NAG C Library Function Document

nag_opt_sparse_nlp_jacobian (e04vjc)

1 Purpose

nag_opt_sparse_nlp_jacobian (e04vjc) may be used before nag_opt_sparse_nlp_solve (e04vhc) to determine the sparsity pattern for the Jacobian.

2 Specification

```
#include <nag.h>
#include <nage04.h>

void nag_opt_sparse_nlp_jacobian (Integer nf, Integer n,
     void (*usrfun)(Integer *status, Integer n, const double x[], Integer needf,
          Integer nf, double f[], Integer needg, Integer leng, double g[],
          Nag_Comm *comm),
     Integer iafun[], Integer javar[], double a[], Integer lena, Integer *nea,
     Integer igfun[], Integer jgvar[], Integer leng, Integer *neg, const double x[],
     const double xlow[], const double xupp[], Nag_E04State *state,
     Nag_Comm *comm, NagError *fail)
```

3 Description

When using nag_opt_sparse_nlp_solve (e04vhc), if you set the optional argument **Derivative Option** $= 0$ and the user-supplied function **usrfun** provides none of the derivatives, you may need to call nag_opt_sparse_nlp_jacobian (e04vjc) to determine the input arrays **iafun**, **javar**, **a**, **igfun** and **jgvar**. These arrays define the pattern of non-zeros in the Jacobian matrix. A typical sequence of calls could be

```
e04vgc (&state, ... );
e04vjc (nf, n, ... );
e04vlc ("Derivative Option" = 0, &state, ... );
e04vhc (start, nf, ... );
```

nag_opt_sparse_nlp_jacobian (e04vjc) determines the sparsity pattern for the Jacobian and identifies the constant elements automatically. To do so, nag_opt_sparse_nlp_jacobian (e04vjc) approximates the problem functions, $F(x)$, at three random perturbations of the given initial point x. If an element of the approximate Jacobian is the same at all three points, then it is taken to be constant. If it is zero, it is taken to be identically zero. Since the random points are not chosen close together, the heuristic will correctly classify the Jacobian elements in the vast majority of cases. In general, nag_opt_sparse_nlp_jacobian (e04vjc) finds that the Jacobian can be permuted to the form:

$$\begin{pmatrix} G(x) & A_3 \\ A_2 & A_4 \end{pmatrix},$$

where A_2, A_3 and A_4 are constant. Note that $G(x)$ might contain elements that are also constant, but nag_opt_sparse_nlp_jacobian (e04vjc) must classify them as nonlinear. This is because nag_opt_sparse_nlp_solve (e04vhc) 'removes' linear variables from the calculation of F by setting them to zero before calling **usrfun**. A knowledgeable user would be able to move such elements from $F(x)$ in **usrfun** and enter them as part of **iafun**, **javar** and **a** for nag_opt_sparse_nlp_solve (e04vhc).

4 References

Hock W and Schittkowski K (1981) *Test Examples for Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical Systems* **187** Springer–Verlag

5 Arguments

Note: all optional arguments are described in detail in Section 11.2 of the document for nag_opt_sparse_nlp_solve (e04vhc).

1: **nf** – Integer *Input*

On entry: nf, the number of problem functions in $F(x)$, including the objective function (if any) and the linear and nonlinear constraints. Simple upper and lower bounds on x can be defined using the arguments **xlow** and **xupp** defined below and should not be included in F.

Constraint: **nf** > 0.

2: **n** – Integer *Input*

On entry: n, the number of variables.

Constraint: **n** > 0.

3: **usrfun** – function, supplied by the user *External Function*

usrfun must define the problem functions $F(x)$. This function is passed to nag_opt_sparse_nlp_jacobian (e04vjc) as the external argument **usrfun**.

Its specification is:

```
void usrfun (Integer *status, Integer n, const double x[], Integer needf,
     Integer nf, double f[], Integer needg, Integer leng, double g[],
     Nag_Comm *comm)
```

1: **status** – Integer * *Input/Output*

On entry: indicates the first call to **usrfun**.

status $= 0$

There is nothing special about the current call to **usrfun**.

status $= 1$

nag_opt_sparse_nlp_jacobian (e04vjc) is calling your function for the *first* time. Some data may need to be input or computed and saved.

On exit: may be used to indicate that you are unable to evaluate F at the current x. (For example, the problem functions may not be defined there).

nag_opt_sparse_nlp_jacobian (e04vjc) evaluates $F(x)$ at random perturbation of the initial point x, say x_p. If the functions cannot be evaluated at x_p, you can set **status** $= -1$, nag_opt_sparse_nlp_jacobian (e04vjc) will use another random perturbation.

If for some reason you wish to terminate the current problem, set **status** ≤ -2.

2: **n** – Integer *Input*

On entry: n, the number of variables, as defined in the call to nag_opt_sparse_nlp_jacobian (e04vjc).

3: **x**[**n**] – const double *Input*

On entry: the variables x at which the problem functions are to be calculated. The array x must not be altered.

4: **needf** – Integer *Input*

On entry: indicates if **f** must be assigned during the call to **usrfun** (see **f** below).

5: **nf** – Integer *Input*

On entry: nf, the number of problem functions.

6: **f**[**nf**] – double *Input/Output*

On entry: this will be set by nag_opt_sparse_nlp_jacobian (e04vjc).

On exit: the computed $F(x)$ according to the setting of **needf**.

If **needf** $= 0$, **f** is not required and is ignored.

If **needf** > 0, the components of $F(x)$ must be calculated and assigned to **f**. nag_opt_sparse_nlp_jacobian (e04vjc) will always call **usrfun** with **needf** > 0.

To simplify the code, you may ignore the value of **needf** and compute $F(x)$ on every entry to **usrfun**.

7: **needg** – Integer *Input*

On entry: nag_opt_sparse_nlp_jacobian (e04vjc) will call **usrfun** with **needg** $= 0$ to indicate that **g** is not required.

8: **leng** – Integer *Input*

On entry: the dimension of the array **g** as declared in the function from which nag_opt_sparse_nlp_jacobian (e04vjc) is called.

9: **g**[**leng**] – double

nag_opt_sparse_nlp_jacobian (e04vjc) will always call **usrfun** with **needg** $= 0$. **g** will not be used but must be declared correctly.

10: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type **Nag_Comm**; the following members are relevant to **usrfun**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_opt_sparse_nlp_jacobian (e04vjc) these pointers may be allocated memory by the user and initialized with various quantities for use by **usrfun** when called from nag_opt_sparse_nlp_jacobian (e04vjc).

4: **iafun**[**lena**] – Integer *Output*
5: **javar**[**lena**] – Integer *Output*
6: **a**[**lena**] – double *Output*

On exit: define the co-ordinates (i,j) and values A_{ij} of the non-zero elements of the linear part A of the function $F(x) = f(x) + Ax$.

In particular, the **nea** triples (**iafun**$[k-1]$, **javar**$[k-1]$, **a**$[k-1]$) define the row and column indices $i =$ **iafun**$[k-1]$ and $j =$ **javar**$[k-1]$ of the element $A_{ij} =$ **a**$[k-1]$.

7: **lena** – Integer *Input*

On entry: the dimension of the arrays **iafun**, **javar** and **a** that hold $\left(i,j,A_{ij}\right)$ as declared in the function from which nag_opt_sparse_nlp_jacobian (e04vjc) is called. **lena** should be an *overestimate* of the number of elements in the linear part of the Jacobian.

Constraint: **lena** ≥ 1.

8: **nea** – Integer * *Output*

On exit: is the number of non-zero entries in A such that $F(x) = f(x) + Ax$.

9: **igfun**[**leng**] – Integer *Output*
10: **jgvar**[**leng**] – Integer *Output*

On exit: define the co-ordinates (i,j) of the non-zero elements of G, the nonlinear part of the derivatives $J(x) = G(x) + A$ of the function $F(x) = f(x) + Ax$.

11: **leng** – Integer *Input*

On entry: the dimension of the arrays **igfun** and **jgvar** that define the varying Jacobian elements $\left(i,j,G_{ij}\right)$ as declared in the function from which nag_opt_sparse_nlp_jacobian (e04vjc) is called. **leng** should be an *overestimate* of the number of elements in the nonlinear part of the Jacobian.

Constraint: **leng** ≥ 1.

12: **neg** – Integer * *Output*

On exit: the number of non-zero entries in G.

13: **x**[**n**] – const double *Input*

On entry: an initial estimate of the variables x. The contents of x will be used by nag_opt_sparse_nlp_jacobian (e04vjc) in the call of **usrfun**, and so each element of **x** should be within the bounds given by **xlow** and **xupp**.

14: **xlow**[**n**] – const double *Input*
15: **xupp**[**n**] – const double *Input*

On entry: contain the lower and upper bounds l_x and u_x on the variables x.

To specify a non-existent lower bound $[l_x]_j = -\infty$, set **xlow**$(j) \leq -bigbnd$, where $bigbnd$ is the **Infinite Bound Size**. To specify a non-existent upper bound **xupp**$(j) \geq bigbnd$.

To fix the jth variable (say, $x_j = \beta$, where $|\beta| < bigbnd$), set **xlow**$(j) =$ **xupp**$(j) = \beta$.

16: **state** – Nag_E04State * *Communication Structure*

Note: **state** is a NAG defined type (see Section 2.2.1.1 of the Essential Introduction).

state contains internal information required for functions in this suite. It must not be modified in any way.

17: **comm** – Nag_Comm * *Communication Structure*

The NAG communication argument (see Section 2.2.1.1 of the Essential Introduction).

18: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Internal error: memory allocation failed when attempting to allocate workspace sizes ⟨*value*⟩, ⟨*value*⟩ and ⟨*value*⟩.

NE_ALLOC_INSUFFICIENT

Internal memory allocation was insufficient. Please contact NAG.

NE_ARRAY_TOO_SMALL

Either ⟨*value*⟩ or ⟨*value*⟩ is too small. Increase both of them and corresponding array sizes. ⟨*value*⟩ = ⟨*value*⟩, ⟨*value*⟩ = ⟨*value*⟩.

NE_BAD_PARAM

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

NE_E04VGC_NOT_INIT

Initialization function nag_opt_sparse_nlp_init (e04vgc) has not been called.

NE_INT

On entry, **lena** < ⟨*value*⟩: **lena** = ⟨*value*⟩.

On entry, **leng** < ⟨*value*⟩: **leng** = ⟨*value*⟩.

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 consult NAG for assistance.

NE_JACOBIAN_STRUCTURE_FAIL

Cannot estimate Jacobian structure at given point **x**

NE_USER_STOP

User-supplied function **usrfun** requested termination.

NE_USRFUN_UNDEFINED

User-supplied function **usrfun** indicates that functions are undefined near given point **x**

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

This example shows how to call nag_opt_sparse_nlp_jacobian (e04vjc) to determine the sparsity pattern of the Jacobian before calling nag_opt_sparse_nlp_jacobian (e04vjc) to solve a sparse nonlinear programming problem without providing the Jacobian information in **usrfun**.

This is a reformulation of Problem 74 from Hock and Schittkowski (1981) and involves the minimization of the nonlinear function

$$f(x) = 10^{-6}x_3^3 + \tfrac{2}{3} \times 10^{-6}x_4^3 + 3x_3 + 2x_4$$

subject to the bounds

$$\begin{aligned} -0.55 \leq x_1 \leq 0.55, \\ -0.55 \leq x_2 \leq 0.55, \\ 0 \leq x_3 \leq 1200, \\ 0 \leq x_4 \leq 1200, \end{aligned}$$

to the nonlinear constraints

$$\begin{array}{lcll} 1000\sin(-x_1-0.25) & + & 1000\sin(-x_2-0.25)-x_3 & = & -894.8, \\ 1000\sin(x_1-0.25) & + & 1000\sin(x_1-x_2-0.25)-x_4 & = & -894.8, \\ 1000\sin(x_2-0.25) & + & 1000\sin(x_2-x_1-0.25) & = & -1294.8, \end{array}$$

and to the linear constraints

$$\begin{aligned} -x_1+x_2 &\geq -0.55, \\ x_1-x_2 &\geq -0.55. \end{aligned}$$

The initial point, which is infeasible, is

$$x_0 = (\,0, \quad 0, \quad 0, \quad 0\,)^{\mathrm{T}},$$

and $f(x_0) = 0$.

The optimal solution (to five figures) is

$$x^* = (0.11887, -0.39623, 679.94, 1026.0)^{\mathrm{T}},$$

and $f(x^*) = 5126.4$. All the nonlinear constraints are active at the solution.

The formulation of the problem combines the constraints and the objective into a single vector (F).

$$F = \begin{pmatrix} 1000\sin(-x_1-0.25) & + & 1000\sin(-x_2-0.25)-x_3 \\ 1000\sin(x_1-0.25) & + & 1000\sin(x_1-x_2-0.25)-x_4 \\ 1000\sin(x_2-0.25) & + & 1000\sin(x_2-x_1-0.25) \\ & -x_1+x_2 & \\ & x_1-x_2 & \\ & 10^{-6}x_3^3+\frac{2}{3}\times 10^{-6}x_4^3+3x_3+2x_4 & \end{pmatrix}$$

9.1 Program Text

```
/* nag_nag_opt_sparse_nlp_jacobian (e04vjc) Example Program.
 *
 * Copyright 2004 Numerical Algorithms Group.
 *
 * Mark 8, 2004.
 */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagx04.h>

static void usrfun(Integer *status, Integer n, const double x[],
                   Integer needf, Integer nf, double f[], Integer needg,
                   Integer leng, double g[], Nag_Comm *comm);
int main(void)
{

  /* Scalars */
  double objadd, sinf;
  Integer exit_status, i, lena, leng, n, nea, neg, nf, nfname, ninf, ns, nxname;
  Integer objrow, start_int;

  /* Arrays */
  char **fnames=0, prob[9], **xnames=0;
  double *a=0, *f=0, *flow=0, *fmul=0, *fupp=0, *x=0, *xlow=0, *xmul=0, *xupp=0;
  Integer *fstate=0, *iafun=0, *igfun=0, *javar=0, *jgvar=0, *xstate=0;

  /* Nag Types*/
  Nag_E04State state;
  NagError fail;
  Nag_Comm comm;
  Nag_Start start;
  Nag_FileID fileid;
```

```
  exit_status = 0;
  INIT_FAIL(fail);
  Vprintf("nag_opt_sparse_nlp_jacobian (e04vjc) Example Program Results\n");

  /* Skip heading in data file */
  Vscanf("%*[^\n] ");

  Vscanf("%ld%ld%*[^\n] ",   &n,  &nf);
  if (n > 0 && nf > 0)
    {
      nfname = 1;
      nxname = 1;
      lena = 300;
      leng = 300;
      /* Allocate memory */
      if ( !(fnames = NAG_ALLOC(nfname, char *)) ||
           !(xnames = NAG_ALLOC(nxname, char *)) ||
           !(a = NAG_ALLOC(lena, double)) ||
           !(f = NAG_ALLOC(nf, double)) ||
           !(flow = NAG_ALLOC(nf, double)) ||
           !(fmul = NAG_ALLOC(nf, double)) ||
           !(fupp = NAG_ALLOC(nf, double)) ||
           !(x = NAG_ALLOC(n, double)) ||
           !(xlow = NAG_ALLOC(n, double)) ||
           !(xmul = NAG_ALLOC(n, double)) ||
           !(xupp = NAG_ALLOC(n, double)) ||
           !(fstate = NAG_ALLOC(nf, Integer)) ||
           !(iafun = NAG_ALLOC(lena, Integer)) ||
           !(igfun = NAG_ALLOC(leng, Integer)) ||
           !(javar = NAG_ALLOC(lena, Integer)) ||
           !(jgvar = NAG_ALLOC(leng, Integer)) ||
           !(xstate = NAG_ALLOC(n, Integer)) )
        {
          Vprintf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      Vprintf("Invalid n or nf\n");
      exit_status = 1;
      goto END;
    }

  /* Call nag_opt_sparse_nlp_init (e04vgc) to initialise e04vjc. */
  /* nag_opt_sparse_nlp_init (e04vgc).
   * Initialization function for nag_opt_sparse_nlp_solve
   * (e04vhc)
   */
  nag_opt_sparse_nlp_init(&state, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Initialisation of nag_opt_sparse_nlp_init (e04vgc) failed.\n");
      exit_status = 1;
      goto END;
    }

  /* Read the bounds on the variables. */
  for (i = 1; i <= n; ++i)
    {
      Vscanf("%lf%lf%*[^\n] ",   &xlow[i - 1],  &xupp[i - 1]);
    }

  for (i = 1; i <= n; ++i)
    {
      x[i - 1] = 0.;
    }

  /* Determine the Jacobian structure. */
  /* nag_opt_sparse_nlp_jacobian (e04vjc).
```

```
   * Determine the pattern of nonzeros in the Jacobian matrix
   * for nag_opt_sparse_nlp_solve (e04vhc)
   */
  nag_opt_sparse_nlp_jacobian(nf, n, usrfun, iafun, javar, a, lena, &nea, igfun,
                              jgvar, leng, &neg, x, xlow, xupp, &state, &comm,
                              &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("nag_opt_sparse_nlp_jacobian (e04vjc) failed to determine the"
              " Jacobian structure\n");
      exit_status = 1;
      goto END;
    }

  /* Print the Jacobian structure. */

  Vprintf("\n");
  Vprintf("NEA (the number of non-zero entries in A) = %3ld\n",   nea);

  Vprintf("  I     IAFUN(I)   JAVAR(I)          A(I)\n");
  Vprintf("----    --------   --------   -----------\n");

  for (i = 1; i <= nea; ++i)
    {
      Vprintf("%3ld%10ld%10ld%18.4e\n", i, iafun[i - 1],
              javar[i - 1], a[i - 1]);
    }

  Vprintf("\n");
  Vprintf("NEG (the number of non-zero entries in G) = %3ld\n",   neg);
  Vprintf("  I     IGFUN(I)   JGVAR(I)\n");
  Vprintf("----    --------   --------\n");

  for (i = 1; i <= neg; ++i)
    {
      Vprintf("%3ld%10ld%10ld\n", i, igfun[i - 1],
              jgvar[i - 1]);
    }

  /* Now that we have the determined the structure of the
   * Jacobian, set up the information necessary to solve
   * the optimization problem.
   */
  start_int = 0;
  if (start_int == 0)
    {
      start = Nag_Cold;
    }
  else
    {
      start = Nag_Warm;
    }
  strcpy(prob, "e04vjce");
  objadd = 0.0;
  for (i = 1; i <= n; ++i)
    {
      x[i - 1] = 0.;
      xstate[i - 1] = 0;
      xmul[i - 1] = 0.;
    }
  for (i = 1; i <= nf; ++i)
    {
      f[i - 1] = 0.;
      fstate[i - 1] = 0;
      fmul[i - 1] = 0.;
    }

  /* The row containing the objective function. */
  Vscanf("%ld%*[^\n] ",   &objrow);

  /* Read the bounds on the functions. */
```

```
  for (i = 1; i <= nf; ++i)
    {
      Vscanf("%lf%lf%*[^\n] ",   &flow[i - 1],  &fupp[i - 1]);
    }

  /* By default nag_opt_sparse_nlp_solve (e04vhc) does not print monitoring
   * information. Call nag_open_file (x04acc) to set the print file fileid
   */
  /* nag_open_file (x04acc).
   * Open unit number for reading, writing or appending, and
   * associate unit with named file
   */
  nag_open_file("", 2, &fileid, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Fileid could not be obtained.\n");
      exit_status = 1;
      goto END;
    }
  /* nag_opt_sparse_nlp_option_set_integer (e04vmc).
   * Set a single option for nag_opt_sparse_nlp_solve (e04vhc)
   * from an integer argument
   */
  nag_opt_sparse_nlp_option_set_integer("Print file", fileid, &state, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("nag_opt_sparse_nlp_option_set_integer (e04vmc) failed to set"
              " Print File\n");
      exit_status = 1;
      goto END;
    }

  /* Tell nag_opt_sparse_nlp_solve (e04vhc) that we supply no derivatives in
   *  usrfun. */
  /* nag_opt_sparse_nlp_option_set_string (e04vlc).
   * Set a single option for nag_opt_sparse_nlp_solve (e04vhc)
   * from a character string
   */
  nag_opt_sparse_nlp_option_set_string("Derivative option 0", &state, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("nag_opt_sparse_nlp_option_set_string (e04vlc) failed to set"
              " Derivative option\n");
      exit_status = 1;
      goto END;
    }
  for (i = 1; i <= nfname; ++i)
    {
      fnames[i -1] = NAG_ALLOC(9, char);
      strcpy(fnames[i-1],"");
    }

  for (i = 1; i <= nxname; ++i)
    {
      xnames[i-1] = NAG_ALLOC(9, char);
      strcpy(xnames[i-1],"");
    }

  /* Solve the problem. */
  /* nag_opt_sparse_nlp_solve (e04vhc).
   * General sparse nonlinear optimizer
   */
  nag_opt_sparse_nlp_solve(start, nf, n, nxname, nfname, objadd, objrow, prob,
                           usrfun, iafun, javar, a, lena, nea, igfun, jgvar,
                           leng, neg, xlow, xupp, xnames, flow, fupp, fnames, x,
                           xstate, xmul, f, fstate, fmul, &ns, &ninf, &sinf,
                           &state, &comm, &fail);

  if (fail.code == NE_NOERROR || fail.code == NW_NOT_FEASIBLE)
    {
      Vprintf("Final objective value = %11.1f\n",   f[objrow - 1]);
```

```
      Vprintf("Optimal X = ");

      for (i = 1; i <= n; ++i)
        {
          Vprintf("%9.2f%s", x[i - 1], i%7 == 0 || i == n ?"\n":" ");
        }
    }
  else
    {
      Vprintf ("Error message from nag_opt_sparse_nlp_solve (e04vhc) %s\n",
               fail.message);
    }

 END:
  for (i=0; i < nxname; i++)
    {
      NAG_FREE(xnames[i]);
    }
  for (i=0; i < nfname; i++)
    {
      NAG_FREE(fnames[i]);
    }
  if (fnames) NAG_FREE(fnames);
  if (xnames) NAG_FREE(xnames);
  if (a) NAG_FREE(a);
  if (f) NAG_FREE(f);
  if (flow) NAG_FREE(flow);
  if (fmul) NAG_FREE(fmul);
  if (fupp) NAG_FREE(fupp);
  if (x) NAG_FREE(x);
  if (xlow) NAG_FREE(xlow);
  if (xmul) NAG_FREE(xmul);
  if (xupp) NAG_FREE(xupp);
  if (fstate) NAG_FREE(fstate);
  if (iafun) NAG_FREE(iafun);
  if (igfun) NAG_FREE(igfun);
  if (javar) NAG_FREE(javar);
  if (jgvar) NAG_FREE(jgvar);
  if (xstate) NAG_FREE(xstate);



  return exit_status;
}

static void usrfun(Integer *status, Integer n, const double x[],
                   Integer needf, Integer nf, double f[], Integer needg,
                   Integer leng, double g[], Nag_Comm *comm)
{
  /* Parameter adjustments */
#define X(I) x[(I)-1]
#define F(I) f[(I)-1]
#define G(I) g[(I)-1]

  /* Function Body */
  if (needf > 0)
    {
      F(1) = sin(-X(1) - .25) * 1e3 + sin(-X(2) - .25) * 1e3 - X(3);
      F(2) = sin(X(1) - .25) * 1e3 + sin(X(1) - X(2) - .25) * 1e3 - X(4);
      F(3) = sin(X(2) - X(1) - .25) * 1e3 + sin(X(2) - .25) * 1e3;
      F(4) = -X(1) + X(2);
      F(5) = X(1) - X(2);
      F(6) = X(3) * (X(3) * X(3)) * 1e-6 + X(4) * (X(4) * X(4)) * 2e-6 / 3.
        + X(3) * 3 + X(4) * 2;
    }

  return;
} /* usrfun */
```

9.2 Program Data

```
nag_opt_sparse_nlp_jacobian (e04vjc) Example Program Data
 4   6              : Values of N and NF
-0.55E0    0.55e0  : Bounds on the variables, XLOW(i), XUPP(i), for i = 1 to N
-0.55E0    0.55E0
 0.0E0   1200.0E0
 0.0E0   1200.0E0

 6                 : Value of OBJROW
 -894.8E0 -894.8E0 : Bounds on the functions, FLOW(i), FUPP(i), for i = 1 to NF
 -894.8E0 -894.8E0
-1294.8E0 -1294.8E0
-0.55E0    1.0E25
-0.55E0    1.0E25
-1.0E25    1.0E25
```

9.3 Program Results

```
nag_opt_sparse_nlp_jacobian (e04vjc) Example Program Results

NEA (the number of non-zero entries in A) =   4
  I     IAFUN(I)   JAVAR(I)          A(I)
----    --------   --------   -----------
  1         4         1      -1.0000e+00
  2         5         1       1.0000e+00
  3         4         2       1.0000e+00
  4         5         2      -1.0000e+00

NEG (the number of non-zero entries in G) =  10
  I     IGFUN(I)   JGVAR(I)
----    --------   --------
  1         1         1
  2         2         1
  3         3         1
  4         1         2
  5         2         2
  6         3         2
  7         6         3
  8         6         4
  9         1         3
 10         2         4

 Parameters
 ==========

 Files
 -----
 Solution file.....         0      Old basis file .....         0      (Print file)...........         6
 Insert file.......         0      New basis file .....         0      (Summary file).........         0
 Punch file........         0      Backup basis file...         0
 Load file.........         0      Dump file...........         0

 Frequencies
 -----------
 Print frequency.....       100      Check frequency.....        60      Save new basis map.....       100
 Summary frequency...       100      Factorization frequency     50      Expand frequency.......     10000

 QP subproblems
 --------------
 QPsolver Cholesky......
 Scale tolerance.....     0.900      Minor feasibility tol..  1.00E-06      Iteration limit........     10000
 Scale option........         0      Minor optimality  tol..  1.00E-06      Minor print level......         1
 Crash tolerance.....     0.100      Pivot tolerance.....  2.05E-11      Partial price..........         1
 Crash option........         3      Elastic weight......  1.00E+04      Prtl price section ( A)         4
                                     New superbasics.....        99      Prtl price section (-I)         5

 The SQP Method
 --------------
 Minimize............               Cold start..........               Proximal Point method..         1
```

```
Nonlinear objectiv vars         2       Objective Row.....          6       Function precision.....  1.72E-13
Unbounded step size....  1.00E+20       Superbasics limit......     4       Difference interval....  4.15E-07
Unbounded objective....  1.00E+15       Reduced Hessian dim....     4       Central difference int.  5.57E-05
Major step limit.......  2.00E+00       Nonderiv.  linesearch..             Derivative option......         0
Major iterations limit.      1000       Linesearch tolerance...   0.90000   Verify level...........         0
Minor iterations limit.       500       Penalty parameter......  0.00E+00   Major Print Level......         1
                                        Major optimality tol...  2.00E-06

Hessian Approximation
---------------------
Full-Memory Hessian....                 Hessian updates........  99999999   Hessian frequency......  99999999
                                                                           Hessian flush..........  99999999

Nonlinear constraints
---------------------
Nonlinear constraints..         3       Major feasibility tol..  1.00E-06   Violation limit........  1.00E+06
Nonlinear Jacobian vars         4

Miscellaneous
-------------
LU factor tolerance....      3.99       LU singularity tol.....  2.05E-11   Timing level...........         0
LU update tolerance....      3.99       LU swap tolerance......  1.03E-04   Debug level............         0
LU partial  pivoting...                 eps (machine precision)  1.11E-16   System information.....        No


Nonlinear constraints       3     Linear constraints       2
Nonlinear variables         4     Linear variables         0
Jacobian  variables         4     Objective variables      2
Total constraints           5     Total variables          4


The user has defined       0   out of      10   first  derivatives


  Itns Major Minors    Step   nCon Feasible  Optimal  MeritFunction     L+U BSwap     nS  condHz Penalty
     3     0      3             1  8.0E+02  1.0E-00  0.0000000E+00      14            1 3.0E+07         _  r
     4     1      1 1.2E-03     2  4.0E+02  1.0E-00  1.7331708E+06      13            1 1.3E+07 5.1E+00 _n rl
     5     2      1 1.3E-03     3  2.7E+02  5.5E-01  1.7301151E+06      13                      5.1E+00 _s  l
     5     3      0 7.5E-03     4  8.8E+01  5.4E-01  8.8193381E+05      13                      2.8E+00 _   l
     5     4      0 2.3E-02     5  2.9E+01  5.3E-01  8.4262004E+05      13                      2.8E+00 _   l
     5     5      0 6.9E-02     6  8.9E+00  5.2E-01  7.3075567E+05      13                      2.8E+00 _   l
     6     6      1 2.2E-01     7  2.3E+00  8.0E+01  4.4817382E+05      13            1 1.2E+04 2.8E+00 _   l
     7     7      1 8.3E-01     8  1.7E-01  9.2E+00  2.4330986E+04      13            1 9.5E+03 2.8E+00 _   l
     8     8      1 1.0E+00     9  6.5E-03  4.0E+01  5.3126065E+03      13     1      1 1.3E+02 2.8E+00 _
     9     9      1 1.0E+00    10  4.6E-03  1.2E+01  5.1602362E+03      13            1 9.4E+01 2.8E+00 _
    10    10      1 1.0E+00    11  2.3E-04  6.1E-02  5.1265654E+03      13            1 9.6E+01 2.8E+00 _
    11    11      1 1.0E+00    12 (1.3E-08) 2.9E-04  5.1264981E+03      13            1 1.2E+02 2.8E+00 _      c
    12    11      2 1.0E+00    12 (1.3E-08) 2.7E-04  5.1264981E+03      13            1 1.2E+02 2.8E+00 _      c
    13    12      1 1.0E+00    13 (5.6E-13) 7.1E-05  5.1264981E+03      13            1 9.5E+01 2.8E+00 _      c
    14    13      1 1.0E+00    14 (1.8E-14)(5.8E-11) 5.1264981E+03      13            1 9.5E+01 2.8E+00 _      c

E04VHF EXIT   0 -- finished successfully
E04VHF INFO   1 -- optimality conditions satisfied

Problem name                 e04vjce
No. of iterations                 14   Objective value      5.1264981096E+03
No. of major iterations           13   Linear objective     0.0000000000E+00
Penalty parameter           2.780E+00   Nonlinear objective  5.1264981096E+03
No. of calls to funobj           106   No. of calls to funcon           106
Calls with modes 1,2 (known g)    14   Calls with modes 1,2 (known g)    14
Calls for forward differencing    48   Calls for forward differencing    48
Calls for central differencing    24   Calls for central differencing    24
No. of superbasics                 1   No. of basic nonlinears            3
No. of degenerate steps            0   Percentage                      0.00
Max x                     2 1.0E+03   Max pi                     3 5.5E+00
Max Primal infeas         0 0.0E+00   Max Dual infeas            1 5.6E-10
Nonlinear constraint violn    1.5E-11

Name          e04vjce                 Objective Value      5.1264981096E+03
```

```
 Status         Optimal Soln             Iteration     14    Superbasics     1

 Objective               (Min)
 RHS
 Ranges
 Bounds

 Section 1 - Rows

  Number  ...Row.. State  ...Activity...  Slack Activity  ..Lower Limit.  ..Upper Limit.  .Dual Activity    ..i

       5  r       1    EQ      -894.80000         0.00000      -894.80000      -894.80000        -4.38698       1
       6  r       2    EQ      -894.80000         0.00000      -894.80000      -894.80000        -4.10563       2
       7  r       3    EQ     -1294.80000         0.00000     -1294.80000     -1294.80000        -5.46328       3
       8  r       4    BS        -0.51511         0.03489        -0.55000            None               .       4
       9  r       5    BS         0.51511         1.06511        -0.55000            None               .       5

 Section 2 - Columns

  Number  .Column. State  ...Activity...  .Obj Gradient.  ..Lower Limit.  ..Upper Limit.  Reduced Gradnt    m+j

       1  x       1    BS         0.11888          .             -0.55000         0.55000         0.00000       6
       2  x       2    BS        -0.39623          .             -0.55000         0.55000         0.00000       7
       3  x       3   SBS       679.94532         4.38698          .           1200.00000        -0.00000       8
       4  x       4    BS      1026.06713         4.10563          .           1200.00000         0.00000       9
Final objective value =      5126.5
Optimal X =      0.12     -0.40    679.95   1026.07
```