nag_opt_lsq_check_deriv (e04yac)

1. Purpose

nag_opt_lsq_check_deriv checks that a user-supplied C 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. Specification

```c
#include <nag.h>
#include <nage04.h>

void nag_opt_lsq_check_deriv(Integer m, Integer n,
    void (*lsqfun)(Integer m, Integer n, double x[], double fvec[],
                    double fjac[], Integer tdj, Nag_Comm *comm),
    double x[], double fvec[], double fjac[], Integer tdj,
    Nag_Comm *comm, NagError *fail)
```

3. Description

The function nag_opt_lsq_deriv (e04gbc) for minimizing a sum of squares of \( m \) nonlinear functions (or ‘residuals’), \( f_i(x_1, x_2, \ldots, x_n) \), for \( i = 1, 2, \ldots, m; \ m \geq n \), requires the user to supply a C function to evaluate the \( f_i \) and their first derivatives. nag_opt_lsq_check_deriv checks the derivatives calculated by such a user-supplied function. As well as the C function to be checked (lsqfun), the user must supply a point \( x = (x_1, x_2, \ldots, x_n)^T \) at which the check is to be made.

nag_opt_lsq_check_deriv 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 \( g_j = \frac{\partial f_i}{\partial x_j} \bigg|_{x} \), 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 \( g^T p_1 \) and \( g^T p_2 \) 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 \( g^T p_1 \) or between \( v_2 \) and \( g^T p_2 \) is judged too large, an error indicator is set.

4. Parameters

\[ m \]

\[ n \]

Input: the number \( m \) of residuals, \( f_i(x) \), and the number \( n \) of variables, \( x_j \).

Constraint: \( 1 \leq n \leq m \).

\[ \text{lsqfun} \]

lsqfun must calculate the vector of values \( f_i(x) \) and their first derivatives \( \frac{\partial f_i}{\partial x_j} \) at any point \( x \). (The minimization routine nag_opt_lsq_deriv (e04gbc) gives the user the option of resetting a parameter, comm->flag, to terminate the minimization process immediately. nag_opt_lsq_check_deriv will also terminate immediately, without finishing the checking process, if the parameter in question is reset to a negative value.)

The specification of lsqfun is:

[NP3275/5/pdf] 3.e04yac.1
void lsqfun(Integer m, Integer n, double x[], double fvec[], double fjac[], Integer tdj, Nag_Comm *comm),

  m
  n
  Input: the numbers m and n of residuals and variables, respectively.

  x[n]
  Input: the point x at which the values of the $f_i$ and the $\frac{\partial f_i}{\partial x_j}$ are required.

  fvec[m]
  Output: unless \texttt{comm->flag} is reset to a negative number, then \texttt{fvec[i - 1]} must contain the value of $f_i$ at the point x, for $i = 1, 2, \ldots, m$.

  fjac[m*tdj]
  Output: unless \texttt{comm->flag} is reset to a negative number, then the value in \texttt{fjac[(i - 1)*tdj+j - 1]} must be the first derivative $\frac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, m; j = 1, 2, \ldots, n$.

  tdj
  Input: the last dimension of the array \texttt{fjac} as declared in the function from which \texttt{nag_opt_lsq_check_deriv} is called.

  comm
  Pointer to structure of type Nag_Comm; the following members are relevant to \texttt{lsqfun}.

    flag – Integer
      Input: \texttt{comm->flag} will be set to 2.
      Output: if \texttt{lsqfun} resets \texttt{comm->flag} to some negative number then \texttt{nag_opt_lsq_check_deriv} will terminate immediately with the error indicator \texttt{NE_USER_STOP}. If \texttt{fail} is supplied to \texttt{nag_opt_lsq_check_deriv}, \texttt{fail.errnum} will be set to the user’s setting of \texttt{comm->flag}.

    first – Boolean
      Input: will be set to TRUE on the first call to \texttt{lsqfun} and FALSE for all subsequent calls.

    nf – Integer
      Input: the number of calls made to \texttt{lsqfun} including the current one.

    user – double *
    iuser – Integer *
    p – Pointer
      The type Pointer will be void * with a C compiler that defines void * and char * otherwise.
      Before calling \texttt{nag_opt_lsq_check_deriv} these pointers may be allocated memory by the user and initialised with various quantities for use by \texttt{lsqfun} when called from \texttt{nag_opt_lsq_check_deriv}.

The array \texttt{x} must not be changed within \texttt{lsqfun}.

\texttt{x[n]}
Input: $x[j - 1]$ ($j = 1, 2, \ldots, n$) must be set to the co-ordinates of a suitable point at which to check the derivatives calculated by \texttt{lsqfun}. ‘Obvious’ settings, such as 0.0 or 1.0, 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 \texttt{x} should have the same value.

\texttt{fvec[m]}
Output: unless \texttt{comm->flag} is set negative in the first call of \texttt{lsqfun}, \texttt{fvec[i - 1]} contains the value of $f_i$ at the point given in \texttt{x}, for $i = 1, 2, \ldots, m$. 
fjac[m][tdj]
Output: unless comm->flag is set negative in the first call of lsqfun, fjac[i−1][j−1] contains the value of the first derivative ∂fi/∂xj at the point given in x, as calculated by lsqfun, for i = 1, 2, …, m; j = 1, 2, …, n.

tdj
Input: the second dimension of the array fjac as declared in the function from which nag_opt_lsq_check_deriv is called.
Constraint: tdj ≥ n.

comm
Input/Output: structure containing pointers for communication to the user defined function; see the above description of lsqfun for details. If the user does not need to make use of this communication feature the null pointer NAGCOMM_NULL may be used in the call to nag_opt_lsq_check_deriv; comm will then be declared internally for use in calls to lsqfun.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings
NE_USER_STOP
User requested termination, user flag value = ⟨value⟩.
This exit occurs if the user sets comm->flag to a negative value in lsqfun. If fail is supplied the value of fail.errnum will be the same as the user’s setting of comm->flag. The check on lsqfun will not have been completed.

NE_INT_ARG_LT
On entry, n must not be less than 1: n = ⟨value⟩.

NE_2_INT_ARG_LT
On entry, m = ⟨value⟩ while n = ⟨value⟩. These parameters must satisfy m ≥ n.
On entry, tdj = ⟨value⟩ while n = ⟨value⟩. These parameters must satisfy tdj ≥ n.

NE_ALLOC_FAIL
Memory allocation failed.

NE_DERIV_ERRORS
Large errors were found in the derivatives of the objective function.
The user should check carefully the derivation and programming of expressions for the ∂fi/∂xj because it is very unlikely that lsqfun is calculating them correctly.

6. Further Comments
nag_opt_lsq_check_deriv calls lsqfun three times.
Before using nag_opt_lsq_check_deriv to check the calculation of the first derivatives, the user should be confident that lsqfun is calculating the residuals correctly.

6.1. Accuracy
fail.code is set to NE_DERIV_ERRORS if
(vk − gTwk)2 ≥ h × ((gTwk)2 + 1)
for k = 1 or 2. (See Section 3 for definitions of the quantities involved.) The scalar h is set equal to √ε, where ε is the machine precision as given by nag_machine_precision (X02AJC).

7. See Also
nag_opt_lsq_deriv (e04gbc)
8. Example

Suppose that it is intended to use \texttt{nag\_opt\_lsq\_deriv (e04gbc)} 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:

<table>
<thead>
<tr>
<th>$y$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.14</td>
<td>1.0</td>
<td>15.0</td>
<td>1.0</td>
</tr>
<tr>
<td>0.18</td>
<td>2.0</td>
<td>14.0</td>
<td>2.0</td>
</tr>
<tr>
<td>0.22</td>
<td>3.0</td>
<td>13.0</td>
<td>3.0</td>
</tr>
<tr>
<td>0.25</td>
<td>4.0</td>
<td>12.0</td>
<td>4.0</td>
</tr>
<tr>
<td>0.29</td>
<td>5.0</td>
<td>11.0</td>
<td>5.0</td>
</tr>
<tr>
<td>0.32</td>
<td>6.0</td>
<td>10.0</td>
<td>6.0</td>
</tr>
<tr>
<td>0.35</td>
<td>7.0</td>
<td>9.0</td>
<td>7.0</td>
</tr>
<tr>
<td>0.39</td>
<td>8.0</td>
<td>8.0</td>
<td>8.0</td>
</tr>
<tr>
<td>0.37</td>
<td>9.0</td>
<td>7.0</td>
<td>7.0</td>
</tr>
<tr>
<td>0.58</td>
<td>10.0</td>
<td>6.0</td>
<td>6.0</td>
</tr>
<tr>
<td>0.73</td>
<td>11.0</td>
<td>5.0</td>
<td>5.0</td>
</tr>
<tr>
<td>0.96</td>
<td>12.0</td>
<td>4.0</td>
<td>4.0</td>
</tr>
<tr>
<td>1.34</td>
<td>13.0</td>
<td>3.0</td>
<td>3.0</td>
</tr>
<tr>
<td>2.10</td>
<td>14.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>4.39</td>
<td>15.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The following program could be used to check the first derivatives calculated by the required function \texttt{lsqfun}. (The tests of whether \texttt{comm->flag} $\neq$ 0 or 1 in \texttt{lsqfun} are present for when \texttt{lsqfun} is called by \texttt{nag\_opt\_lsq\_deriv (e04gbc)}. \texttt{nag\_opt\_lsq\_check\_deriv} will always call \texttt{lsqfun} with \texttt{comm->flag} set to 2.)

8.1. Program Text

```c
/* nag_opt_lsq_check_deriv (e04yac) Example Program */  
*  
* Mark 2, 1991. */  

#include <nag.h>  
#include <stdio.h>  
#include <nag_stdlib.h>  
#include <nage04.h>  

#ifdef NAG_PROTO  
static void lsqfun(Integer m, Integer n, double x[], double fvec[], double fjac[], Integer tdj, Nag_Comm *comm);  
#else  
static void lsqfun();  
#endif  

main()  
{  
#define MMAX 15  
#define NMAX 3  
#define Y(I) comm.user[I]  
#define T(I,J) comm.user[(I)*NMAX + (J) + MMAX]  

double fjac[MMAX][NMAX], fvec[MMAX], x[NMAX];  
double work[MMAX + NMAX*NMAX];  
Integer i, j, m, n, tdj;  
Nag_Comm comm;  
static NagError fail;  
```
Vprintf("e04yac Example Program Results\n");
Vscanf(" %*[\n]"); /* Skip heading in data file */

n = 3;
m = 15;
tdj = NMAX;
fail.print = TRUE;

/* Allocate memory to communication array */
comm.user = work;

/* Observations t (j = 0, 1, 2) are held in T(i, j)
   *( i = 0, 1, 2, ..., 14 )* /
for (i = 0; i < m; ++i)
{
    Vscanf("%lf", &Y(i));
    for (j = 0; j < n; ++j) Vscanf("%lf", &T(i,j));
}

/* Set up an arbitrary point at which to check the 1st derivatives */
x[0] = 0.19;
x[1] = -1.34;
x[2] = 0.88;
Vprintf("\nThe test point is ");
for (j = 0; j < n; ++j)
    Vprintf(" %9.3e", x[j]);
Vprintf("\n");

fail.print = TRUE;
e04yac(m, n, lsqfun, x, fvec, (double *)fjac, tdj, &comm, &fail);
if (fail.code != NE_NOERROR) exit(EXIT_FAILURE);

Vprintf("\nDerivatives are consistent with residual values.\n");
Vprintf("\nAt the test point, lsqfun() gives\n\n");
Vprintf(" Residuals 1st derivatives\n");
for (i = 0; i < m; ++i)
{
    Vprintf("%9.3e ", fvec[i]);
    for (j = 0; j < n; ++j)
        Vprintf(" %9.3e", fjac[i][j]);
    Vprintf("\n");
}
exit(EXIT_SUCCESS);

#endif
#endif
static void lsqfun(Integer m, Integer n, double x[], double fvec[],
                    double fjac[], Integer tdj, Nag_Comm *comm)
static void lsqfun(m, n, x, fvec, fjac, tdj, comm) 
{ /* Function to evaluate the residuals and their 1st derivatives. */ 

#define YC(I) comm->user[(I)]
#define TC(I,J) comm->user[(I)*NMAX + (J) + MMAX]
#define FJAC(I,J) fjac[(I)*tdj + (J)]

    Integer i;
    double denom, dummy;
    for (i = 0; i < m; ++i)
    {
        denom = x[1]*TC(i,1) + x[2]*TC(i,2);
        
    }
if (comm->flag != 1)
    fvec[i] = x[0] + TC(i,0)/denom - YC(i);
if (comm->flag != 0)
{
    PJAC(i,0) = 1.0;
    dummy = -1.0 / (denom * denom);
    PJAC(i,1) = TC(i,0)*TC(i,1)*dummy;
    PJAC(i,2) = TC(i,0)*TC(i,2)*dummy;
}
} /* lsqfun */

8.2. Program Data

e04yac Example Program Data

| 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 |

8.3. Program Results

e04yac Example Program Results

The test point is 1.900e-01 -1.340e+00 8.800e-01

Derivatives are consistent with residual values.

At the test point, lsqfun() gives

<table>
<thead>
<tr>
<th>Residuals</th>
<th>1st derivatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2.029e-03</td>
<td>1.000e+00</td>
</tr>
<tr>
<td>-1.076e+01</td>
<td>-4.061e-02</td>
</tr>
<tr>
<td>-2.330e-01</td>
<td>1.000e+00</td>
</tr>
<tr>
<td>-3.785e-01</td>
<td>-2.707e-03</td>
</tr>
<tr>
<td>-5.836e-01</td>
<td>-1.014e-01</td>
</tr>
<tr>
<td>-8.689e-01</td>
<td>-5.460e-01</td>
</tr>
<tr>
<td>-1.346e+00</td>
<td>-1.408e+00</td>
</tr>
<tr>
<td>-2.374e+00</td>
<td>-7.292e-00</td>
</tr>
<tr>
<td>-2.975e+00</td>
<td>-7.876e+00</td>
</tr>
<tr>
<td>-4.013e+00</td>
<td>-1.040e+01</td>
</tr>
<tr>
<td>-5.323e+00</td>
<td>-1.418e+01</td>
</tr>
<tr>
<td>-7.292e+00</td>
<td>-3.308e+01</td>
</tr>
<tr>
<td>-1.057e+01</td>
<td>-7.089e+01</td>
</tr>
<tr>
<td>-1.713e+01</td>
<td>-7.089e+01</td>
</tr>
<tr>
<td>-3.681e+01</td>
<td>-7.089e+01</td>
</tr>
</tbody>
</table>