NAG Library Function Document

nag_opt_handle_set_quadobj (e04rfc)

1 Purpose

nag_opt_handle_set_quadobj (e04rfc) is a part of the NAG optimization modelling suite and defines the linear or the quadratic objective function of the problem.

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_handle_set_quadobj (void *handle, Integer nnzc,
     const Integer idxc[], const double c[], Integer nnzh,
     const Integer irowh[], const Integer icolh[], const double h[],
     NagError *fail)
```

3 Description

After the initialization function nag_opt_handle_init (e04rac) has been called, nag_opt_handle_set_quadobj (e04rfc) may be used to define the objective function of the problem as a quadratic function $c^{\mathrm{T}}x + \frac{1}{2}x^{\mathrm{T}}Hx$ or a sparse linear function $c^{\mathrm{T}}x$ unless the objective function has been defined previously by nag_opt_handle_set_linobj (e04rec), nag_opt_handle_set_quadobj (e04rfc) or by nag_opt_handle_set_nlnobj (e04rgc). This objective function will typically be used for quadratic programming problems (QP)

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & \frac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} & l_B \leq Bx \leq u_B & \text{(b)} \\ & l_x \leq x \leq u_x & \text{(c)} \end{array} \tag{1}$$

or for semidefinite programming problems with bilinear matrix inequalities (BMI-SDP)

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

The matrix H is a sparse symmetric n by n matrix. It does not need to be positive definite. See nag_opt_handle_init (e04rac) for more details.

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: **nnzc** – Integer *Input*

On entry: the number of nonzero elements in the sparse vector c.

If **nnzc** $= 0$, c is considered to be zero and the arrays **idxc** and **c** will not be referenced and may be **NULL**.

Constraint: **nnzc** ≥ 0.

3: **idxc**[**nnzc**] – const Integer *Input*

4: **c**[**nnzc**] – const double *Input*

On entry: the nonzero elements of the sparse vector c. **idxc**$[i-1]$ must contain the index of **c**$[i-1]$ in the vector, for $i = 1, 2, \ldots,$ **nnzc**. The elements are stored in ascending order. Note that n, the number of variables in the problem, was set in **nvar** during the initialization of the handle by nag_opt_handle_init (e04rac).

Constraints:

$$1 \leq \textbf{idxc}[i-1] \leq n, \text{ for } i = 1, 2, \ldots, \textbf{nnzc};$$
$$\textbf{idxc}[i-1] < \textbf{idxc}[i], \text{ for } i = 1, 2, \ldots, \textbf{nnzc} - 1.$$

5: **nnzh** – Integer *Input*

On entry: the number of nonzero elements in the upper triangle of the matrix H.

If **nnzh** $= 0$, the matrix H is considered to be zero, the objective function is linear and **irowh**, **icolh** and **h** will not be referenced and may be **NULL**.

Constraint: **nnzh** ≥ 0.

6: **irowh**[**nnzh**] – const Integer *Input*

7: **icolh**[**nnzh**] – const Integer *Input*

8: **h**[**nnzh**] – const double *Input*

On entry: arrays **irowh**, **icolh** and **h** store the nonzeros of the upper triangle of the matrix H in coordinate storage (CS) format (see Section 2.1.1 in the f11 Chapter Introduction). **irowh** specifies one-based row indices, **icolh** specifies one-based column indices and **h** specifies the values of the nonzero elements in such a way that $h_{ij} =$ **h**$[l-1]$ where $i =$ **irowh**$[l-1]$, $j =$ **icolh**$[l-1]$, for $l = 1, 2, \ldots,$ **nnzh**. No particular order is expected, but elements should not repeat.

Constraint: $1 \leq$ **irowh**$[l-1] \leq$ **icolh**$[l-1] \leq n$, for $l = 1, 2, \ldots,$ **nnzh**.

9: **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

The objective function has already been defined.

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, **nnzc** $= \langle value \rangle$.
Constraint: **nnzc** ≥ 0.

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

NE_INTARR

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

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, $i = \langle value \rangle$, **icolh**$[i-1] = \langle value \rangle$ and $n = \langle value \rangle$.
Constraint: $1 \leq$ **icolh**$[i-1] \leq n$.

On entry, $i = \langle value \rangle$, **irowh**$[i-1] = \langle value \rangle$ and **icolh**$[i-1] = \langle value \rangle$.
Constraint: **irowh**$[i-1] \leq$ **icolh**$[i-1]$ (elements within the upper triangle).

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

On entry, more than one element of **h** has row index $\langle value \rangle$ and column index $\langle value \rangle$.
Constraint: each element of **h** 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_NOT_INCREASING

On entry, $i = \langle value \rangle$, **idxc**$[i-1] = \langle value \rangle$ and **idxc**$[i] = \langle value \rangle$.
Constraint: **idxc**$[i-1] <$ **idxc**$[i]$ (ascending order).

NE_PHASE

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

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_opt_handle_set_quadobj (e04rfc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example demonstrates how to use nonlinear semidefinite programming to find a nearest correlation matrix satisfying additional requirements. This is a viable alternative to functions nag_nearest_correlation (g02aac), nag_nearest_correlation_bounded (g02abc), nag_nearest_correlation_h_weight (g02ajc) or nag_nearest_correlation_shrinking (g02anc) as it easily allows you to add further constraints on the correlation matrix. In this case a problem with a linear matrix inequality and a quadratic objective function is formulated to find the nearest correlation matrix in the Frobenius norm preserving the nonzero pattern of the original input matrix. However, additional box bounds (nag_opt_handle_set_simplebounds (e04rhc)) or linear constraints (nag_opt_handle_set_linconstr (e04rjc)) can be readily added to further bind individual elements of the new correlation matrix or new matrix inequalities (nag_opt_handle_set_linmatineq (e04rnc)) to restrict its eigenvalues.

The problem is as follows (to simplify the notation only the upper triangular parts are shown). To a given m by m symmetric input matrix G

$$G = \begin{pmatrix} g_{11} & \cdots & g_{1m} \\ & \ddots & \vdots \\ & & g_{mm} \end{pmatrix}$$

find correction terms $x_1, \ldots, x_n$ which form symmetric matrix $\bar{G}$

$$\bar{G} = \begin{pmatrix} \bar{g}_{11} & \bar{g}_{12} & \cdots & \bar{g}_{1m} \\ & \bar{g}_{22} & \cdots & \bar{g}_{2m} \\ & & \ddots & \vdots \\ & & & \bar{g}_{mm} \end{pmatrix} = \begin{pmatrix} 1 & g_{12}+x_1 & g_{13}+x_2 & \cdots & & g_{1m}+x_i \\ & 1 & g_{23}+x_3 & & & \\ & & 1 & & & \vdots \\ & & & \ddots & & \\ & & & & 1 & g_{m-1m}+x_n \\ & & & & & 1 \end{pmatrix}$$

so that the following requirements are met:

(a) It is a correlation matrix, i.e., symmetric positive semidefinite matrix with a unit diagonal. This is achieved by the way $\bar{G}$ is assembled and by a linear matrix inequality

$$\bar{G} = x_1 \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ & 0 & 0 & \cdots & 0 \\ & & 0 & \cdots & 0 \\ & & & \ddots & \vdots \\ & & & & 0 \end{pmatrix} + x_2 \begin{pmatrix} 0 & 0 & 1 & \cdots & 0 \\ & 0 & 0 & \cdots & 0 \\ & & 0 & \cdots & 0 \\ & & & \ddots & \vdots \\ & & & & 0 \end{pmatrix} + x_3 \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ & 0 & 1 & \cdots & 0 \\ & & 0 & \cdots & 0 \\ & & & \ddots & \vdots \\ & & & & 0 \end{pmatrix} + \cdots$$
$$+ x_n \begin{pmatrix} 0 & \cdots & 0 & 0 & 0 \\ & \ddots & \vdots & \vdots & \vdots \\ & & 0 & 0 & 0 \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix} - \begin{pmatrix} -1 & -g_{12} & -g_{13} & \cdots & -g_{1m} \\ & -1 & -g_{23} & \cdots & -g_{2m} \\ & & -1 & \cdots & -g_{3m} \\ & & & \ddots & \vdots \\ & & & & -1 \end{pmatrix} \succeq 0.$$

(b) $\bar{G}$ is nearest to G in the Frobenius norm, i.e., it minimizes the Frobenius norm of the difference which is equivalent to:

$$\text{minimize } \frac{1}{2}\sum_{i \neq j} \left(\bar{g}_{ij} - g_{ij}\right)^2 = \sum_{i=1}^{n} x_i^2.$$

(c) $\bar{G}$ preserves the nonzero structure of G. This is met by defining x_i only for nonzero elements g_{ij}.

For the input matrix

$$G = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}$$

the result is

$$\bar{G} = \begin{pmatrix} 1.0000 & -0.6823 & 0.0000 & 0.0000 \\ -0.6823 & 1.0000 & -0.5344 & 0.0000 \\ 0.0000 & -0.5344 & 1.0000 & -0.6823 \\ 0.0000 & 0.0000 & -0.6823 & 1.0000 \end{pmatrix}.$$

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

10.1 Program Text

```
/* nag_opt_handle_set_quadobj (e04rfc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

/* Compute the nearest correlation matrix in Frobenius norm using nonlinear
 * semidefinite programming. By default, preserve the nonzero structure of
 * the input matrix (preserve_structure = 1).
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagx04.h>

int main(void)
{
  Integer preserve_structure = 1;

  Integer exit_status = 0;
  Integer dima, i, idblk, idx, inform, j, n, nblk, nnzasum, nnzh, nnzu,
         nnzua, nnzuc, nvar;
  double rinfo[32], stats[32];
  double *a = 0, *g = 0, *h = 0, *x = 0;
  Integer *icola = 0, *icolh = 0, *irowa = 0, *irowh = 0, *nnza = 0;
  void *handle = 0;
  /* Nag Types */
  NagError fail;

#define G(I,J) g[(J-1)*n + I-1]

  printf("nag_opt_handle_set_quadobj (e04rfc) Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file. */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  /* Read in the problem size and ignore the rest of the line. */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n]", &n);
#else
  scanf("%" NAG_IFMT "%*[^\n]", &n);
#endif
```

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

  /* Symmetrize G:  G = (G + G')/2 */
  for (j = 1; j <= n; j++)
    for (i = 1; i <= j - 1; i++) {
      G(i, j) = (G(i, j) + G(j, i)) / 2.0;
      G(j, i) = G(i, j);
    }

  /* There are as many variables as nonzeros above the main diagonal in
   * the input matrix. The variables are corrections of these elements. */
  nvar = 0;
  for (j = 2; j <= n; j++)
    for (i = 1; i <= j - 1; i++)
      if (!preserve_structure || G(i, j) != 0.0)
        nvar++;

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

  /* Set up the objective - minimize Frobenius norm of the corrections.
   * Our variables are stored as a vector thus, just minimize
   * sum of squares of the corrections --> H is identity matrix, c = 0.*/
  nnzh = nvar;
  if (!(x = NAG_ALLOC(nvar, double)) ||
      !(irowh = NAG_ALLOC(nnzh, Integer)) ||
      !(icolh = NAG_ALLOC(nnzh, Integer)) || !(h = NAG_ALLOC(nnzh, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  for (i = 0; i < nvar; i++) {
    /* irowh, icolh use 1-based indices */
    irowh[i] = i + 1;
    icolh[i] = i + 1;
    h[i] = 1.0;
  }

  /* nag_opt_handle_set_quadobj (e04rfc).
   * Add the quadratic objective to the handle.*/
  nag_opt_handle_set_quadobj(handle, 0, NULL, NULL, nnzh, irowh, icolh, h,
                             NAGERR_DEFAULT);

  /* Construct linear matrix inequality to request that
   * matrix G with corrections X is positive semidefinite.
   * (Don't forget the sign at A_0!)
   * How many nonzeros do we need? Full triangle for A_0 and
   * one nonzero element for each A_i. */
  nnzasum = n * (n + 1) / 2 + nvar;
```

```
  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;
  }
  nnza[0] = n * (n + 1) / 2;
  for (j = 1; j <= nvar; j++)
    nnza[j] = 1;

  /* Copy G to A_0, only upper triangle with different sign (because -A_0)
   * and set the diagonal to 1.0 as that's what we want independently
   * of what was in G. */
  idx = 0;
  for (j = 1; j <= n; j++) {
    for (i = 1; i <= j - 1; i++) {
      irowa[idx] = i;
      icola[idx] = j;
      a[idx] = -G(i, j);
      idx++;
    }
    /* Unit diagonal. */
    irowa[idx] = j;
    icola[idx] = j;
    a[idx] = -1.0;
    idx++;
  }

  /* A_i has just one nonzero - it binds x_i with its position as
   * a correction. */
  for (j = 2; j <= n; j++)
    for (i = 1; i <= j - 1; i++)
      if (!preserve_structure || G(i, j) != 0.0) {
        irowa[idx] = i;
        icola[idx] = j;
        a[idx] = 1.0;
        idx++;
      }

  /* Just one matrix inequality of the dimension of the original matrix. */
  nblk = 1;
  dima = n;
  idblk = 0;
  /* nag_opt_handle_set_linconstr (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, "Print Options = No", 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. */
  nnzu = 0;
  nnzuc = 0;
  nnzua = 0;
  INIT_FAIL(fail);
  /* nag_opt_handle_solve_pennon (e04svc). */
  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;
  }
```

```
  /* Form the new nearest correlation matrix as the sum
   * of G and the correction X. */
  idx = 0;
  for (j = 1; j <= n; j++) {
    for (i = 1; i <= j - 1; i++)
      if (!preserve_structure || G(i, j) != 0.0) {
        G(i, j) = G(i, j) + x[idx];
        idx++;
      }
    G(j, j) = 1.0;
  }

  /* nag_gen_real_mat_print (x04cac).
   * Print the result as an upper triangular matrix. */
  nag_gen_real_mat_print(Nag_ColMajor, Nag_UpperMatrix, Nag_NonUnitDiag, n, n,
                         g, n, "Nearest Correlation Matrix", NULL,
                         NAGERR_DEFAULT);

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(g);
  NAG_FREE(h);
  NAG_FREE(x);
  NAG_FREE(icola);
  NAG_FREE(icolh);
  NAG_FREE(irowa);
  NAG_FREE(irowh);
  NAG_FREE(nnza);
  return exit_status;
}
```

10.2 Program Data

```
nag_opt_handle_set_quadobj (e04rfc) Example Program Data
 4                          :: n
 2.0   -1.0    0.0    0.0
-1.0    2.0   -1.0    0.0
 0.0   -1.0    2.0   -1.0
 0.0    0.0   -1.0    2.0  :: End of g
```

10.3 Program Results

```
nag_opt_handle_set_quadobj (e04rfc) Example Program Results

 E04SV, NLP-SDP Solver (Pennon)
 ------------------------------
 Number of variables             3                 [eliminated            0]
                           simple  linear  nonlin
 (Standard) inequalities        0       0       0
 (Standard) equalities                  0       0
 Matrix inequalities                    1       0 [dense    1, sparse    0]
                                                  [max dimension         4]

 ------------------------------------------------------------------
  it|  objective |  optim  |   feas  |  compl  | pen min |inner
 ------------------------------------------------------------------
   0  0.00000E+00  0.00E+00  6.19E-01  6.63E+00  1.00E+00   0
   1  4.12017E-01  6.38E-04  0.00E+00  1.44E+00  1.00E+00   5
   2  3.29642E-01  7.76E-04  0.00E+00  4.96E-01  4.65E-01   2
   3  2.65315E-01  1.02E-04  0.00E+00  1.55E-01  2.16E-01   3
   4  2.33229E-01  1.03E-03  0.00E+00  4.71E-02  1.01E-01   3
   5  2.19082E-01  2.22E-03  0.00E+00  1.46E-02  4.68E-02   3
   6  2.13121E-01  2.12E-03  0.00E+00  4.72E-03  2.18E-02   3
```

```
   7  2.10698E-01  1.26E-03  0.00E+00  1.56E-03  1.01E-02   3
   8  2.09756E-01  4.90E-04  0.00E+00  4.85E-04  4.71E-03   3
   9  2.09413E-01  1.13E-04  0.00E+00  1.21E-04  2.19E-03   3
  10  2.09310E-01  1.95E-03  0.00E+00  1.63E-05  1.02E-03   2
  11  2.09297E-01  1.25E-05  0.00E+00  2.77E-06  4.74E-04   2
  12  2.09294E-01  2.68E-07  0.00E+00  3.89E-07  2.21E-04   2
  13  2.09294E-01  2.25E-09  0.00E+00  5.43E-08  1.03E-04   2
------------------------------------------------------------
Status: converged, an optimal solution found
------------------------------------------------------------
Final objective value                2.092940E-01
Relative precision                   2.759238E-07
Optimality                           2.249294E-09
Feasibility                          0.000000E+00
Complementarity                      5.426796E-08
Iteration counts
  Outer iterations                             13
  Inner iterations                             36
  Linesearch steps                             36
Evaluation counts
  Augm. Lagr. values                           50
  Augm. Lagr. gradient                         50
  Augm. Lagr. hessian                          36
------------------------------------------------------------
Nearest Correlation Matrix
        1       2       3       4
1   1.0000 -0.6823  0.0000  0.0000
2          1.0000 -0.5344  0.0000
3                 1.0000 -0.6823
4                        1.0000
```