NAG Library Function Document

nag_opt_handle_set_simplebounds (e04rhc)

1 Purpose

nag_opt_handle_set_simplebounds (e04rhc) is a part of the NAG optimization modelling suite and defines bounds on the variables of the problem.

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_handle_set_simplebounds (void *handle, Integer nvar,
     const double bl[], const double bu[], NagError *fail)
```

3 Description

After the initialization routine nag_opt_handle_init (e04rac) has been called, nag_opt_handle_set_simplebounds (e04rhc) may be used to define the variable bounds $l_x \leq x \leq u_x$ of the problem unless the bounds have already been defined. This will typically be used for problems, such as quadratic programming (QP)

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

nonlinear programming (NLP)

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & f(x) & \text{(a)} \\ \text{subject to} & l_g \leq g(x) \leq u_g & \text{(b)} \\ & l_B \leq Bx \leq u_B & \text{(c)} \\ & l_x \leq x \leq u_x & \text{(d)} \end{array} \tag{2}$$

linear semidefinite programming (SDP)

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

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

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & \frac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} & \displaystyle\sum_{i,j=1}^{n} x_i x_j Q_{ij}^k + \sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1,\ldots,m_A & \text{(b)} \\ & l_B \leq Bx \leq u_B & \text{(c)} \\ & l_x \leq x \leq u_x & \text{(d)} \end{array} \tag{4}$$

where l_x and u_x are n-dimensional vectors. Note that upper and lower bounds are specified for all the variables. This form allows full generality in specifying various types of constraint. If certain bounds are not present, the associated elements of l_x or u_x may be set to special values that are treated as $-\infty$ or $+\infty$. See the description of the optional parameter **Infinite Bound Size** of the solvers in the suite, nag_opt_handle_solve_ipopt (e04stc) and nag_opt_handle_solve_pennon (e04svc). Its value is denoted

as $bigbnd$ further in this text. Note that the bounds are interpreted based on its value at the time of calling this function and any later alterations to **Infinite Bound Size** will not affect these constraints.

See nag_opt_handle_init (e04rac) for more details.

4 References

Candes E and Recht B (2009) Exact matrix completion via convex optimization *Foundations of Computation Mathematics (Volume 9)* 717–772

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: **bl**[**nvar**] – const double *Input*

4: **bu**[**nvar**] – const double *Input*

On entry: l_x, **bl** and u_x, **bu** define lower and upper bounds on the variables, respectively. To specify a nonexistent lower bound (i.e., $l_j = -\infty$), set $\mathbf{bl}[j-1] \leq -bigbnd$; to specify a nonexistent upper bound (i.e., $u_j = \infty$), set $\mathbf{bu}[j-1] \geq bigbnd$. Fixing of the variables is not allowed in this release, however, this limitation will be removed in a future release.

Constraints:

$\mathbf{bl}[j-1] < \mathbf{bu}[j-1]$, for $j = 1, 2, \ldots, \mathbf{nvar}$;
$\mathbf{bl}[j-1] < bigbnd$, for $j = 1, 2, \ldots, \mathbf{nvar}$;
$\mathbf{bu}[j-1] > -bigbnd$, for $j = 1, 2, \ldots, \mathbf{nvar}$.

5: **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_ALREADY_DEFINED

Variable bounds have already been defined.

NE_BAD_PARAM

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

NE_BOUND

On entry, $j = \langle value \rangle$, $\mathbf{bl}[j-1] = \langle value \rangle$, $bigbnd = \langle value \rangle$.
Constraint: $\mathbf{bl}[j-1] < bigbnd$.

On entry, $j = \langle value \rangle$, $\mathbf{bl}[j-1] = \langle value \rangle$ and $\mathbf{bu}[j-1] = \langle value \rangle$.
Constraint: $\mathbf{bl}[j-1] < \mathbf{bu}[j-1]$.

On entry, $j = \langle value \rangle$, $\mathbf{bu}[j-1] = \langle value \rangle$, $bigbnd = \langle value \rangle$.
Constraint: $\mathbf{bu}[j-1] > -bigbnd$.

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_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_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, **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_simplebounds (e04rhc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

There is a vast number of problems which can be reformulated as SDP. This example follows Candes and Recht (2009) to show how a rank minimization problem can be approximated by SDP. In addition, it demonstrates how to work with the monitor mode of nag_opt_handle_solve_pennon (e04svc).

The problem can be stated as follows: Let's have m respondents answering k questions where they express their preferences as a number between 0 and 1 or the question can be left unanswered. The task is to fill in the missing entries, i.e., to guess the unexpressed preferences. This problem falls into the category of matrix completion. The idea is to choose the missing entries to minimize the rank of the matrix as it is commonly believed that only a few factors contribute to an individual's tastes or preferences.

Rank minimization is in general NP-hard but it can be approximated by a heuristic, minimizing the nuclear norm of the matrix. The nuclear norm of a matrix is the sum of its singular values. A rank deficient matrix must have (several) zero singular values. Given the fact that the singular values are

always non-negative, a minimization of the nuclear norm has the same effect as ℓ_1 norm in compress sensing, i.e., it encourages many singular values to be zero and thus it can be considered as a heuristic for the original rank minimization problem.

Let $\hat{Y}$ denote the partially filled in $m \times k$ matrix with the valid responses on $(i,j) \in \Omega$ positions. We are looking for Y of the same size so that the valid responses are unchanged and the nuclear norm (denoted here as $\|\cdot\|_*$) is minimal.

$$\begin{aligned} \underset{Y}{\text{minimize}} \quad & \|Y\|_* \\ \text{subject to} \quad & Y_{ij} = \hat{Y}_{ij} \quad \text{for all} \quad (i,j) \in \Omega. \end{aligned}$$

This is equivalent to

$$\begin{aligned} \underset{W_1,W_2,Y}{\text{minimize}} \quad & \text{trace}(W_1) + \text{trace}(W_2) \\ \text{subject to} \quad & Y_{ij} = \hat{Y}_{ij} \quad \text{for all} \quad (i,j) \in \Omega \\ & \begin{pmatrix} W_1 & Y \\ Y^{\mathrm{T}} & W_2 \end{pmatrix} \succeq 0 \end{aligned}$$

which is the linear semidefinite problem solved in this example, see Candes and Recht (2009) and the references therein for details.

This example has $m = 15$ respondents and $k = 6$ answers. The obtained answers are

$$\hat{Y} = \begin{pmatrix} * & * & * & * & * & 0.4 \\ 0.6 & 0.4 & 0.8 & * & * & * \\ * & * & 0.8 & * & 0.2 & * \\ 0.8 & 0.2 & * & * & * & * \\ * & 0.4 & * & 0.0 & * & 0.2 \\ 0.4 & * & * & 0.2 & * & 0.2 \\ * & 0.8 & 0.2 & 0.6 & * & * \\ * & * & 0.2 & * & * & * \\ * & 0.4 & * & 0.6 & 0.0 & * \\ * & * & 0.4 & * & * & * \\ * & * & 0.2 & 0.2 & 0.4 & 0.4 \\ * & * & * & * & 1.0 & 0.8 \\ 1.0 & * & 0.2 & * & * & 0.6 \\ * & * & * & * & * & 0.2 \\ 0.6 & * & 0.2 & 0.4 & * & * \end{pmatrix}$$

where $*$ denotes missing entries (-1.0 is used instead in the data file). The obtained matrix has rank 4 and it is shown below printed to 1-digit accuracy:

$$Y = \begin{pmatrix} 0.5 & 0.3 & 0.2 & 0.2 & 0.4 & 0.4 \\ 0.6 & 0.4 & 0.8 & 0.2 & 0.3 & 0.4 \\ 0.4 & 0.3 & 0.8 & 0.0 & 0.2 & 0.2 \\ 0.8 & 0.2 & 0.3 & 0.4 & 0.3 & 0.4 \\ 0.0 & 0.4 & 0.2 & 0.0 & 0.2 & 0.2 \\ 0.4 & 0.1 & 0.2 & 0.2 & 0.1 & 0.2 \\ 0.6 & 0.8 & 0.2 & 0.6 & 0.2 & 0.4 \\ 0.1 & 0.1 & 0.2 & 0.0 & 0.0 & 0.1 \\ 0.6 & 0.4 & 0.1 & 0.6 & 0.0 & 0.3 \\ 0.2 & 0.1 & 0.4 & 0.0 & 0.1 & 0.1 \\ 0.5 & 0.3 & 0.2 & 0.2 & 0.4 & 0.4 \\ 0.7 & 0.4 & 0.3 & 0.0 & 1.0 & 0.8 \\ 1.0 & 0.3 & 0.2 & 0.5 & 0.5 & 0.6 \\ 0.2 & 0.1 & 0.1 & 0.1 & 0.2 & 0.2 \\ 0.6 & 0.3 & 0.2 & 0.4 & 0.2 & 0.3 \end{pmatrix}.$$

The example also turns on monitor mode of nag_opt_handle_solve_pennon (e04svc), there is a time limit introduced for the solver which is being checked at the end of every outer iteration. If the time limit is reached, the routine is stopped by setting **inform** $= 0$ within the monitor step.

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_simplebounds (e04rhc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

/* Matrix completion problem (rank minimization) solved approximatelly
 *    by SDP via nuclear norm minimization formulated as:
 *       min    trace(X1) + trace(X2)
 *       s.t.   [ X1, Y; Y', X2 ] >=0
 *              0 <= Y_ij <= 1
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagf08.h>
#include <nagx04.h>
#include <math.h>

int
main(void)
{
  double const stol = 1e-5;
  double const time_limit = 120.0;

  Integer      exit_status = 0;
  Integer      dima, i, idblk, idx, idxend, idxobj, idxx, inform, j, lwork, m,
               maxs,
               n, nblk, nnz, nnzasum, nnzc, nnzu, nnzua, nnzuc, nvar, rank;
  double       rdummy[1], rinfo[32], stats[32];
  double       *a = 0, *bl = 0, *bu = 0, *c = 0, *s = 0, *work = 0, *x = 0,
  *y = 0;
  Integer      *icola = 0, *idxc = 0, *irowa = 0, *nnza = 0;
  void         *handle = 0;
  /* Nag Types */
  NagError     fail;

#define Y(I, J) y[(J-1)*m + I-1]

  printf("nag_opt_handle_set_simplebounds (e04rhc) 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 " %" NAG_IFMT "%*[^\n]", &m, &n);
#else
  scanf("%" NAG_IFMT " %" NAG_IFMT "%*[^\n]", &m, &n);
#endif

  /* Allocate memory for matrix Y and read it in. */
  if (!(y = NAG_ALLOC(m * n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
```

```
  for (i = 1; i <= m; i++)
    for (j = 1; j <= n; j++)
#ifdef _WIN32
      scanf_s("%lf", &Y(i, j));
#else
      scanf("%lf", &Y(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Count the number of specified elements (i.e., nonnegative) */
  nnz = 0;
  for (i = 1; i <= m; i++)
    for (j = 1; j <= n; j++)
      if (Y(i, j) >= 0.0)
        nnz++;

  /* There are as many variables as missing entries in the Y matrix
   *  plus two full symmetric matrices m x m and n x n. */
  nvar = m*(m+1)/2 + n*(n+1)/2 + m*n-nnz;
  if (!(x = NAG_ALLOC(nvar, double)) ||
      !(bl = NAG_ALLOC(nvar, double)) ||
      !(bu = NAG_ALLOC(nvar, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

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

  /* Create bounds for the missing entries in Y matrix to be between 0 and 1 */
  idxend = m*(m+1)/2+n*(n+1)/2;
  for (idx = 0; idx < idxend; idx++)
    {
      bl[idx] = -1e+20;
      bu[idx] = 1e+20;
    }
  for (; idx < nvar; idx++)
    {
      bl[idx] = 0.0;
      bu[idx] = 1.0;
    }

  /* nag_opt_handle_set_simplebounds (e04rhc).
   * Define bounds on the variables. */
  nag_opt_handle_set_simplebounds(handle, nvar, bl, bu, NAGERR_DEFAULT);

  /* Allocate space for the objective - minimize trace of the matrix
   * constraint. There is no quadratic part in the objective. */
  nnzc = m + n;
  if (!(idxc = NAG_ALLOC(nnzc, Integer)) ||
      !(c = NAG_ALLOC(nnzc, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Construct linear matrix inequality to request that
   *   [ X1, Y; Y', X2] is positive semidefinite. */

  /* How many nonzeros do we need? As many as number of variables
   * and the number of specified elements together. */
  nnzasum = m*(m+1)/2 + n*(n+1)/2 + m*n;
  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] = nnz;
for (i = 1; i <= nvar; i++)
  nnza[i] = 1;

/* Copy Y to the upper block of A_0 with different sign (because -A_0)
 * (upper triangle) */
idx = 0;
for (i = 1; i <= m; i++)
  for (j = 1; j <= n; j++)
    if (Y(i, j) >= 0.0)
      {
        irowa[idx] = i;
        icola[idx] = m + j;
        a[idx] = -Y(i, j);
        idx++;
      }
/* One matrix for each variable, A_i has just one nonzero - it binds
 * x_i with its position in the matrix constraint. Set also the objective. */
/* 1,1 - block, X1 matrix (mxm) */
idxobj = 0;
idxx = 1;
for (i = 1; i <= m; i++)
  {
    /* the next element is diagonal ==> part of the objective as a trace() */
    idxc[idxobj] = idxx;
    c[idxobj] = 1.0;
    idxobj++;
    for (j = i; j <= m; j++)
      {
        irowa[idx] = i;
        icola[idx] = j;
        a[idx] = 1.0;
        idx++;
        idxx++;
      }
  }
/* 2,2 - block, X2 matrix (nxn) */
for (i = 1; i <= n; i++)
  {
    /* the next element is diagonal ==> part of the objective as a trace() */
    idxc[idxobj] = idxx;
    c[idxobj] = 1.0;
    idxobj++;
    for (j = i; j <= n; j++)
      {
        irowa[idx] = m + i;
        icola[idx] = m + j;
        a[idx] = 1.0;
        idx++;
        idxx++;
      }
  }
/* 1,2 - block, missing element in Y we are after */
for (i = 1; i <= m; i++)
  for (j = 1; j <= n; j++)
    if (Y(i, j) < 0.0)
      {
        irowa[idx] = i;
        icola[idx] = m + j;
        a[idx] = 1.0;
        idx++;
      }
```

```
  /* nag_opt_handle_set_quadobj (e04rfc).
   * Add the sparse linear objective to the handle.*/
  nag_opt_handle_set_quadobj(handle, nnzc, idxc, c, 0, NULL, NULL, NULL,
                             NAGERR_DEFAULT);

  /* Just one matrix inequality of the dimension of the extended matrix. */
  nblk = 1;
  dima = m + 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);

  /* nag_opt_handle_opt_set (e04zmc).
   * Set optional arguments of the solver:
   * Completely turn off printing, allow timing and
   * turn on the monitor mode to stop every iteration. */
  nag_opt_handle_opt_set(handle, "Print File = -1", NAGERR_DEFAULT);
  nag_opt_handle_opt_set(handle, "Stats Time = Yes", NAGERR_DEFAULT);
  nag_opt_handle_opt_set(handle, "Monitor Frequency = 1", NAGERR_DEFAULT);
  nag_opt_handle_opt_set(handle, "Initial X = Automatic", NAGERR_DEFAULT);
  nag_opt_handle_opt_set(handle, "Dimacs = Check", NAGERR_DEFAULT);

  /* Pass the handle to the solver, we are not interested in
   * Lagrangian multipliers. */
  nnzu = 0;
  nnzuc = 0;
  nnzua = 0;
  while (1)
    {
      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 (inform == 1)
        {
          /* Monitor stop */
          printf("Monitor at iteration  %2" NAG_IFMT
                 ": objective %7.2f, avg.error %9.2e\n", (Integer) stats[0],
                 rinfo[0], (rinfo[1] + rinfo[2] + rinfo[3]) / 3.0);
          fflush(stdout);

          /* Check time limit and possibly stop the solver. */
          if (stats[7] > time_limit)
            inform = 0;
        }
      else
        {
          /* Final exit, solver finished. */
          printf("Finished at iteration %2" NAG_IFMT
                 ": objective %7.2f, avg.error %9.2e\n", (Integer) stats[0],
                 rinfo[0], (rinfo[1] + rinfo[2] + rinfo[3]) / 3.0);
          fflush(stdout);
          break;
        }
    }

  if (fail.code == NE_NOERROR || fail.code == NW_NOT_CONVERGED)
    {
      /* Successful run, fill the missing elements in the matrix Y. */
      idx = m*(m+1)/2 + n*(n+1)/2;
      for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++)
          if (Y(i, j) < 0.0)
            Y(i, j) = x[idx++];

      /* nag_gen_real_mat_print_comp (x04cbc).
```

```
       * Print the matrix. */
      nag_gen_real_mat_print_comp(Nag_ColMajor, Nag_GeneralMatrix,
                                  Nag_NonUnitDiag,
                                  m, n, y, m, "%7.1f", "Completed Matrix",
                                  Nag_IntegerLabels, NULL, Nag_IntegerLabels,
                                  NULL,
                                  80, 0, NULL, NAGERR_DEFAULT);

      /* Compute rank of the matrix via SVD, use the fact that the order
       * of the singular values is descending. */
      lwork = 20*(m > n?m:n);
      maxs = m < n?m:n;
      if (!(s = NAG_ALLOC(maxs, double)) ||
          !(work = NAG_ALLOC((lwork), double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }

      /* nag_dgesvd (f08kbc).
       * Compute the singular values */
      nag_dgesvd(Nag_ColMajor, Nag_NotU, Nag_NotVT, m, n, y, m, s,
                 rdummy, 1, rdummy, 1, work, NAGERR_DEFAULT);
      for (rank = 0; rank < maxs; rank++)
        if (s[rank] <= stol)
          break;
      printf("Rank is %" NAG_IFMT "\n", rank);
    }
  else if (fail.code == NE_USER_STOP)
    {
      printf("The given time limit was reached, run aborted.\n");
    }
  else
    {
      printf("Error from nag_opt_handle_solve_pennon (e04svc).\n%s\n",
             fail.message);
      exit_status = 1;
    }

 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(bl);
  NAG_FREE(bu);
  NAG_FREE(c);
  NAG_FREE(s);
  NAG_FREE(work);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(icola);
  NAG_FREE(irowa);
  NAG_FREE(nnza);
  NAG_FREE(idxc);
  return exit_status;
}
```

10.2 Program Data

```
nag_opt_handle_set_simplebounds (e04rhc) Example Program Data
15   6     :: m, n - number of respondents and questions
   -1.0   -1.0   -1.0   -1.0   -1.0    0.4
    0.6    0.4    0.8   -1.0   -1.0   -1.0
   -1.0   -1.0    0.8   -1.0    0.2   -1.0
    0.8    0.2   -1.0   -1.0   -1.0   -1.0
```

```
   -1.0    0.4   -1.0    0.0   -1.0    0.2
    0.4   -1.0   -1.0    0.2   -1.0    0.2
   -1.0    0.8    0.2    0.6   -1.0   -1.0
   -1.0   -1.0    0.2   -1.0   -1.0   -1.0
   -1.0    0.4   -1.0    0.6    0.0   -1.0
   -1.0   -1.0    0.4   -1.0   -1.0   -1.0
   -1.0   -1.0    0.2    0.2    0.4    0.4
   -1.0   -1.0   -1.0   -1.0    1.0    0.8
    1.0   -1.0    0.2   -1.0   -1.0    0.6
   -1.0   -1.0   -1.0   -1.0   -1.0    0.2
    0.6   -1.0    0.2    0.4   -1.0   -1.0  :: -1.0 for missing entries
```

10.3 Program Results

```
nag_opt_handle_set_simplebounds (e04rhc) Example Program Results

Monitor at iteration   0: objective    0.00, avg.error  3.14e+01
Monitor at iteration   1: objective  154.74, avg.error  4.98e+01
Monitor at iteration   2: objective   71.71, avg.error  2.15e+01
Monitor at iteration   3: objective   36.88, avg.error  9.13e+00
Monitor at iteration   4: objective   22.50, avg.error  3.84e+00
Monitor at iteration   5: objective   16.47, avg.error  1.61e+00
Monitor at iteration   6: objective   13.88, avg.error  6.87e-01
Monitor at iteration   7: objective   12.76, avg.error  2.97e-01
Monitor at iteration   8: objective   12.27, avg.error  1.29e-01
Monitor at iteration   9: objective   12.06, avg.error  5.63e-02
Monitor at iteration  10: objective   11.97, avg.error  2.50e-02
Monitor at iteration  11: objective   11.93, avg.error  1.17e-02
Monitor at iteration  12: objective   11.91, avg.error  5.77e-03
Monitor at iteration  13: objective   11.91, avg.error  3.33e-03
Monitor at iteration  14: objective   11.90, avg.error  9.11e-04
Monitor at iteration  15: objective   11.90, avg.error  3.77e-04
Monitor at iteration  16: objective   11.90, avg.error  1.64e-04
Monitor at iteration  17: objective   11.90, avg.error  7.07e-05
Monitor at iteration  18: objective   11.90, avg.error  3.05e-05
Monitor at iteration  19: objective   11.90, avg.error  1.31e-05
Monitor at iteration  20: objective   11.90, avg.error  5.60e-06
Monitor at iteration  21: objective   11.90, avg.error  2.38e-06
Monitor at iteration  22: objective   11.90, avg.error  1.01e-06
Finished at iteration 23: objective   11.90, avg.error  4.31e-07
 Completed Matrix
          1       2       3       4       5       6
  1     0.5     0.3     0.2     0.2     0.4     0.4
  2     0.6     0.4     0.8     0.2     0.3     0.4
  3     0.4     0.3     0.8     0.0     0.2     0.2
  4     0.8     0.2     0.3     0.4     0.3     0.4
  5     0.0     0.4     0.2     0.0     0.2     0.2
  6     0.4     0.1     0.2     0.2     0.1     0.2
  7     0.6     0.8     0.2     0.6     0.2     0.4
  8     0.1     0.1     0.2     0.0     0.0     0.1
  9     0.6     0.4     0.1     0.6     0.0     0.3
 10     0.2     0.1     0.4     0.0     0.1     0.1
 11     0.5     0.3     0.2     0.2     0.4     0.4
 12     0.7     0.4     0.3     0.0     1.0     0.8
 13     1.0     0.3     0.2     0.5     0.5     0.6
 14     0.2     0.1     0.1     0.1     0.2     0.2
 15     0.6     0.3     0.2     0.4     0.2     0.3
Rank is 4
```