NAG Library Function Document

nag_opt_check_2nd_deriv (e04hdc)

1 Purpose

nag_opt_check_2nd_deriv (e04hdc) checks that a user-supplied function for calculating second derivatives of an objective function is consistent with a user-supplied function 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, const double x[], double *objf, double g[],
          Nag_Comm *comm),
     void (*hessfun)(Integer n, const double x[], double h[], double hd[],
          Nag_Comm *comm),
     const 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 you to provide a subroutine to evaluate the second derivatives of F. nag_opt_check_2nd_deriv (e04hdc) is designed to check the second derivatives calculated by such user-supplied functions. As well as the function to be checked (**hessfun**), you must supply a function (**objfun**) to evaluate the first derivatives, and a point $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ at which the checks will be made. Note that nag_opt_check_2nd_deriv (e04hdc) checks functions of the form required for nag_opt_bounds_2nd_deriv (e04lbc).

nag_opt_check_2nd_deriv (e04hdc) 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 $y^{\mathrm{T}}Hy$ and $z^{\mathrm{T}}Hz$ respectively. The same projections of the Hessian matrix are also estimated by finite differences, giving

$$\begin{aligned} & p = (y^{\mathrm{T}}g(x+hy) - y^{\mathrm{T}}g(x))/h \\ \text{and} \quad & q = (z^{\mathrm{T}}g(x+hz) - z^{\mathrm{T}}g(x))/h \end{aligned}$$

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 $y^{\mathrm{T}}Hy$ or between q and $z^{\mathrm{T}}Hz$ is judged too large, an error indicator is set.

4 References

None.

5 Arguments

1: **n** – Integer *Input*

On entry: the number n of independent variables in the objective function.

Constraint: **n** ≥ 1.

2: **objfun** – function, supplied by the user *External Function*

objfun must evaluate the function $F(x)$ and its first derivatives $\frac{\partial F}{\partial x_j}$ at a specified point. (However, if you do not wish to calculate F or its first derivatives at a particular point, there is the option of setting an argument to cause nag_opt_check_2nd_deriv (e04hdc) to terminate immediately.)

The specification of **objfun** is:

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

1: **n** – Integer *Input*

On entry: the number n of variables.

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

On entry: the point x at which the value of F, or F and the $\frac{\partial F}{\partial x_j}$, are required.

3: **objf** – double * *Output*

On exit: **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 (e04hdc) will then terminate.

4: **g**[**n**] – double *Output*

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

5: **comm** – Nag_Comm *

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

flag – Integer *Output*

On exit: if **objfun** resets **comm→flag** to some negative number then nag_opt_check_2nd_deriv (e04hdc) will terminate immediately with the error indicator NE_USER_STOP. If **fail** is supplied to nag_opt_check_2nd_deriv (e04hdc) **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 **objfun** and Nag_FALSE for all subsequent calls.

nf – Integer *Input*

On entry: 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 (e04hdc) these pointers may be allocated memory and initialized with various quantities for use by **objfun** when called from nag_opt_check_2nd_deriv (e04hdc).

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.

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

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 (e04hdc) to terminate immediately.)

The specification of **hessfun** is:

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

1: **n** – Integer *Input*

On entry: the number n of variables in the objective function.

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

On entry: the point x at which the second derivatives are required.

3: **h**[**n** × (**n** − **1**)/**2**] – double *Output*

This array is allocated internally by nag_opt_check_2nd_deriv (e04hdc).

On exit: 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.)

4: **hd**[**n**] – double *Input/Output*

On entry: the value of $\frac{\partial F}{\partial x_j}$ at the point x, for $j=1,2,\ldots,n$. These values may be useful in the evaluation of the second derivatives.

On exit: 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

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

5: **comm** – Nag_Comm *

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

flag – Integer *Output*

On exit: if **hessfun** resets **comm→flag** to some negative number then nag_opt_check_2nd_deriv (e04hdc) will terminate immediately with the error indicator NE_USER_STOP. If **fail** is supplied to nag_opt_check_2nd_deriv (e04hdc) **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 **hessfun** and Nag_FALSE for all subsequent calls.

nf – Integer *Input*

On entry: 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 (e04hdc) these pointers may be allocated memory and initialized with various quantities for use by **hessfun** when called from nag_opt_check_2nd_deriv (e04hdc).

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

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

On entry: **x**$[j-1]$, for $j = 1, 2, \ldots, n$ must contain the coordinates 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.

5: **g**[**n**] – double *Output*

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

6: **hesl**[$\mathbf{n} \times (\mathbf{n} - \mathbf{1})/\mathbf{2}$] – double *Output*

On exit: 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.

7: **hesd**[**n**] – double *Output*

On exit: 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**.

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 user-supplied functions; see the above description of **objfun** 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_check_2nd_deriv (e04hdc); **comm** will then be declared internally for use in calls to user-supplied functions.

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

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_DERIV_ERRORS

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

NE_INT_ARG_LT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 1.

NE_USER_STOP

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

7 Accuracy

The error NE_DERIV_ERRORS is returned if

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

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.

8 Parallelism and Performance

Not applicable.

9 Further Comments

nag_opt_check_2nd_deriv (e04hdc) calls **hessfun** once and **objfun** three times.

10 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 (e04hdc) is preceded by a call of nag_opt_check_deriv (e04hcc) to check the function **objfun** which calculates the first derivatives.)

10.1 Program Text

```
/* nag_opt_check_2nd_deriv (e04hdc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

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

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL h(Integer n, const double xc[], double fhesl[],
                       double fhesd[], Nag_Comm *comm);

static void NAG_CALL funct(Integer n, const double xc[], double *fc,
                           double gc[], Nag_Comm *comm);
#ifdef __cplusplus
```

```
}
#endif

int main(void)
{
  static double ruser[2] = {-1.0, -1.0};
  Integer  exit_status = 0, i, j, k, n;
  NagError fail;
  Nag_Comm comm;
  double   f, *g = 0, *hesd = 0, *hesl = 0, *x = 0;

  INIT_FAIL(fail);

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

  printf("nag_opt_check_2nd_deriv (e04hdc) Example Program Results\n\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  /* Set up an arbitrary point at which to check the derivatives */
  n = 4;

  if (n >= 1)
    {
      if (!(hesd = NAG_ALLOC(n, double)) ||
          !(hesl = NAG_ALLOC(n*(n-1)/2, double)) ||
          !(g = NAG_ALLOC(n, double)) ||
          !(x = NAG_ALLOC(n, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      printf("Invalid n.\n");
      exit_status = 1;
      return exit_status;
    }

  X(1) = 1.46;
  X(2) = -0.82;
  X(3) = 0.57;
  X(4) = 1.21;

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

  /* Check the 1st derivatives */
  /* nag_opt_check_deriv (e04hcc).
   * Derivative checker for use with nag_opt_bounds_deriv
   * (e04kbc)
   */
  nag_opt_check_deriv(n, funct, &X(1), &f, &G(1), &comm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_opt_check_deriv (e04hcc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* Check the 2nd derivatives */
  /* nag_opt_check_2nd_deriv (e04hdc).
```

```
   * Checks second derivatives of a user-defined function
   */
  nag_opt_check_2nd_deriv(n, funct, h, &X(1), &G(1), &HESL(1), &HESD(1),
                          &comm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_opt_check_2nd_deriv (e04hdc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\n2nd derivatives are consistent with 1st derivatives.\n\n");
  printf("At the test point, funct gives the function value, %13.4e\n",f);
  printf("and the 1st derivatives\n");
  for (j = 1; j <= n; ++j)
    printf("%12.3e%s", G(j), j%4?"":"\n");

  printf("\nh gives the lower triangle of the Hessian matrix\n");
  printf("%12.3e\n", HESD(1));
  k = 1;
  for (i = 2; i <= n; ++i)
    {
      for (j = k; j <= k + i - 2; ++j)
        printf("%12.3e", HESL(j));
      printf("%12.3e\n", HESD(i));
      k = k + i - 1;
    }
 END:
  NAG_FREE(hesd);
  NAG_FREE(hesl);
  NAG_FREE(g);
  NAG_FREE(x);
  return exit_status;
}

static void NAG_CALL funct(Integer n, const double xc[], double *fc,
                           double gc[], Nag_Comm *comm)
{
  /* Routine to evaluate objective function and its 1st derivatives. */

  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback funct, first invocation.)\n");
      comm->user[0] = 0.0;
    }
  *fc = pow(xc[0]+10.0*xc[1], 2.0) + 5.0*pow(xc[2]-xc[3], 2.0)
        + pow(xc[1]-2.0*xc[2], 4.0) + 10.0*pow(xc[0]-xc[3], 4.0);

  gc[0] = 2.0*(xc[0]+10.0*xc[1]) + 40.0*pow(xc[0]-xc[3], 3.0);
  gc[1] = 20.0*(xc[0]+10.0*xc[1]) + 4.0*pow(xc[1]-2.0*xc[2], 3.0);
  gc[2] = 10.0*(xc[2]-xc[3]) - 8.0*pow(xc[1]-2.0*xc[2], 3.0);
  gc[3] = 10.0*(xc[3]-xc[2]) - 40.0*pow(xc[0]-xc[3], 3.0);
}

static void NAG_CALL h(Integer n, const double xc[], double fhesl[],
                       double fhesd[], Nag_Comm *comm)
{
  /* Routine to evaluate 2nd derivatives */

  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback h, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  fhesd[0] = 2.0 + 120.0*pow(xc[0]-xc[3], 2.0);
  fhesd[1] = 200.0 + 12.0*pow(xc[1]-2.0*xc[2], 2.0);
  fhesd[2] = 10.0 + 48.0*pow(xc[1]-2.0*xc[2], 2.0);
  fhesd[3] = 10.0 + 120.0*pow(xc[0]-xc[3], 2.0);
  fhesl[0] = 20.0;
  fhesl[1] = 0.0;
```

```
  fhesl[2] = -24.0*pow(xc[1]-2.0*xc[2], 2.0);
  fhesl[3] = -120.0*pow(xc[0]-xc[3], 2.0);
  fhesl[4] = 0.0;
  fhesl[5] = -10.0;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_opt_check_2nd_deriv (e04hdc) Example Program Results

The test point is
   1.4600  -0.8200   0.5700   1.2100
(User-supplied callback funct, first invocation.)
(User-supplied callback h, first invocation.)

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

h 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
```