PENLIB/SDP User's Guide

Michal Kočvara Michael Stingl

In this guide we give a description of parameters of function `sdp`, solving linear semidefinite programming problems with linear constraints. This function is part of the library PENLIB.

We solve the dual linear SDP problem with linear constraints:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} & \sum_{i=1}^{n} f^i x^i \\ \text{s.t.} \quad & \sum_{k=1}^{n} b_i^k x^k \le c_i, \qquad i = 1,\dots,m_\ell \\ & A_i^0 + \sum_{k=1}^{n} A_i^k x^k \preccurlyeq 0 \qquad i = 1,\dots,m\,. \end{aligned}$$

The matrix constraints can be written as one constraint with block diagonal matrices as follows:

$$\begin{pmatrix} A_0^1 & & & \\ & A_0^2 & & \\ & & \ddots & \\ & & & A_0^m \end{pmatrix} + \begin{pmatrix} A_1^1 & & & \\ & A_1^2 & & \\ & & \ddots & \\ & & & A_1^m \end{pmatrix} x_1 + \begin{pmatrix} A_2^1 & & & \\ & A_2^2 & & \\ & & \ddots & \\ & & & A_2^m \end{pmatrix} x_2 + \ldots + \begin{pmatrix} A_n^1 & & & \\ & A_n^2 & & \\ & & \ddots & \\ & & & A_n^m \end{pmatrix} x_n$$

Here we use the abbreviations $n :=$ `vars`, $m_\ell :=$ `constr`, and $m :=$ `mconstr`.

The input parameters are explained below. We assume that the linear constraint vectors b_i can be sparse, so we give them in standard sparse format. Similarly, we assume that the matrix constraints data can be sparse. Here we distinguish two cases: Some (many) of the matrices A_k^i for the $k-$th constraint can be empty; so we only give those matrices (for each matrix constraint) that are nonempty. Further, each of the nonempty matrices A_k^i can still be sparse; hence we give the matrices A_k^i in sparse format (value, column index, row index). Further, as all the matrices are symmetric, we only give the upper triangle.

The function `sdp` is declared as

```
int sdp(int vars, int constr, int mconstr, int* msizes, double *fx,
          double* x0, double* uoutput, double* fobj, double* ci,
          int* bi_dim, int* bi_idx, double* bi_val,
          int* ai_dim, int* ai_idx, int* ai_nzs,
          double* ai_val, int* ai_col, int* ai_row,
          int* ioptions, double* foptions,
          int* iresults, double* fresults, int* info);
```

(1) `vars`	number of variables
integer number	
(2) `constr`	number of linear constraints
integer number	
(3) `mconstr`	number of matrix constraints (diagonal blocks in each A_k)
integer number	
(4) `msizes`	sizes of the diagonal blocks $A_k^1, A_k^2, \ldots, A_k^{\texttt{mconstr}}$
integer array	*length*: `mconstr`
(5) `fx`	on exit: objective function value
double array	*length*: 1
(6) `x0`	on entry: initial guess for the solution on exit: solution vector x (Not referenced, if `x0 = 0` on entry)
double array	*length*: `vars`
(7) `uoutput`	on exit: linear multipliers u_i ($i = 1, \ldots,$ `constr`) followed by upper triangular parts of matrix multipliers U^j ($j = 1, \ldots,$ `mconstr`) in rowise storage format (Not referenced, if `uoutput = 0` on entry)
double array	*length*: `constr + msizes(1)*(msizes(1)+1)/2 + msizes(2)*(msizes(2)+1)/2 + ...+ msizes(mconstr)*(msizes(mconstr)+1)/2`
(8) `fobj`	objective vector f in full format
double array	*length*: `vars`
(9) `ci`	right-hand side vector of the linear constraint c in full format
double array	*length*: `constr`
(10) `bi_dim`	number of nonzeros in vector b_i for each linear constraint
integer array	*length*: `constr`
(11) `bi_idx`	indices of nonzeros in each vector b_i
integer array	*length*: `bi_dim(1)+bi_dim(2)+ ...+bi_dim(constr)`
(12) `bi_val`	nonzero values in each vector b_i corresponding to indices in `bi_idx`
double array	*length*: `bi_dim(1)+bi_dim(2)+ ...+bi_dim(constr)`
(13) `ai_dim`	number of nonzero blocks $A_0^i, A_1^i, \ldots, A_{\texttt{vars}}^i$ for each matrix constraint $i = 1, 2, \ldots,$ `mconstr`
integer array	*length*: `mconstr`
(14) `ai_idx`	indices of nonzero blocks for each matrix constraint
integer array	*length*: `ai_dim(1)+ai_dim(2)+ ...+ai_dim(mconstr)`
(15) `ai_nzs`	number of nonzero values in each nonzero block $A^i_{\texttt{ai\_idx(1)}}, A^i_{\texttt{ai\_idx(2)}}, \ldots, A^i_{\texttt{ai\_idx(ai\_dim(i))}}$ for each matrix constraint $i = 1, 2, \ldots,$ `mconstr`
integer array	*length*: `ai_dim(1)+ai_dim(2)+ ...+ai_dim(mconstr)`
(16) `ai_val`	nonzero values in the upper triangle of each nonzero block $A^i_{\texttt{ai\_idx(1)}}, A^i_{\texttt{ai\_idx(2)}}, \ldots, A^i_{\texttt{ai\_idx(ai\_dim(i))}}$ for each matrix constraint $i = 1, 2, \ldots,$ `mconstr`
double array	*length*: `ai_nzs(1)+ai_nzs(2)+ ...+ai_nzs(length(ai_nzs))`
(17) `ai_col`	column indices of nonzero values in the upper triangle of each nonzero block $A^i_{\texttt{ai\_idx(1)}}, A^i_{\texttt{ai\_idx(2)}}, \ldots, A^i_{\texttt{ai\_idx(ai\_dim(i))}}$ for each matrix constraint $i = 1, 2, \ldots,$ `mconstr`
integer array	*length*: `ai_nzs(1)+ai_nzs(2)+ ...+ai_nzs(length(ai_nzs))`
(18) `ai_row`	row indices of nonzero values in the upper triangle of each nonzero block $A^i_{\texttt{ai\_idx(1)}}, A^i_{\texttt{ai\_idx(2)}}, \ldots, A^i_{\texttt{ai\_idx(ai\_dim(i))}}$ for each matrix constraint $i = 1, 2, \ldots,$ `mconstr`
integer array	*length*: `ai_nzs(1)+ai_nzs(2)+ ...+ai_nzs(length(ai_nzs))`

(19) `ioptions`	integer valued options (see below)
integer array	*length*: 8
(20) `foptions`	real valued options (see below)
double array	*length*: 7
(21) `iresults`	on exit: integer valued output information (see below) (Not referenced, if `iresults = 0` on entry)
integer array	*length*: 4
(22) `fresults`	on exit: real valued output information (see below) (Not referenced, if `fresults = 0` on entry)
double array	*length*: 3
(23) `info`	on exit: error flag (see below)
integer array	*length*: 1

IOPTIONS	name/value	meaning	default
`ioption(0)`	DEF		
	0	use default values for all options	
	1	use user defined values	
`ioption(1)`	PBM_MAX_ITER	maximum number of iterations of the overall algorithm	50
`ioption(2)`	UM_MAX_ITER	maximum number of iterations for the unconstrained minimization	100
`ioption(3)`	OUTPUT	output level	1
	0	no output	
	1	summary output	
	2	brief output	
	3	full output	
`ioption(4)`	DENSE	check density of the Hessian	0
	0	automatic check. For very large problems with dense Hessian, this may lead to memory difficulties.	
	1	dense Hessian assumed	
`ioption(5)`	LS	linesearch in unconstrained minimization	0
	0	do not use linesearch	
	1	use linesearch	
`ioption(6)`	XOUT	write solution vector x in the output file	0
	0	no	
	1	yes	
`ioption(7)`	UOUT	write computed multipliers in the output file	0
	0	no	
	1	yes	

FOPTIONS	name/value	meaning	default
`foption(0)`	U0	scaling factor for linear constraints; must be positive	1.0
`foption(1)`	MU	restriction for multiplier update for linear constraints	0.7
`foption(2)`	MU2	restriction for multiplier update for matrix constraints	0.1
`foption(3)`	PBM_EPS	stopping criterium for the overall algorithm	1.0e-7
`foption(4)`	P_EPS	lower bound for the penalty parameters	1.0e-6
`foption(5)`	UMIN	lower bound for the multipliers	1.0e-14
`foption(6)`	ALPHA	stopping criterium for unconstrained minimization	1.0e-2

IRESULTS	meaning
`iresults(0)`	number of outer iterations
`iresults(1)`	number of Newton steps
`iresults(2)`	number of linesearch steps
`iresults(3)`	elapsed time in seconds

FRESULTS	meaning
`fresults(0)`	relative precision at x_{opt}
`fresults(1)`	feasibility of linear inequalities at x_{opt}
`fresults(2)`	feasibility of matrix inequalities at x_{opt}

INFO	meaning
`info(0)`	no errors occured
`info(1)`	no progress in objective value, problem probably infeasible.
`info(2)`	Cholesky factorization of Hessian failed. The result still may be useful.
`info(3)`	Maximum iteration limit exceeded. The result still may be useful.
`info(4)`	Linesearch failed. The result still may be useful.
`info(5)`	Wrong input parameters (ioptions, foptions).
`info(6)`	Memory error.
`info(7)`	Unknown error, please contact PENOPT GbR (contact@penopt.com).

*E*xample 1. Let $f = (1, 2, 3)^T$. Assume that we have no linear constraints. Assume further that we have two matrix inequality constraints, first of size (3x3), second of size (2x2). The first constraint contains full matrices, the second one diagonal matrices:

$$\begin{pmatrix} A_0^1 & & \\ & A_0^2 & \end{pmatrix} + \begin{pmatrix} A_1^1 & & \\ & A_1^2 & \end{pmatrix} x_1 + \begin{pmatrix} A_2^1 & & \\ & A_2^2 & \end{pmatrix} x_2 + \begin{pmatrix} A_3^1 & & \\ & A_3^2 & \end{pmatrix} x_3$$

The blocks A_k^1 have then 6 nonzero elements, block A_k^2 only two nonzero elements (recall that we only give elements of the upper triangular matrix). In this case

`vars` = 3

`constr` = 0

`mconstr` = 2

`msizes` = (3,2)

`x0` = (0.0,0.0,0.0) (for example)

`fobj` = (1.0,2.0,3.0)

`ci` = (0.0)

`bi_dim` = (0)

`bi_idx` = (0)

`bi_val` = (0.0)

`ai_dim` = (4,4)

`ai_idx` = (0,1,2,3,0,1,2,3)

`ai_nzs` = (6,6,6,6,2,2,2,2)

`ai_val` = $(A_0^1(1), \ldots, A_0^1(6),\ A_1^1(1), \ldots, A_1^1(6), \ldots\ldots,$
$A_0^2(1), A_0^2(2), A_1^2(1), A_1^2(2), A_2^2(1), A_2^2(2), A_3^2(1), A_3^2(2))$

`ai_col` = (0,1,2,1,2,2, 0,1,2,1,2,2,. . . . . , 0,1,0,1,0,1,0,1)

`ai_row` = (0,0,0,1,1,2, 0,0,0,1,1,2,. . . . . , 0,1,0,1,0,1,0,1)

*E*xample 2. Let again $f = (1, 2, 3)^T$. We have two linear constraints with

$$b_1 = (0, 0, 1)^T, \quad b_2 = (5, 6, 0)^T, \quad c = (3, -3)^T$$

Assume further that we have two matrix inequality constraints, first of size (3x3), second of size (2x2). The first constraint contains sparse matrices, the second one diagonal matrices. Some of the matrices are empty, as shown below:

$$\left(\begin{array}{ccc|cc} 0 & & & & \\ & 0 & & & \\ & & 0 & & \\ \hline & & & 0 & \\ & & & & 1 \end{array}\right) + \left(\begin{array}{ccc|cc} 2 & -1 & 0 & & \\ & 2 & 0 & & \\ & & 2 & & \\ \hline & & & 1 & \\ & & & & -1 \end{array}\right) x_1$$

$$+ \left(\begin{array}{ccc|cc} 0 & & & & \\ & 0 & & & \\ & & 0 & & \\ \hline & & & 3 & \\ & & & & -3 \end{array}\right) x_2 + \left(\begin{array}{ccc|cc} 2 & 0 & -1 & & \\ & 2 & 0 & & \\ & & 2 & & \\ \hline & & & 0 & \\ & & & & 0 \end{array}\right) x_3$$

In this case

`vars` = 3

`constr` = 2

`mconstr` = 2

`msizes` = (3,2)

`x0` = (0.0,0.0,0.0) (for example)

`fobj` = (1.0,2.0,3.0)

`ci` = (3.0 , -3.0)

`bi_dim` = (1,2)

`bi_idx` = (2,0,1)

`bi_val` = (1.0, 5.0, 6.0)

`ai_dim` = (2,3)

`ai_idx` = (1,3,0,1,2)

`ai_nzs` = (4,4,1,2,2)

`ai_val` =(2.0,-1.0,2.0,2.0, 2.0,-1.0,2.0,2.0, 1.0, 1.0,-1.0, 3.0,-3.0)

`ai_col` = (0,1,1,2, 0,2,1,2, 1, 0,1, 0,1)

`ai_row` = (0,0,1,2, 0,0,1,2, 1, 0,1, 0,1)