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

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

Draft, 17 July 2006

Abstract

This software 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. 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 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 SQLP solver. Various techniques to improve the efficiency and stability of the algorithm are incorporated. For example, step-lengths associated with semidefinite cones are calculated via the Lanczos method. Numerical experiments show that this general purpose code can solve more than 80% of a total of about 300 test problems to an accuracy of at least 10^{-6} in relative duality gap and infeasibilities.

[*]Department of Mathematics, National University of Singapore, 2 Science Drive 2, Singapore 117543 (mattohkc@nus.edu.sg); and Singapore-MIT Alliance, E4-04-10, 4 Engineering Drive 3, Singapore 117576. Research supported in parts by NUS Research Grant R146-000-076-112 and SMA IUP Research Grant.

[†]Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213, USA (reha@cmu.edu). Research supported in part by NSF through grants CCR-9875559, CCF-0430868 and by ONR through grant N00014-05-1-0147

[‡]School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York 14853, USA (miketodd@cs.cornell.edu). Research supported in part by NSF through grant DMS-0209457 and by ONR through grant N00014-02-1-0057.

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 SQLP problems:

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

Here, c_j^s, x_j^s are symmetric matrices of dimension s_j and $K_s^{s_j}$ is the cone of positive semidefinite symmetric matrices of the same dimension. 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 vectors of dimension n_l, $K_l^{n_l}$ is the nonnegative orthant $I\!R_+^{n_l}$, and c^u, x^u are 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 jth semidefinite block variable x_j^s. The matrix A_i^q is an $m \times q_i$ dimensional constraint matrix corresponding to the ith 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. 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), ν_j^s, ν_i^q, and ν_k^l are given nonnegative parameters.

In this paper, the vector 2-norm and Frobenius norm are denoted by $\|\cdot\|$ and $\|\cdot\|_F$, respectively. 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 kth 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 purpose, 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 above:

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

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

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:

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

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

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

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

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

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

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 in several dimensions.

1. This version is faster than the previous releases, e.g. [27], [30], 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 requirement.

All the numerical experiments in this paper are performed on a Pentium IV 3.0GHz personal computer with 4GB of physical memory using MATLAB version 7 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. Finally, the last section reports computational results obtained by SDPT3 on about 300 SQLP problems.

Installation. The current version is written in MATLAB version 6.5 and is compatible with MATLAB version 7.0. It is available from the internet sites:

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

`http://www.math.cmu.edu/~reha/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 dimension greater than or equal to 100;
2. a sparse block-diagonal matrix consisting of numerous sub-blocks each of dimension 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 jth block is a semidefinite block consisting of a single block of size $\mathtt{s_j}$, then

$$\mathtt{blk\{j,1\} = 's'}, \quad \mathtt{blk\{j,2\}} = [\mathtt{s_j}],$$
$$\mathtt{At\{j\}} = [\bar{\mathtt{s}}_\mathtt{j} \,\mathtt{x\, m\ sparse}],$$
$$\mathtt{C\{j\},\ X\{j\},\ Z\{j\}} = [\mathtt{s_j\, x\, s_j\ \ double\ or\ sparse}],$$

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

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

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

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

Notice that we store all the constraint matrices associated with the jth semidefinite block in vectorized form as a single $\bar{s}_j \times m$ matrix where the kth column of this matrix corresponds to the kth constraint matrix. That is, `At{j}(:,k) = svec(`$a^s_{j,k}$`)`.

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 [27]. 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 single 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 ith 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

$\mathtt{blk\{i,1\}} = \mathtt{'q'}, \qquad \mathtt{blk\{i,2\}} = [\mathtt{q_{i1}},\ \mathtt{q_{i2}},\ \cdots,\ \mathtt{q_{ip}}],$

$\mathtt{At\{i\}} = [\mathtt{q_i\,x\,m\ sparse}],$

$\mathtt{C\{i\},\ X\{i\},\ Z\{i\}} = [\mathtt{q_i\,x\,1\ \ double\ or\ sparse}].$

If the kth block is the linear block, then

$\mathtt{blk\{k,1\}} = \mathtt{'l'}, \qquad \mathtt{blk\{k,2\}} = \mathtt{n_l},$

$\mathtt{At\{k\}} = [\mathtt{n_l\,x\,m\ \ sparse}],$

$\mathtt{C\{k\},\ X\{k\},\ Z\{k\}} = [\mathtt{n_l\,x\,1\ \ double\ or\ sparse}].$

Similarly, if the kth block is the unrestricted block, then

$\mathtt{blk\{k,1\}} = \mathtt{'u'}, \qquad \mathtt{blk\{k,2\}} = \mathtt{n_u},$

$\mathtt{At\{k\}} = [\mathtt{n_u\,x\,m\ \ sparse}],$

$\mathtt{C\{k\},\ X\{k\},\ Z\{k\}} = [\mathtt{n_u\,x\,1\ \ double\ or\ sparse}].$

(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 choose 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 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_diag.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');
>> [blk,At,C,b] = detect_diag(blk,At,C,b);
```

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 jth 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, $\texttt{blk}\{\texttt{j},2\} = [\texttt{s}_\texttt{j}]$. 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:

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

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

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, ith row) of `dd` has the following form:

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

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 function: `sqlp.m`

The algorithm implemented in SDPT3 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 [17] 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 is `sqlp.m`, whose calling syntax is as follows:

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

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

Caveats.

[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.

(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 dimension more than 2000 on an average PC available today.
(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 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 γ in Algorithm IPC instead of the adaptive strategy used in the default, he 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 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 jth block is a semidefinite block consisting of one or more sub-blocks, say p of them, of dimensions $\mathtt{s}_{j1}, \mathtt{s}_{j2}, \ldots, \mathtt{s}_{jp}$, then

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

If the ith 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

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

If the kth block is the linear block, then

$$\texttt{OPTIONS.parbarrier\{k\}} = [\nu_1^1,\ \nu_2^1,\ \cdots,\ \nu_{\texttt{n}_1}^1],$$

while if the kth block is the unrestricted block, then

$$\texttt{OPTIONS.parbarrier\{k\}} = \texttt{zeros}(1, \texttt{n}_\texttt{u}).$$

The reader is referred to Section 4.2 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 [6] and SeDuMi [22] format into MATLAB cell arrays described above. 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 [20].

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 1.4e+03 9.5e+01  7.0e+05 -1.462827e+04  0:0:0 spchol  1  1
 1  0.981 1.000 2.6e+01 9.8e-15  1.7e+04 -2.429708e+03  0:0:0 spchol  1  1
 2  1.000 1.000 5.0e-14 0.0e+00  2.4e+03 -1.352811e+03  0:0:1 spchol  1  1
 :    :     :      :       :        :          :         :       :
13  1.000 0.996 9.4e-13 5.1e-16  2.1e-05 -3.172643e+02  0:0:4 spchol  1  1
14  1.000 1.000 2.5e-12 4.3e-16  6.5e-07 -3.172643e+02  0:0:5
 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)       = 6.45e-07
 relative gap           = 1.02e-09
 actual relative gap    = 1.02e-09
 rel. primal infeas     = 2.52e-12
```

```
 rel. dual    infeas      = 4.29e-16
 norm(X), norm(y), norm(Z) = 1.3e+02, 2.3e+01, 1.3e+01
 norm(A), norm(b), norm(C) = 1.6e+01, 1.6e+01, 1.4e+01
 Total CPU time (secs)  = 4.7
 CPU time per iteration = 0.3
 termination code       =  0
 DIMACS: 2.5e-12  0.0e+00  4.3e-16  0.0e+00  1.0e-09  1.0e-09
------------------------------------------------------------------
```

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

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

3.3 Stopping criteria

We define

$$n = \sum_{\{j:\nu_j^s=0\}} s_j + \sum_{\{i:\nu_i^q=0\}} q_i + |\{k:\nu_k^l=0\}| \tag{3}$$

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

$$\begin{aligned} \texttt{gap} = {} & \langle x,\, z\rangle - \sum_{\{j:\nu_j^s>0\}} \nu_j^s\Big(s_j + \log\det(x_j^s z_j^s/\nu_j^s)\Big) \\ & - \sum_{\{i:\nu_i^q>0\}} \nu_i^q\Big(1+\log(\gamma(x_i^q)\gamma(z_i^q)/\nu_i^q)\Big) - \sum_{\{k:\nu_k^l>0\}} \nu_k^l\Big(1+\log(x_k^l z_k^l/\nu_k^l)\Big). \end{aligned} \tag{5}$$

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 [7], 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 vector of all ones, and

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

By multiplying the identity matrix I_{s_j} by the factors ξ_j^s and η_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_diag.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 modeled as the difference of two nonnegative variables. The calling syntax is:

```
[bblk,AAt,CC] = detect_ublk(blk,At,C);
```

Complex data.

In earlier versions, 2.3 or earlier, SDPT3 can directly handle complex data in SDP, i.e., 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. But we intend to keep version 2.3 of the code available for users who wish to solve SDP problem (with no quadratic blocks) with complex data directly.

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

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

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 $y_I^k = 0$, $k = 1, \ldots, m$.

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.

Suppose `AA` is an $1 \times m$ cell array such that `AA{k}` $= A_k$. Then `convertcmpsdp.m` has the calling syntax:

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

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

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 sub-blocks, say p of them of dimension `r`$_{i1}$, ...,`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} = [r_i1, ..., r_ip];
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 needs to do is to code the problem data corresponding to the standard form in (P). The simplest way to learn how to generate the data of an SQLP problem in SDPT3 format is through examples. The subdirectory `Examples` in SDPT3 contains many such example files. Here we will just mention a few.

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.

4.1 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:

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

where e is the vector of ones, and $C = -(\text{Diag}\,(Be) - B)/4$. 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.)

4.2 D-optimal experiment design - an example with an explicit barrier term

Given a set of points $\{v_1, \ldots, v_p\}$ in $\mathbb{R}^n$ with $n \le p$, the D-optimal experiment design problem [31] needs to solve the following dual SQLP:

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

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;
OPTIONS.parbarrier{2} = zeros(1,p);
OPTIONS.parbarrier{3} = 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);
```

4.3 An LMI example

Consider the following LMI problem [3]:

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

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`.

4.4 Nearest correlation matrix problem

Given an $n \times n$ symmetric matrix H, the nearest correlation matrix problem is the following [12]:

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

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 dimension n_2, and Q^{n_2+1} denotes the second-order cone of dimension n_2+1. 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`.

4.5 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 $\mathbb{R}^p$ (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:

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

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 $\min \{ \sum_{(i,j)\in\mathcal{E}} |\langle e_{ij}e_{ij}^T, Y\rangle - d_{ij}^2| : Y = X^T X,\ X \in \mathbb{R}^{p\times n}\}$. 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);
```

4.6 Norm minimization problem with complex data

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

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

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; Bk', zeros(q)]; end
for k = 1:m; AA{1,m+k} = [zeros(p), i*Bk; -i*Bk', zeros(q)]; end
AA{1,2*m+1} = -speye(p+q);
C{1} = -[zeros(p), B0; B0', 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`.

4.7 Logarithmic Chebyshev approximation problem

This is an example that contains variables in a rotated cone. The orginal problem [16] is given by $\min_{x\in I\!R^m} \max\{\log(f_i^T x)-\log(d_i) : i=1,\ldots,p\}$, which can equivalently be formulated as: $\min\{s \,:\, t \le f_i^T x/d_i \le s,\ i=1,\ldots,p,\ 1\le st\}$. Let $F=[f_1^T;\ldots;f_p^T]$ and $d=[d_1;\ldots;d_p]$. The latter problem can be formulated in the following standard dual form:

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

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`.

4.8 Maximizing the geometric mean of affine functions

Another example with rotated cone variables comes from maximizing the geometric mean of nonnegative affine functions [16]:

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

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

$$\max \left\{ t_3/\sqrt{2} \,:\, \sqrt{2} y_i \;=\; a_i^T x + b_i,\ y_i \geq 0,\ i = 1, \ldots, 4,\ t_1^2 \leq 2y_1 y_2,\ t_2^2 \leq 2y_3 y_4,\ t_3^2 \leq 2t_1 t_2 \right\}.$$

The corresponding standard (dual) form is as follows:

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

4.9 Conversion of problems into the standard form

SDP problems are usually not formulated in the standard form (P) or (D). It is often quite tedious to convert such problems into the standard form. As such, there is an increasing demand for interfaces to automate the conversion process. Currently, one of the few interfaces that is publicly available for SQLP solvers is YALMIP [15].

Here we shall just give an example on how an SDP with linear inequality constraints can be converted into the standard form given in the Introduction. Suppose we have an SDP of the following form:

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

That is, it has inequality instead of equality constraints. But by introducing a slack

variable x^l, we can easily convert (P_1) into standard form, namely,

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

where $c^l = 0$, and $A^l = I_{m\times m}$. With our use of cell arrays to represent 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} =` $(A^s)^T$`;`	`C{1} =` c^s`;`
`blk{2,1} = 'l';`	`blk{2,2} = m;`
`At{2} = speye(m);`	`C{2} =` c^l`;`

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

$$J_i^q \;=\; \begin{bmatrix} 1 & 0 \\ 0 & -I_{q_i-1} \end{bmatrix}. \tag{9}$$

For the choice of the HKM scaling matrix [10, 14, 18, 28, 29], the search direction $(\Delta x, \Delta y, \Delta z)$ is obtained from the following system of equations:

$$\begin{aligned} \mathcal{A}^s(\Delta x^s) + A^q(\Delta x^q) + A^l\Delta x^l + \; A^u\Delta x^u \;&=\; R_p := b - \mathcal{A}^s(x^s) - A^q(x^q) - A^l x^l - A^u x^u \\ (\mathcal{A}^s)^T\Delta y + \Delta z^s \;&=\; R_d^s := c^s - z^s - (\mathcal{A}^s)^T y \\ (A^q)^T\Delta y + \Delta z^q \;&=\; R_d^q := c^q - z^q - (A^q)^T y \\ (A^l)^T\Delta y + \Delta z^l \;&=\; R_d^l := c^l - z^l - (A^l)^T y \\ (A^u)^T\Delta y \;&=\; R_d^u := c^u - (A^u)^T y \\ \Delta x^s + H^s(\Delta z^s) \;&=\; R_c^s := \Big(\max\{\sigma\mu(x,z), \nu_j^s\}(z_j^s)^{-1} - x_j^s\Big)_{j=1}^{n_s} \\ \Delta x^q + H^q(\Delta z^q) \;&=\; R_c^q := \Big(\max\{\sigma\mu(x,z), \nu_i^q\}(z_i^q)^{-1} - x_i^q\Big)_{i=1}^{n_q} \\ \Delta x^l + H^l(\Delta z^l) \;&=\; R_c^l := \Big(\max\{\sigma\mu(x,z), \nu_k^l\}(z_k^l)^{-1} - x_k^l\Big)_{k=1}^{n_l}, \end{aligned} \tag{10}$$

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,

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

$$H^q(\Delta z^q) = \Big(H_i^q(\Delta z_i^q) := -\frac{\langle x_i^q,\, z_i^q \rangle}{\gamma(z_i^q)^2} J_i^q \Delta z_i^q + x_i^q((z_i^q)^{-1})^T \Delta z_i^q + (z_i^q)^{-1}(x_i^q)^T \Delta z_i^q \Big)_{i=1}^{n_q}, \quad (11)$$

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

We compute the search direction via a Schur complement equation as follows (the reader is referred to [1] and [23] for details). First compute Δy from the Schur complement equation

$$\underbrace{\begin{bmatrix} M & A^u \\ (A^u)^T & 0 \end{bmatrix}}_{\mathcal{M}} \begin{bmatrix} \Delta y \\ \Delta x^u \end{bmatrix} = \begin{bmatrix} h \\ R_d^u \end{bmatrix} \quad (12)$$

where

$$M = \sum_{j=1}^{n_s} \underbrace{\mathcal{A}_j^s H_j^s (\mathcal{A}_j^s)^T}_{M_j^s} + \sum_{i=1}^{n_q} \underbrace{A_i^q H_i^q (A_i^q)^T}_{M_i^q} + \underbrace{A^l H^l (A^l)^T}_{M^l} \quad (13)$$

$$h = R_p + \mathcal{A}^s\Big(H^s(R_d^s) - R_c^s\Big) + A^q\Big(H^q(R_d^q) - R_c^q\Big) + A^l\Big(H^l(R_d^l) - R_c^l\Big).$$

(The notation in (13) 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 Δx and Δz from the equations

$$\Delta z^s = R_d^s - (\mathcal{A}^s)^T \Delta y, \quad \Delta z^q = R_d^q - (A^q)^T \Delta y, \quad \Delta z^l = R_d^l - (A^l)^T \Delta y$$

$$\Delta x^s = R_c^s - H^s(\Delta z^s), \quad \Delta x^q = R_c^q - H^q(\Delta z^q), \quad \Delta x^l = R_c^l - H^l(\Delta z^l).$$

5.2 The NT search direction

The user also has the choice of using the NT direction [21, 28, 29]. 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

$$\omega_i^q = \sqrt{\frac{\gamma(z_i^q)}{\gamma(x_i^q)}}, \quad \xi_i^q = \frac{1}{\omega_i^q} z_i^q + \omega_i^q J_i^q x_i^q, \quad t_i^q = \frac{\sqrt{2}}{\omega_i^q \gamma(\xi_i^q)} J_i^q \xi_i^q. \quad (14)$$

In this case, the search direction $(\Delta x, \Delta y, \Delta z)$ satisfies the same system as in (10) except that H^s and H^q are replaced by

$$H^s(\Delta z^s) = \Big(H_j^s(\Delta z_j^s) := w_j^s \Delta z_j^s\, w_j^s \Big)_{j=1}^{n_s},$$

$$H^q(\Delta z^q) = \Big(H_i^q(\Delta z_i^q) := -\frac{1}{(\omega_i^q)^2} J_i^q \Delta z_i^q + t_i^q (t_i^q)^T \Delta z_i^q \Big)_{i=1}^{n_q}. \quad (15)$$

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, also allowed the AHO direction of Alizadeh, Haeberly, and Overton [1] and the GT direction [25], but these are uncompetitive when the problems are of large scale. We intend to keep version 2.3 of the code available for those who wish to experiment with these other search directions, which tend to give more accurate results on smaller problems.

For the HKM and NT search directions, our computational experience on problems tested in Section 7 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 (11) and (15), 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 (12). 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 (13).

5.4.1 Semidefinite blocks

For problems with semidefinite blocks, the contribution by the jth 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 [23, 27], 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 [7]. Further details on the efficient implementation of these ideas are given in [30].

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 jth 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 $\Theta(s_j^2 mr) + \Theta(s_j m^2 r^2)$. Thus it is definitely advantageous to exploit low-rank structures.

As 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` runs about 6 times faster when the low-rank structure is 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 (13), it is easily shown that the contribution of the quadratic blocks to the matrix M is given by

$$M_i^q = \begin{cases} -\frac{\langle x_i^q, z_i^q \rangle}{\gamma^2 (z_i^q)} A_i^q J_i^q (A_i^q)^T + u_i^q (v_i^q)^T + v_i^q (u_i^q)^T, & \text{for HKM direction} \\ -\frac{1}{(\omega_i^q)^2} A_i^q J_i^q (A_i^q)^T + w_i^q (w_i^q)^T & \text{for NT direction} \end{cases} \tag{16}$$

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 ith 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 (12). 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:

$$M_i^q \;=\; \frac{\langle x_i^q,\, z_i^q \rangle}{\gamma^2(z_i^q)} A_i^q (A_i^q)^T \;+\; u_i^q (v_i^q)^T + v_i^q (u_i^q)^T - k_i k_i^T, \tag{17}$$

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 (17).

5.5 Solving the Schur complement equation

The linear system (12) typically becomes more and more ill-conditioned as μ decreases to 0. Thus iterative refinement is generally recommended to improve the accuracy of the computed solution. An even better approach to solve (12) is via a preconditioned symmetric quasi-minimal residual method (PSQMR) [5] with the preconditioner computed based on the following analytical expression of $\mathcal{M}^{-1}$:

$$\mathcal{M}^{-1} \;=\; \begin{bmatrix} M^{-1} - M^{-1}A^u S^{-1}(A^u)^T M^{-1} & M^{-1}A^u S^{-1} \\ S^{-1}(A^u)^T M^{-1} & -S^{-1} \end{bmatrix}, \tag{18}$$

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:

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

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 the state-of-the-art Cholesky factorization softwares are 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 imported (with slight modifications) the sparse Cholesky solver in SeDuMi [22], which is a C translation of the Fortran programs developed by Esmond Ng, Barry Peyton, and Joseph Liu for sparse Cholesky factorization [19]. When SDPT3 uses the sparse Cholesky solver, it first performs a symmetric re-ordering on M_{sp} and generates a symbolic factorization of the re-ordered M_{sp} to determine the pivot order by examining the sparsity structure of this matrix carefully. Then, this pivot order is re-used in later iterations to compute the Cholesky factors. Contrary to the case of linear programming, however, the sparsity structure of M_{sp} can change during the course of interior-point iterations for SOCP problems. If this happens, the symbolic factorization has to be recomputed. We detect changes in the sparsity structure of M_{sp} by monitoring the nonzero elements of M_{sp}. Since the default initial iterates we use for an SOCP problem are unit vectors but subsequent iterates are not, there is always a change in the sparsity pattern of M_{sp} after the first iteration. After the second iteration, the sparsity pattern remains unchanged for most problems, and only one more change occurs in a small fraction of the test problems in the DIMACS library.

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 (12) by the SQMR method with preconditioner (18) 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 (12) based on (18) 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 (18) 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 use an LDL^T factorization of the symmetric indefinite matrix $\mathcal{M}$ to compute an approximation of $\mathcal{M}^{-1}$. But unfortunately the state-of-the-art LDL^T factorization software available in the public domain is not as highly developed as its counterpart for Cholesky factorization, especially for sparse matrices. When the use of LDL^T factorization of $\mathcal{M}$ is unavoidable, we use the publicly available Fortran subroutine `MA47` from the Harwell Subroutine Library [9] to compute such a factorization if $\mathcal{M}$ is sparse. When $\mathcal{M}$ is dense, the implementation we have at the moment is a little bit complicated. By rights, we should use the LDL^T factorization routine `dsysvx` in LAPACK to compute the required factorization for $\mathcal{M}$, but we have yet to modify and incorporate that routine into SDPT3. At the moment, we simply use the MATLAB routine `lu` to compute an LU factorization of $\mathcal{M}$, and use the computed LU factors to precondition the BiCGSTAB iterative method used to solve (12). Note that because the preconditioner in the latter case is no longer symmetric,

we have to replace the SQMR method by a nonsymmetric iterative method such as the BiGSTAB method.

5.6 Internal handling of unrestricted blocks

As mentioned in the last sub-section, solving the symmetric indefinite system (12) can potentially be very expensive when A^u is ill-conditioned or has a large number of columns because computing an LDL^T factorization of a sparse matrix can be much more costly than that for a symmetric positive definite matrix of the same dimension. 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 (12) 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 (12), 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 the vectors z_+^u, z_-^u. Such a modification in z_+^u, z_-^u ensures that they approach 0 at the same rate as μ, and thus prevents the dual problem (D) from approaching the equality constraint too closely prematurely.

In the current implementation of SDPT3, we always reformulate an unrestricted vector by the difference of 2 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

$$M = M_{sp} + UDU^T \tag{19}$$

where M_{sp} is a sparse symmetric positive semidefinite matrix, U has only few columns, and D is a non-singular matrix. If M_{sp} is positive definite, then we can solve (12) by solving a slightly larger but sparse linear system as follows. Let $\lambda = DU^T\Delta y$. It is easy to show that (12) is equivalent to the following linear system:

$$\begin{bmatrix} M_{sp} & A^u & U \\ (A^u)^T & 0 & 0 \\ U^T & 0 & -D^{-1} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x^u \\ \lambda \end{bmatrix} = \begin{bmatrix} h \\ R_d^u \\ 0 \end{bmatrix}. \tag{20}$$

We can use the same method described in Section 5.5 to solve (20).

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_j^s corresponding to the semidefinite blocks.

The specialized routine to compute M_j^s 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_j^s and $(z_j^s)^{-1}$ if the HKM direction is used; and they should correspond to the NT scaling matrix w_j^s 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_j^s 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 Δx 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 α. In this subsection, we discuss an efficient strategy to compute the step-length α.

For semidefinite blocks, it is straightforward to verify that, for the jth 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:

$$\alpha_j^s \;=\; \begin{cases} \dfrac{-1}{\lambda_{\min}((x_j^s)^{-1}\Delta x_j^s)}, & \text{if the minimum eigenvalue } \lambda_{\min} \text{ is negative} \\ \infty & \text{otherwise.} \end{cases} \tag{21}$$

If the computation of eigenvalues necessary in α_j^s above becomes expensive, then we resort to finding an approximation of α_j^s by estimating extreme eigenvalues using Lanczos iterations [24]. 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 α_i^q that keeps the next iterate feasible with respect to the ith 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 (9). We want the largest positive $\bar{\alpha}$ for which $a_i\alpha^2 + 2b_i\alpha + c_i > 0$ for all smaller positive α'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 α_k^l for the kth 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 α that can be taken in order for $x + \alpha\Delta x$ to satisfy all the conic constraints takes the form

$$\alpha \;=\; \min\left(1,\; \gamma \min_{1\le j\le n_s} \alpha_j^s,\; \gamma \min_{1\le i\le n_q} \alpha_i^q,\; \gamma \min_{1\le k\le n_l} \alpha_k^l\right), \tag{22}$$

where γ (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 Δz, we let the analog of α_j^s, α_i^q and α_k^l be β_j^s, β_i^q and β_k^l, respectively. Similar to the primal direction, the step-length that can be taken by the dual direction Δz is given by

$$\beta \;=\; \min\left(1,\; \gamma \min_{1\le j\le n_s} \beta_j^s,\; \gamma \min_{1\le i\le n_q} \beta_i^q,\; \gamma \min_{1\le k\le n_l} \beta_k^l\right). \tag{23}$$

6 Distances to infeasibilities

Let $d = (A, c, b)$ be the SQLP data assocated with (P) and (D). Let $F_P(d)$ and $F_D(d)$ be the feasible regions of (P) and (D) respectively. 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 "thickness" of $F_P(d)$ and $F_D(d)$ is the concept of primal distance to infeasibility defined by Renegar [**?**]:

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

With 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, which requires solving $2m$ SQLP problems each with roughly the same dimension and structure as the original dual instance; see [4].

However, one is typically interested in the magnitudes of $\rho_P(d)$ and $\rho_D(d)$ rather than the exact values. It turns out that the following cheaper upper estimates proposed by Freund [**?**] usually give enough information about the size of $F_P(d)$ and $F_D(d)$:

$$\begin{aligned} g_P(d) &= \inf\left\{\max\{\|x\|, \frac{\|x\|}{\mathrm{dist}(x, \partial K)}, \frac{1}{\mathrm{dist}(x, \partial K)}\} \,:\, \mathcal{A}(x) = b,\ \ x \in K\right\} \\ g_D(d) &= \inf\left\{\max\{\|z\|, \frac{\|z\|}{\mathrm{dist}(x, \partial K)}, \frac{1}{\mathrm{dist}(x, \partial K)}\} \,:\, \mathcal{A}^T(y) + z = c,\ \ z \in K^*\right\}. \end{aligned}$$

It is readily shown that $1/g_P(d) \geq \rho_P(d)$ and $1/g_D(d) \geq \rho_D(d)$, and $g_P(d) = \infty \Leftrightarrow \rho_P(d) = 0$, $g_D(d) = \infty \Leftrightarrow \rho_D(d) = 0$.

Freund showed that $g_P(d)$ $(g_D(d))$ can be computed at the cost of solving an SQLP problem with roughly the same dimension and 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);
```

7 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 [11].
7. SOCP problems generated by the MATLAB FIR filter toolbox, available at
   `http://www.csee.umbc.edu/~dschol2/opt.html`

Our results were obtained on a Pentium IV PC (3.0GHz) with 4G of memory running Linux, using MATLAB 7.0. Figure 1 shows the performance of SDPT3-4.0 on a total of about 300 SQLP problems. It shows that SDPT3 was able to solve 80% of the problems to an accuracy of at least 10^{-6} in the measure ϕ defined in (6).

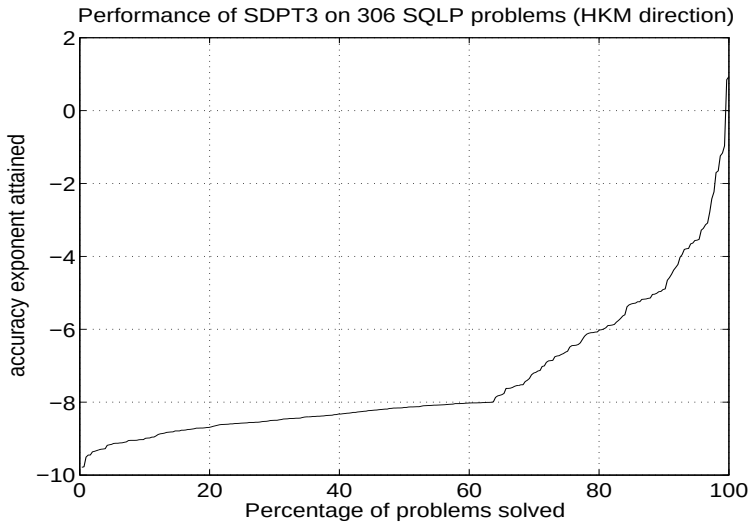

Figure 1: *The accuracy exponent is defined to be* $\log_{10}(\phi)$*, where* ϕ *is defined as in (6).*

DIMACS error measures

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.

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 $\gamma^0 = 0.9$.
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 γ and γ^+ respectively.)

- Compute $\mu(x, z)$ defined in (5), and the accuracy measure ϕ defined in (6). Stop the iteration if ϕ is sufficiently small.
- (Predictor step) Solve the linear system (12) with $\sigma = 0$ in the right-side vector (14). Denote the solution of (10) by $(\delta x, \delta y, \delta z)$. Let α_p and β_p be the step-lengths defined as in (22) and (23) with $\Delta x, \Delta z$ replaced by $\delta x, \delta z$, respectively.
- Take σ 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:

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

- (Corrector step) Solve the linear system (12) with R_c in the the right-hand side vector (14) replaced by

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

  Denote the solution of (10) 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 α and β are computed as in (22) and (23) with γ chosen to be $\gamma = 0.9 + 0.09\min(\alpha_p, \beta_p)$.
- Update the step-length parameter by $\gamma^+ = 0.9 + 0.09\min(\alpha, \beta)$.

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] R.M. Freund, F. Ordóñez, and K.-C. Toh, *Behavioral Measures and their Correlation with IPM Iteration Counts on Semi-Definite Programming Problems*, preprint, 2005.

[5] 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.

[6] 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.

[7] 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.

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

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

[10] 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.

[11] D. Henrion, private communication.

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

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

[14] 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.

[15] 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`

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

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

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

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] T. Tsuchiya, *A polynomial primal-dual path-following algorithm for second-order cone programming*, Technical Report, The Institute of Statistical Mathematics, Minato-Ku, Tokyo, October 1997.

[29] 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.

[30] 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.

[31] 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.

problem	$m \mid n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
arch0	174 | 161; ; 174;	26	-5.66517269-1	-5.66517274-1	5.5-8	3.9-14	2.2-9	2.3-9	05
arch2	174 | 161; ; 174;	24	-6.71515401-1	-6.71515409-1	2.8-9	3.7-14	3.5-9	3.5-9	05
arch4	174 | 161; ; 174;	22	-9.72627407-1	-9.72627420-1	5.4-9	5.0-14	4.4-9	4.4-9	05
arch8	174 | 161; ; 174;	24	-7.05697992 0	-7.05698005 0	1.4-7	4.7-13	8.9-9	6.0-9	05
control1	21 | 15; ; ;	17	-1.77846269 1	-1.77846267 1	5.9-9	7.4-11	-4.0-9	2.0-10	00
control2	66 | 30; ; ;	21	-8.30000364 0	-8.29999999 0	2.0-7	4.1-11	-2.1-7	3.4-10	01
control3	136 | 45; ; ;	21	-1.36332672 1	-1.36332665 1	4.7-7	1.8-10	-2.2-8	1.8-8	03
control4	231 | 60; ; ;	21	-1.97942358 1	-1.97942305 1	3.1-7	4.5-10	-1.3-7	2.2-9	06
control5	351 | 75; ; ;	23	-1.68836048 1	-1.68836008 1	4.3-7	6.1-10	-1.2-7	1.4-7	16
control6	496 | 90; ; ;	22	-3.73043552 1	-3.73044256 1	1.1-6	1.5-9	9.3-7	4.9-7	31
control7	666 | 105; ; ;	24	-2.06250645 1	-2.06250748 1	6.6-7	1.7-9	2.4-7	1.6-7	1:05
control8	861 | 120; ; ;	21	-2.02863293 1	-2.02863711 1	7.7-8	1.9-9	1.0-6	1.0-6	1:39
control9	1081 | 135; ; ;	21	-1.46754201 1	-1.46754272 1	2.9-7	2.0-9	2.3-7	3.6-7	2:48
control10	1326 | 150; ; ;	25	-3.85329154 1	-3.85330586 1	7.4-7	5.1-9	1.8-6	1.9-6	1:41
control11	1596 | 165; ; ;	26	-3.19586089 1	-3.19586886 1	6.4-7	5.0-9	1.2-6	1.2-6	2:45
gpp100	101 | 100; ; ;	15	4.49435469 1	4.49435488 1	2.9-8	1.7-13	-2.2-8	3.7-10	01
gpp124-1	125 | 124; ; ;	17	7.34307507 0	7.34307566 0	6.2-9	4.6-13	-3.8-8	6.7-10	01
gpp124-2	125 | 124; ; ;	15	4.68622922 1	4.68622934 1	3.3-8	2.3-13	-1.3-8	4.1-9	01
gpp124-3	125 | 124; ; ;	14	1.53014124 2	1.53014125 2	3.8-8	1.3-13	-3.5-9	3.8-9	01
gpp124-4	125 | 124; ; ;	17	4.18987602 2	4.18987613 2	4.1-8	3.1-13	-1.2-8	3.5-9	01
gpp250-1	251 | 250; ; ;	17	1.54449168 1	1.54449168 1	1.0-9	2.7-12	-1.4-9	1.5-9	04
gpp250-2	251 | 250; ; ;	16	8.18689572 1	8.18689580 1	2.2-8	4.7-13	-4.8-9	3.3-10	04
gpp250-3	251 | 250; ; ;	15	3.03539317 2	3.03539321 2	8.2-8	4.7-12	-5.2-9	7.0-10	04
gpp250-4	251 | 250; ; ;	14	7.47328311 2	7.47328304 2	8.2-8	2.1-13	4.4-9	8.5-9	04
gpp500-1	501 | 500; ; ;	21	2.53205446 1	2.53205438 1	9.0-11	2.8-10	1.5-8	1.6-8	29
gpp500-2	501 | 500; ; ;	16	1.56060388 2	1.56060387 2	4.4-9	8.4-13	2.1-9	3.2-9	23
gpp500-3	501 | 500; ; ;	15	5.13017612 2	5.13017602 2	2.8-9	9.9-13	9.4-9	9.6-9	22
gpp500-4	501 | 500; ; ;	17	1.56701880 3	1.56701879 3	1.7-8	3.0-13	1.6-9	1.8-9	24
hinf1	13 | 14; ; ;	21	-2.03282194 0	-2.03271083 0	5.8-8	1.8-13	-2.2-5	1.0-8	00
hinf2	13 | 16; ; ;	21	-1.09676614 1	-1.09673573 1	6.2-7	4.4-14	-1.3-5	7.1-9	00

problem	$m \mid n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
hinf3	13 | 16; ; ;	21	-5.69615098 1	-5.69511425 1	8.8-6	1.1-13	-9.0-5	5.8-9	00
hinf4	13 | 16; ; ;	21	-2.74766003 2	-2.74764932 2	1.1-7	8.5-13	-1.9-6	1.8-10	00
hinf5	13 | 16; ; ;	21	-3.62704607 2	-3.62458662 2	**2.9-4**	7.1-13	-3.4-4	4.1-8	00
hinf6	13 | 16; ; ;	21	-4.49067325 2	-4.48998222 2	1.2-5	2.9-13	-7.7-5	1.4-6	00
hinf7	13 | 16; ; ;	19	-3.90831441 2	-3.90821867 2	5.0-5	1.4-13	-1.2-5	4.2-8	00
hinf8	13 | 16; ; ;	21	-1.16170056 2	-1.16157996 2	4.5-5	1.7-13	-5.2-5	3.2-7	00
hinf9	13 | 16; ; ;	21	-2.36249283 2	-2.36249258 2	5.9-6	2.4-14	-5.2-8	3.7-10	00
hinf10	21 | 18; ; ;	32	-1.08872461 2	-1.08792100 2	1.1-6	3.5-11	-3.7-4	1.4-7	01
hinf11	31 | 22; ; ;	34	-6.59694072 1	-6.59157089 1	5.0-7	2.7-11	-4.0-4	5.5-8	01
hinf12	43 | 24; ; ;	46	-1.47406847-1	-7.37049304-2	3.5-10	2.7-9	-6.0-2	2.0-9	01
hinf13	57 | 30; ; ;	34	-4.45474269 1	-4.44451072 1	7.9-5	1.2-11	-1.1-3	4.5-6	01
hinf14	73 | 34; ; ;	26	-1.29918409 1	-1.29908452 1	5.2-7	7.2-12	-3.7-5	6.1-8	01
hinf15	91 | 37; ; ;	29	-2.43319484 1	-2.41545357 1	3.5-5	1.9-11	-3.6-3	**5.1-4**	02
infd1	10 | 30; ; ;	5	-5.13280346 0	1.5952217617		primal infeasible			00
infd2	10 | 30; ; ;	5	5.35505784 0	5.2954130616		primal infeasible			00
infp1	10 | 30; ; ;	7	-5.64104972 9	-7.62162314 0		dual infeasible			00
infp2	10 | 30; ; ;	7	-3.85845797 9	-6.90007199 0		dual infeasible			00
mcp100	100 | 100; ; ;	12	-2.26157350 2	-2.26157352 2	2.3-10	3.0-16	4.3-9	4.3-9	01
mcp124-1	124 | 124; ; ;	13	-1.41990477 2	-1.41990477 2	2.7-11	2.7-16	4.4-10	4.5-10	01
mcp124-2	124 | 124; ; ;	13	-2.69880170 2	-2.69880171 2	5.5-12	4.0-16	5.4-10	5.4-10	01
mcp124-3	124 | 124; ; ;	12	-4.67750108 2	-4.67750115 2	3.7-13	4.6-16	6.7-9	6.7-9	01
mcp124-4	124 | 124; ; ;	13	-8.64411863 2	-8.64411864 2	2.5-11	4.8-16	6.1-10	6.2-10	01
mcp250-1	250 | 250; ; ;	14	-3.17264340 2	-3.17264340 2	5.4-12	4.7-16	1.1-9	1.1-9	02
mcp250-2	250 | 250; ; ;	13	-5.31930080 2	-5.31930084 2	6.3-11	5.1-16	3.4-9	3.4-9	02
mcp250-3	250 | 250; ; ;	13	-9.81172565 2	-9.81172572 2	5.9-11	6.0-16	3.6-9	3.6-9	02
mcp250-4	250 | 250; ; ;	14	-1.68196009 3	-1.68196011 3	6.3-12	0.8-15	6.2-9	6.2-9	03
mcp500-1	500 | 500; ; ;	15	-5.98148516 2	-5.98148517 2	4.5-12	5.2-16	5.9-10	5.9-10	07
mcp500-2	500 | 500; ; ;	16	-1.07005676 3	-1.07005677 3	9.0-12	6.7-16	1.3-9	1.3-9	11
mcp500-3	500 | 500; ; ;	15	-1.84797002 3	-1.84797002 3	7.4-12	0.9-15	2.5-10	2.5-10	12
mcp500-4	500 | 500; ; ;	13	-3.56673799 3	-3.56673805 3	1.9-11	1.0-15	8.5-9	8.5-9	11
qap5	136 | 26; ; ;	10	4.36000000 2	4.36000000 2	1.2-10	7.3-14	4.6-10	6.1-10	00

problem	$m \mid n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
qap6	229 | 37; ; ;	19	3.81432573 2	3.81435498 2	3.3-8	9.6-13	-3.8-6	2.3-9	01
qap7	358 | 50; ; ;	13	4.24830442 2	4.24810252 2	1.8-8	5.3-13	2.4-5	3.3-5	01
qap8	529 | 65; ; ;	20	7.56939331 2	7.56947355 2	2.0-7	3.1-12	-5.3-6	7.4-9	05
qap9	748 | 82; ; ;	20	1.40993093 3	1.40993599 3	3.8-8	1.3-12	-1.8-6	4.4-9	09
qap10	1021 | 101; ; ;	21	1.09258508 3	1.09259635 3	1.0-7	1.1-12	-5.2-6	2.3-9	17
ss30	132 | 294; ; 132;	21	-2.02395100 1	-2.02395106 1	4.2-7	2.3-13	1.4-8	1.0-8	22
theta1	104 | 50; ; ;	11	-2.29999996 1	-2.30000001 1	9.9-12	5.6-15	9.5-9	9.5-9	00
theta2	498 | 100; ; ;	14	-3.28791689 1	-3.28791690 1	6.2-13	1.2-14	1.4-9	1.4-9	01
theta3	1106 | 150; ; ;	14	-4.21669813 1	-4.21669815 1	2.6-12	1.4-14	2.1-9	2.1-9	05
theta4	1949 | 200; ; ;	14	-5.03212213 1	-5.03212220 1	3.4-14	2.1-14	7.8-9	7.8-9	16
theta5	3028 | 250; ; ;	14	-5.72323069 1	-5.72323073 1	1.9-13	2.9-14	3.9-9	3.9-9	49
theta6	4375 | 300; ; ;	14	-6.34770870 1	-6.34770872 1	1.7-12	3.2-14	1.9-9	1.9-9	2:10
truss1	6 | 12; ; 1;	11	8.99999632 0	8.99999631 0	5.8-12	1.0-15	2.7-10	2.7-10	00
truss2	58 | 132; ; 1;	17	1.23380356 2	1.23380356 2	3.8-10	7.2-15	3.8-10	3.8-10	01
truss3	27 | 30; ; 1;	12	9.10999622 0	9.10999619 0	9.0-12	5.7-16	1.6-9	1.6-9	00
truss4	12 | 18; ; 1;	10	9.00999632 0	9.00999626 0	6.0-12	1.2-15	3.1-9	3.1-9	00
truss5	208 | 330; ; 1;	17	1.32635678 2	1.32635678 2	9.1-11	1.0-14	9.3-10	8.9-10	02
truss6	172 | 450; ; 1;	28	9.01001420 2	9.01001363 2	5.0-8	2.3-13	3.1-8	4.4-8	02
truss7	86 | 300; ; 1;	27	9.00001403 2	9.00001391 2	5.3-8	2.4-13	6.9-9	2.2-8	02
truss8	496 | 627; ; 1;	16	1.33114589 2	1.33114588 2	1.4-10	9.3-15	3.5-9	3.5-9	04
maxG11	800 | 800; ; ;	15	-6.29164777 2	-6.29164783 2	2.8-11	7.4-16	5.2-9	5.2-9	26
maxG32	2000 | 2000; ; ;	15	-1.56763962 3	-1.56763964 3	1.3-10	1.1-15	9.1-9	9.1-9	4:21
maxG51	1000 | 1000; ; ;	17	-4.00625552 3	-4.00625552 3	2.6-12	3.6-16	2.8-10	2.8-10	1:06
qpG11	800 | 1600; ; ;	15	-2.44865909 3	-2.44865913 3	3.3-11	0.0-16	8.6-9	8.6-9	26
qpG51	1000 | 2000; ; ;	17	-1.18179999 4	-1.18180000 4	2.3-10	0.0-16	2.2-9	2.2-9	1:03
thetaG11	2401 | 801; ; ;	19	-3.99999995 2	-4.00000000 2	5.0-9	1.7-13	6.8-9	6.8-9	1:33
thetaG51	6910 | 1001; ; ;	34	-3.48999315 2	-3.49000166 2	1.4-10	6.8-13	1.2-6	1.2-6	37:14
equalG11	801 | 801; ; ;	16	-6.29155280 2	-6.29155293 2	8.7-10	1.0-15	9.9-9	9.9-9	1:17
equalG51	1001 | 1001; ; ;	18	-4.00560126 3	-4.00560132 3	6.5-8	1.3-15	7.4-9	7.5-9	2:38
bm1	883 | 882; ; ;	20	2.34398777 1	2.34398186 1	2.2-6	1.8-11	1.2-6	1.2-6	2:04
copo14	1275 | 196; ; 364;	17	2.62856497-10	-6.99780924-10	2.5-11	6.2-15	9.6-10	9.6-10	08

problem	m | $n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
copo23	5820 | 529; ; 1771;	21	2.56114272-9	-6.38325819-9	2.2-12	8.1-15	8.9-9	8.9-9	18:20
hamming-7-	1793 | 128; ; ;	9	-4.26666666 1	-4.26666667 1	6.8-11	0.0-16	2.7-10	2.0-10	10
hamming-9-	2305 | 512; ; ;	9	-2.23999997 2	-2.24000001 2	3.4-13	8.4-14	8.3-9	8.3-9	24
minphase	48 | 48; ; ;	29	5.97872628 0	5.98119359 0	1.3-8	5.7-13	-1.9-4	4.8-7	01
torusg3-8	512 | 512; ; ;	15	-4.83409459 7	-4.83409459 7	3.4-12	7.8-16	9.5-10	9.5-10	11
toruspm3-8	512 | 512; ; ;	14	-5.27808660 2	-5.27808663 2	6.9-11	6.0-16	2.8-9	2.8-9	10
torusg3-15	3375 | 3375; ; ;	16	-6.37621845 3	-6.37621855 3	1.2-9	1.5-15	7.6-9	7.6-9	23:41
toruspm3-1	3375 | 3375; ; ;	16	-3.47513185 3	-3.47513186 3	2.1-11	1.5-15	2.0-9	2.0-9	23:46
filter48	969 | 48; 49; 931;	43	1.41612972 0	1.41612914 0	2.9-7	8.9-14	1.5-7	1.2-8	1:01
filtinf1	983 | 49; 49; 945;	33	0.00000000-16	2.98905416 1	primal infeasible				48
nb	123 | ; 2379; 4;	22	-5.07030865-2	-5.07030948-2	3.1-13	6.1-16	7.5-9	7.5-9	07
nb-L1	915 | ; 2379; 797;	23	-1.30122706 1	-1.30122709 1	1.6-10	9.7-14	9.0-9	9.0-9	13
nb-L2	123 | ; 4191; 4;	17	-1.62897195 0	-1.62897198 0	1.7-10	1.9-15	7.2-9	7.2-9	10
nb-L2-bess	123 | ; 2637; 4;	19	-1.02569502-1	-1.02569511-1	1.0-13	1.1-15	7.4-9	7.4-9	07
nql30	3680 | ; 2700; 3602;	35	-9.46028502-1	-9.46028517-1	5.2-11	2.6-10	4.9-9	5.7-9	04
nql60	14560 | ; 10800; 14402;	33	-9.35052951-1	-9.35052975-1	5.0-11	7.4-11	8.2-9	9.4-9	17
nql180	130080 | ; 97200; 129602;	39	-9.27728621-1	-9.27728643-1	7.8-10	2.0-11	7.8-9	8.8-9	4:38
qssp30	3691 | ; 7564; 2;	17	-6.49667573 0	-6.49667575 0	1.9-10	1.1-13	1.5-9	1.5-9	03
qssp60	14581 | ; 29524; 2;	23	-6.56270608 0	-6.56270649 0	2.9-8	4.0-15	2.8-8	3.4-9	17
qssp180	130141 | ; 261364; 2;	28	-6.63960843 0	-6.63961086 0	3.1-7	1.0-14	1.7-7	4.2-9	6:31
sched-50-5	2527 | ; 2477; 2502;	31	2.66753929 4	2.66728864 4	1.5-5	1.5-7	4.7-5	4.7-5	03
sched-100-	4844 | ; 4744; 5002;	34	1.81900808 5	1.81887534 5	**3.4-4**	1.1-9	3.6-5	3.6-5	07
sched-100-	8338 | ; 8238; 10002;	29	7.17597254 5	7.17350508 5	**1.4-2**	1.0-10	**1.7-4**	**1.7-4**	13
sched-200-	18087 | ; 17887; 20002;	35	1.41946190 5	1.41176207 5	**1.3-4**	5.6-7	**2.7-3**	**2.7-3**	46
sched-50-5	2526 | ; 2475; 2502;	27	7.85203874 0	7.85203843 0	2.3-7	4.3-15	1.9-8	1.9-8	03
sched-100-	4843 | ; 4742; 5002;	29	6.71650648 1	6.71650261 1	1.3-7	8.6-14	2.9-7	2.9-7	06
sched-100-	8337 | ; 8236; 10002;	29	2.73307879 1	2.73307855 1	1.1-7	2.8-14	4.3-8	4.3-8	12
sched-200-	18086 | ; 17885; 20002;	37	5.18119615 1	5.18119610 1	1.0-7	1.2-13	4.5-9	4.5-9	44
biggs	1819 | 702; ; ;	53	-1.41425775 3	-1.41425841 3	3.4-9	1.9-11	2.3-7	4.7-9	1:30
buck1	36 | 49; ; 36;	17	-1.46419152 2	-1.46419152 2	8.2-10	3.2-14	7.7-10	8.1-10	01
buck2	144 | 193; ; 144;	21	-2.92368313 2	-2.92368295 2	8.3-7	7.0-14	-3.1-8	2.0-8	04

problem	m | $n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
buck3	544 | 641; ; 544;	28	-6.07589498 2	-6.07608699 2	**1.2-4**	2.1-13	1.6-5	1.7-5	20
buck4	1200 | 1345; ; 1200;	33	-4.86141048 2	-4.86143830 2	9.9-6	4.0-13	2.9-6	2.8-6	2:12
buck5	3280 | 3521; ; 3280;	37	-4.36170859 2	-4.36283022 2	2.2-5	6.5-13	**1.3-4**	**1.3-4**	25:33
cnhil10	5005 | 220; ; ;	27	0.00000000-16	-1.65482516-4	6.4-8	3.2-12	**1.7-4**	2.9-7	55
cnhil8	1716 | 120; ; ;	25	0.00000000-16	-1.78946733-5	1.8-8	1.1-12	1.8-5	5.0-8	07
cphil10	5005 | 220; ; ;	9	0.00000000-16	-8.11544501-9	1.1-14	0.0-16	8.1-9	8.1-9	31
cphil12	12376 | 364; ; ;	10	0.00000000-16	-2.93810832-10	1.5-14	0.0-16	2.9-10	2.9-10	4:01
G40-mb	2001 | 2000; ; ;	22	-2.86432310 3	-2.86432323 3	9.1-10	2.1-11	2.1-8	2.1-8	21:04
G40mc	2000 | 2000; ; ;	18	-5.72957909 3	-5.72957911 3	1.1-10	5.8-16	1.1-9	1.1-9	7:07
G48mc	3000 | 3000; ; ;	12	-1.20000000 4	-1.20000000 4	5.6-12	5.7-15	4.2-10	4.2-10	10:06
G55mc	5000 | 5000; ; ;	16	-2.20789206 4	-2.20789208 4	9.6-10	2.2-15	4.7-9	4.7-9	1:14:47
G59mc	5000 | 5000; ; ;	19	-1.46246530 4	-1.46246531 4	6.6-10	1.0-15	4.7-9	4.7-9	1:36:36
mater-1	103 | 220; ; 2;	21	1.43465439 2	1.43465438 2	1.4-10	1.6-14	3.5-9	3.5-9	01
mater-2	423 | 1012; ; 2;	20	1.41591867 2	1.41591866 2	6.3-12	7.2-14	1.1-9	1.1-9	04
mater-3	1439 | 3586; ; 2;	24	1.33916258 2	1.33916256 2	7.3-11	2.5-13	9.6-9	9.7-9	16
mater-4	4807 | 12496; ; 2;	29	1.34262716 2	1.34262716 2	1.6-9	8.8-13	2.6-9	3.8-9	1:11
mater-5	10143 | 26818; ; 2;	30	1.33801638 2	1.33801640 2	9.9-9	1.8-12	-6.4-9	4.4-9	3:03
mater-6	20463 | 54626; ; 2;	32	1.33538673 2	1.33538714 2	6.2-8	3.1-12	-1.5-7	8.0-9	8:21
neosfbr12	1441 | 122; ; ;	17	5.29319164 2	5.29319158 2	3.5-11	0.8-15	5.7-9	5.7-9	18
neosfbr20									
neosfbr21									
neosfbr22									
neosfbr24									
neosfbr25									
neosfbr30e									
neu1	3003 | 252; ; 2;	31	4.36376962-7	-1.34195143-5	7.3-10	2.3-11	1.4-5	3.6-5	8:21
neu2	3003 | 252; ; 2;	40	-1.61121339-4	-2.36860767-4	1.6-8	6.8-11	7.6-5	**6.5-4**	10:21
neu2c	3002 | 1253; ; 2;	59	3.41771574 4	3.40921240 4	**3.8-3**	2.5-11	**1.2-3**	**2.0-4**	32:28
neu2g	3002 | 252; ; ;	38	3.40999994 4	3.40999997 4	3.0-9	1.1-10	-5.3-9	1.4-8	9:12
neu3	7364 | 418; ; 2;	46	1.33597463-7	-1.96267080-4	2.4-9	8.7-10	**2.0-4**	2.7-6	8:01
neu3g	8007 | 462; ; ;	47	9.25315729-9	-1.56570129-4	1.6-9	7.7-10	**1.6-4**	2.0-7	11:02

problem	$m \mid n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
r1-6-0	601 | 600; ; ;	21	-5.57968708 4	-5.57968706 4	6.5-7	5.9-11	-1.6-9	1.3-10	41
r1-6-1	601 | 600; ; ;	14	-5.58043922 4	-5.58043924 4	2.8-10	2.1-14	2.2-9	2.2-9	29
r1-6-1e-6	601 | 600; ; ;	14	-5.57968722 4	-5.57968731 4	1.1-7	6.7-12	8.3-9	7.9-9	29
rose13	2379 | 105; ; ;	31	1.20000012 1	1.19999998 1	8.4-9	5.9-16	5.6-8	7.4-9	1:08
rose15	3860 | 135; ; 2;	38	1.82090663-5	-1.26745042-5	7.3-8	1.7-13	3.1-5	5.7-8	5:44
sdmint3	5255 | 379; 5255; ;	29	-4.78135763 3	-4.78131886 3	**2.5-4**	9.4-14	-4.1-6	9.3-6	47:47
shmup1	16 | 81; ; 32;	17	-1.88414831 2	-1.88414832 2	1.1-9	3.0-15	2.8-9	2.8-9	01
shmup2	200 | 881; ; 400;	39	-3.46242668 3	-3.46242684 3	1.0-6	7.3-15	2.3-8	2.6-8	57
shmup3	420 | 1801; ; 840;	31	-2.09883698 3	-2.09883796 3	2.3-6	6.5-15	2.3-7	2.2-7	4:06
shmup4	800 | 3361; ; 1600;	38	-7.99254514 3	-7.99255230 3	7.8-6	7.4-15	4.5-7	4.5-7	20:54
shmup5	1800 | 7441; ; 3600;	36	-2.38576762 4	-2.38591658 4	6.5-6	3.1-14	3.1-5	3.1-5	2:36:00
taha1a	3002 | 1680; ; ;	27	-9.31533327-1	-1.25589295 0	**9.2-2**	4.7-12	**1.0-1**	**9.0-2**	15:27
taha1b	8007 | 1606; ; 3;	37	-7.73313007-1	-7.73313020-1	2.5-11	3.2-15	4.9-9	5.9-9	47:32
trto1	36 | 25; ; 36;	21	-1.10450000 3	-1.10450000 3	4.4-8	4.5-14	6.4-10	3.5-10	01
trto2	144 | 97; ; 144;	21	-1.27999978 4	-1.28000007 4	3.7-7	1.2-12	1.1-7	1.1-7	01
trto3	544 | 321; ; 544;	24	-1.27999344 4	-1.28000857 4	**3.7-4**	3.1-12	5.9-6	5.9-6	09
trto4	1200 | 673; ; 1200;	27	-1.27606346 4	-1.27701954 4	**1.4-3**	7.4-12	**3.7-4**	**3.7-4**	52
trto5	3280 | 1761; ; 3280;	29	-1.27928272 4	-1.28021188 4	**5.7-4**	1.7-11	**3.6-4**	**3.6-4**	10:43
vibra1	36 | 49; ; 36;	13	-4.08190124 1	-4.08190124 1	3.3-10	3.2-15	4.7-10	4.8-10	01
vibra2	144 | 193; ; 144;	23	-1.66015338 2	-1.66015364 2	1.2-5	2.0-14	8.0-8	8.8-8	05
vibra3	544 | 641; ; 544;	29	-1.72605378 2	-1.72614934 2	**1.1-4**	8.0-14	2.8-5	2.8-5	20
vibra4	1200 | 1345; ; 1200;	29	-1.27636319 4	-1.27676506 4	**1.5-3**	9.4-12	**1.6-4**	**1.6-4**	1:50
vibra5	3280 | 3521; ; 3280;	57	-1.65803004 2	-1.65935491 2	7.2-5	2.6-13	**4.0-4**	**4.0-4**	39:13
yalsdp	5051 | 300; ; ;	13	-1.79212601 0	-1.79212676 0	1.5-11	1.6-13	1.6-7	1.6-7	10:33
cancer-100	10469 | 569; ; ;	14	-2.76091613 4	-2.76233961 4	6.0-6	9.8-12	**2.6-4**	**3.0-4**	26:16
checker-1.	3970 | 3970; ; ;	24	3.30388458 3	3.30388454 3	3.6-10	0.1-16	6.4-9	6.4-9	45:04
foot	2209 | 2208; ; ;	20	-5.85252481 5	-5.85298195 5	5.8-5	2.0-8	3.9-5	3.9-5	24:57
hand	1297 | 1296; ; ;	21	-2.47477792 4	-2.47477791 4	6.0-7	2.4-10	-1.4-9	1.1-10	5:31
inc-600	2515 | 600; ; 2514;	25	-6.60104091-1	-6.68915553-1	2.1-5	1.5-9	**3.8-3**	**4.1-3**	2:07
inc-1200	5175 | 1200; ; 5174;	15	2.12231915 1	-1.41967770 0	2.0-5	1.1-9	**9.6-1**	**9.6-1**	7:16
swissroll	3380 | 800; ; ;	31	-5.48284431 5	-5.59816094 5	**2.7-3**	6.4-9	**1.0-2**	**1.1-1**	5:30

problem	$m \mid n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
tiger-text	1802 | 1801; ; ;	25	3.48084179 2	3.44294211 2	2.1-5	1.7-8	**5.5-3**	**1.5-2**	6:22
diamond-pa	5478 | 5477; ; ;	31	3.22638853 1	1.56694179 1	1.6-5	1.6-7	**3.4-1**	**1.2 0**	2:45:18
ice-2.0	8113 | 8113; ; ;	28	6.80838634 3	6.80838622 3	1.1-10	0.0-16	9.1-9	9.1-9	7:13:27
p-auss2-3.									
butcher	6434 | 330; ; 22512;	43	-1.41958589 1	-1.40018649 1	**3.7-2**	1.1-6	-6.6-3	**1.7-3**	45:05
rabmo	5004 | 220; ; 6606;	47	-3.72725362 0	-3.72725497 0	6.2-6	5.1-8	1.6-7	1.7-5	15:53
reimer5	6187 | 462; ; 102144;	24	-1.51824956 1	-1.51835452 1	**1.0-3**	1.0-9	3.3-5	**1.6-4**	59:43
chs-500	9974 | 4980; ; ;	23	9.32785242-10	-8.36219073-9	8.2-14	0.0-16	9.3-9	9.3-9	30
nonc-500	4990 | 2998; ; ;	22	6.25761159-2	6.25597999-2	1.3-8	2.2-15	1.5-5	1.4-5	10
ros-500	4988 | 2992; ; ;	17	2.49499943 0	2.49499945 0	1.2-8	1.8-15	-3.5-9	3.1-9	08
fp210	1000 | 176; ; ; 66	25	3.74999982-1	3.74999999-1	1.3-7	3.9-10	-10.0-9	4.4-9	09
fp22	14 | 15; ; ;	12	-7.99999999 0	-8.00000007 0	4.2-10	1.4-15	4.7-9	6.9-9	00
fp23	209 | 119; ; ;	29	2.13000000 2	2.12999999 2	2.0-11	1.2-13	1.3-9	2.0-9	02
fp24	2379 | 595; ; ;	22	1.95000000 2	1.95000000 2	1.0-11	3.5-13	1.1-9	1.4-9	56
fp25	209 | 133; ; ;	18	1.10000000 1	1.10000000 1	1.2-12	1.2-14	8.7-10	1.1-9	01
fp26	1000 | 407; ; ;	22	2.68014631 2	2.68014631 2	9.5-10	3.0-15	4.5-10	7.7-10	10
fp27	1000 | 341; ; ;	21	3.90000000 1	3.89999997 1	4.2-10	4.7-13	4.5-9	6.4-9	09
fp32	3002 | 1155; ; ;	44	-7.04227560 0	-7.05278292 0	**4.4-3**	9.7-12	**7.0-4**	**2.9-4**	7:05
fp33	125 | 117; ; ;	39	-1.01265940 4	-1.01266024 4	2.6-10	4.3-11	4.1-7	2.5-9	02
fp34	209 | 140; ; ;	22	1.72000000 2	1.72000000 2	1.2-12	1.0-13	-2.9-10	6.2-10	01
fp35	164 | 195; ; ;	18	3.99999993 0	3.99999965 0	4.2-9	1.1-13	3.0-8	5.3-8	03
fp410	14 | 18; ; ; 1	16	1.67388932 1	1.67388931 1	1.6-11	4.7-16	3.3-9	3.6-9	00
fp42	6 | 10; ; ;	10	7.58731237 0	7.58731235 0	6.5-12	6.4-16	1.1-9	1.1-9	00
fp43	50 | 76; ; ;	16	6.54776263 2	6.63152380 2	**2.5-4**	2.1-10	-6.4-3	1.7-6	01
fp44	6 | 10; ; ;	22	4.43671691 2	4.43671704 2	2.7-8	4.5-14	-1.5-8	1.9-9	00
fp45	4 | 7; ; ;	13	1.24858128-9	-5.52323520-10	1.1-9	5.0-16	1.8-9	2.4-9	00
fp46	27 | 22; ; ;	21	9.94242098-10	-4.64240829-9	1.3-10	2.1-16	5.6-9	7.3-9	00
fp47	6 | 10; ; ;	16	2.43000000 2	2.43000000 2	5.3-11	1.9-14	2.1-10	1.6-10	00
fp48	4 | 7; ; ;	9	7.50000000 0	7.50000000 0	2.1-11	1.5-16	3.1-10	3.0-10	00
fp49	14 | 18; ; ; 1	16	1.67388932 1	1.67388931 1	1.6-11	4.7-16	3.3-9	3.6-9	00
l1	14 | 6; ; ;	8	4.92634657-1	4.92634644-1	4.3-12	0.2-16	6.5-9	6.5-9	00

problem	$m \mid n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
l2	14 | 6; ; ;	9	1.14580631 1	1.14580630 1	5.1-11	2.2-16	1.5-9	1.5-9	00
l4	152 | 45; ; ;	19	3.70370379-2	3.70371914-2	4.0-10	1.4-14	-1.4-7	8.6-9	01
l5	14 | 15; ; ;	12	-7.99999999 0	-8.00000007 0	4.2-10	1.4-15	4.7-9	6.9-9	00
5n	31 | 26; ; ;	9	2.24000000 0	2.23999999 0	6.8-12	2.6-16	2.8-9	2.8-9	00
a12	793 | 79; ; ;	12	2.10000000 1	2.10000000 1	3.8-11	1.6-15	9.3-10	9.3-10	03
aw29	465 | 130; ; ;	11	3.00000000 0	3.00000000 0	6.8-11	4.9-16	1.1-9	1.2-9	06
c5	31 | 26; ; ;	8	1.50000001 0	1.49999999 0	1.1-11	1.1-16	3.9-9	4.0-9	00
fp1131	847 | 176; ; ;	11	4.50000001 0	4.49999996 0	1.4-12	1.6-15	4.9-9	4.9-9	22
fp1132	847 | 176; ; ;	12	1.55000000 1	1.54999999 1	5.9-14	5.2-15	3.8-9	3.8-9	24
fp1133	847 | 176; ; ;	12	1.75000000 1	1.75000000 1	5.4-14	5.4-15	1.7-9	1.7-9	24
fp1134	847 | 176; ; ;	12	1.95000000 1	1.94999999 1	4.8-14	5.1-15	3.0-9	3.0-9	24
fp1135	847 | 176; ; ;	12	2.20000000 1	2.20000000 1	1.5-11	1.7-15	8.8-10	8.8-10	24
fp1136	847 | 176; ; ;	12	1.45000000 1	1.44999999 1	5.9-14	5.1-15	4.8-9	4.8-9	24
fp1137	847 | 176; ; ;	12	1.65000000 1	1.64999999 1	8.3-14	4.1-15	2.5-9	2.5-9	24
fp1138	847 | 176; ; ;	13	1.75000000 1	1.75000000 1	5.9-14	5.1-15	6.8-10	6.8-10	26
fp1139	847 | 176; ; ;	12	2.30000000 1	2.29999999 1	2.4-12	2.6-15	3.1-9	3.1-9	24
k5	31 | 31; ; ;	8	1.00000001 0	9.99999995-1	3.7-11	2.1-16	4.5-9	4.5-9	00
p10	847 | 176; ; ;	11	4.50000001 0	4.49999996 0	2.9-12	1.3-15	4.9-9	4.9-9	22
bifur	454 | 84; ; ; 1661	21	-3.37301697-1	-3.37301697-1	2.1-9	2.1-10	-3.4-11	1.7-9	04
boom	3002 | 210; ; ; 8764	29	-3.23707245 2	-3.23707248 2	1.1-8	1.2-6	4.6-9	8.5-7	4:00
brown	461 | 56; ; ; 925	34	3.04680725-11	0.00000000-16	1.7-11	3.0-11	3.0-11	3.8-9	04
butcher	6434 | 330; ; ; 11256	48	-1.39810186 1	-1.39324497 1	4.4-6	**6.1-4**	-1.7-3	**2.2-2**	47:32
camera1s	209 | 28; ; ; 168	57	-1.78688795 4	-1.78686202 4	**2.0-4**	6.0-8	-7.3-6	8.3-6	02
caprasse	209 | 35; ; ; 60	17	-2.36780178-1	-2.36780183-1	9.3-11	2.1-9	3.8-9	6.8-9	01
cassou	4844 | 495; ; ; 35126	27	-3.8317999911	-6.15988262 6		dual infeasible			1:17:04
cdpm5	125 | 21; ; ; 5	11	4.81891306-10	-7.57760884-9	5.4-15	0.0-16	8.1-9	8.1-9	00
chemequ	461 | 56; ; ; 525	75	-7.55021050 8	-4.87753741 7		dual infeasible			07
chemequs	125 | 21; ; ; 45	15	-1.46594733 7	-1.25336541 5		dual infeasible			00
cohn2	209 | 35; ; ; 4	41	2.44989647-10	-1.17751717-8	2.7-7	1.1-8	1.2-8	1.2-8	01
cohn3	209 | 35; ; ; 4	15	1.34558839-11	-4.75678015-10	2.9-13	4.1-15	4.9-10	4.9-10	01
comb3000	1000 | 66; ; ; 595	19	-4.74520651-10	-6.87418379-10	1.6-11	4.4-10	2.1-10	2.2-9	05

problem	$m \mid n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
conform1	83 | 20; ; ; 30	13	-6.04355528 8	-6.64897488 3		dual infeasible			00
conform3	285 | 56; ; ; 630	22	-9.45599155-12	0.00000000-16	1.8-9	5.0-11	-9.5-12	6.7-10	01
conform4	454 | 84; ; ; 1890	21	3.66213726-11	0.00000000-16	2.6-9	2.0-11	3.7-11	3.0-9	04
des18-3	12869 | 495; ; ; 58920	82	-3.23787125 6	-2.76403795 6		dual infeasible			8:58:52
des22-24	1000 | 66; ; ; 660	23	-6.74166365 3	-6.74166365 3	1.6-10	6.9-9	4.5-11	1.0-9	06
discret3	44 | 9; ; ; 8	11	-3.70033135 1	-3.70033140 1	3.2-11	3.4-15	6.7-9	6.7-9	00
eco5	461 | 56; ; ; 525	19	-1.20463311 3	-1.20463311 3	2.5-9	2.6-9	-2.2-12	8.1-10	02
eco6	923 | 84; ; ; 924	26	-1.00281559 4	-1.00281559 4	4.7-11	3.1-9	5.8-11	3.7-10	09
eco7	1715 | 120; ; ; 1512	24	-3.91531048 3	-3.91531047 3	7.3-9	2.0-9	-4.0-10	4.8-10	32
eco8	3002 | 165; ; ; 2340	26	-5.82038418 3	-5.82038418 3	2.7-10	8.1-9	-4.2-12	1.5-10	2:15
fourbar	69 | 15; ; ; 4	11	1.16235821-10	-4.24153800-9	4.7-12	1.8-12	4.4-9	4.5-9	00
geneig	923 | 84; ; ; 546	16	-2.52663014 0	-2.52663014 0	5.6-12	1.5-9	9.6-11	7.9-9	05
heart	3002 | 165; ; ; 4320	17	-8.70927422 1	-8.70927422 1	2.5-11	8.1-9	2.2-11	4.6-9	1:29
i1	1000 | 66; ; ; 10	12	-1.66775273 0	-1.66775273 0	2.3-13	0.9-16	8.6-10	8.6-10	04
ipp	494 | 45; ; ; 360	30	-1.31158853 1	-1.31158853 1	9.9-10	1.5-9	-6.0-11	2.1-9	02
katsura5	209 | 28; ; ; 168	22	-8.16044579-2	-8.16044579-2	3.3-9	1.3-10	3.6-11	2.0-9	01
kinema	714 | 55; ; ; 495	28	-4.19683963 4	-4.19683963 4	2.8-8	4.6-9	-1.0-10	1.2-9	04
ku10	1000 | 66; ; ; 660	24	-7.13900000 3	-7.13900000 3	2.0-10	2.9-9	9.0-11	2.0-9	07
lorentz	69 | 15; ; ; 60	16	-5.00000000 0	-5.00000000 0	5.9-12	2.7-9	6.1-11	2.0-9	00
manocha	90 | 28; ; ; 42	27	-2.45943538-1	-2.45956385-1	4.0-8	1.6-5	8.6-6	**8.9-4**	01
noon3	83 | 20; ; ; 30	11	-2.08695033 1	-2.08695034 1	3.6-13	1.6-9	1.2-9	2.6-9	00
noon4	209 | 35; ; ; 60	15	-1.71283759 1	-1.71283761 1	1.9-11	1.8-9	5.7-9	8.1-9	01
noon5	461 | 56; ; ; 105	17	-1.58524243 1	-1.58524243 1	1.7-12	4.9-9	3.3-10	2.1-9	02
proddeco	69 | 15; ; ; 4	11	1.84597568-11	-3.37004061-10	4.3-14	0.0-16	3.6-10	3.6-10	00
puma	3002 | 165; ; ; 8280	37	-3.05299490 1	-3.05299490 1	2.4-8	2.4-9	-7.9-10	1.7-9	3:16
quadfor2	209 | 35; ; ; 270	19	-6.18518518 0	-6.18518519 0	2.8-12	4.3-9	6.7-10	4.5-9	01
quadgrid	461 | 56; ; ; 505	82	-2.96150529 7	-2.95257544 7		dual infeasible			11
rabmo	5004 | 220; ; ; 3303	59	-3.72725312 0	-3.72725312 0	8.6-8	3.9-10	-3.9-10	2.3-9	19:16
rbpl	923 | 84; ; ; 546	18	-7.94063377 0	-7.94063378 0	5.1-10	9.8-10	5.2-10	9.8-9	06
redeco5	20 | 6; ; ; 5	8	-2.53906250-1	-2.53906251-1	1.6-15	0.4-16	1.2-9	1.2-9	00
redeco6	27 | 7; ; ; 6	8	-2.01600000-1	-2.01600002-1	3.4-14	1.1-16	1.6-9	1.6-9	00

problem	$m \mid n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
redeco7	35 | 8; ; ; 7	8	-1.67438271-1	-1.67438274-1	1.5-14	0.5-16	2.0-9	2.0-9	00
redeco8	44 | 9; ; ; 8	8	-1.43273636-1	-1.43273639-1	9.2-15	0.7-16	2.4-9	2.4-9	00
rediff3	9 | 4; ; ; 3	9	4.04469621-10	-1.21340922-9	1.0-15	0.0-16	1.6-9	1.6-9	00
rose	679 | 120; ; ; 2281	31	-1.72122696 0	-1.70726881 0	1.0-6	**3.4-4**	-3.2-3	**3.7-2**	18
s9-1	494 | 45; ; ; 360	21	-4.27369565 0	-4.27369565 0	5.3-11	5.8-10	2.0-10	5.8-9	02
sendra	65 | 21; ; ; 12	11	-2.37687542 1	-2.37687543 1	2.4-10	3.4-12	2.5-9	2.8-9	00
solotarev	69 | 15; ; ; 32	14	-5.88961333 0	-5.88961337 0	2.3-12	1.3-9	2.6-9	6.6-9	00
stewart1	714 | 55; ; ; 495	35	-8.76585278 0	-8.76585278 0	1.3-10	3.4-9	-9.1-11	6.7-9	05
stewart2	1819 | 91; ; ; 910	32	-1.27531388 1	-1.27531386 1	1.0-7	8.1-10	-1.0-8	1.0-9	39
trinks	209 | 28; ; ; 141	31	-2.43523491-1	-2.43523492-1	3.3-9	1.3-11	9.4-11	9.1-10	01
visasoro	44 | 9; ; ; 8	11	6.97297058-10	-4.26654223-9	1.9-15	0.0-16	5.0-9	5.0-9	00
wood	69 | 15; ; ; 32	20	-6.64233343-2	-6.64233383-2	1.0-9	3.3-10	3.5-9	8.4-9	00
wright	20 | 6; ; ; 5	9	-2.00000000 1	-2.00000002 1	5.9-13	2.1-16	6.5-9	6.5-9	00
nql30o	3680 | ; 2700; 3602;	35	-9.46028502-1	-9.46028517-1	7.0-12	2.6-10	4.9-9	5.7-9	04
nql60o	14560 | ; 10800; 14402;	33	-9.35052951-1	-9.35052975-1	3.9-10	7.4-11	8.1-9	9.4-9	15
nql90o	32640 | ; 24300; 32402;	36	-9.31383164-1	-9.31383188-1	7.0-11	4.2-11	8.1-9	9.2-9	43
nql120o	57920 | ; 43200; 57602;	36	-9.29550235-1	-9.29550250-1	7.1-10	2.0-11	5.1-9	5.9-9	1:29
nql180o	130080 | ; 97200; 129602;	39	-9.27728621-1	-9.27728643-1	4.4-10	2.0-11	7.8-9	8.8-9	4:31
qs30o	1861 | ; 3844; 2;	15	-6.29531550 0	-6.29531557 0	1.0-9	2.4-15	5.0-9	4.3-9	01
qs60o	7321 | ; 14884; 2;	17	-6.38210348 0	-6.38210354 0	1.8-9	6.2-14	4.5-9	5.0-9	05
qs90o	16381 | ; 33124; 2;	20	-6.42377345 0	-6.42377320 0	1.2-8	6.7-15	-1.8-8	1.3-9	16
qs120o	29041 | ; 58564; 2;	21	-6.45014299 0	-6.45014275 0	9.4-9	8.9-15	-1.7-8	1.9-10	38
qs180o	65161 | ; 131044; 2;	22	-6.48350884 0	-6.48350889 0	9.3-10	1.3-14	3.5-9	6.2-9	2:04
qt30o	3924 | ; 2883; 3846;	43	-3.43399895-1	-3.43399908-1	2.2-11	9.0-11	7.8-9	8.1-9	17
qt60o	15044 | ; 11163; 14886;	52	-3.90343193-1	-3.90343204-1	4.4-11	3.8-11	6.4-9	6.1-9	2:43
qt90o	33364 | ; 24843; 33126;	54	-4.05801683-1	-4.05801698-1	9.8-11	3.1-11	8.0-9	7.6-9	10:18
qt120o	58884 | ; 43923; 58566;	53	-4.12995497-1	-4.12995510-1	1.5-10	2.0-11	7.3-9	7.1-9	27:55
qt180o	131524 | ; 98283; ; 65523	38	-4.19731365-1	-4.19960022-1	8.3-10	3.5-6	**1.2-4**	**1.1-3**	1:13:29
q30o	7482 | ; 11163; 2;	32	-9.36405103-1	-9.36405121-1	2.8-10	2.2-15	6.4-9	6.6-9	40
q60o	29362 | ; 43923; 2;	23	-8.73587717-7	-1.74652326-3	6.7-6	8.5-11	**1.7-3**	**4.2-4**	30:17
dsNRL	406 | ; 15897; ;	36	-5.57436155-5	-5.57492818-5	4.4-12	3.6-14	5.7-9	5.7-9	11:02

problem	$m \mid n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
firL1Linfa	3074 | ; 17532; ;	26	-3.06731077-3	-3.06731718-3	3.7-11	2.5-14	6.4-9	6.4-9	2:26
firL1Linfe	7088 | ; 13932; 1;	30	-2.71128527-3	-2.71129475-3	8.2-9	1.4-14	9.4-9	9.4-9	2:05
firL1	6223 | ; 17766; ;	22	-2.92573392-4	-2.92582151-4	1.5-11	3.1-14	8.8-9	8.8-9	7:47
firL2a	1002 | ; 1003; ;	7	-7.14574800-4	-7.14579339-4	0.8-15	0.8-16	4.5-9	4.5-9	22
firL2L1alp	5868 | ; 9611; 1;	18	-5.76340548-5	-5.76372249-5	1.6-12	1.3-14	3.2-9	3.2-9	1:09
firL2L1eps	4124 | ; 11969; ;	18	-8.44806167-4	-8.44815323-4	9.5-12	2.2-14	9.1-9	9.1-9	2:13
firL2Linfa	203 | ; 9029; ;	28	-7.05910969-3	-7.05911673-3	1.1-11	1.6-14	6.9-9	6.9-9	1:21
firL2Linfe	6086 | ; 14711; ;	16	-1.48919889-3	-1.48920536-3	1.2-10	1.6-14	6.5-9	6.5-9	3:51
firL2	102 | ; 103; ;	7	-3.11866437-3	-3.11866484-3	7.5-16	1.1-16	4.7-10	4.7-10	00
firLinf	402 | ; 11886; ;	24	-1.00681685-2	-1.00681770-2	1.4-9	2.5-14	8.3-9	8.2-9	5:23
BeH-2Sigma	948 | 1406; ; ;	31	1.66935641 1	1.66935639 1	3.7-12	5.9-16	7.0-9	7.0-9	2:11
BH-1Sigma+	948 | 1406; ; ;	30	2.72063377 1	2.72063375 1	2.3-11	2.4-16	3.9-9	3.9-9	5:53
BH2-2A1-ST	1743 | 2166; ; ;	30	3.04301171 1	3.04301166 1	3.0-9	3.8-16	7.5-9	7.5-9	24:18
BH+-2Sigma	948 | 1406; ; ;	30	2.69796662 1	2.69796657 1	3.9-12	2.8-16	9.6-9	9.6-9	2:09
CH+-1Sigma	948 | 1406; ; ;	29	4.06927879 1	4.06927873 1	3.6-11	6.9-16	7.4-9	7.4-9	2:06
CH2-1A1-ST	1743 | 2166; ; ;	29	4.48537628 1	4.48537626 1	1.1-9	2.6-16	2.7-9	2.8-9	23:36
CH2-3B1-ST	1743 | 2166; ; ;	30	4.50291330 1	4.50291327 1	1.1-9	1.0-15	2.8-9	2.8-9	24:15
CH-2Pi-STO	948 | 1406; ; ;	29	4.10222179 1	4.10222176 1	6.5-11	4.7-16	3.7-9	3.7-9	5:39
CH–3Sigma	948 | 1406; ; ;	30	4.09070914 1	4.09070909 1	1.9-11	4.9-16	6.5-9	6.5-9	5:52
H2O-1A1-ST	1743 | 2166; ; ;	28	8.49236908 1	8.49236900 1	9.9-12	2.9-16	4.7-9	4.7-9	22:41
H2O+-2B1-S	1743 | 2166; ; ;	30	8.42163768 1	8.42163759 1	9.9-11	6.7-16	5.5-9	5.5-9	24:16
HF-1Sigma+	948 | 1406; ; ;	27	1.04720454 2	1.04720452 2	4.2-12	4.9-16	7.6-9	7.6-9	5:15
HF+-2Pi-ST	948 | 1406; ; ;	27	1.03885668 2	1.03885666 2	2.6-11	5.2-16	8.5-9	8.5-9	5:17
LiH-1Sigma	948 | 1406; ; ;	31	8.96721196 0	8.96721180 0	3.0-11	7.5-16	8.4-9	8.4-9	2:10
NH2-2B1-ST	1743 | 2166; ; ;	30	6.29798021 1	6.29798015 1	1.5-10	0.9-15	4.6-9	4.6-9	24:17
NH+-2Pi-ST	948 | 1406; ; ;	29	5.78593622 1	5.78593619 1	3.1-11	7.3-16	3.0-9	3.0-9	5:41
NH–2Pi-ST	948 | 1406; ; ;	29	5.80546397 1	5.80546388 1	9.1-12	2.3-16	7.4-9	7.4-9	2:09
NH-3Sigma-	948 | 1406; ; ;	29	5.83910025 1	5.83910016 1	4.4-11	3.6-16	8.2-9	8.2-9	2:08
OH–1Sigma	948 | 1406; ; ;	28	7.91680600 1	7.91680587 1	2.2-12	4.3-16	8.0-9	8.0-9	5:32
OH-2Pi-STO	948 | 1406; ; ;	28	7.94670773 1	7.94670763 1	3.2-11	4.1-16	6.5-9	6.5-9	5:28
OH+-3Sigma	948 | 1406; ; ;	29	7.88863800 1	7.88863788 1	1.5-11	0.8-15	7.4-9	7.4-9	2:04

problem	m | $n_s; n_q; n_l; n_u$	it.	primal obj	dual obj	err_1	err_3	err_5	err_6	time
H3O+-1-A1-	2964 | 3008; ; 154;	30	9.01065676 1	9.01065664 1	2.2-11	1.0-15	6.9-9	6.9-9	1:36:21
NH3-1-A1-S	2964 | 3008; ; 154;	30	6.79248736 1	6.79248731 1	2.9-11	7.5-16	3.4-9	3.4-9	1:36:10
Li.2S.STO6	465 | 780; ; ; 35	36	7.40023835 0	7.40023833 0	1.2-8	2.3-10	1.4-9	5.3-9	1:08
Be.1S.STO6	465 | 780; ; ; 35	34	1.45560894 1	1.45560861 1	7.8-8	1.4-9	1.1-7	1.3-7	1:04
BeH+.1Sigm	948 | 1312; ; ; 47	45	1.64575093 1	1.64575041 1	1.4-7	1.4-9	1.5-7	1.7-7	7:44
H3.2A1.DZ.	948 | 1312; ; ; 47	40	3.36464947 0	3.36464790 0	2.6-7	1.1-9	2.0-7	2.5-7	6:53
FH2+.1A1.S	1743 | 2044; ; ; 61	42	1.09990444 2	1.09990363 2	2.1-7	2.1-9	3.6-7	5.4-7	34:00
NH2-.1A1.S	1743 | 2044; ; ; 61	46	6.27062185 1	6.27062092 1	1.5-7	9.0-10	7.3-8	9.2-8	37:18
CH3+.1E.ST	2964 | 3008; ; ; 77	40	4.87507287 1	4.87505865 1	8.0-7	6.6-9	1.4-6	2.3-6	2:08:23
NH3+.2A2.S	2964 | 3008; ; ; 77	41	6.75058463 1	6.75055336 1	3.7-7	1.1-8	2.3-6	3.2-6	2:11:36
CH3.2A2.ST	2964 | 3008; ; ; 77	40	4.92214627 1	4.92211215 1	2.4-7	1.9-8	3.4-6	4.9-6	2:08:28
CH4.1A1.ST	4743 | 4236; ; ; 95	37	5.36627612 1	5.36621129 1	1.4-6	1.8-8	6.0-6	7.6-6	6:47:01
Na.2S.STO6	4743 | 4236; ; ; 95	43	1.61077125 2	1.61076921 2	3.9-7	6.4-9	6.3-7	7.8-7	7:52:40
NH4+.1A1.S	4743 | 4236; ; ; 95	38	7.27767126 1	7.27761528 1	1.6-6	1.4-8	3.8-6	4.8-6	6:58:57