NAG Library Function Document

nag_opt_handle_set_quadmatineq (e04rpc)

1 Purpose

nag_opt_handle_set_quadmatineq (e04rpc) is a part of the NAG optimization modelling suite and defines bilinear matrix terms either in a new matrix constraint or adds them to an existing linear matrix inequality.

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_handle_set_quadmatineq (void *handle, Integer nq,
     const Integer qi[], const Integer qj[], Integer dimq,
     const Integer nnzq[], Integer nnzqsum, const Integer irowq[],
     const Integer icolq[], const double q[], Integer *idblk, NagError *fail)
```

3 Description

After the initialization function nag_opt_handle_init (e04rac) has been called, nag_opt_handle_set_quadmatineq (e04rpc) may be used to define bilinear matrix terms. It may be used in two ways, either to add to the problem formulation a new bilinear matrix inequality (BMI) which does not have linear terms:

$$\sum_{i,j=1}^{n} x_i x_j Q_{ij} \succeq 0 \tag{1}$$

or to extend an existing linear matrix inequality constraint by bilinear terms:

$$\sum_{i,j=1}^{n} x_i x_j Q_{ij}^k. \tag{2}$$

Here Q_{ij}^k are d by d (sparse) symmetric matrices and k, if present, is the number of the existing constraint. This function will typically be used on semidefinite programming problems with bilinear matrix constraints (BMI-SDP)

$$\begin{aligned}
\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \tfrac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x && \text{(a)}\\
\text{subject to} \quad & \sum_{i,j=1}^{n} x_i x_j Q_{ij}^k + \sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1,\ldots,m_A && \text{(b)}\\
& l_B \le Bx \le u_B && \text{(c)}\\
& l_x \le x \le u_x. && \text{(d)}
\end{aligned} \tag{3}$$

The function can be called multiple times to define an additional matrix inequality or to extend an existing one, but it cannot be called twice to extend the same matrix inequality. The argument **idblk** is used to distinguish whether a new matrix constraint should be added (**idblk** $= 0$) or if an existing linear matrix inequality should be extended (**idblk** > 0). In the latter case, **idblk** should be set to k, the number of the existing inequality. See nag_opt_handle_set_linmatineq (e04rnc) for details about formulation of linear matrix constraints and their numbering and a further description of **idblk**. For a generic description of the problem see nag_opt_handle_init (e04rac). In the further text, the index k will be omitted.

3.1 Input data organization

It is expected that only some of the matrices Q_{ij} will be nonzero therefore only their index pairs i, j are listed in arrays **qi** and **qj**. Note that a pair i, j should not repeat, i.e., a matrix Q_{ij} should not be defined more than once. No particular ordering of pairs i, j is expected but other input arrays **irowq**, **icolq**, **q** and **nnzq** need to respect the chosen order.

Note: the dimension of Q_{ij} must respect the size of the linear matrix inequality if they are supposed to expand it (case **idblk** > 0).

Matrices Q_{ij} are symmetric and thus only their upper triangles are passed to the function. They are stored in sparse coordinate storage format (see Section 2.1.1 in the f11 Chapter Introduction), i.e., every nonzero from the upper triangles is coded as a triplet of row index, column index and the numeric value. All these triplets from all Q_{ij} matrices are passed to the function in three arrays: **irowq** for row indices, **icolq** for column indices and **q** for the values. No particular order of nonzeros within one matrix is enforced but all nonzeros belonging to one Q_{ij} matrix need to be stored next to each other. The first **nnzq**$[0]$ nonzeros belong to $Q_{i_1 j_1}$ where $i_1 =$ **qi**$[0]$, $j_1 =$ **qj**$[0]$, the following **nnzq**$[1]$ nonzeros to the next one given by **qi**, **qj** and so on. The array **nnzq** thus splits arrays **irowq**, **icolq** and **q** into sections so that each section defines one Q_{ij} matrix. See Table 1 below. Functions nag_opt_sdp_read_sdpa (e04rdc) and nag_opt_handle_set_linmatineq (e04rnc) use the same data organization so further examples can be found there.

irowq **icolq** **q**	upper triangle nonzeros from $\underbrace{Q_{i_1 j_1}}$ **nnzq**$[0]$ $i_1 =$ **qi**$[0]$ $j_1 =$ **qj**$[0]$	upper triangle nonzeros from $\underbrace{Q_{i_2 j_2}}$ **nnzq**$[1]$ $i_2 =$ **qi**$[1]$ $j_2 =$ **qj**$[1]$	$\cdots$	upper triangle nonzeros from $\underbrace{Q_{i_{\mathbf{nq}} j_{\mathbf{nq}}}}$ **nnzq**$[\mathbf{nq}-1]$ $i_{\mathbf{nq}} =$ **qi**$[\mathbf{nq}-1]$ $j_{\mathbf{nq}} =$ **qj**$[\mathbf{nq}-1]$

Table 1
Coordinate storage format of Q_{ij} matrices in input arrays

4 References

Syrmos V L, Abdallah C T, Dorato P and Grigoriadis K (1997) Static output feedback – a survey *Journal Automatica (Journal of IFAC) (Volume 33)* **2** 125–137

5 Arguments

1: **handle** – void * *Input*

On entry: the handle to the problem. It needs to be initialized by nag_opt_handle_init (e04rac) and **must not** be changed.

2: **nq** – Integer *Input*

On entry: the number of index pairs i, j of the nonzero matrices Q_{ij}.

Constraint: **nq** > 0.

3: **qi**[**nq**] – const Integer *Input*
4: **qj**[**nq**] – const Integer *Input*

On entry: the index pairs i, j of the nonzero matrices Q_{ij} in any order.

Constraint: $1 \leq i, j \leq n$ where n is the number of decision variables in the problem set during the initialization of the handle by nag_opt_handle_init (e04rac). The pairs do not repeat.

5: **dimq** – Integer *Input*

On entry: d, the dimension of matrices Q_{ij}.

Constraints:

dimq > 0;
if **idblk** > 0, **dimq** needs to be identical to the dimension of matrices of the constraint k.

6: **nnzq**[**nq**] – const Integer *Input*

On entry: the numbers of nonzero elements in the upper triangles of Q_{ij} matrices.

Constraint: **nnzq**$[i-1] > 0$.

7: **nnzqsum** – Integer *Input*

On entry: the dimension of the arrays **irowq**, **icolq** and **q**, at least the total number of all nonzeros in all Q_{ij} matrices.

Constraints:

nnzqsum > 0;
nnzqsum $\geq \sum_{k=1}^{\mathbf{nq}} \mathbf{nnzq}[k-1]$.

8: **irowq**[**nnzqsum**] – const Integer *Input*
9: **icolq**[**nnzqsum**] – const Integer *Input*
10: **q**[**nnzqsum**] – const double *Input*

On entry: the nonzero elements of the upper triangles of matrices Q_{ij} stored in coordinate storage format. The first **nnzq**$[0]$ elements belong to the first $Q_{i_1 j_1}$, the following **nnzq**$[1]$ to $Q_{i_2 j_2}$, etc.

Constraint: $1 \leq$ **irowq**$[i-1] \leq$ **dimq**, **irowq**$[i-1] \leq$ **icolq**$[i-1] \leq$ **dimq**.

11: **idblk** – Integer * *Input/Output*

On entry: if **idblk** $= 0$, a new matrix constraint is created; otherwise **idblk** $= k > 0$, the number of the existing linear matrix constraint to be expanded with the bilinear terms.

Constraint: **idblk** ≥ 0.

On exit: if **idblk** $= 0$ on entry, the number of the new matrix constraint is returned. By definition, it is the number of the matrix inequalities already defined plus one. Otherwise, **idblk** > 0 stays unchanged.

12: **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_ALREADY_DEFINED

On entry, **idblk** $= \langle value \rangle$.
Bilinear terms of the matrix inequality block with the given **idblk** have already been defined.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_DIM_MATCH

On entry, **dimq** $= \langle value \rangle$, **idblk** $= \langle value \rangle$.
The correct dimension of the given **idblk** is $\langle value \rangle$.
Constraint: **dimq** must match the dimension of the block supplied earlier.

NE_HANDLE

The supplied **handle** does not define a valid handle to the data structure for the NAG optimization modelling suite. It has not been initialized by nag_opt_handle_init (e04rac) or it has been corrupted.

NE_INDICES

On entry, index pair with **qi** $= \langle value \rangle$ and **qj** $= \langle value \rangle$ repeats.
Constraint: each index pair **qi**, **qj** must be unique.

On entry, $k = \langle value \rangle$, **qi**$[k-1] = \langle value \rangle$ and $n = \langle value \rangle$.
Constraint: $1 \leq$ **qi**$[k-1] \leq n$.

On entry, $k = \langle value \rangle$, **qj**$[k-1] = \langle value \rangle$ and $n = \langle value \rangle$.
Constraint: $1 \leq$ **qj**$[k-1] \leq n$.

NE_INT

On entry, **dimq** $= \langle value \rangle$.
Constraint: **dimq** > 0.

On entry, **idblk** $= \langle value \rangle$.
Constraint: **idblk** ≥ 0.

On entry, **nq** $= \langle value \rangle$.
Constraint: **nq** > 0.

NE_INT_2

On entry, **nnzqsum** $= \langle value \rangle$ and sum (**nnzq**) $= \langle value \rangle$.
Constraint: **nnzqsum** $\geq$ sum (**nnzq**).

NE_INT_ARRAY_1

On entry, $i = \langle value \rangle$ and **nnzq**$[i-1] = \langle value \rangle$.
Constraint: **nnzq**$[i-1] > 0$.

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

On entry, an error occurred in matrix Q_{ij} of index $k = \langle value \rangle$, **qi**$[k-1] = \langle value \rangle$, **qj**$[k-1] = \langle value \rangle$.
For $j = \langle value \rangle$, **icolq**$[j-1] = \langle value \rangle$ and **dimq** $= \langle value \rangle$.
Constraint: $1 \leq$ **icolq**$[j-1] \leq$ **dimq**.

On entry, an error occurred in matrix Q_{ij} of index $k = \langle value \rangle$, **qi**$[k-1] = \langle value \rangle$, **qj**$[k-1] = \langle value \rangle$.
For $j = \langle value \rangle$, **irowq**$[j-1] = \langle value \rangle$ and **dimq** $= \langle value \rangle$.
Constraint: $1 \leq$ **irowq**$[j-1] \leq$ **dimq**.

On entry, an error occurred in matrix Q_{ij} of index $k = \langle value \rangle$, **qi**$[k-1] = \langle value \rangle$, **qj**$[k-1] = \langle value \rangle$.
For $j = \langle value \rangle$, **irowq**$[j-1] = \langle value \rangle$ and **icolq**$[j-1] = \langle value \rangle$.
Constraint: **irowq**$[j-1] \leq$ **icolq**$[j-1]$ (elements within the upper triangle).

On entry, an error occurred in matrix Q_{ij} of index $k = \langle value \rangle$, **qi**$[k-1] = \langle value \rangle$, **qj**$[k-1] = \langle value \rangle$.
More than one element of Q_{ij} has row index $\langle value \rangle$ and column index $\langle value \rangle$.
Constraint: each element of Q_{ij} must have a unique row and column index.

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_PHASE

The problem cannot be modified in this phase any more, the solver has already been called.

NE_REF_MATCH

On entry, **idblk** $= \langle value \rangle$.
The given **idblk** does not match with any existing matrix inequality block.
The maximum **idblk** is currently $\langle value \rangle$.

On entry, **idblk** $= \langle value \rangle$.
The given **idblk** refers to a nonexistent matrix inequality block.
No matrix inequalities have been added yet.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_opt_handle_set_quadmatineq (e04rpc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example demonstrates how semidefinite programming can be used in control theory. See also Section 10 in nag_opt_handle_init (e04rac) for links to further examples in the suite.

The problem, from static output feedback (SOF) control Syrmos *et al.* (1997), solved here is the linear time-invariant (LTI) 'test' system

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \tag{4}$$

subject to static output feedback

$$u = Ky. \tag{5}$$

Here $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$ and $C \in \mathbb{R}^{p \times n}$ are given matrices, $x \in \mathbb{R}^n$ is the vector of state variables, $u \in \mathbb{R}^m$ is the vector of control inputs, $y \in \mathbb{R}^p$ is the vector of system outputs, and $K \in \mathbb{R}^{m \times p}$ is the unknown feedback gain matrix.

The problem is to find K such that (4) is time-stable when subject to (5), i.e., all eigenvalues of the closed-loop system matrix $A + BKC$ belong to the left half-plane. From the Lyapunov stability theory, this holds if and only if there exists a symmetric positive definite matrix P such that

$$(A + BKC)^{\mathrm{T}}P + P(A + BKC) \prec 0.$$

Hence, by introducing the new variable, the Lyapunov matrix P, we can formulate the SOF problem as a feasibility BMI-SDP problem in variables K and P. As we cannot formulate the problem with sharp matrix inequalities, we can solve the following system instead (note that the objective function is added to bound matrix P):

$$\begin{array}{ll} \underset{K,P}{\text{minimize}} & \text{trace}(P) \\ \text{subject to} & (A + BKC)^{\mathrm{T}}P + P(A + BKC) \preceq -I \\ & P \succeq I. \end{array} \tag{6}$$

For $n = p = 2$, $m = 1$,

$$A = \begin{pmatrix} -1 & 2 \\ -3 & -4 \end{pmatrix}, \quad B = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \quad C = I$$

and the unknown matrices expressed as

$$P = \begin{pmatrix} x_1 & x_2 \\ x_2 & x_3 \end{pmatrix}, \quad K = \begin{pmatrix} x_4 & x_5 \end{pmatrix},$$

the problem (6) can be rewritten in the form (3) as follows:

$$\begin{array}{ll} \underset{x \in \mathbb{R}^5}{\text{minimize}} & x_1 + x_3 \\ \text{subject to} & \begin{pmatrix} 2x_1x_4 + 2x_2x_4 & x_1x_5 + x_2x_4 + x_2x_5 + x_3x_4 \\ \text{sym.} & 2x_2x_5 + 2x_3x_5 \end{pmatrix} + \\ & \begin{pmatrix} 2x_1 + 6x_2 & -2x_1 + 5x_2 + 3x_3 \\ \text{sym.} & -4x_2 + 8x_3 \end{pmatrix} - I \succeq 0 \\ & \begin{pmatrix} x_1 & x_2 \\ \text{sym.} & x_3 \end{pmatrix} - I \succeq 0. \end{array}$$

This formulation has been stored in a generic BMI-SDP data file which is processed and solved by the example program.

10.1 Program Text

```
/* nag_opt_handle_set_quadmatineq (e04rpc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

/* Read a 'generic' LMI/BMI SDP problem from the input file,
 * formulate the problem via a handle and pass it to the solver.
 */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>

int main(void)
{
  Integer exit_status = 0;
  Integer blkidx, dimaq, idblk, idlc, idx, idxend, inform, j, midx, nblk,
         nclin, nnzasum, nnzb, nnzc, nnzh, nnzqsum, nq, nvar;
  double *a = 0, *b = 0, *bl = 0, *bu = 0, *cvec = 0, *h = 0, *q = 0, *x = 0;
  double rinfo[32], stats[32];
  Integer *icola = 0, *icolb = 0, *icolh = 0, *icolq = 0, *idxc = 0,
         *irowa = 0, *irowb = 0, *irowh = 0, *irowq = 0, *nnza = 0,
```

```
        *nnzq = 0, *qi = 0, *qj = 0;
  void *handle = 0;
  /* Nag Types */
  NagError fail;

  printf("nag_opt_handle_set_quadmatineq (e04rpc) Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in the data file. */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Read the problem size. */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n]", &nvar);
#else
  scanf("%" NAG_IFMT "%*[^\n]", &nvar);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n]", &nnzh);
#else
  scanf("%" NAG_IFMT "%*[^\n]", &nnzh);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT " %" NAG_IFMT "%*[^\n]", &nclin, &nnzb);
#else
  scanf("%" NAG_IFMT " %" NAG_IFMT "%*[^\n]", &nclin, &nnzb);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n]", &nblk);
#else
  scanf("%" NAG_IFMT "%*[^\n]", &nblk);
#endif

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

  /* Read the linear part of the objective function. */
  if (!(cvec = NAG_ALLOC(nvar, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (j = 0; j < nvar; j++)
#ifdef _WIN32
    scanf_s("%lf", &cvec[j]);
#else
    scanf("%lf", &cvec[j]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  if (nnzh == 0) {
    /* nag_opt_handle_set_linobj (e04rec).
     * Add the linear objective function to the formulation. */
    nag_opt_handle_set_linobj(handle, nvar, cvec, NAGERR_DEFAULT);
    NAG_FREE(cvec);
  }
  else {
    /* The objective function is quadratic. Read nonzeros for H. */
    if (!(irowh = NAG_ALLOC(nnzh, Integer)) ||
        !(icolh = NAG_ALLOC(nnzh, Integer)) ||
        !(h = NAG_ALLOC(nnzh, double)) || !(idxc = NAG_ALLOC(nvar, Integer)))
```

```
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

    for (idx = 0; idx < nnzh; idx++)
#ifdef _WIN32
      scanf_s("%lf %" NAG_IFMT " %" NAG_IFMT "%*[^\n]",
              &h[idx], &irowh[idx], &icolh[idx]);
#else
      scanf("%lf %" NAG_IFMT " %" NAG_IFMT "%*[^\n]",
            &h[idx], &irowh[idx], &icolh[idx]);
#endif

    /* The linear part of the objective was read in as dense,
     * but the sparse format is needed here. */
    for (idx = 0; idx < nvar; idx++)
      idxc[idx] = idx + 1;
    nnzc = nvar;

    /* nag_opt_handle_set_quadobj (e04rfc).
     * Add the quadratic objective to the handle.*/
    nag_opt_handle_set_quadobj(handle, nnzc, idxc, cvec, nnzh, icolh, irowh,
                               h, NAGERR_DEFAULT);
    NAG_FREE(idxc);
    NAG_FREE(cvec);
    NAG_FREE(irowh);
    NAG_FREE(icolh);
    NAG_FREE(h);
  }

  /* Read a block of linear constraints and its bounds if present. */
  if (nclin > 0 && nnzb > 0) {
    if (!(irowb = NAG_ALLOC(nnzb, Integer)) ||
        !(icolb = NAG_ALLOC(nnzb, Integer)) ||
        !(b = NAG_ALLOC(nnzb, double)) ||
        !(bl = NAG_ALLOC(nclin, double)) || !(bu = NAG_ALLOC(nclin, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

    /* Read all matrix B nonzeros. */
    for (idx = 0; idx < nnzb; idx++)
#ifdef _WIN32
      scanf_s("%lf %" NAG_IFMT " %" NAG_IFMT "%*[^\n]",
              &b[idx], &irowb[idx], &icolb[idx]);
#else
      scanf("%lf %" NAG_IFMT " %" NAG_IFMT "%*[^\n]",
            &b[idx], &irowb[idx], &icolb[idx]);
#endif

    /* Read the lower and upper bounds. */
    for (j = 0; j < nclin; j++)
#ifdef _WIN32
      scanf_s("%lf", &bl[j]);
#else
      scanf("%lf", &bl[j]);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif
    for (j = 0; j < nclin; j++)
#ifdef _WIN32
      scanf_s("%lf", &bu[j]);
#else
      scanf("%lf", &bu[j]);
```

```
#endif
#ifdef _WIN32
    scanf_s("%*[^\n]");
#else
    scanf("%*[^\n]");
#endif
    idlc = 0;

    /* nag_opt_handle_set_linconstr (e04rjc).
     * Add the block of linear constraints to the problem formulation. */
    nag_opt_handle_set_linconstr(handle, nclin, bl, bu, nnzb, irowb, icolb,
                                 b, &idlc, NAGERR_DEFAULT);
    NAG_FREE(irowb);
    NAG_FREE(icolb);
    NAG_FREE(b);
    NAG_FREE(bl);
    NAG_FREE(bu);
  }

  /* Read all matrix inequalities. */
  for (blkidx = 0; blkidx < nblk; blkidx++) {
    /* Read the dimension of this matrix inequality. */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[^\n]", &dimaq);
#else
    scanf("%" NAG_IFMT "%*[^\n]", &dimaq);
#endif
    /* Read the number of all nonzeros in linear and bilinear parts. */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT " %" NAG_IFMT "%*[^\n]", &nnzasum, &nnzqsum);
#else
    scanf("%" NAG_IFMT " %" NAG_IFMT "%*[^\n]", &nnzasum, &nnzqsum);
#endif
    idblk = 0;

    if (nnzasum > 0) {
      /* Read a linear matrix inequality composed of (nvar+1) matrices. */
      if (!(nnza = NAG_ALLOC(nvar + 1, Integer)) ||
          !(irowa = NAG_ALLOC(nnzasum, Integer)) ||
          !(icola = NAG_ALLOC(nnzasum, Integer)) ||
          !(a = NAG_ALLOC(nnzasum, double)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }

      idx = 0;
      for (midx = 0; midx <= nvar; midx++) {
        /* Read matrix A_midx. */
#ifdef _WIN32
        scanf_s("%" NAG_IFMT "%*[^\n]", &nnza[midx]);
#else
        scanf("%" NAG_IFMT "%*[^\n]", &nnza[midx]);
#endif
        idxend = idx + nnza[midx];
        for ( ; idx < idxend; idx++)
#ifdef _WIN32
          scanf_s("%lf %" NAG_IFMT " %" NAG_IFMT "%*[^\n]",
                  &a[idx], &irowa[idx], &icola[idx]);
#else
          scanf("%lf %" NAG_IFMT " %" NAG_IFMT "%*[^\n]",
                &a[idx], &irowa[idx], &icola[idx]);
#endif
      }
      idblk = 0;

      /* nag_opt_handle_set_linmatineq (e04rnc).
       * Add the linear matrix constraint to the problem formulation. */
      nag_opt_handle_set_linmatineq(handle, nvar, dimaq, nnza, nnzasum,
                                    irowa, icola, a, 1, NULL, &idblk,
```

```
                                        NAGERR_DEFAULT);

      NAG_FREE(nnza);
      NAG_FREE(irowa);
      NAG_FREE(icola);
      NAG_FREE(a);
    }

    if (nnzqsum > 0) {
      /* Read bilinear part of the matrix inequality composed
       * of nq matrices. */
#ifdef _WIN32
      scanf_s("%" NAG_IFMT "%*[^\n]", &nq);
#else
      scanf("%" NAG_IFMT "%*[^\n]", &nq);
#endif
      if (!(qi = NAG_ALLOC(nq, Integer)) ||
          !(qj = NAG_ALLOC(nq, Integer)) ||
          !(nnzq = NAG_ALLOC(nq, Integer)) ||
          !(irowq = NAG_ALLOC(nnzqsum, Integer)) ||
          !(icolq = NAG_ALLOC(nnzqsum, Integer)) ||
          !(q = NAG_ALLOC(nnzqsum, double)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }

      idx = 0;
      for (midx = 0; midx < nq; midx++) {
        /* Read matrix Q_ij where i=qi[midx], j=qj[midx]. */
#ifdef _WIN32
        scanf_s("%" NAG_IFMT " %" NAG_IFMT "%*[^\n]", &qi[midx], &qj[midx]);
#else
        scanf("%" NAG_IFMT " %" NAG_IFMT "%*[^\n]", &qi[midx], &qj[midx]);
#endif
#ifdef _WIN32
        scanf_s("%" NAG_IFMT "%*[^\n]", &nnzq[midx]);
#else
        scanf("%" NAG_IFMT "%*[^\n]", &nnzq[midx]);
#endif
        idxend = idx + nnzq[midx];
        for ( ; idx < idxend; idx++)
#ifdef _WIN32
          scanf_s("%lf %" NAG_IFMT " %" NAG_IFMT "%*[^\n]",
                  &q[idx], &irowq[idx], &icolq[idx]);
#else
          scanf("%lf %" NAG_IFMT " %" NAG_IFMT "%*[^\n]",
                &q[idx], &irowq[idx], &icolq[idx]);
#endif
      }

      /* nag_opt_handle_set_quadmatineq (e04rpc).
       * Expand the existing linear matrix inequality with the bilinear
       * terms or (if the linear part was not present) create a
       * new matrix inequality. */
      nag_opt_handle_set_quadmatineq(handle, nq, qi, qj, dimaq, nnzq,
                                     nnzqsum, irowq, icolq, q, &idblk,
                                     NAGERR_DEFAULT);

      NAG_FREE(qi);
      NAG_FREE(qj);
      NAG_FREE(nnzq);
      NAG_FREE(irowq);
      NAG_FREE(icolq);
      NAG_FREE(q);
    }
  }

  /* Problem was successfully decoded. */
  printf("SDP problem was read, passing it to the solver.\n\n");
```

```
  fflush(stdout);

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

  /* Allocate memory for the solver. */
  if (!(x = NAG_ALLOC(nvar, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (j = 0; j < nvar; j++)
    x[j] = 0.0;

  /* Call the solver nag_opt_handle_solve_pennon (e04svc),
   * ignore Lagrangian multipliers. */
  INIT_FAIL(fail);
  nag_opt_handle_solve_pennon(handle, nvar, x, 0, NULL, 0, NULL, 0, NULL,
                              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 = 1;
    goto END;
  }

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(x);

  return exit_status;
}
```

10.2 Program Data

```
nag_opt_handle_set_quadmatineq (e04rpc) Example Program Data
     5                 : no of variables
     0                 : no of nonzeros in H matrix
     0         0       : no of linear constraints and nnz in B
     2                 : no of matrix constraints

   1.000000    0.000000    1.000000    0.000000
   0.000000                                          : Linear obj. vector

   2                 : beginning of matrix constr 1, its dimension
   9    8            : no of nonzeroes in all A_i, Q_ij

      2                              : number of nonzeros in A_0
      1.000000       1       1       : Upper triangle of A_0
      1.000000       2       2       : End of matrix A_0

      2                              : number of nonzeros in A_1
      2.000000       1       1       : Upper triangle of A_1
     -2.000000       1       2       : End of matrix A_1

      3                              : number of nonzeros in A_2
      6.000000       1       1       : Upper triangle of A_2
      5.000000       1       2       : in coordinate storage
     -4.000000       2       2       : End of matrix A_2

      2                              : number of nonzeros in A_3
      3.000000       1       2       : Upper triangle of A_3
```

```
      8.000000       2       2     : End of matrix A_3

      0                            : number of nonzeros in A_4
      0                            : number of nonzeros in A_5
   6                : number of Q_ij matrices

      1       4                    : indices giving i & j for Q_ij
      1                            : number of nonzeros in Q_{1,4}
      2.000000       1       1     : End of matrix Q_{1,4}

      2       4                    : indices giving i & j for Q_ij
      2                            : number of nonzeros in Q_{2,4}
      2.000000       1       1     : Upper triangle of Q_{2,4}
      1.000000       1       2     : End of matrix Q_{2,4}

      3       4                    : indices giving i & j for Q_ij
      1                            : number of nonzeros in Q_{3,4}
      1.000000       1       2     : End of matrix Q_{3,4}

      1       5                    : indices giving i & j for Q_ij
      1                            : number of nonzeros in Q_{1,5}
      1.000000       1       2     : End of matrix Q_{1,5}

      2       5                    : indices giving i & j for Q_ij
      2                            : number of nonzeros in Q_{2,5}
      1.000000       1       2     : Upper triangle of Q_{2,5}
      2.000000       2       2     : End of matrix Q_{2,5}

      3       5                    : indices giving i & j for Q_ij
      1                            : number of nonzeros in Q_{3,5}
      2.000000       2       2     : End of matrix Q_{3,5}

   2                : beginning of matrix constr 2, its dimension
   5    0           : no of nonzeroes in all A_i, Q_ij

      2                            : number of nonzeros in A_0
      1.000000       1       1     : Upper triangle of A_0
      1.000000       2       2     : End of matrix A_0

      1                            : number of nonzeros in A_1
      1.000000       1       1     : End of matrix A_1

      1                            : number of nonzeros in A_2
      1.000000       1       2     : End of matrix A_2

      1                            : number of nonzeros in A_3
      1.000000       2       2     : End of matrix A_3

      0                            : number of nonzeros in A_4
      0                            : number of nonzeros in A_5
```

10.3 Program Results

```
nag_opt_handle_set_quadmatineq (e04rpc) Example Program Results

SDP problem was read, passing it to the solver.

 Overview
   Status:                  Problem and option settings are editable.
   No of variables:         5
   Objective function:      linear
   Simple bounds:           not defined yet
   Linear constraints:      not defined yet
   Nonlinear constraints:   not defined yet
   Matrix constraints:      2
 Matrix constraints
   IDblk =     1, size =      2 x     2, polynomial of order 2
   IDblk =     2, size =      2 x     2, linear
 E04SV, NLP-SDP Solver (Pennon)
 ------------------------------
```

```
 Number of variables             5                 [eliminated            0]
                          simple  linear  nonlin
 (Standard) inequalities       0       0       0
 (Standard) equalities                 0       0
 Matrix inequalities                   1       1 [dense    2, 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                     =               User      * d
     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               =                  No     * S
     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               =           Goldstein     * 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  1.82E+01  1.00E+00  4.00E+00  2.00E+00   0
   1  4.11823E+00  3.85E-03  0.00E+00  1.73E+00  2.00E+00   6
   2  2.58252E+00  5.36E-03  0.00E+00  4.93E-01  9.04E-01   4
   3  2.06132E+00  1.02E-03  0.00E+00  7.70E-02  4.08E-01   4
   4  2.00050E+00  3.00E-03  8.91E-03  1.78E-02  1.85E-01   3
   5  1.99929E+00  1.55E-03  3.16E-03  3.65E-03  8.34E-02   2
   6  1.99985E+00  1.03E-04  3.16E-04  7.19E-04  3.77E-02   4
   7  1.99997E+00  7.04E-04  5.76E-05  1.41E-04  1.70E-02   1
   8  2.00000E+00  1.32E-04  6.52E-06  2.76E-05  7.70E-03   1
   9  2.00000E+00  8.49E-06  7.86E-07  5.37E-06  3.48E-03   1
  10  2.00000E+00  5.88E-07  1.06E-07  1.04E-06  1.57E-03   1
  11  2.00000E+00  5.55E-08  4.87E-08  2.02E-07  7.11E-04   1
  12  2.00000E+00  5.34E-09  5.37E-09  3.93E-08  3.21E-04   1
  13  2.00000E+00  5.03E-10  5.45E-09  7.62E-09  1.45E-04   1
  14  2.00000E+00  4.45E-11  5.55E-09  1.48E-09  6.56E-05   1
 ------------------------------------------------------------
  it|  objective |  optim  |   feas  |  compl  | pen min |inner
 ------------------------------------------------------------
  15  2.00000E+00  4.36E-12  5.67E-09  2.87E-10  2.96E-05   1
  16  2.00000E+00  1.61E-11  5.82E-09  5.57E-11  1.34E-05   1
  17  2.00000E+00  3.13E-11  6.00E-09  1.08E-11  6.06E-06   1
  18  2.00000E+00  8.65E-11  6.22E-09  2.10E-12  2.74E-06   1
  19  2.00000E+00  1.31E-10  6.48E-09  4.07E-13  1.24E-06   1
 ------------------------------------------------------------
 Status: converged, an optimal solution found
 ------------------------------------------------------------
```

```
 Final objective value               2.000000E+00
 Relative precision                  8.141636E-16
 Optimality                          1.310533E-10
 Feasibility                         6.484489E-09
 Complementarity                     4.066867E-13
 Iteration counts
   Outer iterations                            19
   Inner iterations                            36
   Linesearch steps                            56
 Evaluation counts
   Augm. Lagr. values                          76
   Augm. Lagr. gradient                        56
   Augm. Lagr. hessian                         36
 ---------------------------------------------------------------
```