0
---------------------------------------------------
  Percentage of CPU time spent in various parts
---------------------------------------------------
  chol   pred   steplen   corr   steplen   misc
  2.5    34.2   16.7       21.7   17.5      7.5
```

----

8 Specialized routines for computing the Schur complement matrix

For SDP problems where the matrices $A_1, \cdots, A_m$ are all low rank matrices except possibly a few of them, we can speed up the computation of the Schur complement matrix M by exploiting this low rank structure. As an example, we will discuss how this is done for the case of symmetric rank one matrices in computing the NT direction. Suppose

$$A_i \;=\; a_i a_i^T, \qquad i = 1, \cdots, m.$$

Given that $M_{ij} = A_i \bullet W A_j W$ for the case of the NT direction, where W is the NT scaling matrix, we have

$$\begin{aligned} M_{ij} &= \mathrm{Tr}\left(a_i a_i^T W a_j a_j^T W\right) \\ &= \mathrm{Tr}\left((a_i^T W a_j)(a_j^T W a_i)\right) \\ &= (a_i^T W a_j)^2. \end{aligned}$$

Thus computing M for the NT direction in this case requires at most $2mn^2 + m^2 n$ flops, to leading orders. Similar simplifications can be done for the HKM direction.

For the GT direction, exploiting low rank structures in the SDP data is still possible but is more involved compared to the HKM and NT directions. The reason for such a difference is that the matrices $\boldsymbol{D}_1, \boldsymbol{D}_2$ in (16) are respectively the matrix of ones and the identity matrix for the case of the HKM and NT directions, whereas $\boldsymbol{D}_1$ is a dense matrix and $\boldsymbol{D}_2 = I$ in the case of the GT direction. More precisely, for the case where all the matrices $A_1, \cdots, A_m$ are all symmetric rank one matrices, we have for the GT direction,

$$\begin{aligned} M_{ij} &= (\tilde{a}_i \circ \tilde{a}_j)^T \boldsymbol{D}_1 (\tilde{a}_i \circ \tilde{a}_j) \\ &= \mathrm{Tr}\left(\left[(\tilde{a}_i \tilde{a}_i^T) \circ \boldsymbol{D}_1\right] (\tilde{a}_j \tilde{a}_j^T)\right), \end{aligned}$$

where $\tilde{a}_i = R a_i$, $i = 1, \cdots, m$, for some matrix R.

For the AHO direction, where $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ are both dense matrices, exploiting low rank structures in the data becomes even more complicated. We shall not elaborate further on this issue but leave it for future work.

In our package, we include the following specialized routines for computing the Schur complement matrix for the HKM and NT directions for a few classes of the SDPLIB problems [6], namely, the `mcp`, `gpp`, `eqG`, and `theta` problems. The specialized function files are as follows:

`mcpHKMsch.m`	`gppHKMsch.m`	`eqGHKMsch.m`	`thetaHKMsch.m`
`mcpNTsch.m`	`gppNTsch.m`	`eqGNTsch.m`	`thetaNTsch.m`.

These function files have a form similar to `thetaHKMsch.m`, whose initial line is

```
[schur,hRd] = thetaHKMsch(X,Zinv,Rd,schurfun_parms).
```

The specialized routine can be used to replace the computation of the Schur complement matrix inside the main solver `sdp.m` or `sdphlf.m` by passing the name of the specialized function file into the main solver through the structure array `OPTIONS`, specifically, by setting `OPTIONS.schurfun` to be the string containing the initial line of the function file, for example,

```
OPTIONS.schurfun = 'thetaHKMsch(X,Zinv,Rd,schurfun_parms)'.
```

The input argument `schurfun_parms` can be omitted if there are no extra input parameter variables needed besides `X` and `Zinv`. However, if in addition to `X` and `Zinv`, extra parameter variables are needed in the specialized routine, then these parameter variables can be passed into the specialized routine while executing inside `sdp.m` or `sdphlf.m` through the structure array `OPTIONS` by assigning

```
OPTIONS.schurfun_parms
```

to be a cell array containing all the required extra parameter variables.

Assuming that the current directory is `SDPT3-2.1` and its subdirectory `Specialschur` contains the specialized routines, we will now illustrate how the specialized routines can be used in `sdp.m`.

```
>> randn('seed',0);  rand('seed',0);
>> startup;    % add appropriate path to MATLAB path.
>> [blk,A,C,b] = read_sdpa('./sdplib/theta3.dat-s'); % read in SDP data from
                                                     % subdirectory sdplib.
>> [X0,y0,Z0] = infeaspt(blk,A,C,b);                 % get starting point.
>> OPTIONS.vers = 2;
>> OPTIONS.schurfun = 'thetaHKMsch(X,Zinv,Rd,schurfun_parms)';
>> OPTIONS.schurfun_parms = listA; % assume that the extra parameter variable
                                   % listA is already computed.
   % It can be computed via:
   % spdensity = 1;
   % [dummy,listA,permA] = nonzerolist(blk,A,spdensity);
   % listA(permA) = listA;

>> [obj,X,y,Z] = sdp(blk,A,C,b,X0,y0,Z0,OPTIONS);

*******************************************************************
         Infeasible path-following algorithms

user supplied Schur routine: thetaHKMsch(X,Zinv,Rd,schurfun_parms)
*******************************************************************
 version  predcorr  gam  expon  use_corrprim  sw2PC_tol  scale_data
   2         1      0.000   1          0          Inf           0
```

```
it  pstep dstep p_infeas d_infeas  gap        obj       sigma
-------------------------------------------------------------------
 0  0.000 0.000 1.3e+04 1.5e+00  1.6e+05 -6.590097e+03
 1  0.130 0.200 1.1e+04 1.2e+00  1.4e+05 -9.645481e+04  0.417
 .    .     .      .       .        .          .          .
 .    .     .      .       .        .          .          .
13  1.000 1.000 4.8e-12 7.5e-17  1.6e-06 -4.216698e+01  0.100
14  1.000 1.000 8.5e-12 7.6e-17  2.4e-08 -4.216698e+01  0.015

 Stop: max(relative gap, infeasibilities) < 1.00e-08
----------------------------------------------------
 number of iterations   = 14
 gap                    = 2.44e-08
 relative gap           = 5.79e-10
 infeasibilities        = 8.52e-12
 Total CPU time         = 244.8
 CPU time per iteration = 17.5
 termination code       =  0
----------------------------------------------------
  Percentage of CPU time spent in various parts
----------------------------------------------------
  chol   pred   steplen   corr   steplen   misc
  0.2    89.2   1.7       6.8   1.7       0.4
----------------------------------------------------
```

9 Numerical results

The tables below show the performance of the algorithms discussed in Section 2 and 3 on the first eight SDP examples described in Section 6. The result for each example is based on ten random instances with normally distributed data generated via the MATLAB command `randn`. The initial iterate for each problem is infeasible, generated from `infeaspt.m` with the default option. Note that the same set of random instances is used throughout for each example.

In Tables 2 and 3, we use the default values (given in Section 5) for the parameters used in the algorithms, except for `OPTIONS.gaptol` and `OPTIONS.scale_data`, which are set to `1e-13` and `1`, respectively.

In our experiments, let us call an SDP instance successfully solved by Algorithm IPC if the algorithm manages to reduce the relative duality gap $X \bullet Z / \max(1, |C \bullet X|)$ to less than 10^{-6} while at the same time the infeasibility measure ϕ is less than the relative duality gap. For Algorithm HPC, we call an SDP instance successfully solved if the relative duality gap is less than 10^{-6} while the infeasibility measure ϕ is at most 5 times more than the relative duality gap. We do not terminate the algorithms when this measure of success is attained.

All of the SDP instances (a total of 640) considered in our experiments were successfully solved, except for only three ETP instances and one Logarithmic Chebyshev instance where Algorithm HPC using the AHO direction failed. This indicates that our algorithms are probably quite robust.

The results in Tables 2 and 3 show that the behavior of Algorithms IPC and HPC are quite similar in terms of efficiency (as measured by the number of iterations) and accuracy on all the the four search directions we implemented. Note that we give the number of iterations and the CPU time required to reduce the duality gap by a factor of 10^{10} compared to its original value (the relative gap may then be still too large to conclude "success"), and also the minimum relative gap achieved by each method. For both algorithms, the AHO and GT directions are more efficient and more accurate than the HKM and NT directions, with the former and latter pairs having similar behavior in terms of efficiency and accuracy. Efficiency can alternatively be measured by the total CPU time required. The performances of Algorithms IPC and HPC are also quite similar in terms of the CPU time taken to reduce the duality gap by a prescribed factor on all the four directions. But in this case, the NT and HKM directions are the fastest, followed by the GT direction which is about 20% to 30% slower, and the AHO direction is the slowest — it is usually at least about 60% slower.

Algorithm IPC		Ave. no. of iterations to reduce the duality gap by 10^{10}				Ave. CPU time (sec.) to reduce the duality gap by 10^{10}				Accuracy mean($\lvert\log_{10}(X \bullet Z)\rvert$)			
		AHO	GT	HKM	NT	AHO	GT	HKM	NT	AHO	GT	HKM	NT
random SDP	$n = 50$ $m = 50$	10.6	11.2	12.7	11.7	15.8	12.1	11.1	10.7	7.4	6.4	6.1	5.8
Norm min. problem	$n = 100$ $m = 26$	9.1	9.4	10.8	11.0	39.4	29.3	23.4	26.5	11.9	12.4	9.6	9.1
Cheby. approx. of a real matrix	$n = 100$ $m = 26$	8.8	9.3	10.8	11.5	37.9	28.4	24.8	27.5	13.7	13.6	10.8	10.5
Maxcut	$n = 50$ $m = 50$	9.9	10.5	11.5	11.7	11.3	7.9	5.8	6.2	10.9	9.8	9.0	8.7
ETP	$n = 100$ $m = 50$	17.1	17.5	20.3	19.9	25.6	17.3	14.2	14.4	8.8	8.8	7.1	7.2
Lovász θ function	$n = 30$ $m \approx 220$	11.7	11.7	12.1	12.1	53.3	29.9	23.8	21.8	11.6	10.9	10.4	10.5
Log. Cheby. problem	$n = 300$ $m = 51$	12.6	13.0	13.7	13.7	24.6	21.2	15.2	18.2	9.6	9.7	9.7	9.8
Cheby. approx. on $\mathbb{C}$	$n = 200$ $m = 41$	9.9	10.2	11.1	11.3	15.7	14.4	11.0	13.6	12.9	13.0	10.9	10.9

Table 2: *Computational results on different classes of SDP for Algorithm IPC. Ten random instances are considered for each class. The computations were done on a DEC AlphaStation/500 (333MHz). The number $X \bullet Z$ above is the smallest number such that relative duality gap $X \bullet Z/(1 + |C \bullet X|)$ is less than 10^{-6} and the infeasibility measure ϕ is less than the relative duality gap.*

Algorithm HPC		Ave. no. of iterations to reduce the duality gap by 10^{10}				Ave. CPU time (sec.) to reduce the duality gap by 10^{10}				Accuracy mean($\lvert\log_{10}(X \bullet Z)\rvert$)			
		AHO	GT	HKM	NT	AHO	GT	HKM	NT	AHO	GT	HKM	NT
random SDP	$n = 50$ $m = 50$	10.4	10.9	11.4	11.0	15.7	11.7	9.7	9.7	8.8	8.1	6.4	6.0
Norm min. problem	$n = 100$ $m = 26$	10.9	10.2	11.9	11.5	48.7	32.3	25.8	27.7	11.1	11.5	9.6	9.0
Cheby. approx. of a real matrix	$n = 100$ $m = 26$	10.1	10.1	11.8	11.1	44.4	31.6	26.9	26.4	13.7	12.8	11.1	10.5
Maxcut	$n = 50$ $m = 50$	9.9	9.7	11.1	10.6	11.8	7.6	5.8	5.8	10.8	10.1	9.2	8.6
ETP	$n = 100$ $m = 50$	14.3*	15.3	17.1	16.6	21.8*	15.5	12.1	12.1	9.4*	9.5	7.2	6.9
Lovász θ function	$n = 30$ $m \approx 220$	11.5	11.7	12.9	12.8	44.6	29.8	24.8	22.4	12.2	11.5	11.0	10.5
Log. Cheby. problem	$n = 300$ $m = 51$	15.0*	12.5	13.2	13.2	31.7*	21.3	15.4	18.4	12.2*	12.9	12.4	12.4
Cheby. approx. on $\mathbb{C}$	$n = 200$ $m = 41$	9.8	9.6	10.1	10.0	16.5	14.5	10.1	12.5	13.5	13.3	11.5	11.5

Table 3: *Same as Table 2, but for the homogenous predictor-corrector algorithm, Algorithm HPC. The duality gap $X \bullet Z$ above is the smallest number such that the relative duality gap $X \bullet Z/(1+|C \bullet X|)$ is less than 10^{-6} and the infeasibility measure ϕ is at most 5 times more than the relative duality gap.*
* *Three of the ETP instances fail because the infeasibility measure ϕ is consistently 5 times more than the relative duality gap $X \bullet Z/(1+|C \bullet X|)$ when the relative duality gap is less than 10^{-6}. One of the Log. Cheby. instances fails due to step lengths going below 10^{-6}. The numbers reported here are based on the successful instances.*

Finally, we report the performance of our algorithms on a collection of SDPLIB problems [6] in Table 4. We use the default values for the parameters used in the algorithms. The figures given for each run are number of iterations, precision ($-k$ indicates relative infeasibility and relative duality gap below 10^{-k}), and CPU time in seconds.

```
The following results for Algorithm IPC are obtained from a Pentium II PC,
with 400MHz CPU and 256M RAM, running Linux.
----------------------------------------------------------------------
              HKM           NT             GT              AHO
----------------------------------------------------------------------
   arch8    19 -9  104    23 -7  140    20 -8  597      20 -8  986
control7    23 -6  489    23 -6  462    22 -5  821      22 -6  1301
control10   25 -6  2776   24 -6  2542   24 -6  4543     23 -6  7183
control11   24 -6  4265   24 -6  4022   24 -6  7118     23 -6  11686
gpp250.4    15 -8  128    15 -9  170    17 -9  2030     16 -9  5814
gpp500.4    16 -8  1379   16 -8  1906   17 -9  36979    14 -5  108356
mcp250.1    13 -9  47     14 -9  93     13 -9  1446     12 -9  3237
mcp500.1    14 -8  428    16 -9  1052   16 -9  33397    13 -9  72025
    qap9    15 -8  105    15 -8  107    15 -8  435      16 -7  811
   qap10    14 -8  228    14 -7  230    14 -8  891      14 -8  1660
    ss30    19 -9  289    22 -7  435    18 -8  3210     19 -9  6061
  theta3    14 -9  279    14 -10 290    13 -8  1950     12 -9  3380
  theta4    15 -8  1421   14 -9  1354   15 -9  10625    13 -9  15962
  theta5    15 -8  5024   14 -9  4750   t               t
  truss8    22 -6  573    21 -6  577    24 -7  985      27 -7  1567
equalG11    17 -8  6050   17 -9  8391   t               t
  maxG11    14 -8  1717   14 -9  3815   15 -9  197092   t
  maxG51    17 -8  4642   16 -9  9738   t               t
   qpG11    14 -8  2457   m             m               m
----------------------------------------------------------------------
m: available memory exceeded.
t: problem was terminated because it was taking too long to finish.
```

Acknowledgments

The authors thank Christoph Helmberg for many constructive suggestions. The authors also thank Brain Borchers for making his collection of SDP test problems [6] available to the public, and to Hans Mittelmann [20] for testing the previous versions of SDPT3 on the SDPLIB problems which shed light into some of the weaknesses of the previous versions.

References

[1] J. O. Aasen, *On the reduction of a symmetric matrix to tridiagonal form*, BIT 11 (1971), pp. 233-242.

[2] F. Alizadeh, *Interior point methods in semidefinite programming with applications to combinatorial optimization*, SIAM J. Optimization, 5 (1995), pp. 13–51.

[3] F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton, *Primal-dual interior-point methods for semidefinite programming: convergence results, stability and numerical results*, SIAM J. Optimization, 8 (1998), pp. 746–768.

[4] F. Alizadeh, J.-P A. Haeberly, M. V. Nayakkankuppam, M.L. Overton, and S. Schmieta, *SDPPACK user's guide*, Technical Report, Computer Science Department, NYU, New York, June 1997.

[5] S. J. Benson, Y. Ye, and X. Zhang, *Solving large-scale sparse semidefinite programs for combinatorial optimization*, working paper, Computational Optimization Laboratory, University of Iowa, May 1998.

[6] B. Borchers, *SDPLIB, A Library of Semidefinite Programming Test Problems*, available from `http://www.nmt.edu/~borchers/sdplib.html`.

[7] B. Borchers, *CSDP, a C library for semidefinite programming*, Technical report, New Mexico Tech., June 1998. Available at `http://www.nmt.edu/~borchers/csdp.html`.

[8] N. Brixius, F.A. Potra, and R. Sheng, *Solving semidefinite programming in Mathematica*, Reports on Computational Mathematics, No 97/1996, Department of Mathematics, University of Iowa, October, 1996. Available at `http://www.cs.uiowa.edu/~brixius/sdp.html`.

[9] N. Brixius, F. A. Potra, and R. Sheng, *SDPHA: A Matlab implementation of homogeneous interior-point algorithms for semidefinite programming*, available from `http://www.cs.uiowa.edu/~brixius/SDPHA/`, May 1998.

[10] S. Burer and R.D.C Monteiro, *An efficient algorithm for solving the MAXCUT relaxation SDP problem*, working paper, School of ISyE, Georgia Tech, USA, December 1998.

[11] K. Fujisawa, M. Kojima, and K. Nakata, *Exploiting sparsity in primal-dual interior-point method for semidefinite programming*, Mathematical Programming, 79 (1997), pp. 235–253.

[12] K. Fujisawa, M. Kojima, and K. Nakata, *SDPA (semidefinite programming algorithm) user's manual — version 4.10*, Research Report, Department of Mathematical and Computing Science, Tokyo Institute of Technology, Tokyo. Available via anonymous ftp at `ftp.is.titech.ac.jp` in `pub/OpRes/software/SDPA`.

[13] C. Helmberg and F. Rendl, *A spectral bundle method for semidefinite programming*, manuscript, Konrad-Zuse-Zentrum fuer Informationstechnik Berlin, Takustrasse 7, D-14195, Berlin, Germany, 1997.

[14] C. Helmberg, F. Rendl, R. Vanderbei and H. Wolkowicz, *An interior-point method for semidefinite programming*, SIAM Journal on Optimization, 6 (1996), pp. 342–361.

[15] E. de Klerk, C. Roos, and T. Terlaky, *Infeasible start, semidefinite programming algorithms via self-dual embeddings*, Report 97-10, TWI, Delft University of Technology, April 1997.

[16] M. Kojima, S. Shindoh, and S. Hara, *Interior-point methods for the monotone linear complementarity problem in symmetric matrices*, SIAM J. Optimization, 7 (1997), pp. 86–125.

[17] Z.-Q. Luo, J. F. Sturm, and S. Zhang, *Duality and self-duality for conic convex programming*, Technical Report 9620/A, Tinbergen Institute, Erasmus University, Rotterdam, 1996.

[18] The MathWorks, Inc., *Using MATLAB*, The MathWorks, Inc., Natick, MA, 1997.

[19] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM J. Optimization, 2 (1992), pp. 575–601.

[20] H. Mittelmann, *several SDP-codes on problems from SDPLIB*, available from `ftp://plato.la.asu.edu/pub/sdplib.txt`.

[21] R. D. C. Monteiro, *Primal–Dual Path-Following Algorithms for Semidefinite Programming*, SIAM J. Optimization, 7 (1997), pp. 663–678.

[22] F. A. Potra and R. Sheng, *On homogeneous interior-point algorithms for semidefinite programming*, Optimization Methods and Software 9: 161-184, 1998.

[23] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston.

[24] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press, Manchester, UK, 1992.

[25] J. F. Sturm, *SeDuMi 1.00: Self-dual-minimization. Matlab 5 toolbox for optimization over symmetric cones*, Communications Research Laboratory, McMaster University, Hamilton, Canada, April 1998.

[26] M. J. Todd, K. C. Toh, R. H. Tütüncü, *On the Nesterov-Todd direction in semidefinite programming*, SIAM J. Optimization, 8 (1998), pp. 769–796.

[27] K. C. Toh, *Search directions for primal-dual interior point methods in semidefinite programming*, Technical Report No. 722, Department of Mathematics, National University of Singapore, Singapore, July, 1997. Revised in March 1998.

[28] K. C. Toh and L. N. Trefethen, *The Chebyshev polynomials of a matrix*, SIAM J. Matrix Analysis and Applications, 20 (1998), pp. 400-419.

[29] K.C. Toh, *A note on the calculation of step-lengths in interior-point methods for semidefinite programming*, Research Report B-355, Department of Mathematical and Computing Science, Tokyo Institute of Technology, Tokyo, September, 1999.

[30] L. Vandenberghe and S. Boyd, *Semidefinite programming*, SIAM Review, 38 (1996), pp. 49–95.

[31] L. Vandenberghe and S. Boyd, *User's guide to SP: software for semidefinite programming*, Information Systems Laboratory, Stanford University, November 1994. Available via anonymous ftp at `isl.stanford.edu` in `pub/boyd/semidef_prog`. Beta version.

[32] X. Xu, P.-F. Hung, and Y. Ye, *A simplified homogeneous and self-dual linear programming algorithm and its implementation*, Annals of Operations Research, 62 (1996), pp. 151–171.

[33] Y. Ye, M. J. Todd, and S. Mizuno, *An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm*, Mathematics of Operations Research, 19 (1994), pp. 53–67.