

## nag\_opt\_bounds\_deriv (e04kbc)

### 1. Purpose

**nag\_opt\_bounds\_deriv (e04kbc)** is a comprehensive quasi-Newton algorithm for finding:

- an unconstrained minimum of a function of several variables
- a minimum of a function of several variables subject to fixed upper and/or lower bounds on the variables.

First derivatives are required. The function `nag_opt_bounds_deriv` is intended for objective functions which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

### 2. Specification

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

void nag_opt_bounds_deriv(Integer n,
                          void (*objfun)(Integer n, double x[], double *objf,
                                           double g[], Nag_Comm *comm),
                          Nag_BoundType bound, double bl[], double bu[],
                          double x[], double *objf, double g[],
                          Nag_E04_Opt *options, Nag_Comm *comm, NagError *fail)
```

### 3. Description

This function is applicable to problems of the form:

$$\begin{aligned} &\text{Minimize} && F(x_1, x_2, \dots, x_n) \\ &\text{subject to} && l_j \leq x_j \leq u_j, \quad j = 1, 2, \dots, n. \end{aligned}$$

Special provision is made for unconstrained minimization (i.e., problems which actually have no bounds on the  $x_j$ ), problems which have only non-negativity bounds, and problems in which  $l_1 = l_2 = \dots = l_n$  and  $u_1 = u_2 = \dots = u_n$ . It is possible to specify that a particular  $x_j$  should be held constant. The user must supply a starting point and a function **objfun** to calculate the value of  $F(x)$  and its first derivatives  $\partial F/\partial x_j$  at any point  $x$ .

A typical iteration starts at the current point  $x$  where  $n_z$  (say) variables are free from both their bounds. The vector  $g_z$ , whose elements are the derivatives of  $F(x)$  with respect to the free variables, is known. A unit lower triangular matrix  $L$  and a diagonal matrix  $D$  (both of dimension  $n_z$ ), such that  $LDL^T$  is a positive-definite approximation to the matrix of second derivatives with respect to the free variables, are also stored. The equations

$$LDL^T p_z = -g_z$$

are solved to give a search direction  $p_z$ , which is expanded to an  $n$ -vector  $p$  by the insertion of appropriate zero elements. Then  $\alpha$  is found such that  $F(x + \alpha p)$  is approximately a minimum (subject to the fixed bounds) with respect to  $\alpha$ ;  $x$  is replaced by  $x + \alpha p$ , and the matrices  $L$  and  $D$  are updated so as to be consistent with the change produced in the gradient by the step  $\alpha p$ . If any variable actually reaches a bound during the search along  $p$ , it is fixed and  $n_z$  is reduced for the next iteration.

There are two sets of convergence criteria – a weaker and a stronger. Whenever the weaker criteria are satisfied, the Lagrange-multipliers are estimated for all the active constraints. If any Lagrange-multiplier estimate is significantly negative, then one of the variables associated with a negative Lagrange-multiplier estimate is released from its bound and the next search direction is computed in the extended subspace (i.e.,  $n_z$  is increased). Otherwise minimization continues in the current subspace provided that this is practicable. When it is not, or when the stronger convergence criteria is already satisfied, then, if one or more Lagrange-multiplier estimates are close to zero, a slight perturbation is made in the values of the corresponding variables in turn until a lower function value is obtained. The normal algorithm is then resumed from the perturbed point.

If a saddle point is suspected, a local search is carried out with a view to moving away from the saddle point. In addition, nag\_opt\_bounds\_deriv gives the user the option of specifying that a local search should be performed when a point is found which is thought to be a constrained minimum.

If the user specifies that the problem is unconstrained, nag\_opt\_bounds\_deriv sets the  $l_j$  to  $-10^{10}$  and the  $u_j$  to  $10^{10}$ . Thus, provided that the problem has been sensibly scaled, no bounds will be encountered during the minimization process and nag\_opt\_bounds\_deriv will act as an unconstrained minimization algorithm.

#### 4. Parameters

**n**

Input: the number  $n$  of independent variables.

Constraint:  $n \geq 1$ .

**objfun**

**objfun** must evaluate the function  $F(x)$  and its first derivatives  $\partial F/\partial x_j$  at any point  $x$ . (However, if the user does not wish to calculate  $F(x)$  or its first derivatives at a particular  $x$ , there is the option of setting a parameter to cause nag\_opt\_bounds\_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  $\partial F/\partial x_j$ , 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\_bounds\_deriv will then terminate.

**g[n]**  
Output: if **comm->flag** = 2 on entry, then **objfun** must set **g[j - 1]** to the value of the first derivative  $\partial F/\partial x_j$  at the current point,  $x$  for  $j = 1, 2, \dots, n$ . If it is not possible to evaluate the first derivatives then **objfun** should assign a negative value to **comm->flag**; nag\_opt\_bounds\_deriv will then terminate.  
(If **comm->flag** = 0 on entry, **objfun** must **not** change the elements of **g**.)

**comm**  
Pointer to structure of type **Nag\_Comm**; the following members are relevant to **objfun**.

**flag** – Integer  
Input: **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.  
Output: if **objfun** resets **comm->flag** to some negative number then nag\_opt\_bounds\_deriv will terminate immediately with the error indicator **NE\_USER\_STOP**. If **fail** is supplied to nag\_opt\_bounds\_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 calculations 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_bounds_deriv` these pointers may be allocated memory by the user and initialized with various quantities for use by **objfun** when called from `nag_opt_bounds_deriv`.

**Note:** **objfun** should be tested separately before being used in conjunction with `nag_opt_bounds_deriv`. The array **x** must **not** be changed by **objfun**.

**bound**

Input: indicates whether the problem is unconstrained or bounded and, if it is bounded, whether the facility for dealing with bounds of special forms is to be used. **bound** should be set to one of the following values:

**bound** = **Nag\_Bounds**

if the variables are bounded and the user will be supplying all the  $l_j$  and  $u_j$  individually.

**bound** = **Nag\_NoBounds**

if the problem is unconstrained.

**bound** = **Nag\_BoundsZero**

if the variables are bounded, but all the bounds are of the form  $0 \leq x_j$ .

**bound** = **Nag\_BoundsEqual**

if all the variables are bounded, and  $l_1 = l_2 = \dots = l_n$  and  $u_1 = u_2 = \dots = u_n$ .

Constraint: **bound** = **Nag\_Bounds**, **Nag\_NoBounds**, **Nag\_BoundsZero** or **Nag\_BoundsEqual**.

**bl[n]**

Input: the lower bounds  $l_j$ .

If **bound** is set to **Nag\_Bounds**, the user must set **bl**[ $j - 1$ ] to  $l_j$ , for  $j = 1, 2, \dots, n$ . (If a lower bound is not required for any  $x_j$ , the corresponding **bl**[ $j - 1$ ] should be set to a large negative number, e.g.,  $-10^{10}$ .)

If **bound** is set to **Nag\_BoundsEqual**, the user must set **bl**[0] to  $l_1$ ; `nag_opt_bounds_deriv` will then set the remaining elements of **bl** equal to **bl**[0].

If **bound** is set to **Nag\_NoBounds** or **Nag\_BoundsZero**, **bl** will be initialized by `nag_opt_bounds_deriv`.

Output: the lower bounds actually used by `nag_opt_bounds_deriv`, e.g., if **bound** = **Nag\_BoundsZero**, **bl**[0] = **bl**[1] =  $\dots$  = **bl**[ $n - 1$ ] = 0.0.

**bu[n]**

Input: the upper bounds  $u_j$ .

If **bound** is set to **Nag\_Bounds**, the user must set **bu**[ $j - 1$ ] to  $u_j$ , for  $j = 1, 2, \dots, n$ . (If an upper bound is not required for any  $x_j$ , the corresponding **bu**[ $j - 1$ ] should be set to a large positive number, e.g.,  $10^{10}$ .)

If **bound** is set to **Nag\_BoundsEqual**, the user must set **bu**[0] to  $u_1$ ; `nag_opt_bounds_deriv` will then set the remaining elements of **bu** equal to **bu**[0].

If **bound** is set to **Nag\_NoBounds** or **Nag\_BoundsZero**, **bu** will be initialized by `nag_opt_bounds_deriv`.

Output: the upper bounds actually used by `nag_opt_bounds_deriv`, e.g., if **bound** = **Nag\_BoundsZero**, **bu**[0] = **bu**[1] =  $\dots$  = **bu**[ $n - 1$ ] =  $10^{10}$ .

**x[n]**

Input: **x**[ $j - 1$ ] must be set to a guess at the  $j$ th component of the position of the minimum, for  $j = 1, 2, \dots, n$ .

Output: the final point  $x^*$ . Thus, if **fail.code** = **NE\_NOERROR** on exit, **x**[ $j - 1$ ] is the  $j$ th component of the estimated position of the minimum.

**objf**

Input: if **options.init\_state** is **Nag\_Init\_None** (the default) or **Nag\_Init\_H\_S**, the user need not initialize **objf**. If **options.init\_state** is **Nag\_Init\_F\_G\_H** or **Nag\_Init\_All**, **objf** must be set on entry to the value of  $F(x)$  at the initial point supplied by the user in **x**.

Output: the function value at the final point given in **x**.

**g[n]**

Input: if **options.init\_state** is **Nag\_Init\_F\_G\_H** or **Nag\_Init\_All**, **g** must be set on entry to the first derivative vector at the initial  $x$ . If **options.init\_state** is **Nag\_Init\_None** or **Nag\_Init\_H\_S**, **g** need not be set.

Output: the first derivative vector corresponding to the final point in **x**. The elements of **g** corresponding to free variables should normally be close to zero.

**options**

Input/Output: a pointer to a structure of type **Nag\_E04\_Opt** whose members are optional parameters for nag\_opt\_bounds\_deriv. These structure members offer the means of adjusting some of the parameter values of the algorithm and on output will supply further details of the results. A description of the members of **options** is given below in Section 7. Some of the results returned in **options** can be used by nag\_opt\_bounds\_deriv to perform a 'warm start' if it is re-entered (see the member **init\_state** in Section 7.2).

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\_bounds\_deriv. However, if the optional parameters are not required the NAG defined null pointer, E04\_DEFAULT, can be used in the function call.

**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\_bounds\_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.

**4.1. Description of Printed Output**

Intermediate and final results are printed out by default. The level of printed output can be controlled by the user with the structure member **options.print\_level** (see Section 7.2). The default print level of **Nag\_Soln\_Iter** provides a single line of output at each iteration and the final result. This section describes the default printout produced by nag\_opt\_bounds\_deriv.

The following line of output is produced at each iteration. In all cases the values of the quantities printed are those in effect *on completion* of the given iteration.

<b>Itn</b>	the iteration count, $k$ .
<b>Nfun</b>	the cumulative number of calls made to <b>objfun</b> .
<b>Objective</b>	the value of the objective function, $F(x^{(k)})$
<b>Norm g</b>	the Euclidean norm of the projected gradient vector, $\ g_z(x^{(k)})\ $ .
<b>Norm x</b>	the Euclidean norm of $x^{(k)}$ .
<b>Norm(x(k-1)-x(k))</b>	the Euclidean norm of $x^{(k-1)} - x^{(k)}$ .
<b>Step</b>	the step $\alpha^{(k)}$ taken along the computed search direction $p^{(k)}$ .
<b>Cond H</b>	the ratio of the largest to the smallest element of the diagonal factor $D$ of the projected Hessian matrix. This quantity is usually a good estimate of the condition number of the projected Hessian matrix. (If no variables are currently free, this value will be zero.)

The printout of the final result consists of:

**x**                    the final point,  $x^*$ .  
**g**                    the final projected gradient vector,  $g_z(x^*)$ .  
**Status**              the final state of the variable with respect to its bound(s).

## 5. Comments

A list of possible error exits and warnings from nag\_opt\_bounds\_deriv is given in Section 8. Details of timing, scaling, accuracy and the use of nag\_opt\_bounds\_deriv for unconstrained minimization are given in Section 9.

## 6. Example 1

This example minimizes the function

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

subject to the bounds

$$\begin{aligned} 1 &\leq x_1 \leq 3 \\ -2 &\leq x_2 \leq 0 \\ 1 &\leq x_4 \leq 3 \end{aligned}$$

starting from the initial guess  $(3.0, -0.9, 0.13, 1.1)^T$ .

This example shows the simple use of nag\_opt\_bounds\_deriv where default values are used for all optional parameters. An example showing the use of optional parameters is given in Section 12. There is one example program file, the main program of which calls both examples. The main program and Example 1 are given below.

### 6.1. Program Text

```
/* nag_opt_bounds_deriv(e04kbc) Example Program
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 */

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

#ifdef NAG_PROTO
static void objfun(Integer n, double x[], double *f,
                  double g[], Nag_Comm *comm);
static void ex1(void);
static void ex2(void);
#else
static void objfun();
static void ex1();
static void ex2();
#endif

#define NMAX 4

main()
{
    /* Two examples are called, ex1() which uses the
     * default settings to solve the problem and
     * ex2() which solves the same problem with
     * some optional parameters set by the user.
    */
}
```

```

    */

    Vprintf("e04kbc Example Program Results.\n");
    ex1();
    ex2();
    exit(EXIT_SUCCESS);
}

#ifdef NAG_PROTO
static void objfun(Integer n, double x[], double *objf,
                  double g[], Nag_Comm *comm)
#else
static void objfun(n, x, objf, g, comm)
    Integer n;
    double x[];
    double *objf;
    double g[];
    Nag_Comm *comm;
#endif
{
    /* Routine to evaluate objective function. */

    double x1, x2, x3, x4;
    double tmp, tmp1, tmp2, tmp3, tmp4;

    x1 = x[0];
    x2 = x[1];
    x3 = x[2];
    x4 = x[3];

    /* Supply a single function value */
    tmp1 = x1 + 10.0*x2;
    tmp2 = x3 - x4;
    tmp3 = x2 - 2.0*x3, tmp3 *= tmp3;
    tmp4 = x1 - x4, tmp4 *= tmp4;
    *objf = tmp1*tmp1 + 5.0*tmp2*tmp2 + tmp3*tmp3 + 10.0*tmp4*tmp4;

    if (comm->flag != 0)
    {
        tmp = x1 - x4;
        g[0] = 2.0*(x1 + 10.0*x2) + 40.0*tmp*tmp*tmp;
        tmp = x2 - 2.0*x3;
        g[1] = 20.0*(x1 + 10.0*x2) + 4.0*tmp*tmp*tmp;
        tmp = x2 - 2.0*x3;
        g[2] = 10.0*(x3 - x4) - 8.0*tmp*tmp*tmp;
        tmp = x1 - x4;
        g[3] = 10.0*(x4 - x3) - 40.0*tmp*tmp*tmp;
    }
}

/* objfun */

static void ex1()
{
    double x[NMAX], g[NMAX], bl[NMAX], bu[NMAX];

    double objf;
    Integer n;
    Nag_BoundType bound;
    static NagError fail;

    fail.print = TRUE;

    Vprintf("\ne04kbc example 1: no option setting.\n");

    n = NMAX;
    x[0] = 3.0;
    x[1] = -0.9;
    x[2] = 0.13;
    x[3] = 1.1;
}

```

```

/* Set bounds on variables */
bound = Nag_Bounds;
bl[0] = 1.0;
bu[0] = 3.0;
bl[1] = -2.0;
bu[1] = 0.0;
/* Third variable is not bounded, so third lower bound
 * is set to a large negative number and third upper
 * bound to a large positive number.
 */
bl[2] = -1.0e10;
bu[2] = 1.0e10;
bl[3] = 1.0;
bu[3] = 3.0;

/* Call optimization routine */
e04kbc(n, objfun, bound, bl, bu, x, &objf,
      g, E04_DEFAULT, NAGCOMM_NULL, &fail);

if (fail.code != NE_NOERROR && fail.code != NW_COND_MIN) exit(EXIT_FAILURE);
} /* ex1 */

```

## 6.2. Program Data

None; but there is an example data file which contains the optional parameter values for Example 2 below.

## 6.3. Program Results

e04kbc Example Program Results.

e04kbc example 1: no option setting.

Parameters to e04kbc

-----

Number of variables..... 4

optim_tol.....	1.05e-07	linesearch_tol.....	9.00e-01
step_max.....	1.00e+05	max_iter.....	200
init_state.....	Nag_Init_None	local_search.....	TRUE
minlin.....	Nag_Lin_Deriv	deriv_check.....	TRUE
print_level.....	Nag_Soln_Iter	machine precision.....	1.11e-16
outfile.....	stdout		

Memory allocation:

state.....	Nag	hesd.....	Nag
hesl.....	Nag		

Results from e04kbc:

-----

Iteration results:

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
0	0	1.7284e+02	2.9e+02	3.3e+00			1.0e+00
1	2	2.6405e+01	2.4e+01	3.7e+00	1.2e+00	4.2e-03	3.5e+01
2	4	2.2918e+01	2.1e+01	3.7e+00	2.9e-01	1.2e-02	3.1e+01
3	7	2.1889e+01	2.5e+01	3.8e+00	2.9e-01	1.1e-02	2.3e+02
4	9	1.8034e+01	3.2e+01	3.1e+00	9.7e-01	4.7e-02	1.3e+01
5	10	1.4651e+01	1.6e+01	2.9e+00	2.3e-01	1.0e+00	1.8e+01
6	11	1.2616e+01	1.9e+01	2.6e+00	2.5e-01	1.0e+00	5.5e+01
7	12	8.2326e+00	1.3e+01	2.0e+00	6.5e-01	6.3e-01	6.6e+00
8	13	5.9714e+00	1.2e+01	1.7e+00	2.8e-01	7.0e-01	3.4e+00
9	14	3.2421e+00	6.2e+00	1.5e+00	3.5e-01	1.0e+00	7.3e+00
10	15	3.1946e+00	1.8e+01	1.5e+00	8.2e-03	1.9e-02	9.9e+01
11	17	2.4347e+00	6.2e-01	1.5e+00	8.8e-02	5.0e-03	2.1e+00
12	18	2.4338e+00	1.6e-02	1.5e+00	3.0e-03	1.0e+00	2.0e+00
13	19	2.4338e+00	8.3e-03	1.5e+00	1.6e-04	1.0e+00	4.1e+00
14	20	2.4338e+00	7.7e-06	1.5e+00	1.8e-04	1.0e+00	4.1e+00

15	21	2.4338e+00	1.5e-08	1.5e+00	9.4e-08	1.0e+00	4.1e+00
16	22	2.4338e+00	1.5e-08	1.5e+00	8.3e-11	1.0e+00	1.0e+00
17	23	2.4338e+00	1.5e-08	1.5e+00	8.3e-11	1.0e+00	1.0e+00

Local search performed.

Final solution:

17	27	2.4338e+00	1.5e-08	1.5e+00	1.5e-08	1.0e+00	1.0e+00
----	----	------------	---------	---------	---------	---------	---------

Variable	x	g	Status
1	1.0000e+00	2.9535e-01	Lower Bound
2	-8.5233e-02	-1.5114e-08	Free
3	4.0930e-01	-2.5169e-10	Free
4	1.0000e+00	5.9070e+00	Lower Bound

## 7. Optional Parameters

A number of optional input and output parameters to nag\_opt\_bounds\_deriv are available through the structure argument **options**, type Nag\_E04\_Opt. A parameter may be selected by assigning an appropriate value to the relevant structure member; those parameters not selected will be assigned default values. If no use is to be made of any of the optional parameters the user should use the NAG defined null pointer, E04\_DEFAULT, in place of **options** when calling nag\_opt\_bounds\_deriv; the default settings will then be used for all parameters.

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.

If assignment of functions and memory to pointers in the **options** structure is required, then this must be done directly in the calling program; they cannot be assigned using using nag\_opt\_read (e04xyc).

### 7.1. Optional Parameter Checklist and Default Values

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

Boolean list	<b>TRUE</b>
Nag_PrintType print_level	<b>Nag_Soln_Iter</b>
char outfile[80]	stdout
void (*print_fun)()	NULL
Boolean deriv_check	<b>TRUE</b>
Nag_InitType init_state	<b>Nag_Init_None</b>
Integer max_iter	50n
double optim_tol	$10\sqrt{\epsilon}$
Nag_LinFun minlin	<b>Nag_Lin_Deriv</b>
double linesearch_tol	0.9 (0.0 if n = 1)
double step_max	100000.0
double f_est	
Boolean local_search	<b>TRUE</b>
Integer *state	size n
double *hesl	size max(n(n-1)/2,1)
double *hesd	size n
Integer iter	
Integer nf	



## 7.2. Description of Optional Parameters

**list** – Boolean Default = **TRUE**

Input: if **options.list** = **TRUE** the parameter settings in the call to `nag_opt_bounds_deriv` will be printed.

**print\_level** – Nag\_PrintType Default = **Nag\_Soln\_Iter**

Input: the level of results printout produced by `nag_opt_bounds_deriv`. The following values are available.

<b>Nag_NoPrint</b>	No output.
<b>Nag_Soln</b>	The final solution.
<b>Nag_Iter</b>	One line of output for each iteration.
<b>Nag_Soln_Iter</b>	The final solution and one line of output for each iteration.
<b>Nag_Soln_Iter_Full</b>	The final solution and detailed printout at each iteration.

Details of each level of results printout are described in Section 7.3.

Constraint: **options.print\_level** = **Nag\_NoPrint**, **Nag\_Soln**, **Nag\_Iter**, **Nag\_Soln\_Iter** or **Nag\_Soln\_Iter\_Full**.

**outfile** – char[80] Default = **stdout**

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

**print\_fun** – pointer to function Default = **NULL**

Input: printing function defined by the user; the prototype of **print\_fun** is

```
void (*print_fun)(const Nag_Search_State *st, Nag_Comm *comm);
```

See Section 7.3.1 below for further details.

**deriv\_check** – Boolean Default = **TRUE**

If **options.init\_state** is set to a value  $\neq$  **Nag\_Init\_None** then the default of **deriv\_check** is changed to **FALSE**.

Input: if **options.deriv\_check** = **TRUE** a check of the derivatives defined by **objfun** will be made at the starting point **x**. The derivative check is carried out by a call to `nag_opt_check_deriv` (e04hcc). If **options.init\_state** is set to a value other than its default value of **Nag\_Init\_None** then the default of **deriv\_check** will be **FALSE**. A starting point of  $x = 0$  or  $x = 1$  should be avoided if this test is to be meaningful, if either of these starting points is necessary then `nag_opt_check_deriv` (e04hcc) should be used to check **objfun** at a different point prior to calling `nag_opt_bounds_deriv`.

**init\_state** – Nag\_InitType Default = **Nag\_Init\_None**

Input: **init\_state** specifies which of the parameters **objf**, **g**, **options.hesl**, **options.hesd** and **options.state** are actually being initialized by the user. Such information will generally reduce the time taken by `nag_opt_bounds_deriv`.

**init\_state** = **Nag\_Init\_None**

No values are assumed to have been set in any of **objf**, **g**, **options.hesl**, **options.hesd** or **options.state**. (`nag_opt_bounds_deriv` will use the unit matrix as the initial estimate of the Hessian matrix.)

**init\_state** = **Nag\_Init\_F\_G\_H**

The parameters **objf** and **g** must contain the value of  $F(x)$  and its first derivatives at the starting point. The elements **options.hesd**[ $j - 1$ ] must have been set to estimates of the derivatives  $\partial^2 F / \partial x_j^2$  at the starting point. No values are assumed to have been set in **options.hesl** or **options.state**.

**init\_state** = **Nag\_Init\_All**

The parameters **objf** and **g** must contain the value of  $F(x)$  and its first derivatives at the starting point. All  $n$  elements of **options.state** must have been set to indicate

which variables are on their bounds and which are free. **options.hesl** and **options.hesd** must contain the Cholesky factors of a positive-definite approximation to the  $n_z$  by  $n_z$  Hessian matrix for the subspace of free variables. (This option is useful for restarting the minimization process if **options.max\_iter** is reached.)

**init\_state** = Nag\_Init\_H\_S

No values are assumed to have been set in **objf** or **g**, but **options.hesl**, **options.hesd** and **options.state** must have been set as for **init\_state** = Nag\_Init\_All. (This option is useful for starting off a minimization run using second derivative information from a previous, similar, run.)

Constraint: **options.init\_state** = Nag\_Init\_None or Nag\_Init\_F\_G\_H or Nag\_Init\_All or Nag\_Init\_H\_S.

**max\_iter** – Integer Default = 50n

Input: the limit on the number of iterations allowed before termination.

Constraint: **options.max\_iter**  $\geq$  0.

**optim\_tol** – double Default =  $10\sqrt{\epsilon}$

Input: the accuracy in  $x$  to which the solution is required.

If  $x_{\text{true}}$  is the true value of  $x$  at the minimum, then  $x_{\text{sol}}$ , the estimated position prior to a normal exit, is such that

$$\|x_{\text{sol}} - x_{\text{true}}\| < \text{optim\_tol} \times (1.0 + \|x_{\text{true}}\|),$$

where  $\|y\| = \sqrt{\sum_{j=1}^n y_j^2}$ . For example, if the elements of  $x_{\text{sol}}$  are not much larger than 1.0 in

modulus and if **optim\_tol** is set to  $10^{-5}$ , then  $x_{\text{sol}}$  is usually accurate to about 5 decimal places. (For further details see Section 9.3.)

If the problem is scaled roughly as described in Section 9.2 and  $\epsilon$  is the *machine precision*, then  $\sqrt{\epsilon}$  is probably the smallest reasonable choice for **optim\_tol**. (This is because, normally, to machine accuracy,  $F(x + \sqrt{\epsilon}e_j) = F(x)$  where  $e_j$  is any column of the identity matrix.)

Constraint:  $\epsilon \leq \text{options.optim\_tol} < 1.0$ .

**minlin** – Nag\_LinFun Default = Nag\_Lin\_Deriv

Input: **minlin** specifies whether the linear minimizations (i.e., minimizations of  $F(x + \alpha p)$  with respect to  $\alpha$ ) are to be performed by a function which just requires the evaluation of  $F(x)$ , **Nag\_Lin\_NoDeriv**, or by a function which also requires the first derivatives of  $F(x)$ , **Nag\_Lin\_Deriv**.

It will often be possible to evaluate the first derivatives of  $F$  in about the same amount of computer time that is required for the evaluation of  $F$  itself – if this is so then nag\_opt\_bounds\_deriv should be called with **minlin** set to **Nag\_Lin\_Deriv**. However, if the evaluation of the derivatives takes more than about 4 times as long as the evaluation of  $F$ , then a setting of **Nag\_Lin\_NoDeriv** will usually be preferable. If in doubt, use the default setting **Nag\_Lin\_Deriv** as it is slightly more robust.

Constraint: **options.minlin** = Nag\_Lin\_Deriv or Nag\_Lin\_NoDeriv.

**linesearch\_tol** – double Default = 0.9 if  $n > 1$ , and 0.0 otherwise

If **options.minlin** is set to **Nag\_Lin\_NoDeriv** then the default value of **linesearch\_tol** will be changed from 0.9 to 0.5 if  $n > 1$ .

Input: every iteration of nag\_opt\_bounds\_deriv involves a linear minimization (i.e., minimization of  $F(x + \alpha p)$  with respect to  $\alpha$ ). **linesearch\_tol** specifies how accurately these linear minimizations are to be performed. The minimum with respect to  $\alpha$  will be located more accurately for small values of **linesearch\_tol** (say 0.01) than for large values (say 0.9).

Although accurate linear minimizations will generally reduce the number of iterations performed by nag\_opt\_bounds\_deriv, they will increase the number of function evaluations required for each iteration. On balance, it is usually more efficient to perform a low accuracy linear minimization.

A smaller value such as 0.01 may be worthwhile:

- (a) if **objfun** takes so little computer time that it is worth using extra calls of **objfun** to reduce the number of iterations and associated matrix calculations
- (b) if  $F(x)$  is a penalty or barrier function arising from a constrained minimization problem (since such problems are very difficult to solve)
- (c) if **options.minlin** is set to **Nag\_Lin\_NoDeriv** and the calculation of first derivatives takes so much computer time (relative to the time taken to evaluate the function) that it is worth using extra function evaluations to reduce the number of derivative evaluations.

If  $\mathbf{n} = 1$ , the default for **linesearch\_tol** = 0.0 (if the problem is effectively 1-dimensional then **linesearch\_tol** should be set to 0.0 by the user even though  $\mathbf{n} > 1$ ; i.e., if for all except one of the variables the lower and upper bounds are equal).

Constraint:  $0.0 \leq \mathbf{options.linesearch\_tol} < 1.0$ .

**step\_max** – double

Default = 100000.0

Input: an estimate of the Euclidean distance between the solution and the starting point supplied by the user. (For maximum efficiency a slight overestimate is preferable.) **nag\_opt\_bounds\_deriv** will ensure that, for each iteration,

$$\sqrt{\sum_{j=1}^n [x_j^{(k)} - x_j^{(k-1)}]^2} \leq \mathbf{step\_max},$$

where  $k$  is the iteration number. Thus, if the problem has more than one solution, **nag\_opt\_bounds\_deriv** is most likely to find the one nearest the starting point. On difficult problems, a realistic choice can prevent the sequence of  $x^{(k)}$  entering a region where the problem is ill-behaved and can also help to avoid possible overflow in the evaluation of  $F(x)$ . However an underestimate of **step\_max** can lead to inefficiency.

Constraint: **options.step\_max**  $\geq$  **options.optim\_tol**.

**f\_est** – double

Input: an estimate of the function value at the minimum. This estimate is just used for calculating suitable step lengths for starting linear minimizations off, so the choice is not too critical. However, it is better for **f\_est** to be set to an underestimate rather than to an overestimate. If no value is supplied then an initial step length of 1.0, subject to the variable bounds, will be used.

**local\_search** – Boolean

Default = **TRUE**

Input: **local\_search** must specify whether or not the user wishes a ‘local search’ to be performed when a point is found which is thought to be a constrained minimum. If **local\_search** = **TRUE** and either the quasi-Newton direction of search fails to produce a lower function value or the convergence criteria are satisfied, then a local search will be performed. This may move the search away from a saddle point or confirm that the final point is a minimum. If **local\_search** = **FALSE** there will be no local search when a point is found which is thought to be a minimum.

The amount of work involved in a local search is comparable to twice that required in a normal iteration to minimize  $F(x + \alpha p)$  with respect to  $\alpha$ . For most problems this will be small (relative to the total time required for the minimization).

**local\_search** could be set **FALSE** if:

- it is known from the physical properties of a problem that a stationary point will be the required minimum;
- a point which is not a minimum could be easily recognized, for example if the value of  $F(x)$  at the minimum is known.

**state** – Integer \*

Default memory = **n**

Input: **state** need not be set if the default option of **options.init\_state** = **Nag\_Init\_None** is used as **n** values of memory will be automatically allocated by **nag\_opt\_bounds\_deriv**.

If the option **init\_state** = **Nag\_Init\_All** or **init\_state** = **Nag\_Init\_H\_S** has been chosen, **state** must point to a minimum of **n** elements of memory. This memory will already be available if the calling program has used the **options** structure in a previous call to **nag\_opt\_bounds\_deriv**

with `options.init_state = Nag_Init_None` and the same value of `n`. If a previous call has not been made sufficient memory must be allocated by the user.

When `init_state = Nag_Init_All` or `Nag_Init_H_S` then `state` must specify information about which variables are currently on their bounds and which are free. If  $x_j$  is:

- (a) fixed on its upper bound, `state[j - 1]` is `-1`;
- (b) fixed on its lower bound, `state[j - 1]` is `-2`;
- (c) effectively a constant (i.e.,  $l_j = u_j$ ), `state[j - 1]` is `-3`;
- (d) free, `state[j - 1]` gives its position in the sequence of free variables.

If `options.init_state` is set to `Nag_Init_None` or `Nag_Init_F_G_H`, `state` will be initialized by `nag_opt_bounds_deriv`.

If `options.init_state` is set to `Nag_Init_All` or `Nag_Init_H_S`, `state` must be initialized by the user before `nag_opt_bounds_deriv` is called.

Output: `state` gives information as above about the final point given in `x`.

`hesl` – double \* Default memory =  $\max(\mathbf{n}(\mathbf{n}-1)/2, 1)$   
`hesd` – double \* Default memory = `n`

Input: `hesl` and `hesd` need not be set if the default of `options.init_state = Nag_Init_None` is used as sufficient memory will be automatically allocated by `nag_opt_bounds_deriv`.

If `options.init_state = Nag_Init_All` or `options.init_state = Nag_Init_H_S` has been set then `hesl` must point to a minimum of  $\max(\mathbf{n}(\mathbf{n}-1)/2, 1)$  elements of memory.

`hesd` must point to at least `n` elements of memory if `options.init_state = Nag_Init_F_G_H`, `options.init_state = Nag_Init_All` or `options.init_state = Nag_Init_H_S` has been chosen.

The appropriate amount of memory will already be available for `hesl` and `hesd` if the calling program has used the `options` structure in a previous call to `nag_opt_bounds_deriv` with `options.init_state = Nag_Init_None` and the same value of `n`. If a previous call has not been made sufficient memory must be allocated by the user.

`hesl` and `hesd` are used to store the factors  $L$  and  $D$  of the current approximation to the matrix of second derivatives with respect to the free variables (see Section 3). (The elements of the matrix are assumed to be ordered according to the permutation specified by the positive elements of `state`, see above.) `hesl` holds the lower triangle of  $L$ , omitting the unit diagonal, stored by rows. `hesd` stores the diagonal elements of  $D$ . Thus if  $n_z$  elements of `state` are positive, the strict lower triangle of  $L$  will be held in the first  $n_z (n_z - 1)/2$  elements of `hesl` and the diagonal elements of  $D$  in the first  $n_z$  elements of `hesd`.

If `options.init_state = Nag_Init_None` (the default), `hesl` and `hesd` will be initialized within `nag_opt_bounds_deriv` to the factors of the unit matrix.

If the user sets `options.init_state` to `Nag_Init_F_G_H`, `hesd[j - 1]` must contain on entry an approximation to the second derivative with respect to  $x_j$ , for  $j = 1, 2, \dots, n$ . `hesl` need not be set.

If the user sets `options.init_state` to `Nag_Init_All` or `Nag_Init_H_S`, `hesl` and `hesd` must contain on entry the Cholesky factors of a positive-definite approximation to the  $n_z$  by  $n_z$  matrix of second derivatives for the subspace of free variables as specified by the user's setting of `state`.

Output: `hesl` and `hesd` hold the factors  $L$  and  $D$  corresponding to the final point given in `x`. The elements of `hesd` are useful for deciding whether to accept the result produced by `nag_opt_bounds_deriv` (see Section 9).

`iter` – Integer

Output: the number of iterations which have been performed in `nag_opt_bounds_deriv`.

`nf` – Integer

Output: the number of times the residuals have been evaluated (i.e., number of calls of `objfun`).

### 7.3. Description of Printed Output

The level of printed output can be controlled with the structure members `options.list` and `options.print_level` (see Section 7.2). If `list = TRUE` then the parameter values to `nag_opt_bounds_deriv` are listed, whereas the printout of results is governed by the value of

**print\_level.** The default of **print\_level = Nag\_Soln\_Iter** provides a single line of output at each iteration and the final result. This section describes all of the possible levels of results printout available from `nag_opt_bounds_deriv`.

When **print\_level = Nag\_Iter** or **Nag\_Soln\_Iter** a single line of output is produced on completion of each iteration, this gives the following values:

<b>Itn</b>	the iteration count, $k$ .
<b>Nfun</b>	the cumulative number of calls to <b>objfun</b> .
<b>Objective</b>	the current value of the objective function, $F(x^{(k)})$
<b>Norm g</b>	the Euclidean norm of the projected gradient vector, $\ g_z(x^{(k)})\ $ .
<b>Norm x</b>	the Euclidean norm of $x^{(k)}$ .
<b>Norm(x(k-1)-x(k))</b>	the Euclidean norm of $x^{(k-1)} - x^{(k)}$ .
<b>Step</b>	the step $\alpha^{(k)}$ taken along the computed search direction $p^{(k)}$ .
<b>Cond H</b>	the ratio of the largest to the smallest element of the diagonal factor $D$ of the projected Hessian matrix. This quantity is usually a good estimate of the condition number of the projected Hessian matrix. (If no variables are currently free, this value will be zero.)

When **options.print\_level = Nag\_Soln\_Iter\_Full** more detailed results are given at each iteration. Additional values output are

<b>x</b>	the current point $x^{(k)}$ .
<b>g</b>	the current projected gradient vector, $g_z(x^{(k)})$ .
<b>Status</b>	the current state of the variable with respect to its bound(s).

If **options.print\_level = Nag\_Soln** or **Nag\_Soln\_Iter** or **Nag\_Soln\_Iter\_Full** the final result is printed out. This consists of:

<b>x</b>	the final point, $x^*$ .
<b>g</b>	the final projected gradient vector, $g_z(x^*)$ .
<b>Status</b>	the final state of the variable with respect to its bound(s).

If **options.print\_level = Nag\_NoPrint** then printout will be suppressed; the user can print the final solution when `nag_opt_bounds_deriv` returns to the calling program.

### 7.3.1. Output of Results via a User-defined Printing Function

Users may also specify their own print function for output of iteration results and the final solution by use of the **options.print\_fun** function pointer, which has prototype

```
void (*print_fun)(const Nag_Search_State *st, Nag_Comm *comm);
```

The rest of this section can be skipped if the default printing facilities provide the required functionality.

When a user defined function is assigned to **options.print\_fun** this will be called in preference to the internal print function of `nag_opt_bounds_deriv`. Calls to the user defined function are again controlled by means of the **options.print\_level** member. Information is provided through **st** and **comm**, the two structure arguments to **print\_fun**.

The results contained in the members of **st** are those on completion of the last iteration or those after a local search. (An iteration may be followed by a local search (see **options.local\_search**, Section 7.2) in which case **print\_fun** is called with the results of the last iteration (**st.local\_search = FALSE**) and then again when the local search has been completed (**st.local\_search = TRUE**)).

If **comm->it\_prt = TRUE** then the results on completion of an iteration of `nag_opt_bounds_deriv` are contained in the members of **st**. If **comm->sol\_prt = TRUE** then the final results from `nag_opt_bounds_deriv`, including details of the final iteration, are contained in the members of **st**. In both cases, the same members of **st** are set, as follows:

**iter** – Integer  
the current iteration count,  $k$ , if **comm->it\_prt** = **TRUE**; the final iteration count,  $k$ , if **comm->sol\_prt** = **TRUE**.

**n** – Integer  
the number of variables.

**x** – double \*  
the co-ordinates of the point  $x^{(k)}$ .

**f** – double  
the value of the current objective function.

**g** – double \*  
points to the **n** memory locations holding the first derivatives of  $F$  at the current point  $x^{(k)}$ .

**gpj\_norm** – double  
the Euclidean norm of the current projected gradient  $g_z$ .

**step** – double  
the step  $\alpha^{(k)}$  taken along the search direction  $p^{(k)}$ .

**cond** – double  
the estimate of the condition number of the Hessian matrix.

**xk\_norm** – double  
the Euclidean norm of  $x^{(k-1)} - x^{(k)}$ .

**state** – Integer \*  
the status of variables  $x_j$ ,  $j = 1, 2, \dots, n$ , with respect to their bounds. See Section 7.2 for a description of the possible status values.

**local\_search** – Boolean  
**TRUE** if a local search has been performed.

**nf** – Integer  
the cumulative number of calls made to **objfun**.

The relevant members of the structure **comm** are:

**it\_prt** – Boolean  
will be **TRUE** when the print function is called with the results of the current iteration.

**sol\_prt** – Boolean  
will be **TRUE** when the print function is called with the final result.

**user** – double \*  
**iuser** – Integer \*  
**p** – Pointer  
pointers for communication of user information. If used they must be allocated memory by the user either before entry to **nag\_opt\_bounds\_deriv** or during a call to **objfun** or **print\_fun**. The type Pointer will be **void \*** with a C compiler that defines **void \*** and **char \*** otherwise.

## 8. Error Indications and Warnings

### NE\_USER\_STOP

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

This exit occurs if the user sets **comm->flag** to a negative value in **objfun**. If **fail** is supplied the value of **fail.errnum** will be the same as the user's setting of **comm->flag**.

### NE\_INT\_ARG\_LT

On entry, **n** must not be less than 1: **n** =  $\langle value \rangle$ .

### NE\_BOUND

The lower bound for variable  $\langle value \rangle$  (array element **bl**[ $\langle value \rangle$ ]) is greater than the upper bound.

**NE\_DERIV\_ERRORS**

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

**NE\_OPT\_NOT\_INIT**

Options structure not initialized.

**NE\_BAD\_PARAM**

On entry parameter **bound** had an illegal value.

On entry parameter **options.print\_level** had an illegal value.

On entry parameter **options.init\_state** had an illegal value.

**NE\_2\_REAL\_ARG\_LT**

On entry, **options.step\_max** =  $\langle value \rangle$  while **options.optim\_tol** =  $\langle value \rangle$ . These parameters must satisfy **step\_max**  $\geq$  **optim\_tol**.

**NE\_INVALID\_INT\_RANGE\_1**

Value  $\langle value \rangle$  given to **options.max\_iter** not valid. Correct range is **max\_iter**  $\geq 0$ .

**NE\_INVALID\_REAL\_RANGE\_EF**

Value  $\langle value \rangle$  given to **options.optim\_tol** not valid. Correct range is  $\epsilon \leq \mathbf{optim\_tol} < 1.0$ .

**NE\_INVALID\_REAL\_RANGE\_FF**

Value  $\langle value \rangle$  given to **options.linesearch\_tol** not valid. Correct range is  $0.0 \leq \mathbf{linesearch\_tol} < 1.0$ .

**NE\_INIT\_MEM**

Option **init\_state** =  $\langle string \rangle$  but the pointer  $\langle string \rangle$  in the option structure has not been allocated memory.

**NE\_NO\_MEM**

Option **init\_state** =  $\langle string \rangle$  but at least one of the pointers  $\langle string \rangle$  in the option structure has not been allocated memory.

**NE\_HESD**

The initial values of the supplied **options.hesd** has some value(s) which is negative or too small or the ratio of the largest element of **hesd** to the smallest is too large.

**NE\_ALLOC\_FAIL**

Memory allocation failed.

When one of the above exits occurs, no values will have been assigned by `nag_opt_bounds_deriv` to **objf** or to the elements of **g**, **options.hesl**, or **options.hesd**.

**NW\_TOO\_MANY\_ITER**

The maximum number of iterations,  $\langle value \rangle$ , have been performed.

If steady reductions in  $F(x)$ , were monitored up to the point where this exit occurred, then the exit probably occurred simply because **options.max\_iter** was set too small, so the calculations should be restarted from the final point held in **x**. This exit may also indicate that  $F(x)$  has no minimum.

**NW\_COND\_MIN**

The conditions for a minimum have not all been satisfied, but a lower point could not be found.

Provided that, on exit, the first derivatives of  $F(x)$  with respect to the free variables are sufficiently small, and that the estimated condition number of the second derivative matrix is not too large, this error exit may simply mean that, although it has not been possible to satisfy the specified requirements, the algorithm has in fact found the minimum as far as the accuracy of the machine permits. This could be because **options.optim\_tol** has been set so small that rounding error in **objfun** makes attainment of the convergence conditions impossible.

If the estimated condition number of the approximate Hessian matrix at the final point is large, it could be that the final point is a minimum but that the smallest eigenvalue of the second derivative matrix is so close to zero that it is not possible to recognize the point as a minimum.

**NE\_CHOLESKY\_OVERFLOW**

An overflow would have occurred during the updating of the Cholesky factors if the calculations had been allowed to continue. Restart from the current point with `options.init_state = Nag_Init_None`.

**NW\_LOCAL\_SEARCH**

The local search has failed to find a feasible point which gives a significant change of function value.

If the problem is a genuinely unconstrained one, this type of exit indicates that the problem is extremely ill conditioned or that the function has no minimum. If the problem has bounds which may be close to the minimum, it may just indicate that steps in the subspace of free variables happened to meet a bound before they changed the function value.

An exit of `fail.code = NW_TOO_MANY_ITER`, `NW_COND_MIN` or `NW_LOCAL_SEARCH` may also be caused by mistakes in `objfun`, by the formulation of the problem or by an awkward function. If there are no such mistakes, it is worth restarting the calculations from a different starting point (not the point at which the failure occurred) in order to avoid the region which caused the failure.

**NE\_NOT\_APPEND\_FILE**

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

**NE\_WRITE\_ERROR**

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

**NE\_NOT\_CLOSE\_FILE**

Cannot close file  $\langle string \rangle$ .

**9. Further Comments****9.1. Timing**

The number of iterations required depends on the number of variables, the behaviour of  $F(x)$ , the accuracy demanded and the distance of the starting point from the solution. The number of multiplications performed in an iteration of `nag_opt_bounds_deriv` is roughly proportional to  $n_z^2$ . In addition, each iteration makes at least one call of `objfun` with `comm->flag = 2` if `options.minlin = Nag_Lin_Deriv` is used or one call of `objfun` with `comm->flag = 0` if `options.minlin = Nag_Lin_NoDeriv` is chosen. So, unless  $F(x)$  can be evaluated very quickly, the run time will be dominated by the time spent in `objfun`.

**9.2. Scaling**

Ideally, the problem should be scaled so that, at the solution,  $F(x)$  and the corresponding values of the  $x_j$  are each in the range  $(-1, +1)$ , and so that at points one unit away from the solution,  $F(x)$  differs from its value at the solution by approximately one unit. This will usually imply that the Hessian matrix at the solution is well conditioned. It is unlikely that the user will be able to follow these recommendations very closely, but it is worth trying (by guesswork), as sensible scaling will reduce the difficulty of the minimization problem, so that `nag_opt_bounds_deriv` will take less computer time.

**9.3. Accuracy**

A successful exit (`fail.code = NE_NOERROR`) is made from `nag_opt_bounds_deriv` when (B1, B2 and B3) or B4 hold, and the local search (if used) confirms a minimum, where

$$\begin{aligned} \text{B1} &\equiv \alpha^{(k)} \times \|p^{(k)}\| < (\text{optim\_tol} + \sqrt{\epsilon}) \times (1.0 + \|x^{(k)}\|) \\ \text{B2} &\equiv |F^{(k)} - F^{(k-1)}| < (\text{optim\_tol}^2 + \epsilon) \times (1.0 + |F^{(k)}|) \\ \text{B3} &\equiv \|g_z^{(k)}\| < (\epsilon^{1/3} + \text{optim\_tol}) \times (1.0 + |F^{(k)}|) \\ \text{B4} &\equiv \|g_z^{(k)}\| < 0.01 \times \sqrt{\epsilon}. \end{aligned}$$

(Quantities with superscript  $k$  are the values at the  $k$ th iteration of the quantities mentioned in Section 3;  $\epsilon$  is the *machine precision*,  $\|\cdot\|$  denotes the Euclidean norm and `optim_tol` is described in Section 7.)



If **fail.code** = **NE\_NOERROR**, then the vector in **x** on exit,  $x_{\text{sol}}$ , is almost certainly an estimate of the position of the minimum,  $x_{\text{true}}$ , to the accuracy specified by **optim.tol**.

If **fail.code** = **NW\_COND\_MIN** or **NW\_LOCAL\_SEARCH**,  $x_{\text{sol}}$  may still be a good estimate of  $x_{\text{true}}$ , but the following checks should be made. Let the largest of the first  $n_z$  elements of **options.hesd** be **hesd[b]**, let the smallest be **hesd[s]**, and define  $k = \text{hesd}[b] / \text{hesd}[s]$ . The scalar  $k$  is usually a good estimate of the condition number of the projected Hessian matrix at  $x_{\text{sol}}$ . If

- (a) the sequence  $\{F(x^{(k)})\}$  converges to  $F(x_{\text{sol}})$  at a superlinear or a fast linear rate,
- (b)  $\|g_z(x_{\text{sol}})\|^2 < 10.0 \times \epsilon$ , and
- (c)  $k < 1.0 / \|g_z(x_{\text{sol}})\|$ ,

then it is almost certain that  $x_{\text{sol}}$  is a close approximation to the position of a minimum. When (b) is true, then usually  $F(x_{\text{sol}})$  is a close approximation to  $F(x_{\text{true}})$ . The quantities needed for these checks are all available in the results printout from `nag_opt_bounds_deriv`; in particular the final value of **Cond H** gives  $k$ .

Further suggestions about confirmation of a computed solution are given in the Chapter Introduction.

#### 9.4. Unconstrained Minimization

If a problem is genuinely unconstrained and has been scaled sensibly, the following points apply:

- (a)  $n_z$  will always be  $n$ ,
- (b) if **options.init.state** is set to **Nag\_Init\_All** or **Nag\_Init\_H\_S** on entry, **options.state[j - 1]** has simply to be set to  $j$ , for  $j = 1, 2, \dots, n$ ,
- (c) **options.hesl** and **options.hesd** will be factors of the full approximate second derivative matrix with elements stored in the natural order,
- (d) the elements of **g** should all be close to zero at the final point,
- (e) the **Status** values given in the printout from `nag_opt_bounds_deriv` and in **options.state** on exit are unlikely to be of interest (unless they are negative, which would indicate that the modulus of one of the  $x_j$  has reached  $10^{10}$  for some reason),
- (f) **Norm g** simply gives the norm of the first derivative vector.

#### 10. References

- Gill P E and Murray W (1972) Quasi-Newton Methods for Unconstrained Optimization *J. Inst. Math. Appl.* **9** 91–108.
- Gill P E and Murray W (1974) Safeguarded steplength algorithms for optimization using descent methods *National Physical Laboratory Report NAC 37*.
- Gill P E and Murray W (1976) Minimization subject to bounds on the variables *National Physical Laboratory Report NAC 72*.
- Gill P E, Murray W and Pitfield R A (1972) The implementation of two revised quasi-Newton algorithms for unconstrained optimization *National Physical Laboratory Report NAC 11*.

#### 11. See Also

`nag_opt_bounds_deriv` (e04kbc)  
`nag_opt_init` (e04xxc)  
`nag_opt_read` (e04xyc)  
`nag_opt_free` (e04xzc)

## 12. Example 2

Example 2 solves the same problem as Example 1 but shows the use of certain optional parameters. The `options` structure is declared and four option values are read from a data file by use of `nag_opt_read` (e04xyc). The memory freeing function `nag_opt_free` (e04xzc) is used to free the memory assigned to the pointers in the option structure. Users should **not** use the standard C function `free()` for this purpose.

### 12.1. Program Text

```
static void ex2()
{
    double x[NMAX], g[NMAX], bl[NMAX], bu[NMAX];

    double objf;
    Integer n;
    Boolean print;
    Nag_BoundType bound;
    Nag_E04_Opt options;
    static NagError fail, fail2;

    fail.print = TRUE;

    Vprintf("\n\ne04kbc example 2: using option setting.\n");

    n = NMAX;
    x[0] = 3.0;
    x[1] = -0.9;
    x[2] = 0.13;
    x[3] = 1.1;

    /* Read option values from file */
    print = TRUE;
    e04xyc("e04kbc", "stdin", &options, print, "stdout", &fail);
    if (fail.code != NE_NOERROR) exit(EXIT_FAILURE);

    bound = Nag_Bounds;
    bl[0] = 1.0;
    bu[0] = 3.0;
    bl[1] = -2.0;
    bu[1] = 0.0;
    /* Third variable is not bounded, so third lower bound
     * is set to a large negative number and third upper
     * bound to a large positive number.
     */
    bl[2] = -1.0e10;
    bu[2] = 1.0e10;
    bl[3] = 1.0;
    bu[3] = 3.0;

    e04kbc(n, objfun, bound, bl, bu, x, &objf,
           g, &options, NAGCOMM_NULL, &fail);

    /* Free memory allocated by e04kbc to pointers hesd, hesl and state */
    fail2.print = TRUE;
    e04xzc(&options, "all", &fail2);
    if (fail.code != NE_NOERROR && fail2.code != NW_COND_MIN ||
        fail2.code != NE_NOERROR) exit(EXIT_FAILURE);
} /* ex2 */
```

### 12.2. Program Data

e04kbc Example Program Data

Following options for e04kbc are read by e04xyc in example 2.

```
begin e04kbc
```

```
print_level = Nag_Soln_Iter_Full /* Print full iterations and solution. */
max_iter = 40 /* Perform maximum of 40 iterations */
```

```

step_max = 4.0 /* Estimate minimum within 4 units of start */
f_est = 0.0 /* Zero is a lower bound on the function value */

end
    
```

### 12.3. Program Results

Optional parameter setting for e04kbc.

-----  
 Option file: stdin

```

print_level set to Nag_Soln_Iter_Full
max_iter set to 40
step_max set to 4.00e+00
f_est set to 0.00e+00
    
```

Parameters to e04kbc

```

-----
Number of variables..... 4

optim_tol..... 1.05e-07      linesearch_tol..... 9.00e-01
f_est..... 0.00e+00
step_max..... 4.00e+00      max_iter..... 40
init_state.....Nag_Init_None local_search..... TRUE
minlin.....Nag_Lin_Deriv    deriv_check..... TRUE
print_level...Nag_Soln_Iter_Full machine precision..... 1.11e-16
outfile..... stdout
    
```

Memory allocation:

```

state..... Nag
hesl..... Nag      hesd..... Nag
    
```

Results from e04kbc:

-----  
 Iteration results:

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
0	0	1.7284e+02	2.9e+02	3.3e+00			1.0e+00

Variable	x	g	Status
1	3.0000e+00	2.6236e+02	Upper Bound
2	-9.0000e-01	-1.2624e+02	Free
3	1.3000e-01	2.7872e+00	Free
4	1.1000e+00	-2.6466e+02	Free

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
1	1	2.6813e+01	2.7e+01	3.7e+00	1.2e+00	4.0e-03	3.4e+01

Variable	x	g	Status
1	3.0000e+00	2.1529e+01	Upper Bound
2	-3.9251e-01	-1.9503e+01	Free
3	1.1880e-01	-1.8450e+01	Free
4	2.1639e+00	-2.9276e+00	Free

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
2	3	2.3453e+01	2.0e+01	3.7e+00	2.6e-01	1.0e-02	4.2e+01

Variable	x	g	Status
1	3.0000e+00	2.9429e+01	Upper Bound
2	-2.2652e-01	1.2204e+01	Free
3	3.1377e-01	-1.3004e+01	Free
4	2.1125e+00	-9.9713e+00	Free

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
3	6	2.1599e+01	2.8e+01	3.7e+00	2.2e-01	8.7e-03	2.3e+02

Variable	x	g	Status
1	3.0000e+00	2.6022e+01	Free
2	-2.5718e-01	-6.5438e-01	Free
3	5.3184e-01	2.3232e+00	Free
4	2.1431e+00	-9.0525e+00	Free

  

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
4	8	1.6189e+01	2.3e+01	3.1e+00	8.0e-01	3.1e-02	8.9e+00

  

Variable	x	g	Status
1	2.1985e+00	-7.4626e-01	Free
2	-2.5748e-01	-1.6715e+01	Free
3	5.3101e-01	2.2448e+00	Free
4	2.1444e+00	1.6128e+01	Free

  

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
5	9	1.4024e+01	1.5e+01	3.0e+00	1.7e-01	1.0e+00	1.8e+01

  

Variable	x	g	Status
1	2.0816e+00	7.7589e-01	Free
2	-1.6942e-01	8.3975e-01	Free
3	5.1518e-01	-1.5466e+00	Free
4	2.0515e+00	1.5362e+01	Free

  

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
6	10	1.1240e+01	2.1e+01	2.6e+00	3.8e-01	1.0e+00	5.4e+01

  

Variable	x	g	Status
1	1.7877e+00	2.1366e+00	Free
2	-7.1798e-02	1.5908e+01	Free
3	5.1964e-01	-2.1209e+00	Free
4	1.8290e+00	1.3096e+01	Free

  

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
7	11	7.9843e+00	1.1e+01	2.1e+00	5.4e-01	6.2e-01	3.9e+00

  

Variable	x	g	Status
1	1.3605e+00	2.5858e+00	Free
2	0.0000e+00	2.3422e+01	Upper Bound
3	4.9104e-01	-2.6186e+00	Free
4	1.5106e+00	1.0331e+01	Free

  

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
8	12	5.3035e+00	9.3e+00	1.7e+00	3.7e-01	1.0e+00	1.1e+01

  

Variable	x	g	Status
1	1.0732e+00	1.7792e+00	Free
2	0.0000e+00	1.8627e+01	Upper Bound
3	4.4590e-01	-2.6928e+00	Free
4	1.2826e+00	8.7338e+00	Free

  

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
9	13	4.6913e+00	1.9e+01	1.6e+00	9.5e-02	6.6e-02	7.3e+01

  

Variable	x	g	Status
1	1.0000e+00	1.5613e+00	Lower Bound
2	0.0000e+00	1.7336e+01	Free
3	4.3666e-01	-2.5266e+00	Free
4	1.2222e+00	8.2939e+00	Free

  

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
10	15	3.9688e+00	8.3e+00	1.6e+00	8.3e-02	4.8e-03	3.9e+00

  

Variable	x	g	Status
1	1.0000e+00	-9.4589e-02	Lower Bound
2	-8.3009e-02	-1.0146e-01	Free
3	4.3672e-01	-8.4789e-01	Free
4	1.2215e+00	8.2818e+00	Free

  

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
-----	------	-----------	--------	--------	-------------------	------	--------

11	16	2.8304e+00	6.4e+00	1.5e+00	1.6e-01	1.0e+00	1.9e+01
Variable		x		g		Status	
1		1.0000e+00		3.1074e-01		Lower Bound	
2		-8.3930e-02		-3.6206e-02		Free	
3		4.2460e-01		1.0297e-01		Free	
4		1.0643e+00		6.4079e+00		Free	
Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
12	17	2.4344e+00	2.4e-01	1.5e+00	6.5e-02	1.1e-01	2.6e+00
Variable		x		g		Status	
1		1.0000e+00		3.0156e-01		Lower Bound	
2		-8.4922e-02		-3.4052e-02		Free	
3		4.1431e-01		2.4239e-01		Free	
4		1.0000e+00		5.8569e+00		Lower Bound	
Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
13	18	2.4339e+00	8.7e-02	1.5e+00	3.2e-03	1.0e+00	4.1e+00
Variable		x		g		Status	
1		1.0000e+00		2.9879e-01		Lower Bound	
2		-8.5060e-02		-7.5964e-05		Free	
3		4.1114e-01		8.7488e-02		Free	
4		1.0000e+00		5.8886e+00		Lower Bound	
Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
14	19	2.4338e+00	7.8e-04	1.5e+00	1.8e-03	1.0e+00	4.1e+00
Variable		x		g		Status	
1		1.0000e+00		2.9535e-01		Lower Bound	
2		-8.5233e-02		-2.7717e-04		Free	
3		4.0932e-01		7.2996e-04		Free	
4		1.0000e+00		5.9068e+00		Lower Bound	
Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
15	20	2.4338e+00	3.5e-06	1.5e+00	1.5e-05	1.0e+00	4.1e+00
Variable		x		g		Status	
1		1.0000e+00		2.9535e-01		Lower Bound	
2		-8.5233e-02		-2.5483e-06		Free	
3		4.0930e-01		2.3869e-06		Free	
4		1.0000e+00		5.9070e+00		Lower Bound	
Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
16	21	2.4338e+00	8.2e-09	1.5e+00	4.6e-08	1.0e+00	4.1e+00
Variable		x		g		Status	
1		1.0000e+00		2.9535e-01		Lower Bound	
2		-8.5233e-02		-8.1823e-09		Free	
3		4.0930e-01		1.0494e-10		Free	
4		1.0000e+00		5.9070e+00		Lower Bound	
Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
17	22	2.4338e+00	8.2e-09	1.5e+00	4.2e-11	1.0e+00	1.0e+00
Variable		x		g		Status	
1		1.0000e+00		2.9535e-01		Lower Bound	
2		-8.5233e-02		-8.1823e-09		Free	
3		4.0930e-01		1.0494e-10		Free	
4		1.0000e+00		5.9070e+00		Lower Bound	
Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
18	23	2.4338e+00	8.2e-09	1.5e+00	4.2e-11	1.0e+00	1.0e+00
Variable		x		g		Status	
1		1.0000e+00		2.9535e-01		Lower Bound	
2		-8.5233e-02		-8.1823e-09		Free	
3		4.0930e-01		1.0494e-10		Free	
4		1.0000e+00		5.9070e+00		Lower Bound	

Local search performed.

Final solution:

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
18	27	2.4338e+00	8.2e-09	1.5e+00	8.2e-09	1.0e+00	1.0e+00

Variable	x	g	Status
1	1.0000e+00	2.9535e-01	Lower Bound
2	-8.5233e-02	-8.1823e-09	Free
3	4.0930e-01	1.0494e-10	Free
4	1.0000e+00	5.9070e+00	Lower Bound

---