nag_opt_check_2nd_deriv (e04hdc)

1. Purpose

nag_opt_check_2nd_deriv (e04hdc) checks that a user-supplied routine for calculating second derivatives of an objective function is consistent with a user-supplied routine for calculating the corresponding first derivatives.

2. Specification

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

void nag_opt_check_2nd_deriv(Integer n,
            void (*objfun)(Integer n, double x[], double  *objf,
                           double g[], Nag_Comm *comm),
            void (*hessfun)(Integer n, double x[], double h[],
                            double hd[], Nag_Comm *comm),
            double x[], double g[], double hesl[],
            double hesd[], Nag_Comm *comm, NagError *fail)
```

3. Description

Routines for minimizing a function $F(x_1, x_2, \ldots, x_n)$ of the variables $x_1, x_2, \ldots, x_n$ may require the user to provide a subroutine to evaluate the second derivatives of F. nag_opt_check_2nd_deriv is designed to check the second derivatives calculated by such user-supplied routines. As well as the routine to be checked (**hessfun**), the user must supply a routine (**objfun**) to evaluate the first derivatives, and a point $x = (x_1, x_2, \ldots, x_n)^T$ at which the checks will be made. Note that nag_opt_check_2nd_deriv checks routines of the form required for nag_opt_bounds_2nd_deriv (e04lbc).

nag_opt_check_2nd_deriv first calls **objfun** and **hessfun** to evaluate the first and second derivatives of F at x. The user-supplied Hessian matrix (H, say) is projected onto two orthogonal vectors y and z to give the scalars $y^T Hy$ and $z^T Hz$ respectively. The same projections of the Hessian matrix are also estimated by finite differences, giving

$$p = (y^T g(x + hy) - y^T g(x))/h$$

$$\text{and} \quad q = (z^T g(x + hz) - z^T g(x))/h$$

respectively, where $g(\ )$ denotes the vector of first derivatives at the point in brackets and h is a small positive scalar. If the relative difference between p and $y^T Hy$ or between q and $z^T Hz$ is judged too large, an error indicator is set.

4. Parameters

n

Input: the number n of independent variables in the objective function.
Constraint: $\mathbf{n} \geq 1$.

objfun

objfun must evaluate the function $F(x)$ and its first derivatives $\partial F/\partial x_j$ at a specified point. (However, if the user does not wish to calculate F or its first derivatives at a particular point, there is the option of setting a parameter to cause nag_opt_check_2nd_deriv to terminate immediately.)

The specification for **objfun** is:

```
void objfun(Integer n, double x[], double *objf, double g[], Nag_Comm *comm)
```

n

Input: the number n of variables.

x[n]

Input: the point x at which the value of F, or F and the $\partial F/\partial x_j$, are required.

objf

Output: **objfun** must set **objf** to the value of the objective function F at the current point x. If it is not possible to evaluate F then **objfun** should assign a negative value to **comm->flag**; nag_opt_check_2nd_deriv will then terminate.

g[n]

Output: unless **comm->flag** is reset to a negative number, **objfun** must set **g**$[j-1]$ to the value of the first derivative $\partial F/\partial x_j$ at the current point x for $j = 1, 2, \ldots, n$.

comm

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

flag – Integer

Output: if **objfun** resets **comm->flag** to some negative number then nag_opt_check_2nd_deriv will terminate immediately with the error indicator **NE_USER_STOP**. If **fail** is supplied to nag_opt_check_2nd_deriv **fail.errnum** will be set to the user's setting of **comm->flag**.

first – Boolean

Input: will be set to **TRUE** on the first call to **objfun** and **FALSE** for all subsequent calls.

nf – Integer

Input: the number of evaluations of the objective function; this value will be equal to the number of calls made to **objfun** (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_check_2nd_deriv these pointers may be allocated memory by the user and initialized with various quantities for use by **objfun** when called from nag_opt_check_2nd_deriv.

Note: nag_opt_check_deriv (e04hcc) should be used to check the first derivatives calculated by **objfun** before nag_opt_check_2nd_deriv (e04hdc) is used to check the second derivatives, since nag_opt_check_2nd_deriv (e04hdc) assumes that the first derivatives are correct.

hessfun

hessfun must calculate the second derivatives of $F(x)$ at any point x. (As with **objfun** there is the option of causing nag_opt_check_2nd_deriv to terminate immediately.)

The specification for **hessfun** is:

```
void hessfun(Integer n, double x[], double h[], double hd[], Nag_Comm *comm)
```

n

Input: the number n of variables in the objective function.

x[n]

Input: the point x at which the second derivatives are required. $\partial F/\partial x_j$, are required.

h[]

This array is allocated internally by nag_opt_check_2nd_deriv.
Output: unless **comm->flag** is reset to a negative number **hessfun** must place the strict lower triangle of the second derivative matrix of F (evaluated at the point x) in **h**, stored by rows, i.e., set

$$\mathbf{h}[(i-1)(i-2)/2+j-1] = \left.\frac{\partial^2 F}{\partial x_i \partial x_j}\right|_{x=\mathbf{x}}, \quad \text{for } i=2,3,\ldots,n;\ j=1,2,\ldots,i-1.$$

(The upper triangle is not required because the matrix is symmetric.)

hd[n]

Input: the value of $\partial F/\partial x_j$ at the point x, for $j = 1, 2, \ldots, n$.
These values may be useful in the evaluation of the second derivatives.
Output: unless **comm->flag** is reset to a negative number **hessfun** must place the diagonal elements of the second derivative matrix of F (evaluated at the point x) in **hd**, i.e., set

$$\mathbf{hd}[j-1] = \left.\frac{\partial^2 F}{\partial {x_j}^2}\right|_{x=\mathbf{x}}, \quad \text{for } j=1,2,\ldots,n.$$

comm

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

flag – Integer
Output: if **hessfun** resets **comm->flag** to some negative number then nag_opt_check_2nd_deriv will terminate immediately with the error indicator **NE_USER_STOP**. If **fail** is supplied to nag_opt_check_2nd_deriv **fail.errnum** will be set to the user's setting of **comm->flag**.

first – Boolean
Input: will be set to **TRUE** on the first call to **hessfun** and **FALSE** for all subsequent calls.

nf – Integer
Input: the number of evaluations of the objective function; this value will be equal to the number of calls made to **hessfun** (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_check_2nd_deriv these pointers may be allocated memory by the user and initialized with various quantities for use by **hessfun** when called from nag_opt_check_2nd_deriv.

Note: The array **x** must **not** be changed by **hessfun**.

x[n]

Input: **x**$[j-1]$, for $j = 1, 2, \ldots, n$ must contain the co-ordinates of a suitable point at which to check the derivatives calculated by **objfun**. '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 could go undetected. Similarly, it is advisable that no two elements of **x** should be the same.

g[n]

Output: unless **comm->flag** is reset to a negative number **g**$[j-1]$ contains the value of the the first derivative $\partial F/\partial x_j$ at the point given in x, as calculated by **objfun** for $j = 1, 2, \ldots, n$.

hesl[n*(n-1)/2]

Output: unless **comm->flag** is reset to a negative number **hesl** contains the strict lower triangle of the second derivative matrix of F, as evaluated by **hessfun** at the point given in **x**, stored by rows.

hesd[n]

Output: unless **comm->flag** is reset to a negative number **hesd** contains the diagonal elements of the second derivative matrix of F, as evaluated by **hessfun** at the point given in **x**.

comm

Input/Output: structure containing pointers for communication to user-supplied functions; see the above description of **objfun** 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_check_2nd_deriv; **comm** will then be declared internally for use in calls to user-supplied functions.

fail

The NAG error parameter, see the Essential Introduction to the NAG C Library.
Users are recommended to declare and initialize **fail** and set **fail.print** = **TRUE** for this function.

5. Error Indications and Warnings

NE_INT_ARG_LT

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

NE_DERIV_ERRORS

Large errors were found in the derivatives of the objective function.

NE_USER_STOP

User requested termination, user flag value = ⟨*value*⟩.

NE_ALLOC_FAIL

Memory allocation failed.

6. Further Comments

nag_opt_check_2nd_deriv calls **hessfun** once and **objfun** three times.

6.1. Accuracy

The error **NE_DERIV_ERRORS** is returned if

$$|y^T Hy - p| \geq \sqrt{h} \times (|y^T Hy| + 1.0)$$
$$\text{or} \quad |z^T Hz - q| \geq \sqrt{h} \times (|z^T Hz| + 1.0)$$

where h is set equal to $\sqrt{\epsilon}$ (ϵ being the ***machine precision*** as given by nag_machine_precision (X02AJC)) and other quantities are as defined in Section 3.

6.2. References

None.

7. See Also

nag_opt_bounds_2nd_deriv (e04lbc) and nag_opt_check_deriv (e04hcc).

8. Example

Suppose that it is intended to use nag_opt_bounds_2nd_deriv (e04lbc) to minimize

$$F = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4.$$

The following program could be used to check the second derivatives calculated by the required **hessfun** function. (The call of nag_opt_check_2nd_deriv is preceded by a call of nag_opt_check_deriv (e04hcc) to check the routine **objfun** which calculates the first derivatives.)

8.1. Program Text

```
/* nag_opt_check_2nd_deriv(e04hdc) Example Program.
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 *
 */
#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <math.h>
#include <nage04.h>

#ifdef NAG_PROTO
static void hess(Integer n,  double xc[], double fhesl[],
                 double fhesd[], Nag_Comm *comm);
#else
static void hess();
#endif

#ifdef NAG_PROTO
static void funct(Integer n,  double xc[], double *fc,
                  double gc[], Nag_Comm *comm);
#else
static void funct();
#endif

main()
{
  double hesd[4];
  double hesl[6],  f;
  double g[4];
  double x[4];

  Integer n;
  Integer i, j, k;

  Nag_Comm comm;

#define X(I) x[(I)-1]
#define HESL(I) hesl[(I)-1]
#define HESD(I) hesd[(I)-1]
#define G(I) g[(I)-1]

  Vprintf("e04hdc Example Program Results\n\n");

  /*  Set up an arbitrary point at which to check the derivatives */
  n = 4;
  X(1) = 1.46;
  X(2) = -.82;
  X(3) = .57;
  X(4) = 1.21;

  Vprintf("The test point is\n");
  for (j = 1; j <= n; ++j)
    Vprintf("%9.4f", X(j));
  Vprintf("\n");
```

```
  /*  Check the 1st derivatives */
  e04hcc(n,  funct,  &X(1),  &f,  &G(1),  &comm, NAGERR_DEFAULT);

  /*  Check the 2nd derivatives */
  e04hdc(n,  funct,  hess,  &X(1),  &G(1),  &HESL(1),  &HESD(1),
         &comm, NAGERR_DEFAULT);

  Vprintf("\n2nd derivatives are consistent with 1st derivatives.\n\n");
  Vprintf("%s%12.4e\n",
          "At the test point, funct gives the function value, ", f);
  Vprintf("and the 1st derivatives\n");
  for (j = 1; j <= n; ++j)
    Vprintf("%12.3e%s", G(j), j%4?"":"\n");

  Vprintf("\nhess gives the lower triangle of the Hessian matrix\n");
  Vprintf("%12.3e\n", HESD(1));
  k = 1;
  for (i = 2; i <= n; ++i)
    {
      for (j = k; j <= k + i - 2; ++j)
        Vprintf("%12.3e", HESL(j));
      Vprintf("%12.3e\n", HESD(i));
      k = k + i - 1;
    }
  exit(EXIT_SUCCESS);
}

#ifdef NAG_PROTO
static void funct(Integer n,  double xc[], double *fc,
                  double gc[], Nag_Comm *comm)
#else
     static void funct(n, xc, fc, gc, comm)
     Integer n;
     double  xc[], *fc, gc[];
     Nag_Comm *comm;
#endif
{

  /*  Routine to evaluate objective function and its 1st derivatives. */

#define GC(I) gc[(I)-1]
#define XC(I) xc[(I)-1]


  *fc = pow(XC(1)+10.0*XC(2), 2.0)
    + 5.0*pow(XC(3)-XC(4), 2.0)
      + pow(XC(2)-2.0*XC(3), 4.0)
        + 10.0*pow(XC(1)-XC(4), 4.0);

  GC(1) = 2.0*(XC(1)+10.0*XC(2)) +
    40.0*pow(XC(1)-XC(4),3.0);
  GC(2) = 20.0*(XC(1)+10.0*XC(2)) +
    4.0*pow(XC(2)-2.0*XC(3),3.0);
  GC(3) = 10.0*(XC(3)-XC(4)) -
    8.0*pow(XC(2)-2.0*XC(3),3.0);
  GC(4) = 10.0*(XC(4)-XC(3)) -
    40.0*pow(XC(1)-XC(4), 3.0);
}

#ifdef NAG_PROTO
static void hess(Integer n,  double xc[], double fhesl[],
                 double fhesd[], Nag_Comm *comm)
#else
     static void hess(n, xc, fhesl, fhesd, comm)
     Integer  n;
     double  xc[], fhesl[];
     double  fhesd[];
     Nag_Comm *comm;
#endif
{
```

```
  /*  Routine to evaluate 2nd derivatives */

#define FHESD(I) fhesd[(I)-1]
#define FHESL(I) fhesl[(I)-1]
#define XC(I) xc[(I)-1]

  FHESD(1) = 2.0 + 120.0*pow(XC(1)-XC(4), 2.0);
  FHESD(2) = 200.0 + 12.0*pow(XC(2)-2.0*XC(3), 2.0);
  FHESD(3) = 10.0 + 48.0*pow(XC(2)-2.0*XC(3), 2.0);
  FHESD(4) = 10.0 + 120.0*pow(XC(1)-XC(4), 2.0);
  FHESL(1) = 20.0;
  FHESL(2) = 0.0;
  FHESL(3) = -24.0*pow(XC(2)-2.0*XC(3), 2.0);
  FHESL(4) = -120.0*pow(XC(1)-XC(4), 2.0);
  FHESL(5) = 0.0;
  FHESL(6) = -10.0;
}
```

8.2. Program Data

None.

8.3. Program Results

```
e04hdc Example Program Results

The test point is
   1.4600  -0.8200   0.5700   1.2100

2nd derivatives are consistent with 1st derivatives.

At the test point, funct gives the function value,   6.2273e+01
and the 1st derivatives
  -1.285e+01  -1.649e+02   5.384e+01   5.775e+00

hess gives the lower triangle of the Hessian matrix
   9.500e+00
   2.000e+01   2.461e+02
   0.000e+00  -9.220e+01   1.944e+02
  -7.500e+00   0.000e+00  -1.000e+01   1.750e+01
```