PENNON User's Guide (Version 0.9)

Michal Kočvara Michael Stingl
www.penopt.com

November 4, 2008

Contents

1 Installation

1.1 Unpacking

UNIX versions
The distribution is packed in file `pennon.tar.gz`. Put this file in an arbitrary directory. After uncompressing the file to `pennon.tar` by command `gunzip pennon.tar.gz`, the files are extracted by command `tar -xvf pennon.tar`.

Win32 version
The distribution is packed in file `pennon.zip`. Put this file in an arbitrary directory and extract the files by `PKZIP`.
In both cases, the directory PENNON0.9 containing the following files and subdirectories will be created

LICENSE: file containing the PENNON license agreement;

bin: directory containing the files

pennon0.9(.exe), the binary executable with AMPL interface,

nlpsdp.mod, a model file of a sample problem in AMPL format,
nlpsdp.sdp, an sdp file of a sample problem in AMPL format,
nlpsdp.nl, an nl-file created by AMPL from `nlpsdp.mod`;

cond.mod, a model file of a sample problem in AMPL format,
cond.sdp, an sdp file of a sample problem in AMPL format,
cond.nl, an nl-file created by AMPL from `cond.mod`;

corr.mod, a model file of a sample problem in AMPL format,
corr.sdp, an sdp file of a sample problem in AMPL format,
corr.nl, an nl-file created by AMPL from `corr.mod`;

fmo.mod, a model file of a sample problem in AMPL format,
fmo.dat, a data file of a sample problem in AMPL format,
fmo.sdp, an sdp file of a sample problem in AMPL format,
fmo.nl, an nl-file created by AMPL from `fmo.mod`;

lib: directory containing the PENNON libraries;
matlab: directory containing the files
pennonm.c, the MATLAB interface file,
penoutm.c, MATLAB version of `penout.c`,
make_pennonm.m, M-file containing MEX link command,
nlp.m, f.m, df.m, hf.m, g.m dg.m, hg.m, M-files
defining a sample problem in PEN format.
nlpsdp, bfgs, cond, directories containing examples from the last section
matlab\cond: directory containing the files for example `cond`
matlab\corr: directory containing the files for example `corr`
matlab\fmo: directory containing the files for example `fmo`
matlab\nlpsdp: directory containing the files for example `nlpsdp`
matlab\truss: directory containing the files for example `truss`

1.2 Compilation

Requirements

For successful compilation and linkage, depending on the operating system and the program to be created, the following software packages have to be installed:

UNIX versions

- MATLAB version 5.0 or later including MEX compiler package and `gcc` compiler package (MATLAB dynamic link library `pennonm.*`)

Win32 version

- MATLAB version 5.0 or later including MEX compiler package and VISUAL C++ version 6.0 or later (MATLAB dynamic link library `pennonm.*`)

To build a MATLAB dynamic link library `pennonm.*`

Start MATLAB, go to directory `matlab` and invoke link command by

`make_pennonm`.

In case the user wants to use his/her own LAPACK, BLAS or ATLAS implementations, the M-file in directory `matlab` has to be modified appropriately.

2 The problem

We solve optimization problems with a nonlinear objective subject to nonlinear inequality and equality constraints and semidefinite bound constraints:

$$\begin{aligned} \min_{x\in\mathbb{R}^n, Y_1\in\mathbb{S}^{p_1},\ldots,Y_k\in\mathbb{S}^{p_k}} & f(x,Y) \\ \text{subject to} \quad & g_i(x,Y) \le 0, && i=1,\ldots,m_g \\ & h_i(x,Y) = 0, && i=1,\ldots,m_h \\ & \underline{\lambda}_i I \preceq Y_i \preceq \overline{\lambda}_i I, && i=1,\ldots,k\,. \end{aligned} \qquad \text{(NLP-SDP)}$$

Here

- $x \in \mathbb{R}^n$ is the vector variable
- $Y_1 \in \mathbb{S}^{p_1}, \ldots, Y_k \in \mathbb{S}^{p_k}$ are the matrix variables, k symmetric matrices of dimensions $p_1 \times p_1, \ldots, p_k \times p_k$
- we denote $Y = (Y_1, \ldots, Y_k)$
- f, g_i and h_i are C^2 functions from $\mathbb{R}^n \times \mathbb{S}^{p_1} \times \ldots \times \mathbb{S}^{p_k}$ to $\mathbb{R}$
- $\underline{\lambda}_i$ and $\overline{\lambda}_i$ are the lower and upper bounds, respectively, on the eigenvalues of Y_i, $i = 1, \ldots, k$

3 The algorithm

To simplify the presentation of the algorithm, *we only consider inequality constraints*. For the treatment of the equality constraints, see [4].
The algorithm is based on a choice of penalty/barrier functions $\varphi_g : \mathbb{R} \to \mathbb{R}$ that penalize the inequality constraints and $\Phi_P : \mathbb{S}^p \to \mathbb{S}^p$ penalizing the matrix inequalities. These

functions satisfy a number of properties (see [4]) that guarantee that for any $p_i > 0,\ i = 1, \ldots, m_g$, we have

$$g_i(x) \le 0 \iff p_i \varphi_g(g_i(x)/p_i) \le 0, \quad i = 1, \ldots, m_g$$

and

$$Z \preceq 0 \iff \Phi_P(Z) \preceq 0, \quad Z \in \mathbb{S}^p\,.$$

This means that, for any $p_i > 0$, problem (NLP-SDP) has the same solution as the following "augmented" problem

$$\begin{aligned} &\min_{x\in\mathbb{R}^n, Y_1\in\mathbb{S}^{p_1},\ldots,Y_k\in\mathbb{S}^{p_k}} f(x,Y) \\ &\text{subject to} \quad p_i\varphi_g(g_i(x,Y)/p_i) \le 0, \qquad i = 1,\ldots,m_g \\ &\qquad\qquad \Phi_P(\underline{\lambda}_i I - Y_i) \preceq 0, \qquad i = 1,\ldots,k \\ &\qquad\qquad \Phi_P(Y_i - \overline{\lambda}_i I) \preceq 0, \qquad i = 1,\ldots,k\,. \end{aligned} \tag{NLP-SDP$_\phi$}$$

The Lagrangian of (NLP-SDP$_\phi$) can be viewed as a (generalized) augmented Lagrangian of (NLP-SDP):

$$\begin{aligned} F(x,Y,u,\underline{U},\overline{U},p,P) = f(x,Y) &+ \sum_{i=1}^{m_g} u_i p_i \varphi_g(g_i(x,Y)/p_i) \\ &+ \sum_{i=1}^{k} \langle \underline{U}_i, \Phi_P(\underline{\lambda}_i I - Y_i)\rangle + \sum_{i=1}^{k} \langle \overline{U}_i, \Phi_P(Y_i - \overline{\lambda}_i I)\rangle\,; \end{aligned} \tag{1}$$

here $u \in \mathbb{R}^{m_g}$ and $\underline{U}_i, \overline{U}_i$ are Lagrange multipliers associated with the standard constraints and the matrix inequality constraints, respectively.
The algorithm combines ideas of the (exterior) penalty and (interior) barrier methods with the Augmented Lagrangian method.

Algorithm 3.1 *Let x^1, Y^1 and $u^1, \underline{U}^1, \overline{U}^1$ be given. Let $p_i^1 > 0,\ i = 1, \ldots, m_g$ and $P^1 > 0$. For $k = 1, 2, \ldots$ repeat till a stopping criterium is reached:*

(i) *Find x^{k+1} and Y^{k+1} such that* $\|\nabla_x F(x^{k+1}, Y^{k+1}, u^k, \underline{U}^k, \overline{U}^k, p^k, P^k)\| \le K$

(ii) $u_i^{k+1} = u_i^k \varphi_g'(g_i(x^{k+1})/p_i^k), \quad i = 1, \ldots, m_g$

$\underline{U}_i^{k+1} = D_{\mathcal{A}} \Phi_{P^k}((\underline{\lambda}_i I - Y_i); \underline{U}_i^k), \quad i = 1, \ldots, k$

$\overline{U}_i^{k+1} = D_{\mathcal{A}} \Phi_{P^k}((Y_i - \overline{\lambda}_i I); \overline{U}_i^k), \quad i = 1, \ldots, k$

(iii) $p_i^{k+1} < p_i^k,\ i = 1, \ldots, m_g$

$P^{k+1} < P^k$.

The approximate unconstrained minimization in Step (i) is performed by the modified Newton method with line-search or by a variant of the Trust Region method (for details, see [4]). The minimization is optionally stopped when either

$$\|\nabla_x F(x^{k+1}, Y^{k+1}, u^k, \underline{U}^k, \overline{U}^k, p^k, P^k)\|_2 \le \alpha$$

or

$$\|\nabla_x F(x^{k+1}, Y^{k+1}, u^k, \underline{U}^k, \overline{U}^k, p^k, P^k)\|_2 \le \alpha \cdot \|u_i^k - u_i^k \varphi_g'(g_i(x^{k+1})/p_i^k)\|_2$$

or

$$\|\nabla_x F(x^{k+1}, Y^{k+1}, u^k, \underline{U}^k, \overline{U}^k, p^k, P^k)\|_{H^{-1}} \le \alpha \|\nabla_x F(x^k, Y^k, u^k, \underline{U}^k, \overline{U}^k, p^k), P^k\|_{H^{-1}}$$

with optional parameter α; by default, $\alpha = 10^{-1}$.
The multipliers calculated in Step (ii) are restricted in order to satisfy:

$$\mu < \frac{u_i^{k+1}}{u_i^k} < \frac{1}{\mu}$$

with some positive $\mu \leq 1$; by default, $\mu = 0.3$. Similar estimates are satisfied for the matrix multipliers.
The update of the penalty parameter p in Step (iii) is performed in two different ways: Either the penalty vector is updated by some constant factor dependent on the initial penalty parameter π, or the penalty vector is updated only in case the progress of the method becomes too slow. In either case, the penalty update is stopped, if p_{eps} (by default 10^{-6}) is reached.
Algorithm 3.1 is stopped when both of the inequalities holds:

$$\frac{|f(x^k, Y^k) - F(x^k, Y^k, u^k, \underline{U}^k, \overline{U}^k, p^k, P^k)|}{1 + |f(x^k, Y^k)|} < \epsilon\,, \qquad \frac{|f(x^k, Y^k) - f(x^{k-1}, Y^{k-1})|}{1 + |f(x^k, Y^k)|} < \epsilon\,,$$

where ϵ is by default 10^{-7} (parameter `precision`). Optionally the user can choose a stopping critrion based on the KKT error.

Equality handling The equality constraints are treated in two principally different ways. We either reformulate the as two inequalities or treat them directly on the level of the subproblem. Below we just describe in detail the second approach. Consider the optimization problem

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &\text{subject to} \\ &\qquad \mathcal{G}(x) \preccurlyeq 0\,, \\ &\qquad \mathcal{S}(x) \preccurlyeq 0\,, \\ &\qquad h(x) = 0\,, \end{aligned} \tag{2}$$

where f, $\mathcal{G}$ and $\mathcal{S}$ are defined as in the previous sections and $h : \mathbb{R}^n \to \mathbb{R}^d$ represents a set of equality constraints. Then we define the augmented Lagrangian

$$\begin{aligned} &\overline{F}(x, U, v, p, s) = \\ &f(x) + \langle U, \Phi_p(\mathcal{G}(x)) \rangle_{\mathbb{S}_m} + s\Phi_{\text{bar}}(\mathcal{S}(x)) + v^\top h(x)\,, \end{aligned} \tag{3}$$

where $U, \Phi, \Phi_{\text{bar}}, p, s$ are defined as before and $v \in \mathbb{R}^d$ is the vector of Lagrangian multipliers associated with the equality constraints. Now, on the level of the subproblem, we attempt to find an approximate solution of the following system (in x and v):

$$\begin{aligned} \nabla_x \overline{F}(x, U, v, p, s) = 0\,, \\ h(x) = 0\,, \end{aligned} \tag{4}$$

where the penalty and barrier parameters s, p, as well as the multiplier U are fixed. In order to solve systems of type (4), we apply the damped Newton method. Descent directions are calculated utilizing the factorization routine MA27 from the Harwell subroutine library ([**?**]) in combination with an inertia correction strategy as described in [**?**]. Moreover, the step length is derived using an augmented Lagrangian merit function defined as

$$\overline{F}(x, U, v, p, s) + \frac{1}{2\mu}\|h(x)\|_2^2$$

along with an Armijo rule. Now we are ready to state the modified algorithm:

Algorithm 3.2 *Let x^1, U^1 and v_1 be given. Let $p^1 > 0, s^1 > 0, \alpha^1 > 0$. For $k = 1, 2, \ldots$ repeat until a stopping criterion is reached:*

$$
\begin{aligned}
(i) \quad & \text{Find } x^{k+1}, v^{k+1} \text{ satisfying} \\
& \|\nabla_x \overline{F}(x^{k+1}, U^k, v^{k+1}, p^k, s^k)\| \le \alpha^k \\
& \|h(x^k)\| \le \alpha^k \\
(ii) \quad & U^{k+1} = D_{\mathcal{G}} \Phi_p(\mathcal{G}(x^{k+1}); U^k) \\
(iii) \quad & p^{k+1} \le p^k\,, \quad s^{k+1} < s^k\,, \quad \alpha^{k+1} < \alpha^k\,.
\end{aligned}
$$

4 AMPL interface

AMPL is a comfortable modelling language for optimization problems. For a description of AMPL we refer to [5] or `www.ampl.com`.

4.1 Matrix variables in AMPL

AMPL does not support matrix variables. However, the format of our problem (2) allows us to use them:

- within an AMPL script (file `<name>`.mod), matrix variables are treated as vectors, using the function svec : $\mathbb{S}^m \to \mathbb{R}^{(m+1)m/2}$ defined by

$$
\text{svec} \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ & a_{22} & \ldots & a_{2m} \\ & & \ddots & \vdots \\ sym & & & a_{mm} \end{pmatrix} = (a_{11}, a_{12}, a_{22}, \ldots, a_{1m}, a_{2m}, \ldots, a_{mm})^T
$$

  Example: Assume we have a matrix variable $X \in \mathbb{S}^3$

$$
X = \begin{pmatrix} x_1 & x_2 & x_4 \\ x_2 & x_3 & x_5 \\ x_4 & x_5 & x_6 \end{pmatrix}
$$

  and a constraint

$$
\text{Tr}\,(XA) = 3 \qquad \text{with } A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}
$$

  The matrix variable is treated as a vector

$$
\text{svec}(X) = (x_1, x_2, \ldots, x_6)^T
$$

  and the above constraint is equivalent to the following constraint:

$$
x_3 + 2x_4 = 3\,.
$$

  The corresponding lines in the `<name>`.mod file would be

```
var x{1..6}
...
subject to c: x[3]+2*x[4] = 3;
```

- The order of the variables should be:

1. nonlinear matrix (the matrix or its elements are invovled in a nonlinear expression)
2. standard (real) variables
3. linear matrix variables

Example: Consider a problem with constraints

$$\begin{aligned} XPX &= I \\ y_1 A_1 + y_2 A_2 &\succeq 0 \end{aligned}$$

in variables $X \in \mathbb{S}^3$ and $y \in \mathbb{R}^2$. The second constraint should be re-written using a slack variable $S \in \mathbb{S}^2$:

$$\begin{aligned} XPX &= I \\ y_1 A_1 + y_2 A_2 &= S \\ S &\succeq 0 \end{aligned}$$

In this example, X is a nonlinear matrix variable, y a standard real variable, and S a linear matrix variable. So the definition of these variables in the `<name>`.mod file must follow the order:

```
var x{1..6}  % vectorized upper triangle of X
var y{1..2}
var s{1..3}  % vectorized upper triangle of S
```

- data needed to identify the matrix variables (their number and size) are included in a file `<name>`.sdp that is read by the PENNON.

4.2 Preparing AMPL input data

4.2.1 The sdp-file

File `<name>`.sdp includes the following data. The order of the data is compulsory:

1. *Number of matrix variables*
   Give the total number of matrix variables (k in (NLP-SDP)).
2. *Number of nonlinear matrix variables*
   The number of matrix variables involved in nonlinear constraints
3. *Number of linear matrix variables*
   The number of matrix variables involved only in linear constraints
4. *Sizes matrix variables*
   k numbers, the sizes of the single matrix variables. If a matrix variable is a $p \times p$ matrix, its size is p.
5. *Lower eigenvalue bounds*
   k numbers, the lower bounds on the eigenvalues in the single matrix variables (numbers $\underline{\lambda}_i$ in problem (NLP-SDP)). For an unconstrained variable, use a large negative number $-1 \cdot 10^{38}$.
6. *Upper eigenvalue bounds*
   k numbers, the upper bounds on the eigenvalues in the single matrix variables (numbers $\overline{\lambda}_i$ in problem (NLP-SDP)). For an unconstrained variable, use a large positive number $1 \cdot 10^{38}$.

7. *Constraint types*
   k numbers identifying the treatment of the lower/upper bound constraints in the algorithm

   0 ... *standard* treatment of the constraints (the constraints may be infeasible at some iterations)

   1 ... *lower strict*; the lower bounds will always be strictly feasible during the iretative process, will be treated by an inner barrier

   2 ... *upper strict*; same as above, now for the upper bounds

   3 ... *lower/upper strict*; both constraints will always remain strictly feasible

   4 ... *slack*, the matrix variable is a slack variable

8. *non-zeroes per matrix variable*
   k numbers; if the matrix variable is dense, input the number $(p+1)p/2$, where p is the matrix size; if the matrix variable is known to be sparse, give the actual number of its nonzero elements (all other elements of this matrix will be ignored by the code)

9. *nonzero structure per matrix variable* (not required if all matrices are dense)
   for each sparse matrix variable with s non-zero elements, give s lines, one for each non-zero element, where each line looks as follows
   `matrix_number  row_index  column_index`
   The numbering of the row/column indices is zero-based!
   For a dense matrix variable, there is no need to give these data. Hence if all the matrix variables are dense, this part of the sdp-file will be void.

Below is an example of the sdp-file with three matrix variables, the first one dense and the other two sparse (diagonal).

```
###################################
# sdp example 1 (PENNON 0.9)      #
###################################
#
# Nr. of matrix variables
#
    3
#
# Nr. of non-lin. matrix variables
#
    0
#
# Nr. of lin. matrix variables
#
    3
#
# Matrix variable sizes
#
    3   2   2
#
# lower eigenvalue bounds
#
        0.  0.   -2.
#
# upper eigenvalue bounds
#
```

```
    1.0E38  1.0E38      0.
#
# Constraint types
# 0=standard, 1=lower strict, 2=upper strict,
# 3=lower/upper strict, 4=slack (not implemented yet)
#
    0   0   0
#
# nonzeroes per matrix variable
#
    6   2   2
#
# nonzero structure per matrix variable
#
    2   0   0
    2   1   1
    3   0   0
    3   1   1
```

4.3 Redundant constraints

Important: AMPL may reorder the variables, which would lead to a collaps of our treatment of the matrix variables. To avoid the reordering, all variables should be involved in nonlinear constraints. If this is not the case, it is important to add redundant nonlinear constraints that include all variables. We recommend to add constraints of the type

```
red{i in 1..n}: x[i]*x[i]<=1000000;
```

This kind of redundant constraints will not change the sparsity structure of the Hessian and will not influence the efficiency of the code.

4.4 Running PENNON

PENNON is called in the standard AMPL style, i.e., either by a sequence like

```
> model example.mod;
> data example.dat;
> option presolve 0;
> options solver pennon;
> options pennon_options 'sdpfile=example.sdp outlev=2';  (for instance)
> solve;
```

within the AMPL environment or from the command line by

```
> ampl -P -obexample example.mod example.dat
> pennon example.nl sdpfile=example.sdp outlev=2
```

It is necessary to surpress AMPL preprocessing, either by the command `option presolve 0;` within AMPL or using option -P when running AMPL from the command line.

Sample files are included in directory `bin`.

4.5 Program options

The options are summarized in Table 1.

Recommendations

- Whenever you know that the problem is convex, use `convex=1`.
- When you have problems with convergence of the algorithm, try to
  - decrease `pinit`, e.g., `pinit=0.01` (This should be particulary helpful for nonconvex problems, if an initial guess of the solution is available).
  - increase (decrease) `uinit`, e.g., `uinit=10000`.
  - switch to Trust Region algorithm by `ncmode=1`
  - decrease `alpha`, e.g., `alpha=1e-3`
  - change stopping criterion for inner loop by setting `nwtstopcrt=1`

PENNON–AMPL options

option	meaning	default
`alpha`	stopping parameter α for the Newton/Trust region method in the inner loop	1.0E-1
`alphaupd`	update of α	1.0e0
`autoini`	automatic initialization of multipliers 0 . . . off 1 . . . nonlinear (nonconvex) mode 2 . . . lp/qp mode	1
`autoscale`	automatic scaling 0 . . . on 1 . . . off	0
`cgtolmin`	minimum tolerance for the conjugate gradient algorithm	5.0e-2
`cgtolup`	update of tolerance for the conjugate gradient algorithm	1.0e0
`cmaxnzs`	tuning parameter for Hessian assembling in "nwtmode" (put > 0 to switch it on)	-1
`convex`	convex problem? 0 . . . generally nonconvex 1 . . . convex	0
`eqltymode`	treatment of equality constraints 0 . . . two inequalities, symmetric 1 . . . two inequalities, unsymmetric 2 . . . augmented lagrangian 3 . . . direct 4 . . . direct (only nonlinear equalities)	3
`filerep`	output to file 0 . . . no 1 . . . yes	0
`hessianmode`	check density of the Hessian 0. . . automatic 1 . . . dense	0
`ignoreinit`	ignore initial solutions 0. . . do not ignore 1 . . . do ignore	0
`KKTscale`	equilibrate linear system matrix 0 . . . no 1 . . . yes	0
`maxit`	maximum number of outer iterations	100
`mu`	restriction factor μ of multiplier update	0.5
`nwtiters`	maximum number of iterations in the inner loop (Newton or Trust region method)	100

PENNON–AMPL options *(cont.)*

`nwtmode`	linear system solver 0 . . . Cholesky method 1 . . . conjugate gradient method 2 . . . conjugate gradient method with approximate Hessian calculation 3 . . . conjugate gradient method to dual system	0
`nwtstopcrit`	stopping criterium for the inner loop 0 . . . $\|\nabla L(x^{k+1})\|_2 < \alpha$ 1 . . . $\|\nabla L(x^{k+1})\|_2 < \alpha \cdot \|u_i^k - u_i^{k+1}\|_2$ 2 . . . $\|\nabla L(x^{k+1})\|_{H^{-1}} < \alpha \cdot \|\nabla L(x^k)\|_{H^{-1}}$	0
`objno`	objective number in the AMPL .mod file	1
`ordering`	ordering for MA57	4
`outlev`	output level 0 . . . silent mode 1 . . . brief output 2 . . . full output	1
`penalty`	penalty function 0. . . logarithmic barrier + quadratic penalty 1 . . . reciprocal barrier + quadratic penalty	0
`penup`	penalty update	0.5
`penupmode`	penalty update is performed: 0 . . . adaptively 1 . . . after each outer iteration	0
`peps`	minimal penalty	1.0E-7
`pinit`	initial penalty	1.0E0
`pivtol`	pivot tolerance for MA27/57	1.0E-2
`precision`	required final precision	1.0E-7
`precKKT`	required final precision of the KKT conditions	1.0E-5
`precond`	preconditioner type 0. . . no preconditioner 1 . . . diagonal 2 . . . L-BFGS 3 . . . approximate inverse 4 . . . symmetric Gauss-Seidel	0
`SDPfile`	name of the SDP input file	
`timing`	timing destination 0 . . . no 1 . . . stdout 2 . . . stderr 3 . . . both	0
`uinit`	initial multiplier scaling factor	1.0
`uinitbox`	initial multiplier scaling factor for box constraints	1.0
`uinitnc`	initial multiplier scaling factor for nonlinear constraints	1.0
`umin`	minimal multiplier	1.0E-10
`usebarrier`	Use (mod.) barrier approach for boxes? 0 . . . no 1 . . . barrier 2 . . . strict modified barrier	0
`usesdpbarrier`	Use barrier approach for SDP variables? 0 . . . no 1 . . . yes	0

PENNON–AMPL options *(cont.)*

`version`	report PENNON version 0 . . . yes 1 . . . no	0
`wantsol`	solution report without -AMPL. Sum of 0 . . . do not write .sol file 1 . . . write .sol file 2 . . . print primal variable 4 . . . print dual variable 8 . . . do not print solution message	0

5 MATLAB interface

5.1 Calling PENNONM from MATLAB

5.1.1 User provided functions

The user is required to provide six MATLAB functions. The names of the functions can be chosen by the user; here we use the following names:

`f` ... evaluates the objective function
`df` ... evaluates the gradient of objective function
`hf` ... evaluates the Hessian of objective function
`g` ... evaluates the constraints
`dg` ... evaluates the gradient of constraints
`hg` ... evaluates the Hessian of constraints

Similarly as in the AMPL interface, **matrix variables are treated as vectors**, using the function svec : $\mathbb{S}^m \to \mathbb{R}^{(m+1)m/2}$ defined by

$$\text{svec}\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ & a_{22} & \dots & a_{2m} \\ & & \ddots & \vdots \\ sym & & & a_{mm} \end{pmatrix} = (a_{11}, a_{12}, a_{22}, \dots, a_{1m}, a_{2m}, \dots, a_{mm})^T$$

Important: The order of the variables should be:

1. standard (real) variables;
2. matrix variables.

The parameter `nvar` contains the number of **all** variables. For instance, if we have two matrix mariables, one dense of size 2 (i.e. 3 unknowns) and one sparse with 7 nonzeros (in the triangular part), and 4 standard (real) variables, then `nvar` $= 3 + 7 + 4 = 14$.
The specification of the user-defined functions will be explained using the sample problem (NLP-SDP2).

`f`

```
function [fx] = f(x)
h = [2.2; -1.1; 1.9; -1.1; 2.1]./6;
x=reshape(x,length(x),[]);
fx = (x-h)'*(x-h);
```

Arguments:

`x`	vector of length n storing current iterate x_{it} (input)
`fx`	variable storing $f(x_{it})$ (output)

Description:

- Compute $f(x_{it})$ and store it in `fx`.

`df`

```
function [nnz,ind,val] = df(x)
h = [2.2; -1.1; 1.9; -1.1; 2.1]./6;
x=reshape(x,length(x),[]);
```

```
nnz = length(x);
ind = 1:nnz;
val = 2.*(x-h);
```

Arguments:

`x`	vector of length n storing current iterate x_{it} (input)
`nnz`	variable storing number of non-zeros of ∇f (output)
`ind`	`nnz` $\times$ 1 matrix storing non-zero structure of ∇f (output)
`val`	`nnz` $\times$ 1 matrix storing non-zero values of ∇f (output)

Description:

1. Compute $\nabla f(x_{it})$;
2. Assign non-zero structure to `ind` and the corresponding values to `val`.

Note:

- Non-zero structure should be constant.

`hf`

```
function [nnz, row, col, val] = hf(x)
nnz = length(x);
row = 1:nnz;
col = 1:nnz;
val = 2.*ones(nnz);
```

Arguments:

`x`	vector of length n storing current iterate x_{it} (input)
`nnz`	variable storing number of non-zeros of $\nabla^2 f$ (output)
`row`	(`nnz` $\times$ 1 matrix) non-zero row indices of $\nabla^2 f$ (output)
`col`	(`nnz` $\times$ 1 matrix) non-zero column indices of $\nabla^2 f$ (output)
`val`	(`nnz` $\times$ 1 matrix) storing non-zero values of $\nabla^2 f$ (output)

Description:

1. Compute $\nabla^2 f(x_{it})$;
2. Assign non-zero structure to `row` and `col` and the corresponding values to `val`.

Note:

- Non-zero structure should be constant;
- **Values should be assigned to lower triangular part of $\nabla^2 f(x_{it})$ only**.

`g`

```
function [gx] = g(i, x)
gx = x(1)+x(3)+x(5)-1.0;
```

Arguments:

`i`	variable storing constraint number (input)
`x`	vector of length n storing current iterate x_{it} (input)
`gx`	variable storing $g_i(x_{it})$ (output)

Description: Compute $g_i(x_{it})$ and store it in `gx`.

Note:

- Parameter i is zero based;
- Linear constraints should be specified **after** nonlinear constraints.

`dg`

```
function [nnz,ind, val] = dg(i, x)
nnz=3;
ind = [1;3;5];
val = [1;1;1];
```

Arguments:

`i`	variable storing constraint number (input)
`x`	vector of length n storing current iterate x_{it} (input)
`nnz`	variable storing number of non-zeros of ∇g_i (output)
`ind`	`nnz` $\times 1$ matrix storing non-zero structure of ∇g_i (output)
`val`	`nnz` $\times 1$ matrix storing non-zero values of ∇g_i (output)

Description:

1. Compute $\nabla g_i(x_{it})$;
2. Assign non-zero structure to `ind` and the corresponding values to `val`.

Note:

- Linear constraints should be specified **after** nonlinear constraints;
- Parameter i is zero based;
- Non-zero structure should be constant.

`hg`

```
function [nnz, row, col, val] = hgs_bfgs(i, x)
nnz = 0;
row = 0;
col = 0;
val = 0;
```

Arguments:

`i`	variable storing constraint number (input)
`x`	vector of length n storing current iterate x_{it} (input)
`nnz`	variable storing number of non-zeros of $\nabla^2 g_i$ (output)
`row`	`nnz` $\times$ 1 matrix non-zero row indices of $\nabla^2 g_i$ (output)
`col`	`nnz` $\times$ 1 matrix non-zero column indices of $\nabla^2 g_i$ (output)
`val`	`nnz` $\times$ 1 matrix storing non-zero values of $\nabla^2 g_i$ (output)

Description:

1. Compute $\nabla^2 g_i(x_{it})$;
2. Assign non-zero structure to `row` and `col` and the corresponding values to `val`.

Note:

- Linear constraints should be specified **after** nonlinear constraints;
- Parameter i is zero based;
- Non-zero structure should be constant;
- **Values should be assigned to lower triangular part of $\nabla^2 g_i(x_{it})$ only.**

5.2 The `pen` input structure in MATLAB

The user must create a MATLAB structure array with fields described below in Table 2. The structure also specifies the names of the user defined functions.

Remark 5.1 Arrays `lbv` and `ubv` define lower and upper bounds on variables; in case of matrix variables, these are bounds on the elements of the matrices. Arrays `lbmv` and `ubmv`, on the other hand, define spectral bounds on the matrices, i.e., bounds on the smallest and largest eigenvalues.

Remark 5.2 When using the (automatic removal of) slack variables (array `mtype`) it is important to follow these points:

- the slack variables should be tha last ones in the list of variables
- the equality constraint that defines the slack variable should be formulated in such a way, that the slack variable has a positive sign
- the constraints on the slack variables are always of the type $S \succeq 0$

Example: Consider a problem in variables $y, z \in \mathbb{R}$, $X \in \mathbb{S}^p$ with constraints

$$X \succeq zI \tag{5}$$
$$X \preceq yI \tag{6}$$

To transorm the constraints into our standard structure we need to introduce two slack matrix variables P and Q, and replace (5) and (6) by

$$X - zI - P = 0 \tag{7}$$
$$X - yI - Q = 0 \tag{8}$$
$$P \succeq 0$$
$$Q \preceq 0$$

If we want to use the automatic slack removal (option `mtype=4`), we have to

Table 2: The pen structure

name	*description*	*type*	*length*
nvars	number of variables	integer	number
nconstr	number of constraints (including linear)	integer	number
nlin	number of linear constraints	integer	number
nsdp	number of matrix variables	integer	number
blks	row dimensions of matrix variables	integer	nsdp
nnz_gradient	maximal number of non-zero entries in user specified gradients	integer	number
nnz_hessian	maximal number of non-zero entries in user specified Hessians	integer	number
lbv	lower bounds on variables	double	nvars
ubv	upper bounds on variables	double	nvars
lbc	lower bounds on constraints	double	nconstr
ubc	upper bounds on constraints	double	nconstr
lbmv	lower bounds on matrix variables (min. eigenvalues)	double	nsdp
ubmv	upper bounds on matrix variables (max. eigenvalues)	double	nsdp
mtype	type of matrix constraints 0 ... standard (constraints may be infeasible at some iterations) 1 ... lower strict (the lower bounds will always be strictly feasible during the iterative process) 2 ... upper strict (same as above, now for the upper bounds) 3 ... lower/upper strict (both constraints will always remain strictly feasible) 4 ... slack (the matrix is a slack variable)	double	nsdp
mnzs	non-zeros per matrix; if the matrix is dense, input the number $(p+1)p/2$, where p is the matrix size; if the matrix is known to be sparse, give the actual number of its nonzero elements (all other elements of this matrix will be ignored by the code)	integer	nsdp
mrow	element row indices per matrix (not required if all matrices are dense); for each sparse matrix with s non-zero elements, give s numbers of row indices of the elements; **the numbers are zero-based**	integer	varies
mcol	element column indices per matrix (not required if all matrices are dense); for each sparse matrix with s non-zero elements, give s numbers of column indices of the elements; **the numbers are zero-based**	integer	varies
xinit	initial guess for the solution	double	nvars
my_f	actual name of the file f.m	char	
my_f_gradient	actual name of the file df.m	char	
my_f_hessian	actual name of the file hf.m	char	
my_g	actual name of the file g.m	char	
my_g_gradient	actual name of the file dg.m	char	
my_g_hessian	actual name of the file hg.m	char	
ioptions	integer valued options	integer	18
doptions	real valued options	double	14

- order the matrix variables as X, P, Q or X, Q, P, in `lbmv, ubmv, mtype`, etc. If only of of the marices P, Q is to be removed, it has to be the last one. For instance, if only Q is to be removed as a slack variable, the order must be X, P, Q and the array `mtype` will be

```
mtype = [0, 0, 4]
```

- constraints (7) and (8) must be reformulated such that P and Q have positive signs, i.e.,

$$-X + zI + P = 0 \tag{9}$$
$$-X + yI + Q = 0 \tag{10}$$
$$P \succeq 0$$
$$Q \preceq 0$$

- constraint (10) must further be reformulated such that the slack Q is positive definite (not negative definite as above). The final version of the constraints is then

$$-X + zI + P = 0$$
$$X - yI + Q = 0$$
$$P \succeq 0$$
$$Q \succeq 0$$

Remark 5.3 For a dense matrix, there is no need to give data `mrow,mcol`. Hence if all the matrices are dense, arrays `nsdp,mrow,mcol` can be omitted.

For the sample problem (NLP1) the structure can be as follows:

```
Infinity = 1.0E38;
n = 5;
pen.nvars = n;
pen.nlin = 1;
pen.nconstr = 1;
pen.nsdp = 1;
pen.blks = [3];
pen.nnz_gradient = n;
pen.nnz_hessian = n;
pen.lbv = -Infinity.*ones(n,1);
pen.ubv = Infinity.*ones(n,1);
pen.lbc = [0];
pen.ubc = [0];
pen.lbmv = [0];
pen.ubmv = [Infinity];
pen.mtype = [0];
pen.mnzs = [5];
pen.mrow = [0;0;1;1;2];
pen.mcol = [0;1;1;2;2];
pen.xinit=[1;0;1;0;1];
pen.my_f = 'f';
pen.my_f_gradient = 'df';
pen.my_f_hessian = 'hf';
pen.my_g = 'g';
```

```
pen.my_g_gradient = 'dg';
pen.my_g_hessian = 'hg';
pen.ioptions = [100 100 2 0 0 0 1 0 0 1 0 0 0 -1 0 1 0];
pen.doptions = [1.0E-2 1.0E0 1.0E-0 1.0E-2 5.0E-1 5.0E-1...
                  1.0E-6 1.0E-12 1.0e-7 0.05 1.0 1.0 1.0];
```

A sample implementation is included in the files `nlp.m`, `f.m`, `df.m`, `hf.m`, `g.m`, `dg.m` and `hg.m` in directory `matlab`.

IOPTIONS	name/value	meaning	default
`ioptions(1)`	maxit	maximum numbers of outer iterations	100
`ioptions(2)`	nwtiters	maximum number of iterations in inner loop	100
`ioptions(3)`	outlev 0 1 2 3	output level no output only options are displayed brief output full output	2
`ioptions(4)`	hessianmode 0 1	check density of hessian automatic dense	0
`ioptions(5)`	autoscale 0 1	automatic scaling on off	0
`ioptions(6)`	convex 0 1	convex problem ? no yes	0
`ioptions(7)`	eqltymode 0 1 2 3	the way to treat equality constraints as two inequalities, unsymmetric initialization as two inequalities, symmetric initialization handled by standard augmented lagrangian direct handling (all equalities)	3
`ioptions(8)`	ignoreinit 0 1	ignore initial solutions ? do not ignore ignore	0
`ioptions(9)`	cholmode 0 1 2	cholesky system mode Solve directly Solve augmented system Split into two systems	0
`ioptions(10)`	nwtstopcrit 0 1 2	stopping criterion for the inner loop $\|\nabla L(x^{k+1})\|_2 < \alpha$ $\|\nabla L(x^{k+1})\|_2 < \alpha \cdot \|u_i^k - u_i^{k+1}\|_2$ $\|\nabla L(x^{k+1})\|_{H^{-1}} < \alpha \cdot \|\nabla L(x^k)\|_{H^{-1}}$	2
`ioptions(11)`	penalty 0 1	penalty function logarithmic barrier + quadratic penalty reciprocal barrier + quadratic penalty	0
`ioptions(12)`	nwtmode 0 1 2 3	mode of solving the Newton system cholesky (standard) cg (with exact hessian) cg (with appr. hessian) cg (with user provided Hessian-vector routine)	0

ioptions(13)	prec 0 1 2 3 4	preconditioner for the cg method no precond diagonal precond bfgs precond appr. inverse precond sgs precond	0
ioptions(14)	cmaxnzs -1 > 0	tuning parameter for Hessian assembling in nwt-mode 1-3 off on	-1
ioptions(15)	autoini 0 1 2	automatic initialization of multipliers off nonlinear (nonconvex) mode lp/qp mode	0
ioptions(16)	penup 0 1	penalty parameter update is performed adaptively after each outer iteration	1
ioptions(17)	usebarrier 0 1 2	box constraint mode no special treatment use (strict) barrier function use (strict) modified barrier function	0
ioptions(18)	dercheck 0 1 2	derivative check no derivative check check gradients check hessians	0

DOPTIONS	name/value	meaning	default
doptions(1)	precision	required final precision	1.0e-7
doptions(2)	uinit	initial multilplier scaling factor	1.0
doptions(3)	pinit	initial penalty	1.0
doptions(4)	alpha	stopping parameter alpha for the Newton/Trust region method in the inner loop	0.01
doptions(5)	mu	restriction factor of multiplier update	0.5
doptions(6)	penup	penalty update	0.1
doptions(7)	peps	minimal penalty	1.0e-8
doptions(8)	umin	minimal multiplier	1.0e-12
doptions(9)	preckkt	precision of the KKT conditions	1.0e-1
doptions(10)	cgtolmin	minimum tolerance of the conjugate gradient algorithm	5.0e-2
doptions(11)	cgtolup	update of tolerance of the conjugate gradient algorithm	1.0e0
doptions(12)	uinitbox	initial multiplier box constraints	1.0e0
doptions(13)	uinitnc	initial multiplier nonlinear constraints	1.0e0
doptions(14)	uinitsdp	initial multiplier sdp constraints	1.0e0

5.3 The PENNONM function call

In MATLAB, PENNONM is called with the following arguments:

```
[f,x,u,status,iresults,dresults] = pennonm(pen);
```

where

pen . . . the input structure described in the previous section

f . . . the value of the objective function at the computed optimum

x . . . the value of the dual variable at the computed optimum

u . . . the value of the primal variable at the computed optimum

status . . . exit information (see below)

iresults . . . a 4x1 matrix with elements as described below

dresults . . . a 5x1 matrix with elements as described below

IRESULTS	meaning
`iresults(1)`	number of outer iterations
`iresults(2)`	number of inner iterations
`iresults(3)`	number of linesearch steps
`iresults(4)`	ellapsed time in seconds

DRESULTS	meaning
`dresults(1)`	primal objective
`dresults(2)`	relative precision at x_{opt}
`dresults(3)`	feasibility at x_{opt}
`dresults(4)`	complementary slackness at x_{opt}
`dresults(5)`	gradient of augmented lagrangian at x_{opt}

STATUS	meaning
`status = 0`	converged: optimal solution
`status = 1`	converged: suboptimal solution (gradient large)
`status = 2`	converged: solution primal infeasible
`status = 3`	aborted: no progress, problem may be primal infeasible
`status = 4`	aborted: primal unbounded or initial multipliers too small
`status = 5`	aborted: iteration limit exceeded
`status = 6`	aborted: line search failure
`status = 7`	aborted: aborted: cholesky solver failed
`status = 8`	aborted: wrong parameters
`status = 9`	aborted: resource limit
`status = 10`	aborted: internal error, please contact PENOPT Gbr (contact @penopt.com)
`status = 11`	aborted: error in user's memory allocation
`status = 12`	aborted: error in user supplied routines

6 Examples

6.1 NLP-SDP example

Consider the following simple NLP-SDP example in matrix variable $X \in \mathbb{S}^3$:

$$\begin{aligned} \min_X \ & \sum_{i,j=1}^{3} (X_{ij} - H_{ij})^2 \\ \text{subject to} \ & \\ & \operatorname{Tr} X = 1 \\ & X \succeq 0 \end{aligned} \tag{11}$$

where

$$H = \frac{1}{6} \begin{pmatrix} 2.2 & -1.1 & 0 \\ -1.1 & 1.9 & -1.15 \\ 0 & -1.15 & 2.1 \end{pmatrix}$$

We will treat the matrix variable as a sparse (tri-diagonal) matrix.

6.1.1 AMPL interface

`nlpsdp.mod`

```
var x{1..5} default 0;
param h{1..5};

minimize Obj: sum{i in 1..5} (x[i]-h[i])^2;
subject to
   l1:
    x[1]+x[3]+x[5] = 6;

data;
param h:=
1 2.2 2 -1.1 3 1.9 4 -1.15 5 2.1;
```

`nlpsdp.sdp`

```
# Nr. of matrix variables
    1
# Nr. of non-lin. matrix variables
    0
# Nr. of lin. matrix variables
    1
# Matrix variable sizes
    3
# lower eigenvalue bounds
        0.
# upper eigenvalue bounds
    1.0E38
# Constraint types
    0
# nonzeroes per matrix variable
    5
# nonzero structure per matrix variable
    1   0   0
```

```
    1   0   1
    1   1   1
    1   1   2
    1   2   2
```

6.1.2 MATLAB interface

f.m

```
function [fx] = f(x)
h = [2.2; -1.1; 1.9; -1.1; 2.1]./6;
x=reshape(x,length(x),[]);
fx = (x-h)'*(x-h);
```

df.m

```
function [nnz,ind,val] = df(x)
h = [2.2; -1.1; 1.9; -1.1; 2.1]./6;
x=reshape(x,length(x),[]);
nnz = length(x);
ind = 1:nnz;
val = 2.*(x-h);
```

hf.m

```
function [nnz,row, col, val] = hf(x)
nnz = length(x);
row = 1:nnz;
col = 1:nnz;
val = 2.*ones(nnz);
```

g.m

```
function [gx] = g(i, x)
gx = x(1)+x(3)+x(5)-1.0;
```

dg.m

```
function [nnz,ind, val] = dg(i, x)
nnz=3;
ind = [1;3;5];
val = [1;1;1];
```

hg.m

```
function [nnz, row, col, val] = hgs_bfgs(i, x)
nnz = 0;
row = 0;
col = 0;
val = 0;
```

nlpsdp.m

```
n = 5;
Infinity = 1.0E38;
pen.nvars = n;
pen.nlin = 1;
pen.nconstr = 1;
pen.nsdp = 1;
pen.blks = [3];

pen.nnz_gradient = n;
pen.nnz_hessian = n;
pen.lbv = -Infinity.*ones(n,1);
pen.ubv = Infinity.*ones(n,1);
pen.lbc = [0];
pen.ubc = [0];
pen.lbmv = [0];
pen.ubmv = [Infinity];
pen.mtype = [0];
pen.mnzs = [5];
pen.mrow = [0;0;1;1;2];
pen.mcol = [0;1;1;2;2];

pen.xinit=[1;0;1;0;1];
pen.my_f = 'f';
pen.my_f_gradient = 'df';
pen.my_f_hessian = 'hf';
pen.my_g = 'g';
pen.my_g_gradient = 'dg';
pen.my_g_hessian = 'hg';
pen.ioptions = [100 100 2 0 0 0 1 0 0 1 0 0 0 -1 0 1 0];
pen.doptions = [1.0E-2 1.0E0 1.0E-0 1.0E-2 5.0E-1 5.0E-1...
                1.0E-6 1.0E-12 1.0e-7 0.05 1.0 1.0 1.0];

[w1,w2]=pennonm(pen)
```

Below is an output of the command

```
>> nlpsdp
Variables: 5
NL-constraints: 0
L-constraints: 1
Number of bounds: 0 (l:0 u:0)

Dense Problem !
Number of nonlinear constraints (ineq): 0
Number of linear constraints (ineq): 2

Number of Variables                          5
   - bounded below                           0
   - bounded above                           0
Number of Matrix variables                   1
   - degrees of freedom                      5
   - bounded below                           1
   - bounded above                           0
Number of Nonlinear Equalities               0
Number of Nonlinear Inequalities             0
```

```
Number of Linear Equalities                1
Number of Linear Inequalities              0

******************************************
PENNON 0.9
------------------------------------------
******************************************

Max./Min. Lin-Mult.: 1.000000 / 1.000000
maximal penalty: 1.000000, penalty update: 0.562341
*************************************************************************
* it |     obj      | (U,G(x)) |  ||dF||  |   feas   |   pmin   | Nwt  | Fact |
*************************************************************************
|   0| 5.88000e+000 | 4.7e+000 | 1.2e+001 | 3.0e+000 | 1.0e+000 |    0 |    0 |
|   1| 4.57213e-002 | 2.8e+000 | 1.7e-004 | 1.7e-001 | 1.0e+000 |    3 |    3 |
|   2| 3.19019e-002 | 6.0e-001 | 7.3e-003 | 2.1e-002 | 5.0e-001 |    1 |    1 |
|   3| 1.64866e-002 | 1.1e-001 | 1.8e-003 | 1.0e-002 | 2.5e-001 |    1 |    1 |
|   4| 1.36399e-002 | 1.9e-002 | 1.2e-004 | 1.8e-003 | 1.3e-001 |    1 |    1 |
|   5| 1.33582e-002 | 2.5e-003 | 8.1e-007 | 1.8e-004 | 6.3e-002 |    1 |    1 |
|   6| 1.33345e-002 | 2.1e-004 | 4.8e-010 | 8.9e-006 | 3.1e-002 |    1 |    1 |
|   7| 1.33334e-002 | 1.3e-005 | 5.9e-014 | 2.2e-007 | 1.6e-002 |    1 |    1 |
*************************************************************************
Objective                        1.3333362749702404e-002
Relative Precision               1.2950484187658137E-005
Gradient Augm. Lagrangian        5.9044538720747749E-014
Complementary Slackness          1.2950484187658137E-005
Feasibility                      2.2061216231605840E-007
Feasibility (LMI)                0.0000000000000000E+000
Outer Iterations                                       7
Inner Iterations                                       9
Linesearch steps                                       9
Start time                     Sun Jan 06 19:05:23 2008
End time                       Sun Jan 06 19:05:24 2008
Process time                         0 h  0 min  1 sec
*************************************************************************

w1 =
    0.0133
w2 =
    2.1333   -1.1000    1.8333   -1.1000    2.0333
>>
```

The optimal matrix is thus

$$X = \begin{pmatrix} 2.1333 & -1.1000 & 0.0 \\ -1.1000 & 1.8333 & -1.1000 \\ 0.0 & -1.1000 & 2.0333 \end{pmatrix}$$

The corresponding MATLAB files for this example are included in the directory `matlab/nlpsdp`.

6.2 Correlation matrix with the constrained condition number

We consider the problem of finding a nearest correlation matrix:

$$\begin{aligned} &\min_X \sum_{i,j=1}^{n} (X_{ij} - H_{ij})^2 \qquad (12)\\ &\text{subject to}\\ &\quad X_{ii} = 1, \quad i = 1, \ldots, n\\ &\quad X \succeq 0 \end{aligned}$$

This problem is based on a practical application; see [7]. Assume that the original correlation matrix is

$$H_{\text{orig}} = \begin{pmatrix} 1 & -0.44 & -0.20 & 0.81 & -0.46 \\ -0.44 & 1 & 0.87 & -0.38 & 0.81 \\ -0.20 & 0.87 & 1 & -0.17 & 0.65 \\ 0.81 & -0.38 & -0.17 & 1 & -0.37 \\ -0.46 & 0.81 & 0.65 & -0.37 & 1 \end{pmatrix}$$

When we solve problem (12) with $H := H_{\text{orig}}$, the solution will be, as expected, the original matrix H_{orig}.
We now add a new asset class, that means, we add one row and column to the original matrix. The new data is based on a different frequency than the original part of the matrix, which means that the new matrix is no longer positive definite:

$$H_{\text{ext}} = \begin{pmatrix} 1 & -0.44 & -0.20 & 0.81 & -0.46 & -0.05 \\ -0.44 & 1 & 0.87 & -0.38 & 0.81 & -0.58 \\ -0.20 & .87 & 1 & -0.17 & 0.65 & -0.56 \\ 0.81 & -0.38 & -0.17 & 1 & -0.37 & -0.15 \\ -0.46 & 0.81 & 0.65 & -0.37 & 1 & -0.08 \\ -0.05 & -0.58 & -0.56 & -0.15 & 0.08 & 1 \end{pmatrix}$$

Let us find the nearest correlation matrix to H_{ext} by solving (12). We obtain the following result (for the presentation of results, we will use matlab output in short precision):

```
X =
    1.0000   -0.4420   -0.2000    0.8096   -0.4585   -0.0513
   -0.4420    1.0000    0.8704   -0.3714    0.7798   -0.5549
   -0.2000    0.8704    1.0000   -0.1699    0.6497   -0.5597
    0.8096   -0.3714   -0.1699    1.0000   -0.3766   -0.1445
   -0.4585    0.7798    0.6497   -0.3766    1.0000    0.0608
   -0.0513   -0.5549   -0.5597   -0.1445    0.0608    1.0000
```

with eigenvalues

```
eigen =
    0.0000    0.1163    0.2120    0.7827    1.7132    3.1757
```

As we can see, one eigenvalue of the nearest correlation matrix is zero. This is highly undesirable from the application point of view. To avoid this, we can add lower (and upper) bounds on the matrix variable, i.e., constraints

$$\underline{\lambda} I \preceq X \preceq \overline{\lambda} I\,.$$

This would be reflected in the following lines in the `nlpsdp.m` matlab code from page 23:

```
pen.lbmv = lambda_min;
pen.ubmv = lambda_max;
```

However, the application requires a more general approach when we only want to bound the condition number of the nearest correlation matrix. This can be guaranteed by introducing a pair of new variables $y, z \in \mathbb{R}$ and adding the following set of constraints to (12):

$$X \succeq zI \tag{13}$$
$$X \preceq yI \tag{14}$$
$$y \leq \kappa z \tag{15}$$

where κ is the required condition number. Notice that the above constraints do not fit into our required NLP-SDP problem structure (see page 3). The standard way to transorm the condition constraints into our standard structure is to introduce two new (slack) matrix variables, say, P and Q, and replace (13) and (14) by

$$\begin{aligned} X - zI - P &= 0 \\ X - yI - Q &= 0 \\ P &\succeq 0 \\ Q &\preceq 0 \end{aligned}$$

We may not like the additional matrix variables, as they increase the problem size considerably. There are two ways how avoid using them.
First, we may use the automatic slack removal option (`mtype = 4`). The above constraints then should be reformulated as

$$\begin{aligned} -X + zI + P &= 0 \\ X - yI + Q &= 0 \\ P &\succeq 0 \\ Q &\succeq 0 \end{aligned}$$

(see Remark 5.2 on page 16). All data and m-files are prepared with the matrices present in the formulation; then we set

```
pen.mtype = [0; 4; 4]
```

end PENNON will automatically remove the slacks from the formulation, such that the actual calculations will only use variables y, z and X. The corersponding m-files can be found in the directory `matlab/cond_slack`.
Second, we can rewrite constraints (13)–(14) as

$$I \preceq \widetilde{X} \preceq \kappa I \tag{16}$$

assuming that $y = \kappa z$ and using the transormation of the variable X:

$$z\widetilde{X} = X\,.$$

Now, of course, we also have to change the other constraints and the objective function; the new problem of finding the nearest correlation matrix with a boudned condition number reads as follows:

$$\min_{z,\widetilde{X}} \sum_{i,j=1}^{n} (z\widetilde{X}_{ij} - H_{ij})^2 \tag{17}$$

subject to

$$\begin{aligned} z\widetilde{X}_{ii} &= 1, \quad i = 1, \ldots, n \\ I &\preceq \widetilde{X} \preceq \kappa I \end{aligned}$$

The new problem now has the NLP-SDP problem structure required by PENNLP (see page 3). When solving the problem by PENNLP, with $\kappa = 10$, we get the solution after 11 outer and 37 inner iterations. The solution is

```
z = 0.2866
```

and

```
Xtilde =
    3.4886   -1.3170   -0.7780    2.4761   -1.4902   -0.2456
   -1.3170    3.4886    2.4175   -1.1005    2.0926   -1.4715
   -0.7780    2.4175    3.4886   -0.5392    1.9269   -1.7145
    2.4761   -1.1005   -0.5392    3.4886   -1.3455   -0.4515
   -1.4902    2.0926    1.9269   -1.3455    3.4886   -0.2008
   -0.2456   -1.4715   -1.7145   -0.4515   -0.2008    3.4886
```

After the back substitution $X = \frac{1}{z}\widetilde{X}$, we get the nearest correlation matrix

```
X =
    1.0000   -0.3775   -0.2230    0.7098   -0.4272   -0.0704
   -0.3775    1.0000    0.6930   -0.3155    0.5998   -0.4218
   -0.2230    0.6930    1.0000   -0.1546    0.5523   -0.4914
    0.7098   -0.3155   -0.1546    1.0000   -0.3857   -0.1294
   -0.4272    0.5998    0.5523   -0.3857    1.0000   -0.0576
   -0.0704   -0.4218   -0.4914   -0.1294   -0.0576    1.0000
```

with eigenvalues

```
eigenvals =
    0.2866    0.2866    0.2867    0.6717    1.6019    2.8664
```

and the condition number equal to 10.
Below we show the corresponding AMPL files `cond.mod` and `cond.sdp`. These can be found in directory `bin`.

`cond.mod`

```
param h{1..21};
set indi within {1..21};
var x{1..21} default 0;
var z ;

minimize Obj: sum{i in 1..21} (z*x[i]-h[i])^2;
subject to
   b{i in 1..21}: x[i]*x[i]<=10000;
   bj:            z*z<=10000;
   l1{i in indi}:
        z*x[i] = 1;

data;
param h:=
   1  1.00  2 -0.44  3  1.00  4 -0.20  5  0.87  6  1.00  7  0.81
   8 -0.38  9 -0.17 10  1.00 11 -0.46 12  0.81 13  0.65 14 -0.37
  15  1.00 16 -0.05 17 -0.58 18 -0.56 19 -0.15 20  0.08 21  1.00;
set indi:=
   1  3  6  10  15  21;
```

`cond.sdp`

```
# Nr. of matrix variables
        1
# Nr. of non-lin. matrix variables
        1
# Nr. of lin. matrix variables
        0
# matrix variable sizes
        6
# lower eigenvalue bounds
        1.0
# upper eigenvalue bounds
        10.
# Constraint types
        0
# nonzeroes per matrix variable
        21
# nonzero structure per matrix variable
```

Important: As mentioned earlier, AMPL may reorder the variables. It is thus important to add redundant nonlinear constraints that include all variables. In this case, we added constraints

```
b{i in 1..21}: x[i]*x[i]<=10000;
bj:            z*z<=10000;
```

The corresponding MATLAB files for this example are included in the directory `matlab/cond`.

6.3 Truss topology optimization

The single-load truss topology optimization problem can be formulated as a linear semidefinite program (see, e.g., [1]):

$$\min_{t\in\mathbb{R}^m} \sum_{i=1}^m t_i \tag{18}$$

subject to

$$\begin{pmatrix} \sum_{i=1}^m t_i A_i & f \\ f^T & \gamma \end{pmatrix} \succeq 0$$

$$t_i \geq 0, \quad i = 1,\ldots,n$$

Here $A_i \in \mathbb{S}^n$, $i = 1,\ldots,m$ are given symmetric matrices and $f \in \mathbb{R}^n$, $\gamma \in \mathbb{R}$ given data. To cast the problem into our canonical NLP-SDP form, we introduce a slack variable $S \in \mathbb{S}^{n+1}$:

$$\min_{t\in\mathbb{R}^m} \sum_{i=1}^m t_i \tag{19}$$

subject to

$$S - \begin{pmatrix} \sum_{i=1}^m t_i A_i & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0_{n\times n} & f \\ f^T & \gamma \end{pmatrix}$$

$$S \succeq 0$$

$$t_i \geq 0, \quad i = 1,\ldots,n\,.$$

Below we will present part of the MATLAB interface. Again, the complete m-files can be found in directory `matlab/truss`. We have two variables in the problem—the matrix S (only involved in linear expressions) and the vector t. In MATLAB interface, vector variables should preceede linear matrix variables. We thus introduce a joint variable $x \in \mathbb{R}^{m+(n+1)(n+2)/2}$:

$$x^T = (t^T,\ \mathrm{svec}(S)^T)\,.$$

The relevant part (after the definition of the data matrices) of the file `truss.m` reads as

```
nnn = (n1+2)*(n1+1)/2; % n1 is the size of matrices A_i
Infinity = 1.0E38;
pen.nvars = m + nnn;
pen.nlin = 0;
pen.nconstr = nnn;
pen.nsdp = 1;
pen.blks = [n1+1];
pen.lbmv = [0];
pen.ubmv = [Infinity];
pen.mtype = [0];
pen.nnz_gradient = m+1;
pen.nnz_hessian = m;
pen.lbv = [0.0.*ones(m,1);-Infinity*ones(nnn,1)];
pen.ubv = [Infinity*ones(m,1);Infinity*ones(nnn,1)];
pen.lbc = [zeros((n1+1)*n1/2,1);ff;gamma];
pen.ubc = [zeros((n1+1)*n1/2,1);ff;gamma];
pen.xinit=[0.1.*ones(m,1);zeros(nnn,1)];
pen.my_f = 'f_truss';
pen.my_f_gradient = 'df_truss';
pen.my_f_hessian = 'hf_truss';
pen.my_g = 'g_truss';
pen.my_g_gradient = 'dg_truss';
pen.my_g_hessian = 'hg_truss';
pen.ioptions = [100 100 2 0 0 0 1 0 0 1 0 0 0 -1 0 1 2];
pen.doptions = [1.0E-2 1.0E0 1.0E-0 1.0E-2 5.0E-1 5.0E-1...
                  1.0E-6 1.0E-12 1.0e-7 0.05 1.0 1.0 1.0];
[w1,w2] = pennonm(pen);
```

We further show the files defining the objective function and constraints:

`f_truss`

```
function [fx] = f_truss(x)
global par
m=par.m;
fx = sum(x(1:m));
```

`g_truss`

```
function [gx] = g_truss(i, x)
global par A
m=par.m; n=par.n; n1=par.n1;
if i < (n1+1)*n1/2
    gx = 0;
    for j=1:m
        gx = gx - A{j}(i+1)*x(j);
    end
```

```
    gx = gx + x(i+1+m);
else
    gx = x(i+1+m);
end
```

6.4 Approximation by nonnegative splines

Consider the problem of approximating a one-dimensional function given only by a large amount of noisy measurments by a cubic spline. Additionally, we require that the function is nonnegative. This kind of problem arises in many application, for instance, in shape optimisation considering unilateral contact or in arrival rate approximation [2].
Assume that function $f : \mathbb{R} \to \mathbb{R}$ is defined on interval $[0, 1]$. We are given its function values b_i, $i = 1, \ldots, n$ at points $t_i \in (0, 1)$. We may further assume that the function values are subject to a random noise. We want to find a smooth approximation of f by a cubic spline, i.e., by a function of the form

$$P(t) = P^{(i)}(t) = \sum_{k=1}^{3} P(i)_k (t - a_{i-1})^k \tag{20}$$

for a point $t \in [a_{i-1}, a_i]$, where $0 = a_0 < a_1 < \ldots < a_m = 1$ are the knots and $P_k^{(i)} (i = 1, \ldots, m,\ k = 0, 1, 2, 3)$ the coefficients of the spline. The spline property that P should be continuous and have continuous first and second derivatives is expressed by the following equalities for $i = 1, \ldots, m - 1$:

$$P_0^{(i+1)} - P_0^{(i)} - P_1^{(i)}(a_i - a_{i-1}) - P_2^{(i)}(a_i - a_{i-1})^2 - P_3^{(i)}(a_i - a_{i-1})^3 = 0 \tag{21}$$

$$P_1^{(i+1)} - P_1^{(i)} - 2P_2^{(i)}(a_i - a_{i-1}) - 3P_3^{(i)}(a_i - a_{i-1})^2 = 0 \tag{22}$$

$$2P_2^{(i+1)} - 2P_2^{(i)} - 6P_3^{(i)}(a_i - a_{i-1}) = 0\,. \tag{23}$$

The function f will be approximated by P in the least square sense, so we want to minimize

$$\sum_{j=1}^{n} (P(t_j) - b_j)^2$$

subject to (21),(22),(23).
Now, the original function f is assumed to be nonnegative and we also want the approximation P to have this property. A simple way to guarantee nonnegativity of a spline is to express is using B-splines and consider only nonnegative B-spline coefficients. However, it was shown by de Boor and Daniel [3] that this may lead to a poor approximation of f. In particular, they showed that while approximation of a nonnegative function by nonnegative splines of order k gives errors of order h^k, approximation by a subclass of nonnegative splines of order k consisting of all those whose B-spline coefficients are nonnegative may yield only errors of order h^2. In order to get the best possible approximation, we use a result by Nesterov [6] saying that $P^{(i)}(t)$ from (20) is nonnegative if and only if there exist two symmetric matrices

$$X^{(i)} = \begin{pmatrix} x_i & y_i \\ y_i & z_i \end{pmatrix}, \qquad S^{(i)} = \begin{pmatrix} s_i & v_i \\ v_i & w_i \end{pmatrix}$$

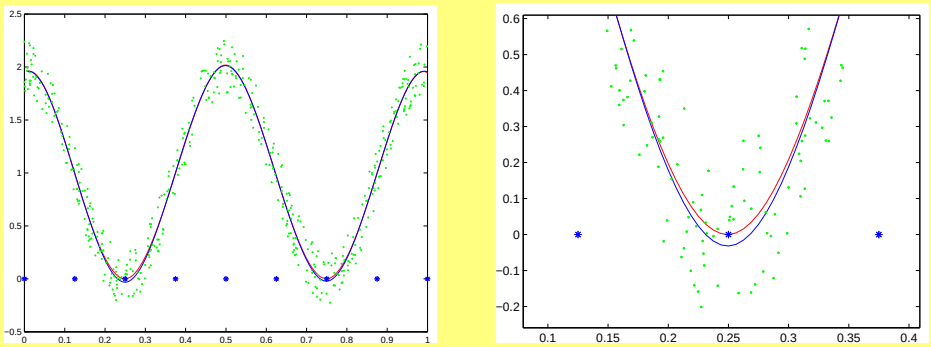

Figure 1: Approximation by nonnegative splines: noisy data given in green, optimal nonnegative spline in red and an optimal spline ignoring the nonnegativity constraint in blue. The right-hand side figure zooms on the left valey.

such that

$$P_0^{(i)} = (a_i - a_{i-1})s_i \tag{24}$$
$$P_1^{(i)} = x_i - s_i + 2(a_i - a_{i-1})v_i \tag{25}$$
$$P_2^{(i)} = 2y_i - 2v_i + (a_i - a_{i-1})w_i \tag{26}$$
$$P_3^{(i)} = z_i - w_i \tag{27}$$
$$X^{(i)} \succeq 0, \quad S^{(i)} \succeq 0\,. \tag{28}$$

Summarizing, we want to solve an NSDP problem

$$\min_{\substack{P_k^{(i)} \in \mathbb{R} \\ i=1,\dots,m,\ k=0,1,2,3}} \sum_{j=1}^{n} (P(t_j) - b_j)^2 \tag{29}$$

subject to

$$(21), (22), (23), \quad i = 1, \dots, m$$
$$(24) - (28), \quad i = 1, \dots, m$$

More complicated ("more nonlinear") objective functions can be obtained when considering, for instance, the the problem of approximating the arrival rate function of a nonhomogeneous Poisson process based on observed arrival data [2].
Again, we have programmed this problem in MATLAB and solved it by PENNON0.9. For instance, a problem of approximating a cosinus function given at 500 points by noisy data of the form `cos(4*pi*rand(500,1))+ 1 +.5.*rand(500,1)-.25` approximated by a nonnegative cubic spline with 7 knotes lead to an NSDP problem in 80 variables, 16 matrix variables, 16 matrix constraints, and 49 linear inequality constraints. The problem was solve by PENNON0.9 in about 1 second using 17 outer and 93 inner iterations. Figure 1 shows the result. The corresponding MATLAB files for this example are included in the directory `matlab/spline`.

References

[1] W. Achtziger and M. Kočvara. Structural Topology Optimization with Eigenvalues. *SIAM J. Optimization* 18(4): 1129–1164, 2007.

[2] F. Alizadeh, J. Eckstein, N. Noyan and G. Rudolf. Arrival Rate Approximation by Nonnegative Cubic Splines. *Operations Research* 56(1): 140–156, 2008.

[3] C. de Boor and J.W. Daniel. Splines with nonnegative B-spline coefficients. *Math. Comp.* 28: 565–568, 1974.

[4] M. Kočvara and M. Stingl. PENNON—a code for convex nonlinear and semidefinite programming. *Optimization Methods and Software*, 8(3):317–333, 2003.

[5] R. Fourer, D. M. Gay and B. W. Kernighan. AMPL—a modelling language for mathematical programming. *Scientific Press*, 1993.

[6] Yu. Nesterov. Squared functional systems and optimization problems. In H. Frenk, K. Roos, T. Terlaky and S. Zhang (eds.), High Performance Optimization. Dordrecht, Kluwer Academic Press, 405–440, 2000.

[7] R. Werner and K. Schöttle. Calibration or corellation matrices—SDP or not SDP. Submitted.