SDPT3 — a MATLAB software package for semidefinite programming

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

March 5, 1998

Abstract

This software package is a MATLAB implementation of infeasible path-following algorithms for solving standard semidefinite programs (SDP). Mehrotra-type predictor-corrector variants are included. Analogous algorithms for the homogeneous formulation of the standard SDP 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 also included. Numerical results for these classes show that our algorithms are fairly efficient and robust on problems with dimensions of the order of a hundred.

[*]Department of Mathematics, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260 (mattohkc@math.nus.edu.sg).

[†]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 grant DMS-9505155 and ONR through grant N00014-96-1-0050.

[‡]Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213, USA (reha+@andrew.cmu.edu).

1 Introduction

This is a software package for solving the standard SDP:

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

where $A_k \in \mathcal{H}^n$, $C \in \mathcal{H}^n$ and $b \in \mathbb{R}^m$ 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. (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):

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

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

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

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

The purpose of this software package is to provide researchers in SDP with a collection of reasonably efficient 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 structure we exploit in this package is block-diagonal structure, where MATLAB cell arrays are used to handle dense and sparse blocks separately. For the purpose of evaluating the performance of algorithms proposed by other authors, we also include a few classes of SDP problems in this software package.

A special feature that distinguishes this SDP software from others (e.g., [3],[4],[5],[10]) is that complex data are allowed. But note that b and y must be real. Another special feature, though also shared by the software of [5], of our package is that the sparsity of matrices A_k is exploited in the computation of the Schur complement matrix required at each iteration of our SDP algorithms.

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

2 Infeasible-interior-point algorithms

2.1 The search direction

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

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

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

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

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.

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

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 [2] and [8] 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 CG or QMR. In our package, we assume that $m = \mathcal{O}(n)$ and (9) is solved by a direct method such as LU or Cholesky decomposition.

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, corresponding to $P = I$;

(2) the HKM direction, corresponding to $P = Z^{1/2}$;

(3) the NT direction, corresponding to $P = N^{-1}$, where N is the unique matrix such that $N^*ZN = N^{-1}XN^{-*} =: D$, and D is a diagonal matrix; and

(4) the GT direction, corresponding to $P = D^{1/2}\bar{G}^{-*}$, where the matrices 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 $D = \Sigma(\Psi\Phi)^{-1}$.

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 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 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

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

where $\odot$ denotes the Hadamard (elementwise) product and the matrices R, T, $\boldsymbol{D}_1$, and $\boldsymbol{D}_2$ depend only on X and Z. Thus the (i,j) element of M in (14) can be written equivalently as

$$M_{ij} \;=\; (R \circledast T(A_i)) \bullet (\boldsymbol{D}_1 \odot [(\boldsymbol{D}_2 \circledast I)(R \circledast T(A_j))]). \tag{15}$$

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 (15).

Computing M via different formulas, (14) or (15), will result in different computational complexities. Roughly speaking, if most of the matrices A_k are dense, then it is cheaper to use (15). 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 (15).

directions	upper bound on complexity by using (15)	upper bound on complexity by 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, except for the HKM direction, all the others require an eigenvalue decomposition of a symmetric matrix in the computation of M.*

The reader is referred to [2], [8], and [9] 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 (15) is given in [9]. The issue of exploiting the sparsity of the matrices A_k is discussed in full detail in [5] for the HKM and NT directions; whereas an analogous discussion for the AHO and GT directions can be found in [9].

In our implementation, 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 (15) 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 accurate estimate of the time required for subsequent iterations. Secondly, there are overheads incurred when variables are first loaded into MATLAB workspace.

We should mention that the issue of exploiting the sparsity of the matrices A_k is not fully resolved in our package. What we have used in our package is only a rough guide. Further improvements on this topic are left for future work.

2.3 The primal-dual path-following algorithm

The algorithmic framework of our primal-dual path-following algorithm is as follows.

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

$$\phi \;=\; \max\left(\frac{\|r_p\|}{1+\|b\|}\,,\,\frac{\|R_d\|_F}{1+\|C\|_F}\right). \tag{16}$$

  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

$$X^+ \;=\; X + \alpha\,\Delta X, \quad y^+ \;=\; y + \beta\,\Delta y, \quad Z^+ \;=\; Z + \beta\,\Delta Z, \tag{17}$$

  where

$$\alpha \;=\; \min\left(1, \frac{-\gamma}{\lambda_{\min}(X^{-1}\Delta X)}\right), \quad \beta \;=\; \min\left(1, \frac{-\gamma}{\lambda_{\min}(Z^{-1}\Delta Z)}\right). \tag{18}$$

  (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

$$\gamma^+ \;=\; 0.9 + 0.09\min(\alpha, \beta), \tag{19}$$

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

Remarks.

(a) The adaptive choice of the step-length parameter in (19) is used as the default in our implementation, but the user has the option of fixing the value of γ.

(b) 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 $\mathcal{D}$ need only to be formed once at the beginning of the algorithm.

(c) We only described termination when approximately optimal solutions are at hand. Nevertheless, our codes stop when other indications arise. For example, if the step makes little progress, or if the step-length taken in either primal or dual spaces becomes very small, we terminate. We also stop if we get an indication of infeasibility. For example, 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.

2.4 The Mehrotra-type predictor-corrector algorithm

The algorithmic framework of the Mehrotra-type predictor-corrector 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 (16). 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 (18) with $\Delta X, \Delta Z$ replaced by $\delta X, \delta Z$, respectively.
- Take σ to be

$$\sigma \;=\; \left[\frac{(X + \alpha_p\,\delta X) \bullet (Z + \beta_p\,\delta Z)}{X \bullet Z}\right]^e, \tag{20}$$

  where the exponent e is chosen as follows:

$$e \;=\; \begin{cases} \max[expon, 3\min(\alpha_p, \beta_p)^2] & \text{if } \mu > 10^{-6}, \\ expon & \text{if } \mu \le 10^{-6}. \end{cases} \tag{21}$$

- (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 (17), where α and β are computed as in (18) with γ chosen to be

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

- Update γ to γ^+ as in (19).

Remarks.

(a) The default choices of *expon* for the AHO, HKM, NT, and GT directions are $expon = 3, 1, 1, 2$, 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 compared to 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`.

(c) Once again, we also terminate if there is lack of progress in either the predictor or corrector steps, the primal or dual step-length is too small, or we get an indication of primal or dual infeasibility.

3 Homogeneous and self-dual algorithms

3.1 The search direction

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

$$\hat{\mathcal{A}} : \mathcal{H}^n \longrightarrow \mathbb{R}^{m+1},$$

$$\hat{\mathcal{A}}X = \begin{pmatrix} A_1 \bullet X \\ \vdots \\ A_m \bullet X \\ -C \bullet X \end{pmatrix}.$$

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

$$\hat{\mathcal{A}}^* : \mathbb{R}^{m+1} \longrightarrow \mathcal{H}^n,$$

$$\hat{\mathcal{A}}^* \begin{pmatrix} y \\ \tau \end{pmatrix} = \sum_{k=1}^m y_k A_k - \tau C.$$

Our homogeneous and self-dual linear feasibility model for SDP is based on those appearing in [11] for linear programming (LP). It has the following form:

$$\hat{\mathcal{A}}^* \begin{pmatrix} y \\ \tau \end{pmatrix} + Z = 0$$

$$\hat{\mathcal{A}}X - \begin{pmatrix} 0 & b \\ -b^T & 0 \end{pmatrix} \begin{pmatrix} y \\ \tau \end{pmatrix} - \begin{pmatrix} 0 \\ \kappa \end{pmatrix} = 0 \qquad (22)$$

$$XZ = 0, \quad \tau\kappa = 0,$$

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 Newton's method to a perturbation of equation (22), namely,

$$\hat{\mathcal{A}}^* \begin{pmatrix} y \\ \tau \end{pmatrix} + Z = \eta \hat{R}_d$$

$$\hat{\mathcal{A}}X - \begin{pmatrix} 0 & b \\ -b^T & 0 \end{pmatrix}\begin{pmatrix} y \\ \tau \end{pmatrix} - \begin{pmatrix} 0 \\ \kappa \end{pmatrix} = \eta\,\hat{r}_p \tag{23}$$

$$XZ = \sigma\mu I, \quad \tau\kappa = \sigma\mu,$$

where the right-hand side quantities are considered as fixed. Here σ and η are parameters, and $\hat{r}_p$ and $\hat{R}_d$ are defined in (26) and (25) below, respectively.

Just as in 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 (23):

$$\begin{aligned}
\hat{\mathcal{A}}^*\begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} + \Delta Z &= \eta\,\hat{R}_d \\
\hat{\mathcal{A}}\Delta X + \begin{pmatrix} 0 & -b \\ b^T & \kappa/\tau \end{pmatrix}\begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} &= \eta\,\hat{r}_p + \begin{pmatrix} 0 \\ r_c/\tau \end{pmatrix} \\
\mathcal{E}\Delta X + \mathcal{F}\Delta Z &= R_c \\
\tau\Delta\kappa + \kappa\Delta\tau &= r_c
\end{aligned} \tag{24}$$

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

$$\hat{R}_d := -\hat{\mathcal{A}}^*\begin{pmatrix} y \\ \tau \end{pmatrix} - Z, \tag{25}$$

$$\hat{r}_p := \begin{pmatrix} 0 \\ \kappa \end{pmatrix} + \begin{pmatrix} 0 & b \\ -b^T & 0 \end{pmatrix}\begin{pmatrix} y \\ \tau \end{pmatrix} - \hat{\mathcal{A}}X, \tag{26}$$

$$\mu := \frac{X \bullet Z + \tau\kappa}{n+1}, \tag{27}$$

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

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

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

$$\left[\hat{M} + \begin{pmatrix} 0 & -b \\ b^T & \kappa/\tau \end{pmatrix}\right]\begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} = \hat{h}, \tag{28}$$

where

$$\hat{M} = \hat{\mathcal{A}}\mathcal{E}^{-1}\mathcal{F}\hat{\mathcal{A}}^*, \tag{29}$$

$$\hat{h} = \eta\,\hat{r}_p + \begin{pmatrix} 0 \\ r_c/\tau \end{pmatrix} + \eta\,\hat{\mathcal{A}}\mathcal{E}^{-1}\mathcal{F}\hat{R}_d - \hat{\mathcal{A}}\mathcal{E}^{-1}R_c. \tag{30}$$

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

$$\begin{aligned} \Delta Z &= \eta \hat{R}_d - \hat{\mathcal{A}}^* \begin{pmatrix} \Delta y \\ \Delta \tau \end{pmatrix} \\ \Delta X &= \mathcal{E}^{-1} R_c - \mathcal{E}^{-1} \mathcal{F} \Delta Z \\ \Delta\kappa &= (r_c - \kappa\Delta\tau)/\tau. \end{aligned} \tag{31}$$

3.2 Primal and dual step-lengths

As in the case of infeasible path-following algorithms, taking 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 $\hat{\alpha}$ 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 $0 < \gamma < 1$).

Let

$$\tau_p \;=\; \tau + \alpha\Delta\tau, \qquad \tau_d \;=\; \tau + \beta\Delta\tau. \tag{32}$$

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

$$\begin{gathered} \tau^+ \;=\; \min(\tau_p,\, \tau_d), \qquad \kappa^+ \;=\; \begin{cases} \kappa + \alpha\,\Delta\kappa, & \text{if } \tau^+ = \tau_p \\ \kappa + \beta\,\Delta\kappa, & \text{if } \tau^+ = \tau_d. \end{cases} \\ X^+ \;=\; \tfrac{\tau^+}{\tau_p}(X + \alpha\,\Delta X), \quad y^+ \;=\; \tfrac{\tau^+}{\tau_d}(y + \beta\,\Delta y), \quad Z^+ \;=\; \tfrac{\tau^+}{\tau_d}(Z + \beta\,\Delta Z). \end{gathered} \tag{33}$$

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

$$r_p^+(\alpha) \;=\; b - \mathcal{A}(X^+/\tau^+), \qquad R_d^+(\beta) \;=\; C - Z^+/\tau^+ - \mathcal{A}^*(y^+/\tau^+), \tag{34}$$

satisfy the relation

$$r_p^+(\alpha) \;=\; \frac{1 - \alpha\eta}{1 + \alpha\Delta\tau/\tau}\, r_p, \qquad R_d^+(\beta) \;=\; \frac{1 - \beta\eta}{1 + \beta\Delta\tau/\tau}\, R_d, \tag{35}$$

where

$$r_p = b - \mathcal{A}(X/\tau), \qquad R_d = C - Z/\tau - \mathcal{A}^*(y/\tau). \tag{36}$$

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. 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

$$\frac{X^+ \bullet Z^+ + \tau^+\kappa^+}{(\tau^+)^2} \leq \frac{X \bullet Z + \tau\kappa}{\tau^2}, \tag{37}$$

when we substitute $\alpha = \hat{\alpha}$ and $\beta = \hat{\beta}$ into (32) and (33).

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 (37) 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 [11] 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 $\gamma^0 = 0.9$, and $\sigma^0 = 0.1$, $\eta^0 = 0.9$.

For $k = 0, 1, \ldots$

(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

$$\phi \;=\; \max\left(\frac{\|\tau b - \mathcal{A}X\|}{\tau(1+\|b\|)}, \frac{\|\tau C - Z - \mathcal{A}^* y\|_F}{\tau(1+\|C\|_F)}\right). \tag{38}$$

  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)

$$\mu/\mu_0 \;<\; 10^{-8}, \qquad \frac{\tau/\kappa}{\tau_0/\kappa_0} \;<\; 10^{-8}.$$

  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 (28)–(31) using $\eta = 1 - \sigma$.

- Let

$$\begin{aligned} \hat{\alpha} &= \min\left(1, \frac{-\gamma}{\lambda_{\min}(X^{-1}\Delta X)}, \frac{-\gamma}{\tau^{-1}\Delta\tau}, \frac{-\gamma}{\kappa^{-1}\Delta\kappa}\right), \\ \hat{\beta} &= \min\left(1, \frac{-\gamma}{\lambda_{\min}(Z^{-1}\Delta Z)}, \frac{-\gamma}{\tau^{-1}\Delta\tau}, \frac{-\gamma}{\kappa^{-1}\Delta\kappa}\right). \end{aligned} \tag{39}$$

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

  If $\Delta\tau \geq -\eta\tau$ and (37) 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

$$\begin{gathered}\tau^+ \;=\; \min(\tau_p,\, \tau_d), \qquad \kappa^+ \;=\; \begin{cases} \kappa + \alpha\,\Delta\kappa, & \text{if } \tau^+ = \tau_p \\ \kappa + \beta\,\Delta\kappa, & \text{if } \tau^+ = \tau_d, \end{cases} \\ X^+ \;=\; \tfrac{\tau^+}{\tau_p}(X + \alpha\,\Delta X), \quad y^+ \;=\; \tfrac{\tau^+}{\tau_d}(y + \beta\,\Delta y), \quad Z^+ \;=\; \tfrac{\tau^+}{\tau_d}(Z + \beta\,\Delta Z).\end{gathered} \tag{40}$$

- Update the step-length parameter by

$$\gamma^+ \;=\; 0.9 + 0.09\min(\alpha, \beta), \tag{41}$$

  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 (38). 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)

$$\mu/\mu_0 \;<\; 10^{-8}, \qquad \frac{\tau/\kappa}{\tau_0/\kappa_0} \;<\; 10^{-8}.$$

- (Predictor step)

  Solve the linear system (28)–(31), 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 (39) with $\Delta X, \Delta Z, \Delta\tau, \Delta\kappa$ replaced by $\delta X, \delta Z, \delta\tau, \delta\kappa$, respectively.

- Take σ to be

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

  where the exponent e is chosen as in (21). 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 (28)–(31) but with R_c, r_c replaced, respectively, by

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

- Update (X, y, Z, τ, κ) to $(X^+, y^+, Z^+, \tau^+, \kappa^+)$ as in (40), where α and β are computed as in (39) with γ chosen to be

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

- Update γ to γ^+ as in (41).

Remarks.

(a) The numerical instability of the Schur complement equation (28) 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 (28) when μ is smaller than 10^{-8}. In essence, 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 the primal infeasibility below a certain small level once that level has been reached. The effect of such a correction can be quite erratic, in contrast to the case of the infeasible path-following algorithms.

(c) Once again, there are other termination criteria: lack of progress or short step-lengths. Here we do not test possible infeasibility in the way we did for our infeasible-interior-point algorithms, because we have a specific termination criterion ((b) in the 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 [5], 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 of the same structure, each consisting of L 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:

$$X^0 = \mathrm{Diag}(\xi_i I_{n_i}), \quad y^0 = 0, \quad Z^0 = \mathrm{Diag}(\eta_i I_{n_i}), \tag{42}$$

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

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

By multiplying 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),
```

where `options = 1 (default)` corresponds to the initial iterate just described; and `options = 2` corresponds to the choice where `X0`, `Z0` are 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,pathreshist,info] = sdp(blk,A,C,b,X0,y0,Z0),
```

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

```
[obj,X,y,Z,gaphist,infeashist,pathreshist,info] = sdphlf(blk,A,C,b,X0,y0,Z0).
```

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`
`trace.m`	`Asum.m`	`cholaug.m`	`aasen.m`
`Atriu.m`	`pathres.m`	`corrprim.m`	`steplength.m`
`scaling.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`.

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. Likewise, computation of the inner product between two matrices is done in a C Mex rountine generated from a C program `mextrace.c` written to take advantage of possible sparsity in either matrix. Another 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`	`mexaasen.c`

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.

Output arguments.

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

`X`,`y`,`Z`: an approximately optimal solution.

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

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

`pathreshist`: a row vector that records the centrality measure $|1 - \lambda_{\min}(XZ))/\mu|$ 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 nine 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) = -10` for incorrect input;
`info(1) = 1` if there is an indication of primal infeasibility;
`info(1) = 2` if there is an indication of dual infeasibility; and
`info(1) = 3` if there is an indication of both primal and dual infeasibilities.

If `info(1)` is positive, the output variables `X,y,Z` have a different meaning: if `info(1) = 1` or `3` 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` or `3` then `X` gives an indication of dual infeasibility: $\mathtt{C}\bullet\mathtt{X} = -1$ and $\mathcal{A}\mathtt{X}$ is small.

Global variables for parameters.

`sdp.m` and `sdphlf.m` use a number of global variables which are set up in the M-file `startup.m`. If desired, the user can change the values of these parameters.

`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 2]`.

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

`gaptol`: the required accuracy in the duality gap as a fraction of the value of the objective functions. The default is `1e-13`.

`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 1 for `sdp.m`, but `0` for `sdphlf.m`.

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

One global parameter, `vers`, is not set in the M-file `startup.m` and has to be specified by the user. It takes the following values:

`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.

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 say less than 10, we 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 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;
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

`blk{i,1} = 'nondiag'  blk{i,2} = ni`
`A{i,k}, C{i} = [ni x ni double] or [ni x ni 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

`blk{i,1} = 'nondiag'  blk{i,2} = [ni(1) ni(2) ... ni(t)]`
`A{i,k}, C{i} = [ni x ni sparse].`

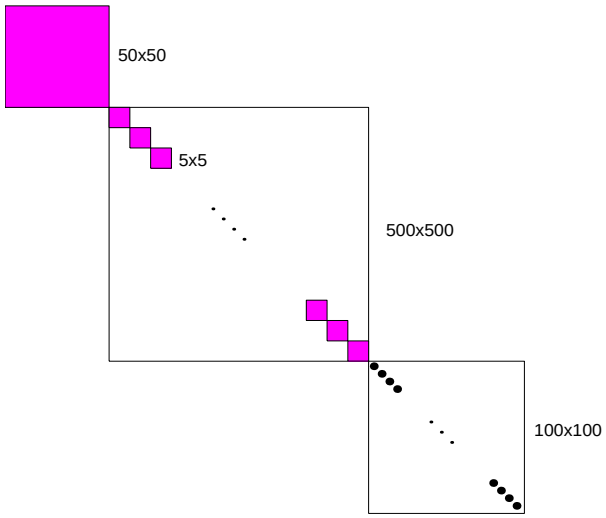

Figure 1: *An example of a block-diagonal matrix.*

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

```
blk{i,1} = 'diag'       blk{i,2} = ni
A{i,k}, C{i} = [nix1 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 the same dimensions.

For the usage of MATLAB cell arrays, refer to [7].

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 this if it 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

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

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, the input variables `feas` and `solve` are used as follows:

$$\texttt{feas} = \begin{cases} \texttt{0} & \text{corresponds to the initial iterate given in (42),} \\ \texttt{1} & \text{corresponds to a feasible initial iterate;} \end{cases}$$

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

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

```
function  [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.

(2) Norm minimization problem:

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

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

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

where `B` is a cell array with $\texttt{B}\{\texttt{k}+\texttt{1}\} = \texttt{B}_\texttt{k}$, `k = 0,...,m`.

(3) Chebyshev approximation problem for a matrix:

$$\min_{p} \|p(B)\|,$$

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

```
function  [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:

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

where $L = B - \mathrm{Diag}(Be)$, e is the vector of all ones and B is the weighted adjacency matrix of a graph [6]. The associated M-file is `maxcut.m`, with initial line

```
function [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):

$$\begin{aligned} &\max_{d \in \mathbf{R}^N} \ e^T d \\ &\text{s.t.} \quad B - \mathrm{Diag}(d) \succeq 0, \quad d \geq 0, \end{aligned}$$

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

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

(6) Lovász θ function for a graph:

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

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

```
function [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:

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

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

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

(8) Chebyshev approximation problem in the complex plane:

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

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

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

where `d` = [`d1  d2  ...  dN`].

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

(9) Control and system problem:

$$\begin{aligned} & \max_{t,P} \; t \\ \text{s.t.} \quad & -B_k^T P - P B_k \succeq 0, \quad k = 1, \ldots, L \\ & P \succeq tI, \quad I \succeq P, \quad P = P^T, \end{aligned}$$

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

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

where `B` is a cell array with `B{k}` = `Bk`, `k = 1,...,L`.

7 Sample Runs

```
>> randn('seed',0) % reset random generator to its initial seed.
>> rand('seed',0)  %
>> startup         % set global variables to default values, 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.
>> vers=1;      % use AHO direction
>> [blk,A,C,b,X0,y0,Z0] = randsdp(de,sp,di,m,feas,solve);
>>
```

```
>> [obj,X,y,Z,gaphist,infeashist] = sdp(blk,A,C,b,X0,y0,Z0); % use IPC

 condition no. of A = 1.75e+00

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

it  pstep dstep p_infeas d_infeas  gap        obj     pathres  sigma  rco
-----------------------------------------------------------------------------
 0  0.000 0.000 1.8e-16 1.3e-16  7.6e+02  1.787474e+02 0.0e+00
 1  0.897 0.614 1.5e-16 1.1e-16  1.8e+02  -5.357108e+01 8.2e-01 0.100 1.5e-01
 .    .     .      .       .        .          .          .      .      .
 .    .     .      .       .        .          .          .      .      .
 9  0.987 0.989 2.2e-16 1.3e-16  7.1e-11  -1.151599e+02 7.6e-01 0.000 3.7e-13
10  0.944 0.978 2.3e-16 1.5e-16  4.0e-12  -1.151599e+02 7.5e-01 0.000 5.0e-15

 Stop: max(relative gap, infeasibilities) < 1.00e-13
-----------------------------------
 number of iterations   = 10
 gap                    = 4.02e-12
 relative gap           = 3.49e-14
 infeasibilities        = 2.34e-16
 Total CPU time         = 3.4
 CPU time per iteration = 0.3
 termination code       =  0
-----------------------------------
>>
>> % next, generate new data with a different block structure
>> feas=0;  % and use the (infeasible) initial iterate given in (42)
>> vers=4;  % use GT direction
>> [blk,A,C,b,X0,y0,Z0] = randsdp([20 15],[4 3 3],5,30,feas,solve);
>>
>> [obj,X,y,Z,gaphist,infeashist] = sdphlf(blk,A,C,b,X0,y0,Z0); % solve using HPC

*****************************************************************************
        Homogeneous self-dual algorithms
*****************************************************************************
 version  predcorr  gam  expon  use_corrprim  sw2PC_tol
   4        1      0.000    2        0         Inf

it pstep dstep p_infeas d_infeas  gap         obj     pathres  sigma  rco
-----------------------------------------------------------------------------
 0  0.000 0.000 6.8e+00 6.2e-01 5.7e+04  6.376164e+03 0.0e+00
 1  0.837 1.000 1.6e+00 1.0e-01 5.0e+03  8.941629e+02 7.1e+00 0.359 1.6e-02
 .    .     .      .       .        .          .          .      .      .
 .    .     .      .       .        .          .          .      .      .
```

```
10  0.989 0.989 1.7e-13 8.3e-15 5.8e-10  -1.917987e+02 8.8e-01 0.000 4.0e-18
11  0.985 0.985 5.1e-15 1.7e-16 8.8e-12  -1.917987e+02 8.9e-01 0.000 4.1e-18

 Stop: max(relative gap, infeasibilities) < 1.00e-13
-----------------------------------
 number of iterations   = 11
 gap                    = 8.77e-12
 relative gap           = 4.57e-14
 infeasibilities        = 5.14e-15
 Total CPU time         = 11.8
 CPU time per iteration = 1.1
 termination code       =  0
-----------------------------------
>>
>> %%%%% MAXCUT PROBLEM %%%%%
>>
>> B = graph(50,0.3); %% generate an adjacency matrix of a 50 node graph
>>                    %% where each edge is present with probability 0.3
>> solve=1;   % generate data, then solve the problem using IPF/IPC
>> feas=1;    % use a feasible initial iterate;
>> vers=2;    % use HKM 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
   2        1      0.000   1         1          Inf

it  pstep dstep p_infeas d_infeas  gap        obj     pathres  sigma  rco
-------------------------------------------------------------------------
 0  0.000 0.000 0.0e+00 7.3e-17  2.2e+02  -2.872000e+02 0.0e+00
 1  1.000 1.000 0.0e+00 4.7e-17  8.1e+01  -2.389877e+02 8.0e-02 0.378 6.0e-01
 .    .     .      .       .        .          .          .      .      .
 .    .     .      .       .        .          .          .      .      .
11  1.000 1.000 0.0e+00 4.7e-17  6.4e-09  -2.488378e+02 2.0e-01 0.040 1.3e-10
12  0.931 1.000 0.0e+00 5.4e-17  9.8e-10  -2.488378e+02 8.7e-01 0.104 5.0e-12

  lack of progress in corrector: mucorr/mu = 0.92, corr_convg_rate = 0.08
  Stop: lack of progress in corrector.

-----------------------------------
 number of iterations   = 13
 gap                    = 9.83e-10
 relative gap           = 3.95e-12
 infeasibilities        = 5.42e-17
 Total CPU time         = 6.8
```

```
 CPU time per iteration = 0.5
 termination code       = -1
-------------------------------------
```

8 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 value (given in Section 5) for the parameters used in the algorithms.

In our experiments, we consider an SDP instance successfully solved by Algorithm IPC if the algorithm manages to reduce the relative duality gap $X \bullet Z/(1 + |C \bullet X|)$ 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 consider 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.

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 Algorithm IPC and HPC are quite similar in terms of efficiency (number of iterations) and accuracy on all the the four search directions we implemented. 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.

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	$n = 50$ $m = 50$	10.3	10.5	11.9	11.4	18.4	13.5	10.7	11.2	8.9	8.5	7.9	7.6
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	$n = 50$ $m = 50$	10.3	9.8	11.2	10.5	18.7	12.6	9.9	10.0	10.1	9.1	8.2	7.6
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.*

References

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

[2] F. Alizadeh, J.-P A. Haeberly, and M.L. Overton, *Primal-dual interior-point methods for semidefinite programming: convergence results, stability and numerical results*, Technical Report 721, Computer Science Department, NYU, New York, May 1996, to appear in SIAM J. Optimization.

[3] 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.

[4] 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`.

[5] K. Fujisawa, M. Kojima, and K. Nakata, *SDPA (semidefinite programming algorithm) — user's manual*, 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`.

[6] 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.

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

[8] M.J. Todd, K.C. Toh, R.H. Tütüncü, *On the Nesterov-Todd direction in semidefinite programming*, Technical Report 1154, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, March 1996, to appear in SIAM J. Optimization.

[9] K.C. Toh, *Search directions for primal-dual interior point methods in semidefinite programming*, manuscript, Department of Mathematics, National University of Singapore, Singapore, December 1997.

[10] 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.

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