NAG Library Routine Document

E04LBF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

E04LBF is a comprehensive modified 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 and second derivatives are required. The routine is intended for functions which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

2 Specification

```
SUBROUTINE E04LBF (N, FUNCT, H, MONIT, IPRINT, MAXCAL, ETA, XTOL,          &
                   STEPMX, IBOUND, BL, BU, X, HESL, LH, HESD, ISTATE, F,    &
                   G, IW, LIW, W, LW, IFAIL)
INTEGER            N, IPRINT, MAXCAL, IBOUND, LH, ISTATE(N), IW(LIW),      &
                   LIW, LW, IFAIL
REAL (KIND=nag_wp) ETA, XTOL, STEPMX, BL(N), BU(N), X(N), HESL(LH),        &
                   HESD(N), F, G(N), W(LW)
EXTERNAL           FUNCT, H, MONIT
```

3 Description

E04LBF is applicable to problems of the form:

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

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 $l_1 = l_2 = \cdots = l_n$ and $u_1 = u_2 = \cdots = u_n$. It is possible to specify that a particular x_j should be held constant. You must supply a starting point, a FUNCT to calculate the value of $F(x)$ and its first derivatives $\frac{\partial F}{\partial x_j}$ at any point x, and a H to calculate the second derivatives $\frac{\partial^2 F}{\partial x_i \partial x_j}$.

A typical iteration starts at the current point x where n_z (say) variables are free from both their bounds. The vector of first derivatives of $F(x)$ with respect to the free variables, g_z, and the matrix of second derivatives with respect to the free variables, H, are obtained. (These both have dimension n_z.)

The equations

$$(H + E)p_z = -g_z$$

are solved to give a search direction p_z. (The matrix E is chosen so that $H + E$ is positive definite.)

p_z is then expanded to an n-vector p by the insertion of appropriate zero elements; α is found such that $F(x + \alpha p)$ is approximately a minimum (subject to the fixed bounds) with respect to α, and x is replaced by $x + \alpha p$. (If a saddle point is found, a special search is carried out so as to move away from the saddle point.)

If any variable actually reaches a bound, 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 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 until the stronger criteria are satisfied. If at this point there are no negative or near-zero Lagrange multiplier estimates, the process is terminated.

If you specify that the problem is unconstrained, E04LBF sets the l_j to -10^6 and the u_j to 10^6. Thus, provided that the problem has been sensibly scaled, no bounds will be encountered during the minimization process and E04LBF will act as an unconstrained minimization algorithm.

4 References

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 (1974) Newton-type methods for unconstrained and linearly constrained optimization *Math. Programming* **7** 311–350

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

5 Parameters

1: N – INTEGER *Input*

On entry: the number n of independent variables.

Constraint: $N \geq 1$.

2: FUNCT – SUBROUTINE, supplied by the user. *External Procedure*

FUNCT must evaluate the function $F(x)$ and its first derivatives $\frac{\partial F}{\partial x_j}$ at any point x. (However, if you do not wish to calculate $F(x)$ or its first derivatives at a particular x, there is the option of setting a parameter to cause E04LBF to terminate immediately.)

The specification of FUNCT is:

```
SUBROUTINE FUNCT (IFLAG, N, XC, FC, GC, IW, LIW, W, LW)

INTEGER              IFLAG, N, IW(LIW), LIW, LW
REAL (KIND=nag_wp) XC(N), FC, GC(N), W(LW)
```

1: IFLAG – INTEGER *Input/Output*

On entry: will have been set to 2.

On exit: if it is not possible to evaluate $F(x)$ or its first derivatives at the point x given in XC (or if it is wished to stop the calculation for any other reason) you should reset IFLAG to some negative number and return control to E04LBF. E04LBF will then terminate immediately with IFAIL set to your setting of IFLAG.

2: N – INTEGER *Input*

On entry: the number n of variables.

3: XC(N) – REAL (KIND=nag_wp) array *Input*

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

4: FC – REAL (KIND=nag_wp) *Output*

On exit: unless IFLAG is reset, FUNCT must set FC to the value of the objective function F at the current point x.

5: GC(N) – REAL (KIND=nag_wp) array *Output*

On exit: unless IFLAG is reset, FUNCT must set GC(j) to the value of the first derivative $\frac{\partial F}{\partial x_j}$ at the point x, for $j = 1, 2, \ldots, n$.

6: IW(LIW) – INTEGER array *Workspace*
7: LIW – INTEGER *Input*
8: W(LW) – REAL (KIND=nag_wp) array *Workspace*
9: LW – INTEGER *Input*

FUNCT is called with the same parameters IW, LIW, W and LW as for E04LBF. They are present so that, when other library routines require the solution of a minimization subproblem, constants needed for the function evaluation can be passed through IW and W. Similarly, you **could** use elements $3, 4, \ldots,$ LIW of IW and elements from $\max(8, 7 \times \mathrm{N} + \mathrm{N} \times (\mathrm{N} - 1)/2) + 1$ onwards of W for passing quantities to FUNCT from the subroutine which calls E04LBF. However, because of the danger of mistakes in partitioning, it is recommended that you should pass information to FUNCT via COMMON global variables and not use IW or W at all. In any case FUNCT **must not change** the first 2 elements of IW or the first $\max(8, 7 \times \mathrm{N} + \mathrm{N} \times (\mathrm{N} - 1)/2)$ elements of W.

FUNCT must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which E04LBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

Note: FUNCT should be tested separately before being used in conjunction with E04LBF.

3: H – SUBROUTINE, supplied by the user. *External Procedure*

H must calculate the second derivatives of F at any point x. (As with FUNCT, there is the option of causing E04LBF to terminate immediately.)

The specification of H is:

```
SUBROUTINE H (IFLAG, N, XC, FHESL, LH, FHESD, IW, LIW, W, LW)
INTEGER              IFLAG, N, LH, IW(LIW), LIW, LW
REAL (KIND=nag_wp) XC(N), FHESL(LH), FHESD(N), W(LW)
```

1: IFLAG – INTEGER *Input/Output*

On entry: is set to a non-negative number.

On exit: if H resets IFLAG to some negative number, E04LBF will terminate immediately with IFAIL set to your setting of IFLAG.

2: N – INTEGER *Input*

On entry: the number n of variables.

3: XC(N) – REAL (KIND=nag_wp) array *Input*

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

4: FHESL(LH) – REAL (KIND=nag_wp) array *Output*

On exit: unless IFLAG is reset, H must place the strict lower triangle of the second derivative matrix of F (evaluated at the point x) in FHESL, stored by rows, i.e., set

$$\text{FHESL}((i-1)(i-2)/2+j) = \left.\frac{\partial^2 F}{\partial x_i \partial x_j}\right|_{\text{XC}}, \text{ for } i = 2,3,\ldots,n \text{ and } j = 1,2,\ldots,i-1.$$

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

5: LH – INTEGER *Input*

On entry: the length of the array FHESL.

6: FHESD(N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: the value of $\frac{\partial F}{\partial x_j}$ at the point x, for $j = 1,2,\ldots,n$.

These values may be useful in the evaluation of the second derivatives.

On exit: unless IFLAG is reset, H must place the diagonal elements of the second derivative matrix of F (evaluated at the point x) in FHESD, i.e., set $\text{FHESD}(j) = \left.\frac{\partial^2 F}{\partial x_j^2}\right|_{\text{XC}}$, $j = 1,2,\ldots,n$.

7: IW(LIW) – INTEGER array *Workspace*
8: LIW – INTEGER *Input*
9: W(LW) – REAL (KIND=nag_wp) array *Workspace*
10: LW – INTEGER *Input*

As in FUNCT, these parameters correspond to the parameters IW, LIW, W, LW of E04LBF. H **must not change** the first two elements of IW or the first $\max(8, 7\times \text{N} + \text{N}\times(\text{N}-1)/2)$ elements of W. Again, it is recommended that you should pass quantities to H via COMMON global variables and not use IW or W at all.

H must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which E04LBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

Note: H should be tested separately before being used in conjunction with E04LBF.

4: MONIT – SUBROUTINE, supplied by the user. *External Procedure*

If IPRINT ≥ 0, you must supply MONIT which is suitable for monitoring the minimization process. MONIT must not change the values of any of its parameters.

If IPRINT < 0, a MONIT with the correct parameter list should still be supplied, although it will not be called.

The specification of MONIT is:

```
SUBROUTINE MONIT (N, XC, FC, GC, ISTATE, GPJNRM, COND, POSDEF,       &
                  NITER, NF, IW, LIW, W, LW)

INTEGER              N, ISTATE(N), NITER, NF, IW(LIW), LIW, LW
REAL (KIND=nag_wp) XC(N), FC, GC(N), GPJNRM, COND, W(LW)
LOGICAL              POSDEF
```

1: N – INTEGER *Input*

On entry: the number n of variables.

2: XC(N) – REAL (KIND=nag_wp) array *Input*

On entry: the coordinates of the current point x.

3: FC – REAL (KIND=nag_wp) *Input*

On entry: the value of $F(x)$ at the current point x.

4: GC(N) – REAL (KIND=nag_wp) array *Input*

On entry: the value of $\frac{\partial F}{\partial x_j}$ at the current point x, for $j = 1, 2, \ldots, n$.

5: ISTATE(N) – INTEGER array *Input*

On entry: information about which variables are currently fixed on their bounds and which are free.

If ISTATE(j) is negative, x_j is currently:

– fixed on its upper bound if ISTATE(j) $= -1$;

– fixed on its lower bound if ISTATE(j) $= -2$;

– effectively a constant (i.e., $l_j = u_j$) if ISTATE(j) $= -3$.

If ISTATE is positive, its value gives the position of x_j in the sequence of free variables.

6: GPJNRM – REAL (KIND=nag_wp) *Input*

On entry: the Euclidean norm of the projected gradient vector g_z.

7: COND – REAL (KIND=nag_wp) *Input*

On entry: the ratio of the largest to the smallest elements of the diagonal factor D of the projected Hessian matrix (see specification of H). This quantity is usually a good estimate of the condition number of the projected Hessian matrix. (If no variables are currently free, COND is set to zero.)

8: POSDEF – LOGICAL *Input*

On entry: is set .TRUE. or .FALSE. according to whether the second derivative matrix for the current subspace, H, is positive definite or not.

9: NITER – INTEGER *Input*

On entry: the number of iterations (as outlined in Section 3) which have been performed by E04LBF so far.

10: NF – INTEGER *Input*

On entry: the number of times that FUNCT has been called so far. Thus NF is the number of function and gradient evaluations made so far.

11: IW(LIW) – INTEGER array *Workspace*
12: LIW – INTEGER *Input*
13: W(LW) – REAL (KIND=nag_wp) array *Workspace*
14: LW – INTEGER *Input*

As in FUNCT, and H, these parameters correspond to the parameters IW, LIW, W, LW of E04LBF. They are included in MONIT's parameter list primarily for when E04LBF is called by other library routines.

MONIT must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which E04LBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

You should normally print out FC, GPJNRM and COND so as to be able to compare the quantities mentioned in Section 7. It is normally helpful to examine XC, POSDEF and NF as well.

5: IPRINT – INTEGER *Input*

On entry: the frequency with which MONIT is to be called.

IPRINT > 0
: MONIT is called once every IPRINT iterations and just before exit from E04LBF.

IPRINT $= 0$
: MONIT is just called at the final point.

IPRINT < 0
: MONIT is not called at all.

IPRINT should normally be set to a small positive number.

Suggested value: IPRINT $= 1$.

6: MAXCAL – INTEGER *Input*

On entry: the maximum permitted number of evaluations of $F(x)$, i.e., the maximum permitted number of calls of FUNCT.

Suggested value: MAXCAL $= 50 \times$ N.

Constraint: MAXCAL ≥ 1.

7: ETA – REAL (KIND=nag_wp) *Input*

On entry: every iteration of E04LBF involves a linear minimization (i.e., minimization of $F(x + \alpha p)$ with respect to α). ETA specifies how accurately these linear minimizations are to be performed. The minimum with respect to α will be located more accurately for small values of ETA (say, 0.01) than for large values (say, 0.9).

Although accurate linear minimizations will generally reduce the number of iterations of E04LBF, this usually results in an increase in the number of function and gradient evaluations required for each iteration. On balance, it is usually more efficient to perform a low accuracy linear minimization.

Suggested value: **ETA $= 0.9$ is usually a good choice** although a smaller value may be warranted if the matrix of second derivatives is expensive to compute compared with the function and first derivatives.

If N $= 1$, ETA should be set to 0.0 (also when the problem is effectively one-dimensional even though $n > 1$; i.e., if for all except one of the variables the lower and upper bounds are equal).

Constraint: $0.0 \leq$ ETA < 1.0.

8: XTOL – REAL (KIND=nag_wp) *Input*

On entry: the accuracy in x to which the solution is required.

If x_{true} is the true value of x at the minimum, then x_{sol}, the estimated position before a normal exit, is such that $\|x_{\text{sol}} - x_{\text{true}}\| < \text{XTOL} \times (1.0 + \|x_{\text{true}}\|)$, where $\|y\| = \sqrt{\sum_{j=1}^{n} y_j^2}$. For example, if the elements of x_{sol} are not much larger than 1.0 in modulus, and if XTOL is set to 10^{-5} then x_{sol} is usually accurate to about five decimal places. (For further details see Section 7.)

If the problem is scaled roughly as described in Section 9 and ϵ is the ***machine precision***, then $\sqrt{\epsilon}$ is probably the smallest reasonable choice for XTOL. (This is because, normally, to machine accuracy, $F\left(x + \sqrt{\epsilon}, e_j\right) = F(x)$ where e_j is any column of the identity matrix.)

If you set XTOL to 0.0 (or any positive value less than ϵ), E04LBF will use $10.0 \times \sqrt{\epsilon}$ instead of XTOL.

Suggested value: XTOL $= 0.0$.

Constraint: XTOL ≥ 0.0.

9: STEPMX – REAL (KIND=nag_wp) *Input*

On entry: an estimate of the Euclidean distance between the solution and the starting point supplied by you. (For maximum efficiency a slight overestimate is preferable.)

E04LBF will ensure that, for each iteration,

$$\sqrt{\sum_{j=1}^{n}\left[x_j^{(k)} - x_j^{(k-1)}\right]^2} \leq \text{STEPMX}$$

where k is the iteration number. Thus, if the problem has more than one solution, E04LBF is most likely to find the one nearest to 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 STEPMX can lead to inefficiency.

Suggested value: STEPMX $= 100000.0$.

Constraint: STEPMX $\geq$ XTOL.

10: IBOUND – INTEGER *Input*

On entry: specifies whether the problem is unconstrained or bounded. If there are bounds on the variables, IBOUND can be used to indicate whether the facility for dealing with bounds of special forms is to be used. It must be set to one of the following values:

IBOUND $= 0$
If the variables are bounded and you are supplying all the l_j and u_j individually.

IBOUND $= 1$
If the problem is unconstrained.

IBOUND $= 2$
If the variables are bounded, but all the bounds are of the form $0 \leq x_j$.

IBOUND $= 3$
If all the variables are bounded, and $l_1 = l_2 = \cdots = l_n$ and $u_1 = u_2 = \cdots = u_n$.

IBOUND $= 4$
If the problem is unconstrained. (The IBOUND $= 4$ option is provided purely for consistency with other routines. In E04LBF it produces the same effect as IBOUND $= 1$.)

Constraint: $0 \leq$ IBOUND ≤ 4.

11: BL(N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: the fixed lower bounds l_j.

If IBOUND is set to 0, you must set BL(j) to l_j, for $j = 1, 2, \ldots, n$. (If a lower bound is not specified for any x_j, the corresponding BL(j) should be set to a large negative number, e.g., -10^6.)

If IBOUND is set to 3, you must set BL(1) to l_1; E04LBF will then set the remaining elements of BL equal to BL(1).

If IBOUND is set to 1, 2 or 4, BL will be initialized by E04LBF.

On exit: the lower bounds actually used by E04LBF, e.g., if IBOUND $= 2$, BL$(1) =$ BL$(2) = \cdots =$ BL$(n) = 0.0$.

12: BU(N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: the fixed upper bounds u_j.

If IBOUND is set to 0, you must set BU(j) to u_j, for $j = 1, 2, \ldots, n$. (If an upper bound is not specified for any variable, the corresponding BU(j) should be set to a large positive number, e.g., 10^6.)

If IBOUND is set to 3, you must set BU(1) to u_1; E04LBF will then set the remaining elements of BU equal to BU(1).

If IBOUND is set to 1, 2 or 4, BU will then be initialized by E04LBF.

On exit: the upper bounds actually used by E04LBF, e.g., if IBOUND $= 2$, BU$(1) =$ BU$(2) = \cdots =$ BU(N) $= 10^6$.

13: X(N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: X(j) must be set to a guess at the jth component of the position of the minimum, for $j = 1, 2, \ldots, n$.

On exit: the final point $x^{(k)}$. Thus, if IFAIL $= 0$ on exit, X(j) is the jth component of the estimated position of the minimum.

14: HESL(LH) – REAL (KIND=nag_wp) array *Output*

On exit: during the determination of a direction p_z (see Section 3), $H + E$ is decomposed into the product LDL^{T}, where L is a unit lower triangular matrix and D is a diagonal matrix. (The matrices H, E, L and D are all of dimension n_z, where n_z is the number of variables free from their bounds. H consists of those rows and columns of the full estimated second derivative matrix which relate to free variables. E is chosen so that $H + E$ is positive definite.)

HESL and HESD are used to store the factors L and D. The elements of the strict lower triangle of L are stored row by row in the first $n_z(n_z - 1)/2$ positions of HESL. The diagonal elements of D are stored in the first n_z positions of HESD. In the last factorization before a normal exit, the matrix E will be zero, so that HESL and HESD will contain, on exit, the factors of the final estimated second derivative matrix H. The elements of HESD are useful for deciding whether to accept the results produced by E04LBF (see Section 7).

15: LH – INTEGER *Input*

On entry: the dimension of the array HESL as declared in the (sub)program from which E04LBF is called.

Constraint: LH $\geq \max(\mathrm{N} \times (\mathrm{N} - 1)/2, 1)$.

16: HESD(N) – REAL (KIND=nag_wp) array *Output*

On exit: during the determination of a direction p_z (see Section 3), $H + E$ is decomposed into the product LDL^{T}, where L is a unit lower triangular matrix and D is a diagonal matrix. (The matrices H, E, L and D are all of dimension n_z, where n_z is the number of variables free from their bounds. H consists of those rows and columns of the full second derivative matrix which relate to free variables. E is chosen so that $H + E$ is positive definite.)

HESL and HESD are used to store the factors L and D. The elements of the strict lower triangle of L are stored row by row in the first $n_z(n_z - 1)/2$ positions of HESL. The diagonal elements of D are stored in the first n_z positions of HESD.

In the last factorization before a normal exit, the matrix E will be zero, so that HESL and HESD will contain, on exit, the factors of the final second derivative matrix H. The elements of HESD are useful for deciding whether to accept the result produced by E04LBF (see Section 7).

17: ISTATE(N) – INTEGER array *Output*

On exit: information about which variables are currently on their bounds and which are free. If ISTATE(j) is:

– equal to -1, x_j is fixed on its upper bound;

– equal to -2, x_j is fixed on its lower bound;

– equal to -3, x_j is effectively a constant (i.e., $l_j = u_j$);

– positive, ISTATE(j) gives the position of x_j in the sequence of free variables.

18: F – REAL (KIND=nag_wp) *Output*

On exit: the function value at the final point given in X.

19: G(N) – REAL (KIND=nag_wp) array *Output*

On exit: the first derivative vector corresponding to the final point given in X. The components of G corresponding to free variables should normally be close to zero.

20: IW(LIW) – INTEGER array *Communication Array*
21: LIW – INTEGER *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which E04LBF is called.

Constraint: LIW ≥ 2.

22: W(LW) – REAL (KIND=nag_wp) array *Communication Array*
23: LW – INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which E04LBF is called.

Constraint: LW $\geq \max(7 \times \mathrm{N} + \mathrm{N} \times (\mathrm{N} - 1)/2, 8)$.

24: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, because for this routine the values of the output parameters may be useful even if IFAIL $\neq 0$ on exit, the recommended value is -1. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Note: E04LBF may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the routine:

IFAIL < 0

A negative value of IFAIL indicates an exit from E04LBF because you have set IFLAG negative in FUNCT or H. The value of IFAIL will be the same as your setting of IFLAG.

IFAIL $= 1$

On entry, N < 1,
or MAXCAL < 1,

or $\text{ETA} < 0.0$,
or $\text{ETA} \geq 1.0$,
or $\text{XTOL} < 0.0$,
or $\text{STEPMX} < \text{XTOL}$,
or $\text{IBOUND} < 0$,
or $\text{IBOUND} > 4$,
or $\text{BL}(j) > \text{BU}(j)$ for some j if $\text{IBOUND} = 0$,
or $\text{BL}(1) > \text{BU}(1)$ if $\text{IBOUND} = 3$,
or $\text{LH} < \max(1, \text{N} \times (\text{N} - 1)/2)$,
or $\text{LIW} < 2$,
or $\text{LW} < \max(8, 7 \times \text{N} + \text{N} \times (\text{N} - 1)/2)$.

(Note that if you have set XTOL to 0.0, E04LBF uses the default value and continues without failing.) When this exit occurs no values will have been assigned to F or to the elements of HESL, HESD or G.

IFAIL $= 2$

There have been MAXCAL function evaluations. If steady reductions in $F(x)$ were monitored up to the point where this exit occurred, then the exit probably occurred simply because MAXCAL 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.

IFAIL $= 3$

The conditions for a minimum have not all been met, 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. Such a situation can arise, for instance, if XTOL has been set so small that rounding errors in the evaluation of $F(x)$ or its derivatives make it impossible to satisfy the convergence conditions.

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

IFAIL $= 4$

Not used. (This is done to make the significance of IFAIL $= 5$ similar for E04KDF and E04LBF.)

IFAIL $= 5$

All the Lagrange multiplier estimates which are not indisputably positive lie relatively close to zero, but it is impossible either to continue minimizing on the current subspace or to find a feasible lower point by releasing and perturbing any of the fixed variables. You should investigate as for IFAIL $= 3$.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.8 in the Essential Introduction for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.7 in the Essential Introduction for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

The values IFAIL $= 2$, 3 or 5 may also be caused by mistakes in user-supplied subroutines FUNCT or H, 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.

7 Accuracy

A successful exit (IFAIL $= 0$) is made from E04LBF when $H^{(k)}$ is positive definite and when (B1, B2 and B3) or B4 hold, where

$$\begin{array}{lcl} \mathrm{B1} & \equiv & \alpha^{(k)} \times \left\|p^{(k)}\right\| < (\mathrm{XTOL} + \sqrt{\epsilon}) \times \left(1.0 + \left\|x^{(k)}\right\|\right) \\ \mathrm{B2} & \equiv & \left|F^{(k)} - F^{(k-1)}\right| < \left(\mathrm{XTOL}^2 + \epsilon\right) \times \left(1.0 + \left|F^{(k)}\right|\right) \\ \mathrm{B3} & \equiv & \left\|g_z^{(k)}\right\| < \left(\epsilon^{1/3} + \mathrm{XTOL}\right) \times \left(1.0 + \left|F^{(k)}\right|\right) \\ \mathrm{B4} & \equiv & \left\|g_z^{(k)}\right\| < 0.01 \times \sqrt{\epsilon}. \end{array}$$

(Quantities with superscript k are the values at the kth iteration of the quantities mentioned in Section 3. ϵ is the ***machine precision*** and $\|.\|$ denotes the Euclidean norm.)

If IFAIL $= 0$, then the vector in X on exit, x_{sol}, is almost certainly an estimate of the position of the minimum, x_{true}, to the accuracy specified by XTOL.

If IFAIL $= 3$ or 5, x_{sol} may still be a good estimate of x_{true}, but the following checks should be made. Let the largest of the first n_z elements of HESD be HESD(b), let the smallest be HESD(s), and define $k = \mathrm{HESD}(b)/\mathrm{HESD}(s)$. The scalar k is usually a good estimate of the condition number of the projected Hessian matrix at x_{sol}. If

(i) the sequence $\left\{F\left(x^{(k)}\right)\right\}$ converges to $F(x_{\mathrm{sol}})$ at a superlinear or fast linear rate,

(ii) $\|g_z(x_{\mathrm{sol}})\|^2 < 10.0 \times \epsilon$, and

(iii) $k < 1.0/\|g_z(x_{\mathrm{sol}})\|$,

then it is almost certain that x_{sol} is a close approximation to the position of a minimum. When (ii) is true, then usually $F(x_{\mathrm{sol}})$ is a close approximation to $F(x_{\mathrm{true}})$. The quantities needed for these checks are all available via MONIT; in particular the value of COND in the last call of MONIT before exit gives k.

Further suggestions about confirmation of a computed solution are given in the E04 Chapter Introduction.

8 Parallelism and Performance

Not applicable.

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 E04LBF is $\frac{n_z^3}{6} + O\left(n_z^2\right)$. In addition, each iteration makes one call of H and at least one call of FUNCT. So, unless $F(x)$ and its derivatives can be evaluated very quickly, the run time will be dominated by the time spent in FUNCT and H.

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 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 E04LBF will take less computer time.

9.3 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) HESL and HESD will be factors of the full second derivative matrix with elements stored in the natural order,

(c) the elements of g should all be close to zero at the final point,

(d) the values of the ISTATE(j) given by MONIT and on exit from E04LBF are unlikely to be of interest (unless they are negative, which would indicate that the modulus of one of the x_j has reached 10^6 for some reason),

(e) MONIT's parameter GPJNRM simply gives the norm of the first derivative vector.

So the following routine (in which partitions of extended workspace arrays are used as BL, BU and ISTATE) could be used for unconstrained problems:

```
      SUBROUTINE UNCLBF(N,FUNCT,H,MONIT,IPRINT,MAXCAL,ETA,XTOL,          &
                        STEPMX,X,HESL,LH,HESD,F,G,IWORK,LIWORK,WORK,     &
                        LWORK,IFAIL)

!     A ROUTINE TO APPLY E04LBF TO UNCONSTRAINED PROBLEMS.

!     THE REAL ARRAY WORK MUST BE OF DIMENSION AT LEAST
!     (9*N + MAX(1, N*(N-1)/2)).  ITS FIRST 7*N + MAX(1, N*(N-1)/2)
!     ELEMENTS WILL BE USED BY E04LBF AS THE ARRAY W.  ITS LAST
!     2*N ELEMENTS WILL BE USED AS THE ARRAYS BL AND BU.

!     THE INTEGER ARRAY IWORK MUST BE OF DIMENSION AT LEAST (N+2)
!     ITS FIRST 2 ELEMENTS WILL BE USED BY E04LBF AS THE ARRAY IW.
!     ITS LAST N ELEMENTS WILL BE USED AS THE ARRAY ISTATE.

!     LIWORK AND LWORK MUST BE SET TO THE ACTUAL LENGTHS OF IWORK
!     AND WORK RESPECTIVELY, AS DECLARED IN THE CALLING SEGMENT.

!     OTHER PARAMETERS ARE AS FOR E04LBF.

!     .. Parameters ..
      INTEGER NOUT
      PARAMETER (NOUT=6)
!     .. Scalar Arguments ..
      REAL (KIND=nag_wp) ETA, F, STEPMX, XTOL
      INTEGER            IFAIL, IPRINT, LH, LIWORK, LWORK, MAXCAL, N
!     .. Array Arguments ..
      REAL (KIND=nag_wp) G(N), HESD(N), HESL(LH), WORK(LWORK), X(N)
      INTEGER            IWORK(LIWORK)
!     .. Subroutine Arguments ..
      EXTERNAL FUNCT, H, MONIT
!     .. Local Scalars ..
      INTEGER IBOUND, J, JBL, JBU, NH
      LOGICAL TOOBIG
!     .. External Subroutines ..
      EXTERNAL E04LBF
!     .. Executable Statements ..
!     CHECK THAT SUFFICIENT WORKSPACE HAS BEEN SUPPLIED
      NH = N*(N-1)/2
      IF (NH.EQ.0) NH = 1
```

```
         IF (LWORK.LT.9*N+NH .OR.  LIWORK.LT.N+2) THEN
         WRITE (NOUT,FMT=99999)
         STOP
         END IF
!        JBL AND JBU SPECIFY THE PARTS OF WORK USED AS BL AND BU
         JBL = 7*N + NH + 1
         JBU = JBL + N
!        SPECIFY THAT THE PROBLEM IS UNCONSTRAINED
         IBOUND = 4
         CALL E04LBF(N,FUNCT,H,MONIT,IPRINT,MAXCAL,ETA,XTOL,STEPMX,         &
                     IBOUND,WORK(JBL),WORK(JBU),X,HESL,LH,HESD,IWORK(3),    &
           F,G,IWORK,LIWORK,WORK,LWORK,IFAIL)
!        CHECK THE PART OF IWORK WHICH WAS USED AS ISTATE IN CASE
!        THE MODULUS OF SOME X(J) HAS REACHED E+6
         TOOBIG = .FALSE.
         DO 20 J = 1, N
            IF (IWORK(2+J).LT.0) TOOBIG = .TRUE.
      20 CONTINUE
         IF ( .NOT.  TOOBIG) RETURN
         WRITE (NOUT,FMT=99998)
         STOP

   99999 FORMAT (' ***** INSUFFICIENT WORKSPACE HAS BEEN SUPPLIED *****')
   99998 FORMAT (' ***** A VARIABLE HAS REACHED E+6 IN MODULUS - NO UNCON', &
                 'STRAINED MINIMUM HAS BEEN FOUND *****')
         END
```

10 Example

A program to minimize

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

subject to the bounds

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

starting from the initial guess $(3, -1, 0, 1)$. Before calling E04LBF, the program calls E04HCF and E04HDF to check the derivatives calculated by user-supplied subroutines FUNCT and H.

10.1 Program Text

```
!   E04LBF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.
    Module e04lbfe_mod

!     E04LBF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                               :: funct, h, monit
!     .. Parameters ..
      Integer, Parameter, Public           :: liw = 2, n = 4, nout = 6
      Integer, Parameter, Public           :: lh = n*(n-1)/2
      Integer, Parameter, Public           :: lw = 7*n + n*(n-1)/2
    Contains
      Subroutine funct(iflag,n,xc,fc,gc,iw,liw,w,lw)
!       Routine to evaluate objective function and its 1st derivatives.

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: fc
        Integer, Intent (Inout)              :: iflag
```

```
        Integer, Intent (In)                 :: liw, lw, n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: gc(n)
        Real (Kind=nag_wp), Intent (Inout)   :: w(lw)
        Real (Kind=nag_wp), Intent (In)      :: xc(n)
        Integer, Intent (Inout)              :: iw(liw)
!       .. Executable Statements ..
        fc = (xc(1)+10.0_nag_wp*xc(2))**2 + 5.0_nag_wp*(xc(3)-xc(4))**2 + &
          (xc(2)-2.0_nag_wp*xc(3))**4 + 10.0_nag_wp*(xc(1)-xc(4))**4
        gc(1) = 2.0_nag_wp*(xc(1)+10.0_nag_wp*xc(2)) + &
          40.0_nag_wp*(xc(1)-xc(4))**3
        gc(2) = 20.0_nag_wp*(xc(1)+10.0_nag_wp*xc(2)) + &
          4.0_nag_wp*(xc(2)-2.0_nag_wp*xc(3))**3
        gc(3) = 10.0_nag_wp*(xc(3)-xc(4)) - 8.0_nag_wp*(xc(2)-2.0_nag_wp*xc(3) &
          )**3
        gc(4) = 10.0_nag_wp*(xc(4)-xc(3)) - 40.0_nag_wp*(xc(1)-xc(4))**3

        Return

      End Subroutine funct
      Subroutine h(iflag,n,xc,fhesl,lh,fhesd,iw,liw,w,lw)
!       Routine to evaluate 2nd derivatives

!       .. Scalar Arguments ..
        Integer, Intent (Inout)              :: iflag
        Integer, Intent (In)                 :: lh, liw, lw, n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)   :: fhesd(n), w(lw)
        Real (Kind=nag_wp), Intent (Out)     :: fhesl(lh)
        Real (Kind=nag_wp), Intent (In)      :: xc(n)
        Integer, Intent (Inout)              :: iw(liw)
!       .. Executable Statements ..
        fhesd(1) = 2.0_nag_wp + 120.0_nag_wp*(xc(1)-xc(4))**2
        fhesd(2) = 200.0_nag_wp + 12.0_nag_wp*(xc(2)-2.0_nag_wp*xc(3))**2
        fhesd(3) = 10.0_nag_wp + 48.0_nag_wp*(xc(2)-2.0_nag_wp*xc(3))**2
        fhesd(4) = 10.0_nag_wp + 120.0_nag_wp*(xc(1)-xc(4))**2
        fhesl(1) = 20.0_nag_wp
        fhesl(2) = 0.0_nag_wp
        fhesl(3) = -24.0_nag_wp*(xc(2)-2.0_nag_wp*xc(3))**2
        fhesl(4) = -120.0_nag_wp*(xc(1)-xc(4))**2
        fhesl(5) = 0.0_nag_wp
        fhesl(6) = -10.0_nag_wp

        Return

      End Subroutine h
      Subroutine monit(n,xc,fc,gc,istate,gpjnrm,cond,posdef,niter,nf,iw,liw,w, &
        lw)
!       Monitoring routine

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: cond, fc, gpjnrm
        Integer, Intent (In)                 :: liw, lw, n, nf, niter
        Logical, Intent (In)                 :: posdef
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: gc(n), xc(n)
        Real (Kind=nag_wp), Intent (Inout)   :: w(lw)
        Integer, Intent (In)                 :: istate(n)
        Integer, Intent (Inout)              :: iw(liw)
!       .. Local Scalars ..
        Integer                              :: isj, j
!       .. Executable Statements ..
        Write (nout,*)
        Write (nout,*) ' Itn     Fn evals              Fn value &
          &          Norm of proj gradient'
        Write (nout,99999) niter, nf, fc, gpjnrm
        Write (nout,*)
        Write (nout,*) &
          ' J          X(J)                G(J)           Status'

        Do j = 1, n
```

```
          isj = istate(j)

          Select Case (isj)
          Case (1:)
            Write (nout,99998) j, xc(j), gc(j), '    Free'
          Case (-1)
            Write (nout,99998) j, xc(j), gc(j), '    Upper Bound'
          Case (-2)
            Write (nout,99998) j, xc(j), gc(j), '    Lower Bound'
          Case (-3)
            Write (nout,99998) j, xc(j), gc(j), '    Constant'
          End Select

        End Do

        If (cond/=0.0_nag_wp) Then

          If (cond>1.0E6_nag_wp) Then
            Write (nout,*)
            Write (nout,*) 'Estimated condition number of projected &
              &Hessian is more than 1.0E+6'
          Else
            Write (nout,*)
            Write (nout,99997) &
              'Estimated condition number of projected Hessian = ', cond
          End If

          If (.Not. posdef) Then
            Write (nout,*)
            Write (nout,*) 'Projected Hessian matrix is not positive definite'
          End If

        End If

        Return

99999   Format (1X,I3,6X,I5,2(6X,1P,E20.4))
99998   Format (1X,I2,1X,1P,2E20.4,A)
99997   Format (1X,A,1P,E10.2)
      End Subroutine monit
    End Module e04lbfe_mod
    Program e04lbfe

!     E04LBF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: e04hcf, e04hdf, e04lbf, nag_wp
      Use e04lbfe_mod, Only: funct, h, lh, liw, lw, monit, n, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                   :: eta, f, stepmx, xtol
      Integer                              :: ibound, ifail, iprint, maxcal, nz
!     .. Local Arrays ..
      Real (Kind=nag_wp)                   :: bl(n), bu(n), g(n), hesd(n),     &
                                              hesl(lh), w(lw), x(n)
      Integer                              :: istate(n), iw(liw)
!     .. Intrinsic Procedures ..
      Intrinsic                            :: count
!     .. Executable Statements ..
      Write (nout,*) 'E04LBF Example Program Results'
      Flush (nout)

!     Set up an arbitrary point at which to check the derivatives

      x(1:n) = (/1.46_nag_wp,-0.82_nag_wp,0.57_nag_wp,1.21_nag_wp/)

!     Check the 1st derivatives

      ifail = 0
      Call e04hcf(n,funct,x,f,g,iw,liw,w,lw,ifail)
```

```
!     Check the 2nd derivatives

      ifail = 0
      Call e04hdf(n,funct,h,x,g,hesl,lh,hesd,iw,liw,w,lw,ifail)

!     Continue setting parameters for E04LBF

!     Set IPRINT to 1 to obtain output from MONIT at each iteration
      iprint = -1

      maxcal = 50*n
      eta = 0.9_nag_wp

!     Set XTOL to zero so that E04LBF will use the default tolerance

      xtol = 0.0_nag_wp

!     We estimate that the minimum will be within 4 units of the
!     starting point

      stepmx = 4.0_nag_wp

      ibound = 0

!     X(3) is unconstrained, so we set BL(3) to a large negative
!     number and BU(3) to a large positive number.

      bl(1:n) = (/1.0_nag_wp,-2.0_nag_wp,-1.0E6_nag_wp,1.0_nag_wp/)
      bu(1:n) = (/3.0_nag_wp,0.0_nag_wp,1.0E6_nag_wp,3.0_nag_wp/)

!     Set up starting point

      x(1:n) = (/3.0_nag_wp,-1.0_nag_wp,0.0_nag_wp,1.0_nag_wp/)

      ifail = -1
      Call e04lbf(n,funct,h,monit,iprint,maxcal,eta,xtol,stepmx,ibound,bl,bu, &
        x,hesl,lh,hesd,istate,f,g,iw,liw,w,lw,ifail)

      Select Case (ifail)
      Case (0,2:)
        Write (nout,*)
        Write (nout,99999) 'Function value on exit is ', f
        Write (nout,99998) 'at the point', x(1:n)
        Write (nout,*) 'The corresponding (machine dependent) gradient is'
        Write (nout,99997) g(1:n)
        Write (nout,99996) 'ISTATE contains', istate(1:n)

        nz = count(istate(1:n)>0)

        Write (nout,99995) 'and HESD contains', hesd(1:nz)
      End Select

99999 Format (1X,A,F9.4)
99998 Format (1X,A,4F9.4)
99997 Format (23X,1P,4E12.3)
99996 Format (1X,A,4I5)
99995 Format (1X,A,4E12.4)
    End Program e04lbfe
```

10.2 Program Data

None.

10.3 Program Results

```
E04LBF Example Program Results
** The conditions for a minimum have not all been satisfied,
** but a lower point could not be found.
** ABNORMAL EXIT from NAG Library routine E04LBF: IFAIL =     3
** NAG soft failure - control returned

Function value on exit is    2.4338
at the point   1.0000  -0.0852   0.4093   1.0000
The corresponding (machine dependent) gradient is
                         2.953E-01  -5.867E-10   1.173E-09   5.907E+00
ISTATE contains   -2    1    2   -2
and HESD contains  0.2098E+03  0.4738E+02
```