PENBMI User's Guide

Michal Kočvara Michael Stingl

In this guide we give a description of parameters of function `bmi`, solving linear semidefinite programming problems with bilinear matrix inequality constraints.

We solve the SDP problem with linear and bilinear matrix inequality constraints:

$$\begin{aligned}
&\min_{x\in\mathbb{R}^n} \sum_{k=1}^{n} f_k x_k \\
&\text{s.t.} \qquad \sum_{k=1}^{n} b_k^i x_k \le c^i, \qquad i = 1,\dots,m_\ell \\
&\qquad\qquad A_0^i + \sum_{k=1}^{n} x_k A_k^i + \sum_{k=1}^{n}\sum_{\ell=1}^{n} x_k x_\ell K_{k\ell}^i \preccurlyeq 0, \qquad i = 1,\dots,m\,.
\end{aligned}$$

The linear part of 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`.

In the same way we can write down the bilinear part of the matrix constraints.

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 and $K_{k\ell}^i$ 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 $A_k^i, K_{k\ell}^i$ can still be sparse; hence we give the matrices $A_k^i, K_{k\ell}^i$ 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 bmi(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* ki_dim, int* ki_idx, int* kj_idx, int* ki_nzs,
          double* ki_val, int* ki_col, int* ki_row,
          int* ioptions, double* foptions);
```

`vars`	number of variables
integer number	
`constr`	number of linear constraints
integer number	
`mconstr`	number of matrix constraints (diagonal blocks in each A_k)
integer number	
`msizes`	sizes of the diagonal blocks $A_k^1, A_k^2, \ldots, A_k^{\texttt{mconstr}}$
integer array	*length*: `mconstr`
`fx`	on exit: objective function value
double array	*length*: 1
`x0`	on entry: initial guess for the solution on exit: solution vector x (Not referenced, if `x0 = 0` on entry)
double array	*length*: `vars`
`uoutput`	on exit: linear multipliers u_i ($i = 1, \ldots,$ `constr`) followed by upper triangular parts (stored row-wise) of matrix multipliers U^j ($j = 1, \ldots,$ `mconstr`) (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`
`fobj`	objective vector f in full format
double array	*length*: `vars`
`ci`	right-hand side vector of the linear constraint c in full format
double array	*length*: `constr`
`bi_dim`	number of nonzeros in vector b_i for each linear constraint
integer array	*length*: `constr`
`bi_idx`	indices of nonzeros in each vector b_i
integer array	*length*: `bi_dim(1)+bi_dim(2)+ ...+bi_dim(constr)`
`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)`
`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`
`ai_idx`	indices of nonzero blocks for each matrix constraint
integer array	*length*: `ai_dim(1)+ai_dim(2)+ ...+ai_dim(mconstr)`
`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)`
`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))`
`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))`
`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))`

`ki_dim`	number of nonzero blocks $K^i_{k\ell}, k, \ell \in \{1, \dots, \texttt{vars}\}$, for each matrix constraint $i = 1, 2, \dots, \texttt{mconstr}$
integer array	*length*: `mconstr`
`ki_idx`	first indices of nonzero blocks for each matrix constraint
integer array	*length*: `ki_dim(1)+ki_dim(2)+ ...+ki_dim(mconstr)`
`kj_idx`	isecond indices of nonzero blocks for each matrix constraint
integer array	*length*: `ki_dim(1)+ki_dim(2)+ ...+ki_dim(mconstr)`
`ki_nzs`	number of nonzero values in each nonzero block $K^i_{\texttt{ki\_idx}(\alpha),\texttt{kj\_idx}(\alpha)}, \alpha = 1, \dots, \texttt{ki\_dim(i)}$, for each matrix constraint $i = 1, 2, \dots, \texttt{mconstr}$
integer array	*length*: `ki_dim(1)+ki_dim(2)+ ...+ki_dim(mconstr)`
`ki_val`	nonzero values in the upper triangle of each nonzero block $K^i_{\texttt{ki\_idx}(\alpha),\texttt{kj\_idx}(\alpha)}, \alpha = 1, \dots, \texttt{ki\_dim(i)}$, for each matrix constraint $i = 1, 2, \dots, \texttt{mconstr}$
double array	*length*: `ki_nzs(1)+ki_nzs(2)+ ...+ki_nzs(length(ki_nzs))`
`ki_col`	column indices of nonzero values in the upper triangle of each nonzero block $K^i_{\texttt{ki\_idx}(\alpha),\texttt{kj\_idx}(\alpha)}, \alpha = 1, \dots, \texttt{ki\_dim(i)}$, for each matrix constraint $i = 1, 2, \dots, \texttt{mconstr}$
integer array	*length*: `ki_nzs(1)+ki_nzs(2)+ ...+ki_nzs(length(ki_nzs))`
`ki_row`	row indices of nonzero values in the upper triangle of each nonzero block $K^i_{\texttt{ki\_idx}(\alpha),\texttt{kj\_idx}(\alpha)}, \alpha = 1, \dots, \texttt{ki\_dim(i)}$, for each matrix constraint $i = 1, 2, \dots, \texttt{mconstr}$
integer array	*length*: `ki_nzs(1)+ki_nzs(2)+ ...+ki_nzs(length(ki_nzs))`

`ioptions`	integer valued options below
integer array	*length*: 8
`foptions`	real valued options below
double array	*length*: 7

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	brief 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

*E*xample 1. Let $f = (1, 2, 3)^T$. Assume that we have no linear constraints. Assume further that we have two linear 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 linear 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)

*E*xample 3. Finally, we solve a problem in three variables with box constraints and one linear-bilinear matrix constraint of dimension three:

$$\begin{aligned} &\min_{x\in\mathbb{R}^3} x_3 \\ &\text{s.t.} \qquad x_1 \in [-0.5, 2], \quad x_2 \in [-3, 7] \\ &\qquad\qquad A_0 + x_1 A_1 + x_2 A_2 + x_1 x_2 K_{12} - x_3 I \preccurlyeq 0 \end{aligned}$$

with

$$A_0 = \begin{pmatrix} -10 & -0.5 & -2 \\ -0.5 & 4.5 & 0 \\ -2 & 0 & 0 \end{pmatrix} \qquad A_1 = \begin{pmatrix} 9 & 0.5 & 0 \\ 0.5 & 0 & -3 \\ 0 & -3 & -1 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} -1.8 & -0.1 & -0.4 \\ -0.1 & 1.2 & -1 \\ -0.4 & -1 & 0 \end{pmatrix} \qquad K_{12} = \begin{pmatrix} 0 & 0 & 2 \\ 0 & -5.5 & 3 \\ 2 & 3 & 0 \end{pmatrix}$$

```
   vars = 3;
 constr = 4;
mconstr = 1;
 msizes = [3];
     x0 =[1.,0.,0.];
   fobj = [0,0,1];
     ci =[0.5,2,3,7];
 bi_dim =[1,1,1,1];
 bi_idx =[0,0,1,1];
 bi_val =[-1,1,-1,1];
 ai_dim =[4];
 ai_idx =[0;1;2;3];
 ai_nzs =[4;4;5;3];
 ai_val =[-10,-0.5,-2,4.5, 9, 0.5,-3,-1, -1.8,-0.1,-0.4,1.2,-1, -1,-1,-1];
 ai_col =[0,1,2,1, 0,1,2,2, 0,1,2,1,2, 0,1,2];
 ai_row =[0,0,0,1, 0,0,1,2, 0,0,0,1,1, 0,1,2];
 ki_dim =[1];
 ki_idx =[1];
 kj_idx =[2];
 ki_nzs =[3];
 ki_val =[2,-5.5,3];
 ki_col =[2,1,2];
 ki_row =[0,1,1];
```