NAG Library Function Document

nag_opt_lsq_check_deriv (e04yac)

1 Purpose

nag_opt_lsq_check_deriv (e04yac) 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

```
#include <nag.h>
#include <nage04.h>
void nag_opt_lsq_check_deriv (Integer m, Integer n,
     void (*lsqfun)(Integer m, Integer n, const double x[], double fvec[],
          double fjac[], Integer tdfjac, Nag_Comm *comm),
     const double x[], double fvec[], double fjac[], Integer tdfjac,
     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 $i = 1, 2, \ldots, m$ and $m \geq n$, requires you to supply a C function to evaluate the f_i and their first derivatives. nag_opt_lsq_check_deriv (e04yac) checks the derivatives calculated by such a user-supplied function. As well as the C function to be checked (**lsqfun**), you must supply a point $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ at which the check is to be made.

nag_opt_lsq_check_deriv (e04yac) 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 $g_j = \dfrac{\partial f}{\partial x_j|_x}$, 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 Arguments

1: **m** – Integer *Input*

2: **n** – Integer *Input*

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

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

3: **lsqfun** – function, supplied by the user *External Function*

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 function nag_opt_lsq_deriv (e04gbc) gives you the option of resetting an argument, **comm**→**flag**, to terminate the minimization process immediately. nag_opt_lsq_check_deriv (e04yac) will also terminate immediately, without finishing the checking process, if the argument in question is reset to a negative value.)

The specification of **lsqfun** is:

```
void lsqfun (Integer m, Integer n, const double x[], double fvec[],
     double fjac[], Integer tdfjac, Nag_Comm *comm)
```

1: **m** – Integer *Input*

2: **n** – Integer *Input*

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

3: **x**[**n**] – const double *Input*

On entry: the point x at which the values of the f_i and the $\frac{\partial f_i}{\partial x_j}$ are required.

4: **fvec**[**m**] – double *Output*

On exit: unless **comm**→**flag** is reset to a negative number, then **fvec**$[i-1]$ must contain the value of f_i at the point x, for $i = 1, 2, \dots, m$.

5: **fjac**[**m** × **tdfjac**] – double *Output*

On exit: unless **comm**→**flag** is reset to a negative number, then the value in **fjac**$[(i-1) \times$ **tdfjac** $+ j - 1]$ must be the first derivative $\frac{\partial f_i}{\partial x_j}$ at the point **x**, for $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$.

6: **tdfjac** – Integer *Input*

On entry: the stride separating matrix column elements in the array **fjac**.

7: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **lsqfun**.

flag – Integer *Input/Output*

On entry: **comm**→**flag** will be set to 2.

On exit: if **lsqfun** resets **comm**→**flag** to some negative number then nag_opt_lsq_check_deriv (e04yac) will terminate immediately with the error indicator NE_USER_STOP. If **fail** is supplied to nag_opt_lsq_check_deriv (e04yac), **fail**.**errnum** will be set to your setting of **comm**→**flag**.

first – Nag_Boolean *Input*

On entry: will be set to Nag_TRUE on the first call to **lsqfun** and Nag_FALSE for all subsequent calls.

nf – Integer *Input*

On entry: the number of calls made to **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 nag_opt_lsq_check_deriv (e04yac) these pointers may be allocated memory and initialized with various quantities for use by **lsqfun** when called from nag_opt_lsq_check_deriv (e04yac).

The array **x** must **not** be changed within **lsqfun**.

4: **x**[**n**] – const double *Input*

On entry: **x**$[j-1]$, 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 0.0 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 **x** should have the same value.

5: **fvec**[**m**] – double *Output*

On exit: unless **comm→flag** is set negative in the first call of **lsqfun**, **fvec**$[i-1]$ contains the value of f_i at the point given in **x**, for $i = 1, 2, \ldots, m$.

6: **fjac**[**m** × **tdfjac**] – double *Output*

On exit: unless **comm→flag** is set negative in the first call of **lsqfun**, **fjac**$[(i-1) \times$ **tdfjac** $+ j - 1]$ 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 $j = 1, 2, \ldots, n$.

7: **tdfjac** – Integer *Input*

On entry: the stride separating matrix column elements in the array **fjac**.

Constraint: **tdfjac** $\geq$ **n**.

8: **comm** – Nag_Comm * *Input/Output*

Note: **comm** is a NAG defined type (see Section 3.2.1.1 in the Essential Introduction).

On entry/exit: structure containing pointers for communication to the user-defined function; see the above description of **lsqfun** for details. If you do 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 (e04yac); **comm** will then be declared internally for use in calls to **lsqfun**.

9: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_2_INT_ARG_LT

On entry, **m** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These arguments must satisfy **m** $\geq$ **n**.

On entry, **tdfjac** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These arguments must satisfy **tdfjac** $\geq$ **n**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_DERIV_ERRORS

Large errors were found in the derivatives of the objective function. You should check carefully the derivation and programming of expressions for the $\frac{\partial f_i}{\partial x_j}$, because it is very unlikely that **lsqfun** is calculating them correctly.

NE_INT_ARG_LT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 1$.

NE_USER_STOP

User requested termination, user flag value $= \langle value \rangle$.

This exit occurs if you set **comm→flag** to a negative value in **lsqfun**. If **fail** is supplied the value of **fail.errnum** will be the same as your setting of **comm→flag**. The check on **lsqfun** will not have been completed.

7 Accuracy

fail.code is set to NE_DERIV_ERRORS 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 (X02AJC).

8 Parallelism and Performance

Not applicable.

9 Further Comments

nag_opt_lsq_check_deriv (e04yac) calls **lsqfun** three times.

Before using nag_opt_lsq_check_deriv (e04yac) to check the calculation of the first derivatives, you should be confident that **lsqfun** is calculating the residuals correctly.

10 Example

Suppose that it is intended to use 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:

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 the required function **lsqfun**. (The tests of whether **comm**→**flag** $\neq 0$ or 1 in **lsqfun** are present for when **lsqfun** is called by nag_opt_lsq_deriv (e04gbc). nag_opt_lsq_check_deriv (e04yac) will always call **lsqfun** with **comm**→**flag** set to 2.)

10.1 Program Text

```
/* nag_opt_lsq_check_deriv (e04yac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

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

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL lsqfun(Integer m, Integer n, const double x[],
                            double fvec[], double fjac[], Integer tdfjac,
                            Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

#define Y(I)        comm.user[I]
#define T(I, J)     comm.user[(I) *n + (J) + m]
#define YC(I)       comm->user[(I)]
#define TC(I, J)    comm->user[(I) *n + (J) + m]
#define FJAC(I, J) fjac[(I) *tdfjac + (J)]

int main(void)
{
  Integer  exit_status = 0, i, j, m, n, tdfjac;
  NagError fail;
  Nag_Comm comm;
  double   *fjac = 0, *fvec = 0, *work = 0, *x = 0;

  INIT_FAIL(fail);

  printf("nag_opt_lsq_check_deriv (e04yac) Example Program Results\n");
```

```
#ifdef _WIN32
  scanf_s(" %*[^\n]"); /* Skip heading in data file */
#else
  scanf(" %*[^\n]"); /* Skip heading in data file */
#endif

  n = 3;
  m = 15;
  if (n >= 1 && m >= 1 && n <= m)
    {
      if (!(fjac = NAG_ALLOC(m*n, double)) ||
          !(fvec = NAG_ALLOC(m, double)) ||
          !(x = NAG_ALLOC(n, double)) ||
          !(work = NAG_ALLOC(m + m*n, double))
          )
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
      tdfjac = n;
    }
  else
    {
      printf("Invalid n or m.\n");
      exit_status = 1;
      return exit_status;
    }

  /* 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)
    {
#ifdef _WIN32
      scanf_s("%lf", &Y(i));
#else
      scanf("%lf", &Y(i));
#endif
#ifdef _WIN32
      for (j = 0; j < n; ++j) scanf_s("%lf", &T(i, j));
#else
      for (j = 0; j < n; ++j) scanf("%lf", &T(i, j));
#endif
    }

  /* Set up an arbitrary point at which to check the 1st derivatives */
  x[0] = 0.19;
  x[1] = -1.34;
  x[2] = 0.88;
  printf("\nThe test point is ");
  for (j = 0; j < n; ++j)
    printf(" %12.3e", x[j]);
  printf("\n");

  /* nag_opt_lsq_check_deriv (e04yac).
   * Least-squares derivative checker for use with
   * nag_opt_lsq_deriv (e04gbc)
   */
  nag_opt_lsq_check_deriv(m, n, lsqfun, x, fvec, fjac, tdfjac, &comm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_opt_lsq_check_deriv (e04yac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\nDerivatives are consistent with residual values.\n");
```

```
  printf("\nAt the test point, lsqfun() gives\n\n");
  printf("     Residuals                   1st derivatives\n");
  for (i = 0; i < m; ++i)
    {
      printf("     %12.3e  ", fvec[i]);
      for (j = 0; j < n; ++j)
        printf("     %12.3e", FJAC(i, j));
      printf("\n");
    }
 END:
  NAG_FREE(fjac);
  NAG_FREE(fvec);
  NAG_FREE(x);
  NAG_FREE(work);
  return exit_status;
}

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

  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)
        {
          FJAC(i, 0) = 1.0;
          dummy = -1.0 / (denom * denom);
          FJAC(i, 1) = TC(i, 0)*TC(i, 1)*dummy;
          FJAC(i, 2) = TC(i, 0)*TC(i, 2)*dummy;
        }
    }
}                                /* lsqfun */
```

10.2 Program Data

```
nag_opt_lsq_check_deriv (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
```

10.3 Program Results

```
nag_opt_lsq_check_deriv (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
```

```
  Residuals                       1st derivatives
 -2.029e-03       1.000e+00     -4.061e-02     -2.707e-03
 -1.076e-01       1.000e+00     -9.689e-02     -1.384e-02
 -2.330e-01       1.000e+00     -1.785e-01     -4.120e-02
 -3.785e-01       1.000e+00     -3.043e-01     -1.014e-01
 -5.836e-01       1.000e+00     -5.144e-01     -2.338e-01
 -8.689e-01       1.000e+00     -9.100e-01     -5.460e-01
 -1.346e+00       1.000e+00     -1.810e+00     -1.408e+00
 -2.374e+00       1.000e+00     -4.726e+00     -4.726e+00
 -2.975e+00       1.000e+00     -6.076e+00     -6.076e+00
 -4.013e+00       1.000e+00     -7.876e+00     -7.876e+00
 -5.323e+00       1.000e+00     -1.040e+01     -1.040e+01
 -7.292e+00       1.000e+00     -1.418e+01     -1.418e+01
 -1.057e+01       1.000e+00     -2.048e+01     -2.048e+01
 -1.713e+01       1.000e+00     -3.308e+01     -3.308e+01
 -3.681e+01       1.000e+00     -7.089e+01     -7.089e+01
```