# NAG Library Function Document

# nag_opt_estimate_deriv (e04xac)

## 1 Purpose

nag_opt_estimate_deriv (e04xac) computes an approximation to the gradient vector and/or the Hessian matrix for use in conjunction with, or following the use of an optimization function (such as nag_opt_nlp (e04ucc)).

## 2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_estimate_deriv (Integer n, double x[],
     void (*objfun)(Integer n, const double x[], double *objf, double g[],
          Nag_Comm *comm),
     double *objf, double g[], double h_forward[], double h_central[],
     double hess[], Integer tdhess, Nag_DerivInfo *deriv_info,
     Nag_E04_Opt *options, Nag_Comm *comm, NagError *fail)
```

## 3 Description

nag_opt_estimate_deriv (e04xac) is based on the routine FDCALC described in Gill *et al.* (1983a). It computes finite difference approximations to the gradient vector and the Hessian matrix for a given function, and aims to provide sufficiently accurate estimates for use with an optimization algorithm.

The simplest approximation of the gradients involves the forward-difference formula, in which the derivative of $f'(x)$ of a univariate function $f(x)$ is approximated by the quantity

$$\rho_F(f,h) = \frac{f(x+h) - f(x)}{h}$$

for some interval $h > 0$, where the subscript 'F' denotes 'forward-difference' (see Gill *et al.* (1983b)).

The choice of which gradients are returned by nag_opt_estimate_deriv (e04xac) is controlled by the optional parameter **options.deriv_want** (see Section 11 for a description of this argument). To summarise the procedure used by nag_opt_estimate_deriv (e04xac) when **options.deriv_want** = Nag_Grad_HessFull (default value) (i.e., for the case when the objective function is available and you require estimates of gradient values and the full Hessian matrix) consider a univariate function $f$ at the point $x$. (In order to obtain the gradient of a multivariate function $F(x)$, where $x$ is an $n$-vector, the procedure is applied to each component of $x$, keeping the other components fixed.) Roughly speaking, the method is based on the fact that the bound on the relative truncation error in the forward-difference approximation tends to be an increasing function of $h$, while the relative condition error bound is generally a decreasing function of $h$, hence changes in $h$ will tend to have opposite effects on these errors (see Gill *et al.* (1983b)).

The 'best' interval $h$ is given by

$$h_F = 2\sqrt{\frac{(1 + |f(x)|)e_R}{|\Phi|}} \tag{1}$$

where $\Phi$ is an estimate of $f''(x)$, and $e_R$ is an estimate of the relative error associated with computing the function (see Chapter 8 of Gill *et al.* (1981)). Given an interval $h$, $\Phi$ is defined by the second-order approximation

$$\Phi = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

The decision as to whether a given value of $\Phi$ is acceptable involves $\hat{c}(\Phi)$, the following bound on the relative condition error in $\Phi$:

$$\hat{c}(\Phi) = \frac{4e_R(1+|f|)}{h^2|\Phi|}$$

(When $\Phi$ is zero, $\hat{c}(\Phi)$ is taken as an arbitrary large number.)

The procedure selects the interval $h_\phi$ (to be used in computing $\Phi$) from a sequence of trial intervals $(h_k)$. The initial trial interval is taken as

$$\bar{h} = 2(1+|x|)\sqrt[4]{e_R}.$$

unless you specify the initial value to be used.

The value of $\hat{c}(\Phi)$ for a trial value $h_k$ is defined as 'acceptable' if it lies in the interval $[0.0001, 0.01]$. In this case $h_\phi$ is taken as $h_k$, and the current value of $\Phi$ is used to compute $h_F$ from (1). If $\hat{c}(\Phi)$ is unacceptable, the next trial interval is chosen so that the relative condition error bound will either decrease or increase, as required. If the bound on the relative condition error is too large, a larger interval is used as the next trial value in an attempt to reduce the condition error bound. On the other hand, if the relative condition error bound is too small, $h_k$ is reduced.

The procedure will fail to produce an acceptable value of $\hat{c}(\Phi)$ in two situations. Firstly, if $f''(x)$ is extremely small, then $\hat{c}(\Phi)$ may never become small, even for a very large value of the interval. Alternatively, $\hat{c}(\Phi)$ may never exceed 0.0001, even for a very small value of the interval. This usually implies that $f''(x)$ is extremely large, and occurs most often near a singularity.

As a check on the validity of the estimated first derivative, the procedure provides a comparison of the forward-difference approximation computed with $h_F$ (as above) and the central-difference approximation computed with $h_\phi$. Using the central-difference formula the first derivative can be approximated by

$$\rho_c(f,h) = \frac{f(x+h) - f(x-h)}{2h}$$

where $h > 0$. If the values $h_F$ and $h_\phi$ do not display some agreement, neither can be considered reliable.

The approximate Hessian matrix $G$ is defined as in Chapter 2 of Gill *et al.* (1981), by

$$G_{ij}(x) = \frac{1}{h_i h_j}\left(f\left(x + h_i e_i + h_j e_j\right) - f(x + h_i e_i) - f\left(x + h_j e_j\right) + f(x)\right).$$

where $h_j$ is the best forward-difference interval associated with the $j$th component of $f$ and $e_j$ is the vector with unity in the $j$th position and zeros elsewhere.

If you require the gradients and only the diagonal of the Hessian matrix (i.e., **options.deriv_want** = Nag_Grad_HessDiag; see Section 11.2), nag_opt_estimate_deriv (e04xac) follows a similar procedure to the default case, except that the initial trial interval is taken as $10\bar{h}$, where

$$\bar{h} = 2(1+|x|)\sqrt{e_R}$$

and the value of $\hat{c}(\Phi)$ for a trial value $h_k$ is defined as acceptable if it lies in the interval $[0.001, 0.1]$. The elements of the Hessian diagonal which are returned in this case are the values of $\Phi$ corresponding to the 'best' intervals.

When both function and gradients are available and you require the Hessian matrix (i.e., **options.deriv_want** = Nag_HessFull; see Section 11.2), nag_opt_estimate_deriv (e04xac) follows a similar procedure to the case above with the exception that the gradient function $g(x)$ is substituted for the objective function and so the forward-difference interval for the first derivative of $g(x)$ with respect to variable $x_j$ is computed. The $j$th column of the approximate Hessian matrix is then defined as in Chapter 2 of Gill *et al.* (1981), by

$$\frac{g\left(x + h_j e_j\right) - g(x)}{h_j}$$

where $h_j$ is the best forward-difference interval associated with the $j$th component of $g$.

# 4 References

Gill P E, Murray W, Saunders M A and Wright M H (1983a) Documentation for FDCALC and FDCORE *Technical Report SOL* 83–6 Stanford University

Gill P E, Murray W, Saunders M A and Wright M H (1983b) Computing forward-difference intervals for numerical optimization *SIAM J. Sci. Statist. Comput.* **4** 310–321

Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

# 5 Arguments

1: **n** – Integer *Input*

*On entry*: the number $n$ of variables.

*Constraint*: **n** $\geq 1$.

2: **x**[**n**] – double *Input*

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

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

**objfun** must evaluate the objective function $F(x)$ and (optionally) its gradient $g(x) = \frac{\partial F}{\partial x_j}$ for a specified $n$ element vector $x$.

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 $F$ and, if **comm→flag** $= 2$, 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_estimate_deriv (e04xac) will then terminate.

4: **g**[**n**] – double *Output*

*On exit*: if **comm→flag** $= 2$ on entry, then **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 $j = 1, 2, \ldots, n$. If it is not possible to evaluate the first derivatives then **objfun** should assign a negative value to **comm→flag**; nag_opt_estimate_deriv (e04xac) will then terminate.

If **comm→flag** $= 0$ on entry, then **g** is not referenced.

5: **comm** – Nag_Comm *

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

**flag** – Integer *Input/Output*

*On entry*: **comm→flag** will be set to 0 or 2. The value 0 indicates that only $F$ itself needs to be evaluated. The value 2 indicates that both $F$ and its first derivatives must be calculated.

*On exit*: if **objfun** resets **comm→flag** to a negative number then nag_opt_estimate_deriv (e04xac) will terminate immediately with the error indicator NE_USER_STOP. If **fail** is supplied to nag_opt_estimate_deriv (e04xac), **fail.errnum** will be set to the user's 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_estimate_deriv (e04xac) these pointers may be allocated memory and initialized with various quantities for use by **objfun** when called from nag_opt_estimate_deriv (e04xac).

**Note**: **objfun** should be thoroughly tested before being used in conjunction with nag_opt_estimate_deriv (e04xac). The array **x** must **not** be changed by **objfun**.

4: **objf** – double * *Output*

*On exit*: the value of the objective function evaluated at the input vector in **x**.

5: **g**[**n**] – double *Output*

*On exit*: if **options.deriv_want** = Nag_Grad_HessFull (the default; see Section 11.2) or **options.deriv_want** = Nag_Grad_HessDiag, **g**$[j-1]$ contains the best estimate of the first partial derivative for the $j$th variable, $j = 1, 2, \ldots, n$. If **options.deriv_want** = Nag_HessFull, **g**$[j-1]$ contains the first partial derivative for the $j$th variable as evaluated by **objfun**.

6: **h_forward**[**n**] – double *Input/Output*

*On entry*: if the optional parameter **options.use_hfwd_init** = Nag_FALSE (the default; see Section 11.2), the values contained in **h_forward** on entry to nag_opt_estimate_deriv (e04xac) are ignored.

If **options.use_hfwd_init** = Nag_TRUE, **h_forward** is assumed to contain meaningful values on entry: if **h_forward**$[j-1] > 0$ then it is used as the initial trial interval for computing the appropriate partial derivative to the $j$th variable, $j = 1, 2, \ldots, n$; if **h_forward**$[j-1] \leq 0.0$, then the initial trial interval for the $j$th variable is computed by nag_opt_estimate_deriv (e04xac) (see Section 11.2).

*On exit*: **h_forward**$[j-1]$ is the best interval found for computing a forward-difference approximation to the appropriate partial derivative for the $j$th variable. If you do not require this information, a **NULL** pointer may be provided, and nag_opt_estimate_deriv (e04xac) will allocate memory internally to calculate the difference intervals.

*Constraint*: **h_forward** must not be **NULL** if **options.use_hfwd_init** = Nag_TRUE.

7: **h_central**[**n**] – double *Output*

*On exit*: **h_central**$[j-1]$ is the best interval found for computing a central-difference approximation to the appropriate partial derivative for the $j$th variable. If you do not require this information, a **NULL** pointer may be provided, and nag_opt_estimate_deriv (e04xac) will allocate memory internally to calculate the difference intervals.

8: **hess**[**n** $\times$ **tdhess**] – double *Output*

**Note**: the $(i, j)$th element of the matrix is stored in **hess**$[(i-1) \times$ **tdhess** $+ j - 1]$.

*On exit*: if the optional parameter **options.deriv_want** = Nag_Grad_HessFull (the default; see Section 11.2) or **options.deriv_want** = Nag_HessFull, the estimated Hessian matrix is contained in the leading $n$ by $n$ part of this array. If **options.deriv_want** = Nag_Grad_HessDiag, the $n$ elements of the estimated Hessian diagonal are contained in the first row of this array.

9: **tdhess** – Integer *Input*

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

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

10: **deriv_info**[**n**] – Nag_DerivInfo * *Output*

*On exit*: **deriv_info**$[j-1]$ contains diagnostic information on the $j$th variable, for $j = 1, 2, \ldots, n$.

**deriv_info**$[j-1]$ = Nag_Deriv_OK
No unusual behaviour observed in estimating the appropriate derivative.

**deriv_info**$[j-1]$ = Nag_Fun_Constant
The appropriate function appears to be constant.

**deriv_info**$[j-1]$ = Nag_Fun_LinearOdd
The appropriate function appears to be linear or odd.

**deriv_info**$[j-1]$ = Nag_2ndDeriv_Large
The second derivative of the appropriate function appears to be so large that it cannot be reliably estimated (e.g., near a singularity).

**deriv_info**$[j-1]$ = Nag_1stDeriv_Small
The forward-difference and central-difference estimates of the appropriate first derivatives do not agree to half a decimal place; this usually occurs because the first derivative is small.

A more detailed explanation of these warnings is given in Section 9.1.

11: **options** – Nag_E04_Opt * *Input/Output*

*On entry/exit*: a pointer to a structure of type Nag_E04_Opt whose members are optional parameters for nag_opt_estimate_deriv (e04xac). These structure members offer the means of adjusting some of the argument values of the computation and on output will supply further details of the results. A description of the members of **options** is given in Section 11.

If any of these optional parameters are required then the structure **options** should be declared and initialized by a call to nag_opt_init (e04xxc) and supplied as an argument to nag_opt_estima te_deriv (e04xac). However, if the optional parameters are not required the NAG defined null pointer, `E04_DEFAULT`, can be used in the function call.

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

**Note**: **comm** is a NAG defined type (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

*On entry/exit*: structure containing pointers for communication with user-supplied functions; see the 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_estimate_deriv (e04xac); **comm** will then be declared internally for use in calls to user-supplied functions.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

### 5.1 Description of Printed Output

Results from nag_opt_estimate_deriv (e04xac) are printed out by default. The level of printed output can be controlled with the structure members **options**.**list** and **options**.**print_deriv** (see Section 11.2). If **options**.**list** = Nag_TRUE then the argument values to nag_opt_estimate_deriv (e04xac) are listed, whereas printout of results is governed by the value of **options**.**print_deriv**.

The default, **options**.**print_deriv** = Nag_D_Print provides the following line of output for each variable.

| | |
|---|---|
| `j` | the index of the variable for which the difference interval has been computed. |
| `X(j)` | the value of $x_j$ as provided in $\mathbf{x}[j-1]$. |
| `Fwd diff int` | the best interval found for computing a forward-difference approximation to the appropriate partial derivative with respect to $x_j$. |
| `Cent diff int` | the best interval found for computing a central-difference approximation to the appropriate partial derivative with respect to $x_j$. |
| `Error est` | a bound on the estimated error in the final forward-difference approximation. When $\mathbf{deriv\_info}[j-1]$ = Nag_Fun_Constant, `Error est` is set to zero. |
| `Grad est` | best estimate of the first partial derivative with respect to $x_j$. |
| `Hess diag est` | best estimate of the second partial derivative with respect to $x_j$. |
| `Nfun` | the number of function evaluations used to compute the final difference intervals for $x_j$. |
| `Info` | gives diagnostic information for $x_j$. `Info` will be one of `OK`, `Constant?`, `Linear or odd?`, `Large 2nd deriv?`, or `Small 1st deriv?`, corresponding to $\mathbf{deriv\_info}[j-1]$ = Nag_Deriv_OK, Nag_Fun_Constant, Nag_Fun_LinearOdd, Nag_2ndDeriv_Large or Nag_1stDeriv_Small, respectively. |

## 6 Error Indicators and Warnings

**NE_2_INT_ARG_LT**

On entry, **tdhess** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These arguments must satisfy **tdhess** ≥ **n**.

**NE_ALLOC_FAIL**

Dynamic memory allocation failed.

**NE_BAD_PARAM**

On entry, argument **options**.**deriv_want** had an illegal value.

On entry, argument **options**.**print_deriv** had an illegal value.

**NE_H_FORWARD_NULL**

**options**.**use_hfwd_init** = Nag_TRUE but argument **h_forward** is **NULL**.

**NE_INT_ARG_LT**

On entry, **n** $= \langle value \rangle$.
Constraint: **n** $\geq 1$.

**NE_INVALID_REAL_RANGE_F**

Value $\langle value \rangle$ given to **options.f_prec** is not valid. Correct range is **options.f_prec** $> 0.0$.

**NE_NOT_APPEND_FILE**

Cannot open file $\langle string \rangle$ for appending.

**NE_NOT_CLOSE_FILE**

Cannot close file $\langle string \rangle$.

**NE_OPT_NOT_INIT**

Options structure not initialized.

**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 **objfun**. If **fail** is supplied, the value of **fail.errnum** will be the same as your setting of **comm→flag**.

**NE_WRITE_ERROR**

Error occurred when writing to file $\langle string \rangle$.

**NW_DERIV_INFO**

On exit, at least one element of the **deriv_info** array does not contain the value **deriv_info** = Nag_Deriv_OK. This does not necessarily represent an unsuccessful exit.

See Section 9.1 for information about the possible values which may be returned in **deriv_info**.

## 7 Accuracy

nag_opt_estimate_deriv (e04xac) exits with **fail.code** = NE_NOERROR if the algorithm terminated successfully, i.e., the forward-difference estimates of the appropriate first derivatives (computed with the final estimate of the 'optimal' forward-difference interval $h_F$) and the central-difference estimates (computed with the interval $h_\phi$ used to compute the final estimate of the second derivative) agree to at least half a decimal place.

## 8 Parallelism and Performance

nag_opt_estimate_deriv (e04xac) is not threaded in any implementation.

## 9 Further Comments

### 9.1 Diagnostic Information

Diagnostic information is returned via the array argument **deriv_info**. If **fail.code** = NE_NOERROR on exit then **deriv_info**$[j-1]$ = Nag_Deriv_OK, for $j = 1, 2, \ldots, n$. If **fail.code** = NW_DERIV_INFO on exit, then, for at least one $j$, **deriv_info**$[j-1]$ contains one of the following values:

Nag_Fun_Constant

The appropriate function appears to be constant. On exit, **h_forward**$[j-1]$ is set to the initial trial interval corresponding to a well scaled problem, and `Error est` in the printed output is set to zero. This value occurs when the estimated relative condition error in the first derivative approximation is unacceptably large for every value of the finite difference interval. If this happens when the function is not constant the initial interval may be too small; in this case, it may be worthwhile to rerun nag_opt_estimate_deriv (e04xac) with larger initial trial interval values supplied in **h_forward** and with the optional parameter **options.use_hfwd_init** set to Nag_TRUE. This error may also occur if the function evaluation includes an inordinately large constant term or if optional parameter **options.f_prec** is too large.

Nag_Fun_LinearOdd

The appropriate function appears to be linear or odd. On exit, **h_forward**$[j-1]$ is set to the smallest interval with acceptable bounds on the relative condition error in the forward- and backward-difference estimates. In this case, the estimated relative condition error in the second derivative approximation remained large for every trial interval, but the estimated error in the first derivative approximation was acceptable for at least one interval. If the function is not linear or odd the relative condition error in the second derivative may be decreasing very slowly. It may be worthwhile to rerun nag_opt_estimate_deriv (e04xac) with larger initial trial interval values supplied in **h_forward** and with **options.use_hfwd_init** set to Nag_TRUE.

Nag_2ndDeriv_Large

The second derivative of the appropriate function appears to be so large that it cannot be reliably estimated (e.g., near a singularity). On exit, **h_forward**$[j-1]$ is set to the smallest trial interval.

This value occurs when the relative condition error estimate in the second derivative remained very small for every trial interval.

If the second derivative is not large the relative condition error in the second derivative may be increasing very slowly. It may be worthwhile to rerun nag_opt_estimate_deriv (e04xac) with smaller initial trial interval values supplied in **h_forward** and with **options.use_hfwd_init** set to Nag_TRUE. This error may also occur when the given value of the optional parameter **options.f_prec** is not a good estimate of a bound on the absolute error in the appropriate function (i.e., **options.f_prec** is too small).

Nag_1stDeriv_Small

The algorithm terminated with an apparently acceptable estimate of the second derivative. However the forward-difference estimates of the appropriate first derivatives (computed with the final estimate of the 'optimal' forward-difference interval) and the central difference estimates (computed with the interval used to compute the final estimate of the second derivative) do not agree to half a decimal place. The usual reason that the forward- and central-difference estimates fail to agree is that the first derivative is small.

If the first derivative is not small, it may be helpful to run nag_opt_estimate_deriv (e04xac) at a different point.

### 9.2 Timing

Unless the objective function can be evaluated very quickly, the run time will usually be dominated by the time spent in **objfun**.

To evaluate an acceptable set of finite difference intervals for a well-scaled problem nag_opt_estimate_deriv (e04xac) will use around two function evaluations per variable; in a badly scaled problem, six function evaluations per variable may be needed.

In the default case where gradients and the full Hessian matrix are required (i.e., optional parameter **options.deriv_want** = Nag_Grad_HessFull), nag_opt_estimate_deriv (e04xac) performs a further $3n(n+1)/2$ function evaluations. If the full Hessian matrix is required, with you supplying both function and gradients (i.e., **options.deriv_want** = Nag_HessFull), a further $n$ function evaluations are performed.

## 10 Example

The example program computes the gradient vector and Hessian matrix of the following function:

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

at the point $(3, -1, 0, 1)^{\mathrm{T}}$.

This example shows the use of some optional parameters which are discussed fully in Section 11.

The same **objfun** is used as in Section 10 and the derivatives are estimated at the same point. The **options** structure is declared and initialized by nag_opt_init (e04xxc). Two options are set to suppress all printout from nag_opt_estimate_deriv (e04xac): **options.list** is set to Nag_FALSE and **options.print_deriv** = Nag_D_NoPrint. **options.deriv_want** = Nag_Grad_HessDiag and nag_opt_estimate_deriv (e04xac) is called. The returned function value and estimated derivative values are printed out and **options.deriv_want** is reset to **options.deriv_want** = Nag_HessFull before nag_opt_estimate_deriv (e04xac) is called again. On return, the computed function value and gradient, and estimated Hessian, are printed out.

## 10.1 Program Text

```
/* nag_opt_estimate_deriv (e04xac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 *
 */
#include <nag.h>
#include <nag_stdlib.h>
#include <stdio.h>
#include <string.h>
#include <nage04.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static void NAG_CALL objfun(Integer n, const double x[], double *objf,
                              double g[], Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

#define H(I, J) h[(I) *tdh + J]

int main(void)
{
  Integer exit_status = 0, i, j, n, tdh;
  double *g = 0, *h = 0, *h_central = 0, *h_forward = 0, *hess_diag = 0,
         objf, *x = 0;
  Nag_Comm comm;
  Nag_DerivInfo *deriv_info = 0;
  Nag_E04_Opt options;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_opt_estimate_deriv (e04xac) Example Program Results\n");
  n = 4;

  if (!(x = NAG_ALLOC(n, double)) ||
      !(h_central = NAG_ALLOC(n, double)) ||
      !(h_forward = NAG_ALLOC(n, double)) ||
      !(g = NAG_ALLOC(n, double)) ||
      !(h = NAG_ALLOC(n * n, double)) ||
      !(hess_diag = NAG_ALLOC(n, double)) ||
      !(deriv_info = NAG_ALLOC(n, Nag_DerivInfo))
         )
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  tdh = n;
```

```
  x[0] = 3.0;
  x[1] = -1.0;
  x[2] = 0.0;
  x[3] = 1.0;

  /* nag_opt_init (e04xxc).
   * Initialization function for option setting
   */
  nag_opt_init(&options);
  options.list = Nag_FALSE;
  options.print_deriv = Nag_D_NoPrint;
  options.deriv_want = Nag_Grad_HessDiag;

  printf("\nEstimate gradient and Hessian diagonals given function only\n");

  /* Note: it is acceptable to pass an array of length n (hess_diag)
   * as the Hessian parameter in this case.
   */
  /* nag_opt_estimate_deriv (e04xac), see above. */
  nag_opt_estimate_deriv(n, x, objfun, &objf, g, h_forward, h_central,
                         hess_diag, tdh, deriv_info, &options, &comm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_opt_estimate_deriv (e04xac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  printf("\nFunction value: %13.4e\n", objf);
  printf("Estimated gradient vector\n");
  for (i = 0; i < n; ++i)
    printf("%13.4e  ", g[i]);
  printf("\nEstimated Hessian matrix diagonal\n");
  for (i = 0; i < n; ++i)
    printf("%13.4e  ", hess_diag[i]);
  printf("\n");

  options.deriv_want = Nag_HessFull;

  printf("\nEstimate full Hessian given function and gradients\n");
  /* nag_opt_estimate_deriv (e04xac), see above. */
  nag_opt_estimate_deriv(n, x, objfun, &objf, g, h_forward, h_central,
                         h, tdh, deriv_info, &options, &comm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_opt_estimate_deriv (e04xac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  printf("\nFunction value: %13.4e\n", objf);
  printf("Computed gradient vector\n");
  for (i = 0; i < n; ++i)
    printf("%13.4e  ", g[i]);
  printf("\nEstimated Hessian matrix\n");
  for (i = 0; i < n; ++i) {
    for (j = 0; j < n; ++j)
      printf("%13.4e  ", H(i, j));
    printf("\n");
  }

END:
  NAG_FREE(x);
  NAG_FREE(h_central);
  NAG_FREE(h_forward);
  NAG_FREE(g);
  NAG_FREE(h);
  NAG_FREE(hess_diag);
  NAG_FREE(deriv_info);

  return exit_status;
}
```

```
static void NAG_CALL objfun(Integer n, const double x[], double *objf,
                            double g[], Nag_Comm *comm)
{
  double a, asq, b, bsq, c, csq, d, dsq;

  a = x[0] + 10.0 * x[1];
  b = x[2] - x[3];
  c = x[1] - 2.0 * x[2];
  d = x[0] - x[3];
  asq = a * a;
  bsq = b * b;
  csq = c * c;
  dsq = d * d;
  *objf = asq + 5.0 * bsq + csq * csq + 10.0 * dsq * dsq;
  if (comm->flag == 2) {
    g[0] = 2.0 * a + 40.0 * d * dsq;
    g[1] = 20.0 * a + 4.0 * c * csq;
    g[2] = 10.0 * b - 8.0 * c * csq;
    g[3] = -10.0 * b - 40.0 * d * dsq;
  }
}

/* objfun */
```

### 10.2 Program Data

None.

### 10.3 Program Results

```
nag_opt_estimate_deriv (e04xac) Example Program Results

Estimate gradient and Hessian diagonals given function only

Function value:    2.1500e+02
Estimated gradient vector
   3.0600e+02    -1.4400e+02    -2.0000e+00    -3.1000e+02
Estimated Hessian matrix diagonal
   4.8200e+02     2.1200e+02     5.7995e+01     4.9000e+02

Estimate full Hessian given function and gradients

Function value:    2.1500e+02
Computed gradient vector
   3.0600e+02    -1.4400e+02    -2.0000e+00    -3.1000e+02
Estimated Hessian matrix
   4.8200e+02     2.0000e+01     0.0000e+00    -4.8000e+02
   2.0000e+01     2.1200e+02    -2.4000e+01     0.0000e+00
   0.0000e+00    -2.4000e+01     5.8000e+01    -1.0000e+01
  -4.8000e+02     0.0000e+00    -1.0000e+01     4.9000e+02
```

## 11 Optional Parameters

A number of optional input and output arguments to nag_opt_estimate_deriv (e04xac) are available through the structure argument **options**, type Nag_E04_Opt. a argument may be selected by assigning an appropriate value to the relevant structure member; those arguments not selected will be assigned default values. If no use is to be made of any of the optional parameters you should use the NAG defined null pointer, `E04_DEFAULT`, in place of **options** when calling nag_opt_estimate_deriv (e04xac); the default settings will then be used for all arguments.

Before assigning values to **options** directly the structure **must** be initialized by a call to the function nag_opt_init (e04xxc). Values may then be assigned to the structure members in the normal C manner.

Option settings may also be read from a text file using the function nag_opt_read (e04xyc) in which case initialization of the **options** structure will be performed automatically if not already done. Any subsequent direct assignment to the **options** structure must **not** be preceded by initialization.

### 11.1 Optional Parameter Checklist and Default Values

For easy reference, the following list shows the members of **options** which are valid for nag_opt_estimate_deriv (e04xac) together with their default values where relevant. The number $\epsilon$ is a generic notation for ***machine precision*** (see nag_machine_precision (X02AJC)).

| | |
|---|---|
| `Boolean list` | Nag_TRUE |
| `Nag_DPrintType print_deriv` | Nag_D_Print |
| `char outfile[80]` | `stdout` |
| `Nag_DWantType deriv_want` | Nag_Grad_HessFull |
| `Boolean use_hfwd_init` | Nag_FALSE |
| `double f_prec` | $\epsilon^{0.9}$ |
| `double f_prec_used` | |
| `Integer nf` | |

### 11.2 Description of the Optional Parameters

**list** – Nag_Boolean Default $=$ Nag_TRUE

*On entry*: if **options.list** $=$ Nag_TRUE the argument settings in the call to nag_opt_estimate_deriv (e04xac) will be printed.

**print_deriv** – Nag_DPrintType Default $=$ Nag_D_Print

*On entry*: controls whether printout is produced by nag_opt_estimate_deriv (e04xac). The following values are available:

Nag_D_NoPrint No output.

Nag_D_Print Printout for each variable as described in Section 5.

*Constraint*: **options.print_deriv** $=$ Nag_D_NoPrint or Nag_D_Print.

**outfile** – const char[80] Default $=$ `stdout`

*On entry*: the name of the file to which results should be printed. If **options.outfile**[0] $=$ '\0' then the `stdout` stream is used.

**deriv_want** – Nag_DWantType Default $=$ Nag_Grad_HessFull

*On entry*: specifies which derivatives nag_opt_estimate_deriv (e04xac) should estimate. The following values are available:

Nag_Grad_HessFull Estimate the gradient and full Hessian, with you supplying the objective function via **objfun**.

Nag_Grad_HessDiag Estimate the gradient and the Hessian diagonal values, with you supplying the objective function via **objfun**.

Nag_HessFull Estimate the full Hessian, with you supplying the objective function and gradients via **objfun**.

*Constraint*: **options.deriv_want** $=$ Nag_Grad_HessFull, Nag_Grad_HessDiag or Nag_HessFull.

**use_hfwd_init** – Nag_Boolean Default $=$ Nag_FALSE

*On entry*: if **options.use_hfwd_init** $=$ Nag_FALSE, then nag_opt_estimate_deriv (e04xac) ignores any values supplied on entry in **h_forward**, and computes the initial trial intervals itself. If **options.use_hfwd_init** $=$ Nag_TRUE, then nag_opt_estimate_deriv (e04xac) uses the forward difference interval provided in **h_forward**$[j-1]$ as the initial trial interval for computing the appropriate partial derivative to the $j$th variable, $j = 1, 2, \ldots, n$; however, if **h_forward**$[j-1] \leq 0.0$ for some $j$, the initial trial interval for the $j$th variable is computed by nag_opt_estimate_deriv (e04xac).

**f_prec** – double Default $= \epsilon^{0.9}$

*On entry*: specifies $e_R$, which is intended to measure the accuracy with which the problem function $F$ can be computed. The value of **options.f_prec** should reflect the relative precision of $1 + |F(x)|$, i.e., acts as a relative precision when $|F|$ is large, and as an absolute precision when $|F|$ is small. For example, if $|F(x)|$ is typically of order 1000 and the first six significant figures are known to be correct, an appropriate value of **options.f_prec** would be $10^{-6}$. The default value of $\epsilon^{0.9}$ will be appropriate for most simple functions that are computed with full accuracy

A discussion of $e_R$ is given in Chapter 8 of Gill *et al.* (1981). If you provide a value of **options.f_prec** which nag_opt_estimate_deriv (e04xac) determines to be either too small or too large, the default value will be used instead and a warning will be output if optional parameter **options.print_deriv** $=$ Nag_D_Print. The value actually used is returned in **options.f_prec_used**.

*Constraint*: **options.f_prec** $> 0$.

**f_prec_used** – double

*On exit*: if **fail.code** $=$ NE_NOERROR or NW_DERIV_INFO, or if **options.nf** $> 1$ and **fail.code** $=$ NE_USER_STOP, then **options.f_prec_used** contains the value of $e_R$ used by nag_opt_estimate_deriv (e04xac). If you supply a value for **options.f_prec** and nag_opt_estimate_deriv (e04xac) considers that the value supplied is neither too large nor too small, then this value will be returned in **options.f_prec_used**; otherwise **options.f_prec_used** will contain the default value, $\epsilon^{0.9}$.

**nf** – double

*On exit*: the number of times the objective function has been evaluated (i.e., number of calls of **objfun**).