

# NAG Library Function Document

## nag\_opt\_bounds\_no\_deriv (e04jbc)

### 1 Purpose

nag\_opt\_bounds\_no\_deriv (e04jbc) 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.

No derivatives are required. nag\_opt\_bounds\_no\_deriv (e04jbc) 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_no_deriv (Integer n,
    void (*objfun)(Integer n, const 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

nag\_opt\_bounds\_no\_deriv (e04jbc) is applicable to problems of the form:

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

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. You must supply a starting point and a function **objfun** to calculate the value of  $F(x)$  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 finite-difference approximations to 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 estimated 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. Most iterations calculate  $g_z$  using forward differences, but central differences are used when they seem necessary.

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_no_deriv` (e04jbc) gives you 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 you specify that the problem is unconstrained, `nag_opt_bounds_no_deriv` (e04jbc) 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_no_deriv` (e04jbc) will act as an unconstrained minimization algorithm. When the problem is unconstrained, the function values used for estimating the first derivatives will always be required in sets of  $n$ . `nag_opt_bounds_no_deriv` (e04jbc) enables you to take advantage (via the argument **bound**) of the fact that such sets can often be evaluated in less computer time than  $n$  separate function evaluations would take in general.

## 4 References

Baker G A Jr and Graves–Morris P R (1981) Padé approximants, Part 1: Basic theory *encyclopaedia of Mathematics and its Applications* Addison–Wesley

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 (1973) Safeguarded steplength algorithms for optimization using descent methods *NPL Report NAC 37* National Physical Laboratory

Gill P E and Murray W (1976) Minimization subject to bounds on the variables *NPL Report NAC 72* National Physical Laboratory

## 5 Arguments

1: **n** – Integer *Input*

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

*Constraint:*  $n \geq 1$ .

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

**objfun** must evaluate the function  $F(x)$  at any  $x$ . If `nag_opt_bounds_no_deriv` (e04jbc) is called with **bound** = `Nag_NoBounds_One_Call`, **objfun** must also be able to provide the set of  $n$  function values used for estimating first derivatives. (However, if you do not wish to calculate  $F$  at a particular  $x$ , there is the option of setting an argument to cause `nag_opt_bounds_no_deriv` (e04jbc) to terminate immediately.)

The specification of **objfun** is:

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

1: **n** – Integer *Input*

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

2: **x[n]** – const double *Input*

*On entry:* the point  $x$  at which the value of  $F$  is required.

3: **objf** – double \* *Output*

*On exit:* if **comm**  $\rightarrow$  **flag** = 0 on entry, then **objfun** must set **objf** to the value of the objective function  $F$  at the current point given in **x**. If it is not possible to evaluate  $F$ ,

then **objfun** should assign a negative value to **comm** → **flag**; nag\_opt\_bounds\_no\_deriv (e04jbc) will then terminate.

4: **g[n]** – double *Input/Output*

*On entry:* if **comm** → **flag** = 3 then **g** contains a set of differencing intervals.

*On exit:* if **comm** → **flag** = 3 on entry, then **objfun** must reset  $\mathbf{g}[j-1]$  to  $F(x_c + \mathbf{g}[j-1] \times e_j)$ , for  $j = 1, 2, \dots, n$ , where  $x_c$  is the point given in **x** and  $e_j$  is the  $j$ th co-ordinate direction. If it is not possible to evaluate the elements of **g** then **objfun** should assign a negative value to **comm** → **flag**; nag\_opt\_bounds\_no\_deriv (e04jbc) will then terminate.

Thus, since the function values are required at  $n$  points which each differ from  $x_c$  only in one co-ordinate, it may be possible to calculate some terms once but use them in the calculation of more than one function value. (If **comm** → **flag** = 0 on entry, **objfun** must **not** change the elements of **g**.)

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 3. The value 0 indicates that a single function value is required. The value 3 (which will only occur if nag\_opt\_bounds\_no\_deriv (e04jbc) is called with **bound** = Nag\_NoBounds\_One\_Call) indicates that a set of  $n$  function values is required.

*On exit:* if **objfun** resets **comm** → **flag** to some negative number then nag\_opt\_bounds\_no\_deriv (e04jbc) will terminate immediately with the error indicator NE\_USER\_STOP. If **fail** is supplied to nag\_opt\_bounds\_no\_deriv (e04jbc), **fail.errnum** will be set to your setting of **comm** → **flag**.

**first** – Nag\_Boolean *Input*

*On entry:* will be set to Nag\_TRUE on the first call to **objfun** and Nag\_FALSE for all subsequent calls.

**nf** – Integer *Input*

*On entry:* the number of calculations of the objective function; this value will be equal to the number of calls made to **objfun**, including the current one, unless the argument **bound** = Nag\_NoBounds\_One\_Call.

**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\_no\_deriv (e04jbc) these pointers may be allocated memory and initialized with various quantities for use by **objfun** when called from nag\_opt\_bounds\_no\_deriv (e04jbc).

**Note:** **objfun** should be tested separately before being used in conjunction with nag\_opt\_bounds\_no\_deriv (e04jbc). The array **x** must **not** be changed by **objfun**.

3: **bound** – Nag\_BoundType *Input*

*On entry:* indicates whether the problem is unconstrained or bounded. If the problem is unconstrained, the value of **bound** can be used to indicate that you wish **objfun** to be called with

**comm** → **flag** set to 3 when a set of  $n$  function values is required for making difference estimates of derivatives. If there are bounds on the variables, **bound** can be used to indicate 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 you will be supplying all the  $l_j$  and  $u_j$  individually.

**bound** = Nag\_NoBounds

If the problem is unconstrained and you wish **objfun** to be called  $n$  times with **comm** → **flag** set to 0 when a set of function values is required for making difference estimates.

**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  $u_1 = u_2 = \dots = u_n$ .

**bound** = Nag\_NoBounds\_One\_Call

If the problem is unconstrained and you wish a single call to be made to **objfun** with **comm** → **flag** = 3 when a set of function values are required for making difference estimates.

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

4: **bl[n]** – double *Input/Output*

*On entry:* the lower bounds  $l_j$ .

If **bound** = Nag\_Bounds, you 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** = Nag\_BoundsEqual, you must set **bl**[0] to  $l_1$ ; nag\_opt\_bounds\_no\_deriv (e04jbc) will then set the remaining elements of **bl** equal to **bl**[0].

If **bound** = Nag\_NoBounds, Nag\_BoundsZero or Nag\_NoBounds\_One\_Call, **bl** will be initialized by nag\_opt\_bounds\_no\_deriv (e04jbc).

*On exit:* the lower bounds actually used by nag\_opt\_bounds\_no\_deriv (e04jbc), e.g., if **bound** = Nag\_BoundsZero, **bl**[0] = **bl**[1] =  $\dots$  = **bl**[ $n - 1$ ] = 0.0.

5: **bu[n]** – double *Input/Output*

*On entry:* the upper bounds  $u_j$ .

If **bound** = Nag\_Bounds, you 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** = Nag\_BoundsEqual, you must set **bu**[0] to  $u_1$ ; nag\_opt\_bounds\_no\_deriv (e04jbc) will then set the remaining elements of **bu** equal to **bu**[0].

If **bound** = Nag\_NoBounds, Nag\_BoundsZero or Nag\_NoBounds\_One\_Call, **bu** will be initialized by nag\_opt\_bounds\_no\_deriv (e04jbc).

*On exit:* the upper bounds actually used by nag\_opt\_bounds\_no\_deriv (e04jbc), e.g., if **bound** = Nag\_BoundsZero, **bu**[0] = **bu**[1] =  $\dots$  = **bu**[ $n - 1$ ] =  $10^{10}$ .

6: **x[n]** – double *Input/Output*

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

*On exit:* 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.

- 7: **objf** – double \* *Input/Output*  
*On entry:* if **options.init\_state** = Nag\_Init\_None or Nag\_Init\_H\_S, you need not initialize **objf**.  
 If **options.init\_state** = Nag\_Init\_All, **objf** must be set on entry to the value of  $F(x)$  at the initial point supplied in **x**.  
*On exit:* the function value at the final point given in **x**.
- 8: **g[n]** – double *Input/Output*  
*On entry:* if **options.init\_state** = Nag\_Init\_All, **g** must be set on entry to an approximation to the first derivative vector at the initial  $x$ . This could be calculated by central differences.  
 If **options.init\_state** = Nag\_Init\_None or Nag\_Init\_H\_S, **g** need not be set.  
*On exit:* a finite difference approximation to the first derivative vector. Note that the elements of **g** corresponding to free variables are updated every iteration, but the elements corresponding to fixed variables are only updated when it is necessary to test the Lagrange-multiplier estimates (see Section 3). So, in the printout from nag\_opt\_bounds\_no\_deriv (e04jbc) (see Section 5 and Section 10.3) and on exit from nag\_opt\_bounds\_no\_deriv (e04jbc), the elements of **g** corresponding to fixed variables may be out of date. The elements of **g** corresponding to free variables should normally be close to zero on exit from nag\_opt\_bounds\_no\_deriv (e04jbc).
- 9: **options** – Nag\_E04\_Opt \* *Input/Output*  
*On entry/exit:* a pointer to a structure of type Nag\_E04\_Opt whose members are optional arguments for nag\_opt\_bounds\_no\_deriv (e04jbc). These structure members offer the means of adjusting some of the argument 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 10. Some of the results returned in **options** can be used by nag\_opt\_bounds\_no\_deriv (e04jbc) to perform a ‘warm start’ if it is re-entered (see the member **options.init\_state** in Section 10.2).  
 If any of these optional arguments 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\_no\_deriv (e04jbc). However, if the optional arguments are not required the NAG defined null pointer, E04\_DEFAULT, can be used in the function call.
- 10: **comm** – Nag\_Comm \* *Input/Output*  
**Note:** **comm** is a NAG defined type (see Section 3.2.1.1 in the Essential Introduction).  
*On entry/exit:* structure containing pointers for communication with user-supplied functions; see the above description of **objfun** for details. If you do not need to make use of this communication feature the null pointer NAGCOMM\_NULL may be used in the call to nag\_opt\_bounds\_no\_deriv (e04jbc); **comm** will then be declared internally for use in calls to user-supplied functions.
- 11: **fail** – NagError \* *Input/Output*  
 The NAG error argument (see Section 3.6 in the Essential Introduction).

## 5.1 Description of Printed Output

Intermediate and final results are printed out by default. The level of printed output can be controlled with the structure member **options.print\_level** (see Section 10.2). The default, **options.print\_level** = 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\_no\_deriv (e04jbc).

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.

Itn	the iteration count, $k$ .
Nfun	the cumulative number of calls made to <b>objfun</b> .
Objective	the value of the objective function, $F(x^{(k)})$

Norm $g$	the Euclidean norm of the projected gradient vector, $\ g_z(x^{(k)})\ $ .
Norm $x$	the Euclidean norm of $x^{(k)}$ .
Norm( $x^{(k-1)} - x^{(k)}$ )	the Euclidean norm of $x^{(k-1)} - x^{(k)}$ .
Step	the step $\alpha^{(k)}$ taken along the computed search direction $p^{(k)}$ .
Cond $H$	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 estimate of the projected gradient vector, $g_z(x^*)$ .
Status	the final state of the variable with respect to its bound.

## 6 Error Indicators and Warnings

When one of `NE_USER_STOP`, `NE_INT_ARG_LT`, `NE_BOUND`, `NE_OPT_NOT_INIT`, `NE_BAD_PARAM`, `NE_2_REAL_ARG_LT`, `NE_INVALID_INT_RANGE_1`, `NE_INVALID_REAL_RANGE_EF`, `NE_INVALID_REAL_RANGE_FF`, `NE_NO_MEM`, `NE_FD_INT`, `NE_HESD` or `NE_ALLOC_FAIL` occurs, no values will have been assigned by `nag_opt_bounds_no_deriv` (e04jbc) to **objf** or to the elements of **g**, **options.hesl**, or **options.hesd**.

An exit of **fail.code** = `NW_TOO_MANY_ITER`, `NW_COND_MIN` and `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\_2\_REAL\_ARG\_LT

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

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### NE\_BAD\_PARAM

On entry, argument **bound** had an illegal value.

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

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

### NE\_BOUND

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

### NE\_CANCEL\_ERR

The overall relative cancellation error in the gradient estimate,  $g$ , or the expected search direction,  $p$ , is larger than 0.1. You should attempt to select another starting point.

### 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`.

**NE\_FD\_INT**

Finite difference interval for variable  $\langle value \rangle$  (array element `options.delta[ $\langle value \rangle$ ]`) is negative or so small that  $x_+$  interval =  $x$ .

**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 `options.hesd` to the smallest is too large.

**NE\_INT\_ARG\_LT**

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

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

Value  $\langle value \rangle$  given to `options.max_iter` is not valid. Correct range is `options.max_iter`  $\geq 0$ .

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

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

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

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

**NE\_NO\_MEM**

Option `options.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\_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`  $\rightarrow$  `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`  $\rightarrow$  `flag`.

**NE\_WRITE\_ERROR**

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

**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 estimated 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 the 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.

## 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.

## 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.

## 7 Accuracy

A successful exit (**fail.code** = NE\_NOERROR) is made from nag\_opt\_bounds\_no\_deriv (e04jbc) when (B1, B2 and B3) or B4 hold, and the local search (if used) confirms a minimum, where

$$B1 \equiv \alpha^{(k)} \times \|p^{(k)}\| < (\mathbf{options.optim\_tol} + \sqrt{\epsilon}) \times (1.0 + \|x^{(k)}\|)$$

$$B2 \equiv |F^{(k)} - F^{(k-1)}| < (\mathbf{options.optim\_tol}^2 + \epsilon) \times (1.0 + |F^{(k)}|)$$

$$B3 \equiv \|g_z^{(k)}\| < (\epsilon^{1/3} + \mathbf{options.optim\_tol}) \times (1.0 + |F^{(k)}|)$$

$$B4 \equiv \|g_z^{(k)}\| < 0.01 \times \sqrt{\epsilon}.$$

(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 **options.optim\_tol** is described in Section 10.)

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 **options.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 **options.hesd[b]**, let the smallest be **options.hesd[s]**, and define  $k = \mathbf{options.hesd[b]}/\mathbf{options.hesd[s]}$ . 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\_no\_deriv (e04jbc); in particular the final value of Cond H gives  $k$ .

Further suggestions about confirmation of a computed solution are given in the e04 Chapter Introduction.

## 8 Further Comments

### 8.1 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_no_deriv` (e04jbc) is roughly proportional to  $n_z^2$ . In addition, each iteration makes at least  $n_z + 1$  function evaluations. So, unless  $F(x)$  can be evaluated very quickly, the run time will be dominated by the time spent in **objfun**.

### 8.2 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 you 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_no_deriv` (e04jbc) will take less computer time.

### 8.3 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` = `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.hes1` 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_no_deriv` (e04jbc) 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 estimated first derivative vector.

## 9 Example

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  $(3, -1, 0, 1)^T$ .

The example program also shows the use of certain optional arguments. It shows option values being assigned directly within the program text and by reading values from a data file. The `options` structure is declared and initialized by `nag_opt_init` (e04xxc). Values are then assigned directly to `options.outfile` and `options.optim_tol` and three further options are read from the 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. You must **not** use the standard C function `free()` for this purpose.

## 9.1 Program Text

```

/* nag_opt_bounds_no_deriv (e04jbc) Example Program.
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

#include <nag.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagx02.h>
#include <nagx04.h>

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

int main(int argc, char *argv[])
{
    FILE          *fpout;
    char          *optionsfile = 0;
    char          *outfile = 0;
    Integer       exit_status = 0;
    Nag_Boolean   print;
    Integer       n;
    Nag_BoundType bound;
    Nag_E04_Opt   options;
    double        *bl = 0, *bu = 0, *g = 0, objf, *x = 0;
    NagError      fail;

    INIT_FAIL(fail);

    /* Check for command-line IO options */
    fpout = nag_example_file_io(argc, argv, "-results", NULL);
    (void) nag_example_file_io(argc, argv, "-options", &optionsfile);
    (void) nag_example_file_io(argc, argv, "-nag_write", &outfile);
    if (!outfile)
    {
        outfile = NAG_ALLOC(7, char);
        strcpy(outfile, "stdout");
    }

    fprintf(fpout, "nag_opt_bounds_no_deriv (e04jbc) Example Program Results\n");
    n = 4;
    if (n >= 1)
    {
        if (!(x = NAG_ALLOC(n, double)) ||
            !(g = NAG_ALLOC(n, double)) ||
            !(bl = NAG_ALLOC(n, double)) ||
            !(bu = NAG_ALLOC(n, double)))
        {
            fprintf(fpout, "Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
    else

```

```

    {
        fprintf(fpout, "Invalid n.\n");
        exit_status = 1;
        return exit_status;
    }
x[0] = 3.0;
x[1] = -1.0;
x[2] = 0.0;
x[3] = 1.0;

/* Initialise options structure */
/* nag_opt_init (e04xxc).
 * Initialization function for option setting
 */
nag_opt_init(&options);
strcpy(options.outfile, outfile);
/* nag_machine_precision (x02ajc).
 * The machine precision
 */
options.optim_tol = 100*sqrt(X02AJC); /* Specify accuracy of solution */

/* Read remaining option values from file */
print = Nag_TRUE;
/* nag_opt_read (e04xyc).
 * Read options from a text file
 */
if (strcmp(outfile, "stdout"))
    fclose(fpout);
nag_opt_read("e04jbc", optionsfile, &options, print, options.outfile, &fail);
if (strcmp(outfile, "stdout"))
    {
        fpout = fopen(outfile, "a");
    }
if (fail.code != NE_NOERROR)
    {
        fprintf(fpout, "Error from nag_opt_read (e04xyc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

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

/* Call optimization routine */
/* nag_opt_bounds_no_deriv (e04jbc), see above. */
if (strcmp(outfile, "stdout"))
    fclose(fpout);
nag_opt_bounds_no_deriv(n, objfun, bound, bl, bu, x, &objf,
                        g, &options, NAGCOMM_NULL, &fail);
if (strcmp(outfile, "stdout"))
    {
        fpout = fopen(outfile, "a");
    }
if (fail.code != NE_NOERROR)
    {
        fprintf(fpout,
                "Error/Warning from nag_opt_bounds_no_deriv (e04jbc).\n%s\n",
                fail.message);
        if (fail.code != NW_COND_MIN)
            exit_status = 1;
    }

```

```

    }

    /* Free Nag allocated memory */
    /* nag_opt_free (e04xzc).
    * Memory freeing function for use with option setting
    */
    nag_opt_free(&options, "all", &fail);
    if (fail.code != NE_NOERROR)
    {
        fprintf(fpout, "Error from nag_opt_free (e04xzc).\n%s\n", fail.message);
        exit_status = 2;
    }

END:
    if (fpout != stdout) fclose(fpout);
    if (x) NAG_FREE(x);
    if (g) NAG_FREE(g);
    if (bl) NAG_FREE(bl);
    if (bu) NAG_FREE(bu);
    if (optionsfile) NAG_FREE(optionsfile);
    if (outfile) NAG_FREE(outfile);

    return exit_status;
}

static void NAG_CALL objfun(Integer n, double x[], double *objf,
                           double g[], Nag_Comm *comm)
{
    /* Routine to evaluate objective function. */

    double a, b, c, d, x1, x2, x3, x4;

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

    /* Supply a single function value */
    a = x1 + 10.0*x2;
    b = x3 - x4;
    c = x2 - 2.0*x3, c *= c;
    d = x1 - x4, d *= d;
    *objf = a*a + 5.0*b*b + c*c + 10.0*d*d;
}
/* objfun */

```

## 9.2 Program Data

nag\_opt\_bounds\_no\_deriv (e04jbc) Example Program Optional Parameters

Following options for e04jbc are read by e04xyc.

begin e04jbc

```

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 */

```

end

## 9.3 Program Results

nag\_opt\_bounds\_no\_deriv (e04jbc) Example Program Results

Optional parameter setting for e04jbc.

-----  
Option file: e04jbce.opt

```

print_level set to Nag_Soln_Iter_Full
max_iter set to 40

```

step\_max set to 4.00e+00

Parameters to e04jbc

-----

Number of variables..... 4

```

optim_tol..... 1.05e-06    linesearch_tol..... 5.00e-01
step_max..... 4.00e+00    max_iter..... 40
init_state..... Nag_Init_None    local_search..... Nag_TRUE
print_level.... Nag_Soln_Iter_Full    machine_precision..... 1.11e-16
outfile..... stdout

```

Memory allocation:

```

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

```

Results from e04jbc:

-----

Iteration results:

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
0	0	2.1500e+02	1.4e+02	3.3e+00			3.7e+00

Variable	x	g	Status
1	3.0000e+00	3.0600e+02	Upper Bound
2	-1.0000e+00	-1.4400e+02	Free
3	0.0000e+00	-2.0000e+00	Free
4	1.0000e+00	-3.1000e+02	Lower Bound

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
1	6	1.6467e+02	3.2e+02	3.2e+00	7.0e-01	1.0e+00	2.1e+02

Variable	x	g	Status
1	3.0000e+00	3.1997e+02	Free
2	-3.0174e-01	-5.5533e-01	Free
3	3.5448e-02	-9.2316e+00	Free
4	1.0000e+00	-3.1035e+02	Lower Bound

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
2	9	8.7285e+00	4.2e+01	1.4e+00	2.0e+00	6.3e-03	3.6e+00

Variable	x	g	Status
1	1.0000e+00	3.1997e+02	Lower Bound
2	-3.0166e-01	-4.0542e+01	Free
3	3.6453e-02	-9.2151e+00	Free
4	1.0000e+00	-3.1035e+02	Lower Bound

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
3	13	2.9252e+00	7.9e+00	1.4e+00	3.3e-01	1.1e+00	5.0e+00

Variable	x	g	Status
1	1.0000e+00	3.1997e+02	Lower Bound
2	-6.7557e-02	5.6219e+00	Free
3	2.6655e-01	-5.6009e+00	Free
4	1.0000e+00	-3.1035e+02	Lower Bound

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
4	17	2.4562e+00	1.5e+00	1.5e+00	1.7e-01	1.7e+00	5.3e+00

Variable	x	g	Status
1	1.0000e+00	3.1997e+02	Lower Bound
2	-8.2099e-02	2.7257e-02	Free
3	4.3958e-01	1.5018e+00	Free
4	1.0000e+00	-3.1035e+02	Lower Bound

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
5	22	2.4356e+00	4.6e-01	1.5e+00	3.9e-02	1.0e+00	4.2e+00

Variable	x	g	Status
1	1.0000e+00	2.9810e-01	Lower Bound
2	-8.5095e-02	1.9441e-01	Free
3	4.0069e-01	-4.1995e-01	Free
4	1.0000e+00	5.9931e+00	Lower Bound

  

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
6	28	2.4338e+00	4.7e-03	1.5e+00	8.7e-03	1.1e+00	4.4e+00

Variable	x	g	Status
1	1.0000e+00	2.9504e-01	Lower Bound
2	-8.5248e-02	-4.0917e-03	Free
3	4.0935e-01	2.3736e-03	Free
4	1.0000e+00	5.9065e+00	Lower Bound

  

Itn	Nfun	Objective	Norm g	Norm x	Norm(x(k-1)-x(k))	Step	Cond H
7	33	2.4338e+00	9.3e-05	1.5e+00	4.6e-05	1.0e+00	4.4e+00

Variable	x	g	Status
1	1.0000e+00	2.9535e-01	Lower Bound
2	-8.5232e-02	8.5739e-05	Free
3	4.0930e-01	-3.6682e-05	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
8	40	2.4338e+00	6.3e-06	1.5e+00	7.0e-07	1.0e+00	4.4e+00

Variable	x	g	Status
1	1.0000e+00	2.9535e-01	Lower Bound
2	-8.5233e-02	-5.7290e-06	Free
3	4.0930e-01	-2.5522e-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
9	47	2.4338e+00	1.4e-07	1.5e+00	7.5e-08	1.0e+00	4.4e+00

Variable	x	g	Status
1	1.0000e+00	2.9535e-01	Lower Bound
2	-8.5233e-02	1.2807e-07	Free
3	4.0930e-01	4.9124e-08	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
9	64	2.4338e+00	1.4e-07	1.5e+00	0.0e+00	0.0e+00	4.4e+00

Variable	x	g	Status
1	1.0000e+00	2.9535e-01	Lower Bound
2	-8.5233e-02	1.2807e-07	Free
3	4.0930e-01	4.9124e-08	Free
4	1.0000e+00	5.9070e+00	Lower Bound

## 10 Optional Arguments

A number of optional input and output arguments to `nag_opt_bounds_no_deriv` (e04jbc) are available through the structure argument **options**, type `Nag_E04_Opt`. An 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 arguments you should use the NAG defined null pointer, `E04_DEFAULT`, in place of **options** when calling `nag_opt_bounds_no_deriv` (e04jbc); 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.

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 `nag_opt_read` (e04xyc).

### 10.1 Optional Argument Checklist and Default Values

For easy reference, the following list shows the members of **options** which are valid for `nag_opt_bounds_no_deriv` (e04jbc) 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_PrintType print_level	Nag_Soln_Iter
char outfile[80]	stdout
void (*print_fun)()	NULL
Nag_InitType init_state	Nag_Init_None
Integer max_iter	50n
double optim_tol	$10\sqrt{\epsilon}$
double linesearch_tol	0.5 (0.0 if n = 1)
double step_max	100000.0
double f_est	
Boolean local_search	Nag_TRUE
double *delta	size n
Integer *state	size n
double *hesl	size $\max(\mathbf{n}[\mathbf{n} - 1]/2, 1)$
double *hesd	size n
Integer iter	
Integer nf	

### 10.2 Description of the Optional Arguments

**list** – Nag\_Boolean Default = Nag\_TRUE

*On entry:* if **options.list** = Nag\_TRUE the argument settings in the call to `nag_opt_bounds_no_deriv` (e04jbc) will be printed.

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

*On entry:* the level of results printout produced by `nag_opt_bounds_no_deriv` (e04jbc). The following values are available:

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

Details of each level of results printout are described in Section 10.3.

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

**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.

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

*On entry:* printing function defined by you; the prototype of **options.print\_fun** is

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

See Section 10.3.1 below for further details.

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

*On entry:* **options.init\_state** specifies which of the arguments **objf**, **g**, **options.hesl**, **options.hesd** and **options.state** are actually being initialized. Such information will generally reduce the time taken by nag\_opt\_bounds\_no\_deriv (e04jbc).

**options.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\_no\_deriv (e04jbc) will use the unit matrix as the initial estimate of the Hessian matrix.)

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

The arguments **objf** and **g** must contain the value of  $F(x)$  and estimates of 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. The pointer **options.delta** must give the  $n$  finite-difference intervals. **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.)

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

No values are assumed to have been set in **objf** or **g**, but **options.hesl**, **options.hesd**, **options.state** and **options.delta** must have been set as for **options.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, Nag\_Init\_All or Nag\_Init\_H\_S.

**max\_iter** – Integer Default = 50n

*On entry:* the limit on the number of iterations allowed before termination.

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

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

*On entry:* 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

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

where  $\|y\| = \left(\sum_{j=1}^n y_j^2\right)^{1/2}$ . For example, if the elements of  $x_{\text{sol}}$  are not much larger than 1.0 in modulus and if **options.optim\_tol** is set to  $10^{-5}$ , then  $x_{\text{sol}}$  is usually accurate to about 5 decimal places. (For further details see Section 8.) If the problem is scaled roughly as described in Section 8 and  $\epsilon$  is the *machine precision*, then  $\sqrt{\epsilon}$  is probably the smallest reasonable choice for **options.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:*  $\epsilon \leq \mathbf{options.optim\_tol} < 1.0$ .

**linesearch\_tol** – double Default = 0.5. (If  $n = 1$ , default = 0.0.)

*On entry:* every iteration of nag\_opt\_bounds\_no\_deriv (e04jbc) involves a linear minimization (i.e., minimization of  $F(x + \alpha p)$  with respect to  $\alpha$ ). **options.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 **options.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_no_deriv` (e04jbc), 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  $F(x)$  can be evaluated unusually quickly (since it may be worth using extra function evaluations 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).

If  $\mathbf{n} = 1$ , the default for **options.linesearch\_tol** = 0.0 (if the problem is effectively one-dimensional then **options.linesearch\_tol** should be set to 0.0 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

*On entry:* an estimate of the Euclidean distance between the solution and the starting point supplied. (For maximum efficiency a slight overestimate is preferable.) `nag_opt_bounds_no_deriv` (e04jbc) will ensure that, for each iteration,

$$\left( \sum_{j=1}^n [x_j^{(k)} - x_j^{(k-1)}]^2 \right)^{1/2} \leq \mathbf{options.step\_max},$$

where  $k$  is the iteration number. Thus, if the problem has more than one solution, `nag_opt_bounds_no_deriv` (e04jbc) 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 **options.step\_max** can lead to inefficiency.

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

**f\_est** – double

*On entry:* 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 **options.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 is used, though this may be reduced to ensure that the bounds are not overstepped.

**local\_search** – Nag\_Boolean

Default = Nag\_TRUE

*On entry:* **options.local\_search** must specify whether or not you wish a ‘local search’ to be performed when a point is found which is thought to be a constrained minimum.

If **options.local\_search** = Nag\_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 **options.local\_search** = Nag\_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). **options.local\_search** could be set Nag\_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.

**delta** – double \*  $r$  Default memory = **n**

*On entry:* suitable step lengths for making difference approximations to the partial derivatives of  $F(x)$ .

If **options.delta** is not allocated memory and **options.init\_state** = Nag\_Init\_None then nag\_opt\_bounds\_no\_deriv (e04jbc) will allocate memory to **options.delta** and assign a suitable set of difference intervals.

If **options.delta** is allocated memory, i.e., **options.delta** is not NULL, and **options.init\_state** = Nag\_Init\_None then difference intervals are assumed to be supplied by **options.delta**.

When **options.init\_state**  $\neq$  Nag\_Init\_None then **options.delta** must hold the finite difference intervals; these may be the values output from a previous call to nag\_opt\_bounds\_no\_deriv (e04jbc).

If you wish to supply difference intervals then the following advice can be given. When the problem is scaled roughly as described in Section 8 and  $\epsilon$  is the *machine precision*, values in the range  $\sqrt{\epsilon}$  to  $\epsilon^{2/3}$  may be suitable. Otherwise, you must choose suitable settings, bearing in mind that, when forward differences are used, the approximation is

$$\frac{\partial F}{\partial x_j} = \frac{F(x + \mathbf{options.delta}[j] \times e_j) - F(x)}{\mathbf{options.delta}[j]}$$

where  $e_j$  is the  $j$ th co-ordinate direction, for  $j = 1, 2, \dots, n$ .

*On exit:* the **n** finite difference intervals used by nag\_opt\_bounds\_no\_deriv (e04jbc). If **options.delta** is NULL on entry and **options.init\_state** = Nag\_Init\_None then memory will have been automatically allocated to **options.delta** and suitable values assigned.

*Constraint:* **options.delta**[ $j$ ]  $\geq$  0.0,  $\mathbf{x}[j] + \mathbf{options.delta}[j] \neq \mathbf{x}[j]$ .

**state** – Integer \*  $i$  Default memory = **n**

*On entry:* **options.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\_no\_deriv (e04jbc).

If the option **options.init\_state** = Nag\_Init\_All or Nag\_Init\_H\_S has been chosen, **options.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\_no\_deriv (e04jbc) with **options.init\_state** = Nag\_Init\_None and the same value of **n**. If a previous call has not been made, you must allocate sufficient memory.

When **options.init\_state** = Nag\_Init\_All or Nag\_Init\_H\_S then **options.state** must specify information about which variables are currently on their bounds and which are free. If  $x_j$  is:

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

If **options.init\_state** = Nag\_Init\_None, **options.state** will be initialized by nag\_opt\_bounds\_no\_deriv (e04jbc).

If **options.init\_state** = Nag\_Init\_All or Nag\_Init\_H\_S, **options.state** must be initialized before nag\_opt\_bounds\_no\_deriv (e04jbc) is called.

*On exit:* **options.state** gives information as above about the final point given in **n**.

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

*On entry:* **options.hesl** and **options.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\_no\_deriv (e04jbc).

If **options.init\_state** = Nag\_Init\_All or **options.init\_state** = Nag\_Init\_H\_S has been set then **options.hesl** must point to a minimum of  $\max(\mathbf{n}[\mathbf{n} - 1]/2, 1)$  elements of memory.

**options.hesd** must point to at least  $\mathbf{n}$  elements of memory if **options.init\_state** = Nag\_Init\_All or Nag\_Init\_H\_S has been chosen.

The appropriate amount of memory will already be available for **options.hesl** and **options.hesd** if the calling program has used the **options** structure in a previous call to nag\_opt\_bounds\_no\_deriv (e04jbc) with **options.init\_state** = Nag\_Init\_None and the same value of  $\mathbf{n}$ . If a previous call has not been made, you must allocate sufficient memory.

**options.hesl** and **options.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 **options.state**, see above.) **options.hesl** holds the lower triangle of  $L$ , omitting the unit diagonal, stored by rows. **options.hesd** stores the diagonal elements of  $D$ . Thus if  $n_z$  elements of **options.state** are positive, the strict lower triangle of  $L$  will be held in the first  $n_z(n_z - 1)/2$  elements of **options.hesl** and the diagonal elements of  $D$  in the first  $n_z$  elements of **options.hesd**.

If **options.init\_state** = Nag\_Init\_None (the default), **options.hesl** and **options.hesd** will be initialized within nag\_opt\_bounds\_no\_deriv (e04jbc) to the factors of the unit matrix.

If you set **options.init\_state** = Nag\_Init\_All or Nag\_Init\_H\_S, **options.hesl** and **options.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 your setting of **options.state**.

*On exit:* **options.hesl** and **options.hesd** hold the factors  $L$  and  $D$  corresponding to the final point given in  $\mathbf{x}$ . The elements of **options.hesd** are useful for deciding whether to accept the result produced by nag\_opt\_bounds\_no\_deriv (e04jbc) (see Section 8).

**iter** – Integer  $i$

*On exit:* the number of iterations which have been performed in nag\_opt\_bounds\_no\_deriv (e04jbc).

**nf** – Integer  $i$

*On exit:* the number of times the residuals have been evaluated.

### 10.3 Description of Printed Output

The level of printed output can be controlled with the structure members **options.list** and **options.print\_level** (see Section 10.2). If **options.list** = Nag\_TRUE then the argument values to nag\_opt\_bounds\_no\_deriv (e04jbc) are listed, whereas the printout of results is governed by the value of **options.print\_level**. The default of **options.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 printout available from nag\_opt\_bounds\_no\_deriv (e04jbc).

When **options.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:

Itn	the iteration count, $k$ .
Nfun	the cumulative number of objective function evaluations.
Objective	the value of the objective function, $F(x^{(k)})$
Norm g	the Euclidean norm of the projected gradient vector, $\ g_z(x^{(k)})\ $ .
Norm x	the Euclidean norm of $x^{(k)}$ .
Norm(x(k-1)-x(k))	the Euclidean norm of $x^{(k-1)} - x^{(k)}$ .

Step the step  $\alpha^{(k)}$  taken along the computed search direction  $p^{(k)}$ .  
 Cond H 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:

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

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

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

If **options.print\_level** = Nag\_NoPrint then printout will be suppressed; you can print the final solution when nag\_opt\_bounds\_no\_deriv (e04jbc) returns to the calling program.

### 10.3.1 Output of results via a user-defined printing function

You may also specify your 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\_no\_deriv (e04jbc). 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 **options.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 10.2) in which case **options.print\_fun** is called with the results of the last iteration (**st** → **local\_search** = Nag\_FALSE) and then again when the local search has been completed (**st** → **local\_search** = Nag\_TRUE).)

If **comm** → **it\_prt** = Nag\_TRUE then the results on completion of an iteration of nag\_opt\_bounds\_no\_deriv (e04jbc) are contained in the members of **st**. If **comm** → **sol\_prt** = Nag\_TRUE then the final results from nag\_opt\_bounds\_no\_deriv (e04jbc), 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** = Nag\_TRUE; the final iteration count,  $k$ , if **comm** → **sol\_prt** = Nag\_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 \*

The estimated value of  $\frac{\partial F}{\partial x_j}$  at  $x^{(k)}$ , for  $j = 1, 2, \dots, n$ .

**gpj\_norm** – double

The Euclidean norm of the current estimate of the 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 3 for a description of the possible status values.

**local\_search** – Nag\_Boolean

Nag\_TRUE if a local search has been performed.

**nf** – Integer

The cumulative number of objective function evaluations.

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

**it\_prt** – Nag\_Boolean

Will be Nag\_TRUE when the print function is called with the results of the current iteration.

**sol\_prt** – Nag\_Boolean

Will be Nag\_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 either before entry to `nag_opt_bounds_no_deriv` (e04jbc) or during a call to **objfun** or **options.print\_fun**. The type Pointer will be `void *` with a C compiler that defines `void *` and `char *` otherwise.