NAG Library Routine Document

E04RDF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

E04RDF 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 routines E04RAF (initialization), E04REF (linear objective function), E04RNF (linear matrix constraints), E04SVF (solver) and E04RZF (deallocation) from the NAG optimization modelling suite.

2 Specification

```
SUBROUTINE E04RDF (INFILE, MAXNVAR, MAXNBLK, MAXNNZ, FILELST, NVAR,        &
                   NBLK, NNZ, CVEC, NNZA, IROWA, ICOLA, A, BLKSIZEA,       &
                   IFAIL)
INTEGER            INFILE, MAXNVAR, MAXNBLK, MAXNNZ, FILELST, NVAR,        &
                   NBLK, NNZ, NNZA(MAXNVAR+1), IROWA(MAXNNZ),              &
                   ICOLA(MAXNNZ), BLKSIZEA(MAXNBLK), IFAIL
REAL (KIND=nag_wp) CVEC(MAXNVAR), A(MAXNNZ)
```

3 Description

E04RDF 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 E04RDF

(a) The input file with the problem needs to be opened for reading by X04ACF (MODE $= 0$). In this way we avoid possible limitations of maximal lengths of lines inherited by Fortran I/O (X04ACF uses the formatted stream access mode to bypass the restriction). If the file is opened by other means or standard input is used instead, lines within the file might be truncated which would produce a file format error message. This would most likely happen on the line defining the objective function. 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 E04RDF 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. Then the exact storage can be allocated and the file reopened. When E04RDF 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 E04RDF. First an empty handle needs to be initialized by E04RAF with NVAR variables. This should be followed by calls to E04REF and E04RNF to formulate the objective function and the constraints, respectively. The arguments of both routines use the same naming and storage as in E04RDF so the variables can be passed unchanged; only DIMA in E04RNF is new and should equal to SUM(BLKSIZEA(1 : NBLK)) and NNZASUM in E04RNF is the same as NNZ in E04RDF. You may at this point want to modify option settings using E04ZMF. 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 E04SVF. The solver should be called and then followed, finally, by a call to E04RZF 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 – INTEGER *Input*

On entry: the unit number associated with the sparse SDPA data file. **Note:** that the file needs to be opened in read mode by X04ACF with MODE $= 0$.

Constraint: INFILE ≥ 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.

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.

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.

Constraint: MAXNNZ ≥ 0.

5: FILELST – INTEGER *Input*

On entry: if FILELST $\neq 0$, a listing of the input data is sent to the current advisory message unit (as defined by X04ABF). 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) – REAL (KIND=nag_wp) array *Output*

On exit: CVEC(i), for $i = 1, 2, \ldots,$ NVAR, stores the dense vector c of the linear objective function.

10: NNZA(MAXNVAR + 1) – INTEGER array *Output*

On exit: NNZA($i + 1$), for $i = 0, 1, \ldots,$ NVAR, stores the number of nonzero elements in matrices A_i.

11: IROWA(MAXNNZ) – INTEGER array *Output*
12: ICOLA(MAXNNZ) – INTEGER array *Output*
13: A(MAXNNZ) – REAL (KIND=nag_wp) array *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) are one-based row indices, ICOLA(j) are one-based column indices and A(j) are the values of the nonzero elements, for $j = 1, 2, \ldots,$ NNZ. See Section 9.

14: BLKSIZEA(MAXNBLK) – INTEGER array *Output*

On exit: BLKSIZEA(k), for $k = 1, 2, \ldots,$ NBLK, stores the sizes of the diagonal blocks in matrices A_i from the top to the bottom.

15: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, because for this routine the values of the output arguments may be useful even if IFAIL $\neq 0$ on exit, the recommended value is -1. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Note: E04RDF may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the routine:

IFAIL $= 1$

At least one of MAXNVAR, MAXNBLK or MAXNNZ is too small. Suggested values are returned in NVAR, NBLK and NNZ, respectively.

IFAIL $= 2$

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

IFAIL $= 3$

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

IFAIL $= 4$

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

IFAIL $= 5$

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

IFAIL $= 6$

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

IFAIL $= 7$

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

IFAIL $= 8$

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

IFAIL $= 9$

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

IFAIL $= 10$

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

IFAIL $= 11$

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).

IFAIL $= 12$

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).

IFAIL $= 13$

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).

IFAIL $= 14$

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).

IFAIL $= 15$

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*⟩.

IFAIL $= 16$

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*⟩.

IFAIL $= 17$

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.

IFAIL $= 18$

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.

IFAIL $= 19$

The input stream seems to be empty. No data was read. This might indicate a problem with opening the file, check that X04ACF was used correctly.

IFAIL $= 20$

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

IFAIL $= 21$

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

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

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

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

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

E04RDF 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$(1) = 3$ 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 E04RNF 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.

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

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

$$\begin{array}{ll} \underset{x\in\mathbb{R}^2}{\text{minimize}} & c^{\mathrm{T}}x \\ \text{subject to} & A_1x_1 + A_2x_2 - A_0 \succeq 0 \end{array}$$

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 E04RAF for links to further examples in the suite.

10.1 Program Text

```
    Program e04rdfe

!     E04RDF Example Program Text
!     Mark 26 Release. NAG Copyright 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.

!     .. Use Statements ..
      Use nag_library, Only: e04raf, e04rdf, e04ref, e04rnf, e04ryf, e04rzf,   &
                             e04svf, e04zmf, nag_wp, x04acf, x04adf, x04ccf
      Use, Intrinsic                   :: iso_c_binding, Only: c_ptr
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: filelst = 0, infile = 42, nout = 6
      Character (*), Parameter         :: fname_default = 'e04rdfe.opt'
!     .. Local Scalars ..
      Type (c_ptr)                     :: handle
      Integer                          :: idblk, idx, ifail, ifail_e04rd,      &
```

```
                                       inform, k, maxnblk, maxnnz, maxnvar, &
                                       nblk, nnz, nnzu, nnzua, nnzuc,       &
                                       ntests, nvar
      Character (256)                  :: fname
      Character (60)                   :: title
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: a(:), cvec(:), u(:), ua(:), uc(:),   &
                                       x(:)
      Real (Kind=nag_wp)               :: rinfo(32), stats(32)
      Integer, Allocatable             :: blksizea(:), icola(:), irowa(:),     &
                                       nnza(:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: command_argument_count,              &
                                       get_command_argument, sum, trim
!     .. Executable Statements ..
      Continue

      Write (nout,*) 'E04RDF Example Program Results'
      Write (nout,*)

!     Use the first command line argument as the filename or
!     choose default hard-coded filename in 'fname_default'.
      ntests = command_argument_count()
      If (ntests==0) Then
!       Assume the default filename.
        fname = fname_default
      Else
        Call get_command_argument(1,fname)
      End If

      Write (nout,*) 'Reading SDPA file: ', trim(fname)
      Flush (nout)

!     Open the input file.
      ifail = 0
      Call x04acf(infile,fname,0,ifail)

!     Go through the file and find the dimension of the problem.
!     Unless the file format is wrong, the routine should finish
!     with IFAIL = 1 (not enough space).
      maxnvar = 0
      maxnblk = 0
      maxnnz = 0
      Allocate (cvec(maxnvar),nnza(maxnvar+1),irowa(maxnnz),icola(maxnnz),     &
        a(maxnnz),blksizea(maxnblk))
      ifail_e04rd = -1
      Call e04rdf(infile,maxnvar,maxnblk,maxnnz,filelst,nvar,nblk,nnz,cvec,    &
        nnza,irowa,icola,a,blksizea,ifail_e04rd)
      Deallocate (cvec,nnza,irowa,icola,a,blksizea)

!     Close the file, it will need to be reopened later.
      ifail = 0
      Call x04adf(infile,ifail)

      If (ifail_e04rd/=1) Then
!       Possible problem with formatting, etc.
        Write (nout,99999) 'Reading the SDPA file failed with IFAIL = ', ifail
99999   Format (1X,A,I3)
        Write (nout,*) 'Terminating the example program.'
        Go To 100
      End If

!     Allocate the right size of arrays for the data.
      Write (nout,*) 'Allocating space for the problem.'
      Write (nout,Fmt=99998) 'NVAR = ', nvar
      Write (nout,Fmt=99998) 'NBLK = ', nblk
      Write (nout,Fmt=99998) 'NNZ  = ', nnz
99998 Format (6X,A,I7)
      Flush (nout)

      maxnvar = nvar
```

```
      maxnblk = nblk
      maxnnz = nnz

      Allocate (cvec(maxnvar),nnza(maxnvar+1),irowa(maxnnz),icola(maxnnz),     &
        a(maxnnz),blksizea(maxnblk))

!     Reopen the file.
      ifail = 0
      Call x04acf(infile,fname,0,ifail)

!     Read the problem data, there should be enough space this time.
      ifail = 0
      Call e04rdf(infile,maxnvar,maxnblk,maxnnz,filelst,nvar,nblk,nnz,cvec,    &
        nnza,irowa,icola,a,blksizea,ifail)

!     Close the file.
      ifail = 0
      Call x04adf(infile,ifail)

!     Problem was successfully decoded.
      Write (nout,*)                                                           &
        'Linear SDP problem was read, start formulating the problem'
      Flush (nout)

!     Initialize the handle of the problem.
      ifail = 0
      Call e04raf(handle,nvar,ifail)

!     Add the linear objective function to the formulation.
      ifail = 0
      Call e04ref(handle,nvar,cvec,ifail)

!     Add all linear matrix constraints to the formulation.
      idblk = 0
      ifail = 0
      Call e04rnf(handle,nvar,sum(blksizea(1:nblk)),nnza,nnz,irowa,icola,a,    &
        nblk,blksizea,idblk,ifail)

      Write (nout,*) 'The problem formulation in a handle is completed.'
      Write (nout,*)
      Flush (nout)

!     Print overview of the handle.
      ifail = 0
      Call e04ryf(handle,nout,'Overview',ifail)

!     Set optional arguments.
      ifail = 0
      Call e04zmf(handle,'DIMACS Measures = Check',ifail)
      ifail = 0
      Call e04zmf(handle,'Initial X = Automatic',ifail)

!     Compute memory needed for primal & dual variables.

!     There are no box constraints or linear constraints set
!     by E04RHF or E04RJF, neither second order cone constraints.
      nnzu = 0
      nnzuc = 0

!     Count size of the matrix multipliers, stored as packed
!     triangle respecting the block structure.
      nnzua = 0
      Do k = 1, nblk
        nnzua = nnzua + blksizea(k)*(blksizea(k)+1)/2
      End Do

      Allocate (x(nvar),ua(nnzua),u(nnzu),uc(nnzuc))

!     Call the solver.
      ifail = 0
      Call e04svf(handle,nvar,x,nnzu,u,nnzuc,uc,nnzua,ua,rinfo,stats,inform,   &
```

```
        ifail)

!     Print results.

      Write (nout,*)
      Write (nout,*) 'Optimal solution:'
      Write (nout,99997) x(1:nvar)
99997 Format (1X,'X = ',2F9.2)
      Flush (nout)

!     Print packed lower triangles of the Lagrangian multipliers.
      idx = 1
      Do k = 1, nblk
        Write (title,99996) 'Lagrangian multiplier for A_', k
99996   Format (A,I0)
        nnz = blksizea(k)*(blksizea(k)+1)/2
        ifail = 0
        Call x04ccf('Lower','N',blksizea(k),ua(idx:idx+nnz-1),title,ifail)
        idx = idx + nnz
      End Do

!     Deallocate memory within the handle.
      ifail = 0
      Call e04rzf(handle,ifail)

100   Continue
    End Program e04rdfe
```

10.2 Program Data

```
" E04RDF 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

```
 E04RDF Example Program Results

 Reading SDPA file: e04rdfe.opt
 ** At least one of MAXNVAR, MAXNBLK or MAXNNZ is too small.
 ** MAXNVAR should be at least                    2, was                    0.
 ** MAXNBLK should be at least                    3, was                    0.
 ** MAXNNZ should be at least                   10, was                   0.
 ** ABNORMAL EXIT from NAG Library routine E04RDF: IFAIL =     1
 ** NAG soft failure - control returned
 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 =      1.00     1.00
 Lagrangian multiplier for A_1
             1
 1     10.0000
 Lagrangian multiplier for A_2
               1
 1    2.4321E-06
 Lagrangian multiplier for A_3
             1          2
 1      2.8571
 2     -2.8571     2.8571
```