NAG Library Function Document

nag_opt_handle_set_linmatineq (e04rnc)

1 Purpose

nag_opt_handle_set_linmatineq (e04rnc) is a part of the NAG optimization modelling suite and defines one or more linear matrix constraints of the problem.

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_handle_set_linmatineq (void *handle, Integer nvar,
     Integer dima, const Integer nnza[], Integer nnzasum,
     const Integer irowa[], const Integer icola[], const double a[],
     Integer nblk, const Integer blksizea[], Integer *idblk, NagError *fail)
```

3 Description

After the initialization function nag_opt_handle_init (e04rac) has been called, nag_opt_handle_set_lin matineq (e04rnc) may be used to add one or more linear matrix inequalities

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

to the problem definition. Here A_i are d by d symmetric matrices. The expression $S \succeq 0$ stands for a constraint on eigenvalues of a symmetric matrix S, namely, all the eigenvalues should be non-negative, i.e., the matrix S should be positive semidefinite.

Typically, this will be used in linear semidefinite programming problems (SDP)

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} & \displaystyle\sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1,\ldots,m_A & \text{(b)} \\ & l_B \le Bx \le u_B & \text{(c)} \\ & l_x \le x \le u_x & \text{(d)} \end{array} \tag{2}$$

or to define the linear part of bilinear matrix inequalities (3)(b) in (BMI-SDP)

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

nag_opt_handle_set_linmatineq (e04rnc) can be called repeatedly to accumulate more matrix inequalities. See nag_opt_handle_init (e04rac) for more details.

3.1 Input data organization

All the matrices A_i, for $i = 0, 1, \ldots, n$, are symmetric and thus only their upper triangles are passed to the routine. 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. These triplets of all (upper triangle) nonzeros from all A_i matrices are passed to the function in three arrays: **irowa** for row indices, **icola** for column indices and **a** for the

values. No particular order of nonzeros within one matrix is enforced but all nonzeros from A_0 must be stored first, followed by all nonzero from A_1, followed by A_2, etc.

The number of stored nonzeros from each A_i matrix is given in **nnza**$[i]$, thus this array indicates which section of arrays **irowa**, **icola** and **a** belongs to which A_i matrix. See Table 1 and the example in Section 9. See also nag_opt_sdp_read_sdpa (e04rdc) which uses the same data organization.

irowa **icola** **a**	upper triangle nonzeros from A_0 **nnza**$[0]$	upper triangle nonzeros from A_1 **nnza**$[1]$	$\cdots$	upper triangle nonzeros from A_n **nnza**$[n]$

Table 1
Coordinate storage format of matrices $A_0, A_1, \ldots, A_n$ in input arrays

There are two possibilities for defining more matrix inequality constraints

$$\sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1, 2, \ldots, m_A \tag{4}$$

to the problem. The first is to call nag_opt_handle_set_linmatineq (e04rnc) m_A times and define a single matrix inequality at a time. This might be more straightforward and therefore it is recommended. Alternatively, it is possible to merge all m_A constraints into one inequality and pass them in a single call to nag_opt_handle_set_linmatineq (e04rnc). It is easy to see that (4) can be equivalently expressed as one bigger matrix inequality with the following block diagonal structure

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

If d_k denotes the dimension of inequality k, the new merged inequality has dimension $d = \sum_{k=1}^{m_A} d_k$ and each of the A_i matrices is formed by $A_i^1, A_i^2, \ldots, A_i^{m_A}$ stored as m_A diagonal blocks. In such a case, **nblk** is set to m_A and **blksizea**$[k-1]$ to d_k, the size of the kth diagonal blocks. This might be useful in connection with nag_opt_sdp_read_sdpa (e04rdc).

On the other hand, if there is no block structure and just one matrix inequality is provided, **nblk** should be set to 1 and **blksizea** is not referenced.

3.2 Definition of Bilinear Matrix Inequalities (BMI)

nag_opt_handle_set_linmatineq (e04rnc) is designed to be used together with nag_opt_handle_set_quadmatineq (e04rpc) to define bilinear matrix inequalities (3)(b). nag_opt_handle_set_linmatineq (e04rnc) sets the linear part of the constraint and nag_opt_handle_set_quadmatineq (e04rpc) expands it by higher order terms. To distinquish which linear matrix inequality (or more precisely, which block) is to be expanded, nag_opt_handle_set_quadmatineq (e04rpc) needs the number of the block, **idblk**. The blocks are numbered as they are added, starting from 1.

Whenever a matrix inequality (or a set of them expressed as diagonal blocks) is stored, the function returns **idblk** of the last inequality added. **idblk** is just the order of the inequality amongst all matrix inequalities accumulated through the calls. The first inequality has **idblk** $= 1$, the second one **idblk** $= 2$, etc. Therefore if you call nag_opt_handle_set_linmatineq (e04rnc) for the very first time with **nblk** $= 42$, it adds 42 inequalities with **idblk** from 1 to 42 and the function returns **idblk** $= 42$ (the number of the last one). A subsequent call with **nblk** $= 1$ would add only one inequality, this time with **idblk** $= 43$ which would be returned.

4 References

None.

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: **nvar** – Integer *Input*

On entry: n, the number of decision variables x in the problem. It must be unchanged from the value set during the initialization of the handle by nag_opt_handle_init (e04rac).

3: **dima** – Integer *Input*

On entry: d, the dimension of the matrices A_i, for $i = 0, 1, \ldots,$ **nvar**.

Constraint: **dima** > 0.

4: **nnza**[**nvar** + **1**] – const Integer *Input*

On entry: **nnza**$[i]$, for $i = 0, 1, \ldots,$ **nvar**, gives the number of nonzero elements in the upper triangle of matrix A_i. To define A_i as a zero matrix, set **nnza**$[i] = 0$. However, there must be at least one matrix with at least one nonzero.

Constraints:

$$\textbf{nnza}[i-1] \geq 0;$$
$$\sum_{i=1}^{n+1} \textbf{nnza}[i-1] \geq 1.$$

5: **nnzasum** – Integer *Input*

On entry: the dimension of the arrays **irowa**, **icola** and **a**, at least the total number of all nonzeros in all matrices A_i.

Constraints:

$$\textbf{nnzasum} > 0;$$
$$\sum_{i=1}^{n+1} \textbf{nnza}[i-1] \leq \textbf{nnzasum}.$$

6: **irowa**[**nnzasum**] – const Integer *Input*
7: **icola**[**nnzasum**] – const Integer *Input*
8: **a**[**nnzasum**] – const double *Input*

On entry: nonzero elements in upper triangle of matrices A_i stored in coordinate storage. The first **nnza**$[0]$ elements belong to A_0, the following **nnza**$[1]$ elements belong to A_1, etc. See explanation above.

Constraints:

$1 \leq$ **irowa**$[i-1] \leq$ **dima**, **irowa**$[i-1] \leq$ **icola**$[i-1] \leq$ **dima**;
irowa and **icola** match the block diagonal pattern set by **blksizea**.

9: **nblk** – Integer *Input*

On entry: m_A, number of diagonal blocks in A_i matrices. As explained above it is equivalent to the number of matrix inequalities supplied in this call.

Constraint: **nblk** ≥ 1.

10: **blksizea**[**nblk**] – const Integer *Input*

On entry: if **nblk** > 1, sizes d_k of the diagonal blocks.

If **nblk** $= 1$, **blksizea** is not referenced and may be **NULL**.

Constraints:

$$\mathbf{blksizea}[i-1] \geq 1;$$
$$\sum_{i=1}^{m_A} \mathbf{blksizea}[i-1] = \mathbf{dima}.$$

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

On entry: if **idblk** $= 0$, new matrix inequalities are created. This is the only value allowed at the moment; nonzero values are reserved for future releases of the NAG C Library.

Constraint: **idblk** $= 0$.

On exit: the number of the last matrix inequality added. By definition, it is the number of the matrix inequalities already defined plus **nblk**.

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 3.2.1.2 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_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_INT

On entry, **dima** $=$ ⟨*value*⟩.
Constraint: **dima** > 0.

On entry, **nblk** $=$ ⟨*value*⟩.
Constraint: **nblk** > 0.

NE_INT_2

On entry, **nnzasum** $=$ ⟨*value*⟩ and sum (**nnza**) $=$ ⟨*value*⟩.
Constraint: **nnzasum** $\geq$ sum (**nnza**).

NE_INT_ARRAY_1

On entry, $i =$ ⟨*value*⟩ and **nnza**$[i-1] =$ ⟨*value*⟩.
Constraint: **nnza**$[i-1] \geq 0$.

On entry, sum (**nnza**) $=$ ⟨*value*⟩.
Constraint: sum (**nnza**) ≥ 1.

NE_INT_ARRAY_2

On entry, $i = \langle value \rangle$ and **blksizea**$[i-1] = \langle value \rangle$.
Constraint: **blksizea**$[i-1] \geq 1$.

NE_INTARR_INT

On entry, **dima** $= \langle value \rangle$ and sum (**blksizea**) $= \langle value \rangle$.
Constraint: sum (**blksizea**) = **dima**.

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 3.6.6 in How to Use the NAG Library and its Documentation for further information.

NE_INVALID_CS

An error occurred in matrix A_i, $i = \langle value \rangle$ (counting indices $1 \ldots$ **nvar** $+ 1$).
On entry, $j = \langle value \rangle$, **icola**$[j-1] = \langle value \rangle$ and **dima** $= \langle value \rangle$.
Constraint: $1 \leq$ **icola**$[j-1] \leq$ **dima**.

An error occurred in matrix A_i, $i = \langle value \rangle$ (counting indices $1 \ldots$ **nvar** $+ 1$).
On entry, $j = \langle value \rangle$, **irowa**$[j-1] = \langle value \rangle$ and **dima** $= \langle value \rangle$.
Constraint: $1 \leq$ **irowa**$[j-1] \leq$ **dima**.

An error occurred in matrix A_i, $i = \langle value \rangle$ (counting indices $1 \ldots$ **nvar** $+ 1$).
On entry, $j = \langle value \rangle$, **irowa**$[j-1] = \langle value \rangle$ and **icola**$[j-1] = \langle value \rangle$.
Constraint: **irowa**$[j-1] \leq$ **icola**$[j-1]$ (elements within the upper triangle).

An error occurred in matrix A_i, $i = \langle value \rangle$ (counting indices $1 \ldots$ **nvar** $+ 1$).
On entry, $j = \langle value \rangle$, **irowa**$[j-1] = \langle value \rangle$ and **icola**$[j-1] = \langle value \rangle$. Maximum column index in this row given by the block structure defined by **blksizea** is $\langle value \rangle$.
Constraint: all elements of A_i must respect the block structure given by **blksizea**.

An error occurred in matrix A_i, $i = \langle value \rangle$ (counting indices $1 \ldots$ **nvar** $+ 1$).
On entry, more than one element of A_i has row index $\langle value \rangle$ and column index $\langle value \rangle$.
Constraint: each element of A_i 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 3.6.5 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** $= \langle value \rangle$.
Constraint: **idblk** $= 0$.

On entry, **nvar** $= \langle value \rangle$, expected value $= \langle value \rangle$.
Constraint: **nvar** must match the value given during initialization of **handle**.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_opt_handle_set_linmatineq (e04rnc) is not threaded in any implementation.

9 Further Comments

The following example demonstrates how the elements of the A_i^k matrices are organized within the input arrays. Let us assume that there are two blocks defined (**nblk** $= 2$). The first has dimension 3 by 3 (**blksizea**$[0] = 3$) and the second 2 by 2 (**blksizea**$[1] = 2$). For simplicity, the number of variables is 2. Please note that the values were chosen to ease orientation rather than to define a valid problem.

$$A_0^1 = \begin{pmatrix} 0.1 & 0 & 0.3 \\ 0 & 0.2 & 0.4 \\ 0.3 & 0.4 & 0 \end{pmatrix}, \qquad A_1^1 \text{ empty} \qquad A_2^1 = \begin{pmatrix} 2.1 & 0 & 0 \\ 0 & 2.2 & 0 \\ 0 & 0 & 2.3 \end{pmatrix},$$

$$A_0^2 = \begin{pmatrix} 0 & -0.1 \\ -0.1 & 0 \end{pmatrix}, \quad A_1^2 = \begin{pmatrix} -1.1 & 0 \\ 0 & -1.2 \end{pmatrix}, \quad A_2^2 = \begin{pmatrix} -2.1 & -2.2 \\ -2.2 & -2.3 \end{pmatrix}.$$

Both inequalities will be passed in a single call to nag_opt_handle_set_linmatineq (e04rnc), therefore the matrices are merged into the following block diagonal form:

$$A_0 = \begin{pmatrix} 0.1 & 0 & 0.3 & & \\ 0 & 0.2 & 0.4 & & \\ 0.3 & 0.4 & 0 & & \\ & & & 0 & -0.1 \\ & & & -0.1 & 0 \end{pmatrix},$$

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

$$A_2 = \begin{pmatrix} 2.1 & 0 & 0 & & \\ 0 & 2.2 & 0 & & \\ 0 & 0 & 2.3 & & \\ & & & -2.1 & -2.2 \\ & & & -2.2 & -2.3 \end{pmatrix}.$$

All matrices are symmetric and therefore only the upper triangles are passed to the function. The coordinate storage format is used. Note that elements within the same matrix do not need to be in any specific order. The table below shows one of the ways the arrays could be populated.

irowa	2	2	4	1	1	4	5	1	2	3	4	4	5
icola	2	3	5	1	3	4	5	1	2	3	4	5	5
a	0.2	0.4	−0.1	0.1	0.3	−1.1	−1.2	2.1	2.2	2.3	−2.1	−2.2	−2.3
	A_0					A_1		A_2					
nnza	5					2		6					

10 Example

There are various problems which can be successfully reformulated and solved as an SDP problem. The following example shows how a maximization of the minimal eigenvalue of a matrix depending on certain parameters can be utilized in statistics.

For further examples, please refer to Section 10 in nag_opt_handle_init (e04rac).

Given a series of M vectors of length p, $\{v_i : i = 1, 2, \ldots, M\}$ this example solves the SDP problem:

$$\begin{aligned} \underset{\lambda_1,\ldots,\lambda_M,t}{\text{maximize}} \quad & t \\ \text{subject to} \quad & \sum_{i=1}^{M} \lambda_i v_i v_i^{\mathrm{T}} \succeq tI \\ & \sum_{i=1}^{M} \lambda_i = 1 \\ & \lambda_i \geq 0, \quad k = 1, \ldots, M. \end{aligned}$$

This formulation comes from an area of statistics called experimental design and corresponds to finding an approximate E optimal design for a linear regression.

A linear regression model has the form:

$$y = X\beta + \epsilon$$

where y is a vector of observed values, X is a design matrix of (known) independent variables and ϵ is a vector of errors. In experimental design it is assumed that each row of X is chosen from a set of M possible vectors, $\{v_i : i = 1, 2, \ldots, M\}$. The goal of experimental design is to choose the rows of X so that the error covariance is 'small'. For an E optimal design this is defined as the X that maximizes the minimum eigenvalue of $X^{\mathrm{T}}X$.

In this example we construct the E optimal design for a polynomial regression model of the form:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \epsilon$$

where $x \in \{1 - j \times 0.05 : j = 0, 1, \ldots, 40\}$.

10.1 Program Text

```
/* nag_opt_handle_set_linmatineq (e04rnc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

/* Compute E-optimal experiment design via semidefinite programming,
 * this can be done as follows
 *   max {lambda_min(A) | A = sum x_i*v_i*v_i^T, x_i>=0, sum x_i = 1}
 * where v_i are given vectors.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>

int main(void)
{
  const double big = 1e+20;
  const double tol = 0.00001;

  Integer exit_status = 0;
  Integer dima, i, idlc, idblk, idx, inform, j, k, m, nblk, nnzasum, nnzb,
         nnzu, nnzua, nnzuc, nvar, p;
  double blb[1], bub[1], c[1], rinfo[32], stats[32];
  Integer idxc[1];
  double *a = 0, *b = 0, *bl = 0, *bu = 0, *v = 0, *x = 0;
  Integer *icola = 0, *icolb = 0, *irowa = 0, *irowb = 0, *nnza = 0;
  void *handle = 0;
  /* Nag Types */
  NagError fail;

#define V(I,J) v[((I)-1)*p + (J)-1]
```

```
  printf("nag_opt_handle_set_linmatineq (e04rnc) Example Program Results\n\n");
  fflush(stdout);

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

  /* Read in the number of vectors and their size. */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n]", &m);
#else
  scanf("%" NAG_IFMT "%*[^\n]", &m);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n]", &p);
#else
  scanf("%" NAG_IFMT "%*[^\n]", &p);
#endif

  /* Allocate memory for the vectors and read them in. */
  if (!(v = NAG_ALLOC(m * p, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 1; i <= m; i++)
    for (j = 1; j <= p; j++)
#ifdef _WIN32
      scanf_s("%lf", &V(i, j));
#else
      scanf("%lf", &V(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Variables of the problem will be x_1, ..., x_m (weights of the vectors)
   * and t (artificial variable for minimum eigenvalue). */
  nvar = m + 1;

  /* 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_quadobj (e04rfc).
   * Add the objective function to the handle: max t. */
  idxc[0] = m + 1;
  c[0] = 1.0;
  nag_opt_handle_set_quadobj(handle, 1, idxc, c, 0, NULL, NULL, NULL,
                             NAGERR_DEFAULT);

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

  /* nag_opt_handle_set_simplebounds (e04rhc).
   * Add simple bounds on variables to the problem formulation. */
  for (i = 0; i < m; i++)
    bl[i] = 0.0;
  bl[m] = -big;
  for (i = 0; i <= m; i++)
```

```
  bu[i] = big;
nag_opt_handle_set_simplebounds(handle, nvar, bl, bu, NAGERR_DEFAULT);

/* Create linear constraint: sum x_i = 1. */
nnzb = m;
if (!(irowb = NAG_ALLOC(nnzb, Integer)) ||
    !(icolb = NAG_ALLOC(nnzb, Integer)) || !(b = NAG_ALLOC(nnzb, double)))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}
for (i = 0; i < m; i++) {
  /* irowb, icolb use 1-based indices */
  irowb[i] = 1;
  icolb[i] = i + 1;
  b[i] = 1.0;
}
blb[0] = 1.0;
bub[0] = 1.0;
idlc = 0;

/* nag_opt_handle_set_linconstr (e04rjc).
 * Add the linear constraint to the problem formulation. */
nag_opt_handle_set_linconstr(handle, 1, blb, bub, nnzb, irowb, icolb, b,
                             &idlc, NAGERR_DEFAULT);

/* Generate matrix constraint as:
 *   sum_{i=1}^m x_i*v_i*v_i^T - t*I >=0 */

nblk = 1;
idblk = 0;
dima = p;
/* Total number of nonzeros */
nnzasum = p + m * (p + 1) * p / 2;

if (!(nnza = NAG_ALLOC(nvar + 1, Integer)) ||
    !(irowa = NAG_ALLOC(nnzasum, Integer)) ||
    !(icola = NAG_ALLOC(nnzasum, Integer)) ||
    !(a = NAG_ALLOC(nnzasum, double)) || !(x = NAG_ALLOC(nvar, double)))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* A_0 is empty */
nnza[0] = 0;
/* A_1, A_2, ..., A_m are v_i*v_i^T */
idx = 0;
for (k = 1; k <= m; k++) {
  nnza[k] = (p + 1) * p / 2;
  for (i = 1; i <= p; i++)
    for (j = i; j <= p; j++) {
      irowa[idx] = i;
      icola[idx] = j;
      a[idx] = V(k, i) * V(k, j);
      idx++;
    }
}
/* A_{m+1} is the -identity */
nnza[m + 1] = p;
for (i = 1; i <= p; i++) {
  irowa[idx] = i;
  icola[idx] = i;
  a[idx] = -1.0;
  idx++;
}

/* nag_opt_handle_set_linmatineq (e04rnc).
 * Add the linear matrix constraint to the problem formulation. */
```

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

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

  /* Pass the handle to the solver, we are not interested in
   * Lagrangian multipliers.
   * nag_opt_handle_solve_pennon (e04svc). */
  nnzu = 0;
  nnzuc = 0;
  nnzua = 0;
  INIT_FAIL(fail);
  nag_opt_handle_solve_pennon(handle, nvar, x, nnzu, NULL, nnzuc, NULL,
                              nnzua, 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;
  }

  /* Print results */
  printf("\n  Weight        Row of design matrix");
  for (i = 1; i <= m; i++)
    if (x[i - 1] > tol) {
      printf("\n %7.2f      ", x[i - 1]);
      for (j = 1; j <= p; j++)
        printf("%7.2f ", V(i, j));
    }
  printf("\n only those rows with a weight > %7.1e are shown\n", tol);

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(b);
  NAG_FREE(bl);
  NAG_FREE(bu);
  NAG_FREE(v);
  NAG_FREE(x);
  NAG_FREE(icola);
  NAG_FREE(icolb);
  NAG_FREE(irowa);
  NAG_FREE(irowb);
  NAG_FREE(nnza);
  return exit_status;
}
```

10.2 Program Data

```
nag_opt_handle_set_linmatineq (e04rnc) Example Program Data
41          : Number of vectors to choose from
5           : Length of vectors
 1.00000000 -1.00000000  1.00000000 -1.00000000  1.00000000
 1.00000000 -0.95000000  0.90250000 -0.85737500  0.81450625
 1.00000000 -0.90000000  0.81000000 -0.72900000  0.65610000
 1.00000000 -0.85000000  0.72250000 -0.61412500  0.52200625
 1.00000000 -0.80000000  0.64000000 -0.51200000  0.40960000
 1.00000000 -0.75000000  0.56250000 -0.42187500  0.31640625
 1.00000000 -0.70000000  0.49000000 -0.34300000  0.24010000
 1.00000000 -0.65000000  0.42250000 -0.27462500  0.17850625
 1.00000000 -0.60000000  0.36000000 -0.21600000  0.12960000
 1.00000000 -0.55000000  0.30250000 -0.16637500  0.09150625
```

```
 1.00000000 -0.50000000  0.25000000 -0.12500000  0.06250000
 1.00000000 -0.45000000  0.20250000 -0.09112500  0.04100625
 1.00000000 -0.40000000  0.16000000 -0.06400000  0.02560000
 1.00000000 -0.35000000  0.12250000 -0.04287500  0.01500625
 1.00000000 -0.30000000  0.09000000 -0.02700000  0.00810000
 1.00000000 -0.25000000  0.06250000 -0.01562500  0.00390625
 1.00000000 -0.20000000  0.04000000 -0.00800000  0.00160000
 1.00000000 -0.15000000  0.02250000 -0.00337500  0.00050625
 1.00000000 -0.10000000  0.01000000 -0.00100000  0.00010000
 1.00000000 -0.05000000  0.00250000 -0.00012500  0.00000625
 1.00000000  0.00000000  0.00000000  0.00000000  0.00000000
 1.00000000  0.05000000  0.00250000  0.00012500  0.00000625
 1.00000000  0.10000000  0.01000000  0.00100000  0.00010000
 1.00000000  0.15000000  0.02250000  0.00337500  0.00050625
 1.00000000  0.20000000  0.04000000  0.00800000  0.00160000
 1.00000000  0.25000000  0.06250000  0.01562500  0.00390625
 1.00000000  0.30000000  0.09000000  0.02700000  0.00810000
 1.00000000  0.35000000  0.12250000  0.04287500  0.01500625
 1.00000000  0.40000000  0.16000000  0.06400000  0.02560000
 1.00000000  0.45000000  0.20250000  0.09112500  0.04100625
 1.00000000  0.50000000  0.25000000  0.12500000  0.06250000
 1.00000000  0.55000000  0.30250000  0.16637500  0.09150625
 1.00000000  0.60000000  0.36000000  0.21600000  0.12960000
 1.00000000  0.65000000  0.42250000  0.27462500  0.17850625
 1.00000000  0.70000000  0.49000000  0.34300000  0.24010000
 1.00000000  0.75000000  0.56250000  0.42187500  0.31640625
 1.00000000  0.80000000  0.64000000  0.51200000  0.40960000
 1.00000000  0.85000000  0.72250000  0.61412500  0.52200625
 1.00000000  0.90000000  0.81000000  0.72900000  0.65610000
 1.00000000  0.95000000  0.90250000  0.85737500  0.81450625
 1.00000000  1.00000000  1.00000000  1.00000000  1.00000000
```

10.3 Program Results

```
nag_opt_handle_set_linmatineq (e04rnc) Example Program Results

 E04SV, NLP-SDP Solver (Pennon)
 ------------------------------
 Number of variables            42                  [eliminated            0]
                           simple  linear  nonlin
 (Standard) inequalities       41       2       0
 (Standard) equalities                  0       0
 Matrix inequalities                    1       0 [dense    1, sparse    0]
                                                  [max dimension         5]

 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         =          Equalities     * S
     Dimacs Measures               =               Check     * d
```

```
    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                          =            Maximize     * U
    P Update Speed                =                  12     * d
End of Options
------------------------------------------------------------
 it|  objective |  optim  |   feas  |  compl  | pen min |inner
------------------------------------------------------------
  0  0.00000E+00  4.80E+01  5.90E-01  2.37E+00  1.00E+00   0
  1 -2.25709E+00  2.53E-03  7.15E-01  2.76E+00  1.00E+00   6
  2 -9.90666E-01  1.29E-03  1.38E-02  1.25E+00  4.65E-01   5
  3 -3.96590E-01  1.52E-03  2.07E-02  5.42E-01  2.16E-01   5
  4 -1.52400E-01  6.63E-04  1.42E-02  2.26E-01  1.01E-01   5
  5 -5.45545E-02  5.47E-03  9.33E-03  8.91E-02  4.68E-02   5
  6 -1.62316E-02  1.05E-02  3.18E-03  3.33E-02  2.18E-02   5
  7 -2.39571E-03  6.74E-03  3.90E-04  1.22E-02  1.01E-02   5
  8  3.39831E-03  5.41E-04  4.33E-05  4.43E-03  4.71E-03   6
  9  6.27924E-03  2.25E-03  3.47E-06  1.64E-03  2.19E-03   5
 10  7.23641E-03  4.07E-03  4.79E-07  5.77E-04  1.02E-03   4
 11  7.56230E-03  5.26E-04  1.76E-05  2.08E-04  4.74E-04   4
 12  7.67523E-03  1.18E-02  2.18E-06  7.69E-05  2.21E-04   3
 13  7.71758E-03  4.26E-03  2.51E-07  2.94E-05  1.03E-04   3
 14  7.73491E-03  4.34E-06  2.95E-08  1.11E-05  4.77E-05   4
------------------------------------------------------------
 it|  objective |  optim  |   feas  |  compl  | pen min |inner
------------------------------------------------------------
 15  7.74186E-03  8.50E-07  2.29E-09  3.96E-06  2.22E-05   4
 16  7.74450E-03  7.25E-08  1.58E-10  1.29E-06  1.03E-05   4
 17  7.74545E-03  2.51E-09  8.39E-12  3.32E-07  4.81E-06   4
 18  7.74574E-03  5.19E-10  3.49E-13  4.73E-08  2.24E-06   4
------------------------------------------------------------
Status: converged, an optimal solution found
------------------------------------------------------------
Final objective value               7.745738E-03
Relative precision                  2.815426E-07
Optimality                          5.188682E-10
Feasibility                         3.486927E-13
Complementarity                     4.732416E-08
DIMACS error 1                      2.594341E-10
DIMACS error 2                      0.000000E+00
DIMACS error 3                      0.000000E+00
DIMACS error 4                      1.743464E-13
DIMACS error 5                      4.676597E-08
DIMACS error 6                      4.662494E-08
Iteration counts
  Outer iterations                            18
  Inner iterations                            81
  Linesearch steps                           186
Evaluation counts
  Augm. Lagr. values                         100
  Augm. Lagr. gradient                       100
  Augm. Lagr. hessian                         81
------------------------------------------------------------

 Weight        Row of design matrix
   0.09         1.00   -1.00    1.00   -1.00    1.00
   0.25         1.00   -0.70    0.49   -0.34    0.24
   0.32         1.00    0.00    0.00    0.00    0.00
   0.25         1.00    0.70    0.49    0.34    0.24
   0.09         1.00    1.00    1.00    1.00    1.00
only those rows with a weight > 1.0e-05 are shown
```