NAG Toolbox for MATLAB

e04xa

1 Purpose

e04xa computes an approximation to the gradient vector and/or the Hessian matrix for use in conjunction with, or following the use of an optimization function (such as e04uf).

2 Syntax

```
[mode, hforw, objf, objgrd, hcntrl, h, iwarn, user, info, lwsav, iwsav,
rwsav, ifail] = e04xa(msglvl, epsrf, x, mode, objfun, hforw, lwsav,
iwsav, rwsav, 'n', n, 'user', user)
```

Before calling e04xa, e04wb **must** be called.

3 Description

e04xa is similar to routine FDCALC described in Gill *et al.* 1983a. It should be noted that this function aims to compute sufficiently accurate estimates of the derivatives for use with an optimization algorithm. If you require more accurate estimates you should refer to Chapter D04.

e04xa computes finite-difference approximations to the gradient vector and the Hessian matrix for a given function. The simplest approximation involves the forward-difference formula, in which the derivative $f'(x)$ of a univariate function $f(x)$ is approximated by the quantity

$$\rho_F(f,h) = \frac{f(x+h) - f(x)}{h}$$

for some interval $h > 0$, where the subscript 'F' denotes 'forward-difference' (see Gill *et al.* 1983b).

To summarize the procedure used by e04xa (for the case when the objective function is available and you require estimates of gradient values and Hessian matrix diagonal values, i.e., **mode** $= 0$) consider a univariate function f at the point x. (In order to obtain the gradient of a multivariate function $F(x)$, where x is an n-vector, the procedure is applied to each component of x, keeping the other components fixed.) Roughly speaking, the method is based on the fact that the bound on the relative truncation error in the forward-difference approximation tends to be an increasing function of h, while the relative condition error bound is generally a decreasing function of h, hence changes in h will tend to have opposite effects on these errors (see Gill *et al.* 1983b).

The 'best' interval h is given by

$$h_F = 2\sqrt{\frac{(1+|f(x)|)e_R}{|\Phi|}} \tag{1}$$

where Φ is an estimate of $f''(x)$, and e_R is an estimate of the relative error associated with computing the function (see Chapter 8 of Gill *et al.* 1981). Given an interval h, Φ is defined by the second-order approximation

$$\Phi = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

The decision as to whether a given value of Φ is acceptable involves $\hat{c}(\Phi)$, the following bound on the relative condition error in Φ:

$$\hat{c}(\Phi) = \frac{4e_R(1+|f|)}{h^2|\Phi|}$$

(When Φ is zero, $\hat{c}(\Phi)$ is taken as an arbitrary large number.)

The procedure selects the interval h_ϕ (to be used in computing Φ) from a sequence of trial intervals (h_k). The initial trial interval is taken as $10\bar{h}$, where

$$\bar{h} = 2(1 + |x|)\sqrt{e_R}$$

unless you specify the initial value to be used.

The value of $\hat{c}(\Phi)$ for a trial value h_k is defined as 'acceptable' if it lies in the interval $[0.001, 0.1]$. In this case h_ϕ is taken as h_k, and the current value of Φ is used to compute h_F from (1). If $\hat{c}(\Phi)$ is unacceptable, the next trial interval is chosen so that the relative condition error bound will either decrease or increase, as required. If the bound on the relative condition error is too large, a larger interval is used as the next trial value in an attempt to reduce the condition error bound. On the other hand, if the relative condition error bound is too small, h_k is reduced.

The procedure will fail to produce an acceptable value of $\hat{c}(\Phi)$ in two situations. Firstly, if $f''(x)$ is extremely small, then $\hat{c}(\Phi)$ may never become small, even for a very large value of the interval. Alternatively, $\hat{c}(\Phi)$ may never exceed 0.001, even for a very small value of the interval. This usually implies that $f''(x)$ is extremely large, and occurs most often near a singularity.

As a check on the validity of the estimated first derivative, the procedure provides a comparison of the forward-difference approximation computed with h_F (as above) and the central-difference approximation computed with h_ϕ. Using the central-difference formula the first derivative can be approximated by

$$\rho_c(f, h) = \frac{f(x + h) - f(x - h)}{2h}$$

where $h > 0$. If the values h_F and h_ϕ do not display some agreement, neither can be considered reliable.

When both function and gradients are available and you require the Hessian matrix (i.e., **mode** = 1) e04xa follows a similar procedure to the case above with the exception that the gradient function $g(x)$ is substituted for the objective function and so the forward-difference interval for the first derivative of $g(x)$ with respect to variable x_j is computed. The jth column of the approximate Hessian matrix is then defined as in Chapter 2 of Gill *et al.* 1981, by

$$\frac{g\left(x + h_j e_j\right) - g(x)}{h_j}$$

where h_j is the best forward-difference interval associated with the jth component of g and e_j is the vector with unity in the jth position and zeros elsewhere.

When only the objective function is available and you require the gradients and Hessian matrix (i.e., **mode** = 2) e04xa again follows the same procedure as the case for **mode** = 0 except that this time the value of $\hat{c}(\Phi)$ for a trial value h_k is defined as acceptable if it lies in the interval $[0.0001, 0.01]$ and the initial trial interval is taken as

$$\bar{h} = 2(1 + |x|)\sqrt[4]{e_R}.$$

The approximate Hessian matrix G is then defined as in Chapter 2 of Gill *et al.* 1981, by

$$G_{ij}(x) = \frac{1}{h_i h_j}\left(f\left(x + h_i e_i + h_j e_j\right) - f(x + h_i e_i) - f\left(x + h_j e_j\right) + f(x)\right).$$

4 References

Gill P E, Murray W, Saunders M A and Wright M H 1983a Documentation for FDCALC and FDCORE *Technical Report SOL* 83–6 Stanford University

Gill P E, Murray W, Saunders M A and Wright M H 1983b Computing forward-difference intervals for numerical optimization *SIAM J. Sci. Statist. Comput.* **4** 310–321

Gill P E, Murray W and Wright M H 1981 *Practical Optimization* Academic Press

5 Parameters

5.1 Compulsory Input Parameters

1: **msglvl – int32 scalar**

Must indicate the amount of intermediate output desired (see Section 8.1 for a description of the printed output). All output is written on the current advisory message unit (see x04ab).

Value	Definition
0	No printout
1	A summary is printed out for each variable plus any warning messages.
Other	Values other than 0 and 1 should normally be used **only** at the direction of NAG.

2: **epsrf – double scalar**

Must define e_R, which is intended to be a measure of the accuracy with which the problem function F can be computed. The value of e_R should reflect the relative precision of $1 + |F(x)|$, i.e., acts as a relative precision when $|F|$ is large, and as an absolute precision when $|F|$ is small. For example, if $F(x)$ is typically of order 1000 and the first six significant digits are known to be correct, an appropriate value for e_R would be 1.0D−6.

A discussion of **epsrf** is given in Chapter 8 of Gill *et al.* 1981. If **epsrf** is either too small or too large on entry a warning will be printed if **msglvl** $= 1$, the parameter **iwarn** set to the appropriate value on exit and e04xa will use a default value of $e_M^{0.9}$, where e_M is the ***machine precision***.

If **epsrf** ≤ 0.0 on entry then e04xa will use the default value internally. The default value will be appropriate for most simple functions that are computed with full accuracy.

3: **x(n) – double array**

The point x at which the derivatives are to be computed.

4: **mode – int32 scalar**

Indicates which derivatives are required.

mode $= 0$

The gradient and Hessian diagonal values having supplied the objective function via user-supplied (sub)program **objfun**.

mode $= 1$

The Hessian matrix having supplied both the objective function and gradients via user-supplied (sub)program **objfun**.

mode $= 2$

The gradient values and Hessian matrix having supplied the objective function via user-supplied (sub)program **objfun**.

5: **objfun – string containing name of m-file**

If **mode** $= 0$ or 2, **objfun** must calculate the objective function; otherwise if **mode** $= 1$, **objfun** must calculate the objective function and the gradients.

Its specification is:

```
[mode, objf, objgrd, user] = objfun(mode, n, x, nstate, user)
```

Input Parameters

1: **mode – int32 scalar**

mode indicates which parameter values within **objfun** need to be set.

To **objfun**, **mode** is always set to the value that you set it to before the call to e04xa.

Its value must not be altered unless you wish to indicate a failure within **objfun**, in which case it should be set to a negative value. If **mode** is negative on exit from **objfun**, the execution of e04xa is terminated with **ifail** set to **mode**.

2: **n – int32 scalar**

The number n of variables as input to e04xa.

3: **x(n) – double array**

The point x at which the objective function (and gradients if **mode** $= 1$) is to be evaluated.

4: **nstate – int32 scalar**

Will be set to 1 on the first call of **objfun** by e04xa, and is 0 for all subsequent calls. Thus, if you wish, **nstate** may be tested within **objfun** in order to perform certain calculations once only. For example you may read data.

5: **user – Any MATLAB object**

objfun is called from e04xa with **user** as supplied to e04xa

Output Parameters

1: **mode – int32 scalar**

mode indicates which parameter values within **objfun** need to be set.

To **objfun**, **mode** is always set to the value that you set it to before the call to e04xa.

Its value must not be altered unless you wish to indicate a failure within **objfun**, in which case it should be set to a negative value. If **mode** is negative on exit from **objfun**, the execution of e04xa is terminated with **ifail** set to **mode**.

2: **objf – double scalar**

Must be set to the value of the objective function.

3: **objgrd(n) – double array**

If **mode** $= 1$, **objgrd**(j) must contain the value of the first derivative with respect to x.

If **mode** $\neq 1$, **objgrd** need not be set.

4: **user – Any MATLAB object**

objfun is called from e04xa with **user** as supplied to e04xa

6: **hforw(n) – double array**

The initial trial interval for computing the appropriate partial derivative to the jth variable.

If **hforw**$(j) \leq 0.0$, then the initial trial interval is computed by e04xa (see Section 3).

7: **lwsav(120) – logical array**
8: **iwsav(610) – int32 array**
9: **rwsav(475) – double array**

The arrays **lwsav**, **iwsav** and **rwsav** **must not** be altered between calls to either of the functions e04wb and e04xa.

5.2 Optional Input Parameters

1: **n – int32 scalar**

Default: The dimension of the arrays **x**, **hforw**, **objgrd**, **hcntrl**, **info**. (An error is raised if these dimensions are not equal.)

the number n of independent variables.

Constraint: $\mathbf{n} \geq 1$.

2: **user – Any MATLAB object**

user is not used by e04xa, but is passed to **objfun**. 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

ldh, work

5.4 Output Parameters

1: **mode – int32 scalar**

Is changed **only** if you set **mode** negative in user-supplied (sub)program **objfun**, i.e., you have requested termination of e04xa.

2: **hforw**(**n**) **– double array**

hforw(j) is the best interval found for computing a forward-difference approximation to the appropriate partial derivative for the jth variable.

3: **objf – double scalar**

The value of the objective function evaluated at the input vector in **x**.

4: **objgrd**(**n**) **– double array**

If **mode** $= 0$ or 2, **objgrd**(j) contains the best estimate of the first partial derivative for the jth variable.

If **mode** $= 1$, **objgrd**(j) contains the first partial derivative for the jth variable evaluated at the input vector in **x**.

5: **hcntrl**(**n**) **– double array**

hcntrl(j) is the best interval found for computing a central-difference approximation to the appropriate partial derivative for the jth variable.

6: **h**(**ldh,**$*$) **– double array**

The first dimension of the array **h** must be at least **n**

The second dimension of the array must be at least 1 if **mode** $= 0$ and at least **n** if **mode** $= 1$ or 2

If **mode** $= 0$, the estimated Hessian diagonal elements are contained in the first column of this array.

If **mode** $= 1$ or 2, the estimated Hessian matrix is contained in the leading n by n part of this array.

7: **iwarn – int32 scalar**

iwarn $= 0$ on successful exit.

If the value of **epsrf** on entry is too small or too large then **iwarn** is set to 1 or 2 respectively on exit and **epsrf** is set to the default value within e04xa.

If **msglvl** > 0 then warnings will be printed if **epsrf** is too small or too large.

8: **user – Any MATLAB object**

user is not used by e04xa, but is passed to **objfun**. 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**.

9: **info**(**n**) **– int32 array**

info(j) represents diagnostic information on variable j. (See Section 6 for more details.)

10: **lwsav**(**120**) **– logical array**

11: **iwsav**(**610**) **– int32 array**

12: **rwsav**(**475**) **– double array**

The arrays **lwsav**, **iwsav** and **rwsav must not** be altered between calls to either of the functions e04wb and e04xa.

13: **ifail – int32 scalar**

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

6 Error Indicators and Warnings

On exit from e04xa both diagnostic parameters **info** and **ifail** should be tested. **ifail** represents an overall diagnostic indicator, whereas the integer array **info** represents diagnostic information on each variable.

Errors or warnings detected by the function:

ifail < 0

A negative value of **ifail** indicates an exit from e04xa because you set **mode** negative in user-supplied (sub)program **objfun**. The value of **ifail** will be the same as your setting of **mode**.

ifail $= 1$

On entry, one or more of the following conditions are satisfied: **n** < 1, **ldh** $<$ **n** or **mode** is invalid.

ifail $= 2$

One or more variables have a nonzero **info** value. This may not necessarily represent an unsuccessful exit – see diagnostic information on **info**.

Diagnostic information returned via **info** is as follows:

info(i) $= 1$

The appropriate function appears to be constant. **hforw**(i) is set to the initial trial interval value (see Section 3) corresponding to a well-scaled problem and `Error est.` in the printed output is set to zero. This value occurs when the estimated relative condition error in the first derivative approximation is unacceptably large for every value of the finite-difference interval. If this happens when the function is not constant the initial interval may be too small; in this case, it may be worthwhile to rerun e04xa with larger initial trial interval values supplied in **hforw** (see Section 3). This error may also occur if the function evaluation includes an inordinately large constant term or if **epsrf** is too large.

info(i) $= 2$

The appropriate function appears to be linear or odd. **hforw**(i) is set to the smallest interval with acceptable bounds on the relative condition error in the forward- and backward-difference estimates. In this case, the estimated relative condition error in the second derivative approximation remained large for every trial interval, but the estimated error in the first derivative approximation was acceptable for at least one interval. If the function is not linear or odd the relative condition error in the second derivative may be decreasing very slowly, it may be worthwhile to rerun e04xa with larger initial trial interval values supplied in **hforw** (see Section 3).

info$(i) = 3$

The second derivative of the appropriate function appears to be so large that it cannot be reliably estimated (i.e., near a singularity). **hforw**(i) is set to the smallest trial interval.

This value occurs when the relative condition error estimate in the second derivative remained very small for every trial interval.

If the second derivative is not large the relative condition error in the second derivative may be increasing very slowly. It may be worthwhile to rerun e04xa with smaller initial trial interval values supplied in **hforw** (see Section 3). This error may also occur when the given value of **epsrf** is not a good estimate of a bound on the absolute error in the appropriate function (i.e., **epsrf** is too small).

info$(i) = 4$

The algorithm terminated with an apparently acceptable estimate of the second derivative. However the forward-difference estimates of the appropriate first derivatives (computed with the final estimate of the 'optimal' forward-difference interval) and the central difference estimates (computed with the interval used to compute the final estimate of the second derivative) do not agree to half a decimal place. The usual reason that the forward- and central-difference estimates fail to agree is that the first derivative is small.

If the first derivative is not small, it may be helpful to execute the procedure at a different point.

7 Accuracy

ifail contains 0 on exit if the algorithm terminated successfully, i.e., the forward-difference estimates of the appropriate first derivatives (computed with the final estimate of the 'optimal' forward-difference interval h_F) and the central-difference estimates (computed with the interval h_ϕ used to compute the final estimate of the second derivative) agree to at least half a decimal place.

In short word length implementations when computing the full Hessian matrix given function values only (i.e., **mode** $= 2$) the elements of the computed Hessian will have at best 1 to 2 figures of accuracy.

8 Further Comments

To evaluate an acceptable set of finite-difference intervals for a well-scaled problem, the function will require around two function evaluations per variable; in a badly scaled problem however, as many as six function evaluations per variable may be needed.

If you request the full Hessian matrix supplying both function and gradients (i.e., **mode** $= 1$) or function only (i.e., **mode** $= 2$) then a further $\mathbf{n}$ or $3 \times \mathbf{n} \times (\mathbf{n} + 1)/2$ function evaluations respectively are required.

8.1 Description of the Printed Output

The following is a description of the printed output from e04xa as controlled by the parameter **msglvl**.

Output when **msglvl** $= 1$ is as follows:

`J`	number of variable for which the difference interval has been computed.
`X`(j)	jth variable of x as set by you.
`F. dif. int.`	the best interval found for computing a forward-difference approximation to the appropriate partial derivative with respect to the jth variable.
`C. dif. int.`	the best interval found for computing a central-difference approximation to the appropriate partial derivative with respect to the jth variable.
`Error est.`	a bound on the estimated error in the final forward-difference approximation. When **info**$(j) = 1$, `Error est.` is set to zero.
`Grad. est.`	best estimate of the first partial derivative with respect to the jth variable.
`Hess diag est.`	best estimate of the second partial derivative with respect to the jth variable.

`fun evals.` the number of function evaluations used to compute the final difference intervals for the jth variable.

`info`(j) the value of **info** for the jth variable.

9 Example

e04xa_objfun.m

```
function [mode, objf, objgrd, user] = objfun(mode, n, x, nstate, user)
  objgrd = zeros(n, 1);

  a = x(1) + 10*x(2);
  b = x(3) - x(4);
  c = x(2) - 2*x(3);
  d = x(1) - x(4);
  objf = a^2 + 5*b^2 + c^4 + 10*d^4;
  if (mode == 1)
     objgrd(1) = 40*x(1)^3 + 2*x(1) - 120*x(4)*x(1)^2 + 120*x(1)*x(4)^2
+ 20*x(2) - 40*x(4)^3;
      objgrd(2) = 200*x(2) + 20*x(1) + 4*x(2)^3 + 48*x(2)*x(3)^2 -
24*x(3)*x(2)^2 - 32*x(3)^3;
       objgrd(3) = 10*x(3) - 10*x(4) - 8*x(2)^3 + 48*x(3)*x(2)^2 -
96*x(2)*x(3)^2 + 64*x(3)^3;
      objgrd(4) = 10*x(4) - 10*x(3) - 40*x(1)^3 + 120*x(4)*x(1)^2 -
120*x(1)*x(4)^2 + 40*x(4)^3;
  end
```

```
msglvl = int32(0);
epsrf = -1;
x = [3;
     -1;
     0;
     1];
mode = int32(0);
hforw = [-1;
     -1;
     -1;
     -1];
lwsav = false(120,1);
iwsav = zeros(610,1,'int32');
rwsav = zeros(475,1);
[modeOut, hforwOut, objf, objgrd, hcntrl, h, iwarn, user, info, ...
 lwsavOut, iwsavOut, rwsavOut, ifail] = ...
     e04xa(msglvl, epsrf, x, mode, 'e04xa_objfun', hforw, lwsav, iwsav,
rwsav)
```

```
modeOut =
           0
hforwOut =
   1.0e-06 *
    0.0886
    0.1336
    0.2553
    0.0879
objf =
   215
objgrd =
  306.0000
 -144.0000
   -2.0000
 -310.0000
hcntrl =
   1.0e-05 *
    0.5293
```

```
    0.2647
    0.1323
    0.2647
h =
  482.0035
  212.0024
   58.0088
  489.9992
iwarn =
           0
user =
     0
info =
           0
           0
           0
           0
lwsavOut =
     array elided
iwsavOut =
     array elided
rwsavOut =
     array elided
ifail =
           0
```