

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 Parameters

Note: all optional parameters are described in detail in Section 11.1 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 parameters 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 parameter 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 \leq -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	<i>Input/Output</i>
	<i>On entry:</i> this will be set by E04VJF.	
	<i>On exit:</i> 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	<i>Input</i>
	<i>On entry:</i> E04VJF will call USRFUN with NEEDG = 0 to indicate that G is not required.	
8:	LENG – INTEGER	<i>Input</i>
	<i>On entry:</i> 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	<i>Input/Output</i>
	<i>On entry:</i> concerns the calculations of the derivatives of the function $f(x)$.	
	<i>On exit:</i> 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	<i>User Workspace</i>
11:	IUSER(*) – INTEGER array	<i>User Workspace</i>
12:	RUSER(*) – REAL (KIND=nag_wp) array	<i>User Workspace</i>
	USRFUN is called with the parameters CUSER, IUSER and RUSER as supplied to E04VJF. You are free to use the arrays CUSER, IUSER and RUSER to supply information to USRFUN as an alternative to using COMMON global variables.	

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

4:	IAFUN(LENA) – INTEGER array	<i>Output</i>
5:	JAVAR(LENA) – INTEGER array	<i>Output</i>
6:	A(LENA) – REAL (KIND=nag_wp) array	<i>Output</i>

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

7:	LENA – INTEGER	<i>Input</i>
	<i>On entry:</i> 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 <i>overestimate</i> of the number of elements in the linear part of the Jacobian.	
	<i>Constraint:</i> $LENA \geq 1$.	
8:	NEA – INTEGER	<i>Output</i>
	<i>On exit:</i> 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:* $LENG \geq 1$.
- 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 j th variable (say, $x_j = \beta$, where $|\beta| < bigbnd$), set $XLOW(j) = XUPP(j) = \beta$.
- 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:* $LENCW \geq 600$.
- 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:* $LENIW \geq 600$.
- 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:* $LENRW \geq 600$.
- 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 may be used to pass information to this routine as an alternative to using COMMON global variables.

25: IFAIL – INTEGER

Input/Output

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction 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 parameter, 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

The initialization routine E04VGF has not been called or at least one of LENCW, LENIW or LENRW is less than 600.

IFAIL = 2

An input parameter is invalid. The output message provides more details of the invalid parameter.

IFAIL = 3

Undefined user-supplied function.

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 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, resulting in insufficient array to store the Jacobian information. Increase LENA and/or LENG

IFAIL = 6

Unable to estimate the Jacobian structure.

IFAIL = 7

Internal memory allocation failed when attempting to obtain the required workspace. Please contact NAG.

IFAIL = 8

Internal memory allocation insufficient. Please contact NAG.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 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

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

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

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

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

$$x_0 = (0, 0, 0, 0)^T,$$

and $f(x_0) = 0$.

The optimal solution (to five figures) is

$$x^* = (0.11887, -0.39623, 679.94, 1026.0)^T,$$

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

```
! E04VJF Example Program Text
! Mark 24 Release. NAG Copyright 2012.

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
!   .. Parameters ..
    Integer, Parameter                :: 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
    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))

!   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,99999) nea
    Write (nout,99998)
    Write (nout,99997)

    Do i = 1, nea
        Write (nout,99996) i, iafun(i), javar(i), a(i)
    End Do

    Write (nout,*)
    Write (nout,99995) neg
    Write (nout,99994)
    Write (nout,99993)

    Do i = 1, neg
        Write (nout,99992) i, igfun(i), jgvar(i)
        Flush (nout)
    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

!   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)

!      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,99991) f(objrow)
        Write (nout,99990)(x(i),i=1,n)
      End Select

99999 Format (1X,'NEA (the number of non-zero entries in A) = ',I3)
99998 Format (1X,' I      IAFUN(I)  JAVAR(I)      A(I)')
99997 Format (1X,'----  -----  -----  -----')
99996 Format (1X,I3,2I10,1P,E18.4)
99995 Format (1X,'NEG (the number of non-zero entries in G) = ',I3)
99994 Format (1X,' I      IGFUN(I)  JGVAR(I)')
99993 Format (1X,'----  -----  -----')
99992 Format (1X,I3,2I10)
99991 Format (1X,'Final objective value = ',F11.1)
99990 Format (1X,'Optimal X = ',7F9.2)
      End Program e04vjfe

```

9.2 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

```

9.3 Program Results

E04VJF 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

QP solver 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.....	2.04E-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.....	2.04E-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

Matrix statistics

	Total	Normal	Free	Fixed	Bounded
Rows	5	2	0	3	0
Columns	4	0	0	0	4

No. of matrix elements 12 Density 60.000
 Biggest 1.0000E+00 (excluding fixed columns,
 Smallest 0.0000E+00 free rows, and RHS)

No. of objective coefficients 0
 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
5	1	2	1.2E-03	2	4.0E+02	1.0E+00	1.7331709E+06	13		1	1.3E+07	5.1E+00 _n r l
6	2	1	1.3E-03	3	2.7E+02	5.5E-01	1.7301152E+06	13				5.1E+00 _s 1
6	3	0	7.5E-03	4	8.8E+01	5.4E-01	8.8193389E+05	13				2.8E+00 _ 1
6	4	0	2.3E-02	5	2.9E+01	5.3E-01	8.4262012E+05	13				2.8E+00 _ 1
6	5	0	6.9E-02	6	8.9E+00	5.2E-01	7.3075574E+05	13				2.8E+00 _ 1
7	6	1	2.2E-01	7	2.3E+00	8.0E+01	4.4817386E+05	13		1	1.2E+04	2.8E+00 _ 1
8	7	1	8.3E-01	8	1.7E-01	9.2E+00	2.4331224E+04	13		1	9.5E+03	2.8E+00 _ 1
9	8	1	1.0E+00	9	6.5E-03	4.0E+01	5.3127971E+03	13	1	1	1.3E+02	2.8E+00 _
10	9	1	1.0E+00	10	4.6E-03	1.2E+01	5.1602365E+03	13		1	9.4E+01	2.8E+00 _
11	10	1	1.0E+00	11	2.3E-04	6.2E-02	5.1265651E+03	13		1	9.6E+01	2.8E+00 _
12	11	1	1.0E+00	12	(1.3E-08)	2.9E-04	5.1264981E+03	13		1	1.2E+02	2.8E+00 _
c												
13	11	2	1.0E+00	12	(1.3E-08)	2.7E-04	5.1264981E+03	13		1	1.2E+02	2.8E+00 _
c												
14	12	1	1.0E+00	13	(5.5E-13)	7.0E-05	5.1264981E+03	13		1	9.5E+01	2.8E+00 _
c												
15	13	1	1.0E+00	14	(1.8E-14)(2.6E-09)		5.1264981E+03	13		1	9.5E+01	2.8E+00 _
c												

E04VHU EXIT 0 -- finished successfully
 E04VHU INFO 1 -- optimality conditions satisfied

Problem name
 No. of iterations 15 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 104 No. of calls to funcon 104
 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 2.5E-08
 Nonlinear constraint violn 1.5E-11

Name Objective Value 5.1264981096E+03
 Status Optimal Soln Iteration 15 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
