NAG Toolbox

nag_opt_lsq_check_deriv (e04ya)

1 Purpose

nag_opt_lsq_check_deriv (e04ya) checks that a user-supplied function for evaluating a vector of functions and the matrix of their first derivatives produces derivative values which are consistent with the function values calculated.

2 Syntax

```
[fvec, fjac, user, ifail] = nag_opt_lsq_check_deriv(m, lsqfun, x, 'n', n, 'user', user)
[fvec, fjac, user, ifail] = e04ya(m, lsqfun, x, 'n', n, 'user', user)
```

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 24: w and **iw** were removed from the interface; **user** was added to the interface

At Mark 22: liw and lw were removed from the interface.

3 Description

Routines for minimizing a sum of squares of m nonlinear functions (or 'residuals'), $f_i(x_1, x_2, \ldots, x_n)$, for $i = 1, 2, \ldots, m$ and $m \geq n$, may require you to supply a function to evaluate the f_i and their first derivatives. nag_opt_lsq_check_deriv (e04ya) checks the derivatives calculated by such user-supplied functions, e.g., functions of the form required for nag_opt_lsq_uncon_quasi_deriv_comp (e04gb), nag_opt_lsq_uncon_mod_deriv_comp (e04gd) and nag_opt_lsq_uncon_mod_deriv2_comp (e04he). As well as the function to be checked (**lsqfun**), you must supply a point $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ at which the check will be made. nag_opt_lsq_check_deriv (e04ya) is essentially identical to CHKLSJ in the NPL Algorithms Library.

nag_opt_lsq_check_deriv (e04ya) first calls **lsqfun** to evaluate the $f_i(x)$ and their first derivatives, and uses these to calculate the sum of squares $F(x) = \sum_{i=1}^{m} [f_i(x)]^2$, and its first derivatives $g_j = \left.\dfrac{\partial F}{\partial x_j}\right|_x$, for $j = 1, 2, \ldots, n$. The components of g along two orthogonal directions (defined by unit vectors p_1 and p_2, say) are then calculated; these will be $g^{\mathrm{T}}p_1$ and $g^{\mathrm{T}}p_2$ respectively. The same components are also estimated by finite differences, giving quantities

$$v_k = \frac{F(x + hp_k) - F(x)}{h}, \quad k = 1, 2$$

where h is a small positive scalar. If the relative difference between v_1 and $g^{\mathrm{T}}p_1$ or between v_2 and $g^{\mathrm{T}}p_2$ is judged too large, an error indicator is set.

4 References

None.

5 Parameters

5.1 Compulsory Input Parameters

1: **m** – INTEGER

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: **lsqfun** – SUBROUTINE, supplied by the user.

lsqfun must calculate the vector of values $f_i(x)$ and their first derivatives $\dfrac{\partial f_i}{\partial x_j}$ at any point x. (The minimization functions mentioned in Section 3 give you the option of resetting a argument to terminate immediately. nag_opt_lsq_check_deriv (e04ya) will also terminate immediately, without finishing the checking process, if the argument in question is reset.)

```
[iflag, fvec, fjac, user] = lsqfun(iflag, m, n, xc, ldfjac, user)
```

Input Parameters

1: **iflag** – INTEGER

To **lsqfun**, **iflag** will be set to 2.

2: **m** – INTEGER

The numbers m of residuals.

3: **n** – INTEGER

The numbers n of variables.

4: **xc**(**n**) – REAL (KIND=nag_wp) array

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

5: **ldfjac** – INTEGER

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

6: **user** – INTEGER array

lsqfun is called from nag_opt_lsq_check_deriv (e04ya) with the object supplied to nag_opt_lsq_check_deriv (e04ya).

Output Parameters

1: **iflag** – INTEGER

If you reset **iflag** to some negative number in **lsqfun** and return control to nag_opt_lsq_check_deriv (e04ya), the function will terminate immediately with **ifail** set to your setting of **iflag**.

2: **fvec**(**m**) – REAL (KIND=nag_wp) array

Unless **iflag** is reset to a negative number, **fvec**(i) must contain the value of f_i at the point x, for $i = 1, 2, \ldots, m$.

3: **fjac**$(ldfjac, \mathbf{n})$ – REAL (KIND=nag_wp) array

Unless **iflag** is reset to a negative number, **fjac**(i, j) must contain the value of $\dfrac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

4: **user** – INTEGER array

3: **x**$(\mathbf{n})$ – REAL (KIND=nag_wp) array

x(j), for $j = 1, 2, \ldots, n$, must be set to the coordinates of a suitable point at which to check the derivatives calculated by **lsqfun**. 'Obvious' settings, such as 0 or 1, should not be used since, at such particular points, incorrect terms may take correct values (particularly zero), so that errors can go undetected. For a similar reason, it is preferable that no two elements of **x** should have the same value.

5.2 Optional Input Parameters

1: **n** – INTEGER

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** – INTEGER array

user is not used by nag_opt_lsq_check_deriv (e04ya), but is passed to **lsqfun**. 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 Output Parameters

1: **fvec**$(\mathbf{m})$ – REAL (KIND=nag_wp) array

Unless you set **iflag** negative in the first call of **lsqfun**, **fvec**(i) contains the value of f_i at the point supplied by you in **x**, for $i = 1, 2, \ldots, m$.

2: **fjac**$(ldfjac, \mathbf{n})$ – REAL (KIND=nag_wp) array

Unless you set **iflag** negative in the first call of **lsqfun**, **fjac**(i, j) contains the value of the first derivative $\dfrac{\partial f_i}{\partial x_j}$ at the point given in **x**, as calculated by **lsqfun**, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

3: **user** – INTEGER array

4: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Note: nag_opt_lsq_check_deriv (e04ya) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail < 0 (*warning*)

A negative value of **ifail** indicates an exit from nag_opt_lsq_check_deriv (e04ya) because you have set **iflag** negative in **lsqfun**. The setting of **ifail** will be the same as your setting of **iflag**. The check on **lsqfun** will not have been completed.

ifail $= 1$

On entry, $\mathbf{m} < \mathbf{n}$,
or $\mathbf{n} < 1$,
or $ldfjac < \mathbf{m}$,
or $liw < 1$,
or $lw < 3 \times \mathbf{n} + \mathbf{m} + \mathbf{m} \times \mathbf{n}$.

ifail $= 2$ (*warning*)

You should check carefully the derivation and programming of expressions for the $\dfrac{\partial f_i}{\partial x_j}$, because it is very unlikely that **lsqfun** is calculating them correctly.

ifail $= -99$

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

ifail $= -399$

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

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

ifail is set to 2 if

$$\left(v_k - g^{\mathrm{T}} p_k\right)^2 \geq h \times \left(\left(g^{\mathrm{T}} p_k\right)^2 + 1\right)$$

for $k = 1$ or 2. (See Section 3 for definitions of the quantities involved.) The scalar h is set equal to $\sqrt{\epsilon}$, where ϵ is the ***machine precision*** as given by nag_machine_precision (x02aj).

8 Further Comments

nag_opt_lsq_check_deriv (e04ya) calls **lsqfun** three times.

Before using nag_opt_lsq_check_deriv (e04ya) to check the calculation of the first derivatives, you should be confident that **lsqfun** is calculating the residuals correctly.

nag_opt_lsq_check_deriv (e04ya) only checks the derivatives calculated by a user-supplied function when **iflag** $= 2$. So, if **lsqfun** is intended for use in conjunction with a minimization function which may set **iflag** to 1, you must check that, for given settings of the $\mathbf{xc}(j)$, **lsqfun** produces the same values for the $\dfrac{\partial f_i}{\partial x_j}$ when **iflag** is set to 1 as when **iflag** is set to 2.

9 Example

Suppose that it is intended to use nag_opt_lsq_uncon_quasi_deriv_comp (e04gb) or nag_opt_lsq_uncon_mod_deriv_comp (e04gd) to find least squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table.

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

The following program could be used to check the first derivatives calculated by **lsqfun**. (The tests of whether **iflag** $= 0$ or 1 in **lsqfun** are present ready for when **lsqfun** is called by nag_opt_lsq_uncon_quasi_deriv_comp (e04gb) or nag_opt_lsq_uncon_mod_deriv_comp (e04gd). nag_opt_lsq_check_deriv (e04ya) will always call **lsqfun** with **iflag** set to 2.)

9.1 Program Text

```
     function e04ya_example

fprintf('e04ya example results\n\n');

m = nag_int(15);
x = [0.19; -1.34;  0.88];
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};

[fvec, fjac, user, ifail] = e04ya( ...
                                  m, @lsqfun, x, 'user', user);
fprintf('The test point is:\n');
fprintf('%9.5f',x);
fprintf('\n\n1st derivatives are consistent with residual values\n\n');
fprintf('At the test point, lsqfun gives:\n\n');
```

```
fprintf('  Residuals         1st derivatives\n');
fprintf('%10.4f  %10.4f%10.4f%10.4f\n',[fvec fjac]');
fprintf('\n');

function [iflag, fvecc, fjacc, user] = lsqfun(iflag, m, n, xc, ljc, user)
  y = user{1};
  t = user{2};

  fvecc = zeros(m, 1);
  fjacc = zeros(ljc, n);

  for i = 1:double(m)
    denom = xc(2)*t(i,2) + xc(3)*t(i,3);
    if (iflag ~= 1)
      fvecc(i) = xc(1) + t(i,1)/denom - y(i);
    end
    if (iflag ~= 0)
      fjacc(i,1) = 1;
      dummy = -1/(denom*denom);
      fjacc(i,2) = t(i,1)*t(i,2)*dummy;
      fjacc(i,3) = t(i,1)*t(i,3)*dummy;
    end
  end
```

9.2 Program Results

```
     e04ya example results

The test point is:
  0.19000 -1.34000  0.88000

1st derivatives are consistent with residual values

At the test point, lsqfun gives:

  Residuals         1st derivatives
   -0.0020      1.0000   -0.0406   -0.0027
   -0.1076      1.0000   -0.0969   -0.0138
   -0.2330      1.0000   -0.1785   -0.0412
   -0.3785      1.0000   -0.3043   -0.1014
   -0.5836      1.0000   -0.5144   -0.2338
   -0.8689      1.0000   -0.9100   -0.5460
   -1.3464      1.0000   -1.8098   -1.4076
   -2.3739      1.0000   -4.7259   -4.7259
   -2.9750      1.0000   -6.0762   -6.0762
   -4.0132      1.0000   -7.8765   -7.8765
   -5.3226      1.0000  -10.3970  -10.3970
   -7.2917      1.0000  -14.1777  -14.1777
  -10.5703      1.0000  -20.4789  -20.4789
  -17.1274      1.0000  -33.0813  -33.0813
  -36.8087      1.0000  -70.8885  -70.8885
```