NAG Library Function Document

nag_opt_sdp_read_sdpa (e04rdc)

1 Purpose

nag_opt_sdp_read_sdpa (e04rdc) reads in a linear semidefinite programming problem (SDP) from a file in sparse SDPA format and returns it in the form which is usable by functions nag_opt_handle_init (e04rac) (initialization), nag_opt_handle_set_linobj (e04rec) (linear objective function), nag_opt_handle_set_linmatineq (e04rnc) (linear matrix constraints), nag_opt_handle_solve_pennon (e04svc) (solver) and nag_opt_handle_free (e04rzc) (deallocation) from the NAG optimization modelling suite.

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_sdp_read_sdpa (Nag_FileID infile, Integer maxnvar,
     Integer maxnblk, Integer maxnnz, Integer filelst, Integer *nvar,
     Integer *nblk, Integer *nnz, double cvec[], Integer nnza[],
     Integer irowa[], Integer icola[], double a[], Integer blksizea[],
     NagError *fail)
```

3 Description

nag_opt_sdp_read_sdpa (e04rdc) is capable of reading linear semidefinite programming problems (SDP) from a text file in sparse SDPA format. The problem is captured and returned in the following form:

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i A_i - A_0 \succeq 0, & \text{(b)} \end{aligned} \tag{1}$$

where A_i denotes symmetric matrices and c is a vector. The expression $S \succeq 0$ stands for a constraint on the eigenvalues of a symmetric matrix S, namely, all the eigenvalues should be non-negative, i.e., the matrix S should be positive semidefinite.

Please note that this form covers even general linear SDP formulations with multiple linear matrix inequalities and a set of standard linear constraints. A set of m_A linear matrix inequalities

$$\sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1, \ldots, m_A \tag{2}$$

can be equivalently expressed as one matrix inequality (1)(b) in the following block diagonal form where the matrices $A_i^1, A_i^2, \ldots, A_i^{m_A}$ create the diagonal blocks of A_i:

$$\sum_{i=1}^{n} x_i \begin{pmatrix} A_i^1 & & & \\ & A_i^2 & & \\ & & \ddots & \\ & & & A_i^{m_A} \end{pmatrix} - \begin{pmatrix} A_0^1 & & & \\ & A_0^2 & & \\ & & \ddots & \\ & & & A_0^{m_A} \end{pmatrix} \succeq 0.$$

In addition, notice that if all matrices A_i^k belonging to the same block, say block k, are themselves diagonal matrices (or have dimension 1×1), the associated matrix inequality

$$\sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0 \tag{3}$$

defines actually a standard linear constraint

$$Bx \geq l$$

where l and columns of the matrix B are formed by the diagonals of matrices A_0^k and $A_1^k, \ldots, A_n^k$, respectively. Precisely, $l_i = \left(A_0^k\right)_{ii}$ and $b_{ij} = \left(A_j^k\right)_{ii}$. See Section 10.

3.1 Sparse SDPA file format

The problem data is written in an ASCII input file in a SDPA sparse format which was first introduced in Fujisawa *et al.* (1998). In the description below we follow closely the specification from Borchers (1999).

The format is line oriented. If more elements are required on the line they need to be separated by a space, a tab or any of the special characters ‘,’, ‘(’, ‘)’, ‘{’ or ‘}’. The file consists of six sections:

1. Comments. The file can begin with arbitrarily many lines of comments. Each line of comments must begin with ‘"’ or ‘*’.
2. The first line after the comments contains integer n, the number of variables. The rest of this line is ignored.
3. The second line after the comments contains integer m_A, the number of blocks in the block diagonal structure of the matrices. Additional text on this line after m_A is ignored.
4. The third line after the comments contains a vector of m_A integers that give the sizes of the individual blocks. Negative numbers may be used to indicate that a block is actually a diagonal submatrix. Thus a block size of ‘-5’ indicates a 5 by 5 block in which only the diagonal elements are nonzero.
5. The fourth line after the comments contains an n-dimensional real vector defining the objective function vector c.
6. The remaining lines of the file contain nonzero entries in the constraint matrices, with one entry per line. The format for each line is

   matno ***blkno*** ***i*** ***j*** ***entry***

   where ***matno*** is the number $(0, \ldots, n)$ of the matrix to which this entry belongs and ***blkno*** specifies the block number $k = 1, 2, \ldots, m_A$ within this matrix. Together, they uniquely identify the block $A_{\boldsymbol{matno}}^{\boldsymbol{blkno}}$. Integers ***i*** and ***j*** are one-based indices which specify a location of the entry within the block. Note that since all matrices are assumed to be symmetric, only entries in the upper triangle of a matrix should be supplied. Finally, ***entry*** should give the real value of the entry in the matrix. Precisely, $\left(A_{\boldsymbol{matno}}^{\boldsymbol{blkno}}\right)_{\boldsymbol{ij}} = \left(A_{\boldsymbol{matno}}^{\boldsymbol{blkno}}\right)_{\boldsymbol{ji}} = \boldsymbol{entry}$.

In the text below and in the file listing (**filelst**) we use the word ‘token’ as a reference to a group of contiguous characters without a space or any other delimeters.

3.2 Recommendation on how best to use nag_opt_sdp_read_sdpa (e04rdc)

(a) The input file with the problem needs to be opened for reading by nag_open_file (x04acc) (**mode** $= 0$). Setting **filelst** $= 1$ might help with possible file formatting errors.

(b) Unless the dimension of the problem (or its overestimate) is known in advance, call nag_opt_sdp_read_sdpa (e04rdc) initially with **maxnvar** $= 0$, **maxnblk** $= 0$ and **maxnnz** $= 0$. In this case the exact size of the problem is computed and returned in **nvar**, **nblk** and **nnz**. No other data will be stored and none of the arrays **cvec**, **nnza**, **irowa**, **icola**, **a**, **blksizea** will be referenced and may be **NULL**. Then the exact storage can be allocated and the file reopened. When nag_opt_sdp_read_sdpa (e04rdc) is called for the second time, the problem is read in and stored in appropriate arrays.

(c) The example in Section 10 shows a typical sequence of calls to solve the problem read in by nag_opt_sdp_read_sdpa (e04rdc). First an empty handle needs to be initialized by nag_opt_han dle_init (e04rac) with **nvar** variables. This should be followed by calls to nag_opt_handle_set_li nobj (e04rec) and nag_opt_handle_set_linmatineq (e04rnc) to formulate the objective function and

the constraints, respectively. The arguments of both routines use the same naming and storage as in nag_opt_sdp_read_sdpa (e04rdc) so the variables can be passed unchanged; only **dima** in nag_opt_handle_set_linmatineq (e04rnc) is new and should equal to sum **blksizea**$[0] + \cdots +$ **blksizea**[**nblk** $- 1]$ and **nnzasum** in nag_opt_handle_set_linmatineq (e04rnc) is the same as **nnz** in nag_opt_sdp_read_sdpa (e04rdc). You may at this point want to modify option settings using nag_opt_handle_opt_set (e04zmc). If dual variables (Lagrangian multipliers) are required from the solver, sufficient space needs to be allocated. The size is equal to the sum of the number of elements of dense triangular matrices for each block. For further details, see the argument **ua** of the solver nag_opt_handle_solve_pennon (e04svc). The solver should be called and then followed, finally, by a call to nag_opt_handle_free (e04rzc) to deallocate memory associated with the problem.

4 References

Borchers B (1999) SDPLIB 1.2, A Library of semidefinite programming test problems. *Optimization Methods and Software* **11(1)** 683–690 http://euler.nmt.edu/~brian/sdplib/

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

5 Arguments

1: **infile** – Nag_FileID *Input*

On entry: the file identifier associated with the sparse SDPA data file. **Note:** that the file needs to be opened in read mode by nag_open_file (x04acc) with **mode** $= 0$.

2: **maxnvar** – Integer *Input*

On entry: the upper limit for the number of variables in the problem. If it is set to zero, **cvec** and **nnza** will not be referenced and may be **NULL**.

Constraint: **maxnvar** ≥ 0.

3: **maxnblk** – Integer *Input*

On entry: the upper limit for the number of matrix constraints (i.e., the number of diagonal blocks within the matrix). If it is set to zero, **blksizea** will not be referenced and may be **NULL**.

Constraint: **maxnblk** ≥ 0.

4: **maxnnz** – Integer *Input*

On entry: the upper limit on the sum of nonzeros in all matrices A_i^k, for $i = 0, 1, \ldots,$ **nvar** and $k = 1, 2, \ldots,$ **nblk**. If it is set to zero, **irowa**, **icola** and **a** will not be referenced and may be **NULL**.

Constraint: **maxnnz** ≥ 0.

5: **filelst** – Integer *Input*

On entry: if **filelst** $\neq 0$, a listing of the input data is sent to stdout. This can be useful for debugging the data file.

If **filelst** $= 0$, no listing is produced.

6: **nvar** – Integer * *Output*
7: **nblk** – Integer * *Output*
8: **nnz** – Integer * *Output*

On exit: the actual number of the variables n, matrix constraints m_A and number of nonzeros of the problem in the file. This also indicates the exact memory needed in **cvec**, **nnza**, **irowa**, **icola**, **a** and **blksizea**.

9: **cvec**[**maxnvar**] – double *Output*

On exit: **cvec**$[i-1]$, for $i = 1, 2, \ldots,$ **nvar**, stores the dense vector c of the linear objective function. If **maxnvar** $= 0$, **cvec** is not referenced and may be **NULL**.

10: **nnza**[**maxnvar** + **1**] – Integer *Output*

On exit: **nnza**$[i]$, for $i = 0, 1, \ldots,$ **nvar**, stores the number of nonzero elements in matrices A_i. If **maxnvar** $= 0$, **nnza** is not referenced and may be **NULL**.

11: **irowa**[**maxnnz**] – Integer *Output*
12: **icola**[**maxnnz**] – Integer *Output*
13: **a**[**maxnnz**] – double *Output*

On exit: **irowa**, **icola** and **a** store the nonzeros in the upper triangle of matrices A_i, for $i = 0, 1, \ldots,$ **nvar**, in the coordinate storage, i.e., **irowa**$[j-1]$ are one-based row indices, **icola**$[j-1]$ are one-based column indices and **a**$[j-1]$ are the values of the nonzero elements, for $j = 1, 2, \ldots,$ **nnz**. See Section 9. If **maxnnz** $= 0$, the arrays are not referenced and may be **NULL**.

14: **blksizea**[**maxnblk**] – Integer *Output*

On exit: **blksizea**$[k-1]$, for $k = 1, 2, \ldots,$ **nblk**, stores the sizes of the diagonal blocks in matrices A_i from the top to the bottom. If **maxnblk** $= 0$, **blksizea** is not referenced and may be **NULL**.

15: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_DIAG_ELEMENTS

Invalid structural data found on line ⟨*value*⟩.
The specified element belongs to a diagonal block but is not diagonal.
The row index is ⟨*value*⟩ and column index is ⟨*value*⟩.

NE_DUPLICATE_ELEMENT

An entry in the constraints with ***matno*** = ⟨*value*⟩, ***blkno*** = ⟨*value*⟩, row index ⟨*value*⟩ and column index ⟨*value*⟩ was defined more than once. All entries need to be unique.

NE_FILE_INCOMPLETE

A premature end of the input stream. The part defining the dimensions of the blocks was not found.

A premature end of the input stream. The part defining the nonzero entries was not found.

A premature end of the input stream. The part defining the number of blocks was not found.

A premature end of the input stream. The part defining the number of variables was not found.

A premature end of the input stream. The part defining the objective function was not found.

NE_FILEID

On entry, **infile** = ⟨*value*⟩.
Constraint: **infile** ≥ 0.

NE_INT

An invalid number of blocks was given on line ⟨*value*⟩.
The number stated there is ⟨*value*⟩ and needs to be at least 1.

An invalid number of variables was given on line ⟨*value*⟩.
The number stated there is ⟨*value*⟩ and needs to be at least 1.

NE_INT_ARG

On entry, **maxnblk** = ⟨*value*⟩.
Constraint: **maxnblk** ≥ 0.

On entry, **maxnnz** = ⟨*value*⟩.
Constraint: **maxnnz** ≥ 0.

On entry, **maxnvar** = ⟨*value*⟩.
Constraint: **maxnvar** ≥ 0.

NE_INT_ARRAY

An invalid size of the block number ⟨*value*⟩ was given on line ⟨*value*⟩.
The number stated there is ⟨*value*⟩ and needs to be nonzero.

NE_INT_MAX

At least one of **maxnvar**, **maxnblk** or **maxnnz** is too small. Suggested values are returned in **nvar**, **nblk** and **nnz**, respectively.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_INVALID_CS

Invalid structural data found on line ⟨*value*⟩.
The given column index is out of bounds, it must respect the size of the block. Its value ⟨*value*⟩ must be between ⟨*value*⟩ and ⟨*value*⟩ (inclusive).

Invalid structural data found on line ⟨*value*⟩.
The given row index is out of bounds, it must respect the size of the block. Its value ⟨*value*⟩ must be between ⟨*value*⟩ and ⟨*value*⟩ (inclusive).

Invalid structural data found on line ⟨*value*⟩.
The specified nonzero element is not in the upper triangle.
The row index is ⟨*value*⟩ and column index is ⟨*value*⟩.

NE_INVALID_FORMAT

The token on line ⟨*value*⟩ at position ⟨*value*⟩ to ⟨*value*⟩ was not recognized as a valid integer.

The token on line ⟨*value*⟩ at position ⟨*value*⟩ to ⟨*value*⟩ was not recognized as a valid real number.

The token on line ⟨*value*⟩ starting at position ⟨*value*⟩ was too long and was not recognized.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_NOT_READ_FILE

The input stream seems to be empty. No data was read. This might indicate a problem with opening the file, check that nag_open_file (x04acc) was used correctly.

NE_OUT_OF_RANGE

Invalid structural data found on line ⟨*value*⟩.
The given block number is out of bounds. Its value ⟨*value*⟩ must be between 1 and m_A (inclusive).

Invalid structural data found on line ⟨*value*⟩.
The given matrix number is out of bounds. Its value ⟨*value*⟩ must be between 0 and n (inclusive).

NE_READ_ERROR

Reading from the stream caused an unknown error on line ⟨*value*⟩.

NE_WRONG_NUM_TOKENS

Not enough data was given on line ⟨*value*⟩ specifying block sizes.
Expected m_A tokens but found only ⟨*value*⟩.

Not enough data was given on line ⟨*value*⟩ specifying nonzero matrix elements.
Expected ⟨*value*⟩ tokens but found only ⟨*value*⟩.

Not enough data was given on line ⟨*value*⟩ specifying the objective function.
Expected n tokens but found only ⟨*value*⟩.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_opt_sdp_read_sdpa (e04rdc) is not threaded in any implementation.

9 Further Comments

The following artificial example demonstrates how the elements of A_i matrices are organized within arrays **nnza**, **irowa**, **icola** and **a**. For simplicity let us assume that **nblk** $= 1$, **blksizea**$[0] = 3$ and **nvar** $= 4$. Please note that the values of the elements were chosen to ease readability rather than to define a valid problem.

Let the matrix constraint (1)(b) be defined by

$$A_0 = \begin{pmatrix} 0 & 0.1 & 0 \\ 0.1 & 0 & 0.2 \\ 0 & 0.2 & 0.3 \end{pmatrix},$$

$$A_1 = \begin{pmatrix} 1.1 & 0 & 0 \\ 0 & 1.2 & 1.3 \\ 0 & 1.3 & 1.4 \end{pmatrix},$$

$$A_2 \text{ empty},$$

$$A_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 3.1 & 0 \\ 0 & 0 & 3.2 \end{pmatrix},$$

$$A_4 = \begin{pmatrix} 4.1 & 4.2 & 4.3 \\ 4.2 & 0 & 0 \\ 4.3 & 0 & 0 \end{pmatrix}.$$

All matrices A_i have to be symmetric and therefore only the elements in the upper triangles are stored. The table below shows how the arrays would be populated.

irowa	1 2 3	1 2 2 3		2 3	1 2 3
icola	2 3 3	1 2 3 3		2 3	1 1 1
a	0.1 0.2 0.3	1.1 1.2 1.3 1.4		3.1 3.2	4.1 4.2 4.3
	A_0	A_1	A_2	A_3	A_4
nnza	3	4	0	2	3

See also Section 3 in nag_opt_handle_set_linmatineq (e04rnc) which accepts the same format.

10 Example

The following example comes from Fujisawa *et al.* (1998).

Imagine that we want to store the following problem in a file in the SDPA format.

$$\begin{aligned} \underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & 10x_1 + 20x_2 \\ \text{subject to} \quad & \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \geq \begin{pmatrix} 1 \\ 1.5 \end{pmatrix} \\ & \begin{pmatrix} 5 & 2 \\ 2 & 6 \end{pmatrix} x_2 - \begin{pmatrix} 3 & 0 \\ 0 & 4 \end{pmatrix} \succeq 0. \end{aligned}$$

There are two variables ($n = 2$) in the problem. One linear matrix constraint and one block of linear constraints can be formed as (1) with two diagonal blocks ($m_A = 2$). Both blocks have dimension 2 but the first one (defining linear constraints) is only diagonal, thus the sizes will be stated as $-2 \quad 2$.

The problem can be rewritten as

$$\begin{aligned} \underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & c^{\mathrm{T}} x \\ \text{subject to} \quad & A_1 x_1 + A_2 x_2 - A_0 \succeq 0 \end{aligned}$$

where

$$c = \begin{pmatrix} 10 & 20 \end{pmatrix}^{\mathrm{T}},$$

$$A_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix},$$

$$A_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 2 \\ 0 & 0 & 2 & 6 \end{pmatrix}.$$

The optimal solution is $x^* = \begin{pmatrix} 1.0 & 1.0 \end{pmatrix}^{\mathrm{T}}$ with the objective function value 30.0. The optimal Lagrangian multipliers (dual variables) are 10.0, 0.0 and $\begin{pmatrix} 20/7, & -20/7 \\ -20/7, & 20/7 \end{pmatrix}$.

See also Section 10 in nag_opt_handle_init (e04rac) for links to further examples in the suite.

10.1 Program Text

```
/* nag_opt_sdp_read_sdpa (e04rdc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

/* Load a linear semidefinite programming problem from a sparse SDPA
 * file, formulate the problem via a handle, pass it to the solver
 * and print both primal and dual variables.
 */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagx04.h>

int main(int argc, char *argv[])
{
  char fname_default[] = "e04rdce.opt";
  Integer filelst = 0;

  Integer exit_status = 0;
  Integer dima, idblk, idx, inform, j, k, maxnblk, maxnnz, maxnvar,
         nblk, nnz, nnzu, nnzuc, nnzua, nvar;
  char *fname = 0;
  char title[60];
  double rinfo[32], stats[32];
  double *a = 0, *cvec = 0, *ua = 0, *x = 0;
  Integer *blksizea = 0, *icola = 0, *irowa = 0, *nnza = 0;
  void *handle = 0;

  /* Nag Types */
  Nag_FileID infile;
  NagError fail;

  printf("nag_opt_sdp_read_sdpa (e04rdc) Example Program Results\n\n");

  /* Use the first command line argument as the filename or
   * choose default filename stored in 'fname_default'. */
  if (argc > 1)
    fname = argv[1];
  else
    fname = fname_default;
  printf("Reading SDPA file: %s\n", fname);
  fflush(stdout);

  /* nag_open_file (x04acc).
   * Open unit number for reading and associate unit with named file. */
  nag_open_file(fname, 0, &infile, NAGERR_DEFAULT);

  /* Go through the file for the first time to find the dimension
   * of the problem. Unless the file format is wrong, the function
   * should finish with fail.code = NE_INT_MAX (not enough space). */
```

```
/* nag_opt_sdp_read_sdpa (e04rdc).
 * A reader of sparse SDPA data files for linear SDP problems. */
INIT_FAIL(fail);
nag_opt_sdp_read_sdpa(infile, 0, 0, 0, filelst, &nvar, &nblk, &nnz,
                      NULL, NULL, NULL, NULL, NULL, NULL, &fail);

/* nag_close_file (x04adc).
 * Close file associated with given unit number. */
nag_close_file(infile, NAGERR_DEFAULT);

if (fail.code != NE_INT_MAX) {
  /* Possible problem with formatting, etc. Stop. */
  printf("Error from nag_opt_sdp_read_sdpa (e04rdc).\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Allocate the right size of arrays for the data. */
printf("Allocating space for the problem.\n");
printf("nvar = %" NAG_IFMT "\n", nvar);
printf("nblk = %" NAG_IFMT "\n", nblk);
printf("nnz  = %" NAG_IFMT "\n", nnz);
fflush(stdout);
maxnvar = nvar;
maxnblk = nblk;
maxnnz = nnz;
if (!(cvec = NAG_ALLOC(maxnvar, double)) ||
    !(nnza = NAG_ALLOC(maxnvar + 1, Integer)) ||
    !(irowa = NAG_ALLOC(maxnnz, Integer)) ||
    !(icola = NAG_ALLOC(maxnnz, Integer)) ||
    !(a = NAG_ALLOC(maxnnz, double)) ||
    !(blksizea = NAG_ALLOC(maxnblk, Integer)))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* Reopen the same file. */
nag_open_file(fname, 0, &infile, NAGERR_DEFAULT);

/* Read the problem data, there should be enough space this time.
 * nag_opt_sdp_read_sdpa (e04rdc).
 * A reader of sparse SDPA data files for linear SDP problems. */
nag_opt_sdp_read_sdpa(infile, maxnvar, maxnblk, maxnnz, filelst, &nvar,
                      &nblk, &nnz, cvec, nnza, irowa, icola, a, blksizea,
                      NAGERR_DEFAULT);

/* nag_close_file (x04adc).
 * Close file associated with given unit number. */
nag_close_file(infile, NAGERR_DEFAULT);

/* Problem was successfully decoded. */
printf("Linear SDP problem was read, start formulating the problem\n");
fflush(stdout);

/* nag_opt_handle_init (e04rac).
 * Initialize an empty problem handle with nvar variables. */
nag_opt_handle_init(&handle, nvar, NAGERR_DEFAULT);

/* nag_opt_handle_set_linobj (e04rec).
 * Add a linear objective function to the formulation. */
nag_opt_handle_set_linobj(handle, nvar, cvec, NAGERR_DEFAULT);

dima = 0;
for (k = 0; k < nblk; k++)
  dima += blksizea[k];
idblk = 0;
/* nag_opt_handle_set_linmatineq (e04rnc).
 * Add all linear matrix constraints to the formulation.*/
```

```
  nag_opt_handle_set_linmatineq(handle, nvar, dima, nnza, nnz, irowa, icola,
                                a, nblk, blksizea, &idblk, NAGERR_DEFAULT);

  printf("The problem formulation in a handle is completed.\n\n");
  fflush(stdout);

  /* nag_opt_handle_print (e04ryc).
   * Print overview of the handle. */
  nag_opt_handle_print(handle, 6, "Overview", NAGERR_DEFAULT);

  /* Set optional arguments of the solver by calling
   * nag_opt_handle_opt_set (e04zmc). */
  nag_opt_handle_opt_set(handle, "DIMACS Measures = Check", NAGERR_DEFAULT);
  nag_opt_handle_opt_set(handle, "Initial X = Automatic", NAGERR_DEFAULT);

  /* Compute memory needed for primal & dual variables. */

  /* There are no box constraints or linear constraints set
   * by e04rhc or by e04rjc, neither second order cone constraints.*/
  nnzu = 0;
  nnzuc = 0;

  /* Count size of the matrix multipliers, they are stored as packed
   * triangles respecting the block structure. */
  nnzua = 0;
  for (k = 0; k < nblk; k++)
    nnzua += blksizea[k] * (blksizea[k] + 1) / 2;

  if (!(x = NAG_ALLOC(nvar, double)) || !(ua = NAG_ALLOC(nnzua, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Call the solver nag_opt_handle_solve_pennon (e04svc). */
  INIT_FAIL(fail);
  nag_opt_handle_solve_pennon(handle, nvar, x, nnzu, NULL, nnzuc, NULL,
                              nnzua, ua, rinfo, stats, &inform, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_opt_handle_solve_pennon (e04svc).\n%s\n",
           fail.message);
    exit_status = 2;
    goto END;
  }

  /* Print results. */
  printf("\nOptimal solution x:\n");
  for (j = 0; j < nvar; j++)
    printf("  %f\n", x[j]);
  fflush(stdout);

  /* Print packed lower triangles of the Lagrangian multipliers. */
  idx = 0;
  for (k = 0; k < nblk; k++) {
#ifdef _WIN32
    sprintf_s(title, (unsigned)_countof(title),
              "Lagrangian multiplier for A_%" NAG_IFMT, k);
#else
    sprintf(title, "Lagrangian multiplier for A_%" NAG_IFMT, k);
#endif
    nnz = blksizea[k] * (blksizea[k] + 1) / 2;
    /* nag_pack_real_mat_print (x04ccc).
     * Print real packed triangular matrix. */
    nag_pack_real_mat_print(Nag_ColMajor, Nag_Lower, Nag_NonUnitDiag,
                            blksizea[k], ua + idx, title, NULL,
                            NAGERR_DEFAULT);
    idx = idx + nnz;
  }

END:
```

```
  /* nag_opt_handle_free (e04rzc).
   * Destroy the problem handle and deallocate all the memory. */
  if (handle)
    nag_opt_handle_free(&handle, NAGERR_DEFAULT);

  NAG_FREE(a);
  NAG_FREE(cvec);
  NAG_FREE(ua);
  NAG_FREE(x);
  NAG_FREE(blksizea);
  NAG_FREE(icola);
  NAG_FREE(irowa);
  NAG_FREE(nnza);
  return exit_status;
}
```

10.2 Program Data

```
" nag_opt_sdp_read_sdpa (e04rdc) Example Program Data
2 =mdim
2 =nblocks
{-2, 2}
10.0 20.0
0 1 1 1 1.0
0 1 2 2 1.5
0 2 1 1 3.0
0 2 2 2 4.0
1 1 1 1 1.0
1 1 2 2 1.0
2 1 2 2 1.0
2 2 1 1 5.0
2 2 1 2 2.0
2 2 2 2 6.0
```

10.3 Program Results

```
nag_opt_sdp_read_sdpa (e04rdc) Example Program Results

Reading SDPA file: e04rdce.opt
Allocating space for the problem.
nvar = 2
nblk = 3
nnz  = 10
Linear SDP problem was read, start formulating the problem
The problem formulation in a handle is completed.

 Overview
   Status:                 Problem and option settings are editable.
   No of variables:        2
   Objective function:     linear
   Simple bounds:          not defined yet
   Linear constraints:     not defined yet
   Nonlinear constraints:  not defined yet
   Matrix constraints:     3
 E04SV, NLP-SDP Solver (Pennon)
 ------------------------------
 Number of variables             2                 [eliminated           0]
                          simple  linear  nonlin
 (Standard) inequalities       0       2       0
 (Standard) equalities                 0       0
 Matrix inequalities                   1       0 [dense    1, sparse    0]
                                                 [max dimension         2]

 Begin of Options
     Outer Iteration Limit        =                100     * d
     Inner Iteration Limit        =                100     * d
     Infinite Bound Size          =         1.00000E+20     * d
     Initial X                    =          Automatic     * U
```

```
     Initial U                     =           Automatic     * d
     Initial P                     =           Automatic     * d
     Hessian Density               =               Dense     * S
     Init Value P                  =         1.00000E+00     * d
     Init Value Pmat               =         1.00000E+00     * d
     Presolve Block Detect         =                 Yes     * d
     Print File                    =                   6     * d
     Print Level                   =                   2     * d
     Print Options                 =                 Yes     * d
     Monitoring File               =                  -1     * d
     Monitoring Level              =                   4     * d
     Monitor Frequency             =                   0     * d
     Stats Time                    =                  No     * d
     P Min                         =         1.05367E-08     * d
     Pmat Min                      =         1.05367E-08     * d
     U Update Restriction          =         5.00000E-01     * d
     Umat Update Restriction       =         3.00000E-01     * d
     Preference                    =               Speed     * d
     Transform Constraints         =                  No     * S
     Dimacs Measures               =               Check     * U
     Stop Criteria                 =                Soft     * d
     Stop Tolerance 1              =         1.00000E-06     * d
     Stop Tolerance 2              =         1.00000E-07     * d
     Stop Tolerance Feasibility    =         1.00000E-07     * d
     Linesearch Mode               =            Fullstep     * S
     Inner Stop Tolerance          =         1.00000E-02     * d
     Inner Stop Criteria           =           Heuristic     * d
     Task                          =            Minimize     * d
     P Update Speed                =                  12     * d
 End of Options
 ------------------------------------------------------------
  it|  objective |  optim  |   feas  |  compl  | pen min |inner
 ------------------------------------------------------------
   0  0.00000E+00  4.06E+01  4.00E+00  3.16E+01  1.00E+00   0
   1  4.02661E+01  1.07E-01  2.78E-01  1.52E+01  1.00E+00   5
   2  2.90783E+01  6.52E-02  9.77E-02  2.78E+00  4.65E-01   5
   3  2.84228E+01  1.67E-01  2.39E-01  7.76E-01  2.16E-01   2
   4  2.97263E+01  3.98E-02  4.39E-02  2.05E-01  1.01E-01   3
   5  2.99618E+01  5.01E-02  6.40E-03  3.32E-02  4.68E-02   2
   6  2.99934E+01  1.45E-01  1.25E-03  6.23E-03  2.18E-02   1
   7  2.99999E+01  3.31E-02  1.28E-05  4.16E-04  1.01E-02   1
   8  3.00001E+01  9.97E-05  3.01E-07  9.67E-05  4.71E-03   1
   9  3.00000E+01  1.37E-04  3.25E-08  2.25E-05  2.19E-03   1
  10  3.00000E+01  1.16E-05  3.52E-09  5.23E-06  1.02E-03   1
  11  3.00000E+01  1.13E-06  3.81E-10  1.22E-06  4.74E-04   1
 ------------------------------------------------------------
 Status: converged, an optimal solution found
 ------------------------------------------------------------
 Final objective value               3.000000E+01
 Relative precision                  3.941484E-08
 Optimality                          1.133096E-06
 Feasibility                         3.806810E-10
 Complementarity                     1.216064E-06
 DIMACS error 1                      5.395697E-08
 DIMACS error 2                      0.000000E+00
 DIMACS error 3                      0.000000E+00
 DIMACS error 4                      7.613621E-11
 DIMACS error 5                      4.324629E-09
 DIMACS error 6                      2.296238E-08
 Iteration counts
   Outer iterations                            11
   Inner iterations                            23
   Linesearch steps                            50
 Evaluation counts
   Augm. Lagr. values                          35
   Augm. Lagr. gradient                        35
   Augm. Lagr. hessian                         23
 ------------------------------------------------------------

Optimal solution x:
  1.000000
```

```
 1.000000
Lagrangian multiplier for A_0
           1
1     10.0000
Lagrangian multiplier for A_1
             1
1    2.4321e-06
Lagrangian multiplier for A_2
           1          2
1      2.8571
2     -2.8571     2.8571
```