NAG Toolbox for MATLAB

e04vj

1 Purpose

e04vj may be used before e04vh to determine the sparsity pattern for the Jacobian.

2 Syntax

```
[iafun, javar, nea, a, igfun, jgvar, neg, cw, iw, rw, user, ifail] =
e04vj(nf, usrfun, lena, leng, x, xlow, xupp, cw, iw, rw, 'n', n,
'lencw', lencw, 'leniw', leniw, 'lenrw', lenrw, 'user', user)
```

3 Description

When using e04vh, if you set the optional parameter **Derivative Option** $= 0$ and user-supplied (sub)program **usrfun** provides none of the derivatives, you may need to call e04vj 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

```
[cw, iw, rw, ifail] = e04vg();
[..., cw, iw, rw, ...] = e04vj(...);
[cw, iw, rw, ifail] = e04vl('Derivative Option = 0', cw, iw, rw);
[..., cw, iw, rw, ...] = e04vh(...);
```

e04vj determines the sparsity pattern for the Jacobian and identifies the constant elements automatically. To do so, e04vj 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, e04vj 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 e04vj must classify them as nonlinear. This is because e04vh 'removes' linear variables from the calculation of F by setting them to zero before calling user-supplied (sub)program **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 e04vh.

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 11.2 of the document for e04vh.

5.1 Compulsory Input Parameters

1: **nf – int32 scalar**

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: **usrfun – string containing name of m-file**

usrfun must define the problem functions $F(x)$. This (sub)program is passed to e04vj as the external parameter **usrfun**.

Its specification is:

```
[status, f, g, user] = usrfun(status, n, x, needf, nf, f, needg,
leng, g, user)
```

Input Parameters

1: **status – int32 scalar**

Indicates the first call to **usrfun**.

status $= 0$

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

status $= 1$

e04vj is calling your (sub)program for the *first* time. Some data may need to be input or computed and saved.

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).

e04vj 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** $= -1$, e04vj will use another random perturbation.

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

2: **n – int32 scalar**

n, the number of variables, as defined in the call to e04vj.

3: **x(n) – double array**

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

4: **needf – int32 scalar**

Indicates if **f** must be assigned during the call to **usrfun** (see **f**).

5: **nf – int32 scalar**

nf, the number of problem functions.

6: **f(nf) – double array**

This will be set by e04vj.

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**. e04vj 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 – int32 scalar**

e04vj will call **usrfun** with **needg** $= 0$ to indicate that **g** is not required.

8: **leng – int32 scalar**

9: **g(leng) – double array**

Concerns the calculations of the derivatives of the function $f(x)$.

e04vj will always call **usrfun** with **needg** $= 0$: **g** is not required to be set on exit but must be declared correctly.

10: **user – Any MATLAB object**

usrfun is called from e04vj with **user** as supplied to e04vj

Output Parameters

1: **status – int32 scalar**

Indicates the first call to **usrfun**.

status $= 0$

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

status $= 1$

e04vj is calling your (sub)program for the *first* time. Some data may need to be input or computed and saved.

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).

e04vj 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** $= -1$, e04vj will use another random perturbation.

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

2: **f(nf) – double array**

This will be set by e04vj.

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**. e04vj 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**.

3: **g(leng) – double array**

Concerns the calculations of the derivatives of the function $f(x)$.

e04vj will always call **usrfun** with **needg** $= 0$: **g** is not required to be set on exit but must be declared correctly.

4: **user – Any MATLAB object**

usrfun is called from e04vj with **user** as supplied to e04vj

3: **lena – int32 scalar**

lena should be an *overestimate* of the number of elements in the linear part of the Jacobian.

Constraint: **lena** ≥ 1.

4: **leng – int32 scalar**

leng should be an *overestimate* of the number of elements in the nonlinear part of the Jacobian.

Constraint: **leng** ≥ 1.

5: **x(n) – double array**

An initial estimate of the variables x. The contents of x will be used by e04vj in the call of user-supplied (sub)program **usrfun**, and so each element of **x** should be within the bounds given by **xlow** and **xupp**.

6: **xlow(n) – double array**
7: **xupp(n) – double array**

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) $= \beta$.

8: **cw(lencw) – string array**

Constraint: **lencw** ≥ 600.

9: **iw(leniw) – int32 array**

Constraint: **leniw** ≥ 600.

10: **rw(lenrw) – double array**

Constraint: **lenrw** ≥ 600.

5.2 Optional Input Parameters

1: **n – int32 scalar**

Default: The dimension of the arrays **x**, **xlow**, **xupp**. (An error is raised if these dimensions are not equal.)

n, the number of variables.

Constraint: **n** > 0.

2: **lencw – int32 scalar**

Default: The dimension of the array **cw**.

Constraint: **lencw** ≥ 600.

3: **leniw – int32 scalar**

Default: The dimension of the array **iw**.

Constraint: **leniw** ≥ 600.

4: **lenrw – int32 scalar**

Default: The dimension of the array **rw**.

Constraint: **lenrw** ≥ 600.

5: **user – Any MATLAB object**

user is not used by e04vj, but is passed to **usrfun**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **iafun(lena) – int32 array**

2: **javar(lena) – int32 array**

3: **nea – int32 scalar**

Is the number of nonzero entries in A such that $F(x) = f(x) + Ax$.

4: **a(lena) – double array**

Define the co-ordinates (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 $(\mathbf{iafun}(k), \mathbf{javar}(k), \mathbf{a}(k))$ define the row and column indices $i = \mathbf{iafun}(k)$ and $j = \mathbf{javar}(k)$ of the element $A_{ij} = \mathbf{a}(k)$.

5: **igfun(leng) – int32 array**

6: **jgvar(leng) – int32 array**

Define the co-ordinates (i,j) of the nonzero elements of G, the nonlinear part of the derivatives $J(x) = G(x) + A$ of the function $F(x) = f(x) + Ax$.

7: **neg – int32 scalar**

The number of nonzero entries in G.

8: **cw(lencw) – string array**

9: **iw(leniw) – int32 array**

10: **rw(lenrw) – double array**

11: **user – Any MATLAB object**

user is not used by e04vj, but is passed to **usrfun**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

12: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail = 1

The initialization function e04vg 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 user-supplied (sub)program **usrfun**. This exit occurs if e04vj 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 e04vj by setting **status** to a value < -1 on exit from user-supplied (sub)program **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

```
e04vj_usrfun.m
```

```
function [status, f, g, user] = usrfun(status, n, x, needf, nf, f,
needg, leng, g, user)

  % Always called with needf > 0
  f(1) = 1000*sin(-x(1)-0.25) + 1000*sin(-x(2) -0.25) - x(3);
```

```
   f(2) = 1000*sin(x(1)-0.25) + 1000*sin(x(1)-x(2) -0.25) - x(4);
   f(3) = 1000*sin(x(2)-x(1)-0.25) + 1000*sin(x(2) -0.25);
   f(4) = -x(1) + x(2);
   f(5) = x(1) - x(2);
   f(6) = 1.0d-6*x(3)^3 + 2.0d-6*x(4)^3/3 + 3*x(3) + 2*x(4);

   % Always called with needg=0
```

```
nf = int32(6);
lena = int32(300);
leng = int32(300);
x = [0;
     0;
     0;
     0];
xlow = [-0.55;
     -0.55;
     0;
     0];
xupp = [0.55;
     0.55;
     1200;
     1200];
[cw, iw, rw, ifail] = e04vg();
[iafun, javar, nea, a, igfun, jgvar, neg, cwOut, iwOut, rwOut, cuser,
ifail] = ...
    e04vj(nf, 'e04vj_usrfun', lena, leng, x, xlow, xupp, cw, iw, rw)
```

```
iafun =
     array elided
javar =
     array elided
nea =
           4
a =
     array elided
igfun =
     array elided
jgvar =
     array elided
neg =
          10
cwOut =
     array elided
iwOut =
     array elided
rwOut =
     array elided
cuser =
     0
ifail =
           0
```