# On the implementation and usage of SDPT3 – a MATLAB software package for semidefinite-quadratic-linear programming, version 4.0

Kim-Chuan Toh [*], Michael J. Todd [†], and Reha H. Tütüncü [‡]

Draft, 16 June 2010

**Abstract**

This software is designed to solve primal and dual semidefinite-quadratic-linear conic programming problems (known as SQLP problems) whose constraint cone is a product of semidefinite cones, second-order cones, nonnegative orthants and Euclidean spaces, and whose objective function is the sum of linear functions and log-barrier terms associated with the constraint cones. This includes the special case of determinant maximization problems with linear matrix inequalities. It employs an infeasible primal-dual predictor-corrector path-following method, with either the HKM or the NT search direction. The basic code is written in MATLAB, but key subroutines in C are incorporated via Mex files. Routines are provided to read in problems in either SDPA or SeDuMi format. Sparsity and block diagonal structure are exploited. We also exploit low-rank structures in the constraint matrices associated with the semidefinite blocks if such structures are explicitly given. To help the users in using our software, we also include some examples to illustrate the coding of problem data for our solver. Various techniques to improve the efficiency and robustness of the main solver are incorporated. For example, step-lengths associated with semidefinite cones are calculated via the Lanczos method. The current version also implements algorithms for solving a 3-parameter homogeneous self-dual model of the primal and dual SQLP problems. Routines are also provided to determine whether

[*]Department of Mathematics, National University of Singapore, Blk S17, 10 Lower Kent Ridge Road, Singapore 119076 (`mattohkc@nus.edu.sg`); and Singapore-MIT Alliance, E4-04-10, 4 Engineering Drive 3, Singapore 117576.

[†]School of Operations Research and Information Engineering, Cornell University, Ithaca, New York 14853, USA (`miketodd@cs.cornell.edu`). Research partially supported by ONR through grant N00014-08-1-0036.

[‡]Quantitative Investment Strategies, Goldman Sachs Asset Management, New York, NY 10282, USA (reha.tutuncu@gs.com).

the primal and dual feasible regions of a given SQLP have empty interiors. Numerical experiments show that this general-purpose code can solve more than 80% of a total of about 430 test problems to an accuracy of at least $10^{-6}$ in relative duality gap and infeasibilities.

## 1 Introduction

The current version of SDPT3, version 4.0, is designed to solve conic programming problems whose constraint cone is a product of semidefinite cones, second-order cones, nonnegative orthants and Euclidean spaces, and whose objective function is the sum of linear functions and log-barrier terms associated with the constraint cones. It solves the following standard form of such problems, henceforth called standard SQLP problems:

$$
\begin{aligned}
(P)\quad \min\ & \sum_{j=1}^{n_s}[\langle c_j^s,\, x_j^s\rangle - \nu_j^s \log\det(x_j^s)] \; + \sum_{i=1}^{n_q}[\langle c_i^q,\, x_i^q\rangle - \nu_i^q \log\gamma(x_i^q)] \\
& + \langle c^l,\, x^l\rangle - \sum_{k=1}^{n_l}\nu_k^l \log x_k^l \qquad + \langle c^u,\, x^u\rangle \\
\text{s.t.}\ & \sum_{j=1}^{n_s}\mathcal{A}_j^s(x_j^s) \quad + \sum_{i=1}^{n_q}A_i^q x_i^q \quad + A^l x^l \quad + A^u x^u \;=\; b, \\
& x_j^s \in K_s^{s_j}\ \forall j, \quad x_i^q \in K_q^{q_i}\ \forall i, \quad x^l \in K_l^{n_l}, \quad x^u \in I\!R^{n_u}.
\end{aligned}
$$

Here, $c_j^s$, $x_j^s$ lie in the space $\mathcal{S}^{s_j}$ of real symmetric matrices of order $s_j$ and $K_s^{s_j}$ is the cone of positive semidefinite symmetric matrices of the same order. Similarly, $c_i^q$, $x_i^q$ are vectors in $I\!R^{q_i}$ and $K_q^{q_i}$ is the quadratic or second-order cone defined by $K_q^{q_i} := \{x = [x_0; \bar{x}] \in I\!R^{q_i} : x_0 \geq \sqrt{\bar{x}^T\bar{x}}\}$. Finally, $c^l$, $x^l$ are real vectors of dimension $n_l$, $K_l^{n_l}$ is the nonnegative orthant $I\!R_+^{n_l}$, and $c^u$, $x^u$ are real vectors of dimension $n_u$. In the notation above, $\mathcal{A}_j^s$ is the linear map from $K_s^{s_j}$ to $I\!R^m$ defined by

$$
\mathcal{A}_j^s(x_j^s) = \Big[\langle a_{j,1}^s,\, x_j^s\rangle;\ \ldots\ ; \langle a_{j,m}^s,\, x_j^s\rangle\Big],
$$

where $a_{j,1}^s, \ldots, a_{j,m}^s \in \mathcal{S}^{s_j}$ are constraint matrices associated with the $j$th semidefinite block variable $x_j^s$. The matrix $A_i^q$ is an $m \times q_i$ dimensional constraint matrix corresponding to the $i$th quadratic block variable $x_i^q$, and $A^l$ and $A^u$ are the $m \times n_l$ and $m \times n_u$ dimensional constraint matrices corresponding to the linear block variable $x^l$ and the unrestricted block variable $x^u$, respectively. The notation $\langle p,\, q\rangle$ denotes the standard inner product in the appropriate space. For a given vector $u = [u_0; \bar{u}]$ in a second-order cone, we define $\gamma(u) := \sqrt{u_0^2 - \bar{u}^T\bar{u}}$. In the problem $(P)$, $\nu_j^s$, $\nu_i^q$, and $\nu_k^l$ are given nonnegative parameters.

In this paper, the vector 2-norm and matrix Frobenius norm are denoted by $\|\cdot\|$. We use the MATLAB notation $[U; V]$ to denote the matrix obtained by appending $V$ below the last row of $U$. For a given matrix $U$, we use the notation $U(k, :)$ and $U(:, k)$ to denote the $k$th row and column of $U$, respectively.

Let $\mathbf{svec} : \mathcal{S}^n \to I\!R^{n(n+1)/2}$ be the vectorization operator on symmetric matrices defined by $\mathbf{svec}(X) = [X_{11}, \sqrt{2}X_{12}, X_{22}, \ldots, \sqrt{2}X_{1n}, \ldots, \sqrt{2}X_{n-1,n}, X_{nn}]^T$. For computational purposes, it is convenient to identify $\mathcal{A}_j^s$ with the following $m \times \bar{s}_j$ matrix (where $\bar{s}_j = s_j(s_j+1)/2$):

$$A_j^s = \Big[\mathbf{svec}(a_{j,1}^s); \ \ldots; \ \mathbf{svec}(a_{j,m}^s)\Big].$$

With the matrix representation of $\mathcal{A}_j^s$, we have that $\mathcal{A}_j^s(x_j^s) = A_j^s \mathbf{svec}(x_j^s)$.

The software also solves the dual problem associated with the problem $(P)$ above:

$$\begin{aligned}(D) \quad \max \ & b^T y + \sum_{j=1}^{n_s} [\nu_j^s \log\det(z_j^s) + s_j \nu_j^s (1 - \log \nu_j^s)] \\ & + \sum_{i=1}^{n_q} [\nu_i^q \log \gamma(z_i^q) + \nu_i^q(1-\log\nu_i^q)] \ + \ \sum_{k=1}^{n_l} [\nu_k^l \log z_k^l + \nu_k^l (1 - \log \nu_k^l)]\end{aligned}$$

$$\begin{array}{rrcrcll} \text{s.t.} & (\mathcal{A}_j^s)^T y & + & z_j^s & = & c_j^s, \quad z_j^s \in K_s^{s_j}, & j = 1\ldots, n_s \\ & (A_i^q)^T y & + & z_i^q & = & c_i^q, \quad z_i^q \in K_q^{q_i}, & i = 1\ldots, n_q \\ & (A^l)^T y & + & z^l & = & c^l, \quad z^l \in K_l^{n_l}, & \\ & (A^u)^T y & & & = & c^l, \quad y \in I\!R^m. & \end{array}$$

In the notation above, $(\mathcal{A}_j^s)^T$ is the adjoint of $\mathcal{A}_j^s$ defined by $(\mathcal{A}_j^s)^T y = \sum_{k=1}^m y_k a_{j,k}^s$.

For later convenience, we introduce the following notation:

$$x^s = (x_1^s; \ldots; x_{n_s}^s), \quad x^q = [x_1^q; \ldots; x_{n_q}^q], \quad A^q = \Big[A_1^q, \ \ldots, \ A_{n_q}^q\Big],$$

where the notation $(x_1^s; \ldots; x_{n_s}^s)$ means that the objects $x_j^s$ are placed in a column format. We define $c^s, z^s, c^q$, and $z^q$ analogously. Let $\mathcal{A}^s(x^s) = \sum_{j=1}^{n_s} \mathcal{A}_j^s(x_j^s)$, $(\mathcal{A}^s)^T y = ((\mathcal{A}_1^s)^T y; \ \ldots; \ (\mathcal{A}_{n_s}^s)^T y)$, and $c = (c^s; c^q; c^l; c^u)$, $x = (x^s; x^q; x^l; x^u)$, $z = (z^s; z^q; z^l; 0)$. Finally, we define

$$\mathcal{A}(x) = \mathcal{A}^s(x^s) + A^q x^q + A^l x^l + A^u x^u, \quad \mathcal{A}^T(y) = \Big((\mathcal{A}^s)^T y; \ (A^q)^T y; \ (A^l)^T y; \ (A^u)^T y\Big),$$

$$\begin{aligned} K &= K_s^{s_1} \times \cdots \times K_s^{s_{n_s}} \times K_q^{q_1} \times \cdots \times K_q^{q_{n_q}} \times K_l^{n_l} \times I\!R^{n_u}, \\ K^* &= K_s^{s_1} \times \cdots \times K_s^{s_{n_s}} \times K_q^{q_1} \times \cdots \times K_q^{q_{n_q}} \times K_l^{n_l} \times \{0\}, \end{aligned}$$

so that the equality constraints of $(P)$ and $(D)$ can be written more compactly as follows:

$$\mathcal{A}(x) = b, \quad x \in K, \qquad \mathcal{A}^T(y) + z = c, \quad z \in K^*. \tag{1}$$

The matrix representation of $\mathcal{A}$ is given by

$$A = \Big[A_1^s, \ldots, A_{n_s}^s, \ A^q, \ A^l, \ A^u\Big]. \tag{2}$$

The software package was originally developed to provide researchers in semidefinite programming with a collection of reasonably efficient and robust algorithms that could solve general SDPs (semidefinite programming problems) with matrices of dimensions of the order of a hundred. The current release expands the family of problems solvable by the software and made several enhancements described below.

1. This version is faster than the previous releases, e.g. [34], [39], especially on large sparse problems, and consequently can solve larger problems.
2. The current release can also solve problems that have explicit log-barrier terms in the objective functions. Hence determinant maximization problems can be solved.
3. The solver is more robust in handling unrestricted variables.
4. Low-rank structures in the constraint matrices associated with the semidefinite blocks can be exploited to improve computational efficiency and memory requirements.
5. The current release also implements primal-dual predictor-corrector path-following algorithms for solving a 3-parameter homogeneous self-dual model of $(P)$ and $(D)$.
6. Routines are provided to compute the geometric measures proposed by Freund [9] for $(P)$ and $(D)$. These geometric measures give information on the "thickness" of the feasible regions of $(P)$ and $(D)$.

All the numerical experiments in this paper were performed on an Intel Xeon 3.0GHz personal computer with 4GB of physical memory using MATLAB version 7.6 on a Linux operating system.

**Organization of the paper.** In Section 2, we describe the representation of SQLP data by cell arrays. The SQLP solver `sqlp.m` in our software is described in Section 3. In Section 4, we present a few SQLP examples to illustrate the usage of our software. Implementation details such as the computation of search directions are given in Section 5. In Section 6, we describe a 3-parameter homogeneous self-dual model for the problem $(P)$ without logarithmic terms and unrestricted variables. In Section 7, the geometric measures of Freund and their computation are presented. Finally, the last section reports computational results obtained by SDPT3 on about 430 SQLP problems.

**Installation.** The current version is written in MATLAB version 7.4 or later releases. It is available from the internet site:

`http://www.math.nus.edu.sg/~mattohkc/sdpt3.html`

Our software uses a number of Mex routines generated from C programs written to carry out certain operations that MATLAB is not efficient at. In particular, operations such as extracting selected elements of a matrix, and performing arithmetic operations on these selected elements are all done in C. As an example, the vectorization operation **svec** is coded in the C program `mexsvec.c`. To install SDPT3 and generate these Mex routines, the user can simply follow the steps below:

(a) unzip SDPT3-4.0.zip;

(b) run MATLAB in the directory SDPT3-4.0;

(c) run the m-file `Installmex.m`.

After that, to see whether you have installed SDPT3 correctly, run the m-files:

```
>> startup
>> sqlpdemo
```

## 2 Cell array representation of problem data

Our implementation exploits the block structure of the given SQLP problem. In the internal representation of the problem data, we classify each semidefinite block into one of the following two types:

1. a dense or sparse matrix of order greater than or equal to 100;
2. a sparse block-diagonal matrix consisting of numerous sub-blocks each of order less than 100.

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.

For the quadratic part, we typically group all quadratic blocks (small or large) into a single block, though it is not mandatory to do so. If there are a large number of small blocks, it is advisable to group them all together as a single large block consisting of numerous small sub-blocks for the same reason we just mentioned.

Let $L$ be the total number of blocks in the SQLP problem. If all the various types of blocks are present in $(P)$, then $L = n_s + n_q + 2$. For each SQLP problem, the block structure of the problem data is described by an $L \times 2$ cell array named `blk`. The content of each of the elements of the cell arrays is given as follows. If the $j$th block is a semidefinite block consisting of a single block of size $\mathtt{s_j}$, then

$$\mathtt{blk\{j,1\} = 's'}, \quad \mathtt{blk\{j,2\}} = [\mathtt{s_j}],$$

$$\mathtt{At\{j\}} = [\bar{\mathtt{s}}_\mathtt{j} \,\mathtt{x}\, \mathtt{m}\ \ \mathtt{sparse}],$$

$$\mathtt{C\{j\}},\ \mathtt{X\{j\}},\ \mathtt{Z\{j\}} = [\mathtt{s_j}\,\mathtt{x}\,\mathtt{s_j}\ \ \mathtt{double\ or\ sparse}],$$

where $\bar{\mathtt{s}}_\mathtt{j} = \mathtt{s_j}(\mathtt{s_j} + 1)/2$.

If the $j$th block is a semidefinite block consisting of numerous small sub-blocks, say $p$ of them, of orders $\mathtt{s_{j1}}, \mathtt{s_{j2}}, \ldots, \mathtt{s_{jp}}$ such that $\sum_{k=1}^{p} \mathtt{s_{jk}} = \mathtt{s_j}$, then

$$\texttt{blk\{j,1\} = 's'}, \quad \texttt{blk\{j,2\}} = [\mathtt{s_{j1}},\ \mathtt{s_{j2}},\ \cdots,\ \mathtt{s_{jp}}],$$
$$\texttt{At\{j\}} = [\mathtt{\bar{s}_j \, x \, m\ \ sparse}],$$
$$\texttt{C\{j\}},\ \texttt{X\{j\}},\ \texttt{Z\{j\}} = [\mathtt{s_j \, x \, s_j\ \ sparse}],$$

where $\mathtt{\bar{s}_j} = \sum_{\mathtt{k=1}}^{\mathtt{p}} \mathtt{s_{jk}}(\mathtt{s_{jk}} + 1)/2$.

Notice that we store all the constraint matrices associated with the $j$th semidefinite block in vectorized form as a single $\bar{s}_j \times m$ matrix where the $k$th column of this matrix corresponds to the $k$th constraint matrix. That is, `At{j}(:,k) = svec(blk(j,:),`$a^s_{j,k}$`)`. (Here `svec` has a new argument because, if the $j$th semidefinite block consists of several small sub-blocks, it needs to concatenate the **svec**'s of each sub-block.)

The above storage scheme for the data matrix $A^s_j$ associated with the semidefinite blocks of the SQLP problem represents a departure from earlier versions (version 2.3 or earlier) of our implementation, such as the one described in [34]. Previously, the constraint matrices were stored in an $n_s \times m$ cell array `AA`, where `AA{j,k} =` $a^s_{j,k}$, and it was stored as an individual matrix in either dense or sparse format. The data format we used in earlier versions of SDPT3 was more natural, but our current data representation was adopted for the sake of computational efficiency. The reason for such a change is again due to the fact that it is less efficient for MATLAB to work with a cell array with many cells. But note that it is easy to use the function `svec.m` provided in SDPT3 to convert `AA` into the new storage scheme as follows: `At(j) = svec(blk(j,:),AA(j,:))`.

While we now store the constraint matrix in vectorized form, the parts of the iterates `X` and `Z` corresponding to semidefinite blocks are still stored as matrices, since that is how the user wants to access them.

The data storage scheme corresponding to quadratic, linear, and unrestricted blocks is rather straightforward. If the $i$th block is a quadratic block consisting of numerous sub-blocks, say $p$ of them, of dimensions $\mathtt{q_{i1}}, \mathtt{q_{i2}}, \ldots, \mathtt{q_{ip}}$ such that $\sum_{k=1}^{p} \mathtt{q_{ik}} = \mathtt{q_i}$, then

$$\texttt{blk\{i,1\} = 'q'}, \quad \texttt{blk\{i,2\}} = [\mathtt{q_{i1}},\ \mathtt{q_{i2}},\ \cdots,\ \mathtt{q_{ip}}],$$
$$\texttt{At\{i\}} = [\mathtt{q_i \, x \, m\ sparse}],$$
$$\texttt{C\{i\}},\ \texttt{X\{i\}},\ \texttt{Z\{i\}} = [\mathtt{q_i \, x \, 1\ \ double\ or\ sparse}].$$

If the $k$th block is the linear block, then

$$\texttt{blk\{k,1\} = 'l'}, \quad \texttt{blk\{k,2\}} = \mathtt{n_l},$$
$$\texttt{At\{k\}} = [\mathtt{n_l \, x \, m\ \ sparse}],$$
$$\texttt{C\{k\}},\ \texttt{X\{k\}},\ \texttt{Z\{k\}} = [\mathtt{n_l \, x \, 1\ \ double\ or\ sparse}].$$

Similarly, if the $k$th block is the unrestricted block, then

$$\texttt{blk\{k,1\} = 'u'}, \quad \texttt{blk\{k,2\}} = \mathtt{n_u},$$
$$\texttt{At\{k\}} = [\mathtt{n_u \, x \, m\ \ sparse}],$$
$$\texttt{C\{k\}},\ \texttt{X\{k\}},\ \texttt{Z\{k\}} = [\mathtt{n_u \, x \, 1\ \ double\ or\ sparse}].$$

(It is possible to have several linear or unrestricted blocks, but it is more efficient to reformulate such a problem by combining all linear blocks and similarly all unrestricted blocks.)

## 2.1 Specifying the block structure of problems

Our software requires the user to specify the block structure of the SQLP problem. Although no technical difficulty will arise if the user chooses to lump a few blocks together and consider it as a single large block, the computational time can be dramatically different. For example, the problem `qpG11` in the SDPLIB library [2] actually has the block structure: `blk{1,1} = 's', blk{1,2} = 800, blk{2,1} = 'l', blk{2,2} = 800`, but the structure specified in the library is `blk{1,1} = 's', blk{1,2} = 1600`. That is, in the former, the linear variables are explicitly identified, rather than being part of a large sparse semidefinite block. The difference in the running time for specifying the block structure differently is dramatic: the former representation is at least six times faster when the HKM direction is used, besides using much less memory space.

It is thus crucial to present problems to the algorithms correctly. We could add our own preprocessor to detect this structure, but believe users are aware of linear variables present in their problems. Unfortunately the versions of `qpG11` (and also `qpG51`) in SDPLIB do not show this structure explicitly. In our software, we provide an m-file, `detect_lblk.m`, to detect problems with linear variables. The user can call this m-file after loading the problem data into Matlab as follows:

```
>> [blk,At,C,b] = read_sdpa('./sdplib/qpG11.dat-s');
>> [blk2,At2,C2] = detect_lblk(blk,At,C,b);
```

Internally, the solvers in SDPT3 would automatically call `detect_lblk.m` to detect the presence of linear variables in a semidefinite block before solving $(P)$ and $(D)$.

## 2.2 Storing constraint matrices with low-rank structures

A new feature of the current version of SDPT3 is that it can exploit low-rank structures present in the constraint matrices associated with the semidefinite blocks. To do so, the user needs to specify the low-rank structures in the constraint matrices explicitly when coding the problem data. The purpose here is to explain how this is done.

Suppose the $j$th row of `blk` corresponds to a semidefinite block. To simplify implementation, we exploit possible low-rank structures only when this semidefinite block is a single block. That is, $\texttt{blk\{j,2\}} = [\texttt{s}_\texttt{j}]$. Suppose that the first $p$ matrices, $a^s_{j,1}, \ldots a^s_{j,p}$, have no low-rank structures, and the remaining matrices $a^s_{j,p+1}, \ldots, a^s_{j,m}$ have such structures with

$$a^s_{j,k} \;=\; V_k D_k V_k^T, \quad k = p+1, \ldots, m,$$

where $V_k \in I\!R^{s_j \times r_{j,k}}$ is a low-rank matrix with $r_{j,k} \ll s_j$, and $D_k \in I\!R^{r_{j,k} \times r_{j,k}}$ is a symmetric matrix. The low-rank structures of these matrices should be recorded as follows:

$$\texttt{blk\{j,1\} = 's'}, \quad \texttt{blk\{j,2\}} = [\texttt{s}_\texttt{j}], \quad \texttt{blk\{j,3\}} = [\texttt{r}_{\texttt{j,p+1}}, \ldots, \texttt{r}_{\texttt{j,m}}],$$

$$\texttt{At\{j,1\}} = [\bar{\texttt{s}}_\texttt{j}\,\texttt{x}\,\texttt{p sparse}], \quad \texttt{At\{j,2\}} = [\texttt{V}_{\texttt{j,p+1}}, \ldots, \texttt{V}_{\texttt{j,m}}], \quad \texttt{At\{j,3\}} = \texttt{dd},$$

where `dd` is a 4-column matrix that records the non-zero elements of $D_k$, $k = p + 1, \ldots, m$, and a row (say, $i$th row) of `dd` has the following form:

$$\texttt{d(i,:)} = [\text{constraint number} - p, \ \text{row index}, \ \text{column index}, \ \text{non-zero value}].$$

If all the matrices $D_k$ are diagonal, then the user can simply set `dd` to be the following column vector:

$$\texttt{dd} = [\text{diag}\,(D_{p+1});\ \ldots;\ \text{diag}\,(D_m)].$$

In the subdirectory `Examples`, we give an m-file `randlowranksdp.m` to generate random SDP problems with low-rank constraint matrices, whose calling syntax is:

```
[blk,At,C,b,bblk,AAt] = randlowranksdp(n,p,m2,r)
```

It will generate an SDP where the first `p` constraint matrices have no low-rank structures, and the remaining `m2` matrices have low-rank structures and each matrix has rank `r`. The output `[blk,At,C,b]` explicitly describes the low-rank structure as above, while `[bblk,AAt,C,b]` encodes the same SDP, but without including the low-rank structure information.

## 3 The main functions: `sqlp.m`, `HSDsqlp.m`, `sdpt3.m`

The main algorithm implemented in SDPT3 for solving $(P)$ and $(D)$ is an infeasible primal-dual path-following algorithm, described in detail in Appendix A. At each iteration, we first compute a *predictor* search direction aimed at decreasing the duality gap as much as possible. After that, the algorithm generates a Mehrotra-type corrector step [23] with the intention of keeping the iterates close to the central path. However, we do not impose any neighborhood restrictions on our iterates.[1] Initial iterates need not be feasible — the algorithm tries to achieve feasibility and optimality of its iterates simultaneously. It should be noted that in our implementation, the user has the option to use a primal-dual path-following algorithm that does not use corrector steps.

The main routine that corresponds to Algorithm IPC described in Appendix A for solving $(P)$ and $(D)$ is `sqlp.m`, whose calling syntax is as follows:

```
[obj,X,y,Z,info,runhist] = sqlp(blk,At,C,b,OPTIONS,X0,y0,Z0).
```

For an SQLP problem without logarithmic terms or unrestricted variables $x^u$, we also implemented an algorithm that is analogous to Algorithm IPC for a 3-parameter homogeneous self-dual model of $(P)$. The routine for solving the HSD model of $(P)$ is `HSDsqlp.m`, with the following syntax:

[1]This strategy works well on most of the problems we tested. However, it should be noted that the occasional failure of the software on problems with poorly chosen initial iterates is likely due to the lack of a neighborhood enforcement in the algorithm.

```
[obj,X,y,Z,info,runhist] = HSDsqlp(blk,At,C,b,OPTIONS,X0,y0,Z0).
```

Note that if there are unrestricted variables $x^u$ present in $(P)$, `HSDsqlp.m` can still be called to solve the problem since it would automatically express $x^u$ as $x^u = x^u_+ - x^u_-$ with $x^u_+, x^u_- \geq 0$ to reformulate $(P)$ into an SQLP without unrestricted variables. Our numerical experience has indicated that for an SQLP with unrestricted variables in $(P)$, `HSDsqlp.m` would typically deliver more accurate solutions than `sqlp.m` since the former generally would encounter less severe numerical difficulties compared to the latter. But for an SQLP problem without unrestricted variables, `HSDsqlp.m` generally would take more iterations than `sqlp.m` to solve the problem to the same accuracy. Based on the consideration of computational efficiency and attainable accuracy/robustness, we have a hybrid solver `sdpt3.m` that would automatically choose between `sqlp.m` and `HSDsqlp.m` based on the characteristics of the SQLP problem to be solved. Again, the calling syntax of `sdpt3.m` is similar to that for `sqlp.m`.

**Input arguments.**

`blk`: a cell array describing the block structure of the SQLP problem.

`At`, `C`, `b`: SQLP data.

`OPTIONS`: a structure array of parameters (optional).

`X0`, `y0`, `Z0`: an initial iterate (optional).

If the input argument `OPTIONS` is omitted, default values specified in the function `sqlparameters.m` are used. More detail on `OPTIONS` is given in Section 3.1.

**Output arguments.**

The names chosen for the output arguments explain their contents. The argument `info` is a structure array containing performance information such as `info.termcode, info.obj, info.gap, info.pinfeas, info.dinfeas, info.cputime` whose meanings are explained in `sqlp.m`. The argument `runhist` is a structure array which records the history of various performance measures during the run; for example, `runhist.gap` records the complementarity gap at each interior-point iteration.

Note that, while `(X,y,Z)` normally gives approximately optimal solutions, if `info.termcode` is 1 the problem is suspected to be primal infeasible and `(y,Z)` is an approximate certificate of infeasibility, with $\texttt{b}^T\texttt{y} = 1$, `Z` in the appropriate cone, and $\texttt{A}^T\texttt{y} + \texttt{Z}$ small, while if `info.termcode` is 2 the problem is suspected to be dual infeasible and `X` is an approximate certificate of infeasibility, with $\langle \texttt{C}, \texttt{X} \rangle = -1$, `X` in the appropriate cone, and `AX` small. Note that `A` is defined in (2).

**Caveats.**

(a) The user should be aware that SQLP 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. In our very limited testing on strongly infeasible problems, our algorithms have been quite successful in detecting infeasibility.
(b) Since our algorithm is a primal-dual method storing the primal iterate `X`, it cannot exploit common sparsity in `C` and the constraint matrices as well as dual methods or nonlinear-programming based methods. Thus our software may not be able to handle dense or sparse semidefinite blocks (with a single block) with order more than 3000 on an average PC available in 2010.
(c) Our interior-point algorithms are designed based on the existence of a central path in the interior of the primal-dual feasible region of $(P)$ and $(D)$. For problems where the primal-dual feasible region is non-empty but has an empty interior, our SQLP solver can generally still deliver a reasonably good approximate optimal solution, but the solver tends to encounter numerical difficulties before a high accuracy solution can be obtained.

## 3.1 The structure array `OPTIONS` for parameters

`sqlp.m` uses a number of parameters which are specified in a Matlab structure array called `OPTIONS` in the m-file `sqlparameters.m`. If desired, the user can change the values of these parameters. The meaning of the specified fields in `OPTIONS` are given in the m-file itself. As an example, if the user does not wish to use corrector steps in Algorithm IPC, then he/she can do so by setting `OPTIONS.predcorr = 0`. If the user wants to use a fixed value, say 0.98, for the step-length parameter $\bar{\gamma}$ in Algorithm IPC instead of the adaptive strategy used in the default, he/she can achieve that by setting `OPTIONS.gam = 0.98`. Similarly, if the user wants to solve the SQLP problem to an accuracy tolerance of `1e-4` instead of the default value of `1e-8` while using the default values for all other parameters, he/she only needs to set `OPTIONS.gaptol = 1e-4`.

The defaults in `sqlparameters.m` assume that the parameters $\nu_j^s, \nu_i^q, \nu_k^l$ in $(P)$ are all 0. For an SQLP problem where some of the parameters $\nu_j^s, \nu_i^q, \nu_k^l$ are positive, the user needs to specify an $L \times 1$ cell array `OPTIONS.parbarrier` to store the values of these parameters (including zeros) as follows. If the $j$th block is a semidefinite block consisting of one or more sub-blocks, say $p$ of them, of orders $\mathtt{s_{j1}}, \mathtt{s_{j2}}, \ldots, \mathtt{s_{jp}}$, then

$$\texttt{OPTIONS.parbarrier}\{\texttt{j}\} = [\nu^{\mathtt{s}}_{\mathtt{j1}},\ \ \nu^{\mathtt{s}}_{\mathtt{j2}},\ \ \cdots,\ \ \nu^{\mathtt{s}}_{\mathtt{jp}}].$$

If the $i$th block is a quadratic block consisting of one or more sub-blocks, say $p$ of them, of dimensions $\mathtt{q_{i1}}, \mathtt{q_{i2}}, \ldots, \mathtt{q_{ip}}$, then

$$\texttt{OPTIONS.parbarrier}\{\texttt{i}\} = [\nu^{\mathtt{q}}_{\mathtt{i1}},\ \ \nu^{\mathtt{q}}_{\mathtt{i2}},\ \ \cdots,\ \ \nu^{\mathtt{q}}_{\mathtt{ip}}].$$

If the $k$th block is the linear block, then

$$\texttt{OPTIONS.parbarrier}\{\texttt{k}\} = [\nu^{\mathtt{l}}_{1},\ \ \nu^{\mathtt{l}}_{2},\ \ \cdots,\ \ \nu^{\mathtt{l}}_{\mathtt{n_l}}],$$

while if the $k$th block is the unrestricted block, then

OPTIONS.parbarrier{k} = zeros(1, n_u).

The reader is referred to Section 4.3 for an example where the objective function in $(D)$ is given by $\log\det(z^s)$.

## 3.2 Running problems in SDPA and SeDuMi format

We provide two m-files, `read_sdpa.m` and `read_sedumi.m`, to respectively convert problem data stored in SDPA [12] and SeDuMi [29] format into Matlab cell arrays described above. For an user who has the problem data generated in SDPT3 format, he/she can convert it to the SeDuMi format by using the function `SDPT3data_SEDUMIdata.m`.

The subdirectory `sdplib` in SDPT3 contains a few problems in SDPA format that are extracted from the SDPLIB library [2], while the subdirectory `dimacs` contains problems in SeDuMi format that are extracted from the DIMACS library [27]. Assuming that the current directory is SDPT3-4.0, we can read in and run the test problem `mcp250-1.dat-s` in the subdirectory `sdplib` as follows:

```
>> startup     % set up Matlab paths
>> [blk,At,C,b] = read_sdpa('./sdplib/mcp250-1.dat-s');
>> [obj,X,y,Z,info] = sqlp(blk,At,C,b);

 num. of constraints = 250
 dim. of sdp    var  = 250,   num. of sdp  blk  =  1
*******************************************************************
   SDPT3: Infeasible path-following algorithms
*******************************************************************
 version  predcorr  gam  expon  scale_data
   HKM      1      0.000   1        0
it  pstep dstep p_infeas d_infeas  gap     mean(obj)    cputime
-------------------------------------------------------------------
 0|0.000|0.000|2.3e+02|1.8e+01|9.9e+05|-4.137500e+04  0.000000e+00| 0:0:00| chol  1  1
 1|0.987|1.000|3.0e+00|3.0e-01|1.7e+04|-7.026217e+02 -4.022482e+03| 0:0:00| chol  1  1
 2|1.000|1.000|1.9e-07|3.0e-02|2.3e+03|-1.700591e+02 -2.468154e+03| 0:0:00| chol  1  1
 :    :     :      :       :        :           :            :        :
13|1.000|0.996|3.5e-13|1.0e-12|2.1e-05|-3.172643e+02 -3.172643e+02| 0:0:01| chol  1  1
14|1.000|1.000|6.7e-13|1.0e-12|6.2e-07|-3.172643e+02 -3.172643e+02| 0:0:01|
 Stop: max(relative gap, infeasibilities) < 1.00e-08
-------------------------------------------------------------------
 number of iterations   = 14
 primal objective value = -3.17264340e+02
 dual   objective value = -3.17264340e+02
 gap := trace(XZ)       = 6.17e-07
 relative gap           = 9.71e-10
 actual relative gap    = 9.70e-10
 rel. primal infeas     = 6.66e-13
 rel. dual   infeas     = 1.00e-12
 norm(X), norm(y), norm(Z) = 1.3e+02, 2.3e+01, 1.3e+01
 norm(A), norm(b), norm(C) = 1.7e+01, 1.7e+01, 1.5e+01
 Total CPU time (secs)  = 1.0
 CPU time per iteration = 0.1
 termination code       =  0
 DIMACS: 5.6e-12  0.0e+00  5.4e-12  0.0e+00  9.7e-10  9.7e-10
-------------------------------------------------------------------
```

We can solve a DIMACS test problem in a similar manner.

```
>> OPTIONS.vers = 2;  % use NT direction
>> [blk,At,C,b,perm] = read_sedumi('./dimacs/nb.mat');
>> [obj,X,y,Z,info] = sdpt3(blk,At,C,b,OPTIONS);
>> [x,y,z] = SDPT3soln_SEDUMIsoln(blk,X,y,Z,perm);
```

In the above, we call the hybrid solver `sdpt3.m` to solve the SQLP, and in the last line, we convert the solution in SDPT3 format to SeDuMi format.

## 3.3 Stopping criteria

We define

$$n \;=\; \sum_{\{j:\nu^s_j=0\}} s_j + \sum_{\{i:\nu^q_i=0\}} q_i + |\{k : \nu^l_k = 0\}| \tag{3}$$

$$\mu(x,z) \;=\; \frac{1}{n}\sum_{\alpha\in\{s,q,l\}}\sum_{j=1}^{n_\alpha}\begin{cases}\langle x^\alpha_j,\, z^\alpha_j\rangle & \text{if } \nu^\alpha_j = 0\\ 0 & \text{otherwise.}\end{cases} \tag{4}$$

$$\begin{aligned}\texttt{gap} \;=\;& \langle x,\, z\rangle - \sum_{\{j:\nu^s_j>0\}} \nu^s_j\Big(s_j + \log\det(x^s_j z^s_j/\nu^s_j)\Big)\\ & - \sum_{\{i:\nu^q_i>0\}} \nu^q_i\Big(1+\log(\gamma(x^q_i)\gamma(z^q_i)/\nu^q_i)\Big) - \sum_{\{k:\nu^l_k>0\}} \nu^l_k\Big(1+\log(x^l_k z^l_k/\nu^l_k)\Big).\end{aligned} \tag{5}$$

Note that if $n = 0$, we define $\mu(x,z) = 0$.

The algorithm is stopped when any of the following cases occur.

1. solutions with the desired accuracy have been obtained, i.e.,

$$\phi := \max\{\texttt{relgap}, \texttt{pinfeas}, \texttt{dinfeas}\} \le \texttt{OPTIONS.gaptol}, \tag{6}$$

   where

$$\texttt{relgap} = \frac{\texttt{gap}}{1+|\langle c,\, x\rangle| + |b^T y|},\; \texttt{pinfeas} = \frac{\|\mathcal{A}(x)-b\|}{1+\|b\|},\; \texttt{dinfeas} = \frac{\|\mathcal{A}^T(y)+z-c\|}{1+\|c\|}.$$

2. primal infeasibility is suggested because

$$b^T y/\|\mathcal{A}^T y + z\| > 10^8;$$

3. dual infeasibility is suggested because

$$-c^T x/\|\mathcal{A}x\| > 10^8;$$

4. slow progress is detected, measured by a rather complicated set of tests including
$$\texttt{relgap} < \max\{\texttt{pinfeas}, \texttt{dinfeas}\} \; ;$$
5. numerical problems are encountered, such as the iterates not being positive definite or the Schur complement matrix not being positive definite; or
6. the step sizes fall below $10^{-6}$.

## 3.4 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 [13], it is desirable to choose an initial iterate that at least has the same order of magnitude as an optimal solution of the SQLP. If a feasible starting point is not known, we recommend that the following initial iterate be used:

$$\begin{aligned} y^0 &= 0, \\ (x_j^s)^0 = \xi_j^s \, I_{s_j}, \quad (z_j^s)^0 &= \eta_j^s \, I_{s_j}, \quad j = 1, \ldots, n_s, \\ (x_i^q)^0 = \xi_i^q \, e_i^q, \quad (z_i^q)^0 &= \eta_i^q \, e_i^q, \quad i = 1, \ldots, n_q, \\ (x^l)^0 = \xi^l \, e^l, \quad (z^l)^0 &= \eta^l \, e^l, \quad (x^u)^0 = 0, \end{aligned}$$

where $I_{s_j}$ is the identity matrix of order $s_j$, $e_i^q$ is the first $q_i$-dimensional unit vector, $e^l$ is the $l$-vector of all ones, and

$$\begin{aligned} \xi_j^s &= \max\left\{10, \sqrt{s_j}, \; s_j \max_{1 \le k \le m} \frac{1 + |b_k|}{1 + \|a_{j,k}^s\|_F}\right\}, \\ \eta_j^s &= \max\Big\{10\,, \sqrt{s_j}, \; \max\{\|c_j^s\|_F, \|a_{j,1}^s\|_F, \ldots, \|a_{j,m}^s\|_F\}\Big\}, \\ \xi_i^q &= \max\left\{10\,, \sqrt{q_i}, \; \sqrt{q_i} \max_{1 \le k \le m} \frac{1 + |b_k|}{1 + \|A_i^q(k,:)\|}\right\}, \\ \eta_i^q &= \max\{10\,, \sqrt{q_i}, \; \max\{\|c_i^q\|, \|A_i^q(1,:)\|, \ldots, \|A_i^q(m,:)\|\}\}, \\ \xi^l &= \max\left\{10\,, \sqrt{n_l}, \; \sqrt{n_l} \max_{1 \le k \le m} \frac{1 + |b_k|}{1 + \|A^l(k,:)\|}\right\}, \\ \eta^l &= \max\Big\{10\,, \sqrt{n_l}, \; \max\{\|c^l\|, \|A^l(1,:)\|, \ldots, \|A^l(m,:)\|\}\Big\}. \end{aligned}$$

By multiplying the identity matrix $I_{s_j}$ by the factors $\xi_j^s$ and $\eta_j^s$ for the semidefinite blocks, and similarly for the quadratic and linear blocks, the initial iterate has a better chance of having the appropriate order of magnitude.

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

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

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

## 3.5 Preprocessing

**Nearly dependent constraints.**

The primal-dual path-following algorithm we implemented assumes that the matrix $A$ in (2) has full column rank. But in our software, the presence of (nearly) dependent constraints is detected automatically, and warning messages are displayed if such constraints exist. When this happens, the user has the option of removing these (nearly) dependent constraints by calling a preprocessing routine to remove them by setting `OPTIONS.rmdepconstr = 1`. The routine (`checkdepconstr.m`) we have coded to detect nearly dependent constraints is based on computing the sparse $LDL^T$ factorization of $AA^T$. Such a method is fast but is not as reliable as the method that is based on sparse $LU$ factorization of $A$.

**Detecting diagonal blocks.**

We provide the m-file, `detect_lblk.m`, to look for diagonal blocks in semidefinite blocks in the data: see Subsection 2.1 for the use of this subroutine.

**Detecting unrestricted blocks.**

We have provided a routine, `detect_ublk.m`, to detect unrestricted variables that have been modelled as the difference of two nonnegative variables.

**Complex data.**

In earlier versions, 2.3 or earlier, SDPT3 could directly handle complex data in SDP, i.e., the case where the constraint matrices are hermitian matrices. However, as problems with complex data rarely occur in practice, and in an effort to simplify the code, we removed this flexibility from subsequent versions.

Users can still solve an SDP with complex data using SDPT3-4.0. This is done by calling the m-file `convertcmpsdp.m` to convert the SDP into one with real data. But unlike the earlier versions, here we convert the problem into one with real data by doubling the size of the constraint matrices. Let $B$ be an $n \times n$ hermitian matrix. The conversion is based on the following equivalence:

$$B \text{ is positive semidefinite } \Leftrightarrow \begin{bmatrix} B^R & -B^I \\ B^I & B^R \end{bmatrix} \text{ is positive semidefinite,}$$

where $B^R$ and $B^I$ denote the real and imaginary parts of $B$, respectively. Note that since $B$ is hermitian, $B^R$ is symmetric and $B^I$ is skew-symmetric.

Now suppose $C, A_1, \dots A_m$ are given $n \times n$ hermitian matrices. Then $C - \sum_{k=1}^m y_k A_k \succeq 0$ if and only

$$\begin{bmatrix} C^R & -C^I \\ C^I & C^R \end{bmatrix} - \sum_{k=1}^m y_k^R \begin{bmatrix} A_k^R & -A_k^I \\ A_k^I & A_k^R \end{bmatrix} - \sum_{k=1}^m y_k^I \begin{bmatrix} -A_k^I & -A_k^R \\ A_k^R & -A_k^I \end{bmatrix} \succeq 0. \qquad (7)$$

Notice that the matrices $[-A_k^I, -A_k^R; A_k^R, -A_k^I]$ are skew-symmetric. For a complex SDP, the vector $b$ must necessarily be real, and the linear term in the objective function in $(D)$ is replaced by $\langle b,\, y^R\rangle$. Since the skew symmetric matrices in (7) do not affect the positiveness condition and $y^I$ does not appear in the objective function in $(D)$, we can take $y_I^k = 0$, $k = 1, \dots, m$.

Note that the conversion of a complex SDP into a real SDP based on (7) would double the storage and if the data is dense, the cost of each interior-point iteration for solving the resulting real SDP is about twice as expensive as that for solving the complex SDP directly.

To convert an SDP with complex data into one with only real data, the m-file `convertcmpsdp.m` has the calling syntax:

```
[bblk,AAt,CC,bb] = convertcmpsdp(blk,At,C,b);
```

where `AAt` corresponds to the $m$ real symmetric constraint matrices in the first summation in (7), `CC` corresponds to the real constant matrix in (7), and `bb` $= b^R$.

Internally, SDPT3 would automatically detect the presence of complex data in an SDP and convert it into one with only real data before solving it. But note that the current version of SDPT3 does not allow complex data corresponding to the quadratic (second-order cone) block.

**Rotated cones.**

Let $K_r^n$ $(n \geq 3)$ be the rotated cone defined by

$$K_r^n \;=\; \{x^r = [u; v; w] \in I\!R^n \;:\; \|w\|^2 \leq 2uv,\; u, v \geq 0\}.$$

Note the constant "2" above. Define the symmetric orthogonal matrix $T_n \in I\!R^{n\times n}$ as follows:

$$T_n = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & \\ 1/\sqrt{2} & -1/\sqrt{2} & \\ & & I_{n-2} \end{bmatrix}.$$

It is easy to show that $x^r \in K_r^n$ if and only if $x^q := T_n x^r \in K_q^n$, i.e., $T_n K_r^n = K_q^n$. Thus we can always convert a rotated cone variable into one belonging to a second-order cone.

In SDPT3-4.0, the user can code a rotated cone block consisting of several subblocks, say $p$ of them of dimension $\mathtt{r_{i1}}$, ....,$\mathtt{r_{ip}}$, as follows:

```
blk{i,1} = 'r'; blk{i,2} = [r_i1, ....,r_ip];
```

Let $D$ be the block diagonal matrix defined by $D = \text{diag}\,(T_{r_{i1}}, ..., T_{r_{ip}})$. Internally, SDPT3 would convert such a rotated cone block and its associated data into a second-order cone block as follows:

```
blk{i,1} = 'q'; blk{i,2} = [ri1, ..., rip];
At{i,1} = D*At{i,1};
C{i,1} = D*C{i,1};
```

## 4 Examples

For an user to solve his SQLP problem using SDPT3, the first task he/she needs to perform is to write his problem in the standard form given in $(P)$ and then to code the problem data corresponding to the standard form. In the last few years, modelling languages such as CVX [8] and YALMIP [21] have been developed for advanced modelling and solution of convex optimization problems. These modelling languages would automatically convert an SQLP problem which is not necessarily expressed in the standard form given in $(P)$ or $(D)$ into the required standard form for solvers such as SDPT3 or SeDuMi. They would also generate the problem data corresponding to the standard form and call a solver to solve the problem. In CVX, SDPT3 is used as the default solver, but SeDuMi is also conveniently available. Similarly, YALMIP can also use SDPT3 or SeDuMi as its computational engine to solve SQLP problems.

For users who are interested only in solving small SQLP problems, we would advise them to use CVX or YALMIP to convert their problems into the required standard form and to generate the corresponding data as the process is usually quite tedious though mechanical. But for users who wish to solve large SQLP problems, it is sometimes highly beneficial for the users themselves to convert their problems into the required standard form rather than relying on CVX or YALMIP as these modelling languages may introduce large number of additional variables or significantly expand the problem dimensions when converting the problems into the standard form since they are based on automatic conversion procedures which do not take into account problem structures that users may be able to exploit.

The simplest way to learn how to convert an SQLP problem into the standard form given in $(P)$ or $(D)$ and to generate the corresponding problem data in SDPT3 format is through examples. (Note that the user can also store the problem data in either the SDPA or SeDuMi format, and then use the m-files `read_sdpa.m` or `read_sedumi.m` to read the data into SDPT3.) The subdirectory `Examples` in SDPT3 contains many such example files. Here we will just mention a few.

### 4.1 Conversion of problems into the standard form

As mentioned before, SQLP problems are usually not formulated in the standard form $(P)$ or $(D)$, and it is often quite tedious to convert such problems into the standard form. As such, users who are solving small to medium scale SQLPs are encouraged to use modelling languages such as CVX [8] or YALMIP [21] to automate the conversion process. To give the user an idea on how the conversion is done, here we shall just give an example to show how an SDP with linear equality and inequality constraints can be converted into the standard form given in $(P)$. Suppose we have an SDP of the following form:

$$
\begin{array}{llrcl}
(P_1) & \min & \langle c^s,\, x^s\rangle & & \\
& \text{s.t.} & \mathcal{A}^s(x^s) & = & b, \\
& & \mathcal{B}^s(x^s) & \leq & d, \\
& & x^s \in K_s^n. & &
\end{array}
$$

where $d \in I\!\!R^p$ and $\mathcal{B}^s$ is a linear map. By introducing a slack variable $x^l$, we can easily convert $(P_1)$ into standard form, namely,

$$
\begin{array}{llcclcl}
(P_1^*) & \min & \langle c^s,\, x^s\rangle & + & \langle c^l,\, x^l\rangle & & \\
& \text{s.t.} & \mathcal{A}^s(x^s) & & & = & b, \\
& & \mathcal{B}^s(x^s) & + & B^l x^l & = & d, \\
& & x^s \in K_s^n. & & x^l \in K_l^p, & &
\end{array}
$$

where $c^l = 0$, and $B^l = I_{p\times p}$. With our use of cell arrays to store SQLP data, it is easy to take the problem data of $(P_1)$ and use them for the standard form $(P_1^*)$ as follows:

```
blk{1,1} = 's';                                  blk{1,2} = n;
At{1,1} = [A^s; B^s]^T;                          C{1,1} = c^s;
blk{2,1} = 'l';                                  blk{2,2} = p;
At{2,1} = [sparse(m,p); speye(p,p)]^T;           C{2,1} = c^l;
b = [b; d];
```

In the subdirectory, an SDP of the form $(P_1)$ is coded in the example file `max_kcut.m`.

## 4.2 The MAXCUT problem

Let $\mathcal{S}_+^n$ be the space of $n \times n$ symmetric positive semidefinite matrices. Let $B$ be the weighted adjacency matrix of a graph. The SDP relaxation of the MAXCUT problem associated with $B$ has the following form:

$$
\begin{array}{ll}
\min & \langle C,\, X\rangle \\
\text{s.t.} & \operatorname{diag}(X) = e, \quad X \in \mathcal{S}_+^n,
\end{array}
$$

where $e$ is the vector of ones, and $C = -(\operatorname{Diag}(Be) - B)/4$. It is clear that we need the cell array, `blk{1,1}='s', blk{1,2}=n`, to record the block structure of the problem. The constraint matrices can be constructed conveniently via an $1 \times n$ cell array as follows:

```
AA = cell(1,n);
for k=1:n; AA{k}=spconvert([k,k,1;n,n,0]); end
At = svec(blk,AA);
```

For more details, see the m-file `maxcut.m` in the subdirectory `Examples`. (We could also create a version of the problem explicitly showing the low-rank structure; however, as the constraint matrices are so sparse, this would not be more efficient.)

## 4.3 D-optimal experiment design - an example with an explicit barrier term

Given a set of points $\{v_1, \ldots, v_p\}$ in $I\!R^n$ with $n \le p$, the D-optimal experiment design problem [40] needs to solve the following dual SQLP:

$$\begin{array}{llcccl}
\max & \log\det(Z) & & & & \\
\text{s.t.} & \sum_{k=1}^{p} y_k(-v_k v_k^T) & + & Z & = & 0, \quad Z \in \mathcal{S}^n_{++} \\
 & -y & + & z^l & = & 0, \quad z^l \in I\!R^p_+ \\
 & e^T y & & & = & 1, \quad y \in I\!R^p.
\end{array}$$

The associated problem data can be coded in SDPT3 format as follows:

```
b = zeros(p,1);
blk{1,1} = 's'; blk{1,2} = n;
AA = cell(1,p); for k=1:p; AA{k} = -vk*vk'; end
At(1) = svec(blk(1,:),AA);  C{1,1} = sparse(n,n);
blk{2,1} = 'l'; blk{2,2} = p;
At{2,1}  = -speye(p);       C{2,1} = zeros(p,1);
blk{3,1} = 'u'; blk{3,2} = 1;
At{3,1}  = ones(1,p);       C{3,1} = 1;
```

Because the problem contains an explicit log-barrier term in the objective function, we also need to set up `OPTIONS.parbarrier` as follows:

```
 OPTIONS.parbarrier{1,1} = 1;
 OPTIONS.parbarrier{2,1} = 0;
 OPTIONS.parbarrier{3,1} = 0;
```

For more details, see the m-file `Doptdesign.m` in the subdirectory `Examples`.

The constraint matrices corresponding to the semidefinite block in this example are all rank-one matrices. The user can explicitly code such structures for SDPT3 as follows:

```
blk{1,1} = 's'; blk{1,2} = n; blk{1,3} = ones(1,p);
At{1,1} = []; At{1,2} = [v1,...,vp]; At{1,3} = -ones(p,1);
```

As an example on the potential speed up one may gain by exploiting the low-rank structure in the constraint matrices, we compare the times taken to solve two D-optimal experiment design problems with randomly generated data $\{v_1, \dots, v_p\}$ in $I\!R^n$. For $n = 200$, $p = 400$ ($n = 100$, $p = 400$), the time taken to solve the problem without exploiting the low-rank structure is 157.1 (44.2) seconds, compared to just 2.9 (2.1) seconds when the low-rank structure is exploited.

## 4.4 An LMI example

Consider the following LMI problem [3]:

$$\begin{array}{lll} \max & -\eta & \\ \text{s.t.} & GY + YG^T & \preceq \; 0 \\ & -Y & \preceq \; -I \\ & Y - \eta I & \preceq \; 0 \\ & Y_{11} & = \; 1, \quad Y \in \mathcal{S}^n, \end{array} \tag{8}$$

where $G \in I\!R^{n \times n}$. This problem can be viewed as a dual SDP with $Y$ identified as a vector $y$ in $I\!R^{n(n+1)/2}$. In this case, we have $(A_1^s)^T y = \mathbf{svec}(G\mathbf{smat}(y) + \mathbf{smat}(y)G^T)$, where **smat** is the inverse of **svec**. The SDP data can be generated for SDPT3 as follows:

```
blk{1,1} = 's'; blk{1,2} = n;    blk{2,1} = 's'; blk{2,2} = n;
blk{3,1} = 's'; blk{3,2} = n;    blk{4,1} = 'u'; blk{4,2} = 1;
n2 = n*(n+1)/2; zz = sparse(n2,1); I = speye(n);
At{1,1} = [lmifun(G,I), zz];
At{2,1} = [-lmifun(I/2,I), zz];
At{3,1} = [lmifun(I/2,I), svec(blk(1,:),-I)];
At{4,1} = [1, zz'];
C{1,1} = sparse(n,n); C{2,1} = -I; C{3,1} = sparse(n,n); C{4,1} = 1;
b = [zz; -1];
```

In the above, `lmifun(G,H)` is a function (available in `Examples`) that generates the matrix representation of the linear map $y \in I\!R^{n(n+1)/2} \mapsto \mathbf{svec}(G\mathbf{smat}(y)H^T + H\mathbf{smat}(y)G^T)$.

For more details, see the m-file `lmiexamp1.m` in the subdirectory `Examples`.

## 4.5 The nearest correlation matrix problem

Given an $n \times n$ symmetric matrix $H$, the nearest correlation matrix problem is the following [18]:

$$\min_X \{ \|H - X\|_F \; : \; \mathrm{diag}\,(X) = e, \; X \in \mathcal{S}_+^n \}.$$

The above problem can be converted to the following SQLP:

$$\min\{ e_1^T y \; : \; \mathrm{diag}\,(X) = e, \; \mathbf{svec}(X) + [0, I_{n_2}]y = \mathbf{svec}(H), \; X \in \mathcal{S}_+^n, \; y \in Q^{n_2+1} \},$$

where $n_2 = n(n+1)/2$, $I_{n_2}$ denotes the identity matrix of order $n_2$, and $Q^{n_2+1}$ denotes the second-order cone of dimension $n_2 + 1$. Also, $e_1$ denotes the first unit vector in $(n_2+1)$-space; in general, $e_i$ will denote the $i$th unit vector of appropriate dimension. The corresponding SQLP data can be coded for SDPT3 as follows:

```
blk{1,1} = 's'; blk{1,2} = n; n2 = n*(n+1)/2;
for k=1:n; AA{1,k} = spconvert([k,k,1; n,n,0]); end;
matrepdiag = svec(blk(1,:),AA);
At{1,1} = [matrepdiag{1}, speye(n2)];
blk{2,1} = 'q'; blk{2,2} = n2+1;
At{2,1} = [sparse(n,n2+1); sparse(n2,1), speye(n2)];
b = [ones(n,1); svec(blk(1,:),H)];
C{1,1} = sparse(n,n); C{2,1} = [1; zeros(n2,1)];
```

For more details, see the m-file `corrmat.m` in the subdirectory `Examples`.

## 4.6 A problem with a convex quadratic objective function

The previous example has a convex quadratic objective function in the primal problem. Here we consider the case where the dual problem has a concave quadratic objective function, namely,

$$\max_{y,Z}\{b^T y - y^T H y \; : \; \mathcal{A}_s^T y + z_s = c_s,\ z_s \in \mathcal{S}_+^n,\ y \in I\!R^m\},$$

where $H$ is a given symmetric positive semidefinite matrix. Suppose $H = R^T R$ is the Cholesky factorization. Then the above problem can be rewritten as follows:

$$\max_{y,Z}\{b^T y - t \; : \; \|Ry\|^2 \le t,\ \mathcal{A}_s^T y + z_s = c_s,\ z_s \in \mathcal{S}_+^n,\ y \in I\!R^m\}. \qquad (9)$$

It can in turn be expressed as a standard SQLP in dual form as follows:

$$\begin{array}{ll} \max & [-1\,;\,b]^T[t\,;\,y] \\ \text{s.t.} & \begin{bmatrix} -1/2 & 0 \\ -1/2 & 0 \\ 0 & R \end{bmatrix}[t;y] + z_q = \begin{bmatrix} 1/2 \\ -1/2 \\ 0 \end{bmatrix} \\ & [0\,,\,\mathcal{A}_s^T][t\,;\,y] + z_s = c_s, \\ & z_q \in K_q^{m+1},\ z_s \in \mathcal{S}_+^n,\ y \in I\!R^m. \end{array}$$

## 4.7 An example from distance geometry

Consider a graph $G = (V, \mathcal{E}, D)$ where $V = \{1, 2, \ldots, n\}$, $\mathcal{E}$, and $D = (d_{ij})$ denote the nodes, edges, and associated weight matrix on the edges, respectively. The problem is to find points $x_1, \ldots, x_n$ in $I\!R^p$ (for some $p$) such that the pairwise Euclidean distance between $x_i$ and $x_j$ is as close as possible to $d_{ij}$ if $(i,j) \in \mathcal{E}$. The associated minimization problem is the following:

$$\min\Big\{ \sum_{(i,j)\in\mathcal{E}} \Big| \|x_i - x_j\|^2 - d_{ij}^2 \Big| \;:\; X := [x_1,\dots,x_n] \in I\!R^{p\times n} \Big\}.$$

By noting that $\|x_i - x_j\|^2 = e_{ij}^T X^T X e_{ij}$, where $e_{ij} = e_i - e_j$, the above problem can equivalently be written as $\min\{\sum_{(i,j)\in\mathcal{E}} |\langle e_{ij}e_{ij}^T, Y\rangle - d_{ij}^2| : Y = X^T X,\ X \in I\!R^{p\times n}\}$. One of the SDP relaxations of the above problem is $\min\{\sum_{(i,j)\in\mathcal{E}} |\langle e_{ij}e_{ij}^T, Y\rangle - d_{ij}^2| : Y \in \mathcal{S}_+^n\}$, where the corresponding problem in standard form is given by

$$\min\Big\{ \sum_{(i,j)\in\mathcal{E}} \alpha_{ij}^+ + \alpha_{ij}^- \;:\; \langle e_{ij}e_{ij}^T, Y\rangle - \alpha_{ij}^+ + \alpha_{ij}^- = d_{ij}^2,\ \alpha_{ij}^+, \alpha_{ij}^- \geq 0\ \forall\,(i,j)\in\mathcal{E},\ Y \in \mathcal{S}_+^n \Big\}.$$

Let $m = |\mathcal{E}|$. The SQLP data can be coded as follows:

```
blk{1,1} = 's';  blk{1,2} = n;
AA = cell(1,m); b = zeros(m,1); cnt = 0;
for i = 1:n-1
   for j = i+1:n
      if (D(i,j) ~= 0)
         cnt = cnt + 1;
         AA{cnt} = spconvert([i,i,1; i,j,-1; j,i,-1; j,j,1; n,n,0]);
         b(cnt) = D(i,j)^2;
      end
   end
end
At(1) = svec(blk(1,:),AA); C{1,1} = sparse(n,n);
blk{2,1} = 'l';  blk{2,2} = 2*m;
At{2,1} = [-speye(m), speye(m)]; C{2,1} = ones(2*m,1);
```

## 4.8 Norm minimization problem with complex data

Let $B_0, \dots B_m$ be $p \times q$ matrices that are possibly complex. Consider the norm minimization problem:

$$\min\{t \;:\; \|\sum_{k=1}^m x_k B_k + B_0\|_2 \leq t,\ x \in \mathbb{C}^m,\ t \in I\!R\},$$

where $\|\cdot\|_2$ denotes the matrix 2-norm. The problem above can equivalently be written as follows:

$$\begin{array}{ll} \max & -t \\ \text{s.t.} & \displaystyle\sum_{k=1}^m x_k^R \begin{bmatrix} 0 & B_k \\ B_k^* & 0 \end{bmatrix} + \sum_{k=1}^m x_k^I \begin{bmatrix} 0 & iB_k \\ (iB_k)^* & 0 \end{bmatrix} - tI \preceq - \begin{bmatrix} 0 & B_0 \\ B_0^* & 0 \end{bmatrix}. \end{array}$$

This is a complex SDP written in the format as in $(D)$. Such a problem can be solve by SDPT3 as follows:

```
blk{1,1} = 's'; blk{1,2} = p+q;
AA = cell(1,2*m+1);
for k = 1:m; AA{1,k} = [zeros(p), Bk; B'k, zeros(q)]; end
for k = 1:m; AA{1,m+k} = [zeros(p), i*Bk; -i*B'k, zeros(q)]; end
AA{1,2*m+1} = -speye(p+q);
C{1} = -[zeros(p), B0; B'0, zeros(q)]; b = [zeros(2*m,1); -1];
[bblk,AAt,CC,bb] = convertcmpsdp(blk,AA,C,b);
[obj,X,y,Z] = sqlp(bblk,AAt,CC,bb);
x = y(1:m) + i*y(m+[1:m]); t = y(2*m+1);
```

For more details, see the m-file `norm_min.m` in the subdirectory `Examples`.

## 4.9 Logarithmic Chebyshev approximation problem

This is an example that contains variables in a rotated cone. The orginal problem [22] is given by $\min_{x\in\mathbb{R}^m} \max\{|\log(f_i^T x) - \log(d_i)| : i = 1,\dots,p\}$, which can equivalently be formulated as: $\min\{s \,:\, t \le f_i^T x/d_i \le s,\ i = 1,\dots,p,\ 1 \le st\}$. Let $F = [f_1^T;\dots;f_p^T]$ and $d = [d_1;\dots;d_p]$. The latter problem can be formulated in the following standard dual form:

$$\begin{array}{ll}
\max & -s \\
\text{s.t.} & \Big[F,\ -d,\ 0,\ 0\Big][x;s;t;u] \le 0 \\
 & \Big[-F,\ 0,\ d,\ 0\Big][x;s;t;u] \le 0 \\
 & -\Big[0,\ -I_3\Big][x;s;t;u] \in K_r^3 \\
 & \Big[0,\ 0,\ 0,\ 1\Big][x;s;t;u] = \sqrt{2}.
\end{array}$$

The corresponding data for SDPT3 can be coded as follows:

```
blk{1,1} = 'l'; blk{1,2} = p;
blk{2,1} = 'l'; blk{2,2} = p;
blk{3,1} = 'r'; blk{3,2} = 3;
blk{4,1} = 'u'; blk{4,2} = 1;
zz = sparse(p,1);
At{1,1} = [F, -d, zz, zz];           C{1,1} = zeros(p,1);
At{2,1} = [-F, zz, d, zz];           C{2,1} = zeros(p,1);
At{3,1} = [sparse(3,m), -speye(3)]; C{3,1} = zeros(3,1);
At{4,1} = [sparse(1,m+2), 1];        C{4,1} = sqrt(2);
b = [zeros(m,1); -1; 0; 0];
```

For more details, see the m-file `logchebyRcone.m` in the subdirectory `Examples`.

## 4.10 Maximizing the geometric mean of affine functions

Another example with rotated cone variables comes from maximizing the geometric mean of nonnegative affine functions [22]:

$$\begin{array}{rl} \max_{x\in\boldsymbol{R}^m} & \Pi_{i=1}^4 (a_i^T x + b_i)^{1/4} \\ \text{s.t.} & a_i^T x + b_i \geq 0,\ i = 1,\ldots,4. \end{array}$$

The above can be reformulated as the following SQLP problem with rotated cone constraints:

$$\max \left\{ \begin{array}{lll} t_3 \ :\ y_i \ = \ a_i^T x + b_i,\ y_i \geq 0,\ i = 1,\ldots,4, & (\sqrt{2}t_1)^2 \leq 2y_1y_2,\ (\sqrt{2}t_2)^2 \leq 2y_3y_4, \\ & (\sqrt{2}t_3)^2 \leq 2t_1t_2 \end{array} \right\}.$$

The corresponding standard (dual) form is as follows:

$$\begin{array}{ll} \max & t_3 \\ \text{s.t.} & \begin{bmatrix} a_1^T & 0 & 0 & 0 \\ a_2^T & 0 & 0 & 0 \\ 0 & \sqrt{2} & 0 & 0 \\ a_3^T & 0 & 0 & 0 \\ a_4^T & 0 & 0 & 0 \\ 0 & 0 & \sqrt{2} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \sqrt{2} \\ a_1^T & 0 & 0 & 0 \\ a_3^T & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ t_1 \\ t_2 \\ t_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ 0 \\ b_3 \\ b_4 \\ 0 \\ 0 \\ 0 \\ 0 \\ b_1 \\ b_3 \end{bmatrix} \in K_r^3 \times K_r^3 \times K_r^3 \times I\!R_+^2. \end{array}$$

# 5 Implementation details

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 some invertible block diagonal scaling matrix that is usually chosen as a function of the current iterate $x, z$.

## 5.1 The HKM search direction

Let

$$J_i^q \ = \ \begin{bmatrix} 1 & 0 \\ 0 & -I_{q_i-1} \end{bmatrix}. \tag{10}$$

For the choice of the HKM scaling matrix [16, 20, 24, 25, 38], the search direction $(\Delta x, \Delta y, \Delta z)$ is obtained from the following system of equations:

$$\begin{aligned}
\mathcal{A}^s(\Delta x^s) + A^q(\Delta x^q) + A^l \Delta x^l + A^u \Delta x^u &= R_p := b - \mathcal{A}^s(x^s) - A^q(x^q) - A^l x^l - A^u x^u \\
(\mathcal{A}^s)^T \Delta y + \Delta z^s &= R_d^s := c^s - z^s - (\mathcal{A}^s)^T y \\
(A^q)^T \Delta y + \Delta z^q &= R_d^q := c^q - z^q - (A^q)^T y \\
(A^l)^T \Delta y + \Delta z^l &= R_d^l := c^l - z^l - (A^l)^T y \\
(A^u)^T \Delta y &= R_d^u := c^u - (A^u)^T y \\
\Delta x^s + H^s(\Delta z^s) &= R_c^s := \Big( \max\{\sigma\mu(x,z), \nu_j^s\}(z_j^s)^{-1} - x_j^s \Big)_{j=1}^{n_s} \qquad (11) \\
\Delta x^q + H^q(\Delta z^q) &= R_c^q := \Big( \max\{\sigma\mu(x,z), \nu_i^q\}(z_i^q)^{-1} - x_i^q \Big)_{i=1}^{n_q} \\
\Delta x^l + H^l(\Delta z^l) &= R_c^l := \Big( \max\{\sigma\mu(x,z), \nu_k^l\}(z_k^l)^{-1} - x_k^l \Big)_{k=1}^{n_l},
\end{aligned}$$

where $\sigma \in (0,1)$ is the centering parameter; $(z_j^s)^{-1}$ and $(z_k^l)^{-1}$ have the usual meaning, and $(z_i^q)^{-1} := J_i^q z_i^q / \gamma(z_i^q)^2$. In the above,

$$H^s(\Delta z^s) = \Big( H_j^s(\Delta z_j^s) := \tfrac{1}{2}\big((z_j^s)^{-1} \Delta z_j^s \, x_j^s + x_j^s \Delta z_j^s (z_j^s)^{-1}\big) \Big)_{j=1}^{n_s},$$

$$H^q(\Delta z^q) = \Big( H_i^q(\Delta z_i^q) := -\frac{\langle x_i^q, \, z_i^q \rangle}{\gamma(z_i^q)^2} J_i^q \Delta z_i^q + x_i^q ((z_i^q)^{-1})^T \Delta z_i^q + (z_i^q)^{-1} (x_i^q)^T \Delta z_i^q \Big)_{i=1}^{n_q}, \qquad (12)$$

$$H^l(\Delta z^l) = \mathrm{Diag}\,(x^l) \mathrm{Diag}\,(z^l)^{-1} \Delta z^l.$$

The search direction can be computed via a Schur complement equation as follows (the reader is referred to [1] and [30] for details). First compute $\Delta y$ from the Schur complement equation

$$\underbrace{\begin{bmatrix} M & A^u \\ (A^u)^T & 0 \end{bmatrix}}_{\mathcal{M}} \begin{bmatrix} \Delta y \\ \Delta x^u \end{bmatrix} = \begin{bmatrix} h \\ R_d^u \end{bmatrix} \qquad (13)$$

where

$$\begin{aligned}
M &= \sum_{j=1}^{n_s} \underbrace{\mathcal{A}_j^s H_j^s (\mathcal{A}_j^s)^T}_{M_j^s} + \sum_{i=1}^{n_q} \underbrace{A_i^q H_i^q (A_i^q)^T}_{M_i^q} + \underbrace{A^l H^l (A^l)^T}_{M^l} \qquad (14) \\
h &= R_p + \mathcal{A}^s\Big(H^s(R_d^s) - R_c^s\Big) + A^q\Big(H^q(R_d^q) - R_c^q\Big) + A^l\Big(H^l(R_d^l) - R_c^l\Big).
\end{aligned}$$

(The notation in (14) should be interpreted as follows: the kth columns of $M_j^s$ and $M_i^q$ are $\mathcal{A}_j^s H_j^s(a_{j,k}^s)$ and $A_i^q H_i^q(a_{i,k}^q)$, with $a_{i,k}^q$ the kth column of $(A_{i,k}^q)^T$. Note that the matrices $M_j^s$, $M_i^q$, $M^l$ are all symmetric positive definite.) Then compute $\Delta x$ and $\Delta z$ from the equations

$$\begin{aligned} \Delta z^s = R_d^s - (\mathcal{A}^s)^T \Delta y, \quad \Delta z^q = R_d^q - (A^q)^T \Delta y, \quad \Delta z^l = R_d^l - (A^l)^T \Delta y \\ \Delta x^s = R_c^s - H^s(\Delta z^s), \quad \Delta x^q = R_c^q - H^q(\Delta z^q), \quad \Delta x^l = R_c^l - H^l(\Delta z^l). \end{aligned}$$

## 5.2 The NT search direction

The user also has the choice of using the NT direction [28, 25, 38]. Let $w_j^s$ be the unique positive definite matrix such that $w_j^s z_j^s w_j^s = x_j^s$. Let

$$\omega_i^q = \sqrt{\frac{\gamma(z_i^q)}{\gamma(x_i^q)}}, \quad \xi_i^q = \frac{1}{\omega_i^q} z_i^q + \omega_i^q J_i^q x_i^q, \quad t_i^q = \frac{\sqrt{2}}{\omega_i^q \gamma(\xi_i^q)} J_i^q \xi_i^q. \tag{15}$$

In this case, the search direction $(\Delta x, \Delta y, \Delta z)$ satisfies the same system as in (11) except that $H^s$ and $H^q$ are replaced by

$$\begin{aligned} H^s(\Delta z^s) &= \Big(H_j^s(\Delta z_j^s) := w_j^s \Delta z_j^s \, w_j^s\Big)_{j=1}^{n_s}, \\ H^q(\Delta z^q) &= \Big(H_i^q(\Delta z_i^q) := -\frac{1}{(\omega_i^q)^2} J_i^q \Delta z_i^q + t_i^q (t_i^q)^T \Delta z_i^q\Big)_{i=1}^{n_q}. \end{aligned} \tag{16}$$

## 5.3 Choice of search direction

The current version of the code allows only two search directions, HKM and NT. In older versions, version 2.3 or earlier, we also implemented the AHO direction of Alizadeh, Haeberly, and Overton [1] and the GT direction [32], where both directions tend to give more accurate results, but these are uncompetitive when the problems are of large scale.

For the HKM and NT search directions, our computational experience on problems tested in Section 8 is that the HKM direction is almost universally faster than NT on problems with semidefinite blocks, especially for sparse problems with large semidefinite blocks. The reason that the latter is slower is because computing the NT scaling matrix $w_j^s$ requires a full eigenvalue decomposition. This computation can dominate the work at each interior-point iteration when the problem is sparse.

The NT direction, however, was faster on sparse SOCP problems. The reason for this behavior is not hard to understand. By comparing the formulae for $H_i^q$ for the HKM and NT directions in (12) and (16), it is clear that more computation is required to assemble the Schur complement matrix and more low-rank updating is necessary for the former direction.

## 5.4 Computation of the Schur complement matrix

Generally, the most expensive part in each iteration of Algorithm IPC lies in the computation and factorization of the Schur complement matrix $M$ defined in (13).

And this depends critically on the size and density of $M$. Note that the density of this matrix depends on two factors: (i) The density of $\mathcal{A}^s$, $A^q$, and $A^l$, and (ii) any additional fill-in introduced because of the terms $H^s$, $H^q$, and $H^l$ in (14).

### 5.4.1 Semidefinite blocks

For problems with semidefinite blocks, the contribution by the $j$th semidefinite block to $M$ is given by $M_j^s := \mathcal{A}_j^s H_j^s (\mathcal{A}_j^s)^T$. The matrix $M_j^s$ is generally dense even if $A_j^s$ is sparse. The computation of each entry of $M_j^s$ involves matrix products, which has the form

$$(M_j^s)_{\alpha\beta} \;=\; \begin{cases} \langle a_{j,\alpha}^s,\, x_j^s\, a_{j,\beta}^s\, (z_j^s)^{-1}\rangle & \text{for the HKM direction.} \\ \langle a_{j,\alpha}^s,\, w_j^s\, a_{j,\beta}^s\, w_j^s\rangle & \text{for the NT direction.} \end{cases}$$

This computation can be very expensive if it is done naively without properly exploiting the sparsity that is generally present in the constraint matrices in $\mathcal{A}_j^s$. In our earlier papers [30, 34], we discussed briefly how sparsity of $\mathcal{A}_j^s$ is exploited in our implementation by following the ideas presented by Fujisawa, Kojima, and Nakata in [13]. Further details on the efficient implementation of these ideas are given in [39].

When the constraint matrices have low-rank structures as described in Section 2.2, we can also compute the element $(M_j^s)_{\alpha,\beta}$ as follows:

$$(M_j^s)_{\alpha\beta} \;=\; \mathrm{Tr}(\widetilde{V}_\beta^T V_\alpha D_\alpha V_\alpha^T \widehat{V}_\beta D_\beta),$$

where $\widetilde{V}_\beta = x_j^s V_\beta$, and $\widehat{V}_\beta = (z_j^s)^{-1} V_\beta$ if the HKM direction is used; and $\widetilde{V}_\beta = \widehat{V}_\beta = w_j^s V_\beta$ if the NT direction is used. Assume for simplicity that all the constraint matrices associated with the $j$th semidefinite block are dense and low-rank, i.e., $p = 0$ in Section 2.2. Suppose that the matrices $\widetilde{V}_k$, $\widehat{V}_k$, $k = 1, \ldots, m$, are pre-computed (at the cost of $\Theta(s_j^2 \sum_{k=1}^m r_{j,k})$ flops). Then it would take an additional $\Theta(s_j(\sum_{k=1}^m r_{j,k})^2)$ flops to compute $M_j^s$ since each element $(M_j^s)_{\alpha\beta}$ can be computed at $\Theta(s_j r_{j,\alpha} r_{j,\beta})$ flops. In contrast, without exploiting the low-rank structures, it would take $\Theta(s_j^3 m) + \Theta(s_j^2 m^2)$ flops to compute $M_j^s$. If the average rank of the constraint matrices is $r$, then the latter complexity is $\Theta(s_j/r^2)$ times larger than the former of $\Theta(s_j^2 m r) + \Theta(s_j m^2 r^2)$. Thus it is definitely advantageous to exploit low-rank structures.

As an example, we generated a random SDP with low rank structure using the m-file `randlowranksdp.m` described in Section 2.2 with $n = 200$ and $m = 1000$; the solver `sqlp.m` ran about 6 times faster when the low-rank structure was exploited.

### 5.4.2 Quadratic and linear blocks

For the linear block, $H^l$ is a diagonal matrix and it does not introduce any additional fill-in. This matrix does, however, affect the conditioning of the Schur complement matrix.

From equation (14), it is easily shown that the contribution of the quadratic blocks to the matrix $M$ is given by

$$M_i^q \;=\; \begin{cases} -\frac{\langle x_i^q, z_i^q\rangle}{\gamma^2(z_i^q)} A_i^q J_i^q (A_i^q)^T \;+\; u_i^q (v_i^q)^T + v_i^q (u_i^q)^T, & \text{for HKM direction} \\ -\frac{1}{(\omega_i^q)^2} A_i^q J_i^q (A_i^q)^T + w_i^q (w_i^q)^T & \text{for NT direction} \end{cases} \tag{17}$$

where $u_i^q = A_i^q x_i^q$, $v_i^q = A_i^q (z_i^q)^{-1}$, $w_i^q = A_i^q t_i^q$.

In the rest of this subsection, we focus our discussion on the HKM direction, but the same holds true for the NT direction.

The appearance of the outer-product terms in the equation above is potentially alarming. If the vectors $u_i^q$, $v_i^q$ are dense, then even if $A_i^q$ is sparse, the corresponding matrix $M_i^q$, and hence the Schur complement matrix $M$, will be dense. A direct factorization of the resulting dense matrix will be very expensive for even moderately large $m$.

The density of the matrix $M_i^q$ depends largely on the particular problem structure. When the problem has many small quadratic blocks, it is often the case that each block appears in only a small fraction of the constraints. In this case, all $A_i^q$ matrices are sparse and the vectors $u_i^q$ and $v_i^q$ turn out to be sparse vectors for each $i$. Consequently, the matrices $M_i^q$ remain relatively sparse for these problems. As a result, $M$ is also sparse and it can be factorized directly with reasonable cost. This behavior is typical for all `nql` and `qssp` problems from the DIMACS library.

The situation is drastically different for problems where one of the quadratic blocks, say the $i$th block, is large. For such problems the vectors $u_i^q$, $v_i^q$ are typically dense, and therefore, $M_i^q$ is likely be a dense matrix even if the data $A_i^q$ is sparse. However, observe that $M_i^q$ is a rank-two perturbation of a sparse matrix when $A_i^q (A_i^q)^T$ is sparse. In such a situation, it is advantageous to use the dense-column handling technique described in Section 5.7 to reduce the computational cost in solving (13). This approach helps tremendously on the scheduling problems from the DIMACS library.

To apply the dense-column handling technique, we need to modify the sparse portion of the matrix $M_i^q$ slightly. Since the diagonal matrix $-J_i$ has a negative component, the matrix $-A_i^q J_i^q (A_i^q)^T$ need not be a positive definite matrix, and therefore the Cholesky factorization of the sparse portion of $M_i^q$ need not exist. To overcome this difficulty, we use the following identity:

$$M_i^q \;=\; \frac{\langle x_i^q,\, z_i^q\rangle}{\gamma^2(z_i^q)} A_i^q (A_i^q)^T \;+\; u_i^q (v_i^q)^T + v_i^q (u_i^q)^T - k_i k_i^T, \tag{18}$$

where $k_i = \sqrt{2}\langle x_i^q,\, z_i^q\rangle / \gamma^2(z_i^q)\, A_i^q(:,1)$. Note that if $A_i^q$ is a large sparse matrix with a few dense columns, we can also explicitly separate the outer-product terms contributed by these dense columns from the sparse part of $A_i^q (A_i^q)^T$ in (18).

## 5.5 Solving the Schur complement equation

The linear system (13) typically becomes more and more ill-conditioned as $\mu(x, z)$ decreases to 0. Thus iterative refinement is generally recommended to improve the accuracy of the computed solution. An even better approach to solve (13) is via a preconditioned symmetric quasi-minimal residual method (PSQMR) [11] with the preconditioner computed based on the following analytical expression of $\mathcal{M}^{-1}$:

$$\mathcal{M}^{-1} = \begin{bmatrix} M^{-1} - M^{-1}A^u S^{-1}(A^u)^T M^{-1} & M^{-1}A^u S^{-1} \\ S^{-1}(A^u)^T M^{-1} & -S^{-1} \end{bmatrix}, \tag{19}$$

where $S = (A^u)^T M^{-1} A^u$. Note that for a given vector $[u; v]$, $\mathcal{M}^{-1}[u; v]$ can be evaluated efficiently as follows:

$$\begin{aligned} \hat{u} &= M^{-1}u \\ t &= S^{-1}\Big((A^u)^T \hat{u} - v\Big) \\ \mathcal{M}^{-1}[u; v] &= [\hat{u} - M^{-1}A^u\, t;\; t]. \end{aligned}$$

Thus if the Cholesky factorization of $M$ and that of $S$ are computed, then each evaluation involves solving four triangular linear systems for $M$ and two triangular linear systems for $S$.

We should mention that state-of-the-art Cholesky factorization software is highly developed and optimized. Thus our preference is to solve a linear system via Cholesky factorizations whenever possible. For most SDP problems, the matrix $M$ is typically dense even when the constraint matrices are sparse. In this case, we use the routine `chol` (based on the LAPACK routine `dpotrf`) in Matlab to compute the Cholesky factorization of a dense matrix.

For most sparse SOCP problems, the matrix $M$ is usually sparse after dense-column handling. Let $M_{sp}$ be the sparse part of $M$ after dense-column handling. In this case, the Cholesky factorization routine `chol` for a dense matrix is not efficient enough since it does not exploit sparsity. To factorize the sparse matrix $M_{sp}$ more efficiently, we use the sparse cholesky solver `cholmod` of Davis [5], which is available in Matlab as `chol`. In earlier versions, we used a C translation of the Fortran programs developed by Ng, Peyton, and Liu for sparse Cholesky factorization [26].

The effect of using a sparse Cholesky solver for sparse SOCP problems was dramatic. We observed speed-ups of up to two orders of magnitude. In our implementation, SDPT3 automatically makes a choice between Matlab's built-in `chol` routine and the sparse Cholesky solver based on the density of $M$. The cutoff density is specified in the parameter `OPTIONS.spdensity`.

The approach of solving (13) by the SQMR method with preconditioner (19) works reasonably well if the following conditions are satisfied: (i) the number of columns of $A^u$ is small and $A^u$ is well-conditioned; (ii) the matrix $M$ is not extremely ill-conditioned. (iii) the matrix $S$ is not extremely ill-conditioned. However, when these conditions are not satisfied, preconditioning (13) based on (19) may not be advisable because either (a) computing $S$ becomes very expensive due to large number of columns in $A^u$, or (b) the computed preconditioner based on (19) is no longer an accurate approximation of $\mathcal{M}^{-1}$. Note that $S$ is typically much more ill-conditioned than $A^u$, especially when $A^u$ is ill-conditioned. When conditions (i)–(iii) are not satisfied, it is advisable to directly use an $LU$ or $LDL^T$ factorization of the symmetric indefinite matrix $\mathcal{M}$ to compute an approximation of $\mathcal{M}^{-1}$. As the matrix $\mathcal{M}$ is usually highly ill-conditioned, we observed that the computed solution based on $LU$ factorization is typically more accurate than the one computed based on $LDL^T$ factorization. Thus, even though $LU$ is twice as expensive as $LDL^T$ factorization when the matrix is dense, we simply use the MATLAB routine `lu` to compute an $LU$ factorization of $\mathcal{M}$, and use the computed $LU$ factors to precondition the SQMR iterative method used to solve (13).

Again, in our implementation, SDPT3 automatically makes a choice on whether to compute a dense or sparse $LU$ factorization based on the density of $\mathcal{M}$. In the case of sparse $LU$ factorization, SDPT3 uses the UMFPACK package of Davis [6], which is available in MATLAB under the `lu` command. We should mention that we have tested the sparse $LDL^T$ factorization of the symmetric matrix $\mathcal{M}$ (when it is a sparse) based on the `MA57` routine [7] (available in MATLAB under the `ldl` command) of the Harwell subroutine library. But we have found that it is not as efficient as the sparse $LU$ factorization based on the UMFPACK package when the matrix is highly ill-conditioned. More importantly, the computed solution based on sparse $LDL^T$ factorization is often not as accurate as the one computed based on sparse $LU$ factorization.

## 5.6 Internal handling of unrestricted blocks

As mentioned in the last sub-section, solving the symmetric indefinite system (13) can potentially be very expensive when $A^u$ is ill-conditioned or has a large number of columns because computing the sparse $LU$ factorization of a sparse matrix can be much more costly than that for a symmetric positive definite matrix of the same order. It is possible to avoid the need to solve a symmetric indefinite system if we reformulate the equality constraint in $(D)$ as

$$\begin{aligned}(A^u)^T y + z^u_+ &= c^u, \quad z^u_+ \geq 0\\ -(A^u)^T y + z^u_- &= -c^u, \quad z^u_- \geq 0,\end{aligned}$$

with the corresponding primal variable $x^u$ expressed as

$$x^u = x^u_+ - x^u_-, \quad x^u_+, x^u_- \geq 0.$$

In this case, the system (13) is replaced by

$$\Big(M + A^u \mathrm{Diag}\,(x^u_+)\mathrm{Diag}\,(z^u_+)^{-1}(A^u)^T + A^u \mathrm{Diag}\,(x^u_-)\mathrm{Diag}\,(z^u_-)^{-1}(A^u)^T\Big)\Delta y = \mathrm{rhs}$$

where rhs denotes the right-hand-side vector. Notice that in contrast to (13), the coefficient matrix is now symmetric positive definite.

But such a reformulation is not without difficulties. In fact, the variables $x^u_+, x^u_-$ tend to become very large and $z^u_+, z^u_-$ tend to become extremely small as the interior-point iteration progresses, and this generally makes the component matrices, $A^u \mathrm{Diag}\,(x^u_+)\mathrm{Diag}\,(z^u_+)^{-1}(A^u)^T$ and $A^u \mathrm{Diag}\,(x^u_-)\mathrm{Diag}\,(z^u_-)^{-1}(A^u)^T$, extremely ill-conditioned.

Fortunately, the following heuristic to modify the vectors $x^u_+, x^u_-$ can typically ameliorate such an ill-conditioning problem:

$$x^u_+ \;:=\; x^u_+ - 0.8\min(x^u_+, x^u_-), \qquad x^u_- \;:=\; x^u_- - 0.8\min(x^u_+, x^u_-).$$

This modification does not change the original variable $x^u$ but does slow down the growth of $x^u_+, x^u_-$. After these modified vectors have been obtained, we also add positive perturbations to the vectors $z^u_+, z^u_-$. Such a modification in $z^u_+, z^u_-$ ensures that they approach 0 at the same rate as $\mu$, and thus prevents the dual problem $(D)$ from approaching the equality constraint too closely prematurely.

For computational efficiency, in the current implementation of SDPT3, we always reformulate an unrestricted vector by the difference of two non-negative vectors.

## 5.7 Dense-column handling

Here we describe our technique to handle dense columns when $M$ is a low-rank perturbation of a sparse matrix. In such a case, the Schur complement matrix $M$ can be written in the form

$$M \;=\; M_{sp} + UDU^T \tag{20}$$

where $M_{sp}$ is a sparse symmetric positive semidefinite matrix, $U$ has only a few columns, and $D$ is a non-singular matrix. If $M_{sp}$ is positive definite, then we can solve (13) by solving a slightly larger but sparse linear system as follows. Let $\lambda = DU^T\Delta y$. It is easy to show that (13) is equivalent to the following linear system:

$$\begin{bmatrix} M_{sp} & A^u & U \\ (A^u)^T & 0 & 0 \\ U^T & 0 & -D^{-1} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x^u \\ \lambda \end{bmatrix} = \begin{bmatrix} h \\ R^u_d \\ 0 \end{bmatrix}. \tag{21}$$

We can use the same method described in Section 5.5 to solve (21).

## 5.8 User supplied routine to compute Schur complement matrix

The current version of SDPT3 allows the user to supply specialized routines to compute the Schur complement matrices $M^s_j$ corresponding to the semidefinite blocks.

The specialized routine to compute $M^s_j$ should have first line that look like:

```
function schurmat = schurfun_jth(U,V,schurfun_jth_par);
```

where the input arguments `U` and `V` should correspond to $x^s_j$ and $(z^s_j)^{-1}$ if the HKM direction is used; and they should correspond to the NT scaling matrix $w^s_j$ if the NT direction is used. The third optional argument `schurfun_jth_par` can be a structure array that stores the parameters needed inside the function `schurfun_jth.m`.

The user can tell SDPT3 to use the specialized routine by setting the $L \times 1$ cell array `OPTIONS.schurfun` as follows: set `OPTIONS.schurfun{j} = schurfun_jth` if $M^s_j$

is to be computed by the specialized routine coded in the function `schurfun_jth.m`; otherwise set `OPTIONS.schurfun{j} = []`. If the function `schurfun_jth.m` requires some parameters, then the $L \times 1$ cell array `OPTIONS.schurfun_par` must also be set correspondingly as follows: set `OPTIONS.schurfun_par{j} = schurfun_jth_par`; otherwise set `OPTIONS.schurfun_par{j} = []`.

Below is an example on we how use the specialized routine `mcpschur.m` in the subdirectory `Examples` to compute the Schur complement matrix when solving the SDP problem `mcp250-1.dat-s`.

```
>> [blk,At,C,b] = read_sdpa('./sdplib/mcp250-1.dat-s');
>> OPTIONS.schurfun{1} = 'mcpschur';
>> [obj,X,y,Z]=sqlp(blk,At,C,b,OPTIONS);
```

In the above example, there is no need to set the cell array `OPTIONS.schurfun_par` since the function `mcpschur.m` does not need any additional parameters.

## 5.9 Step-length computation

Once a direction $\Delta x$ is computed, a full step will not be allowed if $x + \Delta x$ violates the conic constraints. Thus, the next iterate must take the form $x + \alpha\Delta x$ for an appropriate choice of the step-length $\alpha$. In this subsection, we discuss an efficient strategy to compute the step-length $\alpha$.

For semidefinite blocks, it is straightforward to verify that, for the $j$th block, the maximum allowed step-length that can be taken without violating the positive semidefiniteness of the matrix $x_j^s + \alpha_j^s \Delta x_j^s$ is given as follows:

$$\alpha_j^s = \begin{cases} \dfrac{-1}{\lambda_{\min}((x_j^s)^{-1}\Delta x_j^s)}, & \text{if the minimum eigenvalue } \lambda_{\min} \text{ is negative} \\ \infty & \text{otherwise.} \end{cases} \tag{22}$$

If the computation of eigenvalues necessary in $\alpha_j^s$ above becomes expensive, then we resort to finding an approximation of $\alpha_j^s$ by estimating extreme eigenvalues using Lanczos iterations [31]. This approach is quite accurate in general and represents a good trade-off between the computational effort versus quality of the resulting stepsizes.

For quadratic blocks, the largest step-length $\alpha_i^q$ that keeps the next iterate feasible with respect to the $i$th quadratic cone can be computed as follows. Let

$$a_i = \gamma(\Delta x_i^q)^2, \quad b_i = \langle \Delta x_i^q,\, J_i^q x_i^q \rangle, \quad c_i = \gamma(x_i^q)^2, \quad d_i \;=\; b_i^2 - a_i c_i,$$

where $J_i^q$ is the matrix defined in (10). We want the largest positive $\bar{\alpha}$ for which $a_i\alpha^2 + 2b_i\alpha + c_i > 0$ for all smaller positive $\alpha$'s, which is given by

$$\alpha_i^q = \begin{cases} \dfrac{-b_i - \sqrt{d_i}}{a_i} & \text{if } a_i < 0 \text{ or } b_i < 0, a_i \le b_i^2/c_i \\ \dfrac{-c_i}{2b_i} & \text{if } a_i = 0, b_i < 0 \\ \infty & \text{otherwise.} \end{cases}$$

For the linear block, the maximum allowed step-length $\alpha_k^l$ for the $k$th component is given by

$$\alpha_k^l \;=\; \begin{cases} \dfrac{-x_k^l}{\Delta x_k^l}, & \text{if } \Delta x_k^l < 0 \\ \infty & \text{otherwise.} \end{cases}$$

Finally, an appropriate step-length $\alpha$ that can be taken in order for $x+\alpha\Delta x$ to satisfy all the conic constraints takes the form

$$\alpha \;=\; \min\Big(1,\ \bar{\gamma} \min_{1\le j\le n_s} \alpha_j^s,\ \bar{\gamma} \min_{1\le i\le n_q} \alpha_i^q,\ \bar{\gamma} \min_{1\le k\le n_l} \alpha_k^l\Big), \tag{23}$$

where $\bar{\gamma}$ (known as the step-length parameter) is typically chosen to be a number slightly less than 1, say 0.98, to ensure that the next iterate $x+\alpha\Delta x$ stays strictly in the interior of all the cones.

For the dual direction $\Delta z$, we let the analog of $\alpha_j^s$, $\alpha_i^q$ and $\alpha_k^l$ be $\beta_j^s$, $\beta_i^q$ and $\beta_k^l$, respectively. Similar to the primal direction, the step-length that can be taken by the dual direction $\Delta z$ is given by

$$\beta \;=\; \min\Big(1,\ \bar{\gamma} \min_{1\le j\le n_s} \beta_j^s,\ \bar{\gamma} \min_{1\le i\le n_q} \beta_i^q,\ \bar{\gamma} \min_{1\le k\le n_l} \beta_k^l\Big). \tag{24}$$

## 6 Homogeneous self-dual model

In the current version of SDPT3, we have implemented algorithms analogous to Algorithm IPC in Appendix A to solve the following 3-parameter homogeneous self-dual (HSD) model [41] of $(P)$ and $(D)$ when the problems have no logarithmic terms or unrestricted variables:

$$\begin{array}{lll} (P_H) & \min & \bar{\alpha}\theta \\ & \text{s.t.} & \begin{bmatrix} 0 & -\mathcal{A} & b & -\bar{b} \\ \mathcal{A}^T & 0 & -c & \bar{c} \\ -b^T & c^T & 0 & -\bar{g} \\ \bar{b}^T & -\bar{c}^T & \bar{g} & 0 \end{bmatrix} \begin{bmatrix} y \\ x \\ \tau \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ z \\ \kappa \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \bar{\alpha} \end{bmatrix} \\ & & x \in K,\ z \in K^*,\ \tau,\kappa \ge 0,\ y \in I\!R^m,\ \theta \in I\!R, \end{array} \tag{25}$$

where for a given $(x_0, y_0, z_0, \tau_0, \kappa_0, \theta_0)$ such that $x_0 \in \text{int}(K)$, $z_0 \in \text{int}(K^*)$, $\tau_0, \kappa_0, \theta_0 > 0$,

$$\bar{b} = \tfrac{1}{\theta_0}(b\tau_0 - Ax_0), \quad \bar{c} = \tfrac{1}{\theta_0}(c\tau_0 - A^T y_0 - z_0),$$
$$\bar{g} = \tfrac{1}{\theta_0}(\langle c,\, x_0\rangle - b^T y_0 + \kappa_0), \quad \bar{\alpha} = \tfrac{1}{\theta_0}(\langle x_0,\, z_0\rangle + \tau_0\kappa_0).$$

For the (self-dual) model $(P_H)$, we have implemented an algorithm that is analogous to Algorithm IPC in the main program `HSDsqlp.m`. At each iteration of the algorithm, the computation of the search direction is very similar to the case for the problems $(P)$ and $(D)$. For the benefit of readers who are interested in the implementation, here we shall outline the computation of the HKM-like search direction for $(P_H)$. (It is only HKM-like because, to keep corresponding iterates in the problem and its dual, it is necessary to mix the HKM and dual HKM directions suitably.) Let

$$\widehat{\mathcal{A}} = \Big[\mathcal{A};\ -c^T;\ \bar{c}^T\Big], \quad \widehat{y} = \Big[y;\ \tau;\ \theta\Big]. \tag{26}$$

The HKM-like search direction for $(P_H)$ is the solution of the following linear system of equations:

$$\begin{aligned}
\widehat{\mathcal{A}}\Delta x + \widehat{B}\Delta\widehat{y} - [0;\Delta\kappa;0] &= \widehat{R}_p := [0;\kappa;-\bar{\alpha}] - \widehat{\mathcal{A}}x - \widehat{B}\widehat{y} \\
\widehat{\mathcal{A}}^T\Delta\widehat{y} + \Delta z &= R_d := -\widehat{\mathcal{A}}^T\widehat{y} - z \\
\Delta x + H(\Delta z) &= R_c \\
\Delta\kappa + (\kappa/\tau)\Delta\tau &= R_t := \widehat{\mu}/\tau - \kappa,
\end{aligned} \tag{27}$$

where $\widehat{\mu} = (\langle x,\, z\rangle + \tau\kappa)/(n+1)$. In (27), the operator $H$ is understood to be acting on the individual blocks of $\Delta z$ as in (11) and $R_c = [R_c^s; R_c^q; R_c^l]$, and

$$\widehat{B} = \begin{bmatrix} 0 & -b & \bar{b} \\ b^T & 0 & \bar{g} \\ -\bar{b}^T & -\bar{g} & 0 \end{bmatrix}.$$

From here, the equation analogous to the Schur complement equation in (13) without $A^u$ is given as follows:

$$\Big(\widehat{\mathcal{A}}H\widehat{\mathcal{A}}^T + \widehat{B} + \mathrm{diag}\,([0;\kappa/\tau;0])\Big)\Delta\widehat{y} \;=\; \widehat{h} := \widehat{R}_p + \widehat{\mathcal{A}}(H(R_d) - R_c) + [0;R_t;0]. \tag{28}$$

The main difference between (28) and (13) without $A^u$ lies in the term $\widehat{B}$, which is an skew-symmetric matrix with rank at most four, generally containing dense columns. To efficiently solve (28), we use the dense-column handling technique described in Section 5.7 to solve the linear system.

We have found that the solution obtained by the solver `HSDsqlp.m` based on the HSD model $(P_H)$ is generally more accurate than the one obtained by `sqlp.m` based on $(P)$ and $(D)$ when the SQLP problem is feasible but either the primal or dual feasible region has an empty interior. In Figure 1, we plotted the accuracies attained by `HSDsqlp.m` against those attained by `sqlp.m` for about 200 SQLPs whose primal or dual feasible regions are very likely to have empty interiors (based on the fact that the value $\max\{g_P, g_D\}$ described in Section 7 are larger than $10^{12}$). We may observe from the scattered plot in Figure 1 that `HSDsqlp.m` tends to give more accurate solutions compared to `sqlp.m`.

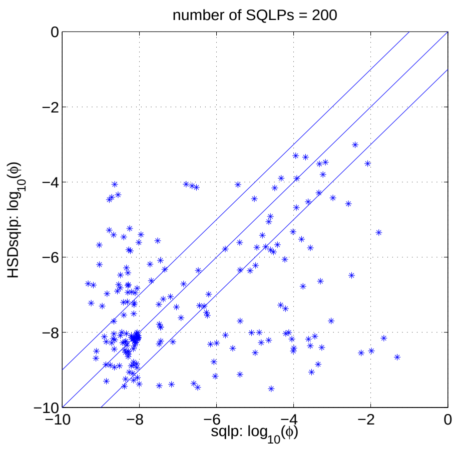

Figure 1: Accuracy attained by `HSDsqlp.m` versus that attained by `sqlp.m` for SQLPs whose primal or dual feasible regions have empty interiors.

## 7 Detecting feasible regions with empty interiors

In this section, we only consider the case where there are no unrestricted variables $x^u$ in $(P)$. Let $d = (A, b, c)$ be the SQLP data associated with $(P)$ and $(D)$ and $F_P(d)$ and $F_D(d)$ be their respective feasible regions. It is often of interest to know whether the interiors, $F_P^\circ(d)$ and $F_D^\circ(d)$, are empty, and how "thick" these regions are. A natural quantitative measure of the "thickness" of $F_P(d)$ and $F_D(d)$ is the concept of primal and dual distances to infeasibility defined by Renegar [36]:

$$\rho_P(d) = \inf\{|||\Delta d||| \ : F_P(d+\Delta d) = \emptyset\}, \quad \rho_D(d) = \inf\{|||\Delta d||| \ : F_D(d+\Delta d) = \emptyset\}.$$

With an appropriately chosen norm in the above definitions, the computation of $\rho_D(d)$ amounts to solving an SQLP problem with roughly the same dimension and structure as the original primal instance. Unfortunately, the computation of $\rho_P(d)$ is extremely costly, requiring the solutions of $2m$ SQLP problems each with roughly the same dimension and structure as the original dual instance; see [10].

However, in practice, one is typically only interested in the magnitudes of $\rho_P(d)$ and $\rho_D(d)$ rather than the exact values. It turns out that the following geometric measures proposed by Freund [9] usually give enough information about the magnitudes of $\rho_P(d)$ and $\rho_D(d)$:

$$g_P(d) \;=\; \inf\left\{\max\left\{|||x|||,\ \frac{|||x|||}{r},\ \frac{1}{r}\right\} \ :\ \mathcal{A}(x) = b,\ \ x - re \in K\right\} \tag{29}$$

$$g_D(d) \;=\; \inf\left\{\max\left\{|||z|||,\ \frac{|||z|||}{r},\ \frac{1}{r}\right\} \ :\ \mathcal{A}^T(y) + z = c,\ \ z - re \in K\right\}, \tag{30}$$

where $e$ is the unit element in $K$ and $|||\cdot|||$ is an appropriately chosen norm. Note that $g_P(d)$ is smaller to the extent that there is a primal feasible solution that is not too large and that is not too close to the boundary of $K$. Similar interpretation holds also for $g_D(d)$. It can be shown that $g_P(d) = \infty \Leftrightarrow \rho_P(d) = 0$ and $g_D(d) = \infty \Leftrightarrow \rho_D(d) = 0$.

Freund [9] showed that $g_P(d)$ ($g_D(d)$) can be computed at the cost of solving a single SQLP problem with roughly the same dimension and structure as the original primal (dual) instance. In the current release of SDPT3, we include the following m-files to compute $g_P(d)$ and $g_D(d)$:

```
function gp = gpcomp(blk,At,C,b);
function gd = gdcomp(blk,At,C,b);
```

The above routines are based on the standard SQLP formulations of (29) and (30) derived in [10].

Let $(x^*, y^*, z^*)$ be an optimal solution to $(P)$ and $(D)$, respectively. The geometric measures $g_P, g_D$ and $\max\{\|x^*\|, \|z^*\|\}$ for the test problems considered in Section 8 are listed in Table 1. In the table, we declare that $g_P$ ($g_D$) is equal to $\infty$ if the computed number is larger than $10^{12}$. We have observed that there is strong correlation between $g := \max\{g_P, g_D\} \max\{\|x^*\|, \|z^*\|\}$ and the accuracy $\phi$ one can obtain when solving $(P)$ and $(D)$. In Figure 2, we plotted the measure $g$ and the accuracy $\phi$ attainable by `sqlp.m` and `HSDsqlp.m` for about 420 SQLP problems. The sample correlation between $\log_{10}(\phi)$ with $\phi$ obtained by `sqlp.m` and $\log_{10}(g)$ for about 170 SQLPs with finite $g$ has a correlation coefficient of 0.75. Coincidentally, we have the same correlation coefficient for the case where $\phi$ is obtained by `HSDsqlp.m`.

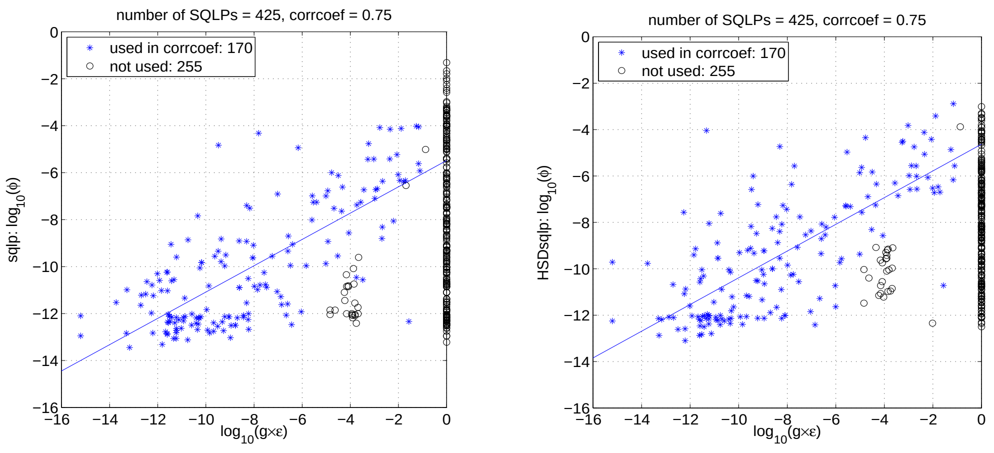

Figure 2: The data points corresponding to "$\circ$" are not used in the calculation of the correlation coefficient because either $1/\max\{g_P, g_D\} = 0$ or because it is not computed to sufficient accuracy to determine whether the number is in fact 0. On the $x$-axis, $\varepsilon = 2.2 \times 10^{-16}$, and $g = \max\{g_P, g_D\} \max\{\|x^*\|, \|z^*\|\}$.

## 8 Computational results

Here we describe the results of our computational testing of SDPT3-4.0 using the default parameters, on problems from the following sources:

1. SDPLIB collection of Borchers, available at
   `http://www.nmt.edu/~borchers/sdplib.html`
2. DIMACS Challenge test problems, available at
   `http://dimacs.rutgers.edu/Challenges/Seventh/Instances/`
3. Sparse SDPs from structural optimization, available at
   `http://www2.am.uni-erlangen.de/~kocvara/pennon/problems.html`
4. Sparse SDP collection of Hans Mittelmann, available at
   `ftp://plato.asu.edu/pub/sdp/`
5. SDPs from electronic structure calculations, available at
   `http://www.cims.nyu.edu/~mituhiro/software.html`
6. SDPs from polynomial optimizations [17].
7. SOCP problems generated by the Matlab FIR filter toolbox, available at
   `http://www.csee.umbc.edu/~dschol2/opt.html`
8. SDPs from polynomial optimizations [37].

Our results were obtained on an Intel Xeon 3.0GHz PC with 4G of memory running Linux and Matlab 7.6. Figure 3 shows the performance of the hybrid solver `sdpt3.m` on a total of about 430 SQLP problems. It shows that `sdpt3.m` was able to solve more than 80% of the problems to an accuracy of at least $10^{-6}$ in the measure $\phi$ defined in (6).

Detailed information such as primal and dual objective values, error measures such as `pinfeas, dinfeas, relgap` as defined in (6), the geometric measures $g_P, g_D$, $\max\{\|x^*\|, \|z^*\|\}$, and the CPU time taken for each problem can be found in Table 1.

## 9 Future work

In a future release of SDPT3, we plan to include an implementation of a path-following algorithm which targets high accuracy solutions at the expense of significantly longer computing time per iteration. The key idea would be to compute the search direction at each iteration based on a reduced augmented equation as formulated in [35] and [4] that has at most twice the dimension of the Schur complement equation.

## Appendix: A primal-dual infeasible-interior-point algorithm

Here we give a **pseudo-code** for the algorithm we implemented. Note that this description makes references to earlier sections where details related to the algorithm are explained.

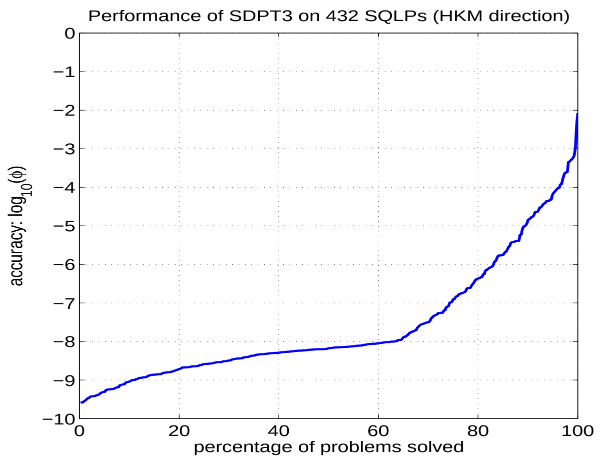

Figure 3: $\phi$ is defined as in (6).

**Algorithm IPC.** Suppose we are given an initial iterate $(x^0, y^0, z^0)$ with $x^0, z^0$ strictly satisfying all the conic constraints. Decide on the type of search direction to use. Set $\bar{\gamma}^0 = 0.9$.
**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 parameter be denoted by $\bar{\gamma}$ and $\bar{\gamma}^+$ respectively.)

- Compute $\mu(x, z)$ defined in (5), and the accuracy measure $\phi$ defined in (6). Stop the iteration if $\phi$ is sufficiently small.
- (Predictor step) Solve the linear system (13) with $\sigma = 0$ in the right-side vector (15). Denote the solution of (11) by $(\delta x, \delta y, \delta z)$. Let $\alpha_p$ and $\beta_p$ be the step-lengths defined as in (23) and (24) with $\Delta x, \Delta z$ replaced by $\delta x, \delta z$, respectively.
- Take $\sigma$ to be

  $$\sigma \;=\; \min\left(1,\; \left[\frac{\mu(x + \alpha_p\, \delta x, z + \beta_p\, \delta z)}{\mu(x, z)}\right]^e\right),$$

  where the exponent $e$ is chosen as follows:

  $$e \;=\; \begin{cases} \max[1, 3\min(\alpha_p, \beta_p)^2] & \text{if } \mu(x, z) > 10^{-6}, \\ 1 & \text{if } \mu(x, z) \leq 10^{-6}. \end{cases}$$

- (Corrector step) Solve the linear system (13) with $R_c$ in the the right-hand side vector (15) replaced by

  $\widehat{R}_c^\tau = R_c^\tau -$ Mehrotra-corrector term generated from $\delta x^\tau$ and $\delta z^\tau$, $\quad \tau \in \{s, q, l\}$.

  Denote the solution of (11) by $(\Delta x, \Delta y, \Delta z)$.

- Update $(x, y, z)$ to $(x^+, y^+, z^+)$ by

$$x^+ = x + \alpha\,\Delta x, \quad y^+ = y + \beta\,\Delta y, \quad z^+ = z + \beta\,\Delta z,$$

  where $\alpha$ and $\beta$ are computed as in (23) and (24) with $\bar{\gamma}$ chosen to be $\bar{\gamma} = 0.9 + 0.09\min(\alpha_p, \beta_p)$.
- Update the step-length parameter by $\bar{\gamma}^+ = 0.9 + 0.09\min(\alpha, \beta)$.

## References

[1] 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.

[2] B. Borchers, *SDPLIB 1.2, a library of semidefinite programming test problems*, Optimization Methods and Software, 11 & 12 (1999), pp. 683–690. Available at `http://www.nmt.edu/~borchers/sdplib.html`.

[3] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory.* SIAM Studies in Applied Mathematics. SIAM, Philadelphia, USA, 1994

[4] Z. Cai and K.C. Toh, *Solving second order cone programming via the augmented systems*, SIAM J. Optimization, 17 (2006), pp. 711–737.

[5] T. A. Davis, *CHOLMOD Version 1.0 User Guide*, Dept. of Computer and Information Science and Engineering, Univ. of Florida, Gainesville, FL, 2005. Details available at `http://www.cise.ufl.edu/research/sparse/cholmod`

[6] T. A. Davis, *UMFPACK Version 4.6 User Guide*, Dept. of Computer and Information Science and Engineering, Univ. of Florida, Gainesville, FL, 2002. Details available at `http://www.cise.ufl.edu/research/sparse/umfpack`

[7] I. S. Duff, *MA57 A new code for the solution of sparse symmetric definite and indefinite systems*, Technical Report RAL-TR-2002-024, Rutherford Appleton Laboratory, 2002.

[8] M. Grant and S. Boyd, *Graph implementations for nonsmooth convex programs*, in Recent Advances in Learning and Control (a tribute to M. Vidyasagar), V. Blondel, S. Boyd, and H. Kimura, editors, Lecture Notes in Control and Information Sciences, Springer, 2008, pp. 95-110. Software is available at `http://cvxr.com/cvx/`

[9] R. M. Freund, *Complexity of convex optimization using geometry-based measures and a reference point*, Mathematical Programming, 99 (2004), pp. 197-221.

[10] R.M. Freund, F. Ordóñez, and K.-C. Toh, *Behavioral Measures and their Correlation with IPM Iteration Counts on Semi-Definite Programming Problems*, Mathematical Programming, 109 (2007), pp. 445–475.

[11] R. W. Freund and N. M. Nachtigal, *A new Krylov-subspace method for symmetric indefinite linear systems*, Proceedings of the 14th IMACS World Congress on Computational and Applied Mathematics, Atlanta, USA, W.F. Ames ed., July 1994, pp. 1253–1256.

[12] K. Fujisawa, M. Kojima, K. Nakata, and M. Yamashita, *SDPA (SemiDefinite Programming Algorithm) User's manual — version 6.2.0*, Research Report B-308, Department Mathematical and Computing Sciences, Tokyo Institute of Technology, December 1995, revised September 2004.

[13] 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.

[14] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., Johns Hopkins University Press, Baltimore, MD, 1989.

[15] Harwell Subroutine Library: http://www.cse.clrc.ac.uk/Activity/HSL

[16] 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.

[17] D. Henrion, private communication.

[18] N. J. Higham, *Computing the nearest correlation matrix — a problem from finance*, IMA J. Numerical Analysis, 22 (2002), pp. 329–343.

[19] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.

[20] 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.

[21] J. Löfberg, *A Toolbox for Modeling and Optimization in* MATLAB, Proceedings of the CACSD Conference, 2004, Taipei, Taiwan.
Available at `http://control.ee.ethz.ch/~joloef/yalmip.php`

[22] M. S. Lobo, L. Vandenberghe, S. Boyd and H. Lebret, *Applications of Second-order Cone Programming*, Linear Algebra Appl., 284 (1998), pp. 193–228.

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

[24] R. D. C. Monteiro, *Primal-dual path-following algorithms for semidefinite programming*, SIAM J. Optimization, 7 (1997), pp. 663–678.

[25] R. D. C. Monteiro, and T. Tsuchiya, *Polynomial convergence of primal-dual algorithms for the second-order cone program based on the MZ-family of directions*, Mathematical Programming, 88 (2000), pp. 61–83.

[26] J.W. Liu, E.G. Ng, and B.W. Peyton, *On finding supernodes for sparse matrix computations*, SIAM J. Matrix Anal. Appl., 1 (1993), pp. 242–252.

[27] G. Pataki and S. Schmieta, *The DIMACS library of mixed semidefinite-quadratic-linear programs.*
Available at `http://dimacs.rutgers.edu/Challenges/Seventh/Instances`

[28] Yu. E. Nesterov and M. J. Todd, *Self-scaled barriers and interior-point methods in convex programming*, Math. Oper. Res., 22 (1997), pp. 1–42.

[29] J. F. Sturm, *Using SeDuMi 1.02, a Matlab toolbox for optimization over symmetric cones*, Optimization Methods and Software, 11 & 12 (1999), pp. 625–653.

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

[31] K. C. Toh, *A note on the calculation of step-lengths in interior-point methods for semidefinite programming*, Computational Optimization and Applications, 21 (2002), pp. 301–310.

[32] K. C. Toh, *Some new search directions for primal-dual interior point methods in semidefinite programming*, SIAM J. Optimization, 11 (2000), pp. 223–242.

[33] K. C. Toh, *Primal-dual path-following algorithms for determinant maximization problems with linear matrix inequalities*, Computational Optimization and Applications, 14 (1999), pp. 309–330.

[34] K. C. Toh, M. J. Todd, and R. H. Tütüncü, *SDPT3 — a Matlab software package for semidefinite programming*, Optimization Methods and Software, 11/12 (1999), pp. 545-581.

[35] K. C. Toh, *Solving large scale semidefinite programs via an iterative solver on the augmented systems*, SIAM J. Optim., 14 (2003), pp. 670–698.

[36] J. Renegar, *Linear programming, complexity theory, and elementary functional analysis*, Mathematical Programming, 70 (1995), pp. 279-351.

[37] H. Waki, S. Kim, M. Kojima, M. Muramatsu and H. Sugimoto, *SparsePOP: a Sparse Semidefinite Programming Relaxation of Polynomial Optimization Problems*, ACM Transactions on Mathematical Software, 35 (2008), article 15.

[38] T. Tsuchiya, *A convergence analysis of the scaling-invariant primal-dual path-following algorithms for second-order cone programming*, Optimization Methods and Software, 11/12 (1999), pp. 141–182.

[39] R. H. Tütüncü, K. C. Toh, and M. J. Todd, *Solving semidefinite-quadratic-linear programs using SDPT3*, Mathematical Programming Ser. B, 95 (2003), pp. 189–217.

[40] L. Vandenberghe, S. Boyd, and S.-P. Wu, *Determinant maximization with linear matrix inequalities*, SIAM J. Matrix Analysis and Applications, 19 (1998), pp. 499–533.

[41] 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.

Table 1: Performance of `sdpt3.m`. In the table, err = [`pinfeas`,`dinfeas`,`relgap`,`relgap2`], where `relgap2` is the same as `relgap` but with the numerator replaced by $|\langle c,\, x\rangle - b^T y|$, and `normXZ` = $\max\{\|x^*\|, \|z^*\|\}$. We declare that $g_P$ ($g_D$) is $\infty$ if the computed number is larger than $10^{12}$.

| problem | $m$ | $n_s; n_q; n_l; n_u$ | it. | primal obj | dual obj | err | | | | time | $g_P$ | $g_D$ | `normXZ` |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| arch0 | 174 | 161; ; 174; | 26 | -5.66517270-1 | -5.66517274-1 | 5.4 -9 | 1.6-11 | 1.9 -9 | 1.8 -9 | 03 | 1.97 4 | 2.03 6 | 5.9 2 |
| arch2 | 174 | 161; ; 174; | 24 | -6.71515400-1 | -6.71515409-1 | 4.2-10 | 3.0-11 | 4.0 -9 | 3.7 -9 | 03 | 1.97 4 | 2.01 6 | 5.7 2 |
| arch4 | 174 | 161; ; 174; | 22 | -9.72627409-1 | -9.72627419-1 | 7.3-10 | 1.9-11 | 3.8 -9 | 3.7 -9 | 03 | 1.97 4 | 1.96 6 | 9.1 2 |
| arch8 | 174 | 161; ; 174; | 23 | -7.05698002 0 | -7.05698004 0 | 9.7 -9 | 2.9-11 | 1.5 -9 | 1.2 -9 | 03 | 1.97 4 | 1.83 6 | 6.3 3 |
| control1 | 21 | 15; ; ; | 17 | -1.77846268 1 | -1.77846267 1 | 3.1 -9 | 2.7-11 | 1.8 -9 | 3.2-10 | 00 | 9.31 4 | 4.98 3 | 5.7 5 |
| control2 | 66 | 30; ; ; | 21 | -8.30000039 0 | -8.29999999 0 | 7.1 -9 | 3.0-11 | 9.4-11 | 2.3 -8 | 01 | 3.03 5 | 1.47 4 | 6.5 5 |
| control3 | 136 | 45; ; ; | 22 | -1.36332647 1 | -1.36332672 1 | 1.9 -7 | 8.7-11 | 1.0 -7 | 8.9 -8 | 02 | 7.67 5 | 3.16 4 | 2.1 6 |
| control4 | 231 | 60; ; ; | 21 | -1.97942325 1 | -1.97942308 1 | 1.9 -7 | 1.5-10 | 1.7 -8 | 4.1 -8 | 04 | 1.34 6 | 4.92 4 | 3.8 6 |
| control5 | 351 | 75; ; ; | 23 | -1.68835936 1 | -1.68836010 1 | 4.3 -7 | 1.6-10 | 2.5 -7 | 2.1 -7 | 11 | 2.02 6 | 6.22 4 | 4.9 6 |
| control6 | 496 | 90; ; ; | 21 | -3.73043648 1 | -3.73044273 1 | 2.2 -7 | 5.8-10 | 8.1 -7 | 8.3 -7 | 22 | 3.12 6 | 9.21 4 | 1.4 7 |
| control7 | 666 | 105; ; ; | 22 | -2.06250581 1 | -2.06250778 1 | 5.9 -8 | 4.9-10 | 4.7 -7 | 4.7 -7 | 43 | 4.09 6 | 1.15 5 | 1.2 7 |
| control8 | 861 | 120; ; ; | 23 | -2.02863478 1 | -2.02863653 1 | 2.0 -7 | 4.8-10 | 4.0 -7 | 4.2 -7 | 1:23 | 5.53 6 | 1.40 5 | 1.3 7 |
| control9 | 1081 | 135; ; ; | 23 | -1.46754157 1 | -1.46754284 1 | 2.7 -7 | 4.7-10 | 4.6 -7 | 4.2 -7 | 2:24 | 6.98 6 | 1.72 5 | 1.3 7 |
| control10 | 1326 | 150; ; ; | 25 | -3.85328687 1 | -3.85330582 1 | 5.0 -7 | 1.4 -9 | 2.3 -6 | 2.4 -6 | 45 | 8.32 6 | 2.00 5 | 3.7 7 |
| control11 | 1596 | 165; ; ; | 24 | -3.19586090 1 | -3.19586862 1 | 5.9 -7 | 1.3 -9 | 8.7 -7 | 1.2 -6 | 1:07 | 1.02 7 | 2.31 5 | 3.4 7 |
| gpp100 | 101 | 100; ; ; | 14 | 4.49435479 1 | 4.49435489 1 | 2.7-10 | 6.2-11 | 9.8 -9 | 1.1 -8 | 00 | $\infty$ | 1.88 2 | 6.2 4 |
| gpp124-1 | 125 | 124; ; ; | 17 | 7.34307525 0 | 7.34307571 0 | 3.6-11 | 7.1-12 | 7.6 -9 | 2.9 -8 | 01 | $\infty$ | 1.92 2 | 1.7 5 |
| gpp124-2 | 125 | 124; ; ; | 15 | 4.68622933 1 | 4.68622939 1 | 1.1-10 | 2.2-11 | 6.1 -9 | 6.3 -9 | 01 | $\infty$ | 2.37 2 | 1.1 5 |
| gpp124-3 | 125 | 124; ; ; | 14 | 1.53014123 2 | 1.53014124 2 | 4.2-10 | 8.5-11 | 4.1 -9 | 5.1 -9 | 01 | $\infty$ | 2.83 2 | 6.9 4 |
| gpp124-4 | 125 | 124; ; ; | 15 | 4.18987595 2 | 4.18987610 2 | 5.6-10 | 7.1-11 | 6.3-10 | 1.8 -8 | 01 | $\infty$ | 3.48 2 | 2.9 5 |
| gpp250-1 | 251 | 250; ; ; | 18 | 1.54449168 1 | 1.54449168 1 | 1.3-12 | 1.5-12 | 2.9 -9 | 8.7-11 | 02 | $\infty$ | 4.01 2 | 1.0 6 |
| gpp250-2 | 251 | 250; ; ; | 15 | 8.18689562 1 | 8.18689574 1 | 1.3-10 | 2.6-11 | 1.2 -9 | 7.4 -9 | 02 | $\infty$ | 4.76 2 | 1.6 5 |
| gpp250-3 | 251 | 250; ; ; | 15 | 3.03539317 2 | 3.03539320 2 | 3.3-10 | 6.7-11 | 2.1 -9 | 5.2 -9 | 02 | $\infty$ | 5.91 2 | 1.9 5 |
| gpp250-4 | 251 | 250; ; ; | 14 | 7.47328306 2 | 7.47328305 2 | 2.2-10 | 5.3-11 | 4.6 -9 | 5.8-10 | 02 | $\infty$ | 7.20 2 | 4.0 5 |
| gpp500-1 | 501 | 500; ; ; | 20 | 2.53205508 1 | 2.53205436 1 | 1.6-12 | 2.6-12 | 1.4 -7 | 1.4 -7 | 12 | $\infty$ | 7.88 2 | 1.3 7 |
| gpp500-2 | 501 | 500; ; ; | 19 | 1.56060387 2 | 1.56060387 2 | 3.1-12 | 6.9-12 | 6.4-10 | 7.7-11 | 11 | $\infty$ | 9.57 2 | 1.6 6 |
| gpp500-3 | 501 | 500; ; ; | 16 | 5.13017610 2 | 5.13017602 2 | 4.3-12 | 2.0-12 | 7.7 -9 | 7.5 -9 | 10 | $\infty$ | 1.17 3 | 1.0 6 |
| gpp500-4 | 501 | 500; ; ; | 17 | 1.56701879 3 | 1.56701879 3 | 8.8-12 | 3.8-12 | 9.2-10 | 8.2-10 | 10 | $\infty$ | 1.50 3 | 7.7 5 |
| hinf1 | 13 | 14; ; ; | 23 | -2.03272656 0 | -2.03267445 0 | 1.3 -7 | 4.1 -8 | 3.4 -6 | 1.0 -5 | 00 | $\infty$ | 7.62 1 | 9.2 3 |
| hinf2 | 13 | 16; ; ; | 16 | -1.09692535 1 | -1.09681526 1 | 3.3 -6 | 1.5-11 | 2.6 -9 | 4.8 -5 | 00 | 1.51 5 | 5.05 3 | 4.6 2 |
| hinf3 | 13 | 16; ; ; | 21 | -5.69679134 1 | -5.69543438 1 | 7.1 -6 | 8.1-12 | 3.1 -9 | 1.2 -4 | 00 | $\infty$ | 1.48 4 | 4.4 3 |
| hinf4 | 13 | 16; ; ; | 21 | -2.74765722 2 | -2.74764791 2 | 7.9 -8 | 1.5 -9 | 2.3 -8 | 1.7 -6 | 00 | $\infty$ | 1.76 3 | 4.6 4 |
| hinf5 | 13 | 16; ; ; | 21 | -3.62897352 2 | -3.62557102 2 | 1.5 -4 | 1.3 -9 | 4.3 -7 | 4.7 -4 | 00 | $\infty$ | 1.04 5 | 2.0 4 |
| hinf6 | 13 | 16; ; ; | 22 | -4.48972738 2 | -4.48952353 2 | 1.5 -5 | 1.3 -8 | 3.6 -6 | 2.3 -5 | 00 | $\infty$ | 6.76 4 | 1.2 5 |
| hinf7 | 13 | 16; ; ; | 18 | -3.90826676 2 | -3.90819918 2 | 9.6 -6 | 1.8-10 | 1.1 -6 | 8.6 -6 | 00 | $\infty$ | 3.55 5 | 2.7 4 |
| hinf8 | 13 | 16; ; ; | 21 | -1.16191071 2 | -1.16168510 2 | 2.8 -5 | 1.0-11 | 3.3 -9 | 9.7 -5 | 00 | $\infty$ | 1.63 4 | 2.5 4 |
| hinf9 | 13 | 16; ; ; | 21 | -2.36249277 2 | -2.36249258 2 | 1.0 -6 | 1.4-14 | 4.1-10 | 3.9 -8 | 00 | 3.09 2 | 1.01 6 | 7.5 4 |
| hinf10 | 21 | 18; ; ; | 28 | -1.08833666 2 | -1.08781552 2 | 1.3 -7 | 6.8 -8 | 6.0 -5 | 2.4 -4 | 00 | $\infty$ | 1.60 3 | 1.7 6 |
| hinf11 | 31 | 22; ; ; | 25 | -6.59349948 1 | -6.59169026 1 | 3.7 -7 | 1.7 -7 | 4.5 -4 | 1.4 -4 | 01 | $\infty$ | 1.26 3 | 1.1 6 |
| hinf12 | 43 | 24; ; ; | 60 | -6.92650138-5 | -5.39089136-5 | 8.9-12 | 3.0 -6 | 4.8 -5 | 1.5 -5 | 01 | $\infty$ | 1.42 3 | 4.1 11 |
| hinf13 | 57 | 30; ; ; | 32 | -4.43539604 1 | -4.43495652 1 | 5.1 -5 | 3.1 -7 | 2.0 -4 | 4.9 -5 | 01 | $\infty$ | 9.37 4 | 1.8 7 |
| hinf14 | 73 | 34; ; ; | 29 | -1.29900752 1 | -1.29900668 1 | 3.5 -7 | 1.9 -7 | 2.5 -5 | 3.1 -7 | 01 | $\infty$ | 3.28 3 | 8.0 5 |
| hinf15 | 91 | 37; ; ; | 30 | -2.40107940 1 | -2.40066089 1 | 3.2 -5 | 3.5 -6 | 4.0 -3 | 8.5 -5 | 01 | $\infty$ | 1.78 5 | 1.2 6 |
| infd1 | 10 | 30; ; ; | 11 | -4.25720801 0 | 1.5422863919 | primal infeasible | | | | 00 | | | |
| infd2 | 10 | 30; ; ; | 11 | 5.26001444 0 | 2.0993105820 | primal infeasible | | | | 00 | | | |
| infp1 | 10 | 30; ; ; | 31 | -9.4170967815 | -9.56504509 0 | dual infeasible | | | | 00 | | | |
| infp2 | 10 | 30; ; ; | 31 | -3.0046017715 | -7.56587983 0 | dual infeasible | | | | 00 | | | |
| mcp100 | 100 | 100; ; ; | 12 | -2.26157351 2 | -2.26157352 2 | 1.2-11 | 1.0-12 | 2.0 -9 | 2.0 -9 | 00 | 1.00 2 | 1.92 2 | 5.0 1 |
| mcp124-1 | 124 | 124; ; ; | 12 | -1.41990475 2 | -1.41990477 2 | 3.6-12 | 1.0-12 | 7.5 -9 | 7.5 -9 | 00 | 1.24 2 | 1.91 2 | 6.5 1 |
| mcp124-2 | 124 | 124; ; ; | 13 | -2.69880170 2 | -2.69880171 2 | 2.0-13 | 1.1-12 | 4.1-10 | 4.1-10 | 01 | 1.24 2 | 2.35 2 | 6.3 1 |
| mcp124-3 | 124 | 124; ; ; | 12 | -4.67750110 2 | -4.67750114 2 | 6.5-13 | 1.0-12 | 4.9 -9 | 4.9 -9 | 01 | 1.24 2 | 2.82 2 | 6.1 1 |
| mcp124-4 | 124 | 124; ; ; | 13 | -8.64411863 2 | -8.64411864 2 | 3.2-12 | 1.5-12 | 4.0-10 | 4.0-10 | 01 | 1.24 2 | 3.53 2 | 6.5 1 |
| mcp250-1 | 250 | 250; ; ; | 14 | -3.17264340 2 | -3.17264340 2 | 6.7-13 | 1.0-12 | 9.7-10 | 9.7-10 | 01 | 2.50 2 | 4.02 2 | 1.3 2 |
| mcp250-2 | 250 | 250; ; ; | 13 | -5.31930081 2 | -5.31930084 2 | 1.4-11 | 1.0-12 | 2.6 -9 | 2.6 -9 | 01 | 2.50 2 | 4.76 2 | 1.1 2 |
| mcp250-3 | 250 | 250; ; ; | 13 | -9.81172566 2 | -9.81172572 2 | 6.9-12 | 1.0-12 | 2.8 -9 | 2.8 -9 | 01 | 2.50 2 | 5.90 2 | 1.0 2 |
| mcp250-4 | 250 | 250; ; ; | 14 | -1.68196010 3 | -1.68196011 3 | 1.8-13 | 1.0-12 | 4.5 -9 | 4.5 -9 | 01 | 2.50 2 | 7.21 2 | 1.1 2 |
| mcp500-1 | 500 | 500; ; ; | 15 | -5.98148516 2 | -5.98148517 2 | 8.5-13 | 1.0-12 | 5.5-10 | 5.5-10 | 04 | 5.00 2 | 7.86 2 | 2.3 2 |
| mcp500-2 | 500 | 500; ; ; | 16 | -1.07005676 3 | -1.07005677 3 | 4.1-13 | 1.2-12 | 1.2 -9 | 1.2 -9 | 05 | 5.00 2 | 9.59 2 | 2.1 2 |
| mcp500-3 | 500 | 500; ; ; | 14 | -1.84796999 3 | -1.84797002 3 | 1.1-12 | 1.0-12 | 9.2 -9 | 9.2 -9 | 05 | 5.00 2 | 1.17 3 | 1.9 2 |
| mcp500-4 | 500 | 500; ; ; | 13 | -3.56673799 3 | -3.56673805 3 | 3.4-12 | 1.0-12 | 8.8 -9 | 8.8 -9 | 05 | 5.00 2 | 1.51 3 | 1.9 2 |
| qap5 | 136 | 26; ; ; | 10 | 4.36000000 2 | 4.36000000 2 | 8.3-12 | 2.7-10 | 7.5-10 | 4.4-10 | 01 | $\infty$ | 1.35 3 | 8.7 3 |
| qap6 | 229 | 37; ; ; | 18 | 3.81393157 2 | 3.81416161 2 | 4.6 -7 | 1.8-10 | 2.6 -9 | 3.0 -5 | 02 | $\infty$ | 3.33 3 | 5.1 4 |

Table 1: Performance of `sdpt3.m`. In the table, err = [`pinfeas`,`dinfeas`,`relgap`,`relgap2`], where `relgap2` is the same as `relgap` but with the numerator replaced by $|\langle c,\, x\rangle - b^T y|$, and `normXZ` $= \max\{\|x^*\|, \|z^*\|\}$. We declare that $g_P$ ($g_D$) is $\infty$ if the computed number is larger than $10^{12}$.

| problem | $m$ | $n_s; n_q; n_l; n_u$ | it. | primal obj | dual obj | err | | | | time | $g_P$ | $g_D$ | `normXZ` |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| qap7 | 358 | 50; ; ; | 16 | 4.24788135 2 | 4.24804149 2 | 3.3 -7 | 3.0-10 | 4.2 -9 | 1.9 -5 | 01 | $\infty$ | 4.07 3 | 5.8 4 |
| qap8 | 529 | 65; ; ; | 17 | 7.56841647 2 | 7.56899243 2 | 9.1 -7 | 2.5 -9 | 3.1 -8 | 3.8 -5 | 01 | $\infty$ | 7.07 3 | 7.8 4 |
| qap9 | 748 | 82; ; ; | 17 | 1.40991866 3 | 1.40992993 3 | 5.8 -8 | 1.7 -9 | 1.5 -8 | 4.0 -6 | 02 | $\infty$ | 1.11 4 | 2.7 5 |
| qap10 | 1021 | 101; ; ; | 17 | 1.09254045 3 | 1.09257436 3 | 2.7 -7 | 1.9 -9 | 2.0 -8 | 1.6 -5 | 05 | $\infty$ | 1.46 4 | 1.6 5 |
| ss30 | 132 | 294; ; 132; | 22 | -2.02395096 1 | -2.02395106 1 | 1.2 -7 | 5.8-11 | 3.2 -8 | 2.4 -8 | 12 | 1.02 3 | 2.40 5 | 1.8 3 |
| theta1 | 104 | 50; ; ; | 11 | -2.29999997 1 | -2.30000001 1 | 1.1-11 | 4.6-12 | 8.2 -9 | 8.2 -9 | 00 | 5.00 1 | 1.15 3 | 2.1 2 |
| theta2 | 498 | 100; ; ; | 13 | -3.28791689 1 | -3.28791690 1 | 1.2-12 | 1.3-12 | 1.4 -9 | 1.4 -9 | 01 | 1.00 2 | 3.29 3 | 4.3 2 |
| theta3 | 1106 | 150; ; ; | 14 | -4.21669813 1 | -4.21669815 1 | 3.5-11 | 1.0-12 | 2.6 -9 | 2.6 -9 | 02 | 1.50 2 | 6.33 3 | 6.6 2 |
| theta4 | 1949 | 200; ; ; | 14 | -5.03212213 1 | -5.03212220 1 | 2.4-13 | 1.0-12 | 7.0 -9 | 7.0 -9 | 08 | 2.00 2 | 1.01 4 | 9.1 2 |
| theta5 | 3028 | 250; ; ; | 14 | -5.72323069 1 | -5.72323073 1 | 1.8-13 | 1.0-12 | 3.5 -9 | 3.5 -9 | 24 | 2.50 2 | 1.43 4 | 1.2 3 |
| theta6 | 4375 | 300; ; ; | 14 | -6.34770870 1 | -6.34770872 1 | 1.4-12 | 1.0-12 | 1.5 -9 | 1.6 -9 | 59 | 3.00 2 | 1.90 4 | 1.4 3 |
| truss1 | 6 | 12; ; 1; | 9 | 8.99999651 0 | 8.99999629 0 | 2.3 -9 | 9.8-11 | 6.9 -9 | 1.2 -8 | 00 | 4.56 2 | 1.30 1 | 2.4 1 |
| truss2 | 58 | 132; ; 1; | 13 | 1.23380357 2 | 1.23380356 2 | 9.3-10 | 5.7-10 | 2.2 -9 | 3.7 -9 | 00 | 6.53 4 | 1.33 2 | 7.1 2 |
| truss3 | 27 | 30; ; 1; | 12 | 9.10999627 0 | 9.10999613 0 | 3.8-14 | 9.9-13 | 7.1 -9 | 7.1 -9 | 00 | 1.14 3 | 3.10 1 | 2.4 1 |
| truss4 | 12 | 18; ; 1; | 11 | 9.00999645 0 | 9.00999629 0 | 3.8 -9 | 1.1-11 | 8.2-11 | 8.2 -9 | 00 | 6.79 2 | 1.90 1 | 2.4 1 |
| truss5 | 208 | 330; ; 1; | 15 | 1.32635678 2 | 1.32635678 2 | 1.5-10 | 3.3-12 | 5.8-10 | 5.7-10 | 01 | 1.75 5 | 3.31 2 | 7.6 2 |
| truss6 | 172 | 450; ; 1; | 25 | 9.01001427 2 | 9.01001389 2 | 5.5 -8 | 2.3-11 | 1.4 -8 | 2.1 -8 | 01 | 1.62 6 | 4.51 2 | 1.1 4 |
| truss7 | 86 | 300; ; 1; | 22 | 9.00001551 2 | 9.00001372 2 | 1.3 -8 | 1.7-11 | 1.0 -7 | 9.9 -8 | 00 | 1.08 6 | 3.01 2 | 1.1 4 |
| truss8 | 496 | 627; ; 1; | 16 | 1.33114589 2 | 1.33114589 2 | 2.2-10 | 8.4-12 | 2.8-10 | 2.8-10 | 03 | 3.35 5 | 6.28 2 | 7.7 2 |
| maxG11 | 800 | 800; ; ; | 15 | -6.29164777 2 | -6.29164783 2 | 1.3-12 | 1.0-12 | 4.8 -9 | 4.8 -9 | 12 | 8.00 2 | 1.41 3 | 3.8 2 |
| maxG32 | 2000 | 2000; ; ; | 15 | -1.56763961 3 | -1.56763964 3 | 7.0-12 | 1.0-12 | 9.9 -9 | 9.9 -9 | 1:47 | 2.00 3 | 3.56 3 | 7.7 2 |
| maxG51 | 1000 | 1000; ; ; | 17 | -4.00625552 3 | -4.00625552 3 | 3.6-13 | 1.0-12 | 2.6-10 | 2.6-10 | 28 | 1.00 3 | 2.05 3 | 3.2 2 |
| qpG11 | 800 | 1600; ; ; | 15 | -2.44865909 3 | -2.44865913 3 | 8.6-13 | 1.0-12 | 8.4 -9 | 8.4 -9 | 12 | 1.60 3 | 6.50 3 | 3.8 2 |
| qpG51 | 1000 | 2000; ; ; | 17 | -1.18179999 4 | -1.18180000 4 | 1.4-11 | 1.1-12 | 2.1 -9 | 2.1 -9 | 25 | 2.00 3 | 2.56 4 | 9.9 2 |
| thetaG11 | 2401 | 801; ; ; | 18 | -3.99999995 2 | -4.00000000 2 | 5.0-12 | 1.0-12 | 6.3 -9 | 6.3 -9 | 40 | 2.40 3 | 9.47 2 | 8.0 2 |
| thetaG51 | 6910 | 1001; ; ; | 39 | -3.48999980 2 | -3.49000001 2 | 4.0 -8 | 2.3-12 | 1.9 -8 | 2.9 -8 | 18:00 | 3.10 4 | 1.10 3 | 7.2 2 |
| equalG11 | 801 | 801; ; ; | 17 | -6.29155292 2 | -6.29155293 2 | 4.6-12 | 9.9-13 | 3.1-10 | 2.8-10 | 31 | 1.60 3 | 2.21 3 | 2.6 5 |
| equalG51 | 1001 | 1001; ; ; | 18 | -4.00560128 3 | -4.00560132 3 | 5.8-11 | 5.1-12 | 4.3 -9 | 4.2 -9 | 59 | 2.00 3 | 3.05 3 | 4.8 5 |
| bm1 | 883 | 882; ; ; | 20 | 2.34398345 1 | 2.34398185 1 | 5.9-12 | 3.1-12 | 3.4 -7 | 3.3 -7 | 44 | $\infty$ | 1.41 3 | 1.2 7 |
| copo14 | 1275 | 196; ; 364; | 17 | 3.05271933-10 | -8.22282898-10 | 1.1-13 | 1.5-12 | 1.1 -9 | 1.1 -9 | 02 | 7.98 2 | 8.40 2 | 7.1 1 |
| copo23 | 5820 | 529; ; 1771; | 20 | 2.21718748-10 | -5.46605094-10 | 7.9-13 | 1.0-12 | 7.7-10 | 7.7-10 | 57 | 2.90 3 | 3.25 3 | 1.5 2 |
| hamming-7- | 1793 | 128; ; ; | 8 | -4.26666661 1 | -4.26666668 1 | 1.0-10 | 3.7-11 | 8.3 -9 | 8.3 -9 | 03 | 1.28 2 | 5.46 3 | 5.0 2 |
| hamming-9- | 2305 | 512; ; ; | 9 | -2.23999998 2 | -2.24000001 2 | 3.4-11 | 2.2-11 | 6.1 -9 | 6.1 -9 | 10 | 5.12 2 | 1.15 5 | 5.5 3 |
| minphase | 48 | 48; ; ; | 11 | 5.77209594-1 | 5.77209579-1 | 2.6-13 | 2.3-12 | 7.0 -9 | 6.9 -9 | 00 | $\infty$ | 1.18 3 | 1.9 2 |
| torusg3-8 | 512 | 512; ; ; | 15 | -4.83409459 7 | -4.83409459 7 | 2.5-13 | 1.0-12 | 1.0 -9 | 1.0 -9 | 05 | 5.12 2 | 4.70 7 | 2.6 6 |
| toruspm3-8 | 512 | 512; ; ; | 14 | -5.27808661 2 | -5.27808663 2 | 4.4-11 | 1.0-12 | 2.2 -9 | 2.2 -9 | 05 | 5.12 2 | 1.04 3 | 2.1 2 |
| torusg3-15 | 3375 | 3375; ; ; | 16 | -6.37621845 3 | -6.37621855 3 | 3.7-13 | 1.0-12 | 7.4 -9 | 7.4 -9 | 9:31 | 3.38 3 | 9.70 3 | 1.1 3 |
| toruspm3-1 | 3375 | 3375; ; ; | 16 | -3.47513185 3 | -3.47513186 3 | 5.7-13 | 1.0-12 | 1.7 -9 | 1.7 -9 | 9:39 | 3.38 3 | 6.85 3 | 9.9 2 |
| filter48 | 969 | 48; 49; 931; | 41 | 1.41612915 0 | 1.41612864 0 | 9.5 -8 | 1.2 -9 | 1.7 -7 | 1.3 -7 | 25 | 1.14 8 | 2.30 3 | 4.6 2 |
| filtinf1 | 983 | 49; 49; 945; | 34 | 0.00000000-16 | 1.20609862 2 | primal infeasible | | | | 22 | | | |
| nb | 123 | ; 2379; 4; | 25 | -5.07030871-2 | -5.07030950-2 | 1.4-10 | 5.4-10 | 7.6 -9 | 7.1 -9 | 07 | 1.59 3 | $\infty$ | 1.7 0 |
| nb-L1 | 915 | ; 2379; 797; | 36 | -1.30122706 1 | -1.30122707 1 | 4.0-10 | 3.8-10 | 9.0 -9 | 4.1 -9 | 14 | 3.16 3 | $\infty$ | 9.8 1 |
| nb-L2 | 123 | ; 4191; 4; | 19 | -1.62897198 0 | -1.62897196 0 | 5.9-11 | 7.3 -9 | 1.3 -9 | 3.0 -9 | 10 | 1.68 3 | $\infty$ | 5.8 0 |
| nb-L2-bess | 123 | ; 2637; 4; | 17 | -1.02569503-1 | -1.02569511-1 | 8.0-12 | 2.4 -9 | 7.3 -9 | 6.4 -9 | 05 | 1.68 3 | $\infty$ | 5.3 0 |
| nql30 | 3680 | ; 2700; 3602; | 35 | -9.46028479-1 | -9.46028497-1 | 1.5 -9 | 2.2-10 | 9.2 -9 | 6.2 -9 | 03 | 5.40 3 | $\infty$ | 5.4 1 |
| nql60 | 14560 | ; 10800; 14402; | 42 | -9.35052923-1 | -9.35052943-1 | 1.4 -9 | 1.3-10 | 6.9 -9 | 7.1 -9 | 21 | 2.16 4 | $\infty$ | 1.1 2 |
| nql180 | 130080 | ; 97200; 129602; | 61 | -9.27728615-1 | -9.27728621-1 | 5.7 -9 | 9.7-12 | 7.9 -9 | 2.0 -9 | 5:07 | 1.94 5 | $\infty$ | 3.2 2 |
| qssp30 | 3691 | ; 7564; 2; | 22 | -6.49667580 0 | -6.49667575 0 | 9.1 -9 | 4.5-10 | 6.0-11 | 3.5 -9 | 03 | 3.78 3 | $\infty$ | 6.2 1 |
| qssp60 | 14581 | ; 29524; 2; | 28 | -6.56270638 0 | -6.56270644 0 | 6.7 -9 | 3.2-10 | 7.4-10 | 4.2 -9 | 17 | 1.48 4 | $\infty$ | 1.2 2 |
| qssp180 | 130141 | ; 261364; 2; | 37 | -6.63959903 0 | -6.63960691 0 | 3.5 -7 | 7.7 -9 | 1.3 -9 | 5.5 -7 | 4:48 | 1.31 5 | $\infty$ | 3.6 2 |
| sched-50-5 | 2527 | ; 2477; 2502; | 37 | 2.66730025 4 | 2.66730009 4 | 5.6 -8 | 3.1 -8 | 3.1 -8 | 3.0 -8 | 02 | 9.76 6 | 2.24 6 | 1.3 5 |
| sched-100- | 4844 | ; 4744; 5002; | 34 | 1.81915139 5 | 1.81884771 5 | 2.2 -5 | 1.1 -6 | 8.4 -5 | 8.3 -5 | 04 | 2.95 7 | 1.53 6 | 2.6 5 |
| sched-100- | 8338 | ; 8238; 10002; | 31 | 7.17499025 5 | 7.17362685 5 | 2.1 -5 | 2.3 -5 | 9.5 -5 | 9.5 -5 | 09 | 2.59 8 | 7.88 7 | 9.6 5 |
| sched-200- | 18087 | ; 17887; 20002; | 39 | 1.41361263 5 | 1.41360169 5 | 7.1 -5 | 2.9 -6 | 3.9 -6 | 3.9 -6 | 39 | 9.07 7 | 6.80 6 | 2.2 5 |
| sched-50-5 | 2526 | ; 2475; 2502; | 28 | 7.85203852 0 | 7.85203844 0 | 3.1 -8 | 7.6-10 | 5.0 -9 | 4.9 -9 | 02 | 5.42 3 | 8.11 5 | 3.7 1 |
| sched-100- | 4843 | ; 4742; 5002; | 31 | 6.71650335 1 | 6.71650307 1 | 3.0 -8 | 2.3-10 | 2.3 -8 | 2.0 -8 | 04 | 1.59 6 | 5.70 5 | 6.0 4 |
| sched-100- | 8337 | ; 8236; 10002; | 26 | 2.73307954 1 | 2.73307853 1 | 2.4 -8 | 4.4-10 | 2.1 -7 | 1.8 -7 | 08 | 1.99 4 | 3.70 7 | 1.6 5 |
| sched-200- | 18086 | ; 17885; 20002; | 32 | 5.18119621 1 | 5.18119610 1 | 2.4 -7 | 1.2-10 | 1.1 -8 | 1.0 -8 | 33 | 6.30 4 | 2.76 6 | 7.7 4 |
| biggs | 1819 | 702; ; ; | 54 | -1.41425840 3 | -1.41425840 3 | 6.2-10 | 9.8-12 | 2.1 -9 | 2.1 -9 | 1:22 | 5.79 4 | 6.06 9 | 3.5 7 |
| buck1 | 36 | 49; ; 36; | 17 | -1.46419152 2 | -1.46419152 2 | 3.1-10 | 1.5-12 | 5.8-10 | 5.8-10 | 01 | 1.86 2 | 2.89 4 | 7.7 2 |
| buck2 | 144 | 193; ; 144; | 21 | -2.92368264 2 | -2.92368296 2 | 5.3 -8 | 4.8-10 | 5.4 -8 | 5.5 -8 | 04 | 4.50 2 | 1.31 6 | 1.0 4 |
| buck3 | 544 | 641; ; 544; | 31 | -6.07601691 2 | -6.07606037 2 | 3.8 -6 | 2.6 -8 | 3.5 -6 | 3.6 -6 | 15 | 1.55 3 | 1.94 7 | 2.2 5 |

Table 1: Performance of `sdpt3.m`. In the table, err = [`pinfeas`,`dinfeas`,`relgap`,`relgap2`], where `relgap2` is the same as `relgap` but with the numerator replaced by $|\langle c,\, x\rangle - b^T y|$, and `normXZ` $= \max\{\|x^*\|, \|z^*\|\}$. We declare that $g_P$ ($g_D$) is $\infty$ if the computed number is larger than $10^{12}$.

| problem | $m$ | $n_s; n_q; n_l; n_u$ | it. | primal obj | dual obj | err | | | | time | $g_P$ | $g_D$ | `normXZ` |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| buck4 | 1200 | 1345; ; 1200; | 37 | -4.86141974 2 | -4.86141983 2 | 8.0 -8 | 7.2-10 | 9.7 -9 | 8.7 -9 | 1:34 | 3.39 3 | 1.75 8 | 2.7 4 |
| buck5 | 3280 | 3521; ; 3280; | 40 | -4.36197041 2 | -4.36263509 2 | 1.3 -6 | 1.7 -7 | 7.6 -5 | 7.6 -5 | 13:49 | 8.69 3 | 9.45 8 | 6.1 4 |
| cnhil10 | 5005 | 220; ; ; | 33 | 0.00000000-16 | -1.60338275-4 | 4.4 -8 | 7.1 -9 | 5.6 -5 | 1.6 -4 | 44 | $\infty$ | 3.50 2 | 9.8 4 |
| cnhil8 | 1716 | 120; ; ; | 31 | 0.00000000-16 | -4.08644723-6 | 1.9 -8 | 2.1 -9 | 3.5 -8 | 4.1 -6 | 06 | $\infty$ | 2.08 2 | 1.3 4 |
| cphil10 | 5005 | 220; ; ; | 10 | 0.00000000-16 | -2.45483691-10 | 6.7-16 | 1.0-12 | 2.6-10 | 2.5-10 | 22 | 5.00 2 | 3.50 2 | 1.4 1 |
| cphil12 | 12376 | 364; ; ; | 10 | 0.00000000-16 | -2.43524851-9 | 0.9-15 | 2.0-12 | 2.5 -9 | 2.4 -9 | 3:13 | 7.85 2 | 5.44 2 | 1.5 1 |
| G40-mb | 2001 | 2000; ; ; | 21 | -2.86432297 3 | -2.86432323 3 | 1.9-12 | 4.2-12 | 4.5 -8 | 4.5 -8 | 16:23 | $\infty$ | 4.91 3 | 7.4 7 |
| G40mc | 2000 | 2000; ; ; | 18 | -5.72957909 3 | -5.72957911 3 | 4.8-12 | 1.0-12 | 1.1 -9 | 1.1 -9 | 4:02 | 2.00 3 | 7.83 3 | 6.4 2 |
| G48mc | 3000 | 3000; ; ; | 12 | -1.20000000 4 | -1.20000000 4 | 2.6-13 | 3.2-12 | 3.8-10 | 3.7-10 | 4:11 | 3.00 3 | 9.00 3 | 3.0 3 |
| mater-1 | 103 | 220; ; 2; | 15 | 1.43465440 2 | 1.43465438 2 | 1.9-10 | 2.3-12 | 8.1 -9 | 8.2 -9 | 00 | 1.05 5 | 2.22 2 | 1.1 3 |
| mater-2 | 423 | 1012; ; 2; | 17 | 1.41591867 2 | 1.41591866 2 | 3.7-11 | 7.6-12 | 1.9 -9 | 1.9 -9 | 01 | 4.72 5 | 1.01 3 | 2.4 3 |
| mater-3 | 1439 | 3586; ; 2; | 18 | 1.33916257 2 | 1.33916256 2 | 1.6-10 | 2.0-11 | 5.3 -9 | 5.3 -9 | 05 | 1.58 6 | 3.59 3 | 4.2 3 |
| mater-4 | 4807 | 12496; ; 2; | 21 | 1.34262717 2 | 1.34262716 2 | 8.8-11 | 1.2-11 | 5.0 -9 | 5.0 -9 | 21 | 5.52 6 | 1.25 4 | 7.9 3 |
| mater-5 | 10143 | 26818; ; 2; | 23 | 1.33801640 2 | 1.33801640 2 | 1.0-10 | 1.4-11 | 2.4 -9 | 2.5 -9 | 53 | 1.18 7 | 2.68 4 | 1.1 4 |
| mater-6 | 20463 | 54626; ; 2; | 29 | 1.33538715 2 | 1.33538715 2 | 4.5-10 | 4.4-11 | 2.7 -9 | 2.7 -9 | 2:26 | 2.40 7 | 5.46 4 | 1.6 4 |
| neosfbr12 | 1441 | 122; ; ; | 17 | 5.29319164 2 | 5.29319158 2 | 8.5-11 | 1.2-12 | 5.3 -9 | 5.3 -9 | 09 | 1.06 4 | 8.15 3 | 7.5 3 |
| prob-1-2-0 | 100 | 200; ; ; | 25 | 4.3881330910 | 4.3881330210 | 1.4 -8 | 6.8-15 | 3.2 -8 | 7.3 -9 | 24 | 5.19 2 | $\infty$ | 1.8 7 |
| prob-1-2-1 | 100 | 200; ; ; | 20 | -5.5663592210 | -5.5663593310 | 4.4-14 | 1.0-12 | 9.7 -9 | 9.6 -9 | 20 | 9.49 2 | 1.22 8 | 9.3 2 |
| prob-2-4-0 | 200 | 400; ; ; | 27 | -6.2807066510 | -6.2807074510 | 3.7 -8 | 1.9-15 | 4.9 -8 | 6.4 -8 | 3:35 | 8.80 2 | $\infty$ | 6.7 7 |
| prob-2-4-1 | 200 | 400; ; ; | 27 | 9.54281067 9 | 9.54281051 9 | 8.0-14 | 3.2-13 | 9.0 -9 | 8.2 -9 | 8:49 | 1.77 3 | 9.51 7 | 1.9 3 |
| neu1 | 3003 | 252; ; 2; | 41 | -7.26676865-9 | -1.99826835-7 | 6.9-10 | 4.2-10 | 4.4 -7 | 1.9 -7 | 8:24 | $\infty$ | $\infty$ | 1.1 3 |
| neu1g | 3002 | 252; ; ; | 33 | 1.25000007 2 | 1.24999847 2 | 7.7-10 | 1.0 -9 | 1.7 -6 | 6.4 -7 | 7:09 | $\infty$ | 1.64 3 | 1.8 6 |
| neu2 | 3003 | 252; ; 2; | 43 | -3.96541382-4 | -2.23051493-4 | 4.4 -9 | 1.5-10 | 5.0 -4 | 1.7 -4 | 8:01 | $\infty$ | $\infty$ | 1.5 5 |
| neu2c | 3002 | 1253; ; 2; | 65 | 3.43713381 4 | 3.38996378 4 | 3.2 -5 | 1.2 -6 | 8.4 -3 | 6.9 -3 | 25:45 | $\infty$ | 1.74 10 | 9.0 8 |
| neu2g | 3002 | 252; ; ; | 33 | 3.41000047 4 | 3.40998796 4 | 4.1 -8 | 3.9 -9 | 4.1 -6 | 1.8 -6 | 6:48 | $\infty$ | 1.64 3 | 1.0 7 |
| neu3 | 7364 | 418; ; 2; | 47 | 3.19225137-10 | -2.49655303-9 | 1.9-12 | 1.2-12 | 6.1 -9 | 2.8 -9 | 5:36 | $\infty$ | $\infty$ | 1.5 3 |
| neu3g | 8007 | 462; ; ; | 49 | 1.58672362-5 | -9.13182744-5 | 5.4-15 | 3.6 -6 | 2.4 -4 | 1.1 -4 | 7:00 | $\infty$ | 2.55 3 | 1.1 11 |
| rendl1-600 | 601 | 600; ; ; | 15 | -5.57968705 4 | -5.57968706 4 | 1.9-10 | 5.5-11 | 2.1 -9 | 1.3 -9 | 36 | $\infty$ | 5.64 4 | 1.2 7 |
| r1-6-0 | 601 | 600; ; ; | 15 | -5.57968705 4 | -5.57968706 4 | 1.9-10 | 5.5-11 | 2.1 -9 | 1.3 -9 | 44 | $\infty$ | 5.64 4 | 1.2 7 |
| r1-6-1 | 601 | 600; ; ; | 14 | -5.58043922 4 | -5.58043924 4 | 4.3-11 | 1.1-12 | 2.3 -9 | 2.2 -9 | 38 | 3.60 5 | 5.64 4 | 2.7 3 |
| r1-6-1e-6 | 601 | 600; ; ; | 19 | -5.57968731 4 | -5.57968731 4 | 1.1 -8 | 1.9-11 | 1.3-10 | 2.8-10 | 56 | $\infty$ | 5.64 4 | 7.7 5 |
| rose13 | 2379 | 105; ; ; | 31 | 1.20000005 1 | 1.19999999 1 | 5.8-10 | 1.9-11 | 4.9 -9 | 2.4 -8 | 1:13 | $\infty$ | 1.19 2 | 1.1 4 |
| rose15 | 3860 | 135; ; 2; | 47 | 1.40096194-7 | -2.71682585-8 | 4.4 -8 | 6.5-12 | 2.6 -8 | 1.7 -7 | 8:07 | $\infty$ | $\infty$ | 7.1 2 |
| sdmint3 | 5255 | 379; 5255; ; | 37 | -1.28667151 1 | -1.28512439 1 | 7.1 -7 | 2.7 -7 | 2.9 -4 | 5.8 -4 | 30:52 | $\infty$ | 8.86 4 | 2.4 5 |
| shmup1 | 16 | 81; ; 32; | 15 | -1.88414829 2 | -1.88414833 2 | 1.8-11 | 1.7-12 | 9.6 -9 | 9.5 -9 | 01 | 1.50 2 | 1.40 6 | 3.0 3 |
| shmup2 | 200 | 881; ; 400; | 32 | -3.46242679 3 | -3.46242683 3 | 2.9 -9 | 1.1-12 | 5.9 -9 | 5.4 -9 | 32 | 1.35 3 | 3.77 7 | 2.5 5 |
| shmup3 | 420 | 1801; ; 840; | 37 | -2.09883786 3 | -2.09883788 3 | 5.8 -9 | 8.6-13 | 6.0 -9 | 5.5 -9 | 3:28 | 2.74 3 | 2.91 7 | 3.3 5 |
| shmup4 | 800 | 3361; ; 1600; | 55 | -7.99255143 3 | -7.99255154 3 | 8.5 -9 | 1.2-12 | 8.7 -9 | 7.4 -9 | 22:20 | 5.10 3 | 6.73 7 | 4.2 5 |
| taha1a | 3002 | 1680; ; ; | 27 | -9.99980977-1 | -1.00007577 0 | 6.2 -5 | 2.9-10 | 3.6 -5 | 3.2 -5 | 11:30 | $\infty$ | 1.07 10 | 1.4 7 |
| taha1b | 8007 | 1606; ; 3; | 33 | -7.73287084-1 | -7.73322140-1 | 3.2-11 | 1.1 -7 | 1.4 -5 | 1.4 -5 | 22:00 | $\infty$ | 2.52 5 | 1.2 10 |
| trto1 | 36 | 25; ; 36; | 13 | -1.10450000 3 | -1.10450000 3 | 7.3-11 | 1.5-12 | 1.4 -9 | 1.4 -9 | 00 | 1.11 2 | 3.14 4 | 4.8 3 |
| trto2 | 144 | 97; ; 144; | 21 | -1.27999961 4 | -1.28000008 4 | 7.6 -7 | 6.0 -9 | 1.9 -7 | 1.8 -7 | 02 | 2.97 2 | 1.37 6 | 1.0 5 |
| trto3 | 544 | 321; ; 544; | 25 | -1.27999912 4 | -1.28000052 4 | 1.7 -5 | 5.7 -8 | 5.4 -7 | 5.5 -7 | 08 | 1.05 3 | 9.69 6 | 2.7 5 |
| trto4 | 1200 | 673; ; 1200; | 33 | -1.27658074 4 | -1.27658288 4 | 3.8 -6 | 8.8 -8 | 9.2 -7 | 8.4 -7 | 46 | 2.29 3 | 3.74 7 | 5.3 5 |
| trto5 | 3280 | 1761; ; 3280; | 35 | -1.27996987 4 | -1.28000014 4 | 8.9 -5 | 6.7 -7 | 1.3 -5 | 1.2 -5 | 10:45 | 5.99 3 | 2.39 8 | 1.3 6 |
| vibra1 | 36 | 49; ; 36; | 13 | -4.08190123 1 | -4.08190124 1 | 1.5-11 | 1.0-12 | 1.8 -9 | 1.8 -9 | 00 | 1.85 2 | 1.63 4 | 1.0 3 |
| vibra2 | 144 | 193; ; 144; | 25 | -1.66015334 2 | -1.66015365 2 | 3.9 -8 | 7.3-10 | 1.0 -7 | 9.2 -8 | 05 | 4.50 2 | 1.34 6 | 3.3 4 |
| vibra3 | 544 | 641; ; 544; | 32 | -1.72612806 2 | -1.72613080 2 | 3.8 -6 | 2.1 -9 | 8.8 -7 | 7.9 -7 | 15 | 1.55 3 | 1.62 7 | 1.5 5 |
| vibra4 | 1200 | 1345; ; 1200; | 35 | -1.27658137 4 | -1.27658264 4 | 5.9 -6 | 1.0 -7 | 6.3 -7 | 4.9 -7 | 1:32 | 3.39 3 | 7.43 7 | 5.3 5 |
| vibra5 | 3280 | 3521; ; 3280; | 66 | -1.65900646 2 | -1.65903463 2 | 9.7 -6 | 1.7 -8 | 9.3 -6 | 8.5 -6 | 35:15 | 8.69 3 | 1.12 9 | 5.4 5 |
| yalsdp | 5051 | 300; ; ; | 13 | -1.79212672 0 | -1.79212675 0 | 2.1-10 | 2.1-12 | 5.9 -9 | 6.4 -9 | 8:42 | 2.59 3 | 4.99 3 | 3.9 2 |
| checker-1. | 3970 | 3970; ; ; | 23 | 3.30388456 3 | 3.30388454 3 | 1.2-12 | 1.0-12 | 2.9 -9 | 2.9 -9 | 17:12 | 3.97 3 | 1.82 4 | 2.9 3 |
| foot | 2209 | 2208; ; ; | 25 | -5.85293968 5 | -5.85298171 5 | 3.2 -6 | 7.9 -8 | 3.6 -6 | 3.6 -6 | 13:10 | $\infty$ | 5.90 5 | 3.7 8 |
| hand | 1297 | 1296; ; ; | 17 | -2.47477790 4 | -2.47477791 4 | 1.1-11 | 7.6-11 | 1.9 -9 | 1.9 -9 | 4:37 | $\infty$ | 2.48 4 | 2.1 7 |
| inc-600 | 2515 | 600; ; 2514; | 32 | -6.68278108-1 | -6.68550231-1 | 4.5 -8 | 1.7 -9 | 1.2 -4 | 1.2 -4 | 2:33 | $\infty$ | 2.67 5 | 2.4 7 |
| inc-1200 | 5175 | 1200; ; 5174; | 44 | -1.15738763 0 | -1.15747045 0 | 5.0 -8 | 1.6 -9 | 3.2 -5 | 2.5 -5 | 10:25 | $\infty$ | 6.25 5 | 9.6 7 |
| tiger-text | 1802 | 1801; ; ; | 36 | 3.44842540 2 | 3.44360815 2 | 7.3 -8 | 1.6 -8 | 1.0 -3 | 7.0 -4 | 6:59 | $\infty$ | 3.33 3 | 1.0 9 |
| butcher | 6434 | 330; ; 22512; | 53 | -1.39999895 1 | -1.39999999 1 | 3.0 -6 | 6.4-10 | 1.4 -7 | 3.6 -7 | 34:56 | 2.28 4 | $\infty$ | 4.3 3 |
| rabmo | 5004 | 220; ; 6606; | 37 | -3.72725267 0 | -3.72724667 0 | 4.0 -7 | 7.5 -8 | 8.7 -7 | 7.1 -7 | 6:40 | 6.83 3 | $\infty$ | 4.8 2 |
| chs-500 | 9974 | 4980; ; ; | 24 | 7.18222801-10 | -6.43232980-9 | 1.2-15 | 1.0-12 | 7.3 -9 | 7.2 -9 | 12 | 7.35 7 | 6.97 3 | 2.2 1 |
| nonc-500 | 4990 | 2998; ; ; | 23 | 6.25761441-2 | 6.25594864-2 | 5.0-10 | 5.9-11 | 1.5 -5 | 1.5 -5 | 03 | 2.26 4 | 6.00 3 | 6.6 1 |
| ros-500 | 4988 | 2992; ; ; | 17 | 2.49499944 0 | 2.49499939 0 | 8.2-10 | 3.4-12 | 7.1 -9 | 7.8 -9 | 02 | 3.18 5 | 4.49 3 | 1.3 2 |

Table 1: Performance of `sdpt3.m`. In the table, err = [`pinfeas`,`dinfeas`,`relgap`,`relgap2`], where `relgap2` is the same as `relgap` but with the numerator replaced by $|\langle c,\, x\rangle - b^T y|$, and `normXZ` $= \max\{\|x^*\|, \|z^*\|\}$. We declare that $g_P$ ($g_D$) is $\infty$ if the computed number is larger than $10^{12}$.

| problem | $m$ | $n_s; n_q; n_l; n_u$ | it. | primal obj | dual obj | err | | | | time | $g_P$ | $g_D$ | `normXZ` |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| fp210 | 1000 | 176; ; ; 66 | 25 | 3.75000000-1 | 3.75000001-1 | 2.9-11 | 1.3 -9 | 1.2-10 | 7.4-10 | 09 | $\infty$ | $\infty$ | 1.5 1 |
| fp22 | 14 | 15; ; ; | 12 | -7.99999999 0 | -8.00000002 0 | 2.1-11 | 1.3-11 | 2.1 -9 | 1.7 -9 | 00 | 1.14 2 | 1.12 5 | 6.7 1 |
| fp23 | 209 | 119; ; ; | 27 | 2.13000000 2 | 2.13000000 2 | 5.5-12 | 4.2-12 | 1.4 -9 | 1.2 -9 | 01 | $\infty$ | 2.95 3 | 2.3 5 |
| fp24 | 2379 | 595; ; ; | 22 | 1.95000000 2 | 1.94999998 2 | 1.7-11 | 1.0-12 | 7.8 -9 | 7.3 -9 | 52 | $\infty$ | 8.15 3 | 9.6 4 |
| fp25 | 209 | 133; ; ; | 18 | 1.10000000 1 | 1.10000000 1 | 9.9-13 | 9.9-13 | 1.4 -9 | 1.3 -9 | 01 | $\infty$ | 5.94 3 | 6.9 3 |
| fp26 | 1000 | 407; ; ; | 22 | 2.68014632 2 | 2.68014630 2 | 1.5-10 | 1.1-11 | 5.5 -9 | 4.5 -9 | 09 | $\infty$ | 5.18 5 | 3.3 5 |
| fp27 | 1000 | 341; ; ; | 22 | 3.90000002 1 | 3.89999996 1 | 7.4-11 | 6.4-12 | 7.5 -9 | 6.5 -9 | 08 | $\infty$ | 2.43 5 | 8.4 5 |
| fp32 | 3002 | 1155; ; ; | 43 | -7.04918382 0 | -7.04983121 0 | 5.4 -4 | 6.6-10 | 6.1 -5 | 4.3 -5 | 7:38 | $\infty$ | $\infty$ | 1.1 8 |
| fp33 | 125 | 117; ; ; | 42 | -1.01265905 4 | -1.01266044 4 | 1.7 -7 | 2.3 -9 | 8.6 -7 | 6.9 -7 | 01 | $\infty$ | 3.30 10 | 3.4 9 |
| fp34 | 209 | 140; ; ; | 22 | 1.72000000 2 | 1.72000000 2 | 7.5-13 | 1.0-12 | 7.7-10 | 6.4-10 | 01 | $\infty$ | 1.08 4 | 2.6 4 |
| fp35 | 164 | 195; ; ; | 19 | 3.99999999 0 | 3.99999999 0 | 1.3 -9 | 2.2-12 | 2.2 -9 | 6.7-10 | 03 | $\infty$ | 2.85 5 | 5.4 4 |
| fp410 | 14 | 18; ; ; 1 | 17 | 1.67388932 1 | 1.67388932 1 | 2.9 -9 | 3.1 -9 | 3.1-10 | 3.2-10 | 00 | $\infty$ | $\infty$ | 1.6 1 |
| fp42 | 6 | 10; ; ; | 10 | 7.58731237 0 | 7.58731236 0 | 2.2-12 | 2.5-11 | 8.5-10 | 7.8-10 | 00 | 7.19 1 | 9.78 1 | 5.5 1 |
| fp43 | 50 | 76; ; ; | 16 | 6.63500060 2 | 6.63500124 2 | 2.7 -9 | 1.9-10 | 4.2-10 | 4.9 -8 | 01 | 3.44 3 | $\infty$ | 9.1 2 |
| fp44 | 6 | 10; ; ; | 21 | 4.43671688 2 | 4.43671704 2 | 4.8 -9 | 2.4-10 | 2.4 -9 | 1.7 -8 | 00 | $\infty$ | 1.22 3 | 6.8 4 |
| fp45 | 4 | 7; ; ; | 12 | 2.32020677-9 | -8.03431099-10 | 8.1-11 | 2.5-11 | 4.2 -9 | 3.1 -9 | 00 | 8.30 1 | 2.96 1 | 1.9 1 |
| fp46 | 27 | 22; ; ; | 20 | 5.45358326-10 | -2.96042821-9 | 2.8-11 | 1.1-11 | 3.9 -9 | 3.5 -9 | 00 | $\infty$ | 1.49 2 | 2.3 3 |
| fp47 | 6 | 10; ; ; | 13 | 2.42999995 2 | 2.42999999 2 | 1.6 -9 | 2.2 -9 | 9.3 -9 | 8.5 -9 | 00 | 1.52 3 | 7.99 1 | 9.6 2 |
| fp48 | 4 | 7; ; ; | 9 | 7.50000001 0 | 7.49999999 0 | 4.7-11 | 4.3-11 | 9.1-10 | 8.5-10 | 00 | 9.68 1 | 2.96 1 | 1.5 1 |
| fp49 | 14 | 18; ; ; 1 | 17 | 1.67388932 1 | 1.67388932 1 | 2.9 -9 | 3.1 -9 | 3.1-10 | 3.2-10 | 00 | $\infty$ | $\infty$ | 1.6 1 |
| l1 | 14 | 6; ; ; | 9 | 4.92634655-1 | 4.92634654-1 | 9.9-13 | 1.5-12 | 5.6-10 | 5.5-10 | 00 | 1.49 1 | 9.00 0 | 5.7 0 |
| l2 | 14 | 6; ; ; | 8 | 1.14580631 1 | 1.14580631 1 | 2.4-11 | 3.9-10 | 1.4 -9 | 1.1 -9 | 00 | 6.83 1 | 9.00 0 | 1.4 1 |
| l4 | 152 | 45; ; ; | 21 | 3.70371192-2 | 3.70377081-2 | 7.3-11 | 4.9-11 | 5.1 -7 | 5.5 -7 | 01 | $\infty$ | 1.06 4 | 7.9 3 |
| l5 | 14 | 15; ; ; | 12 | -7.99999999 0 | -8.00000002 0 | 2.1-11 | 1.3-11 | 2.1 -9 | 1.7 -9 | 00 | 1.14 2 | 1.12 5 | 6.7 1 |
| 5n | 31 | 26; ; ; | 9 | 2.24000001 0 | 2.23999998 0 | 9.1-11 | 1.0-11 | 5.4 -9 | 5.3 -9 | 00 | 2.82 1 | 2.60 1 | 1.1 1 |
| a12 | 793 | 79; ; ; | 12 | 2.10000000 1 | 2.10000000 1 | 1.1-11 | 1.5-12 | 1.3 -9 | 1.2 -9 | 02 | 1.00 2 | 7.90 1 | 4.4 1 |
| aw29 | 465 | 130; ; ; | 11 | 3.00000000 0 | 3.00000000 0 | 4.7-12 | 2.9-11 | 4.5-10 | 4.4-10 | 07 | 1.33 2 | 1.30 2 | 2.1 1 |
| c5 | 31 | 26; ; ; | 8 | 1.50000000 0 | 1.50000000 0 | 3.0-12 | 3.4-12 | 4.2-10 | 4.2-10 | 00 | 2.75 1 | 2.60 1 | 8.6 0 |
| fp1131 | 847 | 176; ; ; | 11 | 4.50000001 0 | 4.49999995 0 | 3.9-11 | 1.0-12 | 6.3 -9 | 6.2 -9 | 17 | 1.81 2 | 1.76 2 | 6.2 1 |
| fp1132 | 847 | 176; ; ; | 12 | 1.55000000 1 | 1.54999999 1 | 1.0-11 | 7.8-12 | 4.6 -9 | 4.6 -9 | 18 | 1.92 2 | 1.76 2 | 1.4 2 |
| fp1133 | 847 | 176; ; ; | 12 | 1.75000000 1 | 1.74999999 1 | 8.2-12 | 4.4-12 | 2.2 -9 | 2.2 -9 | 18 | 1.94 2 | 1.76 2 | 1.4 2 |
| fp1134 | 847 | 176; ; ; | 12 | 1.95000000 1 | 1.94999998 1 | 3.8-11 | 1.0-12 | 7.0 -9 | 6.9 -9 | 18 | 1.96 2 | 1.76 2 | 1.4 2 |
| fp1135 | 847 | 176; ; ; | 12 | 2.20000000 1 | 2.19999999 1 | 9.4-12 | 3.9-12 | 1.9 -9 | 1.8 -9 | 22 | 1.98 2 | 1.76 2 | 5.8 1 |
| fp1136 | 847 | 176; ; ; | 12 | 1.45000000 1 | 1.44999999 1 | 9.0-12 | 3.8-12 | 3.5 -9 | 3.5 -9 | 19 | 1.91 2 | 1.76 2 | 1.4 2 |
| fp1137 | 847 | 176; ; ; | 12 | 1.65000000 1 | 1.64999999 1 | 7.8-12 | 7.5-12 | 3.0 -9 | 3.0 -9 | 19 | 1.93 2 | 1.76 2 | 1.1 2 |
| fp1138 | 847 | 176; ; ; | 12 | 1.75000000 1 | 1.74999997 1 | 4.6-11 | 1.0-12 | 8.6 -9 | 8.4 -9 | 18 | 1.94 2 | 1.76 2 | 1.4 2 |
| fp1139 | 847 | 176; ; ; | 12 | 2.30000000 1 | 2.29999998 1 | 7.2-12 | 6.6-12 | 4.7 -9 | 4.7 -9 | 18 | 1.99 2 | 1.76 2 | 8.5 1 |
| k5 | 31 | 31; ; ; | 8 | 1.00000000 0 | 9.99999999-1 | 8.5-13 | 2.6-12 | 1.1 -9 | 1.1 -9 | 00 | 3.20 1 | 3.10 1 | 7.0 0 |
| p10 | 847 | 176; ; ; | 11 | 4.50000001 0 | 4.49999995 0 | 4.1-11 | 1.0-12 | 6.3 -9 | 6.2 -9 | 16 | 1.81 2 | 1.76 2 | 6.2 1 |
| bifur | 454 | 84; ; ; 1661 | 28 | -3.37301696-1 | -3.37301694-1 | 2.2 -9 | 1.4 -9 | 3.6 -9 | 9.1-10 | 07 | 3.41 3 | $\infty$ | 4.1 1 |
| boom | 3002 | 210; ; ; 8764 | 36 | -3.23707245 2 | -3.23707245 2 | 4.8 -9 | 7.0-10 | 1.0-10 | 2.8-10 | 4:57 | 1.77 4 | 6.72 11 | 2.8 2 |
| brown | 461 | 56; ; ; 925 | 27 | 2.09326284-10 | 0.00000000-16 | 2.2-11 | 3.6 -9 | 2.1-10 | 2.1-10 | 04 | $\infty$ | $\infty$ | 5.7 1 |
| butcher | 6434 | 330; ; ; 11256 | 59 | -1.39999999 1 | -1.40000000 1 | 2.1 -6 | 7.8-12 | 9.4-10 | 2.1 -9 | 30:44 | 2.28 4 | 1.91 8 | 2.7 3 |
| camera1s | 209 | 28; ; ; 168 | 44 | -1.78686514 4 | -1.78686513 4 | 1.9 -6 | 3.8-11 | 3.6-11 | 4.8 -9 | 01 | 3.64 2 | $\infty$ | 1.3 4 |
| caprasse | 209 | 35; ; ; 60 | 16 | -2.36780177-1 | -2.36780177-1 | 1.2 -9 | 6.1-10 | 7.2-12 | 1.5-10 | 02 | 1.55 2 | $\infty$ | 4.4 0 |
| cdpm5 | 125 | 21; ; ; 5 | 14 | 4.50956437-12 | 6.11518578-9 | 2.8 -9 | 5.0 -9 | 1.2-10 | 6.1 -9 | 00 | 3.10 1 | $\infty$ | 3.9 0 |
| chemequ | 461 | 56; ; ; 525 | 16 | -2.77820337 7 | -1.62704475 7 | dual infeasible | | | | 04 | | | |
| chemequs | 125 | 21; ; ; 45 | 9 | -5.75312872 8 | -6.24933457 6 | dual infeasible | | | | 00 | | | |
| cohn2 | 209 | 35; ; ; 4 | 36 | 4.57937090-9 | 1.29980577-7 | 3.8 -7 | 6.9 -7 | 1.4 -7 | 1.3 -7 | 06 | 4.30 1 | $\infty$ | 3.9 0 |
| cohn3 | 209 | 35; ; ; 4 | 35 | 7.86066516-9 | 8.98299495-9 | 2.1 -6 | 3.4 -7 | 2.3 -7 | 1.1 -9 | 06 | 4.30 1 | $\infty$ | 4.2 0 |
| comb3000 | 1000 | 66; ; ; 595 | 25 | -4.80138795-10 | 1.17568288-9 | 6.3-12 | 4.8 -9 | 3.2-10 | 1.7 -9 | 09 | 1.26 3 | 4.88 9 | 6.5 0 |
| conform1 | 83 | 20; ; ; 30 | 9 | -1.69723199 7 | -1.79353139 6 | dual infeasible | | | | 01 | | | |
| conform3 | 285 | 56; ; ; 630 | 21 | 1.01682942-13 | 0.00000000-16 | 2.0-13 | 3.6-10 | 2.2-11 | 1.0-13 | 02 | $\infty$ | $\infty$ | 2.5 0 |
| conform4 | 454 | 84; ; ; 1890 | 21 | 4.05675465-11 | 0.00000000-16 | 3.4-11 | 4.5 -9 | 4.4-10 | 4.1-11 | 06 | $\infty$ | $\infty$ | 9.3 0 |
| des22-24 | 1000 | 66; ; ; 660 | 37 | -6.74166365 3 | -6.74166365 3 | 2.0 -8 | 4.1-12 | 2.1-13 | 1.4-12 | 14 | 1.39 3 | 2.27 10 | 6.7 3 |
| discret3 | 44 | 9; ; ; 8 | 24 | -3.70033109 1 | -3.70033091 1 | 2.4 -7 | 1.5 -8 | 4.0-10 | 2.4 -8 | 00 | 2.50 1 | $\infty$ | 6.9 2 |
| eco5 | 461 | 56; ; ; 525 | 26 | -1.20463311 3 | -1.20463311 3 | 3.2 -9 | 5.8 -9 | 1.4-10 | 1.2-10 | 03 | 1.11 3 | 5.21 10 | 1.2 3 |
| eco6 | 923 | 84; ; ; 924 | 37 | -1.00281559 4 | -1.00281559 4 | 6.5 -9 | 1.3 -9 | 1.4-10 | 9.8-11 | 14 | 1.93 3 | 5.76 10 | 1.0 4 |
| eco7 | 1715 | 120; ; ; 1512 | 37 | -3.91531047 3 | -3.91531047 3 | 5.3 -9 | 2.8 -9 | 1.7-10 | 4.2-12 | 47 | 3.14 3 | 1.28 11 | 3.9 3 |
| eco8 | 3002 | 165; ; ; 2340 | 37 | -5.82038418 3 | -5.82038418 3 | 5.2 -9 | 3.3 -9 | 1.2-10 | 2.6-11 | 4:19 | 4.85 3 | 2.96 11 | 5.8 3 |
| fourbar | 69 | 15; ; ; 4 | 14 | 1.01084283-12 | 1.17629630-9 | 1.9-10 | 7.4-10 | 1.9-11 | 1.2 -9 | 00 | 2.30 1 | $\infty$ | 3.3 0 |
| geneig | 923 | 84; ; ; 546 | 25 | -2.52663014 0 | -2.52663014 0 | 2.2-10 | 2.3 -9 | 3.6-10 | 2.5-10 | 09 | 1.18 3 | 1.38 11 | 8.9 0 |

Table 1: Performance of `sdpt3.m`. In the table, err = [`pinfeas`,`dinfeas`,`relgap`,`relgap2`], where `relgap2` is the same as `relgap` but with the numerator replaced by $|\langle c,\, x\rangle - b^T y|$, and `normXZ` = $\max\{\|x^*\|, \|z^*\|\}$. We declare that $g_P$ ($g_D$) is $\infty$ if the computed number is larger than $10^{12}$.

| problem | $m$ | $n_s; n_q; n_l; n_u$ | it. | primal obj | dual obj | err | | | | time | $g_P$ | $g_D$ | `normXZ` |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| heart | 3002 | 165; ; ; 4320 | 32 | -8.70927420 1 | -8.70927421 1 | 7.2 -9 | 3.7 -9 | 2.8-10 | 4.9-10 | 3:41 | 8.81 3 | 2.79 11 | 1.4 2 |
| i1 | 1000 | 66; ; ; 10 | 17 | -1.66775272 0 | -1.66775269 0 | 3.2 -9 | 5.1 -9 | 7.8-10 | 6.2 -9 | 06 | 8.60 1 | $\infty$ | 9.4 0 |
| ipp | 494 | 45; ; ; 360 | 22 | -1.31158853 1 | -1.31158853 1 | 4.2 -9 | 5.8 -9 | 2.4-10 | 3.1-11 | 02 | 7.65 2 | $\infty$ | 1.7 1 |
| katsura5 | 209 | 28; ; ; 168 | 20 | -8.16044579-2 | -8.16044568-2 | 2.3-10 | 1.3 -9 | 9.7-11 | 9.1-10 | 01 | 3.64 2 | $\infty$ | 5.4 0 |
| kinema | 714 | 55; ; ; 495 | 37 | -4.19683963 4 | -4.19683963 4 | 1.2 -7 | 2.9-12 | 1.7-12 | 2.6-13 | 07 | 1.05 3 | $\infty$ | 4.2 4 |
| ku10 | 1000 | 66; ; ; 660 | 34 | -7.13900000 3 | -7.13900000 3 | 1.5 -7 | 3.8-10 | 2.8-11 | 1.5-10 | 12 | 1.39 3 | $\infty$ | 7.1 3 |
| lorentz | 69 | 15; ; ; 60 | 17 | -5.00000000 0 | -4.99999997 0 | 8.1-10 | 6.3 -9 | 7.7-11 | 2.0 -9 | 00 | 1.35 2 | $\infty$ | 6.1 0 |
| manocha | 90 | 28; ; ; 42 | 36 | -2.46011903-1 | -2.46013262-1 | 3.0 -5 | 7.5 -9 | 3.3 -8 | 9.1 -7 | 02 | 1.12 2 | $\infty$ | 1.4 3 |
| noon3 | 83 | 20; ; ; 30 | 18 | -2.08695033 1 | -2.08695033 1 | 1.1 -9 | 1.0 -9 | 2.3-11 | 4.7-12 | 01 | 8.00 1 | $\infty$ | 2.6 1 |
| noon4 | 209 | 35; ; ; 60 | 18 | -1.71283759 1 | -1.71283759 1 | 9.6-10 | 1.1 -9 | 2.6-11 | 7.3-11 | 01 | 1.55 2 | $\infty$ | 1.9 1 |
| noon5 | 461 | 56; ; ; 105 | 18 | -1.58524243 1 | -1.58524243 1 | 7.5-10 | 1.5 -9 | 1.9-11 | 3.9-11 | 02 | 2.66 2 | $\infty$ | 1.7 1 |
| proddeco | 69 | 15; ; ; 4 | 18 | 1.29678458-11 | 4.63789541-10 | 1.8-10 | 8.9-10 | 1.5-10 | 4.5-10 | 00 | 2.30 1 | $\infty$ | 3.3 0 |
| puma | 3002 | 165; ; ; 8280 | 31 | -3.05299489 1 | -3.05299489 1 | 2.9 -9 | 9.8-11 | 5.5-11 | 3.8-10 | 3:38 | 1.67 4 | 9.28 11 | 3.4 1 |
| quadfor2 | 209 | 35; ; ; 270 | 19 | -6.18518518 0 | -6.18518518 0 | 1.3 -9 | 7.9-10 | 4.1-11 | 1.1-10 | 03 | 5.75 2 | $\infty$ | 1.9 1 |
| quadgrid | 461 | 56; ; ; 505 | 17 | -1.08801588 7 | -8.42937324 6 | dual infeasible | | | | 02 | | | |
| rabmo | 5004 | 220; ; ; 3303 | 42 | -3.72725305 0 | -3.72725176 0 | 6.7 -8 | 1.7 -8 | 1.1 -7 | 1.5 -7 | 7:09 | 6.83 3 | $\infty$ | 4.5 2 |
| rbpl | 923 | 84; ; ; 546 | 26 | -7.94063377 0 | -7.94063377 0 | 2.6 -9 | 7.5-10 | 6.7-11 | 2.3-10 | 10 | 1.18 3 | $\infty$ | 3.1 1 |
| redeco5 | 20 | 6; ; ; 5 | 13 | -2.53906248-1 | -2.53906249-1 | 2.3 -9 | 7.6-10 | 4.6-11 | 8.9-10 | 00 | 1.60 1 | $\infty$ | 2.4 0 |
| redeco6 | 27 | 7; ; ; 6 | 13 | -2.01599999-1 | -2.01599999-1 | 1.5 -9 | 7.7-10 | 2.8-11 | 4.2-10 | 00 | 1.90 1 | $\infty$ | 2.6 0 |
| redeco7 | 35 | 8; ; ; 7 | 13 | -1.67438272-1 | -1.67438268-1 | 7.4-11 | 5.9 -9 | 9.1-11 | 2.3 -9 | 00 | 2.20 1 | $\infty$ | 2.7 0 |
| redeco8 | 44 | 9; ; ; 8 | 13 | -1.43273635-1 | -1.43273635-1 | 1.6 -9 | 6.1-10 | 1.5-11 | 3.0-10 | 00 | 2.50 1 | $\infty$ | 2.9 0 |
| rediff3 | 9 | 4; ; ; 3 | 16 | 7.61297271-13 | 3.66436581-9 | 1.1 -9 | 2.5 -9 | 8.4-12 | 3.7 -9 | 00 | 1.00 1 | $\infty$ | 1.7 0 |
| rose | 679 | 120; ; ; 2281 | 53 | -1.74379054 0 | -1.74376346 0 | 2.5 -4 | 4.7 -9 | 1.2 -4 | 6.0 -6 | 29 | 4.68 3 | $\infty$ | 2.8 5 |
| s9-1 | 494 | 45; ; ; 360 | 22 | -4.27369563 0 | -4.27369564 0 | 5.5 -9 | 1.6 -9 | 4.3-10 | 1.4 -9 | 02 | 7.65 2 | $\infty$ | 9.4 0 |
| sendra | 65 | 21; ; ; 12 | 21 | -2.37687542 1 | -2.37687542 1 | 5.9 -9 | 4.4-10 | 1.6-11 | 9.2-11 | 01 | 4.50 1 | $\infty$ | 4.3 1 |
| solotarev | 69 | 15; ; ; 32 | 18 | -5.88961333 0 | -5.88961333 0 | 5.7-10 | 3.1-10 | 7.7-12 | 9.8-11 | 01 | 7.90 1 | $\infty$ | 1.1 1 |
| stewart1 | 714 | 55; ; ; 495 | 28 | -8.76585278 0 | -8.76585278 0 | 8.6 -9 | 2.0-10 | 1.1-11 | 3.0-11 | 05 | 1.05 3 | $\infty$ | 2.1 1 |
| stewart2 | 1819 | 91; ; ; 910 | 28 | -1.27531385 1 | -1.27531386 1 | 9.9 -9 | 7.6-10 | 2.3-10 | 1.8 -9 | 34 | 1.91 3 | $\infty$ | 1.8 1 |
| trinks | 209 | 28; ; ; 141 | 27 | -2.43523491-1 | -2.43523224-1 | 6.2 -9 | 8.8 -9 | 1.0 -8 | 1.8 -7 | 01 | 3.10 2 | $\infty$ | 2.8 1 |
| visasoro | 44 | 9; ; ; 8 | 15 | 1.73075654-13 | 1.66876819-9 | 1.0 -9 | 6.8-10 | 4.7-12 | 1.7 -9 | 00 | 2.50 1 | $\infty$ | 2.8 0 |
| wood | 69 | 15; ; ; 32 | 18 | -6.64233344-2 | -6.64233342-2 | 1.4-11 | 1.4 -9 | 1.5-11 | 1.7-10 | 01 | 7.90 1 | $\infty$ | 3.7 0 |
| wright | 20 | 6; ; ; 5 | 17 | -2.00000000 1 | -1.99999999 1 | 2.0 -9 | 6.2 -9 | 1.3 -9 | 1.1 -9 | 00 | 1.60 1 | $\infty$ | 2.1 1 |
| nql30o | 3680 | ; 2700; 3602; | 37 | -9.46028486-1 | -9.46028499-1 | 1.0 -9 | 1.6-10 | 7.1 -9 | 4.4 -9 | 04 | 5.40 3 | $\infty$ | 5.4 1 |
| nql60o | 14560 | ; 10800; 14402; | 42 | -9.35052921-1 | -9.35052943-1 | 1.4 -9 | 1.4-10 | 8.2 -9 | 7.6 -9 | 27 | 2.16 4 | $\infty$ | 1.1 2 |
| nql90o | 32640 | ; 24300; 32402; | 49 | -9.31383156-1 | -9.31383163-1 | 5.1-10 | 3.2-11 | 6.7 -9 | 2.4 -9 | 1:01 | 4.86 4 | $\infty$ | 1.6 2 |
| nql120o | 57920 | ; 43200; 57602; | 51 | -9.29550226-1 | -9.29550233-1 | 4.9 -9 | 2.5-11 | 7.4 -9 | 2.4 -9 | 2:03 | 8.64 4 | $\infty$ | 2.1 2 |
| nql180o | 130080 | ; 97200; 129602; | 60 | -9.27728615-1 | -9.27728621-1 | 5.1 -9 | 1.0-11 | 7.6 -9 | 2.0 -9 | 5:07 | 1.94 5 | $\infty$ | 3.2 2 |
| qs30o | 1861 | ; 3844; 2; | 22 | -6.29531577 0 | -6.29531562 0 | 2.6 -8 | 1.2 -9 | 2.0-12 | 1.1 -8 | 02 | 1.92 3 | $\infty$ | 4.4 1 |
| qs60o | 7321 | ; 14884; 2; | 28 | -6.38210431 0 | -6.38210377 0 | 5.0 -8 | 2.2 -9 | 1.1-10 | 3.9 -8 | 10 | 7.44 3 | $\infty$ | 8.7 1 |
| qs90o | 16381 | ; 33124; 2; | 30 | -6.42377450 0 | -6.42377361 0 | 5.7 -8 | 2.4 -9 | 5.5-10 | 6.5 -8 | 27 | 1.66 4 | $\infty$ | 1.3 2 |
| qs120o | 29041 | ; 58564; 2; | 31 | -6.45014648 0 | -6.45014399 0 | 1.3 -7 | 5.1 -9 | 2.5-10 | 1.8 -7 | 53 | 2.93 4 | $\infty$ | 1.7 2 |
| qs180o | 65161 | ; 131044; 2; | 34 | -6.48351741 0 | -6.48351169 0 | 2.0 -7 | 7.9 -9 | 2.5-10 | 4.1 -7 | 2:36 | 6.55 4 | 1.33 11 | 2.6 2 |
| q30o | 7482 | ; 11163; 2; | 35 | -9.36404974-1 | -9.36405064-1 | 5.5 -9 | 5.2-10 | 1.1 -8 | 3.1 -8 | 29 | 7.44 3 | $\infty$ | 7.8 1 |
| q60o | 29362 | ; 43923; 2; | 44 | -9.44560086-10 | -2.85627290-6 | 2.6 -7 | 6.8 -9 | 7.9 -9 | 2.9 -6 | 3:25 | 2.93 4 | $\infty$ | 1.3 2 |
| dsNRL | 406 | ; 15897; ; | 35 | -5.57425079-5 | -5.57492923-5 | 3.3-12 | 1.0-12 | 6.8 -9 | 6.8 -9 | 10:56 | 1.05 4 | 2.10 4 | 1.9 2 |
| firL1Linfa | 6224 | ; 35532; ; | 30 | -2.56478569-3 | -2.56479294-3 | 1.8-11 | 3.3-12 | 7.2 -9 | 7.2 -9 | 17:40 | 1.18 5 | 2.38 4 | 1.5 0 |
| firL1Linfe | 5685 | ; 11172; 1; | 42 | -3.31229666-3 | -3.31231116-3 | 1.1-12 | 3.6-12 | 1.4 -8 | 1.4 -8 | 48 | 1.53 4 | 3.64 4 | 5.7 0 |
| firL1 | 6223 | ; 17766; ; | 22 | -2.92575478-4 | -2.92580521-4 | 1.1-11 | 1.6-12 | 5.0 -9 | 5.0 -9 | 7:44 | 2.41 4 | 1.18 4 | 4.7 -1 |
| firL2a | 2002 | ; 2003; ; | 7 | -5.07008269-4 | -5.07012684-4 | 7.4-14 | 9.5-10 | 5.9 -9 | 4.4 -9 | 3:00 | 2.00 0 | 2.00 0 | 1.4 0 |
| firL2L1alp | 5868 | ; 9611; 1; | 15 | -5.76343274-5 | -5.76374679-5 | 7.1-12 | 3.9-12 | 3.1 -9 | 3.1 -9 | 23 | 1.19 4 | 3.85 3 | 1.8 0 |
| firL2L1eps | 8303 | ; 24108; ; | 23 | -5.35470923-4 | -5.35471575-4 | 2.3-13 | 1.0-12 | 6.5-10 | 6.5-10 | 19:06 | 3.90 4 | 1.58 4 | 3.1 -1 |
| firL2Linfa | 303 | ; 13629; ; | 27 | -6.79117295-3 | -6.79117919-3 | 5.2-12 | 1.2-12 | 6.2 -9 | 6.2 -9 | 4:05 | 1.78 4 | 8.93 3 | 7.7 -1 |
| firL2Linfe | 6086 | ; 14711; ; | 17 | -1.48919871-3 | -1.48920537-3 | 9.4-11 | 5.2-12 | 7.0 -9 | 6.6 -9 | 1:53 | 5.89 3 | 5.83 5 | 1.1 2 |
| firL2 | 102 | ; 103; ; | 7 | -3.11866437-3 | -3.11866351-3 | 4.7-14 | 8.8-10 | 4.7-10 | 8.6-10 | 00 | 2.00 0 | 2.00 0 | 1.4 0 |
| firLinf | 402 | ; 11886; ; | 23 | -1.00681681-2 | -1.00681770-2 | 8.6-10 | 3.8-11 | 8.7 -9 | 8.7 -9 | 5:20 | 7.92 3 | 7.96 3 | 8.1 -1 |
| wbNRL | 460 | ; 1578; 17177; | 30 | -4.15006324-5 | -4.15040502-5 | 1.9-11 | 1.0-12 | 3.4 -9 | 3.4 -9 | 14:14 | 1.72 4 | 6.51 5 | 7.3 0 |
| BeH-2Sigma | 948 | 1406; ; ; | 30 | 1.66935640 1 | 1.66935639 1 | 1.4-12 | 1.0-12 | 3.6 -9 | 3.6 -9 | 3:29 | 1.43 3 | 1.34 10 | 2.4 1 |
| BH-1Sigma+ | 948 | 1406; ; ; | 30 | 2.72063377 1 | 2.72063375 1 | 7.0-12 | 1.2-12 | 4.0 -9 | 3.9 -9 | 3:28 | 1.44 3 | 3.71 10 | 2.5 1 |
| BH2-2A1-ST | 1743 | 2166; ; ; | 30 | 3.04301167 1 | 3.04301166 1 | 3.7-10 | 1.6-11 | 1.0 -9 | 9.1-10 | 15:16 | 2.20 3 | 4.79 9 | 2.1 2 |
| BH+-2Sigma | 948 | 1406; ; ; | 29 | 2.69796660 1 | 2.69796657 1 | 1.6-12 | 1.0-12 | 4.7 -9 | 4.7 -9 | 3:23 | 1.44 3 | 1.34 10 | 2.4 1 |
| CH+-1Sigma | 948 | 1406; ; ; | 29 | 4.06927879 1 | 4.06927873 1 | 2.4-12 | 1.0-12 | 7.9 -9 | 7.9 -9 | 3:22 | 1.45 3 | 3.71 10 | 2.6 1 |
| CH2-1A1-ST | 1743 | 2166; ; ; | 28 | 4.48537630 1 | 4.48537625 1 | 4.6-10 | 1.3-10 | 5.8 -9 | 5.2 -9 | 14:17 | 2.21 3 | 4.95 9 | 6.2 1 |

Table 1: Performance of `sdpt3.m`. In the table, err = [`pinfeas`,`dinfeas`,`relgap`,`relgap2`], where `relgap2` is the same as `relgap` but with the numerator replaced by $|\langle c,\, x\rangle - b^T y|$, and `normXZ` $= \max\{\|x^*\|, \|z^*\|\}$. We declare that $g_P$ ($g_D$) is $\infty$ if the computed number is larger than $10^{12}$.

| problem | $m$ | $n_s; n_q; n_l; n_u$ | it. | primal obj | dual obj | err | | | | time | $g_P$ | $g_D$ | `normXZ` |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CH2-3B1-ST | 1743 | 2166; ; ; | 29 | 4.50291329 1 | 4.50291327 1 | 2.8-10 | 3.8-11 | 1.7 -9 | 1.5 -9 | 14:45 | 2.22 3 | 4.73 9 | 1.3 2 |
| CH-2Pi-STO | 948 | 1406; ; ; | 28 | 4.10222178 1 | 4.10222176 1 | 2.6-11 | 4.4-12 | 2.3 -9 | 2.2 -9 | 3:18 | 1.45 3 | 1.59 10 | 4.4 1 |
| CH–3Sigma | 948 | 1406; ; ; | 28 | 4.09070913 1 | 4.09070909 1 | 8.4-12 | 1.5-12 | 5.5 -9 | 5.5 -9 | 3:16 | 1.45 3 | 1.66 10 | 3.2 1 |
| H2O-1A1-ST | 1743 | 2166; ; ; | 28 | 8.49236907 1 | 8.49236900 1 | 3.7-12 | 2.0-12 | 3.9 -9 | 3.9 -9 | 14:15 | 2.24 3 | 6.09 9 | 4.4 1 |
| H2O+-2B1-S | 1743 | 2166; ; ; | 29 | 8.42163764 1 | 8.42163759 1 | 4.6-11 | 1.1-11 | 2.8 -9 | 2.8 -9 | 14:47 | 2.24 3 | 5.49 9 | 6.7 1 |
| HF-1Sigma+ | 948 | 1406; ; ; | 27 | 1.04720454 2 | 1.04720452 2 | 2.5-12 | 1.0-12 | 8.7 -9 | 8.7 -9 | 3:08 | 1.49 3 | 1.74 9 | 3.7 1 |
| HF+-2Pi-ST | 948 | 1406; ; ; | 26 | 1.03885668 2 | 1.03885666 2 | 3.2-11 | 1.0-12 | 9.4 -9 | 9.5 -9 | 3:01 | 1.49 3 | 1.75 10 | 3.2 1 |
| LiH-1Sigma | 948 | 1406; ; ; | 29 | 8.96721198 0 | 8.96721180 0 | 9.1-12 | 2.5-12 | 9.2 -9 | 9.2 -9 | 3:23 | 1.43 3 | 3.71 10 | 2.2 1 |
| NH2-2B1-ST | 1743 | 2166; ; ; | 29 | 6.29798018 1 | 6.29798015 1 | 7.5-11 | 1.7-11 | 2.7 -9 | 2.7 -9 | 14:49 | 2.23 3 | 5.49 9 | 6.8 1 |
| NH+-2Pi-ST | 948 | 1406; ; ; | 28 | 5.78593622 1 | 5.78593619 1 | 7.1-12 | 5.0-12 | 2.4 -9 | 2.4 -9 | 3:16 | 1.46 3 | 1.59 10 | 4.0 1 |
| NH–2Pi-ST | 948 | 1406; ; ; | 28 | 5.80546396 1 | 5.80546388 1 | 9.7-11 | 1.0-12 | 6.9 -9 | 6.9 -9 | 3:15 | 1.46 3 | 1.75 10 | 3.2 1 |
| NH-3Sigma- | 948 | 1406; ; ; | 27 | 5.83910025 1 | 5.83910016 1 | 3.4-11 | 1.6-12 | 7.7 -9 | 7.7 -9 | 3:08 | 1.46 3 | 1.66 10 | 3.2 1 |
| OH–1Sigma | 948 | 1406; ; ; | 29 | 7.91680602 1 | 7.91680586 1 | 1.5-12 | 1.5-12 | 1.0 -8 | 1.0 -8 | 3:23 | 1.48 3 | 1.74 9 | 3.7 1 |
| OH-2Pi-STO | 948 | 1406; ; ; | 27 | 7.94670771 1 | 7.94670763 1 | 2.3-12 | 1.0-12 | 5.4 -9 | 5.4 -9 | 3:10 | 1.48 3 | 1.75 10 | 3.2 1 |
| OH+-3Sigma | 948 | 1406; ; ; | 27 | 7.88863798 1 | 7.88863789 1 | 1.8-11 | 1.5-12 | 5.6 -9 | 5.6 -9 | 3:09 | 1.48 3 | 1.66 10 | 3.2 1 |
| Li.2S.STO6 | 465 | 780; ; ; 35 | 35 | 7.40023852 0 | 7.40023828 0 | 4.1 -9 | 5.4-10 | 2.4 -8 | 1.5 -8 | 46 | 7.86 2 | ∞ | 1.6 1 |
| Be.1S.STO6 | 465 | 780; ; ; 35 | 38 | 1.45560898 1 | 1.45560885 1 | 4.7 -8 | 7.9-10 | 4.1 -8 | 4.1 -8 | 53 | 7.87 2 | ∞ | 2.4 1 |
| BeH+.1Sigm | 948 | 1312; ; ; 47 | 40 | 1.64575096 1 | 1.64575071 1 | 8.3 -8 | 8.4-10 | 1.0 -7 | 7.4 -8 | 4:17 | 1.32 3 | ∞ | 5.8 1 |
| H3.2A1.DZ. | 948 | 1312; ; ; 47 | 38 | 3.36465433 0 | 3.36464799 0 | 5.2 -8 | 5.1 -9 | 1.6 -6 | 8.2 -7 | 3:59 | 1.32 3 | ∞ | 4.9 1 |
| FH2+.1A1.S | 1743 | 2044; ; ; 61 | 42 | 1.09990409 2 | 1.09990401 2 | 3.1 -8 | 3.5-10 | 5.1 -8 | 3.4 -8 | 20:37 | 2.06 3 | ∞ | 1.7 2 |
| NH2-.1A1.S | 1743 | 2044; ; ; 61 | 41 | 6.27062174 1 | 6.27062139 1 | 2.7 -8 | 2.9-10 | 5.0 -8 | 2.7 -8 | 20:15 | 2.06 3 | ∞ | 1.2 2 |
| quadknap-1 | 5984 | 189; ; 5814; | 45 | 1.20467277 3 | 1.20467277 3 | 1.6 -8 | 1.2-13 | 2.1-10 | 1.6-10 | 9:21 | 1.61 4 | ∞ | 2.1 4 |
| quadknap-1 | 5984 | 189; ; 5814; | 35 | 4.92735912 3 | 4.92736806 3 | 8.9 -7 | 4.8 -8 | 3.5 -6 | 9.1 -7 | 7:26 | 1.59 4 | ∞ | 1.9 4 |
| quadknap-1 | 5984 | 189; ; 5814; | 37 | 4.84881498 3 | 4.84882347 3 | 9.5 -7 | 3.8 -8 | 2.5 -6 | 8.8 -7 | 11:18 | 1.61 4 | ∞ | 2.1 4 |
| quadknap-1 | 5984 | 189; ; 5814; | 44 | 7.88501507 2 | 7.88501506 2 | 7.7 -8 | 1.9-13 | 5.7-10 | 3.1-10 | 12:07 | 1.70 4 | ∞ | 3.2 4 |
| quadknap-1 | 5984 | 189; ; 5814; | 39 | 5.37514786 3 | 5.37515599 3 | 7.9 -7 | 3.8 -8 | 1.6 -6 | 7.6 -7 | 11:56 | 1.70 4 | ∞ | 1.9 4 |
| quadknap-1 | 5984 | 189; ; 5814; | 39 | 2.40839711 3 | 2.40839758 3 | 2.5 -7 | 1.3 -8 | 3.3 -7 | 9.8 -8 | 8:13 | 1.51 4 | ∞ | 7.9 3 |
| quadknap-1 | 5984 | 189; ; 5814; | 39 | 8.04800002 3 | 8.04800000 3 | 5.4 -8 | 1.2-11 | 4.3-10 | 9.6-10 | 8:17 | 1.87 4 | ∞ | 1.1 4 |
| quadknap-1 | 5984 | 189; ; 5814; | 38 | 5.85100008 3 | 5.85100000 3 | 2.9 -8 | 5.8-11 | 7.7-10 | 6.5 -9 | 8:02 | 1.58 4 | ∞ | 9.3 3 |
| quadknap-1 | 5984 | 189; ; 5814; | 37 | 5.13259190 3 | 5.13259584 3 | 2.0 -7 | 6.2 -8 | 3.7 -7 | 3.8 -7 | 7:46 | 1.63 4 | ∞ | 2.1 4 |
| quadknap-1 | 5984 | 189; ; 5814; | 40 | 6.77500003 3 | 6.77500000 3 | 1.5 -6 | 3.6-11 | 9.6-10 | 1.8 -9 | 8:22 | 1.94 4 | ∞ | 1.8 4 |
| stable-17- | 5984 | 477; ; 342; | 29 | -1.42857143-1 | -1.42857142-1 | 3.6-10 | 6.2-10 | 1.3 -9 | 3.2-10 | 6:20 | ∞ | ∞ | 6.9 1 |
| stable-17- | 5984 | 477; ; 342; | 30 | -1.98434223-1 | -1.98434223-1 | 7.5-11 | 1.9-11 | 3.4-10 | 7.7-11 | 6:33 | ∞ | ∞ | 1.1 2 |
| stable-17- | 5984 | 477; ; 342; | 28 | -1.66666666-1 | -1.66666666-1 | 2.6-10 | 7.2-10 | 1.2 -9 | 6.5-10 | 6:13 | ∞ | ∞ | 6.8 1 |
| stable-17- | 5984 | 477; ; 342; | 36 | -1.66131915-1 | -1.66131921-1 | 4.2 -9 | 2.8-11 | 9.9 -9 | 3.9 -9 | 7:48 | ∞ | ∞ | 1.6 2 |
| stable-17- | 5984 | 477; ; 342; | 34 | -1.95961595-1 | -1.95961595-1 | 7.6-10 | 4.7-12 | 6.1-10 | 1.2-12 | 7:15 | ∞ | ∞ | 1.6 2 |
| stable-17- | 5984 | 477; ; 342; | 36 | -1.95658388-1 | -1.95658390-1 | 7.8-10 | 1.9-11 | 3.8 -9 | 1.4 -9 | 7:36 | ∞ | ∞ | 1.8 2 |
| stable-17- | 5984 | 477; ; 342; | 39 | -1.66561011-1 | -1.66561013-1 | 2.8 -8 | 6.6-13 | 5.1-10 | 1.3 -9 | 9:03 | ∞ | ∞ | 1.7 2 |
| stable-17- | 5984 | 477; ; 342; | 28 | -1.66666667-1 | -1.66666666-1 | 1.2-10 | 3.9-10 | 5.3-10 | 4.0-10 | 6:04 | ∞ | ∞ | 6.6 1 |
| stable-17- | 5984 | 477; ; 342; | 32 | -1.66138271-1 | -1.66138271-1 | 9.8-11 | 1.6-11 | 6.8-10 | 2.5-10 | 7:01 | ∞ | ∞ | 1.4 2 |
| stable-17- | 5984 | 477; ; 342; | 29 | -1.66138270-1 | -1.66138270-1 | 6.6-10 | 8.8-10 | 4.9 -9 | 3.5-10 | 6:05 | ∞ | ∞ | 9.9 1 |
| MaxCut-100 | 6252 | 1850; ; 4188; | 35 | 1.46072620 2 | 1.46067965 2 | 1.2 -5 | 2.9 -6 | 8.7 -5 | 1.6 -5 | 50 | 9.59 3 | ∞ | 1.7 3 |
| MaxCut-100 | 6252 | 1850; ; 4188; | 35 | 1.46072620 2 | 1.46067965 2 | 1.2 -5 | 2.9 -6 | 8.7 -5 | 1.6 -5 | 50 | 9.59 3 | ∞ | 1.7 3 |
| MaxCut-100 | 7767 | 2134; ; 4590; | 44 | 1.48043435 2 | 1.48043439 2 | 5.8 -6 | 3.9-10 | 3.7 -9 | 1.2 -8 | 3:00 | 1.09 4 | ∞ | 1.1 3 |
| MaxCut-100 | 5679 | 1775; ; 3876; | 36 | 1.47065095 2 | 1.47063793 2 | 1.4 -5 | 2.8 -6 | 3.4 -5 | 4.4 -6 | 42 | 9.03 3 | ∞ | 1.7 3 |
| MaxCut-100 | 6717 | 1877; ; 4476; | 36 | 1.34059930 2 | 1.34057081 2 | 2.1 -5 | 3.0 -6 | 7.9 -5 | 1.1 -5 | 1:46 | 9.95 3 | ∞ | 2.2 3 |
| MaxCut-100 | 6059 | 1759; ; 4044; | 34 | 1.47028456 2 | 1.47028478 2 | 4.1 -6 | 9.3 -7 | 5.3 -6 | 7.5 -8 | 1:21 | 9.19 3 | ∞ | 6.7 2 |
| MaxCut-100 | 7221 | 2103; ; 4900; | 35 | 1.48038276 2 | 1.48036086 2 | 2.0 -5 | 4.3 -6 | 7.2 -5 | 7.4 -6 | 1:59 | 1.10 4 | ∞ | 2.8 3 |
| MaxCut-100 | 7375 | 2121; ; 4400; | 35 | 1.47051236 2 | 1.47051241 2 | 1.7 -6 | 3.2 -7 | 2.1 -6 | 1.6 -8 | 1:22 | 1.06 4 | ∞ | 7.4 2 |
| MaxCut-100 | 6495 | 1937; ; 4328; | 35 | 1.34096796 2 | 1.34094792 2 | 1.3 -5 | 1.9 -6 | 4.6 -5 | 7.4 -6 | 1:37 | 9.95 3 | ∞ | 1.9 3 |
| MaxCut-100 | 7228 | 1923; ; 4816; | 35 | 1.45015592 2 | 1.45011927 2 | 2.3 -5 | 3.4 -6 | 8.8 -5 | 1.3 -5 | 2:04 | 1.05 4 | ∞ | 2.3 3 |
| Bex2-1-1.g | 251 | 252; ; 182; | 22 | 3.40000004-1 | 3.39999990-1 | 5.1-10 | 1.9-11 | 9.6 -9 | 8.5 -9 | 02 | 6.21 2 | 9.14 4 | 3.3 1 |
| Bex2-1-2.g | 65 | 80; ; 52; | 17 | 1.06500000 0 | 1.06500000 0 | 4.9-13 | 1.0-12 | 5.9-10 | 5.8-10 | 00 | 1.75 2 | 3.34 3 | 1.0 1 |
| Bex2-1-3.g | 134 | 193; ; 113; | 18 | 1.87500000 0 | 1.87500000 0 | 4.0-13 | 1.0-12 | 4.9-10 | 4.8-10 | 01 | 4.44 2 | 7.70 4 | 2.8 1 |
| Bex2-1-4.g | 83 | 126; ; 61; | 17 | 1.37500001 0 | 1.37499998 0 | 1.6-12 | 1.0-12 | 7.8 -9 | 7.8 -9 | 00 | 2.59 2 | 7.77 3 | 1.5 1 |
| Bex2-1-5.g | 285 | 352; ; 295; | 21 | 2.97794036 0 | 2.97794034 0 | 1.1 -9 | 1.2-11 | 3.2 -9 | 3.1 -9 | 02 | 9.80 2 | 5.30 5 | 7.6 1 |
| Bex2-1-8.g | 1789 | 798; ; 2655; 596 | 37 | -7.85090352-1 | -7.85090364-1 | 5.4-10 | 1.3 -9 | 9.0 -9 | 4.7 -9 | 25 | 5.10 3 | ∞ | 3.9 1 |
| Bex3-1-1.g | 310 | 300; ; 604; | 24 | -1.17487427 0 | -1.17487491 0 | 2.9 -7 | 9.7-12 | 2.5 -7 | 1.9 -7 | 02 | 1.09 3 | 2.13 10 | 4.3 3 |
| Bex3-1-2.g | 111 | 110; ; 212; | 19 | -9.33357008-1 | -9.33357019-1 | 1.5-10 | 8.1-12 | 4.3 -9 | 3.9 -9 | 01 | 3.60 2 | 4.54 5 | 5.8 1 |
| Bex3-1-4.g | 164 | 215; ; 73; | 21 | 1.00009958 0 | 1.00010521 0 | 1.2 -5 | 3.9-10 | 2.6 -6 | 1.9 -6 | 03 | 4.52 2 | 1.42 6 | 2.2 3 |
| Bex5-2-2-c | 300 | 212; ; 359; 220 | 40 | 8.33333333-2 | 8.33333333-2 | 1.1 -8 | 4.1-15 | 5.0-13 | 1.5-13 | 02 | 1.15 3 | ∞ | 2.3 2 |
| Bex5-2-2-c | 300 | 212; ; 359; 220 | 36 | 7.50000087-2 | 7.50000039-2 | 2.1 -9 | 1.4-10 | 3.8 -9 | 4.3 -9 | 02 | 1.15 3 | ∞ | 2.5 2 |

Table 1: Performance of `sdpt3.m`. In the table, err = [`pinfeas`,`dinfeas`,`relgap`,`relgap2`], where `relgap2` is the same as `relgap` but with the numerator replaced by $|\langle c,\, x\rangle - b^T y|$, and `normXZ` $= \max\{\|x^*\|, \|z^*\|\}$. We declare that $g_P$ ($g_D$) is $\infty$ if the computed number is larger than $10^{12}$.

| problem | $m$ | $n_s; n_q; n_l; n_u$ | it. | primal obj | dual obj | err | | | | time | $g_P$ | $g_D$ | `normXZ` |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bex5-2-2-c | 300 | 212; ; 359; 220 | 27 | 1.92307719-1 | 1.92307733-1 | 9.6 -9 | 5.0 -9 | 2.0 -9 | 1.0 -8 | 02 | 1.15 3 | $\infty$ | 6.7 1 |
| Bex5-3-2.g | 1131 | 495; ; 1637; 530 | 46 | -1.30519579-1 | -1.30523903-1 | 2.3 -5 | 6.0 -8 | 2.0 -5 | 3.4 -6 | 06 | 3.60 3 | $\infty$ | 5.8 3 |
| Bex5-4-2.g | 310 | 300; ; 604; | 28 | -7.51223010-1 | -7.51223017-1 | 4.2 -8 | 6.2-14 | 2.9 -9 | 2.7 -9 | 02 | 1.09 3 | 2.02 8 | 4.2 3 |
| Bex9-1-1.g | 489 | 368; ; 548; 699 | 29 | 2.16666669-1 | 2.16666667-1 | 1.0 -9 | 8.9 -9 | 3.7 -9 | 1.0 -9 | 04 | 2.53 3 | $\infty$ | 1.7 1 |
| Bex9-1-2.g | 241 | 223; ; 227; 310 | 28 | 1.06666667 0 | 1.06666667 0 | 2.7 -9 | 1.6-10 | 5.1-11 | 2.3 -9 | 01 | 1.20 3 | 1.59 10 | 1.5 1 |
| Bex9-1-4.g | 241 | 223; ; 227; 310 | 24 | 6.60872427-1 | 6.60872425-1 | 6.5-10 | 2.6 -9 | 4.4-10 | 6.6-10 | 02 | 1.20 3 | $\infty$ | 2.2 1 |
| Bex9-1-5.g | 489 | 368; ; 548; 700 | 28 | 2.01411716-2 | 2.01411716-2 | 3.6-11 | 2.6-10 | 4.4-11 | 3.2-11 | 04 | 2.52 3 | $\infty$ | 1.8 1 |
| Bex9-1-8.g | 402 | 330; ; 434; 522 | 29 | 6.50000034-1 | 6.50000005-1 | 1.3 -8 | 1.9 -8 | 6.5 -9 | 1.2 -8 | 03 | 2.01 3 | 4.42 10 | 1.7 1 |
| Bex9-2-1.g | 241 | 223; ; 227; 310 | 25 | 4.59183756-2 | 4.59183697-2 | 4.4 -9 | 5.8 -9 | 1.9 -9 | 5.4 -9 | 02 | 1.18 3 | $\infty$ | 1.3 1 |
| Bex9-2-2.g | 137 | 156; ; 109; 168 | 42 | 2.87494906-6 | 2.25999495-6 | 1.7 -6 | 1.1 -8 | 1.4 -6 | 6.1 -7 | 02 | 6.86 2 | 1.08 11 | 5.1 2 |
| Bex9-2-3.g | 866 | 549; ; 1408; 1176 | 35 | -1.19999994 0 | -1.20000000 0 | 1.4-10 | 5.8-10 | 1.7 -9 | 1.7 -8 | 11 | 6.15 3 | 3.31 10 | 4.4 2 |
| Bex9-2-4.g | 146 | 149; ; 276; 215 | 26 | 1.75001038-4 | 1.75000009-4 | 2.9-10 | 6.5-10 | 2.6 -9 | 1.0 -9 | 02 | 9.09 2 | $\infty$ | 1.4 1 |
| Bex9-2-5.g | 137 | 147; ; 109; 168 | 24 | 8.00000007-2 | 8.00000001-2 | 5.1-10 | 1.8-10 | 2.1-11 | 5.7-10 | 01 | 6.66 2 | $\infty$ | 8.9 0 |
| Bex9-2-6.g | 402 | 317; ; 434; 522 | 28 | 2.42789755-5 | 2.42661365-5 | 6.9 -9 | 1.5 -9 | 1.9 -9 | 1.3 -8 | 03 | 1.96 3 | $\infty$ | 1.8 1 |
| Bex9-2-7.g | 241 | 223; ; 227; 310 | 26 | 4.59183796-2 | 4.59183677-2 | 7.6 -9 | 2.5 -9 | 1.3 -9 | 1.1 -8 | 01 | 1.18 3 | 1.57 10 | 1.3 1 |
| Bex9-2-8.g | 19 | 28; ; 8; 50 | 16 | 1.70246919-11 | 3.26220519-10 | 2.9-12 | 6.3 -9 | 1.2-10 | 3.1-10 | 00 | 1.46 2 | $\infty$ | 2.8 0 |
| Balkyl.gms | 834 | 572; ; 1640; 310 | 32 | 5.89807109-2 | 5.89807733-2 | 2.7 -9 | 4.3-10 | 4.4 -9 | 5.6 -8 | 05 | 3.09 3 | 7.12 10 | 4.9 1 |
| Bst-bpaf1a | 195 | 245; ; 205; | 31 | 1.13449275-1 | 1.13449275-1 | 1.2-10 | 1.1-14 | 3.9-10 | 3.4-10 | 02 | 6.07 2 | 3.64 8 | 2.2 3 |
| Bst-bpaf1b | 195 | 245; ; 205; | 28 | 1.07406396-1 | 1.07406388-1 | 2.7-10 | 1.3-13 | 7.4 -9 | 6.5 -9 | 01 | 6.08 2 | 3.64 8 | 4.1 3 |
| Bst-e05.gm | 40 | 44; ; 70; 22 | 23 | -1.94462048-1 | -1.94462044-1 | 8.7-10 | 7.0-10 | 2.4-10 | 3.4 -9 | 00 | 1.77 2 | $\infty$ | 2.1 1 |
| Bst-e07.gm | 178 | 161; ; 189; 82 | 25 | 3.76913266-1 | 3.76913266-1 | 1.6-10 | 2.7 -9 | 5.4-10 | 1.8-10 | 01 | 6.45 2 | $\infty$ | 1.2 1 |
| Bst-jcbpaf | 285 | 374; ; 295; | 25 | 7.94855924-2 | 7.94855883-2 | 1.1-11 | 2.0-12 | 3.6 -9 | 3.5 -9 | 02 | 8.76 2 | 6.54 5 | 1.7 1 |
| Bhaverly.g | 274 | 206; ; 331; 154 | 46 | 8.00000000-2 | 8.00000000-2 | 2.4 -7 | 1.6-14 | 5.5-12 | 1.6-13 | 02 | 9.82 2 | $\infty$ | 6.0 2 |
| alkylation | 530 | 457; ; 1040; 155 | 44 | 1.18826559-2 | 1.18824854-2 | 3.1 -7 | 7.0-11 | 6.7 -9 | 1.7 -7 | 06 | 2.00 3 | 5.95 10 | 2.8 2 |
| Bst-bpk1.g | 34 | 55; ; 30; | 14 | 1.30000000 1 | 1.30000000 1 | 7.5-13 | 1.0-12 | 1.6 -9 | 1.6 -9 | 00 | 7.01 2 | 3.28 2 | 1.6 2 |
| Bst-bpk2.g | 34 | 55; ; 30; | 14 | 1.30000000 1 | 1.30000000 1 | 7.5-13 | 1.0-12 | 1.6 -9 | 1.6 -9 | 00 | 7.01 2 | 3.28 2 | 1.6 2 |
| Bst-bpv1.g | 29 | 56; ; 17; | 13 | -3.70370361-2 | -3.70370407-2 | 4.6-10 | 1.7-12 | 6.7 -9 | 4.3 -9 | 00 | 1.01 2 | $\infty$ | 5.7 2 |
| Bst-bpv2.g | 25 | 52; ; 14; | 14 | 4.00000003-1 | 3.99999993-1 | 1.1-13 | 1.0-12 | 5.7 -9 | 5.7 -9 | 00 | 8.66 1 | 2.28 3 | 5.9 0 |
| Bst-e42.gm | 104 | 78; ; 103; 56 | 33 | -1.87841999 1 | -1.87842000 1 | 1.3 -8 | 1.8 -9 | 5.4 -9 | 2.3 -9 | 01 | $\infty$ | $\infty$ | 9.8 3 |
| Bst-robot. | 494 | 189; ; 972; 360 | 23 | 2.25821389-4 | 2.25821334-4 | 1.3-12 | 6.5-10 | 8.1-11 | 5.4-11 | 02 | 1.98 3 | 1.63 11 | 4.3 1 |
| Bprolog.gm | 833 | 533; ; 1023; | 91 | 5.22890377-4 | -2.14756935-6 | 9.8-13 | 1.6 -7 | 6.7 -4 | 5.2 -4 | 14 | $\infty$ | 3.95 5 | 3.3 10 |
| st-cqpjk2. | 19 | 32; ; 8; | 13 | 8.33333343-1 | 8.33333317-1 | 1.2-11 | 1.0-12 | 9.9 -9 | 9.9 -9 | 00 | 5.61 1 | 5.32 2 | 4.9 0 |
| st-e01.gms | 12 | 19; ; 8; | 18 | 1.11111113 0 | 1.11111112 0 | 4.9-13 | 1.0-12 | 4.0 -9 | 4.0 -9 | 00 | 3.50 1 | 6.33 2 | 3.9 0 |
| st-e09.gms | 25 | 38; ; 12; | 14 | 2.50000000-1 | 2.50000000-1 | 1.8-11 | 1.0-12 | 3.9-10 | 3.5-10 | 00 | 6.76 1 | 2.30 4 | 2.6 1 |
| st-e10.gms | 10 | 16; ; 3; 1 | 17 | 6.97453884-1 | 6.97453928-1 | 2.0 -9 | 5.7 -9 | 4.7-10 | 1.8 -8 | 00 | 2.94 1 | $\infty$ | 7.8 0 |
| st-e20.gms | 111 | 84; ; 210; 72 | 23 | 2.67968679-1 | 2.67968687-1 | 1.6 -9 | 2.9 -9 | 6.9-10 | 4.9 -9 | 01 | 4.84 2 | $\infty$ | 1.1 1 |
| st-e23.gms | 9 | 21; ; 3; | 14 | 4.33333353-2 | 4.33333298-2 | 2.5-11 | 1.5-12 | 5.1 -9 | 5.1 -9 | 00 | 2.87 1 | 2.90 4 | 1.0 1 |
| st-e34.gms | 164 | 125; ; 187; | 20 | -3.62873428-3 | -3.62873545-3 | 2.6-14 | 1.0-12 | 1.2 -9 | 1.2 -9 | 01 | 4.43 2 | 4.57 3 | 9.3 0 |
| st-e42.gms | 104 | 78; ; 103; 56 | 33 | -1.87841999 1 | -1.87842000 1 | 1.3 -8 | 1.8 -9 | 5.4 -9 | 2.3 -9 | 02 | $\infty$ | $\infty$ | 9.8 3 |
| st-fp5.gms | 285 | 352; ; 295; | 21 | 2.97794036 0 | 2.97794034 0 | 1.1 -9 | 1.2-11 | 3.2 -9 | 3.1 -9 | 02 | 9.80 2 | 5.30 5 | 7.6 1 |
| st-glmp-fp | 34 | 55; ; 14; 30 | 19 | -9.99999994 0 | -9.99999992 0 | 1.7 -9 | 5.5 -9 | 6.6-10 | 9.9-10 | 00 | 6.94 2 | $\infty$ | 2.1 2 |
| st-glmp-fp | 125 | 180; ; 36; 140 | 29 | -7.34454541 0 | -7.34454542 0 | 1.0-10 | 3.7-10 | 4.2-10 | 3.8-10 | 02 | 1.66 4 | $\infty$ | 4.7 3 |
| st-glmp-fp | 34 | 55; ; 14; 30 | 20 | 1.20000000 1 | 1.20000000 1 | 1.4 -9 | 3.2-10 | 1.4-11 | 1.6-10 | 00 | 1.09 3 | $\infty$ | 9.5 1 |
| st-glmp-kk | 40 | 44; ; 14; 36 | 20 | -3.00000000 0 | -2.99999999 0 | 4.9-10 | 7.4-10 | 2.7-11 | 1.4 -9 | 01 | 6.17 2 | $\infty$ | 5.5 1 |
| st-glmp-kk | 34 | 55; ; 14; 30 | 20 | 1.20000000 1 | 1.20000000 1 | 2.3 -9 | 2.2 -9 | 1.7-10 | 6.4-10 | 00 | 7.31 2 | $\infty$ | 1.7 2 |
| st-glmp-kk | 69 | 62; ; 14; 70 | 29 | 2.50000003 0 | 2.50000005 0 | 2.4 -9 | 1.8-10 | 4.6 -9 | 2.6 -9 | 01 | 8.30 3 | $\infty$ | 6.2 2 |
| st-glmp-ss | 40 | 64; ; 14; 36 | 21 | 2.45714286 1 | 2.45714286 1 | 1.6 -9 | 7.2-10 | 7.0-11 | 1.0-10 | 00 | 4.06 3 | $\infty$ | 3.9 2 |
| st-glmp-ss | 40 | 49; ; 14; 36 | 20 | -2.99999996 0 | -2.99999996 0 | 5.8 -9 | 1.9 -9 | 1.7-10 | 2.6-10 | 00 | 1.28 3 | $\infty$ | 6.5 1 |
| st-iqpbk1. | 164 | 216; ; 312; | 19 | 4.15321990 0 | 4.15321989 0 | 4.5-13 | 1.0-12 | 6.2-10 | 6.1-10 | 01 | 6.94 2 | 1.76 4 | 5.4 1 |
| st-iqpbk2. | 164 | 216; ; 312; | 19 | 3.99367031 0 | 3.99367031 0 | 3.4-13 | 1.0-12 | 7.3-10 | 7.3-10 | 01 | 6.94 2 | 1.76 4 | 5.4 1 |
| st-jcbpaf2 | 285 | 374; ; 295; | 25 | 7.94855924-2 | 7.94855883-2 | 1.1-11 | 2.0-12 | 3.6 -9 | 3.5 -9 | 02 | 8.76 2 | 6.54 5 | 1.7 1 |
| st-jcbpafe | 9 | 21; ; 3; | 14 | 4.33333353-2 | 4.33333298-2 | 2.5-11 | 1.5-12 | 5.1 -9 | 5.1 -9 | 00 | 2.87 1 | 2.90 4 | 1.0 1 |
| qp5.gms | 108 | ; ; 109; 30 | 23 | -4.31455897-1 | -4.31455880-1 | 1.5 -9 | 9.2-10 | 6.3-10 | 9.5 -9 | 00 | 1.77 2 | $\infty$ | 4.0 1 |
| Rosenbrock | 1988 | 1195; ; 3; | 18 | 9.95000004-1 | 9.94999999-1 | 1.4-10 | 1.4-12 | 5.8-10 | 1.8 -9 | 01 | 1.28 5 | 2.35 3 | 8.5 1 |
| BroydenBan | 923 | 84; ; ; | 17 | 2.40000000-1 | 2.39999999-1 | 7.4-12 | 1.0-12 | 4.9-10 | 4.9-10 | 06 | 5.16 2 | 1.54 2 | 5.7 0 |
| BroydenTri | 3974 | 1984; ; 3; | 16 | 1.11111111 1 | 1.11111111 1 | 4.3-12 | 3.3-12 | 4.8-10 | 4.8-10 | 03 | 1.75 4 | 3.34 3 | 6.1 1 |
| ChainedSin | 3974 | 1980; ; ; | 24 | 2.88480646-10 | -2.56585137-9 | 4.5-14 | 1.0-12 | 2.9 -9 | 2.9 -9 | 03 | 2.92 7 | 2.77 3 | 1.4 1 |
| ChainedWoo | 899 | 697; ; ; | 11 | 1.09421053 1 | 1.09421052 1 | 3.5-12 | 1.0-12 | 6.4 -9 | 6.4 -9 | 00 | 4.43 5 | 9.79 2 | 5.0 1 |
| nondquar(2 | 3974 | 1980; ; ; | 24 | 2.34095452-6 | -1.86415946-6 | 2.5 -9 | 2.2-10 | 8.1 -6 | 4.2 -6 | 04 | $\infty$ | 2.77 3 | 1.1 5 |
| nonscomp(8 | 110 | 138; ; 204; | 23 | 1.56250002-2 | 1.56249998-2 | 1.4 -9 | 7.6-12 | 7.5-10 | 3.9-10 | 01 | 4.47 2 | 5.12 3 | 1.6 1 |
| optControl | 2682 | 533; ; ; 682 | 25 | 3.30330643-1 | 3.30330643-1 | 5.3-11 | 3.1-10 | 4.2-10 | 7.5-11 | 32 | 3.95 3 | $\infty$ | 7.8 0 |
| optControl | 4759 | 1587; ; ; 1986 | 36 | -3.06689479 2 | -3.06689479 2 | 1.1 -7 | 2.2-10 | 8.8-11 | 1.3-10 | 06 | $\infty$ | $\infty$ | 4.9 3 |

Table 1: Performance of `sdpt3.m`. In the table, err = [`pinfeas`,`dinfeas`,`relgap`,`relgap2`], where `relgap2` is the same as `relgap` but with the numerator replaced by $|\langle c,\, x\rangle - b^T y|$, and `normXZ` = $\max\{\|x^*\|, \|z^*\|\}$. We declare that $g_P$ ($g_D$) is $\infty$ if the computed number is larger than $10^{12}$.

| problem | $m \mid n_s; n_q; n_l; n_u$ | it. \| primal obj \| dual obj | err | time | $g_P \mid g_D \mid$ `normXZ` |
|---|---|---|---|---|---|
| randomUnco | 264 \| 130; ; ; | 17 \| 6.33323683-4 \| 6.33323351-4 | 2.7-14\| 1.0-12\| 3.4-10\| 3.3-10 | 01 | 2.14 2\| 1.88 2\| 4.2 0 |
| randomCons | 650 \| 298; ; ; | 17 \| 3.94559182 0 \| 3.94559180 0 | 2.2-12\| 1.0-12\| 2.1 -9 \| 2.1 -9 | 01 | 4.27 2\| 9.57 3\| 2.7 1 |
| randomwith | 430 \| 130; ; ; 395 | 22 \| 3.13274316-4 \| 3.13274418-4 | 7.5-13\| 3.8-10\| 1.6-11\| 1.0-10 | 01 | 9.99 2\| $\infty$ \| 4.2 0 |