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 6.00

Katsuki Fujisawa†, Masakazu Kojima⋆, Kazuhide Nakata♯, Makoto Yamashita‡

July 2002

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 *LAPACK* [3] for matrix computation. The SDPA incorporates dynamic memory allocation and deallocation. So, the maximum size of an SDP that can be solved depends on the size of computational memory which user's computer loads. The SDPA version 6.00 enjoys the following features:

- Callable library of the SDPA is available.
- Efficient method for computing the search directions when an SDP to be solved is large scale and sparse [5].
- Block diagonal matrix structure and sparse matrix structure in data matrices are available. 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 WWW site

http://www.is.titech.ac.jp/~kojima/sdpa/index.html

Read the "index.html" file there for more details on how to get and install 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:knakata@me.titech.ac.jp

‡ e-mail:Makoto.Yamashita@is.titech.ac.jp

Preface

We are pleased to release a new version, Version 6.00, of the SDPA.

One of the significant points of this version of the SDPA is that it solves semidefinite programming problems much faster than the previous version of the SDPA without losing the excellent numerical stability. It incorporates the LAPACK libraries [3] for dense matrix computation. These libraries have further enhanced the SDPA performance, particularly, in the aspect of speed. The differences between this version and the previous version, Version 5.01, are summarized in Section 11.

We hope that the SDPA supports many researches in various fields. We also welcome any suggestions and comments that you may have. When you want to contact us, please send e-mail to `kojima-sdpa@is.titech.ac.jp` .

Contents

1. Installation. **1**

2. Semidefinite Program. **2**

2.1. Standard Form SDP and Its Dual. 2

2.2. Example 1. 3

2.3. Example 2. 3

3. Files Necessary to Execute the SDPA. **4**

4. Input Date File. **5**

4.1. "example1.dat" — Input Data File of Example 1. 5

4.2. "example2.dat" — Input Data File of Example 2. 5

4.3. Format of Input Data File. 6

4.4. Title and Comment. 6

4.5. The Number of the Primal Variables. 6

4.6. The Number of the Blocks and the Block Structure Vector. 7

4.7. Constant Vector 8

4.8. Constraint Matrices. 8

5. Parameter File. **10**

6. Output. **11**

6.1. Execution of the SDPA. 11

6.2. Output on the Display. 12

6.3. Output to a File. 15

7. Advanced Use of the SDPA. **17**

7.1. Initial Point. 17

7.2. Sparse Input Data File. 18

7.3. Sparse Initial Point File. 19

7.4. More on Parameter File. 19

8. The SDPA Callable Library **20**

8.1. Case 1: 20

8.2. Case 2: 22

9. Transformation to the Standard Form of SDP **31**

9.1. Inequality Constraints 31

9.2. Norm Minimization Problem 32

9.3. Linear Matrix Inequality (LMI) 33

9.4. SDP Relaxation of the Maximum Cut Problem 33

10. Quick Reference **34**

10.1. Variables of the SDPA Class 34

10.2. Variable of Parameter Structure 35

10.3. Methods of the SDPA Class 36

10.4. Functions Access to the SDPA Class 37

10.5. Stand Alone Executable binary 39

11. For the SDPA 5.01 Users **40**

11.1. Option for Executable Binary 40

11.2. Update of Callable Library Interface 40

1. Installation.

The SDPA package is available at the following WWW site:

http://www.is.titech.ac.jp/~kojima/sdpa/index.html

After reading the **index.html** file, one can down-load a source file and a install manuaul of the latest version of SDPA.

We assume that the SDPA is installed in the subdirectory **sdpa** where we can find the following files:

sdpa_doc.ps	A user's manual.
sdpa	An executable binary, which solves 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.
libsdpa.a	A callable library of the SDPA.
Makefile	A makefile to compile source files.
example1-1.cpp, example1-2.cpp	Sample source files for utilizing the callable
example2-1.cpp, example2-2.cpp	library.
example3.cpp, example4.cpp	
example5.cpp, example6.cpp	
sdpa-lib.hpp, sdpa-lib2.hpp,	Header files for utilizing the callable
	library **libsdpa.a**.

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

```
$ ./sdpa
SDPA start ... (built at Jun 12 2002 15:47:57)

*Please assign data file and output file.*

---- option type 1 ------------
./sdpa DataFile OutputFile [InitialPtFile] [-pt parameters]
example1-1: ./sdpa example1.dat example1.result
example1-2: ./sdpa example1.dat-s example1.result
example1-3: ./sdpa example1.dat example1.result example1.ini

---- option type 2 ------------
./sdpa [option filename]+
  -dd : data dense :: -ds : data sparse
  -id : init dense :: -is : init sparse
  -o  : output     :: -p  : parameter
example2-1: ./sdpa -o example1.result -dd example1.dat
example2-2: ./sdpa -ds example1.dat-s -o example2.result -p param.sdpa
```

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

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

$\mathcal{S}$: 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

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

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

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

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 2.2.) of this problem is contained in the file "example1.dat".

2.3. Example 2.

$$
\begin{aligned}
m &= 5, \ n = 7, \ \boldsymbol{c} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix} = \begin{pmatrix} 1.1 \\ -10 \\ 6.6 \\ 19 \\ 4.1 \end{pmatrix}, \\
\boldsymbol{F}_0 &= \begin{pmatrix}
-1.4 & -3.2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
-3.2 & -28 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 15 & -12 & 2.1 & 0.0 & 0.0 \\
0.0 & 0.0 & -12 & 16 & -3.8 & 0.0 & 0.0 \\
0.0 & 0.0 & 2.1 & -3.8 & 15 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.8 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -4.0
\end{pmatrix},
\end{aligned}
$$

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

$$\vdots$$

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

As shown in this example, the SDPA handles block diagonal matrices. The data (see Section 2.3.) 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. 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 7.2. for details.
- "**param.sdpa**" — A file describing the parameters used in the "sdpa". See Section 5. 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 4.1.) and "example2.dat" (see Section 4.2.) 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 `*` 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 2.3.. 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:

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

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

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

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

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

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 4.1.), and

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

in Example 2 (see the file "example2.dat" in Section 4.2.). 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

, ( ) { }

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

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

in Example 1 (see the file "example1.dat" in Section 4.1.), and

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

in Example 2 (see the file "example2.dat" in Section 4.2.).

4.8. Constraint Matrices.

According to the format with the use of nBLOCK and bLOCKsTRUCT stated in Section 4.6., we describe the constraint matrices $\boldsymbol{F}_0$, $\boldsymbol{F}_1$, ..., $\boldsymbol{F}_m$. In addition to blank letter(s) and tab code(s), we can use the letters

, ( ) { }

to separate elements of the matrices $\boldsymbol{F}_0$, $\boldsymbol{F}_1$, ..., $\boldsymbol{F}_m$ and their elements. In the general case of the block diagonal matrix $\boldsymbol{F}$ given in (1), we write the elements of $\boldsymbol{B}_1$, $\boldsymbol{B}_2$, ... , $\boldsymbol{B}_\ell$ 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 4.1..

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 4.2..

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 < omegaStar;
-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

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

  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 start ... (built at Jun 11 2002 20:14:48)
data      is example1.dat : dense
parameter is ./param.sdpa
out       is example1.result
initial   is example1.ini : dense
   mu      thetaP  thetaD  objP      objD      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 2.00e-01
 1 1.6e+03 0.0e+00 9.4e-02 +8.39e+02 +7.51e+01 2.3e+00 9.6e-01 2.00e-01
 2 1.7e+02 0.0e+00 3.6e-03 +1.96e+02 -3.74e+01 1.3e+00 1.0e+00 2.00e-01
 3 1.8e+01 0.0e+00 0.0e+00 -6.84e+00 -4.19e+01 9.9e-01 9.0e+01 1.00e-01
 4 1.9e+00 0.0e+00 0.0e+00 -3.81e+01 -4.19e+01 1.0e-00 9.0e+01 1.00e-01
 5 1.9e-01 0.0e+00 0.0e+00 -4.15e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
 6 1.9e-02 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e-00 9.0e+01 1.00e-01
 7 1.9e-03 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
 8 1.9e-04 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e-00 9.0e+01 1.00e-01
 9 1.9e-05 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
10 1.9e-06 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01

phase.value = pdOPT
   Iteration = 10
          mu = 1.9180668442024166e-06
relative gap = 9.1554458604013074e-08
         gap = 3.8361336884048333e-06
      digits = 7.0383205011714587e+00
objValPrimal = -4.1899996163866355e+01
objValDual   = -4.1899999999997995e+01
p.feas.error = 1.0658141036401503e-14
d.feas.error = 8.5975671026972122e-13
total time   = 0.000
  main loop time = 0.000000
      total time = 0.000000
file   read time = 0.000000
```

- 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.

- objP — The primal objective function value.
- objD — 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 λ_{ave} and $\lambda_{\min}$ 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

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

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

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

- Iteration — The iteration number which the SDPA needs to terminate.
- relative gap — The relative gap means that

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

  This value is compared with **epsilonStar** (Section 5.).
- gap — The gap means that mu $\times\, n$.
- digits — This value indicates how objP and objD resemble by the following definition.

$$\begin{aligned} \text{digits} &= -\log_{10} \frac{|\text{objP} - \text{objD}|}{(|\text{objP}| + |\text{objD}|)/2.0} \\ &= -\log_{10} \frac{|\sum_{i=1}^{m} c_i x_i{}^* - \boldsymbol{F}_0 \bullet \boldsymbol{Y}|}{(|\sum_{i=1}^{m} c_i x_i{}^*| + |\boldsymbol{F}_0 \bullet \boldsymbol{Y}|)/2.0} \end{aligned}$$

- objValPrimal — The primal objective function value.

$$\text{objValPrimal} = \sum_{i=1}^{m} c_i x_i.$$

- objValDual — The dual objective function value.

$$\text{objValD} = \boldsymbol{F}_0 \bullet \boldsymbol{Y}.$$

- p.feas.error — This value is the primal infeasibily in the last iteration,

$$\text{p.feas.error} = \max\left\{ \left| [\sum_{i=1}^{m} \boldsymbol{F}_i x_i + \boldsymbol{Y} - \boldsymbol{F}0]_{p,q} \right| \ :\ p, q = 1, 2, \ldots, n \right\}$$

  This value is compared with **epsilonDash** (Section 5.). Even if primal is feasible, this value may not be 0 since numerical error.

- d.feas.error — This value is the dual infeasibily in the last iteration,

$$\text{d.feas.error} = \max\{|\boldsymbol{F}_i \bullet \boldsymbol{Y} - c_i| \ : i = 1, 2, \ldots, m\}.$$

  This value is compared with **epsilonDash** (Section 5.). Even if dual is feasible, this value may not be 0 since numerical error.

- total time — This value is how much time the SDPA needs to execute all algorithms.

- main loop time —This value is how much time the SDPA needs between the first iteration and the last iteration.

- file read time — This value is how much time the SDPA needs to read from file and makes a structure in memory.

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.

```
*Example 2:
*mDim = 5, nBLOCK = 3, {2,3,-2}
data      is example2.dat
parameter is ./param.sdpa
initial   is (null)
out       is r
   mu      thetaP  thetaD  objP      objD      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 2.00e-01
 1 3.3e+03 1.2e-01 3.4e-01 +4.94e+02 +2.84e+02 1.0e+00 8.2e-01 2.00e-01
 2 9.0e+02 0.0e+00 6.2e-02 +8.66e+02 -2.60e+00 1.0e+00 1.0e+00 2.00e-01
 3 1.4e+02 0.0e+00 0.0e+00 +9.67e+02 +9.12e-01 9.5e-01 5.4e+00 1.00e-01
 4 1.8e+01 0.0e+00 0.0e+00 +1.46e+02 +2.36e+01 9.3e-01 1.5e+00 1.00e-01
 5 2.7e+00 0.0e+00 0.0e+00 +4.62e+01 +2.74e+01 8.1e-01 1.4e+00 1.00e-01
 6 5.7e-01 0.0e+00 0.0e+00 +3.43e+01 +3.03e+01 9.2e-01 9.3e-01 1.00e-01
 7 9.5e-02 0.0e+00 0.0e+00 +3.24e+01 +3.18e+01 9.5e-01 9.6e-01 1.00e-01
 8 1.3e-02 0.0e+00 0.0e+00 +3.21e+01 +3.20e+01 9.8e-01 1.0e-00 1.00e-01
 9 1.5e-03 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 1.00e-01
10 1.5e-04 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 1.00e-01
11 1.5e-05 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 1.00e-01
12 1.5e-06 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 1.00e-01
13 1.5e-07 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 1.00e-01

phase.value = pdOPT
   Iteration = 13
          mu = 1.5017857316949013e-07
relative gap = 3.2787297290713969e-08
         gap = 1.0512500121864308e-06
      digits = 7.4842943814430383e+00
objValPrimal = 3.2062693405e+01
objValDual   = 3.2062692354e+01
```

```
p.feas.error = 5.4012350148e-14
d.feas.error = 1.6875389974e-12
total time   = 0.010

Parameters are
maxIteration =    40
epsilonStar  = 1.000e-07
lambdaStar   = 1.000e+02
omegaStar    = 2.000e+00
lowerBound   = -1.000e+05
upperBound   = 1.000e+05
betaStar     = 1.000e-01
betaBar      = 2.000e-01
gammaStar    = 9.000e-01
epsilonDash  = 1.000e-07
                        Time(sec)  Ratio(% : MainLoop)
 Predictor time  =      -0.000000,  -0.000000
        ... abbreviation ...
 Total           =      0.010000,  100.000000

xVec =
{+1.552e+00,+6.710e-01,+9.815e-01,+1.407e+00,+9.422e-01}
xMat =
{
{ {+6.392e-08,-9.638e-09 },
  {-9.638e-09,+4.539e-08 }   }
{ {+7.119e+00,+5.025e+00,+1.916e+00 },
  {+5.025e+00,+4.415e+00,+2.506e+00 },
  {+1.916e+00,+2.506e+00,+2.048e+00 }   }
{+3.432e-01,+4.391e+00}
}
yMat =
{
{ {+2.640e+00,+5.606e-01 },
  {+5.606e-01,+3.718e+00 }   }
{ {+7.616e-01,-1.514e+00,+1.139e+00 },
  {-1.514e+00,+3.008e+00,-2.264e+00 },
  {+1.139e+00,-2.264e+00,+1.705e+00 }   }
{+4.087e-07,+3.195e-08}
}
    main loop time = 0.010000
        total time = 0.010000
  file   read time = 0.000000
```

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

- Lines with start '*' — These lines are comment in "example2.dat".
- data, parameter, initial, output — These are file name we assign for data, parameter, initial point, and output , respectively.

- Lines between 'predictor time' to 'Total time' — These lines display the profile data. These information may help us tune up parametes, but the details is too complicated, because the profile data seriously depends on internal algorithms.
- xVec — The primal variable vector $\boldsymbol{x}$.
- xMat — The primal variable matrix $\boldsymbol{X}$.
- yMat — The dual variable matrix $\boldsymbol{Y}$.

7. Advanced Use of the SDPA.

7.1. 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.

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 4.7.), and the description of $\boldsymbol{X}^0$ and $\boldsymbol{Y}^0$ the same format as the constraint matrix $\boldsymbol{F}_i$ (see Section 4.8.).

7.2. Sparse Input Data File.

In Section 4., 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 2.2.) 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 4.1. 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
$\ldots$
k_p b_p i_p j_p v_p
$\ldots$
k_q b_q i_q j_q v_q

Here $k_p \in \{0, 1, \ldots, m\}$, $b_p \in \{1, 2, \ldots,$ nBLOCK$\}$, $1 \leq i_p \leq j_p$ 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 $1 \leq i_p \leq j_p \leq \ell$; 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 $1 \leq i_p = j_p \leq \ell$.

7.3. 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 7.1. 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:

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

Here $s_p = 1$ or 2, $b_p \in \{1, 2, \ldots, \mathrm{nBLOCK}\}$, $1 \leq i_p \leq j_p$ 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 $1 \leq i_p \leq j_p \leq \ell$; 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 $1 \leq i_p = j_p \leq \ell$.

7.4. 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 SDPA Callable Library

The easiest way to understand how to use the SDPA callable library is to look at example files. For this purpose, we have chosen a problem "Example1" in Section 4.1. Here we 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 3., 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, we type:

```
$ g++ -O3 -I$(HOME)/sdpa -I$(HOME)/lapack/include example1-1.cpp
$ g++ -O3 -o example1-1.exe example1-1.o -L$(HOME)/sdpa -lsdpa \
-L$(HOME)/lapack/lib -llapack -lcblaswr -lcblas -lf77blas -lI77 -lF77 -latlas
$ ./example1-1.exe example1.dat example1.out
```

In the above command, we assume that we have installed ATLAS and LAPACK header file into `$(HOME)/lapack/include` and library file into `$(HOME)/lapack/lib`, and SDPA into `$(HOME)/sdpa`.

```
/* 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;

        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, we need 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 need to declare an object (variable), what we call `Problem1`, that is a object to the class `SDPA` in the header file **sdpa-lib2.hpp**.

```
SDPA    Problem1;
```

The next procedures are very important and one we 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");
```

We 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 we must copy the file names to `ParameterFileName`, `InputFileName` and `OutputFileName`, respectively.

If we 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, we close 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:

```
$ g++ -O3 -I$(HOME)/sdpa -I$(HOME)/lapack/include example1-1.cpp
$ g++ -O3 -o example2-1.exe example2-1.o -L$(HOME)/sdpa -lsdpa \
```

```
-L$(HOME)/lapack/lib -llapack -lcblaswr -lcblas -lf77blas -lI77 -lF77 -latlas
$ ./example2-1.exe
```

In the above command, we assume that we have installed ATLAS and LAPACK header file into `$(HOME)/lapack/include` and library file into `$(HOME)/lapack/lib`, and the SDPA into `$(HOME)/sdpa`.

```
/* 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} }
*/

void PrintMatrix(int nRow, int nCol, double* element, FILE* fpOut);
void PrintVector(int nDim, double* element, FILE* fpOut);

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


        SDPA    Problem1;

        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);
        fprintf(stdout, "xVec = \n");
        // Problem1.printResultXVec(stdout);
        element = Problem1.getResultXVec();
        PrintVector(Problem1.mDIM,element,stdout);

        fprintf(stdout, "xMat = \n");
        // Problem1.printResultXMat(stdout);
        fprintf(stdout, "{\n");
        for (int l=0; l<Problem1.nBLOCK; ++l) {
          element = Problem1.getResultXMat(l);
          int nRow = Problem1.bLOCKsTRUCT[l];
          int nCol = nRow;
          if (nRow<0) {
            nCol = -nCol;
            nRow = 1;
          }
          PrintMatrix(nRow,nCol,element,stdout);
        }
        fprintf(stdout, "}\n");
```

```
        fprintf(stdout, "yMat = \n");
        // Problem1.printResultYMat(stdout);

        fprintf(stdout, "{\n");
        for (int l=0; l<Problem1.nBLOCK; ++l) {
          element = Problem1.getResultYMat(l);
          int nRow = Problem1.bLOCKsTRUCT[l];
          int nCol = nRow;
          if (nRow<0) {
            nCol = -nCol;
            nRow = 1;
          }
          PrintMatrix(nRow,nCol,element,stdout);
        }
        fprintf(stdout, "}\n");

        Problem1.Delete();

        exit(0);
};

#define FORMAT "%+8.16e"

void PrintMatrix(int nRow, int nCol, double* element, FILE* fpOut)
{
  fprintf(fpOut,"{");
  for (int i=1; i<=nRow-1; ++i) {
    if (i==1) {
      fprintf(fpOut," ");
    } else {
      fprintf(fpOut,"  ");
    }
    fprintf(fpOut,"{");
    for (int j=1; j<=nCol-1; ++j) {
      fprintf(fpOut, FORMAT",",element[(i-1)+nRow*(j-1)]);
    }
    fprintf(fpOut,FORMAT" },\n",element[(i-1)+nRow*(nCol-1)]);
  }
  if (nRow>1) {
    fprintf(fpOut,"  {");
  }
  for (int j=1; j<=nCol-1; ++j) {
    fprintf(fpOut,FORMAT",",element[(nRow-1)+nRow*(j-1)]);
  }
    fprintf(fpOut,FORMAT" },\n",element[(i-1)+nRow*(nCol-1)]);
  }
  if (nRow>1) {
    fprintf(fpOut,"  {");
```

```
  }
  for (int j=1; j<=nCol-1; ++j) {
    fprintf(fpOut,FORMAT",",element[(nRow-1)+nRow*(j-1)]);
  }
  fprintf(fpOut,FORMAT" }",element[(nRow-1)+nRow*(nCol-1)]);
  if (nRow>1) {
    fprintf(fpOut,"   }\n");
  } else {
    fprintf(fpOut,"\n");
  }
}

void PrintVector(int nDim, double* element, FILE* fpOut)
{
  fprintf(fpOut,"{");
  for (int j=1; j<=nDim-1; ++j) {
    fprintf(fpOut,FORMAT",",element[j-1]);
  }
  if (nDim>0) {
    fprintf(fpOut,FORMAT"}\n",element[nDim-1]);
  } else {
    fprintf(fpOut,"  }\n");
  }
}

/* The end of the ‘‘example2-1.cpp’’. */
```

We need four header files and must declare a object, say `Problem1` like `Case 1`. For example, if we 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.InitialPoint           = true;
```

In this case, we 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. We 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 4.6.. We can also implement the declaration of `bLOCKsTRUCT` as follows:

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

We must pay attention to 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 we need, the initial point must 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 call to the SDPA solver in the calling routine `SDPA_Solve(SDPA &)`. In case below, 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.

```
fprintf(stdout, "xVec = \n");
// Problem1.printResultXVec(stdout);
element = Problem1.getResultXVec();
PrintVector(Problem1.mDIM,element,stdout);

fprintf(stdout, "xMat = \n");
// Problem1.printResultXMat(stdout);
fprintf(stdout, "{\n");
for (int l=0; l<Problem1.nBLOCK; ++l) {
  element = Problem1.getResultXMat(l);
  int nRow = Problem1.bLOCKsTRUCT[l];
  int nCol = nRow;
  if (nRow<0) {
    nCol = -nCol;
    nRow = 1;
  }
  PrintMatrix(nRow,nCol,element,stdout);
}
fprintf(stdout, "}\n");


fprintf(stdout, "yMat = \n");
// Problem1.printResultYMat(stdout);

fprintf(stdout, "{\n");
for (int l=0; l<Problem1.nBLOCK; ++l) {
  element = Problem1.getResultYMat(l);
  int nRow = Problem1.bLOCKsTRUCT[l];
  int nCol = nRow;
  if (nRow<0) {
    nCol = -nCol;
    nRow = 1;
  }
  PrintMatrix(nRow,nCol,element,stdout);
}
fprintf(stdout, "}\n");
```

Finally, we close 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.

9. Transformation to the Standard Form of SDP

SDP has many applications. But some problems may not be formulated as the standard form of SDP. In this section, we show how to transform to the standard form of SDP.

9.1. Inequality Constraints

First, we consider the case where inequality constraints are added to the primal standard form of SDP. For example,

$$\begin{cases} \text{minimize} & \displaystyle\sum_{i=1}^{m} c_i x_i \\ \text{subject to} & \boldsymbol{X} = \displaystyle\sum_{i=1}^{m} \boldsymbol{F}_i x_i - \boldsymbol{F}_0,\ \boldsymbol{X} \succeq \boldsymbol{O}, \\ & \displaystyle\sum_{i=1}^{m} \alpha_i^1 x_i \le \beta^1, \quad \sum_{i=1}^{m} \alpha_i^2 x_i \ge \beta^2 \end{cases}$$

Here, $\alpha_i^1, \alpha_i^2 (i = 1, \ldots, m), \beta^1, \beta^2 \in R$. In this case, we add slack variables(t^1, t^2).

$$\begin{cases} \text{minimize} & \displaystyle\sum_{i=1}^{m} c_i x_i \\ \text{subject to} & \boldsymbol{X} = \displaystyle\sum_{i=1}^{m} \boldsymbol{F}_i x_i - \boldsymbol{F}_0,\ \boldsymbol{X} \succeq \boldsymbol{O}, \\ & t^1 = \displaystyle\sum_{i=1}^{m} (-\alpha_i^1) x_i - (-\beta^1), \quad t^2 = \sum_{i=1}^{m} \alpha_i^2 x_i - \beta^2, \quad (t^1, t^2) \ge 0. \end{cases}$$

Hence we can reduce the above problem to the following standard form SDP.

$$\begin{cases} \text{maximize} & \displaystyle\sum_{i=1}^{m} c_i x_i \\ \text{subject to} & \bar{\boldsymbol{X}} = \displaystyle\sum_{i=1}^{m} \bar{\boldsymbol{F}}_i x_i - \bar{\boldsymbol{F}}_0,\ \bar{\boldsymbol{X}} \succeq \boldsymbol{O}, \end{cases}$$

where

$$\bar{\boldsymbol{F}}_i = \begin{pmatrix} \boldsymbol{F}_i & & \\ & -\alpha_i^1 & \\ & & \alpha_i^2 \end{pmatrix}, \bar{\boldsymbol{F}}_0 = \begin{pmatrix} \boldsymbol{F}_0 & & \\ & -\beta^1 & \\ & & \beta^2 \end{pmatrix}, \bar{\boldsymbol{X}} = \begin{pmatrix} \boldsymbol{X} & & \\ & t^1 & \\ & & t^2 \end{pmatrix},$$

$\text{nBlock} = 2, \text{blockStruct} = (n, -2)$.

Next, we consider the case where inequality constraints are added to the primal standard form of SDP. For example,

$$\left\{\begin{array}{ll} \text{maximize} & \boldsymbol{F}_0 \bullet \boldsymbol{Y} \\ \text{subject to} & \boldsymbol{F}_1 \bullet \boldsymbol{Y} = c_1, \boldsymbol{F}_2 \bullet \boldsymbol{Y} = c_2, \\ & \boldsymbol{F}_3 \bullet \boldsymbol{Y} \leq c_3, \boldsymbol{F}_4 \bullet \boldsymbol{Y} \leq c_4, \boldsymbol{F}_5 \bullet \boldsymbol{Y} \geq c_5, \\ & \boldsymbol{Y} \succeq \boldsymbol{O}. \end{array}\right.$$

In this case, we add slack variables(t_3, t_4, t_5) to each inequality constraint.

$$\left\{\begin{array}{ll} \text{minimize} & \boldsymbol{F}_0 \bullet \boldsymbol{Y} \\ \text{subject to} & \boldsymbol{F}_1 \bullet \boldsymbol{Y} = c_1, \boldsymbol{F}_2 \bullet \boldsymbol{Y} = c_2, \\ & \boldsymbol{F}_3 \bullet \boldsymbol{Y} + t_3 = c_3, \boldsymbol{F}_4 \bullet \boldsymbol{Y} + t_4 = c_4, \boldsymbol{F}_5 \bullet \boldsymbol{X} - t_5 = c_5, \\ & \boldsymbol{Y} \succeq \boldsymbol{O}. \quad (t_3, t_4, t_5) \geq 0. \end{array}\right.$$

Thus we can reduce the above problem to the following standard form SDP.

$$\left\{\begin{array}{ll} \text{minimize} & \bar{\boldsymbol{F}}_0 \bullet \bar{\boldsymbol{Y}} \\ \text{subject to} & \bar{\boldsymbol{F}}_1 \bullet \bar{\boldsymbol{Y}} = c_1, \bar{\boldsymbol{F}}_2 \bullet \bar{\boldsymbol{Y}} = c_2, \\ & \bar{\boldsymbol{F}}_3 \bullet \bar{\boldsymbol{Y}} = c_3, \bar{\boldsymbol{F}}_4 \bullet \bar{\boldsymbol{Y}} = c_4, \bar{\boldsymbol{F}}_5 \bullet \bar{\boldsymbol{Y}} = c_5, \\ & \bar{\boldsymbol{Y}} \succeq \boldsymbol{O}, \end{array}\right.$$

where

$$\bar{\boldsymbol{Y}} = \begin{pmatrix} \boldsymbol{Y} & & & \\ & t_3 & & \\ & & t_4 & \\ & & & t_5 \end{pmatrix}, \bar{\boldsymbol{F}}_0 = \begin{pmatrix} \boldsymbol{F}_0 & & & \\ & 0 & & \\ & & 0 & \\ & & & 0 \end{pmatrix}, \bar{\boldsymbol{F}}_1 = \begin{pmatrix} \boldsymbol{F}_1 & & & \\ & 0 & & \\ & & 0 & \\ & & & 0 \end{pmatrix},$$

$$\bar{\boldsymbol{F}}_2 = \begin{pmatrix} \boldsymbol{F}_2 & & & \\ & 0 & & \\ & & 0 & \\ & & & 0 \end{pmatrix}, \bar{\boldsymbol{F}}_3 = \begin{pmatrix} \boldsymbol{F}_3 & & & \\ & 1 & & \\ & & 0 & \\ & & & 0 \end{pmatrix}, \bar{\boldsymbol{F}}_4 = \begin{pmatrix} \boldsymbol{F}_4 & & & \\ & 0 & & \\ & & 1 & \\ & & & 0 \end{pmatrix},$$

$$\bar{\boldsymbol{F}}_5 = \begin{pmatrix} \boldsymbol{F}_5 & & & \\ & 0 & & \\ & & 0 & \\ & & & -1 \end{pmatrix},$$

nBlock $= 2$, blockStruct $= (n, -3)$.

9.2. Norm Minimization Problem

Let $\boldsymbol{G}_i \in R^{q \times r}$ $(0 \leq i \leq p)$. The norm minimization problem is defined as:

$$\begin{array}{ll} \text{minimize} & \left\| \boldsymbol{G}_0 + \sum_{i=1}^{p} \boldsymbol{G}_i x_i \right\| \\ \text{subject to} & x_i \in R \ (1 \leq i \leq p). \end{array}$$

Here $\|\boldsymbol{G}\|$ denotes the 2-norm of $\boldsymbol{G}$, *i.e.*,

$$\|\boldsymbol{G}\| = \max_{\|\boldsymbol{u}\|=1} \|\boldsymbol{G}\boldsymbol{u}\| = \text{the square root of the maximum eigenvalue of } \boldsymbol{G}^T\boldsymbol{G}.$$

We can reduce this problem to an SDP:

$$\begin{array}{ll} \text{minimize} & x_{p+1} \\ \text{subject to} & \displaystyle\sum_{i=1}^{p} \begin{pmatrix} \boldsymbol{O} & \boldsymbol{G}_i^T \\ \boldsymbol{G}_i & \boldsymbol{O} \end{pmatrix} x_i + \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{pmatrix} x_{p+1} + \begin{pmatrix} \boldsymbol{O} & \boldsymbol{G}_0^T \\ \boldsymbol{G}_0 & \boldsymbol{O} \end{pmatrix} \succeq \boldsymbol{O}. \end{array}$$

Thus if we take

$$\begin{aligned} m &= p+1,\ n = r+q,\ \boldsymbol{F}_0 = \begin{pmatrix} \boldsymbol{O} & -\boldsymbol{G}_0^T \\ -\boldsymbol{G}_0 & \boldsymbol{O} \end{pmatrix}, \\ \boldsymbol{F}_i &= \begin{pmatrix} \boldsymbol{O} & \boldsymbol{G}_i^T \\ \boldsymbol{G}_i & \boldsymbol{O} \end{pmatrix},\ c_i = 0\ (1 \le i \le p), \\ \boldsymbol{F}_{p+1} &= \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{pmatrix},\ c_{p+1} = 1, \end{aligned}$$

then we can reformulate the problem as the primal standard form of SDP.

9.3. Linear Matrix Inequality (LMI)

Let $\boldsymbol{G}_i \in \mathcal{S}^n$ $(0 \le i \le p)$. We define the linear combinations with matrices,

$$\boldsymbol{G}(\boldsymbol{x}) = \boldsymbol{G}_0 + \sum_{i=1}^{p} x_i \boldsymbol{G}_i,$$

where $\boldsymbol{x} \in R^p$. The Linear Matrix Inequality (LMI) [4] is defined as the following inequality,

$$\boldsymbol{G}(\boldsymbol{x}) \succeq \boldsymbol{O}.$$

We want to find $\boldsymbol{x} \in R^p$ which satisfies the LMI, or detect any $\boldsymbol{x} \in R^p$ cannot satisfy the LMI. Here we introduce an auxiliary variable x_{p+1} and convert the LMI into a SDP.

$$\begin{array}{ll} \text{minimize} & x_{p+1} \\ \text{subject to} & \boldsymbol{X} = \boldsymbol{G}_0 + \displaystyle\sum_{i=1}^{p} x_i \boldsymbol{G}_i - x_{p+1}\boldsymbol{I} \ \succeq \boldsymbol{O}, \quad \boldsymbol{x} \in R^p \end{array}$$

We can reduce LMI to the primal standard form of SDP if we take

$$m = p+1, \boldsymbol{F}_0 = -\boldsymbol{G}_0, \boldsymbol{F}_i = \boldsymbol{G}_i, c_i = 0\ (1 \le i \le p), \boldsymbol{F}_{p+1} = \boldsymbol{I}, c_{p+1} = 1,$$

One important point of this SDP is that it always has feasible solution. When we can get the optimal solution with $x_{p+1} \ge 0$, then $(x_1, \ldots, x_p)$ satisfies the LMI. On the other hand, if $x_{p+1} < 0$ then we can conclude that the LMI cannot be satisfied by any $\boldsymbol{x} \in R^p$.

9.4. SDP Relaxation of the Maximum Cut Problem

Let $G = (V, E)$ be a complete undirected graph with a vertex set $V = \{1, 2, \ldots, n\}$ and an edge set $E = \{(i, j) : i, j \in V,\ i < j\}$. We assign a weight $W_{ij} = W_{ji}$ to each edge $(i, j) \in E$. The maximum cut problem is to find a partition (L, R) of V that maximizes the *cut* $w(L, R) = \sum_{i \in L, j \in R} W_{ij}$.

Introducing a variable vector $\boldsymbol{u} \in R^n$, we can formulate the problem as a nonconvex quadratic program:

$$\text{maximize} \quad \frac{1}{4}\sum_i \sum_j W_{ij}(1 - u_i u_j) \text{ subject to} \quad u_i^2 = 1 \ (1 \leq i \leq n).$$

Here each feasible solution $\boldsymbol{u} \in R^n$ of this problem is corresponding to a cut (L, R) with $L = \{i \in V : u_i = -1\}$ and $R = \{i \in V : u_i = 1\}$. If we define $\boldsymbol{W}$ to be the $n \times n$ symmetric matrix with elements $W_{ji} = W_{ij}$ $((i,j) \in E)$ and $W_{ii} = 0$ $(1 \leq i \leq n)$, and the $n \times n$ symmetric matrix $\boldsymbol{C} \in \mathcal{S}^n$ by $\boldsymbol{C} = \frac{1}{4}(\text{diag}(\boldsymbol{W}\boldsymbol{e}) - \boldsymbol{W})$, where $\boldsymbol{e} \in R^n$ denotes the vector of ones and $\text{diag}(\boldsymbol{W}\boldsymbol{e})$ the diagonal matrix of the vector $\boldsymbol{W}\boldsymbol{e} \in R^n$, we can rewrite the quadratic program above as

$$\text{maximize} \quad \boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x} \text{ subject to} \quad x_i^2 = 1 \ (1 \leq i \leq n).$$

If $\boldsymbol{x} \in R^n$ is a feasible solution of the latter quadratic program, then the $n \times n$ symmetric and positive semidefinite matrix $\boldsymbol{X}$ whose (i,j)th element X_{ij} is given by $X_{ij} = x_i x_j$ satisfies $\boldsymbol{C} \bullet \boldsymbol{X} = \boldsymbol{x}^T\boldsymbol{C}\boldsymbol{x}$ and $X_{ii} = 1$ $(1 \leq i \leq n)$. This leads to the following semidefinite programming relaxation of the maximum cut problem:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{C} \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{E}_{ii} \bullet \boldsymbol{X} = 1 \ (1 \leq i \leq n), \ \boldsymbol{X} \succeq \boldsymbol{O}. \end{array}$$

Here $\boldsymbol{E}_{ii}$ denotes the $n \times n$ symmetric matrix with (i,i)th element 1 and all others 0. See [6]and references therein for more details.

10. Quick Reference

10.1. Variables of the SDPA Class

Type	Name	
int	mDIM	The number of primal variables.
int	nBLOCK	The number of blocks.
int*	bLOCKsTRUCT	The block structure vector.
bool	InitialPoint	Whether an initial point is set(true) or not(false).
bool	CheckMatrix	Whether the SDPA checks the input data (true) or not(false). It may need long time to check data.
FILE*	DisplayInformation	The file descriptor to where of the SDPA prints information. If this variables is set as NULL, the SDPA does not print anywhere.
char*	InputFileName	The file name of input file name. the SDPA checks the postfix of this name.
char*	OutputFileName	The file name of output file name.
char*	InitialFileName	The file name of input file name. the SDPA checks the postfix of this name.
char*	ParameterFileName	The file name of input file name.
FILE*	InputFile	The file descriptor from where the SDPA reads the data.
FILE*	OutputFile	The file descriptor to where the SDPA write informaiton.
FILE*	InitialFile	The file descriptor from where the SDPA reads an initial point.
FILE*	ParameterFile	The file descriptor from where the SDPA reads parameters.
——	pARAM	The structure which contains parameters. See the next subsection.

10.2. Variable of Parameter Structure

int	maxIteration	When the iteraition number of the SDPA exceeds this parameter, the SDPA terminates.
double	epsilonStar	When the relative gap becomes smaller than this value and both primal and dual are feasible, the SDPA terminates with optimal solution, that is, $$\text{epsilonStar} \geq \frac{\left\lvert \sum_{i=1}^{m} c_i x_i^k - \boldsymbol{F}_0 \bullet \boldsymbol{Y}^k \right\rvert}{\max\left\{ (\left\lvert \sum_{i=1}^{m} c_i x_i^k \right\rvert + \lvert \boldsymbol{F}_0 \bullet \boldsymbol{Y}^k \rvert)/2.0,\ 1.0 \right\}} = \frac{\lvert \text{objPrimal} - \text{objDual} \rvert}{\max\{(\lvert \text{objPrimal} \rvert + \lvert \text{objDual} \rvert)/2.0,\ 1.0\}},$$
double	epsilonDash	When the primal error(the dual error) becomes smaller than epsilonDash, the SDPA indicates primal feasible (dual feasible), that is, if $$\text{epsilonDash} \geq \max\left\{ \left\lvert [X^k - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^k + \boldsymbol{F}_0]_{pq} \right\rvert : p, q = 1, 2, \ldots, n \right\}$$ then primal feasible and if $$\text{epsilonDash} \geq \max\left\{ \left\lvert \boldsymbol{F}_i \bullet \boldsymbol{Y}^k - c_i \right\rvert : i = 1, 2, \ldots, m \right\}$$ then dual feasible.
double	lambdaStar	This parameter determines an initial point $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ such that $\boldsymbol{x}^0 = \boldsymbol{0}$, $\boldsymbol{X}^0 = \text{lambdaStar} \times \boldsymbol{I}$, $\boldsymbol{Y}^0 = \text{lambdaStar} \times \boldsymbol{I}$.
double	omegaStar	This parameter determines the region in which the SDPA searches an optimal solution. $\boldsymbol{O} \preceq \boldsymbol{X} \preceq \text{omegaStar} \times \boldsymbol{X}^0 = \text{omegaStar} \times \text{lambdaStar} \times \boldsymbol{I}, \boldsymbol{O} \preceq \boldsymbol{Y} \preceq \text{omegaStar} \times \boldsymbol{Y}^0 = \text{omegaStar} \times \text{lambdaStar} \times \boldsymbol{I}$.
double	lowerBound	Lower bound of the minimum objective value of the primal problem $\mathcal{P}$. $\sum_{i=1}^{m} c_i x_i^k$ gets smaller than the lowerBound, the SDPA stops the iteration
double	upperBound	Upper bound of the maximum objective value of the dual problem $\mathcal{D}$. $\boldsymbol{F}_0 \bullet \boldsymbol{Y}^k$ gets larger than the upperBound, the SDPA stops the iteration.
double	betaStar	A parameter controlling the search direction when $(\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ is feasible.
double	betaBar	A parameter controlling the search direction when $(\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ is infeasible. The value of betaBar must be not less than the value of betaStar; $0 \leq \text{betaStar} \leq \text{betaBar}$.
double	gammaStar	A reduction factor for the primal and dual step lengths; $0.0 < \text{gammaStar} < 1.0$.

10.3. Methods of the SDPA Class

SDPA();

A constructor

~SDPA();

A destructor

void Delete();

This methods deletes all information of the instance.

double* getResultXVec();

Get a pointer to the primal vector $\boldsymbol{x}$. We can get $\boldsymbol{x}_5$ as follow.

```
double* elements = getResultXVec();
double value = elements[5-1];
```

double* getResultXMat(int l);

Get a pointer to the l-th block of the primal matrix $\boldsymbol{X}$. We can get $\boldsymbol{X}.block\{3\}(2,4)$ as follow.

```
double* elements = Problem1.getResultXMat(3-1);
int nRow = Problem1.blockStruct[3-1];
double value = elements[(2-1)+ nRow*(4-1)];
```

If $\boldsymbol{X}.block\{l\}$ is diagonal matrix, We can get $\boldsymbol{X}^*.block\{5\}(7,7)$ as follow.

```
double* elements = Problem1.getResultXMat(5-1);
int nRow = Problem1.blockStruct[5-1];
if (nRow < 0) {
  double value = elements[7-1];
}
```

double* getResultYMat(int l);

Get a pointer to the l-th block of the dual matrix $\boldsymbol{Y}$. We can get $\boldsymbol{Y}.block\{3\}(2,4)$ as follow.

```
double* elements = Problem1.getResultYMat(3-1);
int nRow = Problem1.blockStruct[3-1];
double value = elements[(2-1)+ nRow*(4-1)];
```

If $\boldsymbol{Y}.block\{l\}$ is diagonal matrix, We can get $\boldsymbol{Y}.block\{5\}(7,7)$ as follow.

```
double* elements = Problem1.getResultYMat(5-1);
int nRow = Problem1.blockStruct[5-1];
if (nRow < 0) {
  double value = elements[7-1];
}
```

10.4. Functions Access to the SDPA Class

- SDPA_initialize

Synopsis	**SDPA_initialize(SDPA& SDP)**		
Purpose			Initialize base structures and file.
Argument	SDPA&	SDP	The instance to handle.
Return	void		nothing

- SDPA_initialize2

Synopsis	**SDPA_initialize2(SDPA& SDP)**		
Purpose			Initialize matrices.
Argument	SDPA&	SDP	The instance to handle.
Return	void		nothing

- SDPA_Input_cVECT

Synopsis	**void SDPA_Input_cVECT(SDPA& SDP, int i, double value);**		
Purpose			Set elements of $\boldsymbol{c}$.
Argument	SDPA&	SDP	The instance to handle.
	int	i	vector index, that is c_i.
	double	value	c_i is set as this value, that is c_i = value.
Return	void		nothing

- SDPA_CountUpperTriangle

Synopsis	**SDPA_CountUpperTriangle(SDPA& SDP, int k, int l, int i, int j, int nonzero)**		
Purpose			Set the number of nonzero elements in the upper triangular part.
Argument	SDPA& int	SDP k	The instance to handle. constraint index, that is $\boldsymbol{F}_k$.
	int	l	block index, that is $\boldsymbol{F}_k.block\{l\}$.
	int	nonzero	the number of nonzero elements in the upper triangluar part, that is $\boldsymbol{F}_k.block\{l\}$ has this value's nonzero elements of this value.
Return	void		nothing

- SDPA_Make_sfMAT

Synopsis	**SDPA_Make_sfMat(SDPA& SDP)**		
Purpose			Initialize memory to store matrices.
Argument	SDPA&	SDP	The instance to handle.
Return	void		nothing

- SDPA_InputElement

Synopsis	**SDPA_InputElement(SDPA& SDP, int k, int l, int i, int j, double value)**		
Purpose			Set elements of constraint and objective matrix.
Argument	SDPA&	SDP	The instance to handle.
	int	k	constraint index, that is $\boldsymbol{F}_k$.
	int	l	block index, that is $\boldsymbol{F}_k.block\{l\}$.
	int	i	row index.
	int	j	column index, that is $\boldsymbol{F}_k.block\{l\}(i,j)$. j must be greater than or equals i.
	double	value	The value to set, that is $\boldsymbol{F}_k.block\{l\}(i,j)$ = value (i≤j).
Return	void		nothing

- SDPA_Input_InixVec

Synopsis	**SDPA_Input_InixVec(SDPA& SDP, int k, double value)**		
Purpose			Set elements of the initial primal vector $\boldsymbol{x}^0$.
Argument	SDPA&	SDP	The instance to handle.
	int	k	the index, that is $\boldsymbol{x}^0_k$.
	double	value	The value to set, that is $\boldsymbol{x}^0_k = value$.
Return	void		nothing

- SDPA_Input_IniXMat

Synopsis	**SDPA_Input_IniXMat(SDPA& SDP, int l, int i, int j, int value)**		
Purpose			Set elements of the initial primal matrix $\boldsymbol{X}^0$.
Argument	SDPA&	SDP	The instance to handle.
	int	l	block index, that is $\boldsymbol{X}^0.block\{l\}$.
	int	i	row index.
	int	j	column index, that is $\boldsymbol{X}^0.block\{l\}(i,j)$. j must be greater than or equals i.
	double	value	The value to set, that is $\boldsymbol{X}^0.block\{l\}(i,j)$ = value (i≤j).
Return	void		nothing

- SDPA_Input_IniYMat

Synopsis	**SDPA_Input_IniYMat(SDPA& SDP, int l, int i, int j, int value)**		
Purpose			Set elements of the initial dual matrix $\boldsymbol{Y}^0$.
Argument	SDPA&	SDP	The instance to handle.
	int	l	block index, that is $\boldsymbol{Y}^0.block\{l\}$.
	int	i	row index.
	int	j	column index, that is $\boldsymbol{Y}^0.block\{l\}(i,j)$. j must be greater than or equals i.
	double	value	The value to set, that is $\boldsymbol{Y}^0.block\{l\}(i,j)$ = value (i≤j).
Return	void		nothing

- SDPA_Check_sfMAT

Synopsis	**SDPA_Check_sfMat(SDPA& SDP)**		
Purpose			Check the consistence of the input data .
Argument	SDPA&	SDP	The instance to handle.
Return	bool		If 'true', then the input data is consistent. On the other hand, if 'false', then the input data is inconstent.

- SDPA_Solve

Synopsis	**SDPA_Solve(SDPA& SDP)**		
Purpose			Solve the problem defined by the input data.
Argument	SDPA&	SDP	The instance to handle.
Return	void		nothing

- SDPA_Copy_Current_To_Ini

Synopsis	**SDPA_Copy_Current_To_Ini(SDPA& SDP)**		
Purpose			After we solve by SDPA_Sole, we can copy current point to initial point. (See example4.cpp,example5.cpp, example6.cpp)
Argument	SDPA&	SDP	The instance to handle.
Return	void		nothing

10.5. Stand Alone Executable binary

Stand alone executable binary"sdpa" has the two following option types.

- option type 1

```
./sdpa DataFile OutputFile [InitialPtFile]
example1-1: ./sdpa example1.dat example1.result
example1-2: ./sdpa example1.dat-s example1.result
example1-3: ./sdpa example1.dat example1.result example1.ini
example1-4: ./sdpa example1.dat example1.result example1.ini-s
```

- option type 2

```
./sdpa [option filename]+
  -dd : data dense :: -ds : data sparse
  -id : init dense :: -is : init sparse
  -o  : output     :: -p  : parameter
example2-1: ./sdpa -o example1.result -dd example1.dat
example2-2: ./sdpa -ds example1.dat-s -o example2.result -p param.sdpa
```

Note that we have to assign at least output file with '-o' and input data file with '-dd' or '-ds'. Since a confusion of options '-dd' and '-ds' would make the SDPA hang up, we has to be careful which options we use, '-dd' or '-ds'. When we do not assign parameter file in the case option types2, the SDPA uses default parameter, regardless of existence of parameter file "param.sdpa". And we can use other file name for parameter file than 'param.sdpa' with '-p' option. On the other hand, in the case option type1, the SDPA always needs parameter file "param.sdpa".

11. For the SDPA 5.01 Users

When we update the SDPA from version 5.01, we change the following points.

11.1. Option for Executable Binary

- Search direction — In the SDPA 5.01, three types of search directions are available; HKM [6, 7, 9], NT [10, 12], and AHO [1, 2]. However in this version, only HKM direction is available. The reason we have chosen HKM direction is that this direction is the most efficient direction in the three types in almost all problems. Since the SDPA of this version uses only HKM direction, options '−1', '−2', '−3' for the SDPA 5.01 are not available.
- Check the input data — The SDPA 5.01 can check whether input matrices are symmetric by adding the option '−c'. However in this version, this option is not available, since checking sometime needs long time.

11.2. Update of Callable Library Interface

In this version we use *LAPACK*[3] for matrix computation instead of *Meschach*[11] used in the SDPA 5.01. Since we have changed a storage of solution to cope with LAPACK, we have to change source code for the SDPA 5.01.

- Compilation — In compilation, header files and library files of LAPACK and the SDPA must be indicated explisitly. We assume the header files of LAPACK are installed in `$(HOME)/lapack/include` and the library files are installed in `$(HOME)/lapack/lib` and the SDPA is installed in `$(HOME)/sdpa` (See the install manual.) For example, we compile "example1-1.cpp" as follow.

```
$ g++ -O3 -I$(HOME)/sdpa -I$(HOME)/lapack/include example1-1.cpp
$ g++ -O3 -o example1-1.exe example1-1.cpp -L$(HOME)/sdpa -l sdpa \
         -L$(HOME)/lapack/lib -llapack -lcblaswr -lcblas -lf77blas \
          -lI77 -lF77 -latlas
```

  we get an executable "example1-1.exe", and execute

```
$ ./example1-1.exe
```

- Search direction — In the SDPA 5.01, three types of search directions are available; HKM [6, 7, 9], NT [10, 12], and AHO [1, 2]. However in this version, only HKM direction is available. The reason we have chosen HKM direction is that this direction is the efficient direction in the three types in almost all problems.

  A line in source code

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

is neglected. We can assign the variable Method as one of `KSH,NT,AHO` for compability to the SDPA 5.01, but it is not used internally.

- Access to the solution — In this version, matrices are stored as one dimension array to suit LAPACK interface. On the other hand, in the SDPA 5.01 they are stored as two dimension. This difference causes that we have to change access to the solution. For example, we accesses the soltion in SDPA 5.01 as follow,

```
  printf("\nxVec = \n");

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

  printf("\nxMat = \n");

  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("\nyMat = \n");
  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");

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;
};
```

We have to change the above source code as follow to this version of the SDPA. (See example2-1.cpp .)

```
fprintf(stdout, "xVec = \n");
// Problem1.printResultXVec(stdout);
element = Problem1.getResultXVec();
PrintVector(Problem1.mDIM,element,stdout);

fprintf(stdout, "xMat = \n");
// Problem1.printResultXMat(stdout);
fprintf(stdout, "{\n");
for (int l=0; l<Problem1.nBLOCK; ++l) {
  element = Problem1.getResultXMat(l);
  int nRow = Problem1.bLOCKsTRUCT[l];
  int nCol = nRow;
  if (nRow<0) {
    nCol = -nCol;
    nRow = 1;
  }
  PrintMatrix(nRow,nCol,element,stdout);
}
fprintf(stdout, "}\n");
```

```
fprintf(stdout, "yMat = \n");
// Problem1.printResultYMat(stdout);
fprintf(stdout, "{\n");
for (int l=0; l<Problem1.nBLOCK; ++l) {
  element = Problem1.getResultYMat(l);
  int nRow = Problem1.bLOCKsTRUCT[l];
  int nCol = nRow;
  if (nRow<0) {
    nCol = -nCol;
    nRow = 1;
  }
  PrintMatrix(nRow,nCol,element,stdout);
}
fprintf(stdout, "}\n");

#define FORMAT "%+8.16e"

void PrintMatrix(int nRow, int nCol, double* element, FILE* fpOut)
{
  fprintf(fpOut,"{");
  for (int i=1; i<=nRow-1; ++i) {
    if (i==1) {
      fprintf(fpOut," ");
    } else {
      fprintf(fpOut,"  ");
    }
    fprintf(fpOut,"{");
    for (int j=1; j<=nCol-1; ++j) {
      fprintf(fpOut, FORMAT",",element[(i-1)+nRow*(j-1)]);
    }
    fprintf(fpOut,FORMAT" },\n",element[(i-1)+nRow*(nCol-1)]);
  }
  if (nRow>1) {
    fprintf(fpOut,"  {");
  }
  for (int j=1; j<=nCol-1; ++j) {
    fprintf(fpOut,FORMAT",",element[(nRow-1)+nRow*(j-1)]);
  }
  fprintf(fpOut,FORMAT" }",element[(nRow-1)+nRow*(nCol-1)]);
  if (nRow>1) {
    fprintf(fpOut,"   }\n");
  } else {
    fprintf(fpOut,"\n");
  }
}

void PrintVector(int nDim, double* element, FILE* fpOut)
```

```
{
  fprintf(fpOut,"{");
  for (int j=1; j<=nDim-1; ++j) {
    fprintf(fpOut,FORMAT",",element[j-1]);
  }
  if (nDim>0) {
    fprintf(fpOut,FORMAT"}\n",element[nDim-1]);
  } else {
    fprintf(fpOut,"  }\n");
  }
}
```

- Copying the solution to the initial point — When we copy the solution to the initail point for solving from the current solution, we can use a function `SDPA_Copy_Current_To_Init` (See example4.cpp,example5.cpp, example6.cpp). For example, in source code in the SDPA 5.01,

```
Problem1.InixVec.initialize(Problem1.mDIM);
Problem1.IniXMat.initialize(Problem1.nBLOCK, Problem1.bLOCKsTRUCT, 0.0);
Problem1.IniYMat.initialize(Problem1.nBLOCK, Problem1.bLOCKsTRUCT, 0.0);

copy(Problem1.xVec, Problem1.InixVec);
copy(Problem1.XMat, Problem1.IniXMat);
copy(Problem1.YMat, Problem1.IniYMat);
```

  we have to change the above codes to a function `SDPA_Copy_Current_To_Init`.

```
SDPA_Copy_Current_To_Ini(Problem1);
```

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] Anderson, E. and Bai, Z. and Bischof, C. and Blackford, S. and Demmel, J. and Dongarra, J. and Du Croz, J. and Greenbaum, A. and Hammarling, S. and McKenney, A. and Sorensen, D. "LAPACK Users' Guide Third" *Society for Industrial and Applied Mathematics 1999 Philadelphia*, PA, ISBN 0-89871-447-8 (paperback).

[4] S. Boyd *et al*, "Linear matrix inequalities in system and control theory" *Society for Industrial and Applied Mathematics 1994 Philadelphia*, PA, ISBN 0-89871-334-X

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

[6] 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.

[7] 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*.

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

[9] 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*.

[10] 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.

[11] 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.

[12] 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.