PENSDP User's Guide (Version 2.2)

Michal Kočvara Michael Stingl
PENOPT GbR
www.penopt.com

March 5, 2006

Contents

Changes to old versions

Changes to version 1.1

- Iterative solvers of the Newton equation added
- Optional approximate Hessian of the Augmented Lagrangian
- DIMACS error measures printed
- Changes in handling the parameter `ALPHA` with the goal to increase the final precision

Changes to version 2.0

- New option for Newton system solution added which, in connection with the CG method, requires no storage of the full Hessian (option `NWT_SYS_MODE=3`)
- Stopping criteria now based on the DIMACS error measures, resulting in more reliable results
- Minor bugs fixed

Changes to version 2.1

- Hybrid mode for the solution of the Newton system included

1 Installation

1.1 Unpacking

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

Win32 version
The distribution is packed in file `pensdp.zip`. Put this file in an arbitrary directory and extract the files by `PKZIP`.
In both cases, the directory PENSDP2.2 containing the following files and subdirectories will be created

LICENSE: a file containing the PENSDP license agreement;

bin: a directory containing the files

pensdp2.2(.exe), a binary executable with SDPA interface,
in.txt, an ASCII file with code parameters,
sdpa.dat, a sample problem in SDPA format;

c: a directory containing the files

driver_sdp_c.c, a sample driver implemented in C,
pensdp.h, a header file to included by C driver,
penout.c, sample implementation of Pensdp output functions;
make_{CC}.txt, a makefile to build a sample C program;

doc: a directory containing this User's Guide and a User's Guide for the MATLAB program `penfeas_lmi.m`;

fortran: a directory containing the files

driver_sdp_f.f, sample driver implemented in FORTRAN,
penout.c, see above;
penout.o(bj), precompiled version of `penout.c`;
make_{FC}.txt, a makefile to build a sample FORTRAN program;

libs: a directory containing the Pensdp library and Atlas libraries;

matlab: a directory containing the files

pensdpm.c, the MATLAB interface file,
penoutm.c, MATLAB version of `penout.c`,
make_pensdpm.m, M-file containing MEX link command,
lmi.m, M-file containing a sample problem in PEN format,
penfeas_lmi.m, checks feasibility of a system of LMIs;
sdpa2pen.m, a SDPA to PEN converter.

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

- `gcc` compiler package (C driver program)
- `g77` compiler package (FORTRAN driver program)
- MATLAB version 5.0 or later including MEX compiler package and `gcc` compiler package (MATLAB dynamic link library `pensdpm.*`)

Win32 version

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

To build a C driver program

UNIX versions

Go to directory `c` and invoke Makefile by command

```
make -f make_gcc.txt.
```

Win32 version

Go to directory `c` and invoke Makefile by command

```
nmake -f make_vc.txt.
```

To build a FORTRAN driver program

UNIX versions

Go to directory `fortran` and invoke Makefile by command

```
make -f make_g77.txt.
```

Win32 version

Go to directory `fortran` and invoke Makefile by command

```
nmake -f make_df.txt.
```

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

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

```
make_pensdpm.
```

In case the user wants to use his/her own LAPACK, BLAS or ATLAS implementations, the makefiles resp. M-files in directories `c`, `fortran` and `matlab` have to be modified appropriately.

2 The problem

We solve the SDP problem with linear matrix inequality constraints:

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

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`.

3 The algorithm

For a detailed description of the algorithm, see [2].
The algorithm is based on a choice of penalty/barrier function Φ_P that penalizes the inequality constraints. This function satisfies a number of properties that guarantee that for any $P > 0$, we have

$$\mathcal{A}(x) \preccurlyeq 0 \iff \Phi_P(\mathcal{A}(x)) \preccurlyeq 0\,.$$

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

$$\begin{aligned} \min_{x\in\mathbb{R}^n} & \sum_{k=1}^{n} f_k x_k \qquad (\text{SDP}_\Phi) \\ \text{s.t.} \quad & \Phi_P(\mathcal{A}(x)) \preccurlyeq 0\,. \end{aligned}$$

The Lagrangian of (SDP_Φ) can be viewed as a (generalized) augmented Lagrangian of (SDP):

$$F(x,U,P) = \sum_{k=1}^{n} f_k x_k + \langle U, \Phi_P\left(\mathcal{A}(x)\right)\rangle_{\mathbb{S}_d}\,; \qquad (1)$$

here d is the dimension of the matrix operator $\mathcal{A}$ and $U \in \mathbb{S}^d$ are Lagrangian multipliers associated with the inequality constraints.
The algorithm combines ideas of the (exterior) penalty and (interior) barrier methods with the Augmented Lagrangian method.

Algorithm 3.1 *Let x^1 and U^1 be given. Let $P^1 > 0$. For $k = 1, 2, \ldots$ repeat till a stopping criterium is reached:*

$$
\begin{aligned}
&(i) && \text{Find } x^{k+1} \text{ such that } \|\nabla_x F(x^{k+1}, U^k, P^k)\| \leq K \\
&(ii) && U^{k+1} = D_{\mathcal{A}} \Phi_p(\mathcal{A}(x); U^k) \\
&(iii) && P^{k+1} < P^k \, .
\end{aligned}
$$

The details of the implementation can be found in [2].

4 SDPA interface

The problem data are written in an ASCII input file in a SDPA sparse format, as introduced in [1].

4.1 Running `pensdp2.2`

- Go to directory `bin`.
- Prepare a data file with your problem data in SDPA sparse format.
- *Rename this file to* `sdpa.dat` and place it in directory `bin`.
- If required, change parameters in `in.txt` (see next page).
- From the command line, run `pensdp2.2` without any arguments.

Depending on the output options `XOUT` and `UOUT` in `in.txt`, the solution may be written in ASCII files `x.out` (primal) and `u.out` (dual) in the working directory.

4.2 SDPA input file

`sdpa.dat` is an ASCII file consisting of six parts:

1. Comments. The file can begin with arbitrarily many lines of comments. Each line of comments must begin with '`"`' or '`*`'.
2. The first line after the comments contains n, the number of variables (i.e., number of constraint matrices).
3. The second line after the comments contains m, the number of blocks in the block diagonal structure of the matrices.
4. The third line after the comments contains a vector of numbers that give the sizes of the individual blocks. The special characters '`,`', '`(`', '`)`', '`{`', and '`}`' can be used as punctuation and are ignored. Negative numbers may be used to indicate that a block is actually a diagonal submatrix. Thus a block size of -5 indicates a 5×5 block in which only the diagonal elements are nonzero.
5. The fourth line after the comments contains the objective function vector f.
6. The remaining lines of the file contain entries in the constraint matrices, with one entry per line. The format for each line is

   ```
   <matno> <blkno> <i> <j> <entry>
   ```

   Here `<matno>` is the number of the matrix to which this entry belongs, `<blkno>` specifies the block within this matrix, `<i>` and `<j>` specify a location within the block, and `<entry>` gives the value of the entry in the matrix. Note that since all matrices are assumed to be symmetric, only entries in the upper triangle of a matrix are given.

4.3 File `in.txt`

Below we give the parameter file `in.txt` with the default setting of parameters. Recommended setting of parameters for different situations are given in Section 7.

```
# ------------------------------------------------------------------------
# U0: multiplier parameter
#
# The initial vector of multipliers is computed by u0 = U0 * (1,1,...,1)
#
       1.0E0
# ------------------------------------------------------------------------
# P0: multiplier parameter
#
       1.0E0
# ------------------------------------------------------------------------
# MU: restrictions for multiplier update
#
# By the factor MU the multiplier update (5) is restricted from above and
# below in the following way
#               MU * U(k)  <  U(k+1)  <  (1 / MU) * U(k).
#
       7.0E-1
# ------------------------------------------------------------------------
# MU2: restrictions for matrix multiplier update
#
# By the factor MU the multiplier update (5) is restricted from above and
# below as
#        MU2 * lambda_min(U(k))  <  lambda_min(U(k+1)) and
#        lambda_max(U(k)) <  (1 / MU2) * lambda_max(U(k)).
#
       1.0E-1
# ------------------------------------------------------------------------
# PRECISION: Stopping criterion for outer iteration
#
# Depending on the problem type, the difference in objective values, the
# feasibility and the duality gap are taken into account.
#
       1.0E-4
# ------------------------------------------------------------------------
# P_EPS: Lower bound for penalty parameters.
#
       1.0E-7
# ------------------------------------------------------------------------
# U_EPS: Lower bound for norm of multipliers
#
       1.0E-14
# ------------------------------------------------------------------------
# ALPHA: Stop criterion for unconstrained minimization
#
# The unconstrained minimization (4) stops, if grad F(X,U,P) < ALPHA or
# the descent of the augmented Lagrangian in the last step is less than
# min(P_i)*ALPHA.
#
       1.0e-2
# ------------------------------------------------------------------------
# ALPHA_UP: update of ALPHA (ALPHA_NEW = ALPHA * ALPHA_UP)
#
       1.0
# ------------------------------------------------------------------------
#
# PRECISION2: Precision of KKT-conditions
#
       1.0E-7
#
 ------------------------------------------------------------------------
# MAX_OUTER_ITER: maximal outer iterations
#
```

```
# The global algorithm stops after maximal PBM_MAX_ITER iterations.
#
       50
# ------------------------------------------------------------------------
# MAX_INNER_ITER: maximal inner iterations
#
# The unconstrained minimization (4) stops after maximal UM_MAX_ITER
# iterations.
#
       100
# ------------------------------------------------------------------------
# OUTPUT: Outputlevel
#
# (0) Silent Mode: No output is written to the standard output.
# (1) Summary Mode: Only the final result is written to the standard
#     output.
# (2) Iteration Mode: After every outer iteration the status is written
#     to standard output.
# (3) Verbose Mode: After every iteration result the current status is
#     written to the standard output.
#
       2
#------------------------------------------------------------------------
# CHECKDENSITY:  sparse/dense data handling of Hessian
#
#          (0) Automatic
#          (1) Dense
#          (2) Sparse
#
       0
#
#------------------------------------------------------------------------
# USELS:  Use Linesearch ?
#
#          (0) No
#          (1) Yes
#
       0
#
#------------------------------------------------------------------------
# XOUT:  Write primal solution to file 'x.dat' ?
#
#          (0) No
#          (1) Yes
#
       0
#
#------------------------------------------------------------------------
# UOUT:  Write dual solution to file 'u.dat' ?
#
#          (0) No
#          (1) Yes
#
       0
#
#------------------------------------------------------------------------
# NWT_SYS_MODE: Use Conjugate Gradient approach instead of Cholesky for
# solution of linear systems ?
#
#          (0) No (Cholesky is used)
#          (1) CG with exact Hessian
#          (2) CG with approximated Hessian
#          (3) CG with exact Hessian not explicitly assembled
#
       3
#
#------------------------------------------------------------------------
```

```
# CG_TOL_DIR:  Stopping criterion for conjugate gradient algorithm
#
#
        5.0e-2
#
#-------------------------------------------------------------------------
# PRECTYPE:  Which preconditioner for CG ? (If USECG=2, only values 0-2
# are accepted)
#
#           (0) No
#           (1) diagonal
#           (2) LBFGS
#           (3) approximate inverse
#           (4) SGS
#
        2
#
#-------------------------------------------------------------------------
# DIMACS:  Print Dimacs error measures ?
#
#           (0) No
#           (1) Yes
#
        0
#
#-------------------------------------------------------------------------
# PENUP:   Penalty update ?
#
#           (0.)  Automatic (recommended)
#           (>0.) Manual
#
        0.0
#
#-------------------------------------------------------------------------
# HYBRIDMODE:  Switch to Cholesky if PCG causes troubles ?
#
#           (0) No
#           (1) Yes, use diagonal preconditioner after Cholesky step
#           (2) Yes, use inverse Cholesky precond. after Cholesky step
#
        2
#
#-------------------------------------------------------------------------
# AUTOINIT
#
#           (0) No
#           (1) Yes
#
        1
#
#-------------------------------------------------------------------------
# STOPMODE
#           (0) strict
#           (1) heuristic
#
        1
#
#-------------------------------------------------------------------------
```

Example 1. Let $f = (1, 2, 3)^T$. Assume that we have two linear matrix inequality constraints, first of size (3x3), second of size (2x2). The second constraint contains only diagonal matrices:

$$\left(\begin{array}{ccc|c} 0 & & & \\ & 0 & & \\ & & 0 & \\ \hline & & & \begin{matrix} 0 & \\ & 1 \end{matrix} \end{array}\right) + \left(\begin{array}{ccc|c} 2 & -1 & 0 & \\ & 2 & 0 & \\ & & 2 & \\ \hline & & & \begin{matrix} 1 & \\ & -1 \end{matrix} \end{array}\right) x_1$$
$$+ \left(\begin{array}{ccc|c} 0 & & & \\ & 0 & & \\ & & 0 & \\ \hline & & & \begin{matrix} 3 & \\ & -3 \end{matrix} \end{array}\right) x_2 + \left(\begin{array}{ccc|c} 2 & 0 & -1 & \\ & 2 & 0 & \\ & & 2 & \\ \hline & & & \begin{matrix} 0 & \\ & 0 \end{matrix} \end{array}\right) x_3$$

In SDPA sparse format, this problem can be written as:

```
"Example 1.
3
2
3 -2
1.0 2.0 3.0
0 2 2 2  1.0
1 1 1 1  2.0
1 1 2 2  2.0
1 1 3 3  2.0
1 1 1 2 -1.0
1 2 1 1  1.0
1 2 2 2 -1.0
2 2 1 1  3.0
2 2 2 2 -3.0
3 1 1 1  2.0
3 1 2 2  2.0
3 1 3 3  2.0
3 1 1 3 -1.0
```

5 C/C++/FORTRAN interface

PENSDP can also be called as a function (or subroutine) from a C or FORTRAN program. In this case, the user should link the PENSDP library to his/her program.

5.1 Calling PENSDP from a C/C++ program

For the implementation of a C interface the user should perform the following steps:

First, the user must include the PENSDP header file as in the following line:

```
#include "pensdp.h"
```

Second, the variables for the problem data have to be declared as in the following piece of code:

```
/* basic problem dimensions */
int vars = 0, constr = 0, mconstr = 0;
int luoutput = 0, lbi = 0, lai = 0, nzsai = 0;

/* error flag */
int inform = 0;

/* function value */
double fX = 0.;

/* array reserved for block sizes of matrix constraints */
int *msizes = 0;

/* array reserved for initial iterate (input)
   and optimal primal variables (output) */
double *x0 = 0;

/* array reserved for optimal dual variables (output) */
double *uoutput = 0;

/* objective */
double *fobj = 0;

/* rhs of linear constraints */
double *ci = 0;

/* arrays for linear constraints */
int *bi_dim = 0, *bi_idx = 0;
double *bi_val = 0;

/* arrays for matrix constraints */
int *ai_dim = 0, *ai_idx = 0, *ai_nzs = 0,
    *ai_col = 0, *ai_row = 0;
double *ai_val = 0;

/* arrays reserved for results */
int iresults[4] = {0,0,0,0};
double fresults[3] = {0.0,0.0,0.0};
```

```
/* default options */
int ioptions[15] = {1,50,100,2,0,1,0,0,0,0,0,0,0,1,1};
double foptions[12] = {1.0,0.7,0.1,1.0E-7,
                       1.0E-6,1.0E-14,1.0E-2,1.0e0,
                       0.0,1.0,1.0E-7,5.0E-2};
```

Third, the user should specify the problem dimensions by assigning values to variables

```
vars, mconstr, constr,lbi, lai, nzsai, luoutput.
```

Using these numbers, the user should allocate memory as it is shown below:

```
msizes = INTEGER (mconstr);
x0 = DOUBLE(vars);
uoutput = DOUBLE(luoutput);
fobj = DOUBLE(vars);

if(constr) {
  ci = DOUBLE(constr);
  bi_dim = INTEGER(constr);
  bi_idx = INTEGER(lbi);
  bi_val = DOUBLE(lbi);
}

if(mconstr) {
  ai_dim = INTEGER(mconstr);
  ai_idx = INTEGER(lai);
  ai_nzs = INTEGER(lai);
  ai_col = INTEGER(nzsai);
  ai_row = INTEGER(nzsai);
  ai_val = DOUBLE(nzsai);
}
```

Next, the problem data should be assigned to arrays

```
x0, fobj, ci, x0, bi_dim, bi_idx, bi_val,
ai_dim, ai_idx, ai_nzs, ai_val, ai_col, ai_row
```

and some non default options could be set by changing entries in the arrays

```
ioptions, foptions.
```

The meaning of the input/output parameters and the options are explaned in detail in section 5.3. Finally, the user should call PENSDP like

```
/* Call pensdp */
pensdp(vars, constr, mconstr, msizes,
  &fX, x0, 0, uoutput, fobj, ci,
  bi_dim, bi_idx, bi_val,
  ai_dim, ai_idx, ai_nzs,
  ai_val, ai_col, ai_row,
  ioptions, foptions,
  iresults, fresults, &inform);
```

and free memory.

A sample implementation is included in the file `driver_sdp_c.c` in directory `c`.

5.2 Calling PENSDP from a FORTRAN program

Given the (upper bounds on) values

```
vars1, mconstr1, constr1
```

of

```
vars, mconstr, constr
```

and dimensions

```
lbi, lai, nzsai, luoutput
```

of

```
bi_idx, ai_idx, ai_col, uoutput
```

(either from outer call or declared as parameters), the user can call subroutine `pensdpf` as in the following piece of code:

```
      integer vars, constr, mconstr, msizes(mconstr1)
      integer bi_dim(constr1), bi_idx(lbi)
      integer ai_dim(mconstr1), ai_idx(lai), ai_nzs(lai)
      integer ai_col(nzsai), ai_row(nzsai)
      integer ioptions(15), iresults(4), info

      real*8 fx, x0(vars1), uoutput(luoutput), fobj, ci(constr1)
      real*8 bi_val(lbi), ai_val(nzsai), foptions(12), fresults(5)
      ...
      ...
      call pensdpf (vars, constr, mconstr, msizes, fx,
     *            x0, 0, uoutput, fobj, ci,
     *            bi_dim, bi_idx, bi_val,
     *            ai_dim, ai_idx, ai_nzs,
     *            ai_val, ai_col, ai_row,
     *            ioptions, foptions,
     *            iresults, fresults, info)
```

The input/output parameters are explained below. A sample implementation is included in the file `driver_sdp_f.f` in directory `fortran`.

5.3 The input/output structure

We assume that the linear constraint vectors b^i can be sparse, so we give them in standard sparse format. We further assume that the matrix constraints data can be sparse. Here we distinguish two cases: Some 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. Each of the nonempty matrices A_k^i can still be sparse; hence they are given in sparse format (value, column index, row index). As all the matrices are symmetric, we only give the upper triangle. *All index arrays (i.e., *.idx, *.col, and *.row) are zero-based, as in the C language.*

`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 $A^i_0, A^i_1, \ldots, A^i_{\texttt{vars}}$ 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 $A^i_{\texttt{ai\_idx(1)}}, A^i_{\texttt{ai\_idx(2)}}, \cdots, A^i_{\texttt{ai\_idx(ai\_dim(i))}}$ 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 $A^i_{\texttt{ai\_idx(1)}}, A^i_{\texttt{ai\_idx(2)}}, \cdots, A^i_{\texttt{ai\_idx(ai\_dim(i))}}$ 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))

`ioptions`	integer valued options (see below)
integer array	*length*: 12
`foptions`	real valued options (see below)
double array	*length*: 12
`iresults`	on exit: integer valued output information (see below) (Not referenced if `iresults = 0` on entry)
integer array	*length*: 4
`fresults`	on exit: real valued output information (see below) (Not referenced if `fresults = 0` on entry)
double array	*length*: 3
`status`	on exit: error flag (see below)
integer array	*length*: 1

6 MATLAB interface

6.1 Calling PENSDPM from MATLAB

In MATLAB, PENSDPM is called with the following arguments:

```
[f,x,u,iflag,niter,feas] = pensdpm(pen);
```

where

pen . . . the input structure described in the next 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

iflag . . . exit information

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

niter . . . a 4x1 matrix with elements

niter(1) . . . number of outer iterations
niter(2) . . . number of Newton steps
niter(3) . . . number of linesearch steps
niter(4) . . . ellapsed time in seconds

feas . . . a 3x1 matrix with stopping criteria values

feas(1) . . . relative precision at $x_{\rm opt}$
feas(2) . . . feasibility of linear inequalities at $x_{\rm opt}$
feas(3) . . . feasibility of matrix inequalities at $x_{\rm opt}$
feas(4) . . . complementary sleckness of linear inequalities at $x_{\rm opt}$
feas(5) . . . complementary sleckness of matrix inequalities at $x_{\rm opt}$

IOPTIONS	name/value	meaning	default
ioption(0)	DEF 0 1	 use default values for all options use user defined values	
ioption(1)	MAX_OUTER_ITER	maximum number of iterations of the overall algorithm	50
ioption(2)	MAX_INNER_ITER	maximum number of iterations for the unconstrained minimization	100
ioption(3)	OUTPUT 0 1 2 3	output level no output summary output brief output full output	1
ioption(4)	DENSE 0 1 2	check density of the Hessian automatic check. For very large problems with dense Hessian, this may lead to memory difficulties. dense Hessian assumed sparse Hessian assumed	0
ioption(5)	LS 0 1	linesearch in unconstrained minimization do not use linesearch use linesearch	0
ioption(6)	XOUT 0 1	write solution vector x in the output file no yes	0
ioption(7)	UOUT 0 1	write computed multipliers in the output file no yes	0
ioption(8)	NWT_SYS_MODE 0 1 2 3	mode of solution of the Newton system Cholesky method preconditioned conjugate gradient method preconditioned CG method with approximate Hessian calculation preconditioned CG method with exact Hessian not explicitly assembled	0
ioption(9)	PREC_TYPE 0 1 2 3 4	preconditioner type for the CG method no preconditioner diagonal LBFGS appoximate inverse symmetric Gauss-Seidel	0
ioption(10)	DIMACS 0 1	print DIMACS error measures no yes	0
ioption(11)	TR_MODE 0 1	Trust-Region mode modified Newton method Trust Region method	0
ioption(12)	HYBRIDMODE 0 1 2	Switch to Cholesky if PCG causes troubles No Yes, use diagonal preconditioner after Cholesky step Yes, use inverse Cholesky precond. after Cholesky step	0
ioption(13)	AUTOINIT 0 1	 No Yes	1
ioption(14)	STOPMODE 0 1	 strict heuristic	1

FOPTIONS	name/value	meaning	default
foptions(0)	U0	scaling factor for linear constraints; must be positive	1.0
foptions(1)	MU	restriction for multiplier update for linear constraints	0.7
foptions(2)	MU2	restriction for multiplier update for matrix constraints	0.1
foptions(3)	PRECISION	stopping criterium for the overall algorithm	1.0e-7
foptions(4)	P_EPS	lower bound for the penalty parameters	1.0e-6
foptions(5)	U_EPS	lower bound for the multipliers	1.0e-14
foptions(6)	ALPHA	stopping criterium for unconstrained minimization	1.0e-2
foptions(7)	P0	initial penalty value; set it lower (0.01–0.1) to maintain feasibility when starting from a feasible point.	1.0
foptions(8)	PEN_UP	penalty update; when set to 0.0, it is computed automatically	0.0
foptions(9)	ALPHA_UP	update of α; should either be equal to 1.0 (= no update) or smaller than 1.0 (e.g. 0.7)	1.0
foptions(10)	PRECISION_2	precision of the KKT conditions	1.0e-7
foptions(11)	CG_TOL_DIR	stopping criterion for the conjugate gradient algorithm	5.0e-2

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

FRESULTS	meaning
fresults(0)	relative precision at $x_{\rm opt}$
fresults(1)	feasibility of linear inequalities at $x_{\rm opt}$
fresults(2)	feasibility of matrix inequalities at $x_{\rm opt}$
fresults(3)	complementary slackness of linear inequalities at $x_{\rm opt}$
fresults(4)	complementary slackness of matrix inequalities at $x_{\rm opt}$

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

6.2 `pen` input structure in MATLAB

The user must create a MATLAB structure array with fields described in Section 5.3.

Example 2. 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` = [0.0,0.0,0.8]; (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,$\ldots\ldots$, 0,1,0,1,0,1,0,1];
`ai_row` = [0,0,0,1,1,2, 0,0,0,1,1,2,$\ldots\ldots$, 0,1,0,1,0,1,0,1];

Example 3. 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:

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

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

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];

7 General recommendations

7.1 Dense versus Sparse

For the efficiency of PENSDP, it is important to know if the problem has sparse or dense Hessian. The program can check this automatically. The check may take some time and memory, so if you know that the Hessian is dense (and this is the case of most problems), you can avoid the check. The check is switched on and off by parameter `CHECKDENSITY` in 'in.txt' (SDPA version) or by option `DENSE` in `ioptions` (MATLAB version). The default is 'on' (do the check).

7.2 High accuracy

The default values of the parameters in file `in.txt` are set to achieve relatively high accuracy in the primal and dual solutions. The program first tries to reach `PRECISION` and then switches to `PRECISION2`. Parameter `PRECISION2` directly controls the DIMACS error criteria: setting, e.g.,

`PRECISION2` to 1.0e-7

means that all DIMACS criteria will be equal to or smaller than 10^{-7}, if the code finishes successfully. The DIMACS criteria measure the 1st-order optimality, primal and dual feasibility and complementary slackness.
If you require lower accuracy, just set

`PRECISION` to 1.0e-4

and

`PRECISION2` to, e.g., 1.0,

so that it is ignored. For higher accuracy, set

`PRECISION` to 1.0e-4 – 1.0e-6

and

`PRECISION2` to the required DIMACS accuracy.

The choice of too high accuracy may lead to an inefficient code.

7.3 Cholesky versus iterative solvers

When computing the search direction in the Newton method, we have to solve a system of linear equations with a symmetric positive definite matrix (Hessian of the augmented Lagrangian). This system can either be solved by (possibly sparse) Cholesky decomposition or by the preconditioned conjugate gradient method. The choice is done by parameter

`NWT_SYS_MODE`

When this parameter is set to 1, 2 or 3, CG method is used in connection with a preconditioner chosen by parameter

`PRECTYPE`

We recommend to use CG when *the number of variables is large and, in particular, when additionally the size of the matrix constraint is relatively small*. While PENSDP with CG can reach the same high accuracy as with Cholesky decomposition, the code becomes much more efficient when the required accuracy is decreased by setting

`PRECISION2` to 1e-2.

The choice of a preconditioner depends to a large extent on the problem. We recommend to try several preconditioners, in the order 2–4–1–3–0.

Warning: For certain kind of problems, the choice of CG method may lead to a rather inefficient code, compared to Cholesky decomposition.

New in version 2.2: We have implemented a hybrid mode for the solution of the Newton system. When the option

`HYBRIDMODE` is set to 2

and

NWT_SYS_MODE is set to 3

the code starts to solve the system using the CG method. When the number of CG steps becomes too large, the codes switches to the Cholesky solver to solve the current system. For the next system, the code tries again CG method with the Cholesky factor from the previous system as a preconditioner. This mode is only active when the number of variables is significantly larger than the size of the matrix constraint, otherwise the code switches to the Cholesky solver. *We recommend to use this mode as default mode for general problems.*

7.4 Exact versus approximate Hessian

The use of the CG method allows us to use an exact or an approximate formula for the Hessian-vector product. This may be particularly efficient when the Hessian evaluation is expensive. Further, as the Hessian does not have to be stored, *the memory requirements of the approximate version are much smaller*. The option is chosen by setting

NWT_SYS_MODE to 2 (approximate Hessian-vector product)

NWT_SYS_MODE to 3 (exact Hessian-vector product)

The use of exact Hessian-vector product (option 3) is generally preferable.

As above, to increase the efficiency, we recommend to set

PRECISION2 to 1e-2.

For

NWT_SYS_MODE set to 2 or 3,

only

PRECTYPE set to 0 or 2 is allowed.

Warning: For certain kind of problems, the choice of CG method with approximate Hessian may lead to a rather inefficient code, compared to Cholesky decomposition with exact Hessian.

References

[1] K. Fujisawa, M. Kojima, and K. Nakata. SDPA (Semidefinite Programming Algorithm) User's Manual. Technical Report B-308, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology. Revised, May, 1998.

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