NAG Toolbox for MATLAB

e04hy

1 Purpose

e04hy is an easy-to-use modified Gauss–Newton algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables $(m \geq n)$. First and second derivatives are required.

It is intended for functions which are continuous and which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

2 Syntax

```
[x, fsumsq, user, ifail] = e04hy(m, lsfun2, lshes2, x, 'n', n, 'user', user)
```

3 Description

e04hy is similar to the (sub)program LSSDN2 in the NPL Algorithms Library. It is applicable to problems of the form:

$$\text{Minimize } F(x) = \sum_{i=1}^{m} [f_i(x)]^2$$

where $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ and $m \geq n$. (The functions $f_i(x)$ are often referred to as 'residuals'.)

You must supply a (sub)program to evaluate the residuals and their first derivatives at any point x, and a (sub)program to evaluate the elements of the second derivative term of the Hessian matrix of $F(x)$.

Before attempting to minimize the sum of squares, the algorithm checks the user-supplied (sub)programs for consistency. Then, from a starting point supplied by you, a sequence of points is generated which is intended to converge to a local minimum of the sum of squares. These points are generated using estimates of the curvature of $F(x)$.

4 References

Gill P E and Murray W 1978 Algorithms for the solution of the nonlinear least-squares problem *SIAM J. Numer. Anal.* **15** 977–992

5 Parameters

5.1 Compulsory Input Parameters

1: **m – int32 scalar**

the number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \mathbf{n} \leq \mathbf{m}$.

2: **lsfun2 – string containing name of m-file**

You must supply this function to calculate the vector of values $f_i(x)$ and the Jacobian matrix of first derivatives $\dfrac{\partial f_i}{\partial x_j}$ at any point x. It should be tested separately before being used in conjunction with e04hy (see the E04 Chapter Introduction).

Its specification is:

```
[fvecc, fjacc, user] = lsfun2(m, n, xc, ljc, user)
```

Input Parameters

1: **m – int32 scalar**
2: **n – int32 scalar**

The numbers m and n of residuals and variables, respectively.

3: **xc(n) – double array**

The point x at which the values of the f_i and the $\dfrac{\partial f_i}{\partial x_j}$ are required.

4: **ljc – int32 scalar**

The first dimension of the array **fjacc**.

5: **user – Any MATLAB object**

lsfun2 is called from e04hy with **user** as supplied to e04hy

Output Parameters

1: **fvecc(m) – double array**

fvecc(i) must be set to the value of f_i at the point x, for $i = 1, 2, \ldots, m$.

2: **fjacc(ljc,n) – double array**

fjacc(i,j) must be set to the value of $\dfrac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

3: **user – Any MATLAB object**

lsfun2 is called from e04hy with **user** as supplied to e04hy

3: **lshes2 – string containing name of m-file**

You must supply this function to calculate the elements of the symmetric matrix

$$B(x) = \sum_{i=1}^{m} f_i(x) G_i(x),$$

at any point x, where $G_i(x)$ is the Hessian matrix of $f_i(x)$. It should be tested separately before being used in conjunction with e04hy (see the E04 Chapter Introduction).

Its specification is:

```
[b, user] = lshes2(m, n, fvecc, xc, lb, user)
```

Input Parameters

1: **m – int32 scalar**
2: **n – int32 scalar**

The numbers m and n of residuals and variables, respectively.

3: **fvecc(m) – double array**

The value of the residual f_i at the point in **xc**, for $i = 1, 2, \ldots, m$, so that the values of the f_i can be used in the calculation of the elements of B.

4: **xc(n) – double array**

The point x at which the elements of **b** are to be evaluated.

5: **lb – int32 scalar**

The length of the array **b**.

6: **user – Any MATLAB object**

lshes2 is called from e04hy with **user** as supplied to e04hy

Output Parameters

1: **b(lb) – double array**

Must contain the lower triangle of the matrix $B(x)$, evaluated at the point x, stored by rows. (The upper triangle is not required because the matrix is symmetric.) More precisely, $\mathbf{b}(j(j-1)/2+k)$ must contain $\sum_{i=1}^{m} f_i \frac{\partial^2 f_i}{\partial x_j \partial x_k}$ evaluated at the point x, for $j = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, j$.

2: **user – Any MATLAB object**

lshes2 is called from e04hy with **user** as supplied to e04hy

4: **x(n) – double array**

$\mathbf{x}(j)$ must be set to a guess at the jth component of the position of the minimum, for $j = 1, 2, \ldots, n$. The function checks user-supplied (sub)program **lsfun2** and user-supplied (sub)program **lshes2** at the starting point and so is more likely to detect any error in your functions if the initial $\mathbf{x}(j)$ are nonzero and mutually distinct.

5.2 Optional Input Parameters

1: **n – int32 scalar**

Default: For **n**, the dimension of the array **x**.

the number m of residuals, $f_i(x)$, and the number n of variables, x_j.

Constraint: $1 \leq \mathbf{n} \leq \mathbf{m}$.

2: **user – Any MATLAB object**

user is not used by e04hy, but is passed to **lsfun2** and **lshes2**. 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

w, lw

5.4 Output Parameters

1: **x(n) – double array**

The lowest point found during the calculations. Thus, if **ifail** $= 0$ on exit, **x**(j) is the jth component of the position of the minimum.

2: **fsumsq – double scalar**

The value of the sum of squares, $F(x)$, corresponding to the final point stored in **x**.

3: **user – Any MATLAB object**

user is not used by e04hy, but is passed to **lsfun2** and **lshes2**. 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**.

4: **ifail – int32 scalar**

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

6 Error Indicators and Warnings

Note: e04hy may return useful information for one or more of the following detected errors or warnings.

ifail $= 1$

On entry, **n** < 1,
or **m** $<$ **n**,
or **lw** $< 8 \times$ **n** $+ 2 \times$ **n** $\times$ **n** $+ 2 \times$ **m** $\times$ **n** $+ 3 \times$ **m**, when **n** > 1,
or **lw** $< 11 + 5 \times$ **m**, when **n** $= 1$.

ifail $= 2$

There have been $50 \times n$ calls of user-supplied (sub)program **lsfun2**, yet the algorithm does not seem to have converged. This may be due to an awkward function or to a poor starting point, so it is worth restarting e04hy from the final point held in **x**.

ifail $= 3$

The final point does not satisfy the conditions for acceptance as a minimum, but no lower point could be found.

ifail $= 4$

An auxiliary function has been unable to complete a singular value decomposition in a reasonable number of sub-iterations.

ifail $= 5$
ifail $= 6$
ifail $= 7$
ifail $= 8$

There is some doubt about whether the point x found by e04hy is a minimum of $F(x)$. The degree of confidence in the result decreases as **ifail** increases. Thus, when **ifail** $= 5$, it is probable that the final x gives a good estimate of the position of a minimum, but when **ifail** $= 8$ it is very unlikely that the function has found a minimum.

ifail $= 9$

It is very likely that you have made an error in forming the derivatives $\frac{\partial f_i}{\partial x_j}$ in user-supplied (sub)program **lsfun2**.

ifail $= 10$

It is very likely that you have made an error in forming the quantities B_{jk} in user-supplied (sub)program **lshes2**.

If you are not satisfied with the result (e.g., because **ifail** lies between 3 and 8), it is worth restarting the calculations from a different starting point (not the point at which the failure occurred) in order to avoid the region which caused the failure. Repeated failure may indicate some defect in the formulation of the problem.

7 Accuracy

If the problem is reasonably well scaled and a successful exit is made, then, for a computer with a mantissa of t decimals, one would expect to get about $t/2-1$ decimals accuracy in the components of x and between $t-1$ (if $F(x)$ is of order 1 at the minimum) and $2t-2$ (if $F(x)$ is close to zero at the minimum) decimals accuracy in $F(x)$.

8 Further Comments

The number of iterations required depends on the number of variables, the number of residuals and their behaviour, and the distance of the starting point from the solution. The number of multiplications performed per iteration of e04hy varies, but for $m \gg n$ is approximately $n \times m^2 + O(n^3)$. In addition, each iteration makes at least one call of user-supplied (sub)program **lsfun2** and some iterations may call user-supplied (sub)program **lshes2**. So, unless the residuals and their derivatives can be evaluated very quickly, the run time will be dominated by the time spent in **lsfun2** (and, to a lesser extent, in **lshes2**).

Ideally, the problem should be scaled so that the minimum value of the sum of squares is in the range $(0,+1)$ and so that at points a unit distance away from the solution the sum of squares is approximately a unit value greater than at the minimum. It is unlikely that you will be able to follow these recommendations very closely, but it is worth trying (by guesswork), as sensible scaling will reduce the difficulty of the minimization problem, so that e04hy will take less computer time.

When the sum of squares represents the goodness-of-fit of a nonlinear model to observed data, elements of the variance-covariance matrix of the estimated regression coefficients can be computed by a subsequent call to e04yc, using information returned in segments of the workspace array **w**. See e04yc for further details.

9 Example

e04hy_lsfun2.m

```
function [fvecc, fjacc, user] = lsfun2(m, n, xc, ljc, user)
  fvecc = zeros(m, 1);
  fjacc = zeros(ljc, n);

  for i = 1:m
   denom = xc(2)*user{2}(i,2) + xc(3)*user{2}(i,3);
   fvecc(i) = xc(1) + user{2}(i,1)/denom - user{1}(i);
   fjacc(i,1) = 1.0d0;
   dummy = -1.0d0/(denom*denom);
   fjacc(i,2) = user{2}(i,1)*user{2}(i,2)*dummy;
   fjacc(i,3) = user{2}(i,1)*user{2}(i,3)*dummy;
  end
```

e04hy_lshes2.m

```
function [b, user] = lshes2(m, n, fvecc, xc, lb, user)
  b = zeros(lb, 1);

  sum22 = 0.0d0;
```

```
   sum32 = 0.0d0;
   sum33 = 0.0d0;
   for i = 1:m
                                  dummy        =        2.0d0*user{2}(i,1)/
(xc(2)*user{2}(i,2)+xc(3)*user{2}(i,3))^3;
     sum22 = sum22 + fvecc(i)*dummy*user{2}(i,2)^2;
     sum32 = sum32 + fvecc(i)*dummy*user{2}(i,2)*user{2}(i,3);
     sum33 = sum33 + fvecc(i)*dummy*user{2}(i,3)^2;
   end
   b(3) = sum22;
   b(5) = sum32;
   b(6) = sum33;
```

```
m = int32(15);
x = [0.5;
     1;
     1.5];

x = [0.5;
     1;
     1.5];
y     =     [0.14,0.18,0.22,0.25,0.29,0.32,0.35,0.39,0.37,0.58,0.73,0.96,
1.34,2.10,4.39];
t = [[1.0, 15.0, 1.0],
     [2.0, 14.0, 2.0],
     [3.0, 13.0, 3.0],
     [4.0, 12.0, 4.0],
     [5.0, 11.0, 5.0],
     [6.0, 10.0, 6.0],
     [7.0, 9.0, 7.0],
     [8.0, 8.0, 8.0],
     [9.0, 7.0, 7.0],
     [10.0, 6.0, 6.0],
     [11.0, 5.0, 5.0],
     [12.0, 4.0, 4.0],
     [13.0, 3.0, 3.0],
     [14.0, 2.0, 2.0],
     [15.0, 1.0, 1.0]];

user = {y, t, 3};

[xOut, fsumsq, user, ifail] = e04hy(m, 'e04hy_lsfun2', 'e04hy_lshes2', x,
'user', user)
```

```
xOut =
    0.0824
    1.1330
    2.3437
fsumsq =
    0.0082
user =
    [1x15 double]    [15x3 double]    [3]
ifail =
           0
```