Research Reports on Mathematical and Computing Sciences
Series B : Operations Research

Department of Mathematical and Computing Sciences
Tokyo Institute of Technology
2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-0033, Japan

SDPA (SemiDefinite Programming Algorithm) User's Manual — Version 5.01

Katsuki Fujisawa†, Masakazu Kojima⋆, Kazuhide Nakata♯

August 2000

Abstract. The SDPA (SemiDefinite Programming Algorithm) is a software package for solving semidefinite program (SDP). It is based on a Mehrotra-type predictor-corrector infeasible primal-dual interior-point method. The SDPA handles the standard form SDP and its dual. It is implemented in C++ language utilizing the *Meschach* [9] for matrix computation. The SDPA incorporates dynamic memory allocation and deallocation. So, the maximum size of an SDP to be solved depends on the size of computational memory which user's computer loads. The SDPA version 5.01 enjoys the following features:

- Callable library of the SDPA is available.
- Three types of search directions are available; the one proposed by [4, 5] (see also [7]), the one by [8, 10], and the one by [1, 2].
- Efficient method for computing the search directions when an SDP to be solved is large scale and sparse [3].
- Block diagonal matrix structure and sparse matrix structure in data matrices are available.
- Mehrotra-type predictor-corrector [1, 2, 10] is incorporated.
- Some information on infeasibility of a semidefinite program to be solved is provided.

This manual and the SDPA can be found in the directory

ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA/

Read the file "README" there for more details on how to get the SDPA.

Key words Semidefinite Programming, Interior-Point Method, Computer Software

† e-mail:fujisawa@is-mj.archi.kyoto-u.ac.jp

⋆ e-mail:kojima@is.titech.ac.jp

♯ e-mail:nakata@momonga.t.u-tokyo.ac.jp

Contents

1. Installation.

The SDPA package is available at the following ftp site:

ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA/

After reading the **README** file one can click on **5.00** to enter into a directory of the latest version of the SDPA and down-load a file appropriate to your platfome. Here, we assume that this appropriate platform is **SPARC Station (Solaris 2.x) egcs version**. The down-loaded file (**SOLARIS_egcs1.1.2.tar.gz**) will be moved in a suitable directory. To resolve this file, execute the following procedures:

tar xvzf SOLARIS_egcs1.1.2.tar.gz

The tar command will create the subdirectory **sdpaSUN5** in which one can find the following files:

sdpa_doc.ps	An user's manual.
sdpa	An executable binary, which solves the SDPs.
param.sdpa	A parameter file, which contains 9 parameters to control the SDPA.
example1.dat, example2.dat	Sample input files in the dense data format.
example1.dat-s	A sample input file in the sparse data format.
example1.ini	An initial point file in the dense data format.
example1.ini-s	An initial point file in the sparse data format.
sdpa.a	A callable library of the SDPA.
meschach.a	A meschach library for matrix computations.
Makefile	A makefile to compile sample source files.
example1-1.cpp, example1-2.cpp	
example2-1.cpp, example2-2.cpp	
example3.cpp, example4.cpp	
example5.cpp	Sample source files for utilizing the callable library.
sdpa-lib.hpp, sdpa-lib2.hpp,	
matrix.h, matrix2.h,	
sparse.h, sparse2.h	Header files for utilizing the callable library **sdpa.a**.

Here, check whether the **sdpa** file has a execute permission. If the **sdpa** has not a execute permission, type

% **chmod +x sdpa**

Before using sdpa, type **sdpa** and make sure that the following message will be displayed.

```
% sdpa

************************************************
*                                              *
*               SDPA version 5.00              *
*        (C) :  Kyoto University,              *
*        Tokyo Institute of Technology., 1999  *
*                                              *
************************************************
```

```
Usage            : sdpa  DataFile  OutputFile  [-c] [InitialPtFile] [SearchDirection]
 -c              : check whether input matrices are symmetric.
SearchDirection  : -1 (HRVW/KSH/M), -2 (NT) or -3 (AHO)
%
```

We also recommend executing the **ranlib** command before using the callable library.

% **ranlib meschach.a**
% **ranlib sdpa.a**

2. Semidefinite Program.

2.1. Standard Form SDP and Its Dual.

The SDPA(Semidefinite Programming Algorithm) solves the following standard form semidefinite program and its dual. Here

$$\text{SDP} \quad \begin{cases} \mathcal{P}: & \text{minimize} & \displaystyle\sum_{i=1}^{m} c_i x_i \\ & \text{subject to} & \boldsymbol{X} = \displaystyle\sum_{i=1}^{m} \boldsymbol{F}_i x_i - \boldsymbol{F}_0, \quad \boldsymbol{X} \succeq \boldsymbol{O}, \\ & & \boldsymbol{X} \in \mathcal{S}. \\ \mathcal{D}: & \text{maximize} & \boldsymbol{F}_0 \bullet \boldsymbol{Y} \\ & \text{subject to} & \boldsymbol{F}_i \bullet \boldsymbol{Y} = c_i \ (i = 1, 2, \ldots, m), \quad \boldsymbol{Y} \succeq \boldsymbol{O}. \\ & & \boldsymbol{Y} \in \mathcal{S}, \end{cases}$$

$\mathcal{S}$: the set of $n \times n$ real symmetric matrices.

$\boldsymbol{F}_i \in \mathcal{S}$ $(i = 0, 1, 2, \ldots, m)$: constraint matrices.

$\boldsymbol{O} \in \mathcal{S}$: the zero matrix.

$\boldsymbol{c} = \begin{pmatrix} c_1 \\ c_2 \\ \cdot \\ c_m \end{pmatrix} \in R^m$: a cost vector, $\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ x_m \end{pmatrix} \in R^m$: a variable vector,

$\boldsymbol{X} \in \mathcal{S}$, $\boldsymbol{Y} \in \mathcal{S}$: variable matrices,

$\boldsymbol{U} \bullet \boldsymbol{V}$: the inner product of $\boldsymbol{U}$, $\boldsymbol{V} \in \mathcal{S}$, *i.e.*, $\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} U_{ij} V_{ij}$

$\boldsymbol{U} \succeq \boldsymbol{O}$, $\Longleftrightarrow$ $\boldsymbol{U} \in \mathcal{S}$ is positive semidefinite.

Throughout this manual, we denote the primal-dual pair of $\mathcal{P}$ and $\mathcal{D}$ by the SDP. The SDP is determined by m, n, $\boldsymbol{c} \in R^m$, $\boldsymbol{F}_i \in \mathcal{S}$ $(i = 0, 1, 2, \ldots, m)$. When $(\boldsymbol{x}, \boldsymbol{X})$ is a feasible solution (or a minimum solution, resp.) of the primal problem $\mathcal{P}$ and $\boldsymbol{Y}$ is a feasible solution (or a maximum solution, resp.), we call $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ a feasible solution (or an optimal solution, resp.) of the SDP.

We assume:

Condition 1.1. $\{\boldsymbol{F}_i \ : \ i = 1, 2, \ldots, m\} \subset \mathcal{S}$ is linearly independent.

If the SDP did not satisfy this assumption, it might cause some trouble (numerical instability) that would abnormally stop the execution of the SDPA.

If we deal with a different primal-dual pair of $\mathcal{P}$ and $\mathcal{D}$ of the form

$$\text{SDP'} \left\{ \begin{array}{lll} \mathcal{P}: & \text{minimize} & \boldsymbol{A}_0 \bullet \boldsymbol{X} \\ & \text{subject to} & \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i \ (i = 1, 2, \ldots, m), \quad \boldsymbol{X} \succeq \boldsymbol{O}, \\ & & \boldsymbol{X} \in \mathcal{S}. \\ \mathcal{D}: & \text{maximize} & \displaystyle\sum_{i=1}^{m} b_i y_i \\ & \text{subject to} & \displaystyle\sum_{i=1}^{m} \boldsymbol{A}_i y_i + \boldsymbol{Z} = \boldsymbol{A}_0, \quad \boldsymbol{Z} \succeq \boldsymbol{O}, \\ & & \boldsymbol{Z} \in \mathcal{S}. \end{array} \right.$$

we can easily transform from the SDP' into the SDP as follows:

$$\begin{array}{rcl} -\boldsymbol{A}_i (i = 0, \ldots, m) & \longrightarrow & \boldsymbol{F}_i (i = 0, \ldots, m) \\ -a_i (i = 1, \ldots, m) & \longrightarrow & c_i (i = 1, \ldots, m) \\ \boldsymbol{X} & \longrightarrow & \boldsymbol{Y} \\ \boldsymbol{y} & \longrightarrow & \boldsymbol{x} \\ \boldsymbol{Z} & \longrightarrow & \boldsymbol{X} \end{array}$$

2.2. Example 1.

$$\left. \begin{array}{lll} \mathcal{P}: & \text{minimize} & 48y_1 - 8y_2 + 20y_3 \\ & \text{subject to} & \boldsymbol{X} = \begin{pmatrix} 10 & 4 \\ 4 & 0 \end{pmatrix} y_1 + \begin{pmatrix} 0 & 0 \\ 0 & -8 \end{pmatrix} y_2 + \begin{pmatrix} 0 & -8 \\ -8 & -2 \end{pmatrix} y_3 - \begin{pmatrix} -11 & 0 \\ 0 & 23 \end{pmatrix} \\ & & \boldsymbol{X} \succeq \boldsymbol{O}. \\ \mathcal{D}: & \text{maximize} & \begin{pmatrix} -11 & 0 \\ 0 & 23 \end{pmatrix} \bullet \boldsymbol{Y} \\ & \text{subject to} & \begin{pmatrix} 10 & 4 \\ 4 & 0 \end{pmatrix} \bullet \boldsymbol{Y} = 48, \ \begin{pmatrix} 0 & 0 \\ 0 & -8 \end{pmatrix} \bullet \boldsymbol{Y} = -8 \\ & & \begin{pmatrix} 0 & -8 \\ -8 & -2 \end{pmatrix} \bullet \boldsymbol{Y} = 20, \ \boldsymbol{Y} \succeq \boldsymbol{O}. \end{array} \right\}$$

Here

$$\begin{aligned} m &= 3, \ n = 2, \ \boldsymbol{c} = \begin{pmatrix} 48 \\ -8 \\ 20 \end{pmatrix}, \ \boldsymbol{F}_0 = \begin{pmatrix} -11 & 0 \\ 0 & 23 \end{pmatrix}, \\ \boldsymbol{F}_1 &= \begin{pmatrix} 10 & 4 \\ 4 & 0 \end{pmatrix}, \ \boldsymbol{F}_2 = \begin{pmatrix} 0 & 0 \\ 0 & -8 \end{pmatrix}, \ \boldsymbol{F}_3 = \begin{pmatrix} 0 & -8 \\ -8 & -2 \end{pmatrix}. \end{aligned}$$

The data (see Section 4.1.) of this problem is contained in the file "example1.dat".

2.3. Example 2.

$$m = 5,\ n = 7,\ \boldsymbol{c} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix} = \begin{pmatrix} 1.1 \\ -10 \\ 6.6 \\ 19 \\ 4.1 \end{pmatrix},$$

$$\boldsymbol{F}_0 = \begin{pmatrix} -1.4 & -3.2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -3.2 & -28 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 15 & -12 & 2.1 & 0.0 & 0.0 \\ 0.0 & 0.0 & -12 & 16 & -3.8 & 0.0 & 0.0 \\ 0.0 & 0.0 & 2.1 & -3.8 & 15 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.8 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -4.0 \end{pmatrix},$$

$$\boldsymbol{F}_1 = \begin{pmatrix} 0.5 & 5.2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 5.2 & -5.3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 7.8 & -2.4 & 6.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -2.4 & 4.2 & 6.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 6.0 & 6.5 & 2.1 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -4.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -3.5 \end{pmatrix}$$

$$\vdots$$

$$\boldsymbol{F}_5 = \begin{pmatrix} -6.5 & -5.4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -5.4 & -6.6 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 6.7 & -7.2 & -3.6 & 0.0 & 0.0 \\ 0.0 & 0.0 & -7.2 & 7.3 & -3.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -3.6 & -3.0 & -1.4 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 6.1 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.5 \end{pmatrix}.$$

As shown in this example, the SDPA handles block diagonal matrices. The data (see Section 4.2.) of this example is contained in the file "example2.dat".

3. Files Necessary to Execute the SDPA.

We need the following files to execute the SDPA

- "**sdpa**" — An executable binary for solving an SDP.
- "an input data file" — Any file name with the postfix "**.dat**" or "**.dat-s**" is possible; for example, "problem.dat" and "example.dat-s" are legitimate names for input files. However, the SDPA distinguishes a dense input data file with the postfix "**.dat**" from a sparse input data file with the postfix "**.dat-s**". See Section 4. and 7.4. for details.

- "**param.sdpa**" — A file describing the parameters used in the "sdpa". See Section 4 for details. The name is fixed to "**param.sdpa**".
- "an output file" — Any file name except "**sdpa**" and "**param.sdpa**". For example, "problem.1" and "example.out" are legitimate names for output files. See Section 7. for more details.

The files "example1.dat" (see Section 3.1) and "example2.dat" (see Section 3.2) contain the input date of Example 1 and Example 2, respectively, which we have stated in the previous section. To solve Example 1, type

% sdpa example1.dat example1.out

Here "example1.out" denotes "an output file" in which the SDPA stores computational results such as an approximate optimal solution, an approximate optimal value of Example 1, etc.. Similarly we can solve Example 2 by using the "sdpa".

4. Input Date File.

4.1. "example1.dat" — Input Data File of Example 1.

```
"Example 1: mDim = 3, nBLOCK = 1, {2}"
   3  =  mDIM
   1  =  nBLOCK
   2  = bLOCKsTRUCT
{48, -8, 20}
{ {-11,  0}, { 0, 23} }
{ { 10,  4}, { 4,  0} }
{ {  0,  0}, { 0, -8} }
{ {  0, -8}, {-8, -2} }
```

4.2. "example2.dat" — Input Data File of Example 2.

```
*Example 2:
*mDim = 5, nBLOCK = 3, {2,3,-2}"
   5  =  mDIM
   3  =  nBLOCK
   (2, 3, -2)   = bLOCKsTRUCT
{1.1, -10, 6.6, 19, 4.1}
{
{ { -1.4, -3.2 },
  { -3.2,-28   }   }
{ { 15,  -12,    2.1 },
  {-12,   16,   -3.8 },
  {  2.1, -3.8, 15   }   }
  {  1.8, -4.0 }
}
```

```
{
{ {  0.5,  5.2 },
  {  5.2, -5.3 }   }
{ {  7.8, -2.4,  6.0 },
  { -2.4,  4.2,  6.5 },
  {  6.0,  6.5,  2.1 }   }
  { -4.5, -3.5 }
}
```

•
•
•

```
{
{ { -6.5, -5.4 },
  { -5.4, -6.6 }   }
{ {  6.7, -7.2, -3.6 },
  { -7.2,  7.3, -3.0 },
  { -3.6, -3.0, -1.4 }   }
  {  6.1, -1.5 }
}
```

4.3. Format of Input Data File.

In general, the structure of an input data file is as follows:

Title and Comment
m — the number of the primal variables x_i's
nBLOCK — the number of blocks
bLOCKsTRUCT — the block structure vector
$\boldsymbol{c}$
$\boldsymbol{F}_0$
$\boldsymbol{F}_1$
.
.
$\boldsymbol{F}_m$

In Sections 4.4. through 4.8. , we explain each item of the input data file in details.

4.4. Title and Comment.

On top of the input data file, we can write a single or multiple lines of Title and Comment. Each line of Title and Comment must begin with " or $\star$ and consist of no more than 75 letters; for example

```
"Example 1: mDim = 3, nBLOCK = 1, {2}"
```

in the file "example1.dat", and

```
*Example 2:
*mDim = 5, nBLOCK = 3, {2,3,-2}
```

in the file "example2.dat". The SDPA displays Title and Comment when it starts. Title and Comment can be omitted.

4.5. The Number of the Primal Variables.

We write the number m of the primal variables in a line following the line(s) of Title and Comment in the input data file. All the letters after m through the end of the line are neglected. We have

```
3  =  mDIM
```

in the file "example1.dat", and

```
5  =  mDIM
```

in the file "example2.dat". In either case, the letters "= mDIM" are neglected.

4.6. The Number of the Blocks and the Block Structure Vector.

The SDPA handles block diagonal matrices as we have seen in Section 1.3. In terms of the number of blocks, denoted by nBLOCK, and the block structure vector, denoted by bLOCKsTRUCT, we express a common matrix data structure for the constraint matrices $\boldsymbol{F}_0,\ \boldsymbol{F}_1,\ \ldots,\ \boldsymbol{F}_m$. If we deal with a block diagonal matrix $\boldsymbol{F}$ of the form

$$\left.\begin{array}{rcl} \boldsymbol{F} & = & \begin{pmatrix} \boldsymbol{B}_1 & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{B}_2 & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \cdot & \cdot & \cdot & \cdots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{B}_\ell \end{pmatrix}, \\ \boldsymbol{B}_i & : & \text{a } p_i \times p_i \text{ symmetric matrix } \ (i = 1, 2, \ldots, \ell), \end{array}\right\} \tag{1}$$

we define the number nBLOCK of blocks and the block structure vector bLOCKsTRCTURE as follows:

$$\begin{aligned} \text{nBLOCK} &= \ell, \\ \text{bLOCKsTRUCT} &= (\beta_1,\ \beta_2,\ \ldots,\ \beta_\ell), \\ \beta_i &= \begin{cases} p_i & \text{if } \boldsymbol{B}_i \text{ is a symmetric matrix,} \\ -p_i & \text{if } \boldsymbol{B}_i \text{ is a diagonal matrix.} \end{cases} \end{aligned}$$

For example, if $\boldsymbol{F}$ is of the form

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 \\ 2 & 4 & 5 & 0 & 0 & 0 & 0 \\ 3 & 5 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 \end{pmatrix}, \tag{2}$$

we have

$$\text{nBLOCK} = 3 \quad \text{and} \quad \text{bLOCKsTRUCT} = \quad (3,\ 2,\ -2)$$

If

$$\boldsymbol{F} = \begin{pmatrix} \star & \star & \star \\ \star & \star & \star \\ \star & \star & \star \end{pmatrix}, \quad \text{where } \star \text{ denotes a real number,}$$

is a usual symmetric matrix with no block diagonal structure, we define

$$\text{nBLOCK} = 1 \quad \text{and} \quad \text{bLOCKsTRUCT} = 3$$

We separately write each of nBLOCK and bLOCKsTRUCT in one line. Any letter after either of nBLOCK and bLOCKsTRUCT through the end of the line is neglected. In addition to blank letter(s), and the tab code(s), we can use the letters

$$, \quad (\quad) \quad \{ \quad \}$$

to separate elements of the block structure vector bLOCKsTRUCT. We have

```
1  =  nBLOCK
2  = bLOCKsTRUCT
```

in Example 1 (see the file "example1.dat" in Section 3.1), and

```
3  =  nBLOCK
2    3   -2   = bLOCKsTRUCT
```

in Example 2 (see the file "example2.dat" in Section 3.2). In either case, the letters "= nBLOCK" and "= bLOCKsTRUCT" are neglected.

4.7. Constant Vector

We write all the elements c_1, c_2, ..., c_m of the cost vector $\boldsymbol{c}$. In addition to blank letter(s) and tab code(s), we can use the letters

$$, \quad (\quad) \quad \{ \quad \}$$

to separate elements of the vector $\boldsymbol{c}$. We have

```
{48, -8, 20}
```

in Example 1 (see the file "example1.dat" in Section 3.1), and

```
{1.1, -10, 6.6, 19, 4.1}
```

in Example 2 (see the file "example2.dat" in Section 3.2).

4.8. Constraint Matrices.

According to the format with the use of nBLOCK and bLOCKsTRUCT stated in Section 3.6, we describe the constraint matrices $\boldsymbol{F}_0,\ \boldsymbol{F}_1,\ \ldots,\ \boldsymbol{F}_m$. In addition to blank letter(s) and tab code(s), we can use the letters

$$,\quad (\quad)\quad \{\quad \}$$

to separate elements of the matrices $\boldsymbol{F}_0,\ \boldsymbol{F}_1,\ \ldots,\ \boldsymbol{F}_m$ and their elements. In the general case of the block diagonal matrix $\boldsymbol{F}$ given in (1), we write the elements of $\boldsymbol{B}_1,\ \boldsymbol{B}_2,\ \ldots\ ,\boldsymbol{B}_\ell$ sequentially; when $\boldsymbol{B}_i$ is a diagonal matrix, we write only the diagonal element sequentially. If the matrix $\boldsymbol{F}$ is given by (2) (nBLOCK = 3, bLOCKsTRUCT = $(3, 2, -2)$), the corresponding representation of the matrix $\boldsymbol{F}$ turns out to be

```
{ {{1  2  3} {2  4  5} {3  5  6}}, {{1  2} {2  3}}, 4, 5 }
```

In Example 1 with nBLOCK = 1 and bLOCKsTRUCT = 2, we have

```
{ {-11,  0}, { 0, 23} }
{ { 10,  4}, { 4,  0} }
{ {  0,  0}, { 0, -8} }
{ {  0, -8}, {-8, -2} }
```

See the file "example1.dat" in Section 3.1.

In Example 2 with nBLOCK = 3 and bLOCKsTRUCT = $(2, 3, -2)$, we have

```
{
{ { -1.4, -3.2 },
  { -3.2,-28   }   }
{ { 15,  -12,    2.1 },
  {-12,   16,   -3.8 },
  {  2.1, -3.8, 15   }   }
  {  1.8, -4.0 }
}
{
{ {  0.5,  5.2 },
  {  5.2, -5.3 }   }
{ {  7.8, -2.4,  6.0 },
  { -2.4,  4.2,  6.5 },
  {  6.0,  6.5,  2.1 }   }
  { -4.5, -3.5 }
}
```

•
•
•

```
{
{ { -6.5, -5.4 },
  { -5.4, -6.6 }   }
```

```
{ {  6.7, -7.2, -3.6 },
  { -7.2,  7.3, -3.0 },
  { -3.6, -3.0, -1.4 }   }
  {  6.1, -1.5 }
}
```

See the file "example2.dat" in Section 3.2.

Remark. We could also write the input data of Example 1 without using any letters

, ( ) { }

such as

```
"Example 1: mDim = 3, nBLOCK = 1, {2}"
3
1
2
 48  -8  20
-11   0   0  23
 10   4   4   0
  0   0   0  -8
  0  -8  -8  -2
```

5. Parameter File.

First we show the default parameter file "param.sdpa" below.

```
40      unsigned int maxIteration;
1.0E-7  double 0.0 < epsilonStar;
1.0E2   double 0.0 < lambdaStar;
2.0     double 1.0 < megaStar;
-1.0E5  double lowerBound;
1.0E5   double upperBound;
0.1     double 0.0 <= betaStar <  1.0;
0.2     double 0.0 <= betaBar  <  1.0, betaStar <= betaBar;
0.9     double 0.0 < gammaStar  <  1.0;
1.0E-7  double 0.0 < epsilonDash;
```

The file "param.sdpa" needs to have these 9 lines which respectively presents 9 parameters. Each line of the file "param.sdpa" contains one of the 9 parameters followed by any comment. When the SDPA reads the file "param.sdpa", it neglects the comment.

- maxIteration — The maximum number of iterations. The SDPA stops when the iteration exceeds the maxIteration.

- epsilonStar, epsilonDash — The accuracy of an approximate optimal solution of the SDP. When the current iterate $(\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ is satisfies the inequalities

$$\begin{aligned}
\text{epsilonDash} &\geq \max\left\{\left|[X^k - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^k + \boldsymbol{F}_0]_{pq}\right| \ : \ p,q = 1,2,\ldots,n\right\}, \\
\text{epsilonDash} &\geq \max\left\{\left|\boldsymbol{F}_i \bullet \boldsymbol{Y}^k - c_i\right| \ : i = 1,2,\ldots,m\right\}, \\
\text{epsilonStar} &\geq \frac{|\sum_{i=1}^{m} c_i x_i^k - \boldsymbol{F}_0 \bullet \boldsymbol{Y}^k|}{\max\left\{(|\sum_{i=1}^{m} c_i x_i^k| + |\boldsymbol{F}_0 \bullet \boldsymbol{Y}^k|)/2.0,\ 1.0\right\}} \\
&= \frac{|\text{the primal objective value} - \text{the dual objective value}|}{\max\left\{(|\text{the primal objective value}| + |\text{the dual objective value}|)/2.0,\ 1.0\right\}},
\end{aligned}$$

  the SDPA stops. Too small epsilonStar and epsilonDash may cause a numerical instability. A reasonable choice is epsilonStar $\geq 1.0E-7$.

- lambdaStar — This parameter determines an initial point $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ such that

$$\boldsymbol{x}^0 = \boldsymbol{0},\ \boldsymbol{X}^0 = \text{lambdaStar} \ \times \ \boldsymbol{I},\ \boldsymbol{Y}^0 = \text{lambdaStar} \ \times \ \boldsymbol{I}.$$

  Here $\boldsymbol{I}$ denotes the identity matrix. It is desirable to choose an initial point $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ having the same order of magnitude as an optimal solution $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ of the SDP. In general, however, choosing such a lambdaStar is difficult. If there is no information on the magnitude of an optimal solution $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ of the SDP, we strongly recommend to take a sufficiently large lambdaStar such that

$$\boldsymbol{X}^* \preceq \text{lambdaStar} \ \times \ \boldsymbol{I} \quad \text{and} \quad \boldsymbol{Y}^* \preceq \text{lambdaStar} \ \times \ \boldsymbol{I}.$$

- omegaStar — This parameter determines the region in which the SDPA searches an optimal solution. For the primal problem $\mathcal{P}$, the SDPA searches a minimum solution $(\boldsymbol{x}, \boldsymbol{X})$ within the region

$$\boldsymbol{O} \preceq \boldsymbol{X} \preceq \text{omegaStar} \ \times \ \boldsymbol{X}^0 = \text{omegaStar} \times \text{lambdaStar} \ \times \ \boldsymbol{I},$$

  and stops the iteration if it detects that the primal problem $\mathcal{P}$ has no minimum solution in this region. For the dual problem $\mathcal{D}$, the SDPA searches a maximum solution $\boldsymbol{Y}$ within the region

$$\boldsymbol{O} \preceq \boldsymbol{Y} \preceq \text{omegaStar} \ \times \ \boldsymbol{Y}^0 = \text{omegaStar} \times \text{lambdaStar} \ \times \ \boldsymbol{I},$$

  and stops the iteration if it detects that the dual problem $\mathcal{D}$ has no maximum solution in this region. Again we recommend to take a larger lambdaStar and a smaller omegaStar > 1.

- lowerBound — Lower bound of the minimum objective value of the primal problem $\mathcal{P}$. When the SDPA generates a primal feasible solution $(\boldsymbol{x}^k, \boldsymbol{X}^k)$ whose objective value $\sum_{i=1}^{m} c_i x_i^k$ gets smaller than the lowerBound, the SDPA stops the iteration; the primal problem $\mathcal{P}$ is likely to be unbounded and the dual problem $\mathcal{D}$ is likely to be infeasible if the lowerBound is sufficiently small.

- upperBound — Upper bound of the maximum objective value of the dual problem $\mathcal{D}$. When the SDPA generates a dual feasible solution $\boldsymbol{Y}^k$ whose objective value $\boldsymbol{F}_0 \bullet \boldsymbol{Y}^k$ gets larger than the upperBound, the SDPA stops the iteration; the dual problem $\mathcal{D}$ is likely to be unbounded and the primal problem $\mathcal{P}$ is likely to be infeasible if the upperBound is sufficiently large.

- betaStar — A parameter controlling the search direction when $(\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ is feasible. As we take a smaller betaStar > 0.0, the search direction can get close to the affine scaling direction without centering.
- betaBar — A parameter controlling the search direction when $(\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ is infeasible. As we take a smaller betaBar > 0.0, the search direction can get close to the affine scaling direction without centering. The value of betaBar must be not less than the value of betaStar; $0 \leq$ betaStar $\leq$ betaBar.
- gammaStar — A reduction factor for the primal and dual step lengths; $0.0 <$ gammaStar < 1.0.

6. Output.

6.1. Execution of the SDPA.

To execute the SDPA, we specify and type the names of three files, "sdpa", "an input data file" and "an output file" as follows.

% sdpa "an input data file" "an output file"

To solve Example 1, type:

% sdpa example1.dat example1.out

6.2. Output on the Display.

The SDPA shows some information on the display. In the case of Example 1, we have

```
        ************************************************
        *                                              *
        *               SDPA version 5.00              *
        *        (C) :  Kyoto University,              *
        *        Tokyo Institute of Technology., 1999  *
        *                                              *
        ************************************************

Search Direction = HRVW/KSH/M
   mu      thetaP  thetaD   objValP   objValD  alphaP  alphaD  beta
 0 1.0e+04 1.0e+00 1.0e+00 +0.00e+00 +1.20e+03 1.0e+00 9.1e-01 0.20
 1 1.6e+03 0.0e+00 9.4e-02 +8.39e+02 +7.51e+01 2.3e+00 9.6e-01 0.20
 2 1.7e+02 0.0e+00 3.6e-03 +1.96e+02 -3.74e+01 1.3e+00 1.0e+00 0.20
 3 1.8e+01 0.0e+00 0.0e+00 -6.84e+00 -4.19e+01 9.9e-01 9.9e-01 0.10
 4 1.9e+00 0.0e+00 0.0e+00 -3.81e+01 -4.19e+01 1.0e+00 9.0e+01 0.10
 5 1.9e-01 0.0e+00 0.0e+00 -4.15e+01 -4.19e+01 1.0e+00 1.0e+00 0.10
 6 1.9e-02 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 0.10
 7 1.9e-03 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 0.10
```

```
 8 1.9e-04 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 0.10
 9 1.9e-05 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 0.10
10 1.9e-06 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 0.10
phase.value = pdOPT
relative gap = 9.16e-08
gap = 3.84e-06, mu = 1.92e-06
objValPrimal   = -4.190000e+01
objValDual     = -4.190000e+01
p. feas. error = 1.776357e-14
d. feas. error = 4.796163e-13
total time in seconds = 0.010000
main loop time in seconds = 0.010000
```

- The 1st line — "Title and Comment" that is written on top of the input data file.
- Search direction = HRVW/KSH/M — Here the HRVW/KSH/M search direction [4, 5, 7] is used. At the user's option, the NT search direction [8, 10], and the AHO search direction [1, 2] are available. See Section 6.2 for more details.
- mu — The average complementarity $\boldsymbol{X}^k \bullet \boldsymbol{Y}^k/n$ (an optimality measure). When both $\mathcal{P}$ and $\mathcal{D}$ get feasible, the relation

$$\begin{aligned} \text{mu} &= \left(\sum_{i=1}^{m} c_i x_i^k - \boldsymbol{F}_0 \bullet \boldsymbol{Y}^k \right) /n \\ &= \frac{\text{the primal objective function - the dual objective function}}{n} \end{aligned}$$

  holds.
- thetaP — The SDPA starts with thetaP = 0.0 if the initial point $(\boldsymbol{x}^0, \boldsymbol{X}^0)$ of the primal problem $\mathcal{P}$ is feasible, and thetaP = 1.0 otherwise; hence it usually starts with thetaP = 1.0. In the latter case, the thetaP at the kth iteration is given by

$$\text{thetaP} = \frac{\|\boldsymbol{X}^k - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^k - \boldsymbol{F}_0\|}{\|\boldsymbol{X}^0 - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^0 - \boldsymbol{F}_0\|};$$

  The thetaP is monotone nonincreasing, and when it gets 0.0, we obtain a primal feasible solution $(\boldsymbol{x}^k, \boldsymbol{X}^k)$. In the example above, we obtained a primal feasible solution in the 1st iteration.
- thetaD — The SDPA starts with thetaD = 0.0 if the initial point $\boldsymbol{Y}^0$ of the dual problem $\mathcal{D}$ is feasible, and thetaD = 1.0 otherwise; hence it usually starts with thetaD = 1.0. In the latter case, the thetaD at the kth iteration is given by

$$\text{thetaD} = \frac{\left(\sum_{i=1}^{m} (\boldsymbol{F}_i \bullet \boldsymbol{Y}^k - c_i)^2 \right)^{1/2}}{\left(\sum_{i=1}^{m} (\boldsymbol{F}_i \bullet \boldsymbol{Y}^0 - c_i)^2 \right)^{1/2}};$$

  The thetaD is monotone nonincreasing, and when it gets 0.0, we obtain a dual feasible solution $\boldsymbol{Y}^k$. In the example above, we obtained a dual feasible solution in the 3rd iteration.
- objValP — The primal objective function value.

- objValD — The dual objective function value.
- alphaP — The primal step length.
- alphaD — The dual step length.
- beta — The search direction parameter. which takes a value in the interval [0, 1). Here $\lambda_{\rm ave}$ and $\lambda_{\rm min}$ denote the average and the minimum of all the eigenvalues of $\boldsymbol{X}^k\boldsymbol{Y}^k$. As either of $\boldsymbol{X}^k$ and $\boldsymbol{Y}^k$ get close to the boundary of the positive semidefinite cone, the delta gets larger.
- phase.value — The status when the iteration stops, taking one of the values pdOPT, noINFO, pFEAS, dFEAS, pdFEAS, pdINF, pFEAS_dINF, pINF_dFEAS, pUNBD and dUNBD.

  pdOPT : The normal termination yielding both primal and dual approximate optimal solutions.

  noINFO : The iteration has exceeded the maxIteration and stopped with no information on the primal feasibility and the dual feasibility.

  pFEAS : The primal problem $\mathcal{P}$ got feasible but the iteration has exceeded the maxIteration and stopped.

  dFEAS : The dual problem $\mathcal{D}$ got feasible but the iteration has exceeded the maxIteration and stopped.

  pdFEAS : Both primal problem $\mathcal{P}$ and the dual problem $\mathcal{D}$ got feasible, but the iteration has exceeded the maxIteration and stopped.

  pdINF : At least one of the primal problem $\mathcal{P}$ and the dual problem $\mathcal{D}$ is expected to be infeasible. More precisely, there is no optimal solution $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ of the SDP such that

$$\begin{aligned}
&\boldsymbol{O} \preceq \boldsymbol{X} \preceq \text{omegaStar} \times \boldsymbol{X}^0,\\
&\boldsymbol{O} \preceq \boldsymbol{Y} \preceq \text{omegaStar} \times \boldsymbol{Y}^0,\\
&\sum_{i=1}^{m} c_i x_i = \boldsymbol{F}_0 \bullet \boldsymbol{Y}.
\end{aligned}$$

  pFEAS_dINF : The primal problem $\mathcal{P}$ has become feasible but the dual problem is expected to be infeasible. More precisely, there is no dual feasible solution $\boldsymbol{Y}$ such that

$$\boldsymbol{O} \preceq \boldsymbol{Y} \preceq \text{omegaStar} \times \boldsymbol{Y}^0 = \text{lambdaStar} \times \text{omegaStar} \times \boldsymbol{I}.$$

  pINF_dFEAS : The dual problem $\mathcal{D}$ has become feasible but the primal problem is expected to be infeasible. More precisely, there is no feasible solution $(\boldsymbol{x}, \boldsymbol{X})$ such that

$$\boldsymbol{O} \preceq \boldsymbol{X} \preceq \text{omegaStar} \times \boldsymbol{X}^0 = \text{lambdaStar} \times \text{omegaStar} \times \boldsymbol{I}.$$

  pUNBD : The primal problem is expected to be unbounded. More precisely, the SDPA has stopped generating a primal feasible solution $(\boldsymbol{x}^k, \boldsymbol{X}^k)$ such that

$$\text{objValP} = \sum_{i=1}^{m} c_i x_i^k < \text{lowerBound}.$$

  dUNBD : The dual problem is expected to be unbounded. More precisely, the SDPA has stopped generating a dual feasible solution $\boldsymbol{Y}^k$ such that

$$\text{objValD} = \boldsymbol{F}_0 \bullet \boldsymbol{Y}^k > \text{upperBound}.$$

relative gap : The relative gap means that

$$\frac{|\text{objValP} - \text{objValD}|}{\max\{1.0,\ (|\text{objValP}| + |\text{objValD}|)/2\}}.$$

gap : The gap means that mu $\times\, n$.

6.3. Output to a File.

We show the content of the file "example2.out" on which the SDPA has written the computational results of Example 2.

```
Data   file name = example2.dat
Output file name = example2.out
*Example 2:
*mDim = 5, nBLOCK = 3, {2,3,-2}
maxIteration =  40
epsilonStar  = 1.00e-07
lambdaStar   = 1.00e+02
omegaStar    = 1.00e+02
lowerBound   = -1.00e+07
upperBound   = 1.00e+07
alphaStar    = 0.00e+00
betaStar     = 1.00e-01
betaBar      = 2.00e-01
gammaStar    = 9.00e-01
deltaStar    = 1.00e+02
epsilonDash  = 1.00e-07
Search Direction = HRVW/KSH/M
   mu      thetaP  thetaD   objValP   objValD  alphaP  alphaD  beta
 0 1.0e+04 1.0e+00 1.0e+00 +0.00e+00 +1.44e+03 8.8e-01 6.6e-01 0.20
 1 3.3e+03 1.2e-01 3.4e-01 +4.94e+02 +2.84e+02 1.0e+00 8.2e-01 0.20
 2 9.0e+02 0.0e+00 6.2e-02 +8.66e+02 -2.60e+00 1.0e+00 1.0e+00 0.20
 3 1.4e+02 0.0e+00 0.0e+00 +9.67e+02 +9.12e-01 9.5e-01 5.4e+00 0.10
 4 1.8e+01 0.0e+00 0.0e+00 +1.46e+02 +2.36e+01 9.3e-01 1.5e+00 0.10
 5 2.7e+00 0.0e+00 0.0e+00 +4.62e+01 +2.74e+01 8.1e-01 1.4e+00 0.10
 6 5.7e-01 0.0e+00 0.0e+00 +3.43e+01 +3.03e+01 9.2e-01 9.3e-01 0.10
 7 9.5e-02 0.0e+00 0.0e+00 +3.24e+01 +3.18e+01 9.5e-01 9.6e-01 0.10
 8 1.3e-02 0.0e+00 0.0e+00 +3.21e+01 +3.20e+01 9.8e-01 1.0e+00 0.10
 9 1.5e-03 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 0.10
10 1.5e-04 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 0.10
11 1.5e-05 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 0.10
12 1.5e-06 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 0.10
13 1.5e-07 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 0.10
Computational Result :
No of Iterations =  13
phase.value = pdOPT
mu0 = 1.000e+04
mu  = 1.502e-07
```

```
relative gap = 3.28e-08
gap = 1.05e-06, mu = 1.50e-07
objValPrimal   = 3.206269e+01
objValDual     = 3.206269e+01
p. feas. error = 7.194245e-14
d. feas. error = 2.550848e-12
.xVect =
{+1.55164460010148541613,+0.67096725438335802494,+0.98149165903359281149,+1.406569462598729947C
.xMat =
{
{ {+6.39176521E-08,-9.63772306E-09 },
  {-9.63772306E-09,+4.53939370E-08 }   }
{ {+7.11915568E+00,+5.02467127E+00,+1.91629481E+00 },
  {+5.02467127E+00,+4.41474591E+00,+2.50602213E+00 },
  {+1.91629481E+00,+2.50602213E+00,+2.04812384E+00 }   }
{+3.43246578E-01,+4.39116898E+00 }
}
.yMat =
{
{ {+2.64026632E+00,+5.60564126E-01 },
  {+5.60564126E-01,+3.71763811E+00 }   }
{ {+7.61550055E-01,-1.51352507E+00,+1.13936912E+00 },
  {-1.51352507E+00,+3.00802056E+00,-2.26441298E+00 },
  {+1.13936912E+00,-2.26441298E+00,+1.70463144E+00 }   }
{+4.08731192E-07,+3.19496128E-08 }
}
.xEigenValues =
{
{+0.00000006802245872866,+0.00000004128913035012}
{+11.91598935316592289269,+1.66603604411658356987,+0.00000002984067721019}
{+0.34324657780398809548,+4.39116898405516309367}
}
.yEigenValues =
{
{+2.40151113125386528324,+3.95639329602223366322}
{+5.47420196139708803429,+0.00000001177280279186,+0.00000008420984931629}
{+0.00000040873119209784,+0.00000003194961279829}
}
.xyEigenValues =
{
{+10000.000000000000000000,+10000.000000000000000000}
{+10000.000000000000000000,+10000.000000000000000000,+10000.000000000000000000}
{+10000.000000000000000000,+10000.000000000000000000}
}
total time in seconds = 0.020000
```

Now we explain items appeared above in the file "example2.out".

- xVect — The primal variable vector $\boldsymbol{x}$.

- xMat — The primal variable matrix $\boldsymbol{X}$.
- yMat — The dual variable matrix $\boldsymbol{Y}$.
- xEigenValues — The eigenvalues of the primal variable matrix $\boldsymbol{X}$. All the values here must be nonnegative so that the primal variable matrix $\boldsymbol{X}$ is positive semidefinite.
- yEigenValues — The eigenvalues of the dual variable matrix $\boldsymbol{Y}$. All the values here must be nonnegative so that the dual variable matrix $\boldsymbol{Y}$ is positive semidefinite.
- xyEigenValues — The eigenvalues of the matrix $\boldsymbol{XY}$. We can verify the optimality to check whether all the values here are sufficiently small.

7. Advanced Use of the SDPA.

7.1. Checking Input Data File.

We can check whether input matrices are symmetric by adding the option "−c"; for example,

% sdpa example2.dat example2.out −c

In Example 2, if the matrix $\boldsymbol{B}_1$ of the block diagonal matrix $\boldsymbol{F}_5$ was not symmetric, then the SDPA would display an error message. For example, if

```
{
{ { -6.5, -5.4 },
  { -5.4, -6.6 }   }
{ {  6.7, -7.2,  3.6 },
  { -7.2,  7.3, -3.0 },
  { -3.6, -3.0, -1.4 }   }
  {  6.1, -1.5 }
}
```

then

```
Matrix F[5]->block[1] is not symmetric : [0,2] = 3.600000, [2,0] = -3.600000
```

7.2. Three Types of Search Directions.

The SDPA version 2 incorporated three types of search directions; the search direction HRVW/KSH/M proposed by [4, 5] (see also [7]), the search direction NT by [8], and the search direction AHO by [1, 2]. Adding the option "-1", "-2" or "-3", we can specify one of these search directions. For example, if we want to use the search direction NT proposed by [8] to solve Example 2, type

% sdpa example2.dat example2.out −2

Without any of the options "-1", "-2" and or "-3", the SDPA uses the search direction HRVW/KSH/M proposed by [4, 5].

7.3. Initial Point.

If a feasible-interior solution $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ is known in advance, we may want to start the SDPA from $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$. In such a case, we can optinally specify a file which contains the data of a feasible-interior solution when we execute the SDPA; for example if we want to solve Example 1 from a feasible-interior initial point

$$(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0) = \left(\begin{pmatrix} 2.0 \\ 0.0 \\ 0.0 \end{pmatrix} \begin{pmatrix} 31.0 & 3.0 \\ 3.0 & 5.0 \end{pmatrix} \begin{pmatrix} 2.0 & 0.0 \\ 0.0 & 2.0 \end{pmatrix} \right),$$

type

% sdpa example1.dat example1.out example1.ini

Here "example1.ini" denotes an initial point file containing the data of a feasible-interior solution:

```
{0.0, -4.0, 0.0}
{ {11.0, 0.0}, {0.0, 9.0} }
{ {5.9,  -1.375}, {-1.375, 1.0} }
```

In general, the initial point file can have any name with the postfix ".ini'; for example, "example.ini" are legitimate initial point file names. It must be placed after the output file. The other options "−c", "−1", "−2" and/or "−3" can be put in any place; both

% sdpa example1.dat example1.out example1.ini −c −3

% sdpa example1.dat example1.out −3 −c example1.ini

are acceptable. An initial point file contains the data

$\boldsymbol{x}^0$
$\boldsymbol{X}^0$
$\boldsymbol{Y}^0$

in this order, where the description of the m-dimensional vector $\boldsymbol{x}^0$ must follow the same format as the constant vector $\boldsymbol{c}$ (see Section 3.7), and the description of $\boldsymbol{X}^0$ and $\boldsymbol{Y}^0$ the same format as the constraint matrix $\boldsymbol{F}_i$ (see Section 3.8).

7.4. Sparse Input Data File.

In Section 3, we have stated the dense data format for inputting the data m, n, $\boldsymbol{c} \in R^m$ and $\boldsymbol{F}_i \in \mathcal{S}$ $(i = 0, 1, 2, \ldots, m)$. When not only the constant matrices $\boldsymbol{F}_i \in \mathcal{S}$ $(i = 0, 1, 2, \ldots, m)$ are block diagonal but also each block is sparse, the sparse data format described in this section gives us a compact description of the constant matrices.

A sparse input data file must have a name with the postfix ".dat-s"; for example, "problem.dat-s" and "example.dat-s" are legimate names for sparse input data files. The SDPA distinguishs a sparse input data file with the postfix ".dat-s" from a dense input data file with the postfix ".dat"

We show below the file "example1.dat-s", which contains the date of Example 1 (Section 3.1) in the sparse data format.

```
"Example 1: mDim = 3, nBLOCK = 1, {2}"
   3  =  mDIM
   1  =  nBLOCK
   2  = bLOCKsTRUCT
{48, -8, 20}
0 1 1 1 -11
0 1 2 2 23
1 1 1 1 10
1 1 1 2 4
2 1 2 2 -8
3 1 1 2 -8
3 1 2 2 -2
```

Compare the dense input data file "example1.dat" described in Section 3.1 with the sparse input data file "example1.dat-s" above . The first 5 lines of the file "example1.dat-s" are the same as those of the file "example1.dat". Each line of the rest of the file "example1.dat-s" describes a single element of a constant matrix $\boldsymbol{F}_i$; the 6th line "0 1 1 1 -11" means that the $(1,1)$th element of the 1st block of the matrix $\boldsymbol{F}_0$ is -11, and the 11th line "3 1 1 2 -8" means that the $(1,2)$th element of the 1st block of the matrix $\boldsymbol{F}_3$ is -8.

In general, the structure of a sparse input data file is as follows:

Title and Comment
m — the number of the primal variables x_i's
nBLOCK — the number of blocks
bLOCKsTRUCT — the block structure vector
$\boldsymbol{c}$
$k_1\ b_1\ i_1\ j_1\ v_1$
$k_2\ b_2\ i_2\ j_2\ v_2$
$\quad\ldots$
$k_p\ b_p\ i_p\ j_p\ v_p$
$\quad\ldots$
$k_q\ b_q\ i_q\ j_q\ v_q$

Here $k_p \in \{0, 1, \ldots, m\}$, $b_p \in \{1, 2, \ldots, \text{nBLOCK}\}$, $1 \le i_p \le j_p$ and $v_p \in R$. Each line "k_p, b_p, i_p, j_p, v_p" means that the value of the (i_p, j_p)th element of the b_pth block of the constant matrix $\boldsymbol{F}_{k_p}$ is v_p. If the b_pth block is an $\ell \times \ell$ symmetric (non-diagonal) matrix then (i_p, j_p) must satisfy $1 \le i_p \le j_p \le \ell$; hence only nonzero elements in the upper triangular part of the b_pth block are described in the file. If the b_pth block is an $\ell \times \ell$ diagonal matrix then (i_p, j_p) must satisfy $1 \le i_p = j_p \le \ell$.

7.5. Sparse Initial Point File.

We show below the file "example1.ini-s", which contains an initial point data of Example 1 in the sparse data format.

```
{0.0, -4.0, 0.0}
1 1 1 1 11
1 1 2 2 9
```

```
2 1 1 1 5.9
2 1 1 2 -1.375
2 1 2 2 1
```

Compare the dense initial point file "example1.ini" described in Section 6.3 with the sparse initial file "example1.ini-s" above . The first line of the file "example1.ini-s" is the same as that of the file "example1.ini", which describes $\boldsymbol{x}^0$ in the dense format. Each line of the rest of the file "example1.ini-s" describes a single element of an initial matrix $\boldsymbol{X}^0$ if the first number of the line is 1 or a single element of an initial matrix $\boldsymbol{Y}^0$ if the first number of the line is 2; The 2nd line "1 1 1 1 11" means that the $(1,1)$th element of the 1st block of the matrix $\boldsymbol{X}^0$ is 11, the 5th line "2 1 1 2 -1.375" means that the $(1,2)$th element of the 1st block of the matrix $\boldsymbol{Y}^0$ is -1.375.

A sparse initial point file must have a name with the postfix ".ini-s"; for example, "problem.ini-s" and "example.ini-s" are legimate names for sparse input data files. The SDPA distinguishs a sparse input data file with the postfix ".ini" from a dense input data file with the postfix ".ini-s"

In general, the structure of a sparse input data file is as follows:

$\boldsymbol{x}^0$
$s_1\ b_1\ i_1\ j_1\ v_1$
$s_2\ b_2\ i_2\ j_2\ v_2$
$\quad\ \ldots$
$s_p\ b_p\ i_p\ j_p\ v_p$
$\quad\ \ldots$
$s_q\ b_q\ i_q,\ j_q\ v_q$

Here $s_p = 1$ or 2, $b_p \in \{1, 2, \ldots, \mathrm{nBLOCK}\}$, $1 \le i_p \le j_p$ and $v_p \in R$. When $s_p = 1$, each line "$s_p\ b_p\ i_p\ j_p\ v_p$" means that the value of the (i_p, j_p)th element of the b_pth block of the constant matrix $\boldsymbol{X}^0$ is v_p. When $s_p = 2$, the line "$s_p\ b_p\ i_p\ j_p\ v_p$" means that the value of the (i_p, j_p)th element of the b_pth block of the constant matrix $\boldsymbol{Y}^0$ is v_p. If the b_pth block is an $\ell \times \ell$ symmetric (non-diagonal) matrix then (i_p, j_p) must satisfy $1 \le i_p \le j_p \le \ell$; hence only nonzero elements in the upper triangular part of the b_pth block are described in the file. If the b_pth block is an $\ell \times \ell$ diagonal matrix then (i_p, j_p) must satisfy $1 \le i_p = j_p \le \ell$.

7.6. More on Parameter File.

We may encounter some numerical difficulty during the execution of the SDPA with the default parameter file "param.sdpa", and/or we may want to solve many easy SDPs with similar data more quickly. In such a case, we need to adjust some of the defualt parameters, betaStar, betaBar, gammaStar and deltaStar. We presnt below two sets of those parameters. The one is the set "Stable_but_Slow" for difficult SDPs, and the other is the set "Unstable_but_Fast" for easy SDPs.

Stable_but_Slow

```
0.10 double 0.0 <= betaStar <  1.0;
0.20 double 0.0 <= betaBar  <  1.0, betaStar <= betaBar;
0.90 double 0.0 < gammaStar  <  1.0;
```

Unstable_but_Fast

```
0.01 double 0.0 <= betaStar <  1.0;
0.02 double 0.0 <= betaBar  <  1.0, betaStar <= betaBar;
0.98 double 0.0 < gammaStar  <  1.0;
```

Besides these parameters, the value of the parameter lambdaStar, which determines an initial point $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$, affects the computational efficiency and the numerical stability. Usually a larger lambdaStar is safe although the SDPA may consume a few more iterations.

8. The Callable Library of SDPA

The easiest way to understand how to use the callable library is to look at example files. For this purpose, we have chosen a problem "Example1" in Section 2. Here consider several cases when solving problems with the callable library. Roughly speaking, we provides two usages for this callable library. The first usage needs input data, output and initial point files (Case 1). And you can also generate your own problem in your C++ source file and solve this problem directly by calling several functions of the callable library (Case 2).

8.1. Case 1:

As we have already seen in Section 2, to solve Example1, type:

% `sdpa example1.dat example1.out`

We show below a source program in the "`example1-1.cpp`" for reading a problem from an input data file `example1.dat` and putting its output into an output file `example1.out`. To compile and execute this source file, one type:

```
make PROG=example1-1
example1-1 example1.dat example1.out
```

```
/* The beginning of the ‘‘example1-1.cpp’’. */
#include <stdio.h>
#include <stdlib.h>

#include "sdpa-lib.hpp"
#include "sdpa-lib2.hpp"

int main (int argc, char *argv[])
{
        if (argc != 3)
        {
                fprintf(stderr, "%s [Input] [Output] \n", argv[0]);
                exit(EXIT_FAILURE);
        }

        SDPA    Problem1;

        Problem1.Method          = KSH;
```

```
        strcpy(Problem1.ParameterFileName, "param.sdpa");
        Problem1.ParameterFile = fopen(Problem1.ParameterFileName, "r");
        strcpy(Problem1.InputFileName, argv[1]);
        Problem1.InputFile = fopen(Problem1.InputFileName,"r");
        strcpy(Problem1.OutputFileName, argv[2]);
        Problem1.OutputFile = fopen(Problem1.OutputFileName,"w");

        Problem1.DisplayInformation = stdout;

        SDPA_initialize(Problem1);
        SDPA_Solve(Problem1);

        fclose(Problem1.InputFile);
        fclose(Problem1.OutputFile);

        Problem1.Delete();

        exit(0);
};
/* The end of the ‘‘example1-1.cpp’’. */
```

To use the SDPA library functions, one needs to several header files as follows:

```
/* The beginning of the ‘‘example1-1.cpp’’. */
#include <stdio.h>
#include <stdlib.h>

#include "sdpa-lib.hpp"
#include "sdpa-lib2.hpp"
```

Notice that all header files must be in the same directory. We needs to declare a object (variable), what we call `Problem1`, that is a object to the class `SDPA` in the header file **sdpa-lib2.hpp**.

```
        SDPA    Problem1;
```

After this declaration, one usually select a search direction among HRVW/KSH/M (`KSH`), NT (`NT`) and AHO (`AHO`) directions. If one will not select a search direction, the default direction will be set to `KSH`.

```
        Problem1.Method          = KSH;
```

The next procedures are very important and one must carefully specify the file names and their file pointers.

```
        strcpy(Problem1.ParameterFileName, "param.sdpa");
        Problem1.ParameterFile = fopen(Problem1.ParameterFileName, "r");
        strcpy(Problem1.InputFileName, argv[1]);
        Problem1.InputFile = fopen(Problem1.InputFileName,"r");
        strcpy(Problem1.OutputFileName, argv[2]);
        Problem1.OutputFile = fopen(Problem1.OutputFileName,"w");
```

One will not necessarily set a parameter file name to "param.sdpa". Because this callable library also distinguishes the dense data format with the postfix ".dat" from the sparse data format with the postfix "dat-s" and one must copy the file names to `ParameterFileName`, `InputFileName` and `OutputFileName`, respectively.

If one wants to show some informations on the display, the following line will be needed (the default value is `NULL`).

```
Problem1.DisplayInformation = stdout;
```

After calling the function `SDPA_initialize(SDPA &)`, we are now ready to put the call to the solver in the calling routine `SDPA_Solve(SDPA &)`.

```
SDPA_initialize(Problem1);
SDPA_Solve(Problem1);
```

Finally, one closes all used file pointers and free a object `Problem1` from the computational memory space by calling the function `Delete()`.

```
fclose(Problem1.InputFile);
fclose(Problem1.OutputFile);

Problem1.Delete();
```

See also "exmaple1-2.cpp", which reads a problem from an input data file, an initial point data file, and puts its output into an output file.

8.2. Case 2:

In this section, we show how to generate a problem in our source file and solve this by calling the functions. The C++ source program below is contained in the "`example2-1.cpp`". To compile and execute this source file, one type:

```
make PROG=example2-1
example2-1
```

```
/* The beginning of the ‘‘example2-1.cpp’’. */
#include <stdio.h>
#include <stdlib.h>

#include "sdpa-lib.hpp"
#include "sdpa-lib2.hpp"

/*
example1.dat:

"Example 1: mDim = 3, nBLOCK = 1, {2}"
   3  =  mDIM
```

```
   1  =  nBOLCK
   2  = bLOCKsTRUCT
{48, -8, 20}
{ {-11,  0}, { 0, 23} }
{ { 10,  4}, { 4,  0} }
{ {  0,  0}, { 0, -8} }
{ {  0, -8}, {-8, -2} }
*/

/*
example1.ini:

{0.0, -4.0, 0.0}
{ {11.0, 0.0}, {0.0, 9.0} }
{ {5.9,  -1.375}, {-1.375, 1.0} }
*/

boolean MatrixDisplay(int mRow, int nCol, double** element, FILE* outFile);
boolean VectorDisplay(int nDim, double* element, FILE* outFile);

int main ()
{
        int             mRow, nCol;
        double**        element;
        double*         element2;

        SDPA    Problem1;

        Problem1.Method                 = KSH;
        Problem1.InitialPoint           = true;

        Problem1.pARAM.maxIteration     = 50;
        Problem1.pARAM.epsilonStar      = 1.0E-8;
        Problem1.pARAM.lambdaStar       = 1.0E2;
        Problem1.pARAM.omegaStar        = 2.0;
        Problem1.pARAM.lowerBound       = -1.0E5;
        Problem1.pARAM.upperBound       = 1.0E5;
        Problem1.pARAM.betaStar         = 0.1;
        Problem1.pARAM.betaBar          = 0.2;
        Problem1.pARAM.gammaStar        = 0.9;

        Problem1.DisplayInformation     = stdout;

        SDPA_initialize(Problem1);

        Problem1.mDIM   = 3;
        Problem1.nBLOCK = 1;
        Problem1.bLOCKsTRUCT    = new int [Problem1.nBLOCK];
        Problem1.bLOCKsTRUCT[0] = 2;
```

```
        SDPA_initialize2(Problem1);

// cVECT = {48, -8, 20}
        SDPA_Input_cVECT(Problem1, 1, 48);
        SDPA_Input_cVECT(Problem1, 2, -8);
        SDPA_Input_cVECT(Problem1, 3, 20);

// F_0 = { {-11,  0}, { 0, 23} }
        SDPA_CountUpperTriangle(Problem1, 0, 1, 2);

// F_1 = { { 10,  4}, { 4,  0} }
        SDPA_CountUpperTriangle(Problem1, 1, 1, 2);

// F_2 = { {  0,  0}, { 0, -8} }
        SDPA_CountUpperTriangle(Problem1, 2, 1, 1);

// F_3 = { {  0, -8}, {-8, -2} }
        SDPA_CountUpperTriangle(Problem1, 3, 1, 2);


        SDPA_Make_sfMAT(Problem1);


// F_0 = { {-11,  0}, { 0, 23} }
        SDPA_InputElement(Problem1, 0, 1, 1, 1, -11);
        SDPA_InputElement(Problem1, 0, 1, 2, 2, 23);

// F_1 = { { 10,  4}, { 4,  0} }
        SDPA_InputElement(Problem1, 1, 1, 1, 1, 10);
        SDPA_InputElement(Problem1, 1, 1, 1, 2, 4);

// F_2 = { {  0,  0}, { 0, -8} }
        SDPA_InputElement(Problem1, 2, 1, 2, 2, -8);

// F_3 = { {  0, -8}, {-8, -2} }
        SDPA_InputElement(Problem1, 3, 1, 1, 2, -8);
        SDPA_InputElement(Problem1, 3, 1, 2, 2, -2);


// X^0 = { {11.0, 0.0}, {0.0, 9.0} }
        SDPA_Input_IniXMat(Problem1, 1, 1, 1, 11);
        SDPA_Input_IniXMat(Problem1, 1, 2, 2, 9);

// x^0 = {0.0, -4.0, 0.0}
        SDPA_Input_InixVec(Problem1, 2, -4);

// Y^0 = { {5.9, -1.375}, {-1.375, 1.0} }
        SDPA_Input_IniYMat(Problem1, 1, 1, 1, 5.9);
```

```
        SDPA_Input_IniYMat(Problem1, 1, 1, 2, -1.375);
        SDPA_Input_IniYMat(Problem1, 1, 2, 2, 1);


        SDPA_Solve(Problem1);


        fprintf(stdout, "\nStop iteration = %d\n", Problem1.Iteration);
        fprintf(stdout, "objValPrimal   = %10.6e\n", Problem1.PrimalObj);
        fprintf(stdout, "objValDual     = %10.6e\n", Problem1.DualObj);
        fprintf(stdout, "p. feas. error = %10.6e\n", Problem1.PrimalError);
        fprintf(stdout, "d. feas. error = %10.6e\n\n", Problem1.DualError);


        printf("\nxVec = \n");
//      Problem1.xVec.display(stdout);

        mRow     = Problem1.xVec.nDim;
        element2 = Problem1.xVec.element;
        VectorDisplay(mRow, element2, stdout);


        printf("\nxMat = \n");
//      Problem1.XMat.display(stdout);

        if (Problem1.XMat.nBlock > 1) fprintf(stdout, "{\n");
        for(int k = 0; k < Problem1.XMat.nBlock; k++)
        {
                mRow    = Problem1.XMat.block[k].mRow;
                nCol    = Problem1.XMat.block[k].nCol;
                element = Problem1.XMat.block[k].element;
                MatrixDisplay(mRow, nCol, element, stdout);
        }
        if (Problem1.XMat.nBlock > 1) fprintf(stdout, "}\n\n");


        printf("\nxMat = \n");
//      Problem1.YMat.display(stdout);

        if (Problem1.YMat.nBlock > 1) fprintf(stdout, "{\n");
        for(int k = 0; k < Problem1.YMat.nBlock; k++)
        {
                mRow    = Problem1.YMat.block[k].mRow;
                nCol    = Problem1.YMat.block[k].nCol;
                element = Problem1.YMat.block[k].element;
                MatrixDisplay(mRow, nCol, element, stdout);
        }
        if (Problem1.YMat.nBlock > 1) fprintf(stdout, "}\n\n");
```

```
        Problem1.Delete();

        exit(0);
};

boolean MatrixDisplay(int mRow, int nCol, double** element, FILE *outFile)
{
        fprintf(outFile, "{");
        for(int i = 0; i < mRow-1; i++)
        {
                if (i == 0)
                        fprintf(outFile," ");
                else
                        fprintf(outFile,"  ");
                fprintf(outFile, "{");
                for(int j = 0; j < nCol-1; j++)
                        fprintf(outFile, "%+8.8E,",element[i][j]);
                fprintf(outFile,"%+8.8E },\n",element[i][nCol-1]);
        }
        if (mRow > 1)
                fprintf(outFile,"  {");
        for(int j = 0; j < nCol-1; j++)
                fprintf(outFile, "%+8.8E,",element[mRow-1][j]);
        fprintf(outFile, "%+8.8E }",element[mRow-1][nCol-1]);
        if (mRow > 1)
                fprintf(outFile,"   }\n");
        else
                fprintf(outFile,"\n");

        return true;
};

boolean VectorDisplay(int nDim, double* element, FILE *outFile)
{
        fprintf(outFile, "{");
        for(int i = 0; i < nDim-1; i++)
                fprintf(outFile, "%+8.3E,",element[i]);
        if (nDim > 0)
                fprintf(outFile, "%+8.3E}\n",element[nDim-1]);
        else
                fprintf(outFile,"  }\n");

        return true;
};
/* The end of the ‘‘example2-1.cpp’’. */
```

One needs to four header files and declare a object, say `Problem1` like `Case 1`. For example, if you want to use the iostream which provides C++ input/output, "`#include <iostram.h>`"

must be added to this source file.

```
/* The beginning of the ‘‘example1-1.cpp’’. */
#include <stdio.h>
#include <stdlib.h>

#include "sdpa-lib.hpp"
#include "sdpa-lib2.hpp"
                      .
                      .
                      .
int main ()
{
        SDPA    Problem1;
```

The variable `Problem1` is generated as a object to the class `SDPA` with a call to `SDPA::SDPA()`.

```
        Problem1.Method                 = NT;
        Problem1.InitialPoint           = true;
```

In this case, we select the NT direction and set an initial point.

```
        Problem1.pARAM.maxIteration      = 50;
        Problem1.pARAM.epsilonStar      = 1.0E-8;
        Problem1.pARAM.lambdaStar       = 1.0E2;
        Problem1.pARAM.omegaStar        = 2.0;
        Problem1.pARAM.lowerBound       = -1.0E5;
        Problem1.pARAM.upperBound       = 1.0E5;
        Problem1.pARAM.betaStar       = 0.1;
        Problem1.pARAM.betaBar          = 0.2;
        Problem1.pARAM.gammaStar        = 0.9;
```

As we have seen in Section 5, the SDPA has 9 parameters which controls a search direction and decides a stopping-criterion. One must input these parameters from a parameter file like `Case 1`, or set all fields of a object `Problem1.pARAM` to a class `parameterClass` as above.

```
        Problem1.DisplayInformation     = stdout;

        SDPA_initialize(Problem1);

        Problem1.mDIM   = 3;
        Problem1.nBLOCK = 1;
        Problem1.bLOCKsTRUCT    = new int [Problem1.nBLOCK];
        Problem1.bLOCKsTRUCT[0] = 2;
```

After calling the `SDPA_initialize(Problem1)`, we begin by specifying the number of the primal variables, the block and block structure vector. If the meaning of `bLOCKsTRUCT` is not clear, one refers the Section 4. We can also implement the declaration of `bLOCKsTRUCT` as follows:

```
        Problem1.bLOCKsTRUCT    = (int *)malloc(Problem1.nBLOCK * sizeof(int));
```

One must pay attention to put the call `SDPA_initialize2(Problem1)` after setting `mDIM`, `nBLOCK` and `bLOCKsTRUCT` above. Here the array `cVECT(mDIM)` must be set as follows:

```
        SDPA_initialize2(Problem1);

//      cVECT = {48, -8, 20}
        SDPA_Input_cVECT(Problem1, 1, 48);
        SDPA_Input_cVECT(Problem1, 2, -8);
        SDPA_Input_cVECT(Problem1, 3, 20);
```

Next we set the number of **nonzero** elements of the **upper triangular part** of each block of each $\boldsymbol{F}_i (i = 0, \ldots, m)$.

```
//      F_0 = { {-11,  0}, { 0, 23} }
        SDPA_CountUpperTriangle(Problem1, 0, 1, 2);

//      F_1 = { { 10,  4}, { 4,  0} }
        SDPA_CountUpperTriangle(Problem1, 1, 1, 2);

//      F_2 = { {  0,  0}, { 0, -8} }
        SDPA_CountUpperTriangle(Problem1, 2, 1, 1);

//      F_3 = { {  0, -8}, {-8, -2} }
        SDPA_CountUpperTriangle(Problem1, 3, 1, 2);
```

Here `SDPA_CountUpperTriangle(Problem1, 0, 1, 2)` means that the number of **nonzero** elements of the upper triangular part of **1**st block of the constant matrix $\boldsymbol{F}_0$ is **2**.

```
        SDPA_Make_sfMAT(Problem1);


        SDPA_InputElement(Problem1, 0, 1, 1, 1, -11);
        SDPA_InputElement(Problem1, 0, 1, 2, 2, 23);

        SDPA_InputElement(Problem1, 1, 1, 1, 1, 10);
        SDPA_InputElement(Problem1, 1, 1, 1, 2, 4);

        SDPA_InputElement(Problem1, 2, 1, 2, 2, -8);

        SDPA_InputElement(Problem1, 3, 1, 1, 2, -8);
        SDPA_InputElement(Problem1, 3, 1, 2, 2, -2);
```

After calling the `SDPA_Make_sfMAT(Problem1)`, we must set only nonzero elements of the upper triangular part of each block. The call `SDPA_InputElement(Problem1, 1, 1, 1, 2, 4)` means that the (**1,2**) element of the **1** st block of the matrix $\boldsymbol{F}_1$ is **4**, `SDPA_InputElement(Problem1, 3, 1, 2, 2, -2)` means that the (**2,2**) element of the **1** st block of the matrix $\boldsymbol{F}_3$ is **-2**.

Similarly, if one needs, the initial point must therefore be initialized as follows:

```
//      X^0 = { {11.0, 0.0}, {0.0, 9.0} }
        SDPA_Input_IniXMat(Problem1, 1, 1, 1, 11);
        SDPA_Input_IniXMat(Problem1, 1, 2, 2, 9);

//      x^0 = {0.0, -4.0, 0.0}
        SDPA_Input_InixVec(Problem1, 2, -4);

//      Y^0 = { {5.9,  -1.375}, {-1.375, 1.0} }
        SDPA_Input_IniYMat(Problem1, 1, 1, 1, 5.9);
        SDPA_Input_IniYMat(Problem1, 1, 1, 2, -1.375);
        SDPA_Input_IniYMat(Problem1, 1, 2, 2, 1);
```

Here the call `SDPA_Input_IniXMat(Problem1, 1, 2, 2, 9)` means that the (**2,2**) element of the **1** st block of the matrix $\boldsymbol{X}^0$ is **9**, `SDPA_Input_IniYMat(Problem1, 1, 1, 2, -1.375)` means that the (**1,2**) element of the **1** st block of the matrix $\boldsymbol{Y}^0$ is **-1.375**.

We are now ready to put the call to the solver in the calling routine `SDPA_Solve(SDPA &)`. In this case, we output the total number of iteration, the primal objective function value, the dual objective function value and other final informations on the display.

```
        SDPA_Solve(Problem1);


        fprintf(stdout, "\nStop iteration = %d\n", Problem1.Iteration);
        fprintf(stdout, "objValPrimal   = %10.6e\n", Problem1.PrimalObj);
        fprintf(stdout, "objValDual     = %10.6e\n", Problem1.DualObj);
        fprintf(stdout, "p. feas. error = %10.6e\n", Problem1.PrimalError);
        fprintf(stdout, "d. feas. error = %10.6e\n\n", Problem1.DualError);
```

Next, we output the final solution $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ on the display. If we need $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ in our program, we convert the following procedures.

```
        printf("\nxVec = \n");
//      Problem1.xVec.display(stdout);

        mRow     = Problem1.xVec.nDim;
        element2 = Problem1.xVec.element;
        VectorDisplay(mRow, element2, stdout);


        printf("\nxMat = \n");
//      Problem1.XMat.display(stdout);

        if (Problem1.XMat.nBlock > 1) fprintf(stdout, "{\n");
        for(int k = 0; k < Problem1.XMat.nBlock; k++)
        {
                mRow    = Problem1.XMat.block[k].mRow;
                nCol    = Problem1.XMat.block[k].nCol;
                element = Problem1.XMat.block[k].element;
```

```
                MatrixDisplay(mRow, nCol, element, stdout);
        }
        if (Problem1.XMat.nBlock > 1) fprintf(stdout, "}\n\n");


        printf("\nxMat = \n");
//      Problem1.YMat.display(stdout);

        if (Problem1.YMat.nBlock > 1) fprintf(stdout, "{\n");
        for(int k = 0; k < Problem1.YMat.nBlock; k++)
        {
                mRow    = Problem1.YMat.block[k].mRow;
                nCol    = Problem1.YMat.block[k].nCol;
                element = Problem1.YMat.block[k].element;
                MatrixDisplay(mRow, nCol, element, stdout);
        }
        if (Problem1.YMat.nBlock > 1) fprintf(stdout, "}\n\n");
```

Finally, one closes all used file pointers and free a object `Problem1` from the computational memory space by calling the function `Delete()`.

```
        Problem1.Delete();
```

See also "example2-2.cpp", which generates a problem corresponds to **example2.dat** ans solves this by calling the functions.

References

[1] F. Alizadeh, J. -P. A. Haeberly and M. L. Overton, "Primal-dual interior-point methods for semidefinite programming," Working Paper, 1994.

[2] F. Alizadeh, J. -P. A. Haeberly and M. L. Overton, "Primal-dual interior point methods for semidefinite programming: convergence rates, stability and numerical results," Report 721, Computer Science Department, New York University, New York, NY, 1966.

[3] K. Fujisawa, M. Kojima and K. Nakata, "Exploiting Sparsity in Primal-Dual Interior-Point Methods for Semidefinite Programming," *Mathematical Programming* **79** (1997) 235–253.

[4] C. Helmberg, F. Rendl, R. J. Vanderbei and H. Wolkowicz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization* **6** (1996) 342–361.

[5] M. Kojima, S. Shindoh and S. Hara, "Interior-point methods for the monotone semidefinite linear complementarity problems," Research Report #282, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152, Japan, April 1994, Revised October 1995. To appear in *SIAM Journal on Optimization*.

[6] S.Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on Optimization* **2** (1992) 575–601.

[7] R.D.C. Monteiro, "Primal-dual path following algorithms for semidefinite programming," Working Paper, School Industrial and Systems Engineering, Georgia Tech., Atlanta, GA 30332, September 1995. To appear in *SIAM Journal on Optimization*.

[8] Ju. E. Nesterov and M. J. Todd, "Self-scaled cones and interior-point methods in nonlinear programming," Working Paper, CORE, Catholic University of Louvain, Louvain-la-Neuve, Belgium, April 1994.

[9] D. E. Stewart and Z. leyd, *Meschach:* Matrix Computation in C," Proceedings of the Center for Mathematics and Its Applications, The Australian National University, Volume 32, 1994.

[10] M. J. Todd, K. C. Toh and R. H. Tütüncü, "On the Nesterov-Todd direction in semidefinite programming," Technical Report, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853-3801, USA, 1996.