USING SEDUMI 1.02, A MATLAB* TOOLBOX FOR OPTIMIZATION OVER SYMMETRIC CONES† (Updated for Version 1.05)

JOS F. STURM‡
Department of Econometrics, Tilburg University,
Tilburg, The Netherlands.
http://fewcal.kub.nl/~sturm, E-mail: j.f.sturm@kub.nl.

August 1998 – October 2001

Abstract

SeDuMi is an add-on for MATLAB, which lets you solve optimization problems with linear, quadratic and semidefiniteness constraints. It is possible to have complex valued data and variables in SeDuMi. Moreover, large scale optimization problems are solved efficiently, by exploiting sparsity. This paper describes how to work with this toolbox.

Keywords: Symmetric cone, semidefinite programming, second order cone programming, self–duality, MATLAB, SeDuMi

SeDuMi stands for Self-Dual-Minimization: it implements the *self-dual* embedding technique for optimization over *self-dual* homogeneous cones. The self-dual embedding technique as proposed by Ye, Todd and Mizuno [31], essentially makes it possible to solve certain optimization problems in a single phase, leading either to an optimal solution, or a certificate of infeasibility. Optimization over self–dual homogeneous cones, or more concisely, optimization over symmetric cones, was first studied by Nesterov and Todd [21], and is currently an active area of research.

Semidefinite programming is a special case of optimization over symmetric cones. The popular package SP by Vandenberghe and Boyd [29] is one of the first software tools that was developed for semidefinite programming. Some control theorists use SP indirectly via LMI-TOOL, by El Ghaoui, Nikoukhah and Delebecque [8], or MRCT, by Dussy and El Ghaoui [6],

*MATLAB is a registered trademark of The MathWorks, Inc.

†Appeared in: Optimization Methods and Software 11–12 (1999) 625–653

‡Research up to version 1.02 performed at Communications Research Laboratory, McMaster University, Hamilton, Canada. Supported by Netherlands Organization for Scientific Research (NWO).

which are user-friendly front-ends for SP. More recently, the user-friendly front-end *SeDuMi Interface*[22] is available from

`http://www.laas.fr/~peaucell/SeDuMiInt.html`

A more recent and faster solver for semidefinite programming is SDPA, by Fujisawa, Kojima and Nakata [11]. Other solvers for semidefinite programming are CSDP by Borchers [2], SDPHA by Brixius and Potra [3] and SDPT3 by Toh, Todd and Tütüncü [28]. See Mittelmann [19] for a comparison of the performance of various solvers (including `SeDuMi`) on semidefinite programming problems of the SDPLIB test set.

For optimization over symmetric cones, there were until recently only two software tools available, viz. SDPPack, by Alizadeh et al. [1], and SeDuMi. (SDPT3 has been extended to this class of problems in 2001.) Both operate under the MATLAB environment, so that they can easily be used within specific applications. SeDuMi has some features that are not available in SDPPack, namely it

- allows the use of complex valued data,
- generates Farkas-dual solutions for infeasible problems,
- takes full advantage of sparsity, leading to significant speed benefits,
- has a theoretically proven $O(\sqrt{n}\log(1/\epsilon))$ worst-case iteration bound,
- promotes sparsity by handling dense columns separately (since version 1.03), using a technique proposed by Goldfarb and Scheinberg [13],
- can import linear programs in MPS format (either using a link to LIPSOL [32] or by using the loadmps add-on by the author), and semidefinite programs in SDPA [11] format.

It is also possible to convert optimization problems from SDPPack [1] format into `SeDuMi`. Notice that earlier versions of the semidefinite programming solver SDPT3 were also able to handle complex valued data [28], but this option has been removed recently. The issue of expoiting sparsity in semidefinite programming was studied by Fujisawa, Kojima and Nakata [10]. Unlike the approach of [10], SeDuMi uses always the same sparsity expoiting procedure to form the normal equations; this procedure is efficient regardless of the degree of sparsity. See Ross [24] for a comparison between SDPpack and `SeDuMi`.

The remainder of this document is a step-by-step tutorial for `SeDuMi`. The on-line help pages serve as a reference to the toolbox. In addition, the Appendix to this document has the details of the calling sequence for the main function, `sedumi`.

1 introduction to sedumi

Throughout this document, we assume that SeDuMi is correctly installed, and that you are working under MATLAB Version 5 or later. Entering the MATLAB command '`help`' should produce a list of all installed MATLAB Toolboxes, including the following lines:

```
>> help

HELP topics:

matlab/SeDuMi105        - SeDuMi 1.05    (OCT2001)
SeDuMi105/conversion    - Conversion to SeDuMi.

For more help on directory/topic, type "help topic".
```

The command `help conversion` produces a list of functions for importing data into SeDuMi. This includes an 'umbrella' script, `getproblem`[1], which works as follows:

```
>> pname = 'truss2'; getproblem, who

Your variables are:

At          MATNAME   b           pname
K           PROBDIR   c
```

This imports problem '`truss2`', and places it in the variables `At`, `b`, `c` and `K`. To do this, SeDuMi must be able to find the requested problem somewhere on your disk. It can locate sparse SDPA problems, if you have assigned a UNIX or DOS environment variable '`SDPLIB`' to the directory path where SDPA problems are stored. (SDPLIB and `SeDuMi` have a different canonical form; if y is a dual optimal solution calculated by `SeDuMi`, then $-y$ is an optimal solution in the SDPLIB canonical form.) If LIPSOL is installed, it uses LIPSOL's function `findprob` to locate linear programming problems in MPS format. Finally, if SDPPack is installed, and the environment variable `SDPPACK` points to the SDPPack directory, then SDPPack problems are searched for in the directory 'SDPPACK/problems'.

Typing '`help SeDuMi105`' produces a list of the functions that you can use to build and solve optimization models over symmetric cones. They are: `sedumi`, `eigK`, `vec`, `mat` and `eyeK`. Online help is provided by `help sedumi`, `help eigK`, and so on. The following sections give a more detailed explanation of these functions, with some illustrating examples.

[1] This function has not been updated since Version 1.00, and may not be compatible with the latest software

2 linear programming

It is possible to formulate your linear programming model in either the primal standard form,

$$\begin{array}{cc} \text{minimize} & c^{\mathrm{T}}x \\ \text{such that} & Ax = b \\ & x_i \geq 0 \text{ for } i = 1, 2, \ldots, n, \end{array} \tag{1}$$

or the dual standard form,

$$\begin{array}{cc} \text{maximize} & b^{\mathrm{T}}y \\ \text{such that} & c_i - a_i^{\mathrm{T}}y \geq 0 \text{ for } i = 1, 2, \ldots, n. \end{array} \tag{2}$$

Mixed form and symmetric form linear programming models may be formulated using the 'free variable' definition in SeDuMi, as outlined in Section 2.1.

Suppose that you want to solve the following linear programming problem:

$$\begin{array}{cc} \text{minimize} & x_1 - x_2 \\ \text{such that} & 10x_1 - 7x_2 \geq 5 \\ & x_1 + x_2/2 \leq 3 \\ & x_1 \geq 0,\ x_2 \geq 0. \end{array} \tag{3}$$

In order to formulate this LP problem in the primal standard form, we have to add slack variables, say x_3 and x_4. In MATLAB, we can then enter the b and c vectors, and the A matrix as follows:

```
>>  c = [1; -1; 0; 0];
>>  A = [10, -7, -1, 0; 1, 1/2, 0, 1];
>>  b = [5; 3];
```

We can now solve problem (3) in the primal form (1) by invoking the function `sedumi`. Remark that MATLAB is case sensitive, and it is therefore essential to write `sedumi` in lower case.

```
>> sedumi(A,b,c)
SeDuMi by Jos F. Sturm, 1998.
Alg = 2: xz-corrector, theta = 0.250, beta = 0.500
eqs m = 2, order n = 5, dim = 5, blocks = 1
 it :     cx        gap   delta  rate  t/maxt   feas
  0 :           5.00E+00 0.000
  1 :  7.81E-01 9.79E-01 0.000 0.1959 0.9000   0.77
  2 : -5.52E-02 9.40E-02 0.000 0.0959 0.9900   0.93
```

```
* 3 : -1.25E-01 5.70E-04 0.000 0.0061 0.9990   1.08
iter seconds digits       c*x               b*y
  3      0.1  15.1 -1.2500000000e-01 -1.2500000000e-01
|Ax-b| =   1.8e-15, |x|=  2.9e+00, |y|=  2.8e-01


ans =

    1.9583
    2.0833
         0
         0
```

This shows that the optimal value is -0.125, as listed under `c*x`. The function `sedumi` returns an optimal solution, which in this case is $x_1 = 1.9583$ and $x_2 = 2.0833$. Notice that x is indeed feasible for (1), because all its components are nonnegative, and $Ax = b$, as can be checked by the commands `min(x)` and `norm(Ax-b)`, respectively. Of course, some round-off errors may occur, as can be seen from the following MATLAB output:

```
>> A=sparse(A);norm(A*x-b)

ans =

   1.7764e-15

>> norm(A*(24*x)-24*b)

ans =

     0
```

The first quantity is listed as $|\mathtt{Ax} - \mathtt{b}| = \mathtt{1.8e} - \mathtt{15}$ in the output of `SeDuMi`. The second line shows that the reported value of `|Ax-b|` does not only contain the residual, but also errors in computing the residual. The meaning of the other parts in the output of `SeDuMi` is explained in Section 5.

Using dual solutions, it is possible to check also optimality. Namely, if we let $z := c - A^{\mathrm{T}}y$, then if x and y are feasible to (1) and (2) respectively, we have

$$0 \leq z^{\mathrm{T}}x = c^{\mathrm{T}}x - y^{\mathrm{T}}Ax = c^{\mathrm{T}}x - b^{\mathrm{T}}y.$$

Thus, if $c^{\mathrm{T}}x - b^{\mathrm{T}}y = 0$, then $c^{\mathrm{T}}x$ must be minimal, and $b^{\mathrm{T}}y$ must be maximal, over all feasible solutions. The dual solution y to (2) is assigned to the second output argument of `sedumi`, as in

```
>> [x,y] = sedumi(A,b,c)
```

In this example, we have $y_1 = 0.125$ and $y_2 = 0.25$. Issueing the command

```
>> z = c - A'*y
z =

         0
         0
    0.1250
    0.2500
```

we see that $z_i x_i = 0$ for all i, proving optimality. However, due to some round-off errors, $c^{\mathrm{T}}x - b^{\mathrm{T}}y$ is positive in this case. The quantity `digits` = 15.1 in the output of `SeDuMi`, is defined as follows:

$$\texttt{digits} = \begin{cases} -\log_{10}((c^{\mathrm{T}}x - b^{\mathrm{T}}y)/(|b^{\mathrm{T}}y| + 10^{-10})) \text{ if } c^{\mathrm{T}}x - b^{\mathrm{T}}y > 0 \\ \infty \text{ otherwise.} \end{cases} \tag{4}$$

As is well known, y is a subgradient of the optimal value function in terms of changes in b. If the optimal value function is locally not differentiable in b, i.e. if there are multiple dual optimal solutions, then it is said to be primal degenerate. `SeDuMi` usually generates a solution y in the relative interior of the subgradient set, because it uses a Mehrotra-Ye [18] type termination procedure for linear programs. For a detailed treatment of sensitivity analysis based on such solutions, we refer to Monteiro and Mehrotra [20] and the book of Roos, Terlaky and Vial [23].

For large problems, it is usually not feasible to store A as a full matrix, due to memory limitations. In this case, A should be stored in sparse format; type `help sparfun` for details. Internally, `SeDuMi` always converts A to sparse format. The b and c vectors can also be in sparse format, if desired.

In the preceding, we defined b and c in MATLAB as column vectors, but this is not essential; `SeDuMi` produces the same output if b and/or c are defined as row vectors. Similarly, `SeDuMi` is not picky about the orientation of A: it will detect the correct orientation based on the b and c vectors (except in the unrealistic case that A is square). In fact, it is good practice to store A in such a way that it has more rows than columns, which is the transpose orientation of the A matrices that we have seen so far. Namely, if A is stored in sparse format, then it is stored as a set of sparse column vectors. Hence, if there are fewer columns, it will occupy less space.

There is a third output argument of `SeDuMi`, called `info`. In our example,

```
>> [x,y,info]=sedumi(A,b,c); info

info =
```

```
    cpusec: 0.1100
      iter: 3
 feasratio: 1
    numerr: 0
      pinf: 0
      dinf: 0
```

This is a compound output argument, or structure, with a field `cpusec` for the solution time, `iter` for the number of iterations, a field `numerr` which is nonzero in case of numerical problems (1 means premature termination: results are inaccurate, 2 means failure), a field `feasratio` for the final value of the feasibility indicator, and two fields, `pinf` and `dinf`, for the detected feasibility status of the optimization problem. If `pinf` $= 1$, then the primal problem (1) is infeasible, and y is a Farkas dual solution.

For instance, if we change the b vector in the preceding example to $b = \begin{bmatrix} 5, & 0.4 \end{bmatrix}$, then `SeDuMi` yields `info.pinf` $= 1$, $b^{\mathrm{T}}y = 0.0955 > 0$, $\max_{i=1,2,3,4} a_i^{\mathrm{T}}y = -0.1866 \leq 0$. Notice that for any x with $Ax = b$, we have $y^{\mathrm{T}}Ax = b^{\mathrm{T}}y = 0.0955 > 0$, whereas $y^{\mathrm{T}}Ax \leq 0$ for nonnegative x, because all components of $A^{\mathrm{T}}y$ are nonpositive. A Farkas dual solution thus provides a certificate of infeasibility. In this example, there appears to be a Farkas dual solution for which all entries of $A^{\mathrm{T}}y$ are strictly negative. In general though, they are merely nonpositive. For numerical reasons, $A^{\mathrm{T}}y$ can then contain some small positive components, and in this case we have an approximate Farkas dual solution. Loosely speaking, such solutions demonstrate that there cannot be any reasonably sized primal feasible solution; see Todd and Ye [27] for details.

Suppose now that we want to solve a problem in the dual standard form (2). In this case, y with $b^{\mathrm{T}}y > 0$ and $A^{\mathrm{T}}y \leq 0$ has the interpretation of an improving direction. Namely, if there exists a feasible solution $\bar{y}$, i.e. if $c - A^{\mathrm{T}}y \geq 0$, then $\bar{y} + ty$ is feasible for all $t \geq 0$, and $\lim_{t\to\infty} b^{\mathrm{T}}y = \infty$. In this case, we say that the problem is unbounded. The other possibility is that there does not exist any feasible solution $\bar{y}$, i.e. the problem is infeasible. To distinguish between an infeasible and an unbounded problem, we have to go through a second stage, by solving a feasibility problem:

```
>> [x,y,info] = sedumi(A,zeros(length(b),1),c)
```

which may be entered in simplified form (since version 1.05) as

```
>> [x,y,info] = sedumi(A,0,c)
```

In our previous example, the dual problem is feasible, and the above command yields a feasible solution $\bar{y}$. The need for this second stage feasibility problem is typical for interior point methods with the self–dual embedding technique of Ye, Todd and Mizuno [31].

The interpretation of `info.dinf` is analogous to that of `info.pinf`. Namely, if `info.dinf` $= 1$ then the dual problem (2) is infeasible, and this claim is certified by a Farkas solution x with

$$c^{\mathrm{T}}x < 0,\ Ax = 0,\ x_i \geq 0 \text{ for } i = 1, 2, \ldots, n.$$

To distinguish between primal unboundedness and primal infeasibility, we would then solve the feasibility problem

```
>> [x,y,info] = sedumi(A,b,zeros(length(c),1))
```

`SeDuMi` can also generate Gordan-Stiemke dual solutions. For instance, if we restore the vector b to $b^{\mathrm{T}} = \left[\begin{array}{cc} 5, & 3 \end{array}\right]$, and solve the feasibility problem `x = sedumi(A,b,zeros(4,1))`, we obtain a strictly positive vector x. This is because interior point methods try to find a solution in the relative interior of the solution set. To see what happens if feasible solutions can merely be nonnegative, consider the following example:

```
>> b = [5, 1/2];
>> [x,y,info]=sedumi(A,b,zeros(4,1));
>> [x -A'*y]

ans =

    0.5000         0
         0    1.1875
         0    0.0990
         0    0.9895

>> b*y

ans =

   1.3878e-17
```

In this example, the primal does not have an interior solution, i.e. it is weakly feasible, and this is demonstrated by a Gordan-Stiemke dual solution y. Namely, $0 \neq A^{\mathrm{T}}y \leq 0$, and $b^{\mathrm{T}}y = 0$, which clearly implies that there cannot be any $x > 0$ such that $A * x = b$.

`SeDuMi` treats the primal and dual in a symmetric way, i.e. it does not favor one over the other. From a modeling point of view however, the primal standard form and the dual standard form are quite different, and it depends on the application which one is more favorable. The primal form has the advantage of explicit equality constraints. In principle, equality constraints can be constructed in the dual form also, simply by means of two inequality constraints, such as

$a_i^{\mathrm{T}} y \le c_i$ and $a_i^{\mathrm{T}} y \ge c_i$. However, this technique is not recommended, since such constraint pairs tend to get a pair of very large primal multipliers x_i, hence leading to numerical difficulties. It may be better to enforce an equality constraint by eliminating a y variable. However, the latter technique may destroy the sparsity structure of the A-matrix, thus leading to longer solution times.

Exactly the same problems arise in modeling a free (i.e. unresistricted in sign) variable in the primal standard form. Splitting such a variable into two, its positive part and its negative part, often results in numerical difficulties. One may also try to eliminate such a variable by removing an equality constraint, but this usually causes an increase in the number of nonzeros in the A-matrix. An alternative is to model all free variables in a quadratic cone. Quadratic cones are discussed in Section 3. To prevent numerical difficulties with this technique, it is desirable to fix a – possibly large – upper bound on the norm of the vector of free variables, which is easily done in a quadratic cone.

Since Version 1.05, the user does not need to worry about these issues, since free variables are allowed. This is the topic of the section below.

2.1 Free variables

It is possible to formulate your linear programming model in a primal form with free variables as follows:

$$\begin{array}{lc} \text{minimize} & c^{\mathrm{T}} x \\ \text{such that} & Ax = b \\ & x_i \in \Re \text{ for } i = 1, 2, \ldots, \mathtt{K.f}, \\ & x_j \ge 0 \text{ for } j = \mathtt{K.f} + 1, \mathtt{K.f} + 2, \ldots, n, \end{array} \tag{5}$$

where `K.f` is the number of *free variables*. The associated slack variables in the dual problem are then restricted to be zero, thus allowing *equality constraints* in the dual:

$$\begin{array}{lc} \text{maximize} & b^{\mathrm{T}} y \\ \text{such that} & c_i - a_i^{\mathrm{T}} y = 0 \text{ for } i = 1, 2, \ldots, \mathtt{K.f} \\ & c_j - a_j^{\mathrm{T}} y \ge 0 \text{ for } j = \mathtt{K.f} + 1, \mathtt{K.f} + 2, \ldots, n \end{array} \tag{6}$$

3 quadratic and semidefinite constraints

In `SeDuMi`, it is possible to impose quadratic or semidefinite constraints, by restricting variables to a quadratic cone or the cone of positive semidefinite matrices, respectively. Such a restriction then replaces the nonnegativity restriction in linear programming. Thus, instead of requiring $x \in \Re^n_+$ as in (1), we will now require $x \in \mathcal{K}$, where $\mathcal{K}$ is a so-called symmetric cone. A symmetric cone is a Cartesian product of a nonnegative orthant, quadratic cones and cones of positive semidefinite matrices. The standard primal form for such optimization problems is

$$\begin{array}{ll} \text{minimize} & c^{\mathrm{T}} x \\ \text{such that} & Ax = b \\ & x \in \mathcal{K} \end{array} \tag{7}$$

and the dual standard form is

$$\begin{array}{ll} \text{maximize} & b^{\mathrm{T}} y \\ \text{such that} & c - A^{\mathrm{T}} y \in \mathcal{K}. \end{array} \tag{8}$$

3.1 The quadratic cone

A quadratic cone is by definition a cone of the form

$$\text{Qcone} := \{(x_1, x_2) \in \Re \times \Re^{N-1} \mid x_1 \geq \|x_2\|\}, \tag{9}$$

where $\|\cdot\|$ denotes the Euclidean norm (the function `norm` in MATLAB). The quadratic cone is also known as the second order cone or Lorentz cone. As an example, consider the following optimization problem:

$$\min \left\{ y_1 + y_2 \,\middle|\, y_1 \geq \|q - P y_3\|,\ y_2 \geq \sqrt{1 + \|y_3\|^2} \right\}, \tag{10}$$

where P is a given matrix, and q a given vector. The above is a robust least squares problem, see El Ghaoui and Lebret [7]. The decision variables are the scalars y_1 and y_2, and the vector y_3. This problem has two quadratic constraints, viz.

$$(y_1, q - P y_3) \in \text{Qcone}, \quad \left(y_2, \begin{bmatrix} 1 \\ y_3 \end{bmatrix} \right) \in \text{Qcone}. \tag{11}$$

Given P and q, the following MATLAB function (`rls.m`) constructs problem (10) in the standard dual form (8). The A matrix will be in transposed orientation, and is hence denoted as `At`.

```
% [At,b,c,K] = rls(P,q)
% Creates dual standard form for robust least squares problem "Pu=q".
function [At,b,c,K] = rls(P,q)

[m, n] = size(P);
% ---------- minimize y_1 + y_2 ----------
b = -sparse([1; 1; zeros(n,1)]);
% ---------- (y_1, q - P y_3) in Qcone ----------
```

```
At = sparse([-1, zeros(1,1+n); ...
             zeros(m,2), P]);
c = [0;q];
K.q = [1+m];
% ---------- (y_2, (1,y_3)) in Qcone ----------
At = [At; 0, -1, zeros(1,n); ...
          zeros(1,2+n); ...
          zeros(n,2), -eye(n)];
c = sparse([c; 0;1;zeros(n,1)]);
K.q = [K.q, 2+n];
```

Notice first that the above function uses sparse data types, in order to save memory. Furthermore, a structure `K` is defined, with a field `K.q` that lists the dimensions of the quadratic cones. (The 'q' in `K.q` stands for 'quadratic'.) The `K`–structure will be used to tell **SeDuMi** that the components of $c - A^{\mathrm{T}}y$ are *not* restricted to be nonnegative as they would be in linear programming. Instead, the first `K.q(1)` entries are restricted to a quadratic cone, and the last `K.q(2)` entries are restricted to another quadratic cone. This is the way in which we model the symmetric cone $\mathcal{K}$ in (7) and (8), and hence construct the two quadratic constraints in (11).

As a numerical example, we solve a 4×3 robust least squares problem with dependent columns in P. The example is from [7].

```
>> P = [3 1 4;0 1 1;-2 5 3; 1 4 5]; q = [0;2;1;3];
>> [At,b,c,K] = rls(P,q);
>> [x,y,info] = sedumi(At,b,c,K);
SeDuMi by Jos F. Sturm, 1998.
Alg = 1: v-corrector, theta = 0.250, beta = 0.500
eqs m = 5, order n = 5, dim = 11, blocks = 3
 it :     cx       gap   delta  rate   t/maxt   feas
  0 :           5.00E+00 0.000
  1 : -1.23E+01 1.30E+00 0.000 0.2605 0.9000  -0.18
  2 : -5.94E+00 3.34E-01 0.000 0.2568 0.9000   0.54
  3 : -3.60E+00 6.14E-02 0.116 0.1838 0.9000   0.86
  4 : -3.34E+00 1.80E-03 0.000 0.0293 0.9900   1.10
* 5 : -3.33E+00 4.00E-06 0.000 0.0022 0.9990   1.00
* 6 : -3.33E+00 9.59E-09 0.000 0.0024 0.9990   1.00
* 7 : -3.33E+00 6.06E-10 0.153 0.0632 0.9900   1.00
* 8 : -3.33E+00 1.24E-10 0.000 0.2037 0.9000   1.00
iter seconds digits       c*x               b*y
  8      0.1  11.3 -3.3329085968e+00 -3.3329085968e+00
|Ax-b| =   2.8e-16, |x|=  2.0e+00, |y|=  2.5e+00
```

In the above call to SeDuMi, we see a new input argument, viz. K. This argument makes SeDuMi solve an optimization problem in the form (7)–(8), where the symmetric cone $\mathcal{K}$ is described by the structure $\mathcal{K}$. Without the fourth input argument (K), SeDuMi would solve a linear programming problem of the form (1)–(2).

To check that (11) is indeed satisfied by the solution y, it is in principle possible to verify the inequality in definition (9) directly. However, it is more convenient to use the function eigK, which is part of SeDuMi. This function returns the eigenvalues (or spectral values) of a vector with respect to a symmetric cone. A symmetric cone consists of those vectors which have nonnegative eigenvalues, see e.g. the book by Faraut and Korányi [9]. For a quadratic cone (9), there are merely two eigenvalues, viz. given a vector $(x_1, x_2) \in \Re \times \Re^{N-1}$, we have $\lambda_1(x_1, x_2) = (x_1 - \|x_2\|)/\sqrt{2}$ and $\lambda_2(x_1, x_2) = (x_1 + \|x_2\|)/\sqrt{2}$.

We can thus check feasibility and optimality as follows:

```
>> [eigK(x,K), eigK(c-At*y,K)]

ans =

    0.0000   -0.0000
    1.4142    3.2307
    0.0000   -0.0000
    1.4142    1.4827

>> x'*(c-At*y)

ans =

   1.5807e-11
```

For symmetric cones $\mathcal{K}$, it holds that $x^{\mathrm{T}}z \geq 0$ for all $x \in \mathcal{K}$ and $z \in \mathcal{K}$. Therefore, x provides an optimality certificate for y just as in the case of linear programming. The interpretation of Farkas dual solutions extends in the same way. See the survey paper of Luo, Sturm and Zhang [16] for the details. However, a paradoxal phenomenon can occur, viz. that x and y are almost feasible, whereas $c^{\mathrm{T}}x - b^{\mathrm{T}}y$ is considerably negative ($\|x\|$ and/or $\|y\|$ must then obviously be very large). SeDuMi will then report an infinite number of digits in accuracy, according to formula (4). This phenomenon was explained by Luo, Sturm and Zhang [15] and Sturm [25].

It is possible that an optimization model has both nonnegativity and quadratic cone constraints. For instance, we may extend the above example with the restriction that $y_3[1] \leq -0.1$, where $y_3[1]$ denotes the first component in the vector y_3. This restriction can be added to the model as follows:

```
>> a1 = zeros(1,length(y)); a1(3) = 1;
```

```
>> c = [-0.1; c];   At = [a1;At];
>> K.l = 1;
>> [x,y,info] = sedumi(At,b,c,K);
>> eigK(c-At*y,K)'

ans =

    0.0000    -0.0000     3.2307    -0.0000     1.4904
```

The field `K.l` is the number of nonnegative variables, which in this case is one. (The 'l' in `K.l` stands for 'linear'.) By convention, the nonnegative variables are always the first components, so that $\mathcal{K} = \Re_+ \times \text{Qcone} \times \text{Qcone}$ in our case. As can be seen from the output of `eigK`, there are 5 eigenvalues for this cone: 1 for each nonnegativity constraint, and 2 for each quadratic constraint. We say that $\mathcal{K}$ is a symmetric cone of order 5. (`SeDuMi` reports 'order $n = 6$', because of its internal self–dual reformulation.)

`SeDuMi` supports an alternative form of the quadratic cone, viz.

$$\text{Rcone} := \left\{ (x_1, x_2, x_3) \in \Re \times \Re \times \Re^{N-2} \,\middle|\, x_1 x_2 \geq \frac{1}{2}\|x_3\|^2,\ x_1 + x_2 \geq 0 \right\}. \tag{12}$$

Geometrically, Rcone is simply a rotation of Qcone. The specific form of Rcone is convenient for modeling convex quadratic functions. Namely, by adding the linear equality constraint '$x_1 = 1$' to the model, we obtain the restriction

$$x_2 \geq \frac{1}{2}\|x_3\|^2.$$

Throughout the model, we can then use x_2 as a tight upper bound on $\|x_3\|^2/2$. Fractions are also conveniently modeled by Rcone constraints. For instance, we may minimize $1/x_1$ for $x_1 > 0$ by solving the model

$$\min\{x_2 \mid x_1 x_2 \geq 1,\ x_1 + x_2 \geq 0\}.$$

Notice that this problem does not have a solution: the infimum of $1/x_1$ is zero, for $x_1 \to \infty$.

```
>> clear K;
>> c = [0, 1, 0]; b = sqrt(2); A = [0, 0, 1]; K.r = 3;
>> [x,y,info] = sedumi(A,b,c,K);
>> x(2), x(1)*x(2)

ans =

   1.5360e-05
```

```
ans =

    1.0147
```

You may find that x_2 is not yet close enough to zero, and that x_1 is not equal to ∞ either. However, the primal solution is feasible, the dual solution is almost feasible, and the duality gap is even negative. This illustrates an error bound difficulty, which is usual for this type of irregular problems. In Section 5, we will see how to obtain a more accurate solution, by setting an optional parameter, `pars.eps`.

As illustrated by the above example, the field `K.r` serves to list the dimensions of Rcone constraints, analogously to the definition of Qcone constraints by `K.q`. (The 'r' in `K.r` stands for 'rotated quadratic cone'.) Setting both `K.l`, `K.q` and `K.r` fields yields a symmetric cone of the form

$$\mathcal{K} = \Re_+^{\mathtt{K.l}} \times (\text{Qcone} \times \cdots \times \text{Qcone}) \times (\text{Rcone} \times \cdots \times \text{Rcone}).$$

For instance, we can add a bound '$x_1 \leq 10^7$' to the model as follows:

```
>> c = [0, 0, 1, 0]; b = [sqrt(2); 1E7]; A = [0, 0, 0, 1;1, 1, 0, 0];
>> K.l = 1; K.r = 3;
>> [x,y,info] = sedumi(A,b,c,K);
```

Some applications of Qcone and Rcone constraints are discussed in Lobo et al. [14].

3.2 The positive semidefinite cone

Semidefiniteness constraints are an important class of restrictions that can be modeled with `SeDuMi`. As an example, consider the following problem:

$$\min \left\{ \sum_{i=1}^{m} (m-i) x_{ii} \;\middle|\; \sum_{i=1}^{m-k} x_{i,i+k} = b_k \text{ for } k = 0, \ldots, m-1,\ X \text{ is psd} \right\}. \tag{13}$$

Here, X is an $m \times m$ symmetric matrix, and x_{ij} denotes the entry on row i and column j. The length m vector b is given. The abbreviation 'psd' stands for 'positive semidefinite'. The above optimization problem yields a minimal phase spectral factorization of an autocorrelation vector b, see Davidson, Luo and Sturm [4]. Problem (13) is stated in terms of an $m \times m$ symmetric matrix of decision variables, whereas `SeDuMi` works with a vector of decision variables, as in (7)–(8). This small issue is resolved by using the well known technique of vectorization. Vectorization is implemented by the functions `vec` and `mat`, which are part of `SeDuMi`. The function `vec(X)` creates a long vector, by stacking the columns of the matrix X, as in:

```
>> x = vec([1, 5, -3;  5, 2, -9;  -3, -9, 4])'

x =

     1     5    -3     5     2    -9    -3    -9     4
```

The inverse of `vec` is `mat`. Thus, if x is a vector of length n^2, then $\texttt{mat}(x)$ constructs an $n \times n$ matrix, and fills it with the entries of the vector x, starting at the first column.

```
>> mat(x)

ans =

     1     5    -3
     5     2    -9
    -3    -9     4
```

The following MATLAB function produces a standard primal form for problem (13).

```
% [At,b,c,K] = specfac(b)
% Creates primal standard form for minimal phase spectral factorization.
function [At,b,c,K] = specfac(b)

m = length(b);
% ---------- minimize sum (m-i)*X(i,i) ----------
c = vec(spdiags((m-1:-1:0)',0,m,m));
% ----- Let e be all-1, and allocate space for the A-matrix -----
e = ones(m,1);
At = sparse([],[],[],m^2,m,m*(m+1)/2);
% ---------- sum(diag(X,k)) = b(k) ----------
for k = 1:m
  At(:,k) = vec(spdiags(e,k-1,m,m));
end
K.s = [m];
```

The field $\texttt{K.s} = [m]$ tells `SeDuMi` that we want the $m \times m$ matrix `mat(x)` to be symmetric positive semidefinite. (The 's' in `K.s` stands for 'semidefinite'.) We can now solve problem (13) as follows:

```
>> b = [2; 0.2; -0.3];
>> [At,b,c,K] = specfac(b);
>> [x,y,info] = sedumi(At,b,c,K);
```

```
SeDuMi by Jos F. Sturm, 1998.
Alg = 1: v-corrector, theta = 0.250, beta = 0.500
eqs m = 3, order n = 4, dim = 10, blocks = 2
 it :     cx       gap    delta  rate  t/maxt   feas
  0 :           4.00E+00 0.000
  1 :  8.14E+00 1.40E+00 0.000 0.3497 0.9000   0.32
  2 :  2.29E+00 4.68E-01 0.000 0.3346 0.9000   0.59
  3 :  3.42E-01 1.12E-01 0.337 0.2391 0.9000   0.84
  4 :  1.26E-01 1.92E-03 0.000 0.0172 0.9900   1.24
* 5 :  1.23E-01 3.97E-06 0.000 0.0021 0.9990   1.00
* 6 :  1.23E-01 8.88E-10 0.000 0.0002 0.9999   1.00
* 7 :  1.23E-01 2.27E-12 0.000 0.0026 0.9990   1.00
iter seconds digits       c*x               b*y
  7      0.1  10.7  1.2273256502e-01  1.2273256502e-01
|Ax-b| =   0.0e+00, |x|=  2.0e+00, |y|=  7.6e-01
```

To check positive semidefiniteness, we can either use the function `eig` that is part of MATLAB, or the function `eigK`, which comes with `SeDuMi`.

```
>> [eig(mat(x)), eigK(x,K)]

ans =

    0.0000    0.0000
    0.0000    0.0000
    2.0000    2.0000
```

The use of `eigK` is more convenient, especially if there are multiple semidefiniteness constraints, or if there are also nonnegativity or quadratic cone constraints. `SeDuMi` will always produce symmetric matrix variables, i.e. `mat(x)` is symmetric. Do not add symmetry constraints explicitly, as in '$x_{ij} - x_{ji} = 0$'. At best, such constraints will be removed by `SeDuMi` from the A matrix.

However, the dual solution $c - A^{\mathrm{T}}y$ need not be symmetric, as can be seen in the numerical example that we are dealing with:

```
>> mat(c-At*y)

ans =

    2.0727   -0.3130    0.6849
         0    1.0727   -0.3130
         0         0    0.0727
```

In this case, the dual solution is upper triangular, because `mat(c)` is diagonal, and `mat(At(:,k))` is upper triangular for all $k = 1, 2, \ldots, m$. Letting $Z = \mathtt{mat}(c - A^{\mathrm{T}}y)$, SeDuMi restricts the symmetric part of Z, which is $(Z + Z^{\mathrm{T}})/2$, to be positive semidefinite. The function `eigK` yields the eigenvalues of the symmetric part. Thus,

```
>> eigK(c-At*y,K)

ans =

    1.0583
    2.1597
   -0.0000
```

produces the same result as

```
>> Z = mat(c-At*y); eig(Z+Z')/2
```

Notice that problem (13) is equivalent to

$$\min \left\{ \sum_{i=1}^{m} (m-i) x_{ii} \,\middle|\, \sum_{i=1}^{m-k} \frac{x_{i,i+k} + x_{i+k,i}}{2} = b_k \text{ for } k = 0, \ldots, m-1,\ X \text{ is psd} \right\}. \tag{14}$$

Namely, $x_{i,i+k} = (x_{i,i+k} + x_{i+k,i})/2$, because X is symmetric. Thus, we may change the A matrix as follows:

```
>> for k=1:size(At,2), Ak = mat(At(:,k)); At(:,k) = vec(Ak+Ak')/2; end
```

The solutions x and y, as produced by SeDuMi, will be exactly the same. However, since the constraints in the A matrix have been symmetrized, we find that $\mathtt{mat}(\mathtt{c} - \mathtt{At} * \mathtt{y})$ is now symmetric; it is the matrix $(Z + Z')/2$.

For SeDuMi, it does not make any difference whether the constraints in A and the objective c are symmetrized or not. However, when modeling in the primal standard form, you will probably find it more natural to work with upper or lower triangular matrices in A and c; your model will also use less memory like this. On the other hand, symmetric matrices are more natural when modeling in the dual form.

There can be multiple positive semidefiniteness constraints, in which case `K.s` lists the orders of the respective matrices. This is analogous to the definition of multiple quadratic constraints in `K.q` and/or `K.r`. The positive semidefinite variables are always the last components of x and $c - A^{\mathrm{T}}y$, i.e.

$$\mathcal{K} = \Re_{+}^{\mathtt{K.l}} \times (\text{Qcone} \times \cdots \times \text{Qcone}) \times (\text{Rcone} \times \cdots \times \text{Rcone}) \times (\text{Scone} \times \cdots \times \text{Scone}),$$

where Scone denotes the cone of positive semidefinite matrices. It is easy to remember the above arrangement, by noting the alphabetical order of 'l', 'q', 'r' and 's'.

4 complex values

In some application areas, such as signal processing, optimization problems may involve complex valued data. An example is the Toeplitz Hermitian covariance estimation problem, which is discussed in Wu, Luo and Wong [30]. Other structured covariance estimation problems, such as discussed in Deng and Hu [5], can be treated similarly. Given a Hermitian matrix P, the goal is to find a Hermitian positive definite matrix Z with a Toeplitz structure, such that $\|P - Z\|_F$ is minimal. Thus, the optimization problem is:

$$\begin{array}{rl} \text{minimize} & \sum_{i=1}^{m} \left((z_{ii} - p_{ii})^2 + 2 \sum_{j=i+1}^{m} |z_{ij} - p_{ij}|^2 \right) \\ \text{such that} & Z \text{ is Toeplitz, i.e. } z_{i,j} = z_{i+1,j+1} \text{ for all } i,j = 1,2,\ldots,m-1 \\ & Z \text{ is psd.} \end{array} \tag{15}$$

If the matrix P has complex entries, then we will usually also see complex entries in the optimal solution Z. Notice that the Toeplitz property is better modeled in the dual form, than in the primal form. In fact, `mat(At*y)` in (13) is an upper triangular real Toeplitz matrix, and in (14), it is a symmetric Toeplitz matrix. The MATLAB formulation of (15) therefore resembles the MATLAB formulation of (13).

```
% [At,b,c,K] = toepest(P)
% Creates dual standard form for Toeplitz-covariance estimation
function [At,b,c,K] = toepest(P)

m = size(P,1);
% ---------- maximize y(m+1) ----------
b = [sparse(m,1); 1];
% ----- Let e be all-1, and allocate space for the A-matrix -----
e = ones(m,1);
K.q = [1 + m*(m+1)/2];
K.xcomplex = 2:K.q(1);      %Norm-bound entries are complex valued
At = sparse([],[],[],K.q(1) + m^2,m+1,1 + 2*m^2);
% ---------- constraints ----------
% -y(m+1) >= norm( vec(P) - sum(y_i * Ti) )      (Qcone)
% sum(y_i * Ti) is psd                            (Scone)
% ---------------------------------
At(:,1) = [sparse(2:(m+1),1,1,K.q(1),1); -vec(speye(m))];
c = [0; diag(P)];
firstk = m+2;
for k = 1:(m-1)
  lastk = firstk + m-k-1;
  Ti = spdiags(e,k,m,m);
```

```
  At(:,k+1) = [sqrt(2) * sparse(firstk:lastk,1,1,K.q(1),1); -2*vec(Ti)];
  c = [c; sqrt(2) * diag(P,k)];
  firstk = lastk + 1;
end
At(:,m+1) = [1; sparse(K.q(1) + m^2-1,1)];   % "objective" variable y(m+1)
c = [c; zeros(m^2,1)];                       % all-0 in the psd-part
K.s = [m];
K.scomplex=1;                                %Complex Hermitian PSD
% ---------- y(2:m) complex, y(1) and y(m+1) real ----------
K.ycomplex = 2:m;
```

We have modeled the objective function by means of an artificial variable, y_{m+1}, and y_{m+1}^2 is bounded from below by the original quadratic objective function, using a $1 + m(m+1)/2$-dimensional quadratic cone. The Toeplitz matrix is modeled as

$$y_1 I + 2\sum_{i=1}^{m-1} y_{i+1} T_i, \tag{16}$$

where T_i is all-1 along the k-th upper diagonal, and zero everywhere else. Recall from problem (13) in Section 3.2, that in the real case, SeDuMi restricts the symmetric part of $\mathtt{mat}(c - A^{\mathrm{T}}y)$ to be positive semidefinite. In the complex case, SeDuMi restricts the *Hermitian part*, i.e. $\mathtt{mat}(c - A^{\mathrm{T}}y) + \mathtt{mat}(c - A^{\mathrm{T}}y)'$, to be positive semidefinite. Letting Z denote the Hermitian part of (16), we have

$$Z = y_1 I + \sum_{i=1}^{m-1} (y_{i+1} T_i + \overline{y}_{i+1} T_i^{\mathrm{T}}),$$

where $\overline{y}_{i+1}$ denotes the complex conjugate of y_{i+1}. Thus, we have indeed modeled Z as a Hermitian Toeplitz matrix, and SeDuMi further restricts it to be positive semidefinite, because of the field `K.s`. Furthermore, we tell SeDuMi to allow complex values for $y_2, y_3, \ldots, y_m$, by setting `K.ycomplex = 2:m`. Remark that unlike `K.l`, `K.q`, `K.r` and `K.s`, the field `K.ycomplex` is not involved in the definition of the symmetric cone $\mathcal{K}$ in (7)–(8).

The following lines show how to solve problem (15), for a particular 3×3 Hermitian matrix P, which is neither Toeplitz, nor positive semidefinite.

```
>> i = sqrt(-1);
>> P = [4, 1+2*i, 3-i;  1-2*i, 3.5, 0.8+2.3*i; 3+i, 0.8-2.3*i, 4]

P =

   4.0000             1.0000 + 2.0000i   3.0000 - 1.0000i
   1.0000 - 2.0000i   3.5000             0.8000 + 2.3000i
```

```
   3.0000 + 1.0000i   0.8000 - 2.3000i   4.0000

>> [At,b,c,K] = toepest(P);
>> [x,y,info] = sedumi(At,b,c,K);
>> z = c-At*y; Z = mat(z(K.q+1:length(z))); Z = (Z+Z')/2

Z =

   4.2827             0.8079 + 1.7342i   2.5574 - 0.7938i
   0.8079 - 1.7342i   4.2827             0.8079 + 1.7342i
   2.5574 + 0.7938i   0.8079 - 1.7342i   4.2827

>> eigK(z,K)'

ans =

   -0.0000    2.0517    0.0000    7.2810    5.5670
```

Instead of using the `mat()` function, one may use the `cellK()` function as follows:

```
>> z = cellK(c-At*y,K); Z=z.s{1}; Z=(Z+Z')/2
```

We have found the optimal positive semidefinite Toeplitz matrix Z, which has eigenvalues 0, 7.281 and 5.567. Checking the objective values reveals a new phenomenon:

```
>> [c'*x; b'*y]

ans =

  -1.4508 - 0.2428i
  -1.4508
```

The value of $c^{\mathrm{H}}x$, where $^{\mathrm{H}}$ means complex conjugate transpose, may no longer be real, and the same is true for $b^{\mathrm{H}}y$ in general. Obviously, we cannot minimize or maximize complex valued functions. Instead, `SeDuMi` minimizes $\operatorname{Re} c^{\mathrm{H}}x$ in the primal, and maximizes $\operatorname{Re} b^{\mathrm{H}}y$ in the dual. Here, Re stands for real part. In the sequel, we will also use the notation Im, to denote the imaginary part.

If we make `K.ycomplex = []`, then all dual multipliers y_i are restricted to be real.

```
>> K.ycomplex = [];
>> [x2,y2,info2]=sedumi(At,b,c,K);
>> [c'*x2; b'*y2]
```

```
ans =

  -4.5592 - 0.3816i
  -4.5592
```

Clearly, by restricting y to be real, the dual optimal value $\operatorname{Re} b^{\mathrm{H}} y = -y_{m+1}$ gets worse. Apparently, something has changed in the primal problem as well, since the primal optimal value has improved from -1.4508 to -4.5592. The difference is in the '$Ax = b$' restriction, as the following lines show:

```
>> [b-At'*x b-At'*x2]

ans =

   0.0000                -0.0000
   0.0000 + 0.0000i  -0.0000 + 1.8863i
  -0.0000                -0.0000 - 0.4387i
        0                      0
```

The restriction '$Ax = b$' is interpreted by `SeDuMi` as

$$\begin{cases} a_i^{\mathrm{H}} x = b_i \text{ if } i \in \texttt{K.ycomplex} \\ \operatorname{Re} a_i^{\mathrm{H}} x = b_i \text{ otherwise.} \end{cases} \tag{17}$$

By making `K.ycomplex = []`, we therefore removed the restrictions on $\operatorname{Im} Ax$, and implicitly added the restriction that $\operatorname{Im} y = 0$. Complex y-variables in the dual form correspond with complex equality constraints in the primal form.

If `size(A,2) = length(b)`, then the primal feasibility requirements are `A'*x = b`, using the complex conjugate transpose A^{H}.

The field `K.scomplex` contains a list of the PSD matrix variables, of order `K.s(K.scomplex)`, which are restricted to be Hermitian positive semidefinite matrices. For the remaining matrix variables, the primal x-variables are restricted to be real symmetric positive semidefinite, whereas the dual slack variables are restricted to be positive semi-definite on the real part only (the dual imaginary part is then unrestricted).

The field `K.xcomplex` lists the primal x-variables that are allowed to have a nonzero imaginary part. For the free and nonnegative x-components, ths imaginary part is then unrestricted in sign. For example, the restriction '$x_i \geq 0$' is interpreted by `SeDuMi` as

$$\begin{cases} x_i \in \Re_+ \text{ if } i \notin \texttt{K.xcomplex} \\ \operatorname{Re} x_i \geq 0 \text{ if } i \in \texttt{K.xcomplex}. \end{cases} \tag{18}$$

A similar convention holds for the first entry in a `q`-second order cone. The remaining entries in a second order cone that are listed in `K.xcomplex` are simply complex variables that appear in the norm-bound restriction. Only entries in the `f,l,q,r`-cones can be listed in `K.xcomplex`; the matrix variables are handled by the field `K.scomplex`.

On the dual side, `K.xcomplex` lists the equality and nonnegativity constraints for which the restriction $\operatorname{Im} c_i - a_i^{\mathrm{T}} y = 0$ must be imposed on the imaginary part. This interpretation also works for the first entry in a second order `q`-cone. The remaining entries in a second order cone that are listed in `K.xcomplex` are simply complex variables that appear in the norm-bound restriction (this is completely symmetric to the primal).

For sensitivity analysis, it is interesting to note that $\operatorname{Re} (\Delta c)^{\mathrm{H}} x$ is a supergradient for the optimal value function, under perturbations of the form $c + t\Delta c$, whereas $\operatorname{Re} (\Delta b)^{\mathrm{H}} y$ is a subgradient of the optimal value function under perturbation of b. For a discussion of sensitivity analysis in (real symmetric) semidefinite programming, see Goldfarb and Scheinberg [12].

5 optional settings

By default, `SeDuMi` fills your terminal screen with some output concerning its iterative progress. This can be an annoying feature, in particular if `SeDuMi` is merely used as a subroutine within a larger program. To suppress the on-screen output of `SeDuMi`, it suffices to set an optional parameter, `pars.fid`, to zero.

```
>> load truss1
>> pars.fid = 0;
>> [x,y,info] = sedumi(At,b,c,K,pars);
```

The structure `pars` is not only used for suppressing iterative output of `SeDuMi`. It can contain a number of optional fields, which we will discuss in this section.

The abbreviation 'fid' in `pars.fid` stands for 'file identifier': the output of `SeDuMi` will be sent to the file whose file identifier is `pars.fid`. The file identifier for the null-device is 0, which is useful for suppressing output, and for the terminal screen it is 1. Output can also be redirected to a file, e.g.

```
>> pars.fid = fopen('truss1.out','w');
>> [x,y,info]=sedumi(At,b,c,K,pars);
>> fclose(pars.fid); pars.fid = 1;
```

With the above lines, the output is redirected to the file 'truss1.out', as can be checked with the command `dbtype truss1.out`.

`SeDuMi` uses a variant of the primal–dual interior point method, which is known as the centering–predictor–corrector method [25]. There are 3 variants of the centering–predictor–corrector method implemented, which can be selected with the field `pars.alg`. With `pars.alg`

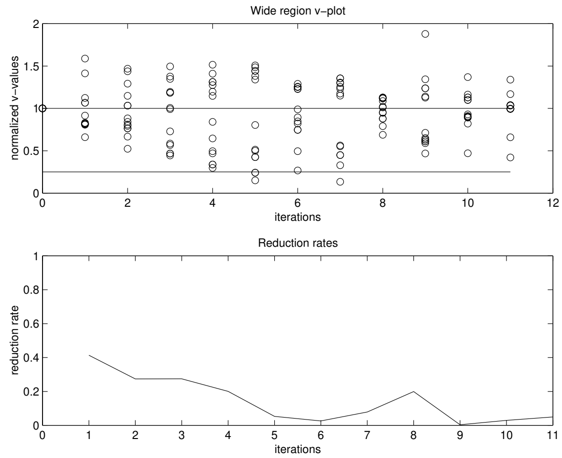

Figure 1: Plot produced by setting `pars.vplot = 1`.

`= 0`, you select a longest-step algorithm, without any second order corrector. To enhance the algorithm with a second order corrector, you can either set `pars.alg = 1` or `pars.alg = 2`. With `pars.alg = 1`, the second order corrector is derived by linearization of the so-called v-values, whereas `pars.alg = 2` uses linearization of the squared v-values, which is also known as xz-linearization. For linear programming, xz-linearization results in the well-known Mehrotra's corrector [17]. In all three variants, the centering step is determined by the central region parameter, `pars.theta`. This parameter can take any value in $(0, 1]$. At one extreme, `pars.theta = 1` results in path–following, which typically involves relatively short step lengths. Setting `pars.theta` to a smaller value, such as $1/4$, makes the algorithm work in the neighborhood of a full dimensional central region, and this typically allows for larger step lengths, see Sturm and Zhang [26]. The size of the neighborhood is controled by the parameter `pars.beta`, which can be assigned any value in $(0, 1)$. In the output of `SeDuMi` on the terminal screen, there is a column labeled 'delta', which lists the actual distance to the central region in each iteration. The step length will always be such that this is at most `pars.beta`. The ratio of the actual step length and the maximal steplength to the boundary of the cone $\mathcal{K}$ is listed in the column labeled 't/maxt'. For some iterations, an asterisk ('*') appears in front of the output line. At these iterations, the residual vector of the self–dual model has been recomputed (to avoid accumulation of numerical errors).

For research purposes, `SeDuMi` can produce a plot of the iterative v–values. This feature is activated by setting `pars.vplot` $= 1$. For problem `truss1`, this results in the plots of Figure 1. For each iteration, the first plot shows all the v–values, divided by the mean of the v-vector in that iteration. It also gives a horizontal line at value 1, representing the central path, and a horizontal line at the central region threshold, `pars.theta = 1/4`. Any v-values below this threshold will be corrected by the centering component in the succeeding iteration. The second plot shows the rate of linear reduction, which is simply

$$\frac{\text{duality gap in iteration } k}{\text{duality gap in iteration } k-1}.$$

The rate of linear reduction is also listed in the column 'rate' in the on-screen output of `SeDuMi`, and the iterative duality gap is listed under 'gap'. This is the duality gap in an artificial self–dual model, in which your original model is embedded by `SeDuMi`, using the technique of Ye, Todd and Mizuno [31]. The self–dual model gives rise to a feasibility indicator, listed in the column 'feas'. Ideally, the indicator converges to $+1$ for feasible problems, and to -1 for (primal and/or dual) infeasible problems.

Termination control is provided by the fields `maxiter`, and `eps` in the `pars` structure. `SeDuMi` will terminate successfully if it finds a solution that violates feasibility and optimality requirements by no more than `pars.eps`. The parameter `pars.maxiter` allows you to set a maximum on the number of iterations. By default, `pars.eps = 1E-9` and `pars.maxiter = 150`. A possible experiment with these parameters is to set `pars.eps = 0` in the example of minimizing $1/x_1$, which was discussed in Section 3.1.

Acknowledgments. I thank T. Terlaky for encouraging me to write this manual, and for pointing out a bug in the first public release of `SeDuMi`. P. Apkarian, M. Bengtsson, T.N. Davidson, F. Glineur, V. Prodanovic and A. Ross helped to improve the software, by providing bug reports and suggestions on the first release of the software. Numerous other users have contributed in the same manner since.Two anonymous referees have contributed in improving this document.

References

[1] F. Alizadeh, J.A. Haeberly, M.V. Nayakkankuppann, M. Overton, and S. Schmieta. *SDP-Pack user's guide.* New York University, New York, USA, 1997.

[2] B. Borchers. CSDP, a C library for semidefinite programming. Technical report, New Mexico Tech Mathematics Faculty, USA, 1997.

[3] N. Brixius and F.A. Potra. Sdpha: A MATLAB implementation of homogeneous interior-point algorithms for semidefinite programming. Technical report, Department of Computer Science, University of Iowa, Iowa City, Iowa, USA, 1998.

[4] T.N. Davidson, Z.-Q. Luo, and J.F. Sturm. A (primal form of the) positive real lemma for FIR systems. Technical report, Communications Research Laboratory, McMaster University, Hamilton, Canada, 1998. To appear.

[5] S. Deng and H. Hu. Computable error bounds for semidefinite programming. Technical report, Department of Mathematical Sciences, Northern Illinois University, DeKalb, Illinois, USA, 1996.

[6] S. Dussy and L. El Ghaoui. Multiobjective robust control toolbox for LMI-based control. Technical report, Laboratoire de Mathématiques Appliquées, ENSTA, Paris, France, 1997.

[7] L. El Ghaoui and H. Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1035–1064, 1997.

[8] L. El Ghaoui, R. Nikoukhah, and F. Delebecque. *LMITOOL: A front-end for LMI optimization, user's guide.* Laboratoire de Mathématiques Appliquées, ENSTA, Paris, France, 1995.

[9] J. Faraut and A. Korányi. *Analysis on Symmetric Cones.* Oxford Mathematical Monographs. Oxford University Press, New York, 1994.

[10] K. Fujisawa, M. Kojima, and K. Nakata. Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. *Mathematical Programming*, 79:235–253, 1997.

[11] K. Fujisawa, M. Kojima, and K. Nakata. SDPA (semidefinite programming algorithm) user's manual — version 4.10. Technical report, Dept. of Information Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152, Japan, 1998.

[12] D. Goldfarb and K. Scheinberg. On parametric semidefinite programming. Technical report, Columbia University, Department of IEOR, New York, USA, 1997.

[13] D. Goldfarb and K. Scheinberg. Stability and efficiency of matrix factorizations in interior point methods. Conference presentation, HPOPT IV Workshop, June 16–18, Rotterdam, The Netherlands.

[14] M.S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second–order cone programming. Technical report, Information Systems Lab, Stanford University, 1997. To appear in *Linear Algebra and Applications.*

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

[16] Z.-Q. Luo, J.F. Sturm, and S. Zhang. Duality results for conic convex programming. Technical Report 9719/A, Econometric Institute, Erasmus University Rotterdam, Rotterdam, The Netherlands, 1997.

[17] S. Mehrotra. On the implementation of a primal–dual interior point method. *SIAM Journal on Optimization*, 2:575–601, 1992.

[18] S. Mehrotra and Y. Ye. Finding an interior point in the optimal face of linear programs. *Mathematical Programming*, 62:497–515, 1993.

[19] H.D. Mittelmann. Several SDP-codes on problems from SDPLIB. `ftp://plato.la.asu.edu/pub/sdplib.txt`, 1998.

[20] R.D.C. Monteiro and S. Mehrotra. A general parametric analysis approach and its implication to sensitivity analysis in interior point methods. *Mathematical Programming*, 72:65–82, 1996.

[21] Y. Nesterov and M.J. Todd. Self–scaled barriers and interior–point methods for convex programming. *Mathematics of Operations Research*, 22(1):1–42, 1997.

[22] D. Peaucelle, D. Henrion and Y. Labit. User's guide for SeDuMi Interface 1.01: Solving LMI problems with SeDuMi. LAAS - CNRS, Toulouse, France. Available from `http://www.laas.fr/ peaucell/SeDuMiInt.html`.

[23] C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and algorithms for linear optimization. An interior point approach.* Series in discrete mathematics and optimization. John Wiley & Sons, New York, 1997.

[24] A.M. Ross. Optimization over cones: SDPpack versus SeDuMi. Technical Report 1998-10-22, Department IEOR, University of California at Berkeley, 1998.

[25] J.F. Sturm. *Primal–Dual Interior Point Approach to Semidefinite Programming*, volume 156 of *Tinbergen Institute Research Series.* Thesis Publishers, Amsterdam, The Netherlands, 1997.

[26] J.F. Sturm and S. Zhang. On a wide region of centers and primal–dual interior point algorithms for linear programming. *Mathematics of Operations Research*, 22(2):408–431, 1997.

[27] M.J. Todd and Y. Ye. Approximate Farkas lemmas and stopping rules for iterative infeasible–point algorithms for linear programming. *Mathematical Programming*, 81:1–21, 1998.

[28] K.C. Toh, M.J. Todd, and R.H. Tütüncü. SDPT3 - a MATLAB package for semidefinite programming. Technical report, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, USA, 1996.

[29] L. Vandenberghe and S. Boyd. *SP: Software for semidefinite programming.* Information Systems Laboratory, Electrical Engineering Department, Stanford University, Stanford, USA, 1994.

[30] S. Wu, Z.-Q. Luo, and K. Wong. Direction finding for coherent sources via Toeplitz approximation. Technical report, Communications Research Laboratory, McMaster University, Hamilton, Ontario, Canada, 1997.

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

[32] Y. Zhang. Solving large–scale linear programs by interior–point methods under the MATLAB environment. Technical report, Department of Computational and Applied Mathematics, Rice University, Houstan, Texas, USA, 1997.

A Calling Sequence

The primal canonical form for solving optimization problems with SeDuMi is

$$\min\{c^{\mathrm{T}}x \mid Ax = b,\, x \in \mathcal{K}\},$$

and the dual canonical form is

$$\max\{b^{\mathrm{T}}y \mid c - A^{\mathrm{T}}y \in \mathcal{K}\}.$$

The general calling sequence for solving the above primal–dual pair is

```
[x,y,info] = sedumi(A,b,c,K,pars)
```

Here, `K` is a MATLAB structure to define the symmetric cone $\mathcal{K}$; it consists of the following (optional) fields:

`K.f` The number of free primal variables, i.e. the number of dual equality constraints

`K.l` The number of nonnegativity constraints

`K.q` A list of dimensions of quadratic cone constraints

`K.r` A list of dimensions of rotated quadratic cone constraints

`K.xcomplex` A list of primal variables in the `f,l,q,r` blocks that are allowed to have a nonzero imaginary part. The imaginary parts of the associated dual constraints are then explicitly restricted.

`K.s` A list of orders of positive semidefiniteness constraints

`K.scomplex` A list of matrix variables that are restricted to be Hermitian positive semidefinite.

`K.ycomplex` This field is not related to $\mathcal{K}$. It lists the components of the y-variables that are complex valued. Equivalently, it lists the primal equality constraints $(Ax)_i = b_i$ that have to be satisfied not only in their real parts, but also in their imaginary parts.

The structure `K` defines $\mathcal{K}$ to be

$$\mathcal{K} = \Re^{\texttt{K.f}} \times \Re_+^{\texttt{K.l}} \times (\text{Qcone} \times \cdots \times \text{Qcone}) \times (\text{Rcone} \times \cdots \times \text{Rcone}) \times (\text{Scone} \times \cdots \times \text{Scone}),$$

In total, the number of Qcone (Rcone, Scone) components is `length(K.q)` (`length(K.r)`, `length(K.s)`). The ith Qcone is

$$\text{Qcone}_i = \{(x_1, x_2) \in \Re \times \mathcal{C}^{\texttt{K.q[i]}-1} \mid x_1 \geq \|x_2\|\},$$

where $\mathcal{C}^n$ denotes the space of complex n–tuples. The jth Rcone is

$$\text{Rcone}_j = \left\{ (x_1, x_2, x_3) \in \Re \times \Re \times \mathcal{C}^{\texttt{K.r[j]}-2} \middle| x_1 x_2 \geq \frac{1}{2}\|x_3\|^2,\ x_1 + x_2 \geq 0 \right\},$$

and the kth Scone is

$$\text{Scone}_k = \left\{ x \in \mathcal{C}^{\texttt{K.s[k]}^2} \mid \texttt{mat(x)} \text{ is Hermitian positive semidefinite} \right\},$$

for primal components. For dual components $z = c - A^{\mathrm{T}}y$, we use a slightly milder definition of Scone, viz.

$$\text{Scone}_k = \left\{ z \in \mathcal{C}^{\texttt{K.s[k]}^2} \mid \texttt{mat(z)} + \texttt{mat(z)}' \text{ is positive semidefinite} \right\}.$$

If k is *not* listed in `K.scomplex`, then the definition in the dual is even milder, namely,

$$\text{Scone}_k = \left\{ z \in \mathcal{C}^{\texttt{K.s[k]}^2} \mid \ \text{Re}\,\texttt{mat(z)} + \texttt{mat(z)}' \text{ is positive semidefinite} \right\}.$$

The length of the vector c should be

```
length(c) = K.f + K.l + sum(K.q) + sum(K.r) + sum(K.s.^2)
```

If the data (A, b, c) is real valued, then x and y will also be real valued.

The parameter `pars` is a MATLAB structure, consisting of the following (optional) fields:

`pars.fid` By default, `pars.fid=1`, which tells `SeDuMi` to produce iterative statistics on the screen. If `pars.fid=0`, then `SeDuMi` runs quietly, i.e. no screen output. In general, output is written to the file or device that is identified by the file handle `pars.fid`. A file handle is assigned to a file by the MATLAB function `fopen`, as in

```
pars.fid = fopen('truss1.out','w').
```

`pars.alg` By default, `pars.alg=2`. If `pars.alg=0`, then a first-order algorithm is used, which is not recommendable. If `pars.alg=1`, then `SeDuMi` uses the centering-predictor-corrector algorithm with v-linearization. If `pars.alg=2` then xz-linearization is used in the corrector, similar to Mehrotra's algorithm. All 3 algorithms are special instances of the generic wide-region algorithm, as discussed in Chapter 7 of Sturm [25].

`pars.theta, pars.beta` By default, `pars.theta=0.25` and `pars.beta=0.5`. These are the wide region and neighborhood parameters. Valid choices are $0 < \theta <= 1$ and $0 < \beta < 1$.

`pars.stepdif, pars.w` By default, `pars.stepdif=1` and `pars.w = [1 1]`. This means that primal/dual step length differentiation is enabled (disabled if `pars.stepdif=0`). The priorities of the relative primal, dual and gap residuals are weighted as `w(1):w(2):1`, in order to find the optimal step differentiation.

`pars.vplot` If this field is 1, then SeDuMi produces a fancy v-plot, for research purposes. Default: `vplot = 0`.

`pars.eps` The desired accuracy.

`pars.bigeps` The required accuracy to get `info.numerr < 2`.

`pars.maxiter` Maximum number of iterations, before termination.

`pars.denq` Proportion of x(i)'s for which the sparsity in A(:,i) is considered normal. Default: 0.75.

`pars.denf` A column is treated as dense if it has `pars.denf` times more nonzeros than normal. Default: 10.

`pars.stopat` Enters MATLAB debugging mode at the beginning of iteration `pars.stopat`. Default: -1.

`pars.cg` Various parameters for controling the Preconditioned conjugate gradient method (CG), which is only used if results from Cholesky are inaccurate. Type 'help sedumi' for details.

`pars.chol` Various parameters for controling the Cholesky solve. Type 'help sedumi' for details.

The output parameter `info` is a MATLAB structure, with the following fields:

`info.pinf` and `info.dinf` The feasibility status of the primal-dual problem pair, as detected by `SeDuMi`. There are three cases:

1. `pinf = dinf = 0` Then x and y are (approximate) optimal solutions, i.e. $Ax = b$, $x \in \mathcal{K}$, $c - A^{\mathrm{T}}y \in \mathcal{K}$, and $c^{\mathrm{T}}x \leq b^{\mathrm{T}}y$ (approximately).

2. `pinf = 1` Primal is infeasible, i.e. $\{x \in \mathcal{K} \mid Ax = b\} = \emptyset$. Then y is a Farkas-type solution, i.e. $b^{\mathrm{T}}y > 0$ and $-A^{\mathrm{T}}y \in \mathcal{K}$.
3. `dinf = 1` Dual is infeasible, i.e. $\{y \mid c - A^{\mathrm{T}}y \in \mathcal{K}\} = \emptyset\}$. Then x is a Farkas-type solution, i.e. $c^{\mathrm{T}}x < 0$, $Ax = 0$ and $x \in \mathcal{K}$.

`info.numerr` A positive value of `info.numerr` means that `SeDuMi` terminated without achieving the desired accuracy, because of numerical problems. If `info.numerr = 1` then the results are merely inaccurate: the solution has still achieved the accuracy denoted by `pars.bigeps`, which is `1E-3` by default. If `info.numerr = 2` then `SeDuMi` failed completely.