nag_check_deriv_1 (c05zcc)

1. Purpose

nag_check_deriv_1 (c05zcc) 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 <nagc05.h>

void nag_check_deriv_1(Integer n, double x[], double fvec[], double fjac[],
                         Integer tdfjac,
                         void (*f)(Integer n, double x[], double fvec[],
                                   double fjac[], Integer tdfjac, Integer *userflag),
                         Nag_User *comm, NagError *fail)
```

3. Description

nag_check_deriv_1 checks the derivatives calculated by user-supplied C functions, e.g. functions of the form required for nag_zero_nonlin_eqns_deriv_1 (c05ubc). As well as the C function to be checked f, the user must supply a point \( x = (x_1, x_2, \ldots, x_n)^T \) at which the check will be made.

nag_check_deriv_1 first calls f to evaluate both the \( f_i(x) \) and their first derivatives, and uses these to calculate the sum of squares

\[
F(x) = \sum_{i=1}^{n} |f_i(x)|^2,
\]

and its first derivatives

\[
g_j = \left. \frac{\partial F}{\partial x_j} \right|_x, \quad \text{for } j = 1, 2, \ldots, n.
\]

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

**n**

Input: the number \( n \) of variables, \( x_j \), for use with nag_zero_nonlin_eqns_deriv_1 (c05ubc).

Constraint: \( n \geq 0 \).

**x[n]**

Input: \( x[j-1] \), for \( j = 1, 2, \ldots, n \) must be set to the co-ordinates of a suitable point at which to check the derivatives calculated by \( f \). ‘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.

**fvec[n]**

Output: unless userflag is set negative when evaluating \( f_i \) at the point given in \( x \), fvec[\( i-1 \)] contains the value of \( f_i \) at the point given by the user in \( x \), for \( i = 1, 2, \ldots, n \).
**nag_check_deriv_1**

**fjac[n][tdfjac]**

Output: unless **userflag** is set negative when evaluating the Jacobian at the point given in **x**, **fjac[i-1][j-1]** contains the value of the first derivative \( \partial f_i / \partial x_j \) at the point given in **x**, as calculated by **f**, for \( i = 1, 2, \ldots, n; j = 1, 2, \ldots, n \).

**tdfjac**

Input: the last dimension of array **fjac** as declared in the function from which **nag_check_deriv_1** is called.

Constraint: \( tdfjac \geq n \).

**f**

**f** must calculate the values of the functions at a point **x** or return the Jacobian at **x**. **nag_zero_nonlin_eqns_deriv_1** (**c05ubc**) gives the user the option of resetting a parameter to terminate immediately. **nag_check_deriv_1** will also terminate immediately, without finishing the checking process, if the parameter in question is reset.

The specification of **f** is:

```c
void f(Integer n, double x[], double fvec[], double fjac[],
        Integer tdfjac, Integer *userflag)

n
Input: the number of equations, \( n \).

x[n]
Input: the components of the point \( x \) at which the functions or the Jacobian must be evaluated.

fvec[n]
Output: if **userflag** = 1 on entry, **fvec** must contain the function values \( f_i(x) \) (unless **userflag** is set to a negative value by **f**).
If **userflag** = 2 on entry, **fvec** must not be changed.

fjac[n*tdfjac]
Output: if **userflag** = 2 on entry, \( fjac[(i-1)*tdfjac+j-1] \) must contain the value of \( \partial f_i / \partial x_j \) at the point \( x \), for \( i = 1, 2, \ldots, n; j = 1, 2, \ldots, n \) (unless **userflag** is set to a negative value by **f**).
If **userflag** = 1 on entry, **fjac** must not be changed.

tdfjac
Input: the last dimension of array **fjac** as declared in the function from which **nag_check_deriv_1** is called.

userflag
Input: **userflag** = 1 or 2.
If **userflag** = 1, **fvec** is to be updated.
If **userflag** = 2, **fjac** is to be updated.

Output: in general, **userflag** should not be reset by **f**. If, however, the user wishes to terminate execution (perhaps because some illegal point \( x \) has been reached), then **userflag** should be set to a negative integer. This value will be returned through **fail.errnum**.

**comm**

Input/Output: pointer to a structure of type Nag_User with the following member:

**p** - Pointer
Input/Output: the pointer **p**, of type Pointer, allows the user to communicate information to and from the user-defined function **f()**. An object of the required type should be declared by the user, e.g. a structure, and its address assigned to the pointer **p** by means of a cast to Pointer in the calling program, e.g. **comm.p = (Pointer)&s**. The type pointer will be void * with a C compiler that defines void * and char * otherwise.

**fail**

The NAG error parameter, see the Essential Introduction to the NAG C Library.
5. Error Indications and Warnings

**NE_INT_ARG_LE**
On entry, \(n\) must not be less or equal to 0: \(n = \langle\text{value}\rangle\).

**NE_2_INT_ARG_LT**
On entry \(t\_dfjac = \langle\text{value}\rangle\) while \(n = \langle\text{value}\rangle\). These parameters must satisfy \(t\_dfjac \geq n\).

**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 \(\partial f_i/\partial x_j\), because it is very unlikely that \(f\) is calculating them correctly.

**NE_USER_STOP**
User requested termination, user flag value = \(\langle\text{value}\rangle\).

6. Further Comments

Before using \texttt{nag\_check\_deriv\_1} to check the calculation of the first derivatives, the user should be confident that \(f\) is evaluating the functions correctly.

6.1. Accuracy

\texttt{fail.code} is set to **NE_DERIV_ERRORS** if

\[ (v_k - g^T p_k)^2 \geq h \times ((g^T p_k)^2 + 1) \]

for \(k = 1\) or 2. (See Section 3 for definitions of the quantities involved.) The scalar \(h\) is set equal to \(\sqrt{\varepsilon}\), where \(\varepsilon\) is the machine precision.

7. See Also

\texttt{nag\_zero\_nonlin\_eqns\_deriv\_1 (c05ubc)}

8. Example

This example checks the Jacobian matrix for the problem solved in the example program for \texttt{nag\_zero\_nonlin\_eqns\_deriv\_1 (c05ubc)}.

8.1. Program Text

/* \texttt{nag\_check\_deriv\_1(c05zcc) Example Program} */
* *
* *
* /

#include <nag.h>
#include <stdio.h>
#include <nag\_stdlib.h>
#include <nagc05.h>

#ifdef NAG_PROTO
static void f(Integer n, double xc[], double fvecc[],
               double fjacc[], Integer tdj, Integer *userflag, Nag_User *comm);
#else
static void f();
#endif

main()
{
#define NMAX 5


nag_check_deriv_1

double fjac[NMAX][NMAX], fvec[NMAX], x[NMAX];
Integer i, j, n, tdfjac;
static NagError fail;
Nag_User comm;

fail.print = TRUE;
Vprintf("c05zcc Example Program Results\n");
n = 3;
 tdfjac = NMAX;

/* Set up an arbitrary point at which to check the 1st derivatives */
x[0] = 9.2e-01;
x[1] = 1.3e-01;
x[2] = 5.4e-01;
Vprintf("The test point is ");
for (j=0; j<n; ++j)
  Vprintf("%13.3e", x[j]);
Vprintf("\n\n");
c05zcc(n, x, fvec, (double *)fjac, tdfjac, f, &comm, &fail);
if (fail.code != NE_NOERROR) exit(EXIT_FAILURE);
Vprintf("1st derivatives are consistent with residual values.\n\n");
Vprintf("At the test point, f() gives\n\n");
for (i=0; i<n; ++i)
  {
    Vprintf("%13.3e", fvec[i]);
    for (j=0; j<n; ++j)
      Vprintf("%13.3e", fjac[i][j]);
    Vprintf("\n");
  }
exi(EXIT_SUCCESS);

#define FJAC(I,J) fjac[((I))*tdfjac+(J)]

if (*userflag != 2)
{
  /* Calculate the function values */
  for (k=0; k<n; k++)
    { 
      fvec[k] = (3.0-x[k]*2.0) * x[k] + 1.0;
      if (k>0) fvec[k] -= x[k-1];
      if (k<n-1) fvec[k] -= x[k+1] * 2.0;
    }
}
else
{
  /\* Calculate the corresponding first derivatives */
  for (k=0; k<n; k++)
    { 
      for (j=0; j<n; j++)
        FJAC(k,j)=0.0;
      FJAC(k,k) = 3.0 - x[k] * 4.0;
      if (k>0)
        FJAC(k,k-1) = -1.0;
      if (k<n-1)
8.2. Program Data

None.

8.3. Program Results

c05zcc Example Program Results

The test point is 9.200e-01 1.300e-01 5.400e-01

1st derivatives are consistent with residual values.

At the test point, f() gives

<table>
<thead>
<tr>
<th>Residuals</th>
<th>1st derivatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.807e+00</td>
<td>-6.800e-01 -2.000e+00 0.000e+00</td>
</tr>
<tr>
<td>-6.438e-01</td>
<td>-1.000e+00 2.480e+00 -2.000e+00</td>
</tr>
<tr>
<td>1.907e+00</td>
<td>0.000e+00 -1.000e+00 8.400e-01</td>
</tr>
</tbody>
</table>