Research Reports on Mathematical and Computing Sciences

SDPA (SemiDefinite Programming Algorithm) and SDPA-GMP User's Manual — Version 7.1.1

Katsuki Fujisawa, Mituhiro Fukuda, Kazuhiro Kobayashi, Masakazu Kojima, Kazuhide Nakata, Maho Nakata, and Makoto Yamashita

June 18th 2008, B–448

Department of Mathematical and Computing Sciences
Tokyo Institute of Technology

SERIES B: Operations Research

Research Report B-448
Department of Mathematical and Computing Sciences
Tokyo Institute of Technology
June 2008

SDPA (SemiDefinite Programming Algorithm) and SDPA-GMP User's Manual — Version 7.1.1

Katsuki Fujisawa[*1], Mituhiro Fukuda[*2], Kazuhiro Kobayashi[*3], Masakazu Kojima[*4], Kazuhide Nakata[*5], Maho Nakata[*6], and Makoto Yamashita[*7]

Abstract. The SDPA (SemiDefinite Programming Algorithm) [5] is a software package for solving semidefinite programs (SDPs). 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* [1] for matrix computations. The SDPA version 7.1.1 enjoys the following features:

- Efficient method for computing the search directions when the SDP to be solved is large scale and sparse [4].
- Block diagonal matrix structure and sparse matrix structure are supported for data matrices.
- Sparse or dense Cholesky factorization for the Schur matrix is automatically selected.
- An initial point can be specified.
- Some information on infeasibility of the SDP is provided.

Also we provide the SDPA-GMP, multiple precision arithmetic version of the SDPA via the GMP Library (the GNU Multiple Precision Arithmetic Library) with additional feature.

- Ultra highly accurate SDP solution by utilizing multiple precision arithmetic.

This manual and the SDPA can be downloaded from the WWW site

http://sdpa.indsys.chuo-u.ac.jp/sdpa/index.html

Key words. Semidefinite programming, interior-point method, computer software, multiple precision arithmetic

∗1 Department of Industrial and Systems Engineering
Chuo University
1-13-27 Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan

∗2 Global Edge Institute
Tokyo Institute of Technology
2-12-1-S6-5 Oh-Okayama, Meguro-ku, Tokyo 152-8550, Japan

∗3 Center for Logistics Research
National Maritime Research Institute
6-38-1 Shinkawa, Mitaka-shi, Tokyo 181-0004, Japan

∗4 Department of Mathematical and Computing Sciences
Tokyo Institute of Technology
2-12-1-W8-29 Oh-Okayama, Meguro-ku, Tokyo 152-8552, Japan

∗5 Department of Industrial Engineering and Management
Tokyo Institute of Technology
2-12-1-W9-62, Oh-Okayama, Meguro-ku, Tokyo 152-8552, Japan

∗6 Advanced Center for Computing and Communication
RIKEN
2-1 Hirosawa, Wako-shi, Saitama 351-0198, Japan

∗7 Department of Mathematical and Computing Sciences
Tokyo Institute of Technology
2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552, Japan

Preface

We are pleased to release a new version 7.1.1 of the SDPA and the SDPA-GMP.

The SDPA 7.1.1 was completely revised from its source code, and there are great performance improvements on its computational time and memory usage. In particular, it uses and stores less variables internally, and its overall memory usage is less than half of the previous version. Also it performs sparse Cholesky factorization when the Schur Complement Matrix is sparse. As a consequence, the SDPA can efficiently solve SDPs with a large number of block diagonal matrices or non-negative constraints. Additional, there are some improvements on its numerical stability due to a better control in the interior-point algorithm.

The differences between this version and the previous version, 6.2.1, are summarized in Section 11.

Finally, we also provide the SDPA-GMP to solve SDPs highly accurately utilizing multiple precision arithmetic via the GMP Library (the GNU Multiple Precision Arithmetic Library). Note that the SDPA-GMP is typically several hundred times slower than the SDPA. Details of the SDPA-GMP are summarized in the Appendix.

We hope that the SDPA and the SDPA-GMP support many researches in various fields. We also welcome any suggestions and comments that you may have. When you want to contact us, please send an e-mail to the following address.

kojima-sdpa@is.titech.ac.jp

Contents

1. Build and Installation

This section describes how to build and install the SDPA. Usually, the SDPA is distributed as source codes, therefore, users must build it by themselves. We have done build and installation tests on Fedora 8, 9 (i386/x86_64/ppc), Red Hat Enterprise Linux (CentOS) 4, 5 (i386/x86_64), Ubuntu 7.10 (i386/x86_64), Vine Linux 4.2 (i386), MacOSX Leopard/Tiger (Intel/PowerPC) and FreeBSD 6/7. You may also possibly build on other UNIX like platforms and the Windows cygwin environment.

For better performance, the SDPA should link against optimized Basic Linear Algebra Subprograms (BLAS) and Linear Algebra PACKage (LAPACK). Please refer to the Section 1.6 for more details.

Alternatively, you can also use our online solver, or check for pre-build binary packages at the SDPA homepage (see Section 1.2).

1.1 Prerequisites

It is necessary to have at least the following programs installed on your system except for the MacOSX.

- C, C++ compiler.
- FORTRAN compiler.
- BLAS (**http://www.netlib.org/blas/**) and LAPACK (**http://www.netlib.org/lapack/**).

For MacOSX Leopard (Intel/PowerPC), you will need the Xcode 3.0, and you cannot build the SDPA on the Leopard with Xcode 2.5. For MacOSX Tiger (Intel/PowerPC), you will need either the Xcode 2.4.1 or the Xcode 2.5.

You can download the Xcode at Apple Developer Connection (**http://developer.apple.com/**) at free of charge.

In the following instructions, we install C, C++, FORTRAN compilers, BLAS and LAPACK and/or Xcode as well. You can skip this part if your system already have them. If not, you may need root access or administrative privilege to your computer. Please ask your system administrator for installing development tools.

If your system only lacks BLAS and LAPACK, you can build them by yourself without root privileges and passing appropriate flags at the command line.

1.2 How to Obtain the Source Code

You can obtain the source code following the links at the SDPA homepage:
http://sdpa.indsys.chuo-u.ac.jp/sdpa/index.html

You can find the latest news at about the SDPA at the **index.html** file.

1.3 How to Build

This section describes how to build the SDPA on Fedora 8, 9 (i386/x86_64/ppc), Ubuntu 7.10 (i386/x86_64), Vine Linux 4.2 (i386) MacOSX Intel/PowerPC Leopard (Xcode 3.0)/Tiger (Xcode 2.4.1 and 2.5) and FreeBSD 6/7. We assume that users have a root access via the command "su" or "sudo". If you do not have such privileges, please ask your system administrators.

1.3.1 Fedora 8 and 9 (i386/x86_64/ppc), Red Hat Enterprise Linux (CentOS) 4, 5 (i386/x86_64)

To build the SDPA, type the following.

```
$ bash
$ su
Password:
# yum update glibc glibc-common
# yum install gcc gcc-c++ gcc-gfortran
# yum install lapack lapack-devel blas blas-devel
# yum install atlas atlas-devel
# exit
$ tar xvfz sdpa.7.1.1.src.2008xxxx.tar.gz
$ cd sdpa-7.1.1
$ ./configure
$ make
```

1.3.2 Ubuntu 7.10 Desktop (i386/x86_64)

To build the SDPA, type the following.

```
$ bash
$ sudo apt-get install g++ patch
$ sudo apt-get install lapack3 lapack3-dev
$ sudo apt-get install atlas3-base atlas3-base-dev
$ tar xvfz sdpa.7.1.1.src.2008xxxx.tar.gz
$ cd sdpa-7.1.1
$ ./configure
$ make
```

1.3.3 Vine Linux 4.2 (i386)

```
$ bash
$ su
```

Modify /etc/apt/sources.list (add the word "extras" between "updates" and "nonfree") from

```
rpm     [vine] http://updates.vinelinux.org/apt 4.2/$(ARCH) main plus updates nonfree
rpm-src [vine] http://updates.vinelinux.org/apt 4.2/$(ARCH) main plus updates nonfree
```

to

```
rpm     [vine] http://updates.vinelinux.org/apt 4.2/$(ARCH) main plus updates extras nonfree
rpm-src [vine] http://updates.vinelinux.org/apt 4.2/$(ARCH) main plus updates extras nonfree
```

Then, type

```
# apt-get update
# apt-get install gcc-g77
# apt-get install lapack lapack-devel blas blas-devel
# exit
$ tar xvfz sdpa.7.1.1.src.2008xxxx.tar.gz
$ cd sdpa-7.1.1
$ ./configure
$ make
```

1.3.4 MacOSX Leopard (Intel/PowerPC)

Download and install the Xcode 3 from Apple Developer Connection (**http://developer.apple.com/**) and install it. Note that we support only the Xcode 3 on Leopard. We do not support Leopard with Xcode 2.5.

```
$ bash
$ tar xvfz sdpa.7.1.1.src.2008xxxx.tar.gz
$ cd sdpa-7.1.1
$ ./configure
$ make
```

1.3.5 MacOSX Tiger (Intel/PowerPC)

Download and install either the Xcode 2.4.1 or the Xcode 2.5 from Apple Developer Connection. Then type the following.

```
$ bash
$ tar xvfz sdpa.7.1.1.src.2008xxxx.tar.gz
$ cd sdpa-7.1.1
$ ./configure
$ make
```

1.3.6 FreeBSD 6 and 7

```
$ cd /usr/ports
$ su
Password:
# make update
# cd /usr/ports/math/sdpa
# make install
# exit
```

1.4 How to Install

You will find the "sdpa" executable binary file at this point, and you can install it as:

```
$ su
Password:
# mkdir /usr/local/bin
# cp sdpa /usr/local/bin
# chmod 777 /usr/local/bin/sdpa
```

or you can install it to your favorite directory.

```
$ mkdir /home/nakata/bin
$ cp sdpa /home/nakata/bin
```

1.5 Test Run

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

```
$ sdpa
SDPA start at    Tue Jan 22 15:28:59 2008

*** Please assign data file and output file.***

---- option type 1 ------------
sdpa DataFile OutputFile [InitialPtFile] [-pt parameters]
parameters = 0 default, 1 aggressive, 2 stable
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 -pt 2

---- option type 2 ------------
sdpa [option filename]+
  -dd : data dense :: -ds : data sparse
  -id : init dense :: -is : init sparse
  -o  : output     :: -p  : parameter
  -pt : parameters , 0 default, 1 aggressive
                     2 stable
example2-1: sdpa -o example1.result -dd example1.dat
example2-2: sdpa -ds example1.dat-s -o example2.result -p param.sdpa
example2-3: sdpa -ds example1.dat-s -o example3.result -pt 2
```

Let us solve the SDP "example1.dat-s".

```
$ sdpa -ds example1.dat-s -o example1.out
SDPA start at    Fri Dec  7 18:30:56 2007
data      is example1.dat-s : sparse
parameter is ./param.sdpa
out       is example1.out
```

```
DENSE computations
   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 2.3e-16 3.6e-03 +1.96e+02 -3.74e+01 1.3e+00 1.0e+00 2.00e-01
 3 1.8e+01 2.3e-16 1.5e-17 -6.84e+00 -4.19e+01 9.9e-01 9.9e-01 1.00e-01
 4 1.9e+00 2.6e-16 1.5e-17 -3.81e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
 5 1.9e-01 2.6e-16 3.0e-17 -4.15e+01 -4.19e+01 1.0e-00 9.0e+01 1.00e-01
 6 1.9e-02 2.5e-16 1.6e-15 -4.19e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
 7 1.9e-03 2.6e-16 2.2e-17 -4.19e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
 8 1.9e-04 2.5e-16 1.5e-17 -4.19e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
 9 1.9e-05 2.7e-16 7.5e-18 -4.19e+01 -4.19e+01 1.0e-00 9.0e+01 1.00e-01
10 1.9e-06 2.6e-16 5.0e-16 -4.19e+01 -4.19e+01 1.0e-00 9.0e+01 1.00e-01

phase.value = pdOPT
   Iteration = 10
          mu = 1.9180668442024010e-06
relative gap = 9.1554505577840628e-08
         gap = 3.8361336884048019e-06
      digits = 7.0383202783481345e+00
objValPrimal = -4.1899996163866390e+01
objValDual   = -4.1899999999999999e+01
p.feas.error = 3.2137639581876834e-14
d.feas.error = 4.7961634663806763e-13
total time   = 0.010
  main loop time = 0.010000
      total time = 0.010000
file   read time = 0.000000
```

1.6 Performance Tuning: Optimized BLAS and LAPACK

We highly recommend to use optimized BLAS and LAPACK for better performance, *e.g.*,

- Automatically Tuned Linear Algebra Software (ATLAS): **http://math-atlas.sourceforge.net/** . Note that in Section 1.3, we described how to use ATLAS which comes with Fedora Core 6, Fedora 7, 8, and Ubuntu. ATLAS optimizes itself while it is built, and we recommend users to rebuild it on the target computers. For convenience, some pre-build packages are available at : **https://sourceforge.net/project/showfiles.php?group_id=23725** .
- GotoBLAS: **http://www.tacc.utexas.edu/resources/software/** .
- Intel Math Kernel Library (MKL): **http://www.intel.com/cd/software/products/asmo-na/eng/266858.htm** .

Table 1 and 2 show how the SDPA 7.0.5 performs on several benchmark problems when replacing the BLAS and LAPACK libraries. Typically, the SDPA 7.0.5 with optimized BLAS and LAPACK seems much faster than the one with BLAS/LAPACK 3.1.1. Furthermore, we can receive benefits of multi-thread computing when using SMP and/or multi-core machines.

1.6.1 Configure Options

There are some configure options you need to specify. We specify at least two of them when using optimized BLAS and LAPACK. Type "configure –help" for more details.

- `--with-blas`

Table 1: Numerical experiments(SDPA 7.0.5 + BLAS library) : time:sec.(# of iterations)

BLAS library(# of threads)	Prob(1)	Prob(2)	Prob(3)	Prob(4)
BLAS/LAPACK 3.1.1(1)	70.51(36)	128.36(15)	226.24(18)	342.42(20)
ATLAS 3.8.1(1)	53.00(35)	49.86(15)	41.24(18)	182.65(19)
ATLAS 3.8.1(4)	32.76(35)	19.66(15)	18.26(18)	81.73(19)
Intel MKL 10.0.2.018(1)	44.25(35)	39.92(15)	34.63(18)	172.05(21)
Intel MKL 10.0.2.018(2)	32.88(35)	24.43(15)	22.44(18)	110.41(22)
Intel MKL 10.0.2.018(4)	26.61(35)	16.38(15)	16.37(18)	67.78(18)
Intel MKL 10.0.2.018(8)	25.17(35)	13.38(15)	14.83(18)	63.01(19)
GotoBLAS 1.24(1)	42.94(34)	40.37(15)	32.42(18)	160.75(21)
GotoBLAS 1.24(2)	31.81(35)	23.89(15)	21.07(18)	99.76(20)
GotoBLAS 1.24(4)	26.60(36)	14.81(15)	15.59(18)	64.29(18)
GotoBLAS 1.24(8)	23.84(35)	11.57(15)	13.95(18)	67.47(21)

CPU : Intel Xeon 5345 (2.33GHz) $\times$ 2 CPUs, 8 cores
OS : CentOS Ver 5.1 64bit
Compiler : Intel (C/C++ & Fortran) 10.1.012

Table 2: Numerical experiments(SDPA 7.0.5 + BLAS library) : time:sec.(# of iterations)

BLAS library(# of threads)	Prob(1)	Prob(2)	Prob(3)	Prob(4)
BLAS/LAPACK 3.1.1(1)	49.48(36)	67.23(15)	143.84(18)	220.38(20)
ATLAS 3.8.1(1)	40.29(35)	37.27(15)	32.14(18)	145.72(20)
ATLAS 3.8.1(2)	29.76(35)	23.22(15)	21.12(18)	99.04(21)
Intel MKL 10.0.2.018(1)	33.39(35)	30.49(15)	26.68(18)	129.72(21)
Intel MKL 10.0.2.018(2)	25.00(35)	18.72(15)	17.56(18)	84.24(22)
GotoBLAS 1.24(1)	33.91(36)	40.23(15)	24.53(18)	109.72(19)
GotoBLAS 1.24(2)	24.52(35)	18.37(15)	16.11(18)	71.53(19)

CPU : Intel Core 2 E8200 (2.66GHz) $\times$ 1 CPU, 2 cores
OS : CentOS Ver 5.1 64bit
Compiler : Intel (C/C++ & Fortran) 10.1.012

Prob(1) : Structural Optimization : g1717.dat-s
Prob(2) : Combinatorial Optimization(1) : mcp1000-10.dat-s
Prob(3) : Combinatorial Optimization(2) : theta5.dat-s
Prob(4) : Quantum Chemistry : CH.2Pi.STO6G.pqg.dat-s

Supply a command line to link against your BLAS, e.g.,
`--with-blas="-L/home/nakata/gotoblas/lib -lgoto"` .

- `--with-lapack`

  Supply a command line to link against your LAPACK, e.g.,
  `--with-lapack="-L/home/nakata/gotoblas/lib -lgoto -llapack"` .

1.6.2 Linking Against ATLAS

Install ATLAS, and assume that the ATLAS libraries are installed at "/home/nakata/atlas/lib".
If there exists "liblapack.a" at '/home/nakata/atlas/lib", we recommend rename or remove it. Then, reconfigure and rebuild the SDPA like following.

```
$ make clean
$ ./configure --with-blas="-L/home/nakata/atlas/lib -lptf77blas -lptcblas -latlas"
$ make
```

The number of cores uses is determined at the build process, and you need a multi-core CPU to accelerate BLAS operations.

1.6.3 Linking Against GotoBLAS

Install GotoBLAS, and assume that the GotoBLAS libraries are installed at "/home/nakata/gotoblas/lib".
Reconfigure and rebuild the SDPA like following.

```
$ make clean
$ ./configure --with-blas="-L/home/nakata/gotoblas/lib -lgoto" \
  --with-lapack="-L/home/nakata/gotoblas/lib -lgoto -llapack"
$ make
```

You can set the number of threads uses via setting the "OMP_NUM_THREADS" environment variable before running the SDPA like following. In this case, two cores will be used for BLAS operations. You need a multi-core CPU to accelerate BLAS operations.

```
$ export OMP_NUM_THREADS=2
$ sdpa -ds example1.dat-s -o example1.out
```

1.6.4 Linking Against Intel Math Kernel Library

Install Intel Math Kernel Library, and assume that the library is installed at "/opt/intel/mkl/10.0.3.020/lib/em64t".
Reconfigure and rebuild the SDPA like following.

```
$ make clean
$ ./configure --with-blas="-L/opt/intel/mkl/10.0.3.020/lib/em64t -lmkl_em64t -lguide" \
  --with-lapack="-L/opt/intel/mkl/10.0.3.020/lib/em64t -lmkl_lapack -lmkl_sequential -lmkl_core"
$ make
```

You can set the number of threads uses via setting the "OMP_NUM_THREADS" environment variable before running the SDPA like following. In this case, four cores will be used for BLAS operations. You need a multi-core CPU to accelerate BLAS operations.

```
$ export OMP_NUM_THREADS=4
$ sdpa -ds example1.dat-s -o example1.out
```

1.7 Performance Tuning: Compiler Options

We also recommend adding some optimized flags to the compilers. Our recommendations are

- `-O2 -funroll-all-loops` (default)
- `-static -O3 -funroll-all-loops -march=nocona -msse3 -mfpmath=sse` (on Core2).

The above default options are sufficient for better performance in many cases. To pass your optimization flags, set them in CFLAGS and CXXFLAGS environment variables. Here is an example:

```
$ bash
$ make clean
$ export CFLAGS="-static -funroll-all-loops -O3 -m64 -march=nocona -msse3 -mfpmath=sse"
$ export CXXFLAGS="-static -funroll-all-loops -O3 -m64 -march=nocona -msse3 -mfpmath=sse"
$ ./configure --with-blas="-L/home/nakata/gotoblas/lib -lgoto" \
  --with-lapack="-L/home/nakata/gotoblas/lib -lgoto -llapack"
$ make
```

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.

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

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

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

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

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

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

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

$\boldsymbol{U} \succeq \boldsymbol{O} \iff \boldsymbol{U} \in \mathcal{S}$ is positive semidefinite

Throughout this manual, we denote the primal SDP by $\mathcal{P}$ and its dual problem by $\mathcal{D}$. The SDP is determined by m, n, $\boldsymbol{c} \in \mathbb{R}^m$, and $\boldsymbol{F}_i \in \mathcal{S}$ $(i = 0, 1, \ldots, m)$. 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 a given SDP does not satisfy this assumption, the SDPA can abnormally stop due to some numerical instability, but it does not mean that it will necessarily happen.

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

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

we can easily transform it into the standard form SDP as follows:

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

2.2 Example 1

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

Here

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

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

2.3 Example 2

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

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

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

$$
\vdots
$$

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

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

3. Files Necessary to Execute the SDPA

We need the following files to execute the SDPA:

- "**sdpa**" — The executable binary for solving an SDP.
- "input data file" — Any file name with the postfix "**.dat**" or "**.dat-s**" are accepted; 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 Sections 4. and 7.2 for details.
- "**param.sdpa**" — The file describing the parameters used in the "sdpa". See Section 5. for details. The name is fixed to "**param.sdpa**".
- "output file" — Any file name excepting "**sdpa**" and "**param.sdpa**". For example, "problem.1" and "example.out" are legitimate names for output files. See Section 6. 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.

4. Input Data 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 the Input Data File

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

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

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

4.4 Title and Comments

On the top of the input data file, we can write a single or multiple lines for "Title and Comments". Each line of "Title and Comments" must begin with `"` or `*` and should 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 Comments" when it starts. "Title and Comments" can be omitted.

4.5 Number of the Primal Variables

We write the number m of the primal variables in a line following the line(s) of "Title and Comments" 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 Number of Blocks and the Block Structure Vector

The SDPA handles block diagonal matrices as we have seen in Section 2.3. We express a common matrix data structure for the constraint matrices $\boldsymbol{F}_0,\ \boldsymbol{F}_1,\ \ldots,\ \boldsymbol{F}_m$ using the terms number of blocks, denoted by nBLOCK, and block structure vector, denoted by bLOCKsTRUCT. If we deal with a block diagonal matrix $\boldsymbol{F}$ of the form

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

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

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

If

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

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

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

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

Specify 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

We describe the constraint matrices $\boldsymbol{F}_0$, $\boldsymbol{F}_1$, ..., $\boldsymbol{F}_m$ according to the data format specified by nBLOCK and bLOCKsTRUCT stated in Section 4.6. 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 elements sequentially. For instance, for the matrix $\boldsymbol{F}$ 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 can 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 10 lines which sets 10 parameters. Each line of this file contains one of the 10 parameters followed by an comment. When the SDPA reads the file "param.sdpa", it neglects the comments.

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

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

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

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

- 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 of an optimal solution $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ of the SDP. In general, however, choosing such 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 large lambdaStar and a small 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 — 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 tends to get closer to the affine scaling direction without centering.

- betaBar — 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 tends to get closer to the affine scaling direction without centering. The value of betaBar must be no less than the value of betaStar; $0 \leq$ betaStar $\leq$ betaBar.

- gammaStar — 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  "input data file"  "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 at    Fri Dec  7 18:30:56 2007
data      is example1.dat-s : sparse
parameter is ./param.sdpa
out       is example.1.out

DENSE computations
   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 2.3e-16 3.6e-03 +1.96e+02 -3.74e+01 1.3e+00 1.0e+00 2.00e-01
 3 1.8e+01 2.3e-16 1.5e-17 -6.84e+00 -4.19e+01 9.9e-01 9.9e-01 1.00e-01
 4 1.9e+00 2.6e-16 1.5e-17 -3.81e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
 5 1.9e-01 2.6e-16 3.0e-17 -4.15e+01 -4.19e+01 1.0e-00 9.0e+01 1.00e-01
 6 1.9e-02 2.5e-16 1.6e-15 -4.19e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
 7 1.9e-03 2.6e-16 2.2e-17 -4.19e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
 8 1.9e-04 2.5e-16 1.5e-17 -4.19e+01 -4.19e+01 1.0e-00 1.0e-00 1.00e-01
 9 1.9e-05 2.7e-16 7.5e-18 -4.19e+01 -4.19e+01 1.0e-00 9.0e+01 1.00e-01
10 1.9e-06 2.6e-16 5.0e-16 -4.19e+01 -4.19e+01 1.0e-00 9.0e+01 1.00e-01

phase.value = pdOPT
   Iteration = 10
          mu = 1.9180668442024010e-06
relative gap = 9.1554505577840628e-08
         gap = 3.8361336884048019e-06
      digits = 7.0383202783481345e+00
objValPrimal = -4.1899996163866390e+01
objValDual   = -4.1899999999999999e+01
p.feas.error = 3.2137639581876834e-14
d.feas.error = 4.7961634663806763e-13
total time   = 0.010
  main loop time = 0.010000
      total time = 0.010000
file   read time = 0.000000
file   read time = 0.000000
```

- mu — The average complementarity $\boldsymbol{X}^k \bullet \boldsymbol{Y}^k/n$ (optimality measure). When both $\mathcal{P}$ and $\mathcal{D}$ get feasible, the relation

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

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

$$\text{thetaP} = \frac{\max\left\{\left|[\boldsymbol{X}^k - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^k + \boldsymbol{F}_0]_{p,q}\right| \ : \ p,q = 1,2,\ldots,n\right\}}{\max\left\{\left|[\boldsymbol{X}^0 - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^0 + \boldsymbol{F}_0]_{p,q}\right| \ : \ p,q = 1,2,\ldots,n\right\}};$$

The thetaP is theoretically monotone non-increasing, 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

$$\text{thetaD} = \frac{\max\left\{\left|\boldsymbol{F}_i \bullet \boldsymbol{Y}^k - c_i\right| \ : i = 1,2,\ldots,m\right\}}{\max\left\{\left|\boldsymbol{F}_i \bullet \boldsymbol{Y}^0 - c_i\right| \ : i = 1,2,\ldots,m\right\}};$$

The thetaD is theoretically monotone non-increasing, 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.
- 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 when the SDPA terminated.
- relative gap — The relative gap

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

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

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

- objValPrimal — The primal objective function value

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

- objValDual — The dual objective function value

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

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

$$\text{p.feas.error} = \max\left\{ \left| [\boldsymbol{X}^k - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^k + \boldsymbol{F}_0]_{p,q} \right| \ :\ p, q = 1, 2, \ldots, n \right\}$$

  This value is compared with **epsilonDash** (Section 5.). Even if the primal problem is feasible, this value may not be 0 due to numerical errors.
- d.feas.error — This value indicates the dual infeasibily in the last iteration,

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

  This value is compared with **epsilonDash** (Section 5.). Even if the dual problem is feasible, this value may not be 0 due to numerical errors.

- total time — This value indicates how much time the SDPA needs to execute all subroutines.
- main loop time —This value indicates 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 the input file and store the data 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.

```
SDPA start at Fri Dec  7 18:32:10 2007
*Example 2:
*mDim = 5, nBLOCK = 3, {2,3,-2}
data      is example2.dat
parameter is ./param.sdpa
out       is example2.out
   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 4.9e-16 6.2e-02 +8.66e+02 -2.60e+00 1.0e+00 1.0e+00 2.00e-01
 3 1.4e+02 4.9e-16 8.8e-17 +9.67e+02 +9.12e-01 9.5e-01 5.4e+00 1.00e-01
 4 1.8e+01 2.6e-16 8.6e-16 +1.46e+02 +2.36e+01 9.3e-01 1.5e+00 1.00e-01
 5 2.7e+00 3.3e-16 4.2e-16 +4.62e+01 +2.74e+01 8.1e-01 1.4e+00 1.00e-01
 6 5.7e-01 3.2e-16 2.0e-16 +3.43e+01 +3.03e+01 9.2e-01 9.3e-01 1.00e-01
 7 9.5e-02 3.3e-16 4.1e-17 +3.24e+01 +3.18e+01 9.5e-01 9.6e-01 1.00e-01
 8 1.3e-02 3.2e-16 1.9e-17 +3.21e+01 +3.20e+01 9.8e-01 1.0e-00 1.00e-01
 9 1.5e-03 3.5e-16 2.8e-17 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 1.00e-01
10 1.5e-04 3.2e-16 1.7e-17 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 1.00e-01
11 1.5e-05 3.4e-16 3.7e-17 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 1.00e-01
12 1.5e-06 3.2e-16 4.3e-17 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 1.00e-01
13 1.5e-07 3.3e-16 3.1e-17 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 1.00e-01

phase.value = pdOPT
   Iteration = 13
          mu = 1.5017857203245200e-07
relative gap = 3.2787330975500860e-08
         gap = 1.0512500042271640e-06
      digits = 7.4842939352608049e+00
objValPrimal = 3.2062693405e+01
objValDual   = 3.2062692354e+01
p.feas.error = 3.7747582837e-14
d.feas.error = 5.8841820305e-14
total time   = 0.000


Parameters are
maxIteration =    100
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,  nan
        ... abbreviation ...
 Total           =      0.000000,  nan

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 = 0.000000
        total time = 0.000000
  file   read time = 0.000000
```

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

- Lines with start '*' — These lines are comments in "example2.dat".
- Data, parameter, initial, output — These are the file names we assigned 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 to tune up the parameters, but the details are rather complicate, because the profile data seriously depends on internal algorithms.
- xVec — Approximate optimal primal variable vector $\boldsymbol{x}$.
- xMat — Approximate optimal primal variable matrix $\boldsymbol{X}$.
- yMat — Approximate optimal dual variable matrix $\boldsymbol{Y}$.

6.4 Printing DIMACS Errors

To display the error measures defined at the 7th DIMACS Implementation Challenge on Semidefinite and Related Optimization Problems [6],

1. Go to the subdirectory where SDPA source code is.
2. Edit the file sdpa_io.cpp, line 22

`#define DIMACS_PRINT 0`

to

`#define DIMACS_PRINT 1`

3. Type "make clean".
4. Type "make" to re-compile SDPA.

The SDPA will output on the display and in the output file the following DIMACS errors at the last iteration:

- Err1 — The relative dual feasibility error on the constraints
$$\frac{\sqrt{\sum_{i=1}^{m}(\boldsymbol{F}_i \bullet \boldsymbol{Y}^k - c_i)^2}}{1 + \max\{|c_i| \ : \ i = 1, 2, \ldots, m\}}$$
- Err2 — The relative dual feasibility error on the semidefiniteness
$$\max\left\{0, \frac{-\lambda_{\min}(\boldsymbol{Y}^k)}{1 + \max\{|c_i| \ : \ i = 1, 2, \ldots, m\}}\right\}$$
- Err3 — The relative primal feasibility error on the constraints
$$\frac{\|\boldsymbol{X}^k - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^k + \boldsymbol{F}_0\|_f}{1 + \max\{|[\boldsymbol{F}_0]_{ij}| \ : \ i, j = 1, 2, \ldots, n\}}\}$$
- Err4 — The relative primal feasibility error on the semidefiniteness
$$\max\left\{0, \frac{-\lambda_{\min}(\boldsymbol{X}^k)}{1 + \max\{|[\boldsymbol{F}_0]_{ij}| \ : \ i, j = 1, 2, \ldots, n\}}\right\}$$
- Err5 — The relative duality gap 1
$$\frac{\sum_{i=1}^{m} c_i x_i^k - \boldsymbol{F}_0 \bullet \boldsymbol{Y}^k}{1 + |\sum_{i=1}^{m} c_i x_i^k| + |\boldsymbol{F}_0 \bullet \boldsymbol{Y}^k|}$$
- Err6 — The relative duality gap 2
$$\frac{\boldsymbol{X}^k \bullet \boldsymbol{Y}^k}{1 + |\sum_{i=1}^{m} c_i x_i^k| + |\boldsymbol{F}_0 \bullet \boldsymbol{Y}^k|}$$

where $\|\cdot\|_f$ is a norm defined as a sum of the Frobenius norm of each block diagonal matrix, and $\lambda_{\min}(\cdot)$ is the smallest eigenvalue of a matrix.

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 optionally 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

$$(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0) = \left(\begin{pmatrix} 0.0 \\ -4.0 \\ 0.0 \end{pmatrix}, \begin{pmatrix} 11.0 & 0.0 \\ 0.0 & 9.0 \end{pmatrix}, \begin{pmatrix} 5.9 & -1.375 \\ -1.375 & 1.0 \end{pmatrix}\right),$$

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" is a legitimate initial point file name.

An initial point file contains the data

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

in this order, where the description for 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, $\boldsymbol{c} \in \mathbb{R}^m$ and $\boldsymbol{F}_i \in \mathcal{S}$ $(i = 0, 1, \ldots, m)$. When not only the constant matrices $\boldsymbol{F}_i \in \mathcal{S}$ $(i = 0, 1, \ldots, m)$ 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 legitimate names for sparse input data files. The SDPA distinguishes 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 data 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". Following them, each line 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 Comments
m — the number of the primal variables x_i's
nBLOCK — the number of blocks
bLOCKsTRUCT — the block structure vector
$\boldsymbol{c}$
$k_1\ b_1\ i_1\ j_1\ v_1$
$k_2\ b_2\ i_2\ j_2\ v_2$
$\cdots$
$k_p\ b_p\ i_p\ j_p\ v_p$
$\cdots$
$k_q\ b_q\ i_q\ j_q\ v_q$

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

7.3 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 legitimate names for sparse input data files. The SDPA distinguishes 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:

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

Here $s_p = 1$ or 2, $b_p \in \{1, 2, \ldots,$ nBLOCK$\}$, $1 \le i_p \le j_p$ and $v_p \in \mathbb{R}$. 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 Obtaining More Precision on the Approximate Solution

By default, each element of the approximate optimal solution $(\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ saved in the output file (see Section 6.3) has a coefficient with 4 digits in scientific notation.

It is possible to increase its precision.

1. Go to the subdirectory where SDPA source code is.
2. Edit the file sdpa_struct.cpp, line 25

   ```
   #define P_FORMAT "%8.3e"
   ```

   to, for instance,

   ```
   #define P_FORMAT "%8.5e"
   ```

3. Type "make clean".
4. Type "make" to re-compile SDPA.

Now, each element of the approximate optimal solution will have a coefficient with 6 digits in scientific notation.

7.5 More on Parameter File

We may encounter some numerical difficult 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 default parameters: betaStar, betaBar, and gammaStar. We present 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

```
1.0E4   double 0.0 < lambdaStar;
0.10 double 0.0 <= betaStar <  1.0;
0.30 double 0.0 <= betaBar  <  1.0, betaStar <= betaBar;
0.80 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;
0.95 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 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 the 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 "example1-2.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, 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 we 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 corresponding to **example2.dat** and solves this by calling the functions.

9. Transformation to the Standard Form of SDP

SDP has many applications, but frequently problems are not described in the standard form of SDP. In this section, we show some examples on 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, α_i^1, α_i^2 $(i = 1, \ldots, m), \beta^1, \beta^2 \in \mathbb{R}$. 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.

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

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 dual standard form of SDP. For example,

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

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

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

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

$$\begin{cases} \text{maximize} & \bar{\boldsymbol{F}}_0 \bullet \bar{\boldsymbol{Y}} \\ \text{subject to} & \bar{\boldsymbol{F}}_1 \bullet \bar{\boldsymbol{Y}} = c_1,\ \bar{\boldsymbol{F}}_2 \bullet \bar{\boldsymbol{Y}} = c_2, \\ & \bar{\boldsymbol{F}}_3 \bullet \bar{\boldsymbol{Y}} = c_3,\ \bar{\boldsymbol{F}}_4 \bullet \bar{\boldsymbol{Y}} = c_4,\ \bar{\boldsymbol{F}}_5 \bullet \bar{\boldsymbol{Y}} = c_5, \\ & \bar{\boldsymbol{Y}} \succeq \boldsymbol{O}, \end{cases}$$

where

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

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

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

$\text{nBlock} = 2, \text{blockStruct} = (n, -3)$.

9.2 Norm Minimization Problem

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

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

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:

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

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 of the matrices,

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

where $\boldsymbol{x} \in \mathbb{R}^p$. The Linear Matrix Inequality (LMI) [3] is defined as follows

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

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

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

We can reduce this 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 a 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 $\boldsymbol{x} \in \mathbb{R}^p$.

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

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

Here each feasible solution $\boldsymbol{u} \in \mathbb{R}^n$ of this problem corresponds 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 $\boldsymbol{C} = \frac{1}{4}(\mathrm{diag}(\boldsymbol{W}\boldsymbol{e}) - \boldsymbol{W})$, where $\boldsymbol{e} \in \mathbb{R}^n$ denotes the vector of ones and $\mathrm{diag}(\boldsymbol{W}\boldsymbol{e})$ the diagonal matrix of the vector $\boldsymbol{W}\boldsymbol{e} \in \mathbb{R}^n$, 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 $\boldsymbol{x} \in \mathbb{R}^n$ 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:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{C} \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{E}_{ii} \bullet \boldsymbol{X} = 1 \ (1 \leq i \leq n),\ \boldsymbol{X} \succeq \boldsymbol{O}. \end{array} \tag{3}$$

Here $\boldsymbol{E}_{ii}$ denotes the $n \times n$ symmetric matrix with (i, i)th element 1 and all others 0.

9.5 Choosing Between the Primal and Dual Standard Forms

Any SDP can be formulated as a primal standard form $\mathcal{P}$ or as a dual standard form $\mathcal{D}$ (see Section 2.1). This does not mean that one is the dual of the other, but simply that they are indeed two different formulations of the same problem, each one having its dual counterpart.

Many times, it is advantageous to choose between the primal and the dual standard forms since one of the formulations is more natural, has a smaller size, it is faster to solve, or it avoids numerical instability on the software.

Let us consider as an example the SDP relaxation of the maximum cut problem (Section 9.4). We formulated this SDP as a dual standard form $\mathcal{D}$ where the problem size is:

$$\begin{aligned} m &= n \\ \text{nBLOCK} &= 1 \\ \text{bLOCKsTRUCT} &= n. \end{aligned}$$

We can also formulate (mathematically) the same problem as a primal standard form $\mathcal{P}$, too.

Let $\boldsymbol{H}_{ij}$ an $n \times n$ symmetric matrix with (i, j)th and (j, i)th element(s) 1 and all others 0. Problem (3) is equivalent to

$$\begin{array}{ll} \text{minimize} & -2 \displaystyle\sum_{i=1}^{n} \sum_{j>i}^{n} C_{ij} x_{ij} - \sum_{i=1}^{n} C_{ii} x_{ii} \\ \text{subject to} & \displaystyle\sum_{i=1}^{n} \sum_{j=i}^{n} \boldsymbol{H}_{ij} x_{ij} \succeq \boldsymbol{O}, \\ & x_{ii} \geq 1 \ (1 \leq i \leq n), \\ & x_{ii} \leq 1 \ (1 \leq i \leq n), \end{array} \tag{4}$$

where x_{ij} $(1 \leq i \leq n,\ i \leq j \leq n)$ is a variable vector now.

This SDP is in primal standard form $\mathcal{P}$ and its size is

$$\begin{aligned} m &= \frac{n(n+1)}{2} \\ \text{nBLOCK} &= 2 \\ \text{bLOCKsTRUCT} &= (n, -2n) \end{aligned}$$

which seems less advantageous than the dual standard from $\mathcal{D}$ (3). Also the problem (4) in primal standard form $\mathcal{P}$ does not have a strict feasible solution which many cause numerical instability.

In the case of $n = 100$, a typical running time for the formulation (3) in dual standard form is 0.18s, while for the for the formulation (4) in primal standard form is 201.3s.

In some cases, however, a slight increase in the size of the problem can be advantageous if the new formulation has more sparsity in its data.

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 information.
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 iteration 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 an optimal solution, that is, $$\begin{aligned} \text{epsilonStar} &\geq \frac{|\sum_{i=1}^{m} c_i x_i^k - \boldsymbol{F}_0 \bullet \boldsymbol{Y}^k|}{\max\left\{(|\sum_{i=1}^{m} c_i x_i^k| + |\boldsymbol{F}_0 \bullet \boldsymbol{Y}^k|)/2.0,\ 1.0\right\}} \\ &= \frac{|\text{objPrimal} - \text{objDual}|}{\max\{(|\text{objPrimal}| + |\text{objDual}|)/2.0,\ 1.0\}}, \end{aligned}$$
double	epsilonDash	When the primal error (the dual error) becomes smaller than epsilonDash, the SDPA indicates primal feasible (dual feasible), that is, if $$\begin{aligned} \text{epsilonDash} \geq \max\Bigg\{ &\left| [X^k - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^k + \boldsymbol{F}_0]_{pq} \right| \\ &:\ p, q = 1, 2, \ldots, n \ \Bigg\} \end{aligned}$$ then primal feasible and if $$\text{epsilonDash} \geq \max\left\{ \left| \boldsymbol{F}_i \bullet \boldsymbol{Y}^k - c_i \right| \ : i = 1, 2, \ldots, m \right\}$$ 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 no 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 be handled.
Return	void		nothing

- SDPA_initialize2

Synopsis	**SDPA_initialize2(SDPA& SDP)**		
Purpose			Initialize matrices.
Argument	SDPA&	SDP	The instance to be handled.
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 be handled.
	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 nonzero)**		
Purpose			Set the number of nonzero elements in the upper triangular part (including diagonal elements).
Argument	SDPA&	SDP	The instance to be handled.
	int	k	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 including diagonal elements, that is, the number of non zero elements of $\boldsymbol{F}_k.block\{l\}$ to be set by the SDPA_InputElement function.
Return	void		nothing

- SDPA_Make_sfMAT

Synopsis	**SDPA_Make_sfMat(SDPA& SDP)**		
Purpose			Initialize memory to store matrices.
Argument	SDPA&	SDP	The instance to be handled.
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 be handled.
	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 be 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 be handled.
	int	k	The index, that is $\boldsymbol{x}^0_k$.
	double	value	The value to be 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 be handled.
	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 be 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 be handled.
	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 be set, that is $\boldsymbol{Y}^0.block\{l\}(i,j) =$ value (i≤j).
Return	void		nothing

- SDPA_Check_sfMAT

Synopsis	**bool SDPA_Check_sfMat(SDPA& SDP)**		
Purpose			Check the consistence of the input data .
Argument	SDPA&	SDP	The instance to be handled.
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 be handled.
Return	void		nothing

- SDPA_Copy_Current_To_Ini

Synopsis	**SDPA_Copy_Current_To_Ini(SDPA& SDP)**		
Purpose			After we solve by SDPA_Solve, we can copy the current point to an initial point. (See example4.cpp,example5.cpp, example6.cpp)
Argument	SDPA&	SDP	The instance to be handled.
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 6.2.1 Users

The major differences between the SDPA 6.2.1 and the SDPA 7.1.1 are the followings:

- The source code was completely revised, unnecessary variables were eliminated, and auxiliary variables re-used. Consequently, the memory usage became less than half of the previous version.
- It utilizes the sparse Cholesky factorization when the Schur Complement Matrix becomes sparse. For that, it uses the SPOOLES library for sparse matrices [2] to obtain an ordering of rows/columns which possibly produces lesser fill-in. Now, the SDPA can solve much more efficiently SDPs with
  - multiple block diagonal matrices
  - multiple non-negative constraints
- There is a modification in the control subroutine in the interior-point algorithm which improved its numerical stability when compared to the previous version.

Though it should be noted there is a remaining problem:

- The current version does not have a callable library interface as SDPA 6.2.1 does, but it will be implemented in the updated version of the SDPA.

A SDPA-GMP

The SDPA-GMP is an SDP solver intended to solve SDPs very accurately by utilizing the GNU Multiple Precision Arithmetic Library (GMP). Current version of the SDPA-GMP is 7.1.1 which shares the same features with the SDPA except for user settable accuracy usually for extraordinary accurate calculations. Expect for newly added one parameters "precision", user experience is the same as the SDPA. Note that the SDPA-GMP is typically several ten or hundred times slower than the SDPA.

A1 Build and Installation

This subsection describes how to build and install the SDPA-GMP. Usually, the SDPA is distributed as source codes, therefore, users must build it by themselves. We have build and installation tests on Fedora 8, Ubuntu 7.10, MacOSX Leopard and Tiger, and FreeBSD 6/7. You may also possibly build on other UNIX like platforms and Windows cygwin environment.

A2 Prerequisites

It is necessary to have at least the following programs installed on your system except for the MacOSX.

- C and C++ compiler.
- GMP library (http://gmplib.org/).

For MacOSX Leopard, you will need the Xcode 3.0, and you cannot build the SDPA on the Leopard with Xcode 2.5. For MacOSX Tiger, you will need either the Xcode 2.4.1 or the Xcode 2.5.

You can download the Xcode at Apple Developer Connection (**http://developer.apple.com/**) at free of charge.

In the following instructions, we install C/C++ compilers, GMP library and/or Xcode as well. You can skip this part if your system already have them. If not, you may need root access or administrative privilege to your computer. Please ask your system administrator for installing development tools.

A3 How to Obtain the Source Code

Current source code is "sdpa-gmp.7.1.1.src.2008xxxx.tar.gz"

You can obtain the source code following the links at the SDPA homepage:
http://sdpa.indsys.chuo-u.ac.jp/sdpa/index.html

You can find the latest news at about the SDPA at the **index.html** file.

A4 How to Build

A4.1 Fedora 8 and 9

To build the SDPA-GMP, type the following.

```
$ bash
$ su
Password:
# yum update glibc glibc-common
# yum install gcc gcc-c++
# yum install gmp gmp-devel
# exit
$ tar xvfz sdpa-gmp.7.1.1.src.2008xxxx.tar.gz
$ cd sdpa-gmp-7.1.1
$ ./configure
$ make
```

A4.2 MacOSX

If you use Leopard:

> Download and install the Xcode 3 from Apple Developer Connection (**http://developer.apple.com/**) and install it. Note that we support only the Xcode 3 on Leopard. We do not support Leopard with Xcode 2.5.

If you use Tiger:

> Download and install either the Xcode 2.4.1 or the Xcode 2.5 from Apple Developer Connection. First, download GMP from `http://gmplib.org/` install GMP by: (You may change "/Users/maho/gmp" to appropriate one)

```
$ bash
$ tar xvfz gmp-4.2.2.tar.gz
$ cd gmp-4.2.2
$ ./configure --enable-cxx --prefix=/Users/maho/gmp
$ make install
$ make check
```

Then, build the SDPA-GMP by:

```
$ bash
$ tar xvfz sdpa-gmp.7.1.1.src.2008xxxx.tar.gz
$ cd sdpa-gmp-7.1.1
$ CXXFLAGS="-I/Users/maho/gmp/include"; export CXXFLAGS
$ CPPFLAGS="-I/Users/maho/gmp/include"; export CPPFLAGS
$ CFLAGS="-I/Users/maho/gmp/include"; export CFLAGS
$ LDFLAGS="-L/Users/maho/gmp/lib"; export LDFLAGS
$ ./configure
$ make
```

.

A5 How to Install

You will find the "sdpa_gmp" executable binary file at this point, and you can install it as:

```
$ su
Password:
# mkdir /usr/local/bin
# cp sdpa_gmp /usr/local/bin
# chmod 777 /usr/local/bin/sdpa
```

or you can install it to your favorite directory.

```
$ mkdir /home/nakata/bin
$ cp sdpa_gmp /home/nakata/bin
```

A6 Test Run

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

```
$ sdpa_gmp
SDPA-GMP start at    Fri Apr  4 16:02:00 2008

*** Please assign data file and output file.***

---- option type 1 ------------
./sdpa_gmp DataFile OutputFile [InitialPtFile] [-pt parameters]
parameters = 0 default, 1 fast (unstable), 2 slow (stable)
example1-1: ./sdpa_gmp example1.dat example1.result
example1-2: ./sdpa_gmp example1.dat-s example1.result
example1-3: ./sdpa_gmp example1.dat example1.result example1.ini
example1-4: ./sdpa_gmp example1.dat example1.result -pt 2

---- option type 2 ------------
./sdpa_gmp [option filename]+
  -dd : data dense :: -ds : data sparse
  -id : init dense :: -is : init sparse
  -o  : output     :: -p  : parameter
  -pt : parameters , 0 default, 1 fast (unstable)
                     2 slow (stable)
example2-1: ./sdpa_gmp -o example1.result -dd example1.dat
example2-2: ./sdpa_gmp -ds example1.dat-s -o example2.result -p param.sdpa
example2-3: ./sdpa_gmp -ds example1.dat-s -o example3.result -pt 2
```

Let us solve the SDP "example1.dat-s".

```
$ sdpa_gmp -ds example1.dat-s -o example1.out
SDPA-GMP start at    Fri Apr  4 16:05:39 2008
data      is example1.dat-s : sparse
out       is example1.out

set       is DEFAULT
DENSE computations
   mu      thetaP  thetaD  objP      objD      alphaP  alphaD  beta
 0 1.0e+08 1.0e+00 1.0e+00 +0.00e+00 +1.20e+05 1.0e+00 9.0e-01 3.00e-01
 1 1.6e+07 0.0e+00 1.0e-01 +1.10e+05 +1.20e+04 9.0e-01 9.0e-01 3.00e-01
 2 2.5e+06 2.6e-71 9.9e-03 +1.68e+05 +1.15e+03 9.1e-01 9.1e-01 3.00e-01
 3 4.6e+05 4.8e-71 9.4e-04 +2.40e+05 +7.05e+01 3.9e+00 9.6e-01 3.00e-01
 4 7.3e+04 7.8e-71 3.6e-05 +1.00e+05 -3.76e+01 1.4e+00 1.0e+00 3.00e-01
 5 1.5e+04 7.4e-71 5.2e-72 +2.99e+04 -4.19e+01 9.7e-01 9.7e-01 1.00e-01
 6 2.0e+03 7.8e-71 5.9e-72 +3.88e+03 -4.19e+01 9.9e-01 9.9e-01 1.00e-01
 7 2.2e+02 8.1e-71 1.2e-71 +3.93e+02 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
 8 2.2e+01 5.9e-71 1.2e-71 +2.19e+00 -4.19e+01 1.0e+00 9.0e+01 1.00e-01
 9 2.2e+00 6.3e-71 2.4e-70 -3.75e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
10 2.2e-01 6.8e-71 1.8e-71 -4.15e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
11 2.2e-02 7.7e-71 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
12 2.2e-03 7.3e-71 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
13 2.2e-04 7.7e-71 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 9.0e+01 1.00e-01
14 2.2e-05 7.9e-71 3.1e-70 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
15 2.2e-06 8.3e-71 5.9e-72 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
```

```
16 2.2e-07 8.5e-71 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
17 2.2e-08 9.4e-71 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
18 2.2e-09 1.0e-70 5.2e-72 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
19 2.2e-10 1.1e-70 5.6e-72 -4.19e+01 -4.19e+01 1.0e+00 9.0e+01 1.00e-01
20 2.2e-11 1.4e-70 1.5e-69 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
21 2.2e-12 1.4e-70 5.9e-72 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
22 2.2e-13 1.5e-70 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
23 2.2e-14 1.6e-70 5.9e-72 -4.19e+01 -4.19e+01 1.0e+00 9.0e+01 1.00e-01
24 2.2e-15 1.6e-70 4.1e-70 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
25 2.2e-16 1.8e-70 5.9e-72 -4.19e+01 -4.19e+01 1.0e+00 9.0e+01 1.00e-01
26 2.2e-17 2.0e-70 3.6e-70 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
27 2.2e-18 2.3e-70 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
28 2.2e-19 2.7e-70 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 9.0e+01 1.00e-01
29 2.2e-20 3.0e-70 1.3e-69 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
30 2.2e-21 3.3e-70 5.9e-72 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
31 2.2e-22 3.3e-70 5.9e-72 -4.19e+01 -4.19e+01 1.0e+00 9.0e+01 1.00e-01
32 2.2e-23 3.6e-70 3.0e-70 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
33 2.2e-24 3.9e-70 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
34 2.2e-25 4.1e-70 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
35 2.2e-26 4.1e-70 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
36 2.2e-27 4.4e-70 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
37 2.2e-28 4.5e-70 1.2e-71 -4.19e+01 -4.19e+01 1.0e+00 9.0e+01 1.00e-01
38 2.2e-29 4.6e-70 4.9e-70 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01
39 2.2e-30 4.8e-70 1.1e-71 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 1.00e-01

phase.value = pdOPT
   Iteration = 39
          mu = 2.2051719065342495e-30
relative gap = 1.0525880222120523e-31
         gap = 4.4103438130684991e-30
      digits = 3.0977741576287968e+01
objValPrimal = -4.1900000000000000e+01
objValDual   = -4.1900000000000000e+01
p.feas.error = 4.7832028277324550e-66
d.feas.error = 1.1127618452062264e-66
relative eps = 3.7092061506874214e-68
total time   = 0.080
  main loop time = 0.080000
      total time = 0.080000
file   read time = 0.000000
```

A7 Parameters

First we show the default parameter file "param.sdpa" below. Only the difference between Section 5. is the "precision" parameter.

```
200     unsigned int maxIteration;
1.0E-30 double 0.0 < epsilonStar;
1.0E4   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;
0.3     double 0.0 <= betaBar  <  1.0, betaStar <= betaBar;
0.9     double 0.0 < gammaStar  <  1.0;
```

```
1.0E-30 double 0.0 < epsilonDash;
200     precision
```

The file "param.sdpa" needs to have these 11 lines which sets 11 parameters. Each line of this file contains one of the 11 parameters followed by an comment. When the SDPA reads the file "param.sdpa", it neglects the comments.

The newly added "precision" sets the number of significant bits used in the SDPA-GMP. More precicely passing "precision" to "mpf_set_default_prec" function of the GMP library. When precision is "X", then $\log_{10} X$ is the approximate significant digits in the floating point calculation. At default, precision is set to 200, then

$$\log_{10} 2^{200} = 60.205999133$$

Therefore, we calculate approximately floating point numbers with sixty significant digits. If we use double precision, this is approximately sixteen. Note that the GMP is not IEEE754 compliant.

References

[1] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, "LAPACK Users' Guide, Third Edition" *Society for Industrial and Applied Mathematics (1999)* Philadelphia, PA.

[2] C. Ashcraft, D. Pierce, D. K. Wah, and J. Wu, "The reference manual for SPOOLES, release 2.2: An object oriented software library for solving sparse linear systems of equations," Boeing Shared Service Group. Seattle, WA, January 1999. Available at http://www.netlib.org/linalg/spooles/spooles.2.2.html.

[3] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, "Linear Matrix Inequalities in System and Control Theory" *Society for Industrial and Applied Mathematics (1994)* Philadelphia, PA.

[4] K. Fujisawa, M. Kojima, and K. Nakata, "Exploiting sparsity in primal-dual interior-point methods for semidefinite programming," *Mathematical Programming, Series B* **79** (1997) 235–253.

[5] K. Fujisawa, K. Nakata, M. Yamashita, and M. Fukuda, "SDPA project: Solving large-scale semidefinite programs," *Journal of Operations Research Society of Japan* **50** (2007) 278–298.

[6] H. D. Mittelmann, "An independent benchmarking of SDP and SOCP solvers," *Mathematical Programming, Series B* **95** (2003) 407–430.