ADDENDUM TO THE SEDUMI USER GUIDE VERSION 1.1

IMRE PÓLIK

1. Introduction

The main goal of this reference guide is to give a summary of all the options in SeDuMi. The default value of the options is satisfactory for general use. If you experience difficulties solving some problems please report them at the SeDuMi Forum (`http://sedumi.mcmaster.ca`). If you need a longer description of SeDuMi and examples of use then consult the old (1.05R5) manual [3].

2. Input format

A full-featured SeDuMi call in Matlab is

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

where most of the parts can be omitted. With this call SeDuMi solves the following primal-dual optimization problem:

$$\begin{aligned} &\min c^T x & &\max b^T y \\ &Ax = b & &A^T y + s = c \\ &x \in K & &s \in K^*, \end{aligned}$$

where $A \in \mathbb{R}^{m\times n}$, $x, s, c \in \mathbb{R}^n$, $y, b \in \mathbb{R}^m$, K is a cone and K^* is its dual cone. Omitting `K` implies that all the variables are nonnegative. A, b and c contain the problem data, if either b or c is missing then it is treated as zero.

The structure K defines the cone in the following way: it can have fields `K.f`, `K.l`, `K.q`, `K.r` and `K.s`, for Free, Linear, Quadratic, Rotated quadratic and Semidefinite. In addition, there are fields `K.xcomplex`, `K.scomplex` and `K.ycomplex` for complex-variables.

(1) `K.f` is the number of FREE, i.e., UNRESTRICTED primal components. The dual components are restricted to be zero. E.g. if `K.f = 2` then `x(1:2)` is unrestricted, and `s(1:2)=0`. These are ALWAYS the first components in `x`.
(2) `K.l` is the number of NONNEGATIVE components. E.g., if `K.f=2`, `K.l=8` then `x(3:10)>=0`.
(3) `K.q` lists the dimensions of LORENTZ (quadratic, second-order cone) constraints. E.g., if `K.l=10` and `K.q = [3 7]` then `x(11) >= norm(x(12:13))`, `x(14)>= norm(x(15:20))`. These components ALWAYS immediately follow the `K.l` nonnegative ones. If the entries in A and/or c are COMPLEX, then the x-components in

Date: June 16, 2005.
Based on the original SeDuMi User Guide and the contents of the SeDuMi help.

norm(x(#,#)) take complex-values, whenever that is beneficial. Use K.ycomplex to impose constraints on the imaginary part of A*x.

(4) K.r lists the dimensions of Rotated LORENTZ constraints. E.g., if K.l=10, K.q=[3,7] and K.r = [4 6], then 2*x(21)x(22) >= norm(x(23:24))^2 and 2*x(25)x(26)>=norm(x(27:30))^2. These components ALWAYS immediately follow the K.q ones. Just as for the K.q-variables, the variables in norm(x(#,#)) are allowed to be complex, if you provide complex data. Use K.ycomplex to impose constraints on the imaginary part of A*x.

(5) K.s lists the dimensions of POSITIVE SEMI-DEFINITE (PSD) constraints. If K.l=10, K.q = [3 7] and K.s = [4 3], then mat(x(21:36),4) is PSD, mat(x(37:45),3) is PSD. These components are ALWAYS the last entries in x.
   - K.xcomplex lists the components in f,l,q,r blocks that are allowed to have nonzero imaginary part in the primal.
   - K.scomplex lists the PSD blocks that are Hermitian rather than real symmetric.
   - Use K.ycomplex to impose constraints on the imaginary part of A*x.

The dual multipliers y have analogous meaning as in the x>=0 case, except that instead of c-A'*y>=0 resp. -A'*y>=0, one should read that c-A'*y resp. -A'*y are in the cone that is described by K.l, K.q and K.s. In the above example, if z = c-A'*y and mat(z(21:36),4) is not symmetric/Hermitian, then positive semi-definiteness reflects the symmetric/Hermitian parts, i.e. s + s' is PSD.

If the model contains COMPLEX data, then you may provide a list K.ycomplex, with the following meaning:

- y(i) is complex if ismember(i,K.ycomplex)
- y(i) is real otherwise

The equality constraints in the primal are then as follows:

- A(i,:)*x = b(i) if imag(b(i)) ~= 0 or ismember(i,K.ycomplex)
- real(A(i,:)*x) = b(i) otherwise.

Thus, equality constraints on both the real and imaginary part of A(i,:)*x should be listed in the field K.ycomplex.

You may use EIGK(x,K) and EIGK(c-A'*y,K) to check that x and c-A'*y are in the cone K.

3. Options

Most of the options in SeDuMi can be overridden by specifying them explicitly in the structure pars. This structure can have the following fields (the default value is in brackets).

3.1. General options.

pars.fid (1): The output of SeDuMi is written to the file with handle fid. If fid=0, then SeDuMi runs quietly, i.e., there is no screen output. Use fopen to assign a handle to a file.

pars.maxiter (150): The maximum number of iterations. In most cases SeDuMi stops after 20-30 iterations, and almost never needs more than 100 iterations.

pars.eps (10^{-8})**:** Desired accuracy. If this accuracy is achieved then `info.numerr` is set to 0. If `pars.eps=0` then SeDuMi runs as long as it can make progress.

pars.bigeps (10^{-3})**:** In case the desired accuracy `pars.eps` cannot be achieved, the solution is tagged as `info.numerr=1` if it is accurate to `pars.bigeps`, otherwise it yields `info.numerr=2`.

pars.alg (2): If `pars.alg=0`, then a first-order wide region algorithm is used, not recommended. 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. The wide-region centering predictor-corrector algorithm was proposed in Chapter 7 of [2].

pars.theta (0.25), pars.beta (0.5): These are the wide region and neighborhood parameters. Valid choices are $0 <$ `pars.theta` ≤ 1 and $0 <$ `pars.beta` < 1. Setting `pars.theta=1` would restrict the iterates to follow the central path in an $N_2(\beta)$-neighbourhood. In practice, SeDuMi rounds `pars.beta` to be between 0.1 and 0.9 and `pars.theta` to be between 0.01 and 1.

pars.stepdif (2): If `pars.stepdif=0` then the primal-dual step differentiation is always disabled, while if `pars.stepdif=1` then it is always enabled. Using step differentiation helps if the problem is ill-conditioned or the algorithm is close to convergence. Setting `pars.stepdif=2` uses an adaptive scheme: it starts SeDuMi with step-differentiation disabled and enables it

- after 20 iterations, or
- if `feasratio` is between 0.9 and 1.1, or
- more than one conjugate gradient iterate is needed.

pars.w $(1, 1)$**:** If step-differentiation is enabled, SeDuMi weights the relative primal, dual and gap residuals as w(1):w(2):1 in order to find the optimal step differentiation. These number should be greater than 10^{-8}.

pars.numtol (5×10^{-7})**, pars.bignumtol** (0.9)**, pars.numlvl (0):** These options control some numerical behaviour, don't tamper with them.

3.2. **Options controlling the preprocessing.**

pars.denf (10), pars.denq (0.75): Parameters used in deciding which columns are dense and sparse.

pars.free (1): Specifies how SeDuMi handles free variables. If `pars.free=0` then free variables are converted into the difference of two nonnegative variables. This method is numerically unstable and unnecessarily increases the number of variables. If `pars.free=1` then free variables are placed inside a Lorentz cone whose first variable is unconstrained. This method is more stable and has also been suggested by Jos Sturm in [4]. Please note however, that this makes the problem nonlinear and it might take longer to solve it. If you experience such behaviour then change this setting to 0.

pars.sdp (1): Enables the SDP preprocessing routines, such as detecting diagonal SDP blocks.

3.3. **Options controlling the Cholesky factorization.** These options are subfields of `pars.chol`.

pars.chol.skip (1): Enables skipping bad pivots.

pars.chol.canceltol (10^{-12})**:** Relative tolerance for detecting cancellation during Cholesky factorization.

pars.chol.abstol (10^{-20})**:** Skips pivots falling below this value.

pars.chol.maxuden (500): Pivots in dense-column factorization so that these factors satisfy max(abs(Lk)) $\leq$ maxuden.

3.4. **Options controlling the Conjugate Gradient refinement.** Various parameters for controlling the Preconditioned conjugate gradient method (CG), which is only used if results from Cholesky factorization are inaccurate.

pars.cg.maxiter (49): Maximum number of CG-iterates (per solve). Theoretically needed is `|add|+2*|skip|`, the number of added and skipped pivots in Cholesky.

pars.cg.refine (1): Number of refinement loops that are allowed. The maximum number of actual CG-steps will thus be `1+(1+pars.cg.refine)*pars.cg.maxiter`.

pars.cg.stagtol (5×10^{-14})**:** Terminates if relative function progress is less than this number.

pars.cg.restol (5×10^{-3})**:** Terminates if residual is a `pars.cg.restol` fraction of the duality gap. Should be smaller than 1 in order to make progress.

pars.cg.qprec (1): Stores CG-iterates in quadruple precision if `pars.cg.qprec=1`.

3.5. **Debugging options.**

pars.vplot (0): If this field is 1, then SeDuMi produces a plot for research purposes.

pars.stopat (-1): SeDuMi enters debugging mode at the iterations specified in this vector.

pars.errors (0): If `pars.errors=1` then SeDuMi outputs some error measures as outlined at the 7th DIMACS challenge. The errors are both displayed in the output and returned in `info.err`.

4. Output format

4.1. **Description of the output fields.** When calling SeDuMi with three output variables (`[x,y,info]=sedumi(...)`) some useful information is returned in the structure `info`. It has the following fields:

info.cpusec: Total CPU (not wall clock) time in seconds spent in optimization.

info.timing: Detailed CPU timing information in seconds. The three numbers give the time spent in preprocessing, IPM iterations and postprocessing, respectively. Although in most cases IPM iterations take more than 90% of the time, in some cases it can be as low as 50%.

info.feasratio: The final value of the feasibility indicator. This indicator converges to 1 for problems with a complementary solution, and to -1 for strongly infeasible problems. If `info.feasratio` is somewhere in between, and the problem is not purely linear then the problem may be nasty (e.g., the optimum is not attained). Otherwise, if the problem is linear then the reason must lie in numerical problems: try to rescale the problem.

info.pinf, info.dinf: If `info.pinf=1` then there cannot be an $x \geq 0$ with $Ax = b$, and this is certified by y, viz. $b^T y > 0$ and $A^T y \leq 0$ thus y is a Farkas solution. On the other hand if `info.dinf=1` then there cannot be a y such that $c - A^T y >= 0$, and this is certified by x, viz. $c^T x < 0$, $Ax = 0$, $x \geq 0$. Thus x is a Farkas solution. If both `info.pinf` and `info.dinf` are 0 then the solution given by SeDuMi is both near primal and dual feasible.

info.numerr: Indicates how accurate the solution is. If `info.numerr = 0` then the desired accuracy (specified by `pars.eps`) is achieved. If `info.numerr = 1` then only reduced accuracy (given by `pars.bigeps`) is achieved. Finally, if `info.numerr = 2` indicates complete failure due to numerical problems.

info.err: If the option `pars.errors` is set to 1 then SeDuMi outputs some error measures as described in [1].

References

1. H. D. Mittelmann, *An independent benchmarking of SDP and SOCP solvers*, Mathematical Programming (2003), no. 95, 407–430.
2. Jos F. Sturm, *Primal-dual interior point approach to semidefinite programming*, Ph.D. thesis, Tilburg University, The Netherlands, 1997, TIR 156, Thesis Publishers Amsterdam.
3. ______, *Using SeDuMi 1.02, a Matlab toolbox for optimization over symmetric cones*, Optimization Methods and Software (1999), no. 11-12, 625–653.
4. ______, *Implementation of interior point methods for mixed semidefinite and second order cone optimization problems*, EconPapers 73, Tilburg University, Center for Economic Research, August 2002.

McMaster University, Advanced Optimization Lab
E-mail address: `poliki@mcmaster.ca`