On the implementation of SDPT3 (version 3.1) – a MATLAB software package for semidefinite-quadratic-linear programming

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

January 12, 2004

Abstract

This code is designed to solve conic programming problems whose constraint cone is a product of semidefinite cones, second-order cones, nonnegative orthants and Euclidean spaces. It employs a 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 Fortran and C are incorporated via Mex interface. Routines are provided to read in problems in either SDPA or SeDuMi format. Sparsity and block diagonal structure are exploited, but the latter needs to be given explicitly. Various techniques to improve the efficiency and stablility 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 80% of a total of about 300 problems to an accuracy of at least 10^{-6} in relative duality gap and infeasibilities.

1 Introduction

Let $\mathcal{S}^n$ ($\mathcal{S}^n_+$) be the space of $n \times n$ symmetric (positive semidefinite) matrices, endowed with the standard trace inner product. For each matrix $x \in \mathcal{S}^n$, it can be identified linear isometrically as a vector in $\mathbb{R}^{n(n+1)/2}$ through the following vectorization operator

$$\mathbf{svec}(x) = [x_{11}, \sqrt{2}x_{12}, x_{22}, \ldots, \sqrt{2}x_{1n}, \sqrt{2}x_{n-1,n}, x_{nn}]^T.$$

We denote the inverse of **svec** by **smat**. We use the notation $[u; v]$ to denote the concatenation of the columns vectors u, v.

[*]Department of Mathematics, National University of Singapore, 2 Science Drive 2, Singapore 117543. (mattohkc@math.nus.edu.sg). Research supported in part by NUS ARF grant R-146-000-312.

[†]Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213, USA (reha+@andrew.cmu.edu). Research supported in part by NSF through grant CCR-9875559.

[‡]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-9805602 and ONR through grant N00014-96-1-0050.

Let $\mathcal{K} = \mathcal{S}^{n_1} \times \cdots \times \mathcal{S}^{n_p}$ and $\mathcal{K}_+ = \mathcal{S}^{n_1}_+ \times \cdots \times \mathcal{S}^{n_p}_+$. We consider a standard semidefinite program with the form:

$$\begin{array}{lll} (P) & \min & \langle c^s,\, x^s\rangle + \langle c^u,\, x^u\rangle \\ & \text{s.t.} & \mathcal{A}^s\,(x^s) + A^u\,x^u \;=\; b, \\ & & x^s \in \mathcal{K}_+,\;\; x^u \in I\!R^{n_u}. \end{array} \tag{1}$$

The notation $\langle p,\, q\rangle$ denotes the standard inner product in the appropriate space. The pair $c^s, x^s \in \mathcal{K}$ are block diagonal matrices with $c^s = (c^s_1, \ldots, c^s_p)$, $x^s = (x^s_1, \ldots, x^s_p)$. The linear map $\mathcal{A}^s : \mathcal{K} \to I\!R^m$ is given by $\mathcal{A}^s(x^s) = \sum_{j=1}^p \mathcal{A}^s_j(x^s_j)$, with

$$\mathcal{A}^s_j(x^s_j) = [\langle a^s_{j,1},\, x^s_j\rangle; \ldots; \langle a^s_{j,m},\, x^s_j\rangle],$$

where $a^s_{j,1}, \ldots, a^s_{j,m} \in \mathcal{S}^{n_j}$ are constraint matrices associated with the jth block. For computational purpose, it is convenient to identify $\mathcal{A}^s_j$ with the following $m \times n_j(n_j+1)/2$ matrix:

$$A^s_j \;=\; \left[\mathbf{svec}(a^s_{j,1}), \ldots, \mathbf{svec}(a^s_{j,m})\right]^T. \tag{2}$$

With the matrix representation of $\mathcal{A}^s_j$, we have that $\mathcal{A}^s_j(x^s_j) = A^s_j \mathbf{svec}(x^s_j)$.

The dual of the standard semidefinite program described in (1) is the following:

$$\begin{array}{lll} (D) & \max & b^T y \\ & \text{s.t.} & (\mathcal{A}^s)^T y + z^s \;=\; c^s, \\ & & (A^u)^T y \;=\; c^u, \\ & & z^s \in \mathcal{K}_+, \end{array} \tag{3}$$

where $(\mathcal{A}^s)^T : I\!R^m \to \mathcal{K}$ is the adjoint of $\mathcal{A}^s$ defined by

$$(\mathcal{A}^s)^T y \;=\; ((\mathcal{A}^s_1)^T y, \ldots, (\mathcal{A}^s_p)^T y) \;=\; \left(\sum_{k=1}^m y_k a^s_{1,k}, \ldots, \sum_{k=1}^m y_k a^s_{p,k}\right).$$

The program (D) only has semidefinite and equality constraints. But we should mention that SDPT3 can also explicitly handle second-order cone and linear constraints. That is, the dual problem may have additional constraints of the form $c^l - (A^l)^T y \geq 0$ and $c^q - (A^q)^T y \in \mathcal{Q}^{t_1} \times \ldots \times \mathcal{Q}^{t_q}$, where each $\mathcal{Q}^{t_j}$ is a second-order cone.

2 Symmetrized Newton equation

The computationally most expensive step in each iteration of our interior-point method (IPM) is the computation of the search direction $(\Delta x^s, \Delta x^u, \Delta y, \Delta z^s)$ from the *symmetrized Newton equation* with respect to an invertible block diagonal scaling matrix P (usually chosen as a function of the

current iterate x^s, z^s). For the choice of the HKM scaling matrix, the search direction is obtained from the following linear system:

$$\begin{aligned} \mathcal{A}^s(\Delta x^s) + A^u \Delta x^u &= r_p := b - \mathcal{A}^s(x^s) - A^u x^u \\ (\mathcal{A}^s)^T \Delta y + \Delta z^s &= r_d := c^s - z^s - (\mathcal{A}^s)^T y \\ (A^u)^T \Delta y &= r_u := c^u - (A^u)^T y \\ \Delta x^s + H(\Delta z^s) &= r := \sigma\mu (z^s)^{-1} - x^s, \end{aligned} \tag{4}$$

where $\mu = \langle x^s, z^s \rangle / (\sum_{j=1}^p n_j)$, and $\sigma \in (0,1)$ is a parameter. In the above, H is the linear operator on $\mathcal{K}$ defined by

$$\begin{aligned} H(\Delta z^s) &= (H_1(\Delta z_1^s), \ldots, H_p(\Delta z_p^s)) \\ H_j(\Delta z_j^s) &= \frac{1}{2}\left((z_j^s)^{-1} (\Delta z_j^s) x_j^s + x_j^s (\Delta z_j^s) (z_j^s)^{-1} \right). \end{aligned}$$

3 Computation of search direction

The standard method to compute the solution of the Newton equation (4) is to first solve the Schur complement equation (SCE):

$$\underbrace{\begin{bmatrix} M & A^u \\ (A^u)^T & 0 \end{bmatrix}}_{\mathcal{M}} \begin{bmatrix} \Delta y \\ \Delta x^u \end{bmatrix} = \begin{bmatrix} h \\ r_u \end{bmatrix}, \tag{5}$$

where $h = r_p + \mathcal{A}^s(H(r_d) - r)$. The matrix M is symmetric positive definite with $M = \mathcal{A}^s H (\mathcal{A}^s)^T = \sum_{j=1}^p \mathcal{A}_j^s H_j (\mathcal{A}_j^s)^T$. Once Δy has been computed, Δz^s and Δx^s can readily be obtained from the second and last equation of (4).

The matrix M is generally dense even if the constraint matrices A_j^s are sparse, and its computational cost is $\sum_{j=1}^p O(mn_j^3) + O(m^2 n_j^2)$ if the constraint matrices are dense. This cost may be reduced substantially if sparsity in the constraint matrices are properly exploited. In our implementation, we exploit the sparsity based on ideas proposed in [1]. On the other hand, if there are only second-order cone and linear constraints, M is typically a sparse matrix (possibly plus a low rank matrix) if the constraint matrices A^q and A^l are sparse.

The linear system (5) 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 (5) is via a preconditioned symmetric quasi-minimal residual method (PSQMR) with the preconditioner computed based on the following analytical expression of $\mathcal{M}^{-1}$:

$$\mathcal{M}^{-1} = \begin{bmatrix} M^{-1} - M^{-1} A^u S^{-1} (A^u)^T M^{-1} & M^{-1} A^u S^{-1} \\ S^{-1} (A^u)^T M^{-1} & -S^{-1} \end{bmatrix}, \tag{6}$$

where $S = (A^u)^T M^{-1} A^u$. The approach using PSQMR works reasonably well if the number of columns of A^u is small and if they are not nearly dependent. However, when those conditions are not

satisfied, the PSQMR approach is not a good method to use because (a) computing S becomes very expensive, and (b) the computed preconditioner based on (6) is no longer an accurate approximation for $\mathcal{M}^{-1}$. In this case, 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

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

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 IPM progresses, and this generally makes the component matrices, $A^u \mathrm{diag}(x_+^u./z_+^u)(A^u)^T$ and $A^u \mathrm{diag}(x_-^u./z_-^u)(A^u)^T$, in M extremely ill-conditioned. Fortunately, the following heuristic to modify the vectors x_+^u, x_-^u can typically ameliorate such an ill-conditioning problem:

$$\begin{aligned}x_+^u &:= x_+^u - 0.8\min(x_+^u, x_-^u)\\ x_-^u &:= x_-^u - 0.8\min(x_+^u, x_-^u).\end{aligned}$$

This modification does not change the original variable x^u but slow down the growth of x_+^u, x_-^u. After these modified vectors have been obtained, we also modify the vectors z_+^u, z_-^u as follows if $\mu \leq 10^{-4}$:

$$(z_+^u)_i := \frac{0.5\mu}{\max(1, (x_+^u)_i)}, \quad (z_-^u)_i := \frac{0.5\mu}{\max(1, (x_-^u)_i)}, \quad i = 1, \ldots, n_u.$$

Such a modification in z_+^u, z_-^u ensures that they approach 0 at the same rate as μ, and thus preventing the dual problem (D) from attaining the equality constraint prematurely.

4 Computation of step-length

Once a direction Δx^s is computed, a full step will not be allowed if $x^s + \Delta x^s$ violates the positive semidefinite constraint. Thus, the next iterate must take the form $x^s + \alpha \Delta x^s$ for an appropriate choice of the step-length α. 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:

$$\alpha_j^s = \begin{cases} \dfrac{-1}{\lambda_{\min}(R_j^{-T} \Delta x_j^s R_j^{-1})}, & \text{if the minimum eigenvalue } \lambda_{\min} \text{ is negative} \\ \infty & \text{otherwise,} \end{cases} \tag{7}$$

where $x_j^s = R_j^T R_j$ is the Cholesky factorization of x_j^s. 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 [3]. This approach is quite accurate in general and

represents a good trade-off between the computational cost versus quality of the resulting stepsizes. An appropriate step-length α that can be taken in order for $x^s + \alpha \Delta x^s$ to stay in $\mathcal{K}$ takes the form

$$\alpha \;=\; \min(1,\; \gamma\alpha_1^s, \ldots, \gamma\alpha_p^s), \tag{8}$$

where $\gamma \in (0,1)$ is an appropriate step-length parameter.

5 Primal-dual predictor-corrector path-following algorithm

For notational convenience, we let $c = (c^s, c^u)$, $x = (x^s, x^u)$ and $z = (z^s, z^u)$, where $z^u = 0$.

Algorithm IPC. *Suppose we are given an initial iterate (x^0, y^0, z^0) with $x^0, z^0 \in \mathcal{K} \times R^{n_u}$. 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.)

- *Set $\mu = \langle x,\, z\rangle / \sum_{j=1}^p n_j$, and*

$$\phi \;=\; \max\left(\frac{\langle x,\, z\rangle}{1 + |\langle c,\, x\rangle| + |\langle b,\, y\rangle|},\; \frac{\|r_p\|}{1+\|b\|},\; \frac{\|r_d\|}{1+\|c^s\|},\; \frac{\|r_u\|}{1+\|c^u\|}\right). \tag{9}$$

  Stop if ϕ is sufficiently small.

- *(Predictor step)*
  Solve the linear system (5), with $\sigma = 0$ for the component r in (4). Denote the solution of (4) by $(\delta x, \delta y, \delta z)$, with $\delta z^u = 0$. Let α_p be the step-length defined as in (8) with Δx^s replaced by δx^s; and let β_p be the corresponding step-length associated with z^s.

- *Take σ to be*

$$\sigma \;=\; \min\left(1,\; \frac{\langle x + \alpha_p\, \delta x,\; z + \beta_p\, \delta z\rangle}{\langle x,\, z\rangle}\right).$$

- *(Corrector step)*
  Solve the linear system (5) with r in (4) replaced by

$$r \;=\; \sigma\mu(z^s)^{-1} - x^s - \left((z^s)^{-1}\delta z^s \delta x^s + \delta x^s \delta z^s (z^s)^{-1}\right)/2.$$

  Denote the solution of (4) by $(\Delta x, \Delta y, \Delta z)$, with $\Delta z^u = 0$.

- *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 is computed as in (8) with γ chosen to be $\gamma = 0.9 + 0.09\min(\alpha_p, \beta_p)$. The step-length β associated with z^s is computed similarly.

- *Update the step-length parameter by*

$$\gamma^+ \;=\; 0.9 + 0.09\min(\alpha, \beta).$$

6 Cell array representation for problem data

Our implementation exploits the block structure of the given SDP problem. The block structure of the problem data is described by a $(p+1) \times 2$ cell array named `blk`. The constraint matrices are stored in a $(p+1) \times 1$ cell array named `At`, and the data for the objective function and the decision variables is stored in a $(p+1) \times 1$ cell array named `C`, `X`, and `Z`, respectively. For the jth block with $j = 1, \ldots, p$, the content of the elements of the cell arrays is given as follows:

`blk{j,1} = 's'`, `blk{j,2} =` n_j

`At{j} =` $(A_j^s)^T$

`C{j} =` c_j^s, `X{j} =` x_j^s, `Z{j} =` z_j^s.

The unrestricted block is coded as follows:

`blk{p+1,1} = 'u'`, `blk{p+1,2} =` n_u

`At{p+1} =` $(A^u)^T$

`C{p+1} =` c^u, `X{p+1} =` x^u, `Z{p+1} =` 0.

An LMI example. Consider the following LMI problem:

$$\begin{array}{lllll} \max & -\eta & & & \\ \text{s.t.} & GY + YG^T & \preceq & 0 & \\ & -Y & \preceq & -I & \\ & Y - \eta I & \preceq & 0 & \\ & Y_{11} & = & 1, \quad Y \in \mathcal{S}^n, & \end{array} \tag{10}$$

where $G \in I\!R^{n \times n}$ is given. This problem is a dual SDP with Y identified as a vector y in $I\!R^{n(n+1)/2}$. In this case, we have $(\mathcal{A}_1)^T y = \mathbf{svec}(G\mathbf{smat}(y) + \mathbf{smat}(y)G^T)$. 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;
At{1,1} = [ lmifun(G,I),    sparse(n2,1)];
At{2,1} = [-lmifun(I/2,I),  sparse(n2,1)];
At{3,1} = [ lmifun(I/2,I),  svec(blk(1,:),-I)];
At{4,1} = [ 1, zeros(1,n2)];
C{1,1} = sparse(n,n);  C{2,1} = -I;
C{3,1} = sparse(n,n);  C{4,1} = 1;
b = [zeros(n2,1); -1];
```

In the above, `lmifun(G,H)` is a function (available in SDPT3) that generates the matrix representation of the linear map $y \in I\!R^{n(n+1)/2} \mapsto \mathbf{svec}(G\mathbf{smat}(y)H^T + H\mathbf{smat}(y)G^T)$.

Sample run.

```
>> [obj,X,y,Z] = sqlp(blk,At,C,b);
 num. of constraints =  7
 dim. of sdp    var  =  9,   num. of sdp  blk  =  3
 dim. of free   var  =  1
******************************************************************
   SDPT3: Infeasible path-following algorithms
******************************************************************
 version  predcorr  gam  expon  scale_data
   HKM      1      0.000   1        0
it  pstep dstep p_infeas d_infeas  gap     mean(obj)    cputime
------------------------------------------------------------------
 0  0.000 0.000 2.0e+01 3.5e+00  9.6e+01 -6.363961e+00  0.0 chol  1  1
 1  0.788 0.824 4.3e+00 6.1e-01  2.1e+01 -1.002356e+01  0.0 chol  1  1
 2  0.735 0.878 1.1e+00 7.5e-02  8.0e+00 -6.284520e+00  0.1 chol  1  1
 3  0.815 0.920 2.1e-01 6.0e-03  2.0e+00 -4.837402e+00  0.1 chol  1  1
 4  0.971 0.904 6.0e-03 5.8e-04  3.3e-01 -4.559935e+00  0.2 chol  1  1
 5  0.748 0.942 1.5e-03 3.3e-05  8.0e-02 -4.551777e+00  0.2 chol  1  1
 6  0.926 0.909 1.1e-04 3.0e-06  7.3e-03 -4.565940e+00  0.3 chol  1  1
 7  1.000 0.768 3.2e-09 7.0e-07  1.5e-03 -4.565860e+00  0.3 chol  1  1
 8  0.905 0.836 3.0e-10 1.1e-07  5.1e-04 -4.565946e+00  0.3 chol  1  1
 9  0.779 0.856 5.3e-09 1.6e-08  1.9e-04 -4.565945e+00  0.4 chol  1  1
10  1.000 0.851 2.0e-08 2.5e-09  6.4e-05 -4.565966e+00  0.5 chol  1  1
11  0.852 0.792 3.5e-08 5.1e-10  2.6e-05 -4.565971e+00  0.5 chol  1  1
12  0.865 0.841 1.2e-07 8.1e-11  1.2e-05 -4.565973e+00  0.5 chol  1  1
13  1.000 0.827 7.4e-08 1.4e-11  4.7e-06 -4.565975e+00  0.6 chol  1  1
14  0.805 0.813 4.7e-07 2.6e-12  2.1e-06 -4.565976e+00  0.6
 Stop: relative gap < infeasibility.
-------------------------------------------------
 number of iterations   = 14
 primal objective value = -4.56597554e+00
 dual   objective value = -4.56597647e+00
 gap := trace(XZ)       = 2.06e-06
 relative gap           = 4.50e-07
 actual relative gap    = 1.67e-07
 rel. primal infeas     = 4.70e-07
 rel. dual   infeas     = 2.62e-12
 norm(X), norm(y), norm(Z) = 2.1e+05, 6.6e+00, 1.6e+01
 norm(A), norm(b), norm(C) = 1.4e+01, 1.0e+00, 2.0e+00
 Total CPU time (secs)  = 0.6
 CPU time per iteration = 0.0
 termination code       =  0
------------------------------------------------------------------
 Percentage of CPU time spent in various parts
------------------------------------------------------------------
 preproc  Xchol Zchol  pred   pred_steplen   corr   corr_steplen    misc
   1.6    8.2   3.3    29.5   4.9      4.9   13.1   3.3     11.5    19.7
------------------------------------------------------------------
>> Y = smat(blk(1,:),y)
```

```
Y =
    1.0000    0.0000    0.0000
    0.0000    3.2451    1.7220
    0.0000    1.7220    2.3211
```

7 Numerical results

Here we describe the results of our computational testing of SDPT3, 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 SDPs 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 [2].
7. Second-order cone 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 (2.2GHz) with 4G of memory running Linux, using MATLAB 6.5. Figure 1 shows the performance of SDPT3 (version 3.1) on a total of about 300 semdefinite-quadratic-linear programming (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 (9).

References

[1] 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.

[2] D. Henrion, private communication.

[3] 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.

[4] K.C. Toh, M.J. Todd, and R.H. Tütüncü, *SDPT3- A Matlab Software package for Semidefinite Programming*, Optimization Methods and Software, 11 (1999), pp. 545–581.

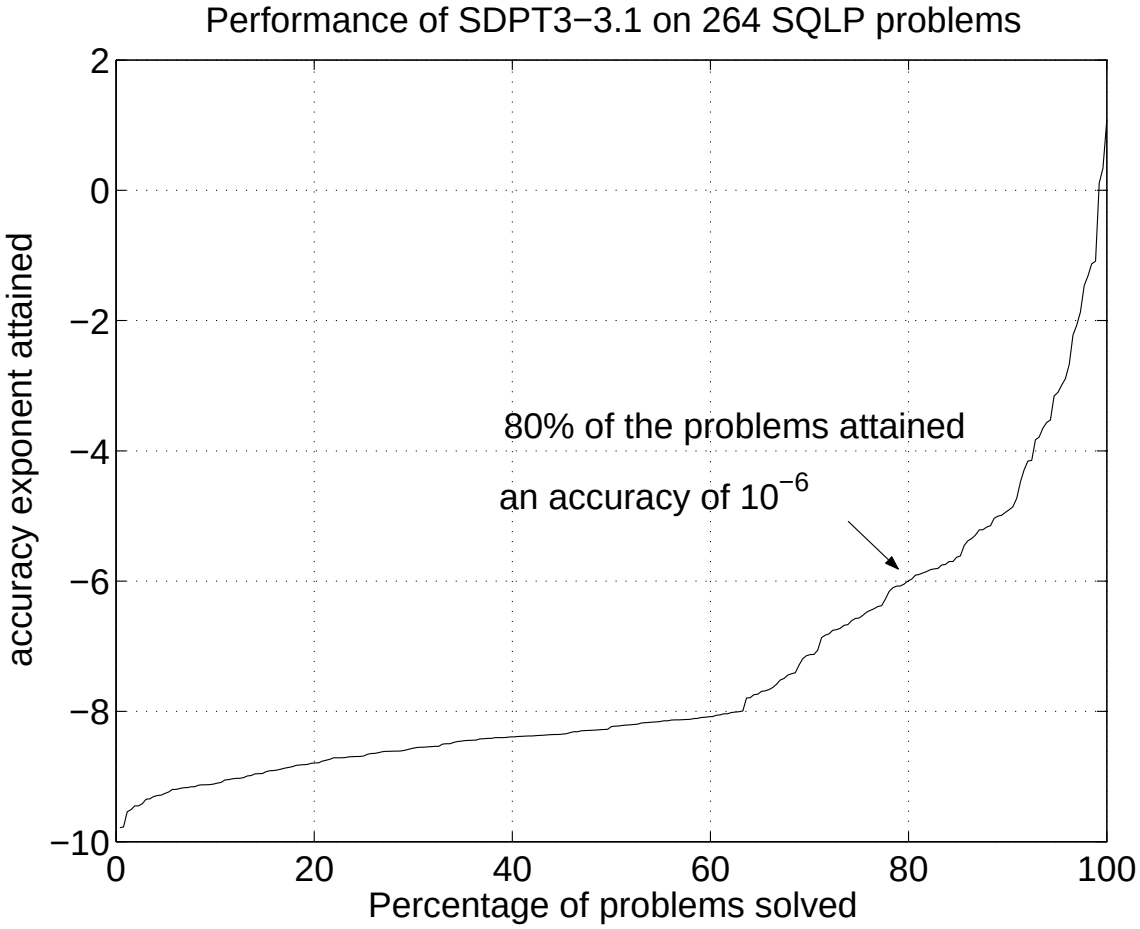

Figure 1: *The accuracy exponent is defined to be* $\log_{10}(\phi)$*, where* ϕ *is defined as in (9).*

[5] R.H. Tütüncü, K. C. Toh and M. J. Todd, *Solving Semidefinite-Quadratic-Linear Programming Using SDPT3*, Mathematical Programming Ser. B, 95 (2003), pp. 189–217.