NAG Library Routine Document

E04VJF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

E04VJF may be used before E04VHF to determine the sparsity pattern for the Jacobian.

2 Specification

```
SUBROUTINE E04VJF (NF, N, USRFUN, IAFUN, JAVAR, A, LENA, NEA, IGFUN,        &
                   JGVAR, LENG, NEG, X, XLOW, XUPP, CW, LENCW, IW,          &
                   LENIW, RW, LENRW, CUSER, IUSER, RUSER, IFAIL)
INTEGER            NF, N, IAFUN(LENA), JAVAR(LENA), LENA, NEA,              &
                   IGFUN(LENG), JGVAR(LENG), LENG, NEG, LENCW,              &
                   IW(LENIW), LENIW, LENRW, IUSER(*), IFAIL
REAL (KIND=nag_wp) A(LENA), X(N), XLOW(N), XUPP(N), RW(LENRW),              &
                   RUSER(*)
CHARACTER(8)       CW(LENCW), CUSER(*)
EXTERNAL           USRFUN
```

3 Description

When using E04VHF, if you set the optional parameter **Derivative Option** $= 0$ and USRFUN provides none of the derivatives, you may need to call E04VJF to determine the input arrays IAFUN, JAVAR, A, IGFUN and JGVAR. These arrays define the pattern of nonzeros in the Jacobian matrix. A typical sequence of calls could be

```
CALL E04VGF (CW, LENCW, ... )
 CALL E04VJF (NF, N, ... )
 CALL E04VLF ('Derivative Option = 0', CW, ... )
 CALL E04VHF (START, NF, ... )
```

E04VJF determines the sparsity pattern for the Jacobian and identifies the constant elements automatically. To do so, E04VJF 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, E04VJF 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 E04VJF must classify them as nonlinear. This is because E04VHF '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 E04VHF.

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 parameters are described in detail in Section 12.1 in E04VHF.

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 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 – SUBROUTINE, supplied by the user. *External Procedure*

USRFUN must define the problem functions $F(x)$. This subroutine is passed to E04VJF as the external argument USRFUN.

The specification of USRFUN is:

```
SUBROUTINE USRFUN (STATUS, N, X, NEEDF, NF, F, NEEDG, LENG, G,       &
                   CUSER, IUSER, RUSER)

INTEGER            STATUS, N, NEEDF, NF, NEEDG, LENG, IUSER(*)
REAL (KIND=nag_wp) X(N), F(NF), G(LENG), RUSER(*)
CHARACTER(8)       CUSER(*)
```

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 $= 1$
E04VJF is calling your subroutine 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).

E04VJF 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 $= -1$, E04VJF 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 E04VJF.

3: X(N) – REAL (KIND=nag_wp) array *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).

5: NF – INTEGER *Input*

On entry: nf, the number of problem functions.

6: F(NF) – REAL (KIND=nag_wp) array *Input/Output*

On entry: this will be set by E04VJF.

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. E04VJF 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: E04VJF 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 (sub)program from which E04VJF is called.

9: G(LENG) – REAL (KIND=nag_wp) array *Input/Output*

On entry: concerns the calculations of the derivatives of the function $f(x)$.

On exit: E04VJF will always call USRFUN with NEEDG $= 0$: G is not required to be set on exit but must be declared correctly.

10: CUSER($*$) – CHARACTER(8) array *User Workspace*
11: IUSER($*$) – INTEGER array *User Workspace*
12: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

USRFUN is called with the arguments CUSER, IUSER and RUSER as supplied to E04VJF. You should use the arrays CUSER, IUSER and RUSER to supply information to USRFUN.

USRFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which E04VJF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

4: IAFUN(LENA) – INTEGER array *Output*
5: JAVAR(LENA) – INTEGER array *Output*
6: A(LENA) – REAL (KIND=nag_wp) array *Output*

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

In particular, NEA triples $(\mathrm{IAFUN}(k), \mathrm{JAVAR}(k), \mathrm{A}(k))$ define the row and column indices $i = \mathrm{IAFUN}(k)$ and $j = \mathrm{JAVAR}(k)$ of the element $A_{ij} = \mathrm{A}(k)$.

7: LENA – INTEGER *Input*

On entry: the dimension of the arrays IAFUN, JAVAR and A that hold (i, j, A_{ij}) as declared in the (sub)program from which E04VJF 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 nonzero entries in A such that $F(x) = f(x) + Ax$.

9: IGFUN(LENG) – INTEGER array *Output*

10: JGVAR(LENG) – INTEGER array *Output*

On exit: define the coordinates (i, j) of the nonzero 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 (i, j, G_{ij}) as declared in the (sub)program from which E04VJF 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 nonzero entries in G.

13: X(N) – REAL (KIND=nag_wp) array *Input*

On entry: an initial estimate of the variables x. The contents of x will be used by E04VJF in the call of USRFUN, and so each element of X should be within the bounds given by XLOW and XUPP.

14: XLOW(N) – REAL (KIND=nag_wp) array *Input*

15: XUPP(N) – REAL (KIND=nag_wp) array *Input*

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

To specify a nonexistent lower bound $[l_x]_j = -\infty$, set XLOW(j) $\leq -bigbnd$, where $bigbnd$ is the optional parameter **Infinite Bound Size**. To specify a nonexistent upper bound XUPP(j) $\geq bigbnd$.

To fix the jth variable (say, $x_j = \beta$, where $|\beta| < bigbnd$), set XLOW(j) = XUPP(j) = β.

16: CW(LENCW) – CHARACTER(8) array *Communication Array*

17: LENCW – INTEGER *Input*

On entry: the dimension of the array CW as declared in the (sub)program from which E04VJF is called.

Constraint: LENCW ≥ 600.

18: IW(LENIW) – INTEGER array *Communication Array*

19: LENIW – INTEGER *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which E04VJF is called.

Constraint: LENIW ≥ 600.

20: RW(LENRW) – REAL (KIND=nag_wp) array *Communication Array*

21: LENRW – INTEGER *Input*

On entry: the dimension of the array RW as declared in the (sub)program from which E04VJF is called.

Constraint: LENRW ≥ 600.

22: CUSER($*$) – CHARACTER(8) array *User Workspace*
23: IUSER($*$) – INTEGER array *User Workspace*
24: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

CUSER, IUSER and RUSER are not used by E04VJF, but are passed directly to USRFUN and should be used to pass information to this routine.

25: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this argument, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, LENCW $= \langle value \rangle$.
Constraint: LENCW ≥ 600.

On entry, LENIW $= \langle value \rangle$.
Constraint: LENIW ≥ 600.

On entry, LENRW $= \langle value \rangle$.
Constraint: LENRW ≥ 600.

The initialization routine E04VGF has not been called.

IFAIL $= 2$

On entry, LENA $= \langle value \rangle$.
Constraint: LENA ≥ 1.

On entry, LENG $= \langle value \rangle$.
Constraint: LENG ≥ 1.

IFAIL $= 3$

User-supplied routine USRFUN indicates that functions are undefined near given point X.

You have indicated that the problem functions are undefined by setting STATUS $= -1$ on exit from USRFUN. This exit occurs if E04VJF is unable to find a point at which the functions are defined.

IFAIL $= 4$

User-supplied routine USRFUN requested termination.

You have indicated the wish to terminate the call to E04VJF by setting STATUS to a value < -1 on exit from USRFUN.

IFAIL $= 5$

Either LENA or LENG is too small. Increase both of them and corresponding array sizes. LENA $= \langle value \rangle$ and LENG $= \langle value \rangle$.

IFAIL $= 6$

Cannot estimate Jacobian structure at given point X.

IFAIL $= 7$

Internal error: memory allocation failed when attempting to allocate workspace sizes $\langle value \rangle$, $\langle value \rangle$ and $\langle value \rangle$. Please contact NAG.

IFAIL $= 8$

Internal memory allocation was insufficient. Please contact NAG.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

E04VJF makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

This example shows how to call E04VJF to determine the sparsity pattern of the Jacobian before calling E04VHF to solve a sparse nonlinear programming problem without providing the Jacobian information in USRFUN.

It 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

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

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

$$x_0 = \begin{pmatrix} 0, & 0, & 0, & 0 \end{pmatrix}^{\mathrm{T}},$$

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).

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

10.1 Program Text

```
!   E04VJF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module e04vjfe_mod

!     E04VJF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                           :: usrfun
!     .. Parameters ..
      Integer, Parameter, Public       :: lencw = 600, leniw = 600,           &
                                          lenrw = 600, nin = 5, nout = 6
    Contains
      Subroutine usrfun(status,n,x,needf,nf,f,needg,leng,g,cuser,iuser,ruser)

!       .. Scalar Arguments ..
        Integer, Intent (In)           :: leng, n, needf, needg, nf
        Integer, Intent (Inout)        :: status
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout) :: f(nf), g(leng), ruser(*)
        Real (Kind=nag_wp), Intent (In) :: x(n)
        Integer, Intent (Inout)        :: iuser(*)
        Character (8), Intent (Inout)  :: cuser(*)
!       .. Intrinsic Procedures ..
        Intrinsic                      :: sin
!       .. Executable Statements ..
        If (needf>0) Then
          f(1) = 1000.0E+0_nag_wp*sin(-x(1)-0.25E+0_nag_wp) +                &
            1000.0E+0_nag_wp*sin(-x(2)-0.25E+0_nag_wp) - x(3)
          f(2) = 1000.0E+0_nag_wp*sin(x(1)-0.25E+0_nag_wp) +                 &
            1000.0E+0_nag_wp*sin(x(1)-x(2)-0.25E+0_nag_wp) - x(4)
          f(3) = 1000.0E+0_nag_wp*sin(x(2)-x(1)-0.25E+0_nag_wp) +            &
            1000.0E+0_nag_wp*sin(x(2)-0.25E+0_nag_wp)
          f(4) = -x(1) + x(2)
          f(5) = x(1) - x(2)
          f(6) = 1.0E-6_nag_wp*x(3)**3 + 2.0E-6_nag_wp*x(4)**3/3.0E+0_nag_wp + &
            3.0E0_nag_wp*x(3) + 2.0E0_nag_wp*x(4)
        End If

        Return

      End Subroutine usrfun
    End Module e04vjfe_mod
    Program e04vjfe

!     E04VJF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: e04vgf, e04vhf, e04vjf, e04vlf, e04vmf, nag_wp
      Use e04vjfe_mod, Only: lencw, leniw, lenrw, nin, nout, usrfun
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: objadd, sinf
      Integer                          :: i, ifail, lena, leng, n, nea, neg,   &
                                          nf, nfname, ninf, ns, nxname,        &
                                          objrow, start
      Logical                          :: verbose_output
      Character (8)                    :: prob
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: a(:), f(:), flow(:), fmul(:),       &
                                          fupp(:), x(:), xlow(:), xmul(:),     &
                                          xupp(:)
      Real (Kind=nag_wp)               :: ruser(1), rw(lenrw)
      Integer, Allocatable             :: fstate(:), iafun(:), igfun(:),       &
                                          javar(:), jgvar(:), xstate(:)
      Integer                          :: iuser(1), iw(leniw)
      Character (8)                    :: cuser(1), cw(lencw)
      Character (8), Allocatable       :: fnames(:), xnames(:)
!     .. Executable Statements ..
      Write (nout,*) 'E04VJF Example Program Results'

!     Skip heading in data file
      Read (nin,*)

      Read (nin,*) n, nf
      lena = 300
      leng = 300
      nxname = 1
      nfname = 1
      Allocate (iafun(lena),javar(lena),igfun(leng),jgvar(leng),xstate(n),     &
        fstate(nf),a(lena),xlow(n),xupp(n),flow(nf),fupp(nf),x(n),xmul(n),     &
        f(nf),fmul(nf),xnames(nxname),fnames(nfname))

      Write (nout,99999) n

!     Call E04VGF to initialise E04VJF.

      ifail = 0
      Call e04vgf(cw,lencw,iw,leniw,rw,lenrw,ifail)

!     Read the bounds on the variables.
```

```
      Do i = 1, n
        Read (nin,*) xlow(i), xupp(i)
      End Do

      x(1:n) = 0.0E0_nag_wp

!     Determine the Jacobian structure.

      ifail = 0
      Call e04vjf(nf,n,usrfun,iafun,javar,a,lena,nea,igfun,jgvar,leng,neg,x,   &
        xlow,xupp,cw,lencw,iw,leniw,rw,lenrw,cuser,iuser,ruser,ifail)

!     Print the Jacobian structure.

      Write (nout,*)
      Write (nout,99998) nea
      Write (nout,99997)
      Write (nout,99996)

      Do i = 1, nea
        Write (nout,99995) i, iafun(i), javar(i), a(i)
      End Do

      Write (nout,*)
      Write (nout,99994) neg
      Write (nout,99993)
      Write (nout,99992)

      Do i = 1, neg
        Write (nout,99991) i, igfun(i), jgvar(i)
      End Do

!     Now that we have the determined the structure of the
!     Jacobian, set up the information necessary to solve
!     the optimization problem.

      start = 0
      prob = ' '
      objadd = 0.0E0_nag_wp
      x(1:n) = 0.0E0_nag_wp
      xstate(1:n) = 0
      xmul(1:n) = 0.0E0_nag_wp
      f(1:nf) = 0.0E0_nag_wp
      fstate(1:nf) = 0
      fmul(1:nf) = 0.0E0_nag_wp

!     The row containing the objective function.

      Read (nin,*) objrow

!     Read the bounds on the functions.

      Do i = 1, nf
        Read (nin,*) flow(i), fupp(i)
      End Do

!     Set this to .True. to cause e04nqf to produce intermediate
!     progress output
      verbose_output = .False.

      If (verbose_output) Then
!       By default E04VHF does not print monitoring
!       information. Set the print file unit or the summary
!       file unit to get information.
        ifail = 0
        Call e04vmf('Print file',nout,cw,iw,rw,ifail)
      End If

!     Tell E04VHF that we supply no derivatives in USRFUN.
```

```
      ifail = 0
      Call e04vlf('Derivative option 0',cw,iw,rw,ifail)

!     Solve the problem.

      ifail = -1
      Call e04vhf(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,cw,lencw,iw,leniw,rw,  &
        lenrw,cuser,iuser,ruser,ifail)

      Select Case (ifail)
      Case (0,4)
        Write (nout,*)
        Write (nout,99990) f(objrow)
        Write (nout,99989)(x(i),i=1,n)
      End Select

99999 Format (1X,/,1X,'NLP problem contains ',I3,' variables')
99998 Format (1X,'NEA (the number of nonzero entries in A) = ',I3)
99997 Format (1X,'  I     IAFUN(I)   JAVAR(I)          A(I)')
99996 Format (1X,'----    --------   --------   -----------')
99995 Format (1X,I3,2I10,1P,E18.4)
99994 Format (1X,'NEG (the number of nonzero entries in G) = ',I3)
99993 Format (1X,'  I     IGFUN(I)   JGVAR(I)')
99992 Format (1X,'----    --------   --------')
99991 Format (1X,I3,2I10)
99990 Format (1X,'Final objective value = ',F11.1)
99989 Format (1X,'Optimal X = ',1P,7E12.3)
    End Program e04vjfe
```

10.2 Program Data

```
E04VJF Example Program Data
 4   6             : Values of N and NF
-0.55D0    0.55D0  : Bounds on the variables, XLOW(i), XUPP(i), for i = 1 to N
-0.55D0    0.55D0
 0.0D0   1200.0D0
 0.0D0   1200.0D0

 6                 : Value of OBJROW
 -894.8D0 -894.8D0 : Bounds on the functions, FLOW(i), FUPP(i), for i = 1 to NF
 -894.8D0 -894.8D0
-1294.8D0 -1294.8D0
-0.55D0    1.0D25
-0.55D0    1.0D25
-1.0D25    1.0D25
```

10.3 Program Results

```
 E04VJF Example Program Results

 NLP problem contains   4 variables

 NEA (the number of nonzero 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 nonzero 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

 Final objective value =       5126.5
 Optimal X =    1.189E-01  -3.962E-01   6.799E+02   1.026E+03
```