NAG Toolbox for MATLAB

e04lb

1 Purpose

e04lb 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 function is intended for functions which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

2 Syntax

```
[bl, bu, x, hesl, hesd, istate, f, g, iw, w, ifail] = e04lb(funct, h,
monit, ibound, bl, bu, x, lh, iw, w, 'n', n, 'iprint', iprint, 'maxcal',
maxcal, 'eta', eta, 'xtol', xtol, 'stepmx', stepmx, 'liw', liw, 'lw',
lw)
```

3 Description

e04lb is applicable to problems of the form:

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

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 user-supplied (sub)program **funct** to calculate the value of $F(x)$ and its first derivatives $\dfrac{\partial F}{\partial x_j}$ at any point x, and a user-supplied (sub)program **h** to calculate the second derivatives $\dfrac{\partial^2 F}{\partial x_i \partial x_j}$.

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, e04lb 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 e04lb 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. Program.* **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

5.1 Compulsory Input Parameters

1: **funct – string containing name of m-file**

funct must evaluate the function $F(x)$ and its first derivatives $\dfrac{\partial F}{\partial x_j}$ 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 e04lb to terminate immediately.)

Its specification is:

```
[iflag, fc, gc, iw, w] = funct(iflag, n, xc, iw, liw, w, lw)
```

Input Parameters

1: **iflag – int32 scalar**

Will have been set to 2.

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 e04lb. e04lb will then terminate immediately with **ifail** set to your setting of **iflag**.

2: **n – int32 scalar**

The number n of variables.

3: **xc(n) – double array**

The point x at which F and the $\dfrac{\partial F}{\partial x_j}$ are required.

4: **iw(liw) – int32 array**

5: **liw – int32 scalar**

6: **w(lw) – double array**

7: **lw – int32 scalar**

funct is called with the same parameters **iw**, **liw**, **w** and **lw** as for e04lb. They are present so that, when other library functions 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 $\max(8, 7 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2) + 1$ onwards of **w** for passing quantities to **funct** from the (sub)program which calls e04lb. However, because of the danger of mistakes in

partitioning, it is recommended that you should pass information to **funct** via 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 $\max(8, 7 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2)$ elements of **w**.

Output Parameters

1: **iflag – int32 scalar**

Will have been set to 2.

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 e04lb. e04lb will then terminate immediately with **ifail** set to your setting of **iflag**.

2: **fc – double scalar**

Unless **iflag** is reset, **funct** must set **fc** to the value of the objective function F at the current point x.

3: **gc**(**n**) **– double array**

Unless **iflag** is reset, **funct** must set **gc**(j) to the value of the first derivative $\dfrac{\partial F}{\partial x_j}$ at the point x, for $j = 1, 2, \ldots, n$.

4: **iw**(**liw**) **– int32 array**

5: **w**(**lw**) **– double array**

funct is called with the same parameters **iw**, **liw**, **w** and **lw** as for e04lb. They are present so that, when other library functions 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 $\max(8, 7 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2) + 1$ onwards of **w** for passing quantities to **funct** from the (sub)program which calls e04lb. However, because of the danger of mistakes in partitioning, it is recommended that you should pass information to **funct** via 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 $\max(8, 7 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2)$ elements of **w**.

Note: funct should be tested separately before being used in conjunction with e04lb.

2: **h – string containing name of m-file**

h must calculate the second derivatives of F at any point x. (As with user-supplied (sub)program **funct**, there is the option of causing e04lb to terminate immediately.)

Its specification is:

```
[iflag, fhesl, fhesd, iw, w] = h(iflag, n, xc, lh, fhesd, iw, liw,
w, lw)
```

Input Parameters

1: **iflag – int32 scalar**

Is set to a nonnegative number.

If **h** resets **iflag** to some negative number, e04lb will terminate immediately with **ifail** set to your setting of **iflag**.

2: **n – int32 scalar**

The number n of variables.

3: **xc(n) – double array**

The point x at which the second derivatives of F are required.

4: **lh – int32 scalar**

The length of the array **fhesl**.

5: **fhesd(n) – double array**

The value of $\dfrac{\partial F}{\partial x_j}$ at the point x, for $j = 1, 2, \ldots, n$.

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

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 $\mathbf{fhesd}(j) = \left.\dfrac{\partial^2 F}{\partial x_j^2}\right|_{\mathbf{xc}}$, $j = 1, 2, \ldots, n$.

6: **iw(liw) – int32 array**

7: **liw – int32 scalar**

8: **w(lw) – double array**

9: **lw – int32 scalar**

As in user-supplied (sub)program **funct**, these parameters correspond to the parameters **iw**, **liw**, **w**, **lw** of e04lb. **h must not change** the first two elements of **iw** or the first $\max(8, 7 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2)$ elements of **w**. Again, it is recommended that you should pass quantities to **h** via global variables and not use **iw** or **w** at all.

Output Parameters

1: **iflag – int32 scalar**

Is set to a nonnegative number.

If **h** resets **iflag** to some negative number, e04lb will terminate immediately with **ifail** set to your setting of **iflag**.

2: **fhesl(lh) – double array**

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 $\mathbf{fhesl}((i-1)(i-2)/2 + j) = \left.\dfrac{\partial^2 F}{\partial x_i \partial x_j}\right|_{\mathbf{xc}}$, for $i = 2, 3, \ldots, n$ and $j = 1, 2, \ldots, i-1$. (The upper triangle is not required because the matrix is symmetric.)

3: **fhesd(n) – double array**

The value of $\dfrac{\partial F}{\partial x_j}$ at the point x, for $j = 1, 2, \ldots, n$.

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

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 $\mathbf{fhesd}(j) = \left.\dfrac{\partial^2 F}{\partial x_j^2}\right|_{\mathbf{xc}}$, $j = 1, 2, \ldots, n$.

4: **iw(liw) – int32 array**

5: **w(lw) – double array**

As in user-supplied (sub)program **funct**, these parameters correspond to the parameters **iw**, **liw**, **w**, **lw** of e04lb. **h must not change** the first two elements of **iw** or the first $\max(8, 7 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2)$ elements of **w**. Again, it is recommended that you should pass quantities to **h** via global variables and not use **iw** or **w** at all.

Note: h should be tested separately before being used in conjunction with e04lb.

3: **monit – string containing name of m-file**

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.

Its specification is:

```
[iw, w] = monit(n, xc, fc, gc, istate, gpjnrm, cond, posdef, niter,
nf, iw, liw, w, lw)
```

Input Parameters

1: **n – int32 scalar**

The number n of variables.

2: **xc(n) – double array**

The co-ordinates of the current point x.

3: **fc – double scalar**

The value of $F(x)$ at the current point x.

4: **gc(n) – double array**

The value of $\dfrac{\partial F}{\partial x_j}$ at the current point x, for $j = 1, 2, \ldots, n$.

5: **istate(n) – int32 array**

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 – double scalar**

The Euclidean norm of the projected gradient vector g_z.

7: **cond – double scalar**

The ratio of the largest to the smallest elements of the diagonal factor D of the projected Hessian matrix (see specification of user-supplied (sub)program **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 scalar**

Is set **true** or **false** according to whether the second derivative matrix for the current subspace, H, is positive-definite or not.

9: **niter – int32 scalar**

The number of iterations (as outlined in Section 3) which have been performed by e04lb so far.

10: **nf – int32 scalar**

The number of times that user-supplied (sub)program **funct** has been called so far. Thus **nf** is the number of function and gradient evaluations made so far.

11: **iw**(**liw**) **– int32 array**
12: **liw – int32 scalar**
13: **w**(**lw**) **– double array**
14: **lw – int32 scalar**

As in user-supplied (sub)program **funct**, and user-supplied (sub)program **h**, these parameters correspond to the parameters **iw**, **liw**, **w**, **lw** of e04lb. They are included in **monit**'s parameter list primarily for when e04lb is called by other library functions.

Output Parameters

1: **iw**(**liw**) **– int32 array**
2: **w**(**lw**) **– double array**

As in user-supplied (sub)program **funct**, and user-supplied (sub)program **h**, these parameters correspond to the parameters **iw**, **liw**, **w**, **lw** of e04lb. They are included in **monit**'s parameter list primarily for when e04lb is called by other library functions.

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.

4: **ibound – int32 scalar**

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 functions. In e04lb it produces the same effect as **ibound** $= 1$.)

Constraint: $0 \leq$ **ibound** ≤ 4.

5: **bl**(**n**) – **double array**

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; e04lb 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 e04lb.

6: **bu**(**n**) – **double array**

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; e04lb 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 e04lb.

7: **x**(**n**) – **double array**

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

8: **lh** – **int32 scalar**

Constraint: **lh** $\geq \max(\mathbf{n} \times (\mathbf{n} - 1)/2, 1)$.

9: **iw**(**liw**) – **int32 array**

Constraint: **liw** ≥ 2.

10: **w**(**lw**) – **double array**

Constraint: **lw** $\geq \max(7 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2, 8)$.

5.2 Optional Input Parameters

1: **n – int32 scalar**

Default: The dimension of the arrays **bl**, **bu**, **x**, **hesd**, **istate**, **g**. (An error is raised if these dimensions are not equal.)

the number n of independent variables.

Constraint: $\mathbf{n} \geq 1$.

2: **iprint – int32 scalar**

The frequency with which user-supplied (sub)program **monit** is to be called.

iprint > 0

user-supplied (sub)program **monit** is called once every **iprint** iterations and just before exit from e04lb.

iprint $= 0$

user-supplied (sub)program **monit** is just called at the final point.

iprint < 0

user-supplied (sub)program **monit** is not called at all.

iprint should normally be set to a small positive number.

Suggested value: **iprint** $= 1$.

Default: 1

3: **maxcal – int32 scalar**

The maximum permitted number of evaluations of $F(x)$, i.e., the maximum permitted number of calls of user-supplied (sub)program **funct**.

Suggested value: **maxcal** $= 50 \times$ **n**.

Default: $50 \times$ **n**

Constraint: **maxcal** ≥ 1.

4: **eta – double scalar**

Every iteration of e04lb 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 e04lb, 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).

Default:

if **n** $= 1$, 0.0;
0.9 otherwise.

Constraint: $0.0 \leq$ **eta** < 1.0.

5: **xtol – double scalar**

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 $\|x_{\text{sol}} - x_{\text{true}}\| < \textbf{xtol} \times (1.0 + \|x_{\text{true}}\|)$, 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 8 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), e04lb will use $10.0 \times \sqrt{\epsilon}$ instead of **xtol**.

Suggested value: **xtol** $= 0.0$.

Default: 0.0

Constraint: **xtol** ≥ 0.0.

6: **stepmx – double scalar**

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

e04lb will ensure that, for each iteration,

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

where k is the iteration number. Thus, if the problem has more than one solution, e04lb 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** $= 100000.0$.

Default: 100000.0

Constraint: **stepmx** $\geq$ **xtol**.

7: **liw – int32 scalar**

Default: The dimension of the array **iw**.

Constraint: **liw** ≥ 2.

8: **lw – int32 scalar**

Default: The dimension of the array **w**.

Constraint: **lw** $\geq \max(7 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2, 8)$.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **bl**(**n**) **– double array**

The lower bounds actually used by e04lb, e.g., if **ibound** $= 2$, **bl**$(1) =$ **bl**$(2) = \cdots =$ **bl**(**n**) $= 0.0$.

2: **bu(n) – double array**

The upper bounds actually used by e04lb, e.g., if **ibound** $= 2$, $\mathbf{bu}(1) = \mathbf{bu}(2) = \cdots = \mathbf{bu}(\mathbf{n}) = 10^6$.

3: **x(n) – double array**

The final point $x^{(k)}$. Thus, if **ifail** $= 0$ on exit, $\mathbf{x}(j)$ is the jth component of the estimated position of the minimum.

4: **hesl(lh) – double array**

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 e04lb (see Section 7).

5: **hesd(n) – double array**

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 e04lb (see Section 7).

6: **istate(n) – int32 array**

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.

7: **f – double scalar**

The function value at the final point given in **x**.

8: **g(n) – double array**

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.

9: **iw(liw) – int32 array**

10: **w(lw) – double array**

11: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Note: e04lb may return useful information for one or more of the following detected errors or warnings.

ifail < 0

A negative value of **ifail** indicates an exit from e04lb because you have set **iflag** negative in the user-supplied (sub)programs **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 **eta** < 0.0,
or **eta** ≥ 1.0,
or **xtol** < 0.0,
or **stepmx** $<$ **xtol**,
or **ibound** < 0,
or **ibound** > 4,
or **bl**$(j) >$ **bu**(j) for some j if **ibound** $= 0$,
or **bl**$(1) >$ **bu**(1) if **ibound** $= 3$,
or **lh** $< \max(1, \mathbf{n} \times (\mathbf{n} - 1)/2)$,
or **liw** < 2,
or **lw** $< \max(8, 7 \times \mathbf{n} + \mathbf{n} \times (\mathbf{n} - 1)/2)$.

(Note that if you have set **xtol** to 0.0, e04lb 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 e04kd and e04lb.)

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** $= 3$.

The values **ifail** $= 2$, 3 or 5 may also be caused by mistakes in user-supplied (sub)programs **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 e04lb when $H^{(k)}$ is positive-definite and when (B1, B2 and B3) or B4 hold, where

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

(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 $k =$ **hesd**(b)/**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 $F(x_{\text{sol}})$ at a superlinear or fast linear rate,

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

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

then it is almost certain that x_{sol} is a close approximation to the position of a minimum. When (ii) is true, then usually $F(x_{\text{sol}})$ is a close approximation to $F(x_{\text{true}})$. The quantities needed for these checks are all available via user-supplied (sub)program **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 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 e04lb is $\frac{n_z^3}{6} + O\left(n_z^2\right)$. In addition, each iteration makes one call of user-supplied (sub)program **h** and at least one call of user-supplied (sub)program **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**.

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 e04lb 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) **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 user-supplied (sub)program **monit** and on exit from e04lb 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) user-supplied (sub)program **monit**'s parameter **gpjnrm** simply gives the norm of the first derivative vector.

So the following function (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)
C
C     A ROUTINE TO APPLY E04LBF TO UNCONSTRAINED PROBLEMS.
C
C     THE REAL ARRAY WORK MUST BE OF DIMENSION AT LEAST
C     (9*N + MAX(1, N*(N-1)/2)).  ITS FIRST 7*N + MAX(1, N*(N-1)/2)
C     ELEMENTS WILL BE USED BY E04LBF AS THE ARRAY W.  ITS LAST
C     2*N ELEMENTS WILL BE USED AS THE ARRAYS BL AND BU.
C
C     THE INTEGER ARRAY IWORK MUST BE OF DIMENSION AT LEAST (N+2)
C     ITS FIRST 2 ELEMENTS WILL BE USED BY E04LBF AS THE ARRAY IW.
C     ITS LAST N ELEMENTS WILL BE USED AS THE ARRAY ISTATE.
C
C     LIWORK AND LWORK MUST BE SET TO THE ACTUAL LENGTHS OF IWORK
C     AND WORK RESPECTIVELY, AS DECLARED IN THE CALLING SEGMENT.
C
C     OTHER PARAMETERS ARE AS FOR E04LBF.
C
C     .. Parameters ..
      INTEGER NOUT
      PARAMETER (NOUT=6)
C     .. Scalar Arguments ..
      real ETA, F, STEPMX, XTOL
      INTEGER IFAIL, IPRINT, LH, LIWORK, LWORK, MAXCAL, N
C     .. Array Arguments ..
      real G(N), HESD(N), HESL(LH), WORK(LWORK), X(N)
      INTEGER IWORK(LIWORK)
C     .. Subroutine Arguments ..
      EXTERNAL FUNCT, H, MONIT
C     .. Local Scalars ..
      INTEGER IBOUND, J, JBL, JBU, NH
      LOGICAL TOOBIG
C     .. External Subroutines ..
      EXTERNAL E04LBF
C     .. Executable Statements ..
C     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
C     JBL AND JBU SPECIFY THE PARTS OF WORK USED AS BL AND BU
      JBL = 7*N + NH + 1
      JBU = JBL + N
C     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)
C     CHECK THE PART OF IWORK WHICH WAS USED AS ISTATE IN CASE
C     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
C
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
```

9 Example

e04lb_funct.m

```
function [iflag, fc, gc] = funct(iflag, n, xc)
  gc = zeros(n, 1);

  fc = (xc(1)+10*xc(2))^2 + 5*(xc(3)-xc(4))^2 + (xc(2)-2*xc(3))^4 + ...
         10*(xc(1)-xc(4))^4;
  gc(1) = 2*(xc(1)+10*xc(2)) + 40*(xc(1)-xc(4))^3;
  gc(2) = 20*(xc(1)+10*xc(2)) + 4*(xc(2)-2*xc(3))^3;
  gc(3) = 10*(xc(3)-xc(4)) - 8*(xc(2)-2*xc(3))^3;
  gc(4) = 10*(xc(4)-xc(3)) - 40*(xc(1)-xc(4))^3;
```

e04lb_hess.m

```
function [iflag, fhesl, fhesd] = hess(iflag, n, xc, lh, fhesd)
  fhesl = zeros(lh, 1);

  fhesd(1) = 2.0d0 + 120.0d0*(xc(1)-xc(4))^2;
  fhesd(2) = 200.0d0 + 12.0d0*(xc(2)-2.0d0*xc(3))^2;
  fhesd(3) = 10.0d0 + 48.0d0*(xc(2)-2.0d0*xc(3))^2;
  fhesd(4) = 10.0d0 + 120.0d0*(xc(1)-xc(4))^2;
  fhesl(1) = 20.0d0;
  fhesl(2) = 0.0d0;
  fhesl(3) = -24.0d0*(xc(2)-2.0d0*xc(3))^2;
  fhesl(4) = -120.0d0*(xc(1)-xc(4))^2;
  fhesl(5) = 0.0d0;
  fhesl(6) = -10.0d0;
```

e04lb_monit.m

```
function [] = monit(n, xc, fc, gc, istate, gpjnrm, cond, posdef, niter,
nf)

  fprintf('\n Itn     Fn evals              Fn value            Norm of
proj gradient\n');
   fprintf(' %3d        %5d     %20.4f        %20.4f\n', niter, nf, fc,
gpjnrm);
  fprintf('\n J           X(J)                G(J)         Status\n');
```

```
  for j = 1:n
    isj = istate(j);
    if (isj > 0)
      fprintf('%2d %16.4f%20.4f   %s\n', j, xc(j), gc(j), '    Free');
    elseif (isj == -1)
       fprintf('%2d %16.4f%20.4f    %s\n', j, xc(j), gc(j), '    Upper
Bound');
    elseif (isj == -2)
       fprintf('%2d %16.4f%20.4f    %s\n', j, xc(j), gc(j), '    Lower
Bound');
    elseif (isj == -3)
           fprintf('%2d  %16.4f%20.4f     %s\n',  j,  xc(j),  gc(j),  '
Constant');
    end
  end
  if (cond ~= 0.0d0)
    if (cond > 1.0d6)
        fprintf('\nEstimated condition number of projected Hessian is
more than 1.0e+6\n');
    else
        fprintf('\nEstimated condition number of projected Hessian =
%10.2f\n', cond);
    end
    if ( not(posdef) )
%     The following statement is included so that this MONIT
%     can be used in conjunction with either of the routines
%     E04KDF or  E04LBF
      fprintf('\nProjected Hessian matrix is not positive definite\n');
    end
  end
```

```
ibound = int32(0);
bl = [1;
     -2;
     -1000000;
     1];
bu = [3;
     0;
     1000000;
     3];
x = [3;
     -1;
     0;
     1];
lh = int32(6);
iw = [int32(0);
     int32(0)];
w = zeros(34,1);
[blOut, buOut, xOut, hesl, hesd, istate, f, g, iwOut, wOut, ifail] = ...
     e04lb('e04lb_funct', 'e04lb_hess', 'e04lb_monit', ibound, bl, bu, x,
lh, iw, w)
```

```
Itn     Fn evals              Fn value            Norm of proj gradient
   0          1                215.0000                   144.0139

 J           X(J)                G(J)         Status
 1           3.0000            306.0000       Upper Bound
 2          -1.0000            -144.0000      Free
 3           0.0000              -2.0000      Free
 4           1.0000            -310.0000      Lower Bound

Estimated condition number of projected Hessian =       3.83

 Itn     Fn evals              Fn value            Norm of proj gradient
   1          2                163.0642                   320.3345

 J           X(J)                G(J)         Status
 1           3.0000            320.3345       Free
```

```
 2          -0.2833             -0.0351       Free
 3           0.3311              0.0703       Free
 4           1.0000           -313.3106       Lower Bound

Estimated condition number of projected Hessian =       9.46

 Itn     Fn evals              Fn value            Norm of proj gradient
   2          3                 34.3864                   94.9936

 J           X(J)                G(J)         Status
 1           2.3327             94.9936       Free
 2          -0.2172             -0.0026       Free
 3           0.3565              0.0052       Free
 4           1.0000            -88.2372       Lower Bound

Estimated condition number of projected Hessian =       4.35

 Itn     Fn evals              Fn value            Norm of proj gradient
   3          4                  8.8929                   28.2250

 J           X(J)                G(J)         Status
 1           1.8870             28.2250       Free
 2          -0.1731             -0.0009       Free
 3           0.3742              0.0017       Free
 4           1.0000            -21.6542       Lower Bound

Estimated condition number of projected Hessian =       4.23

 Itn     Fn evals              Fn value            Norm of proj gradient
   4          5                  3.8068                    8.4415

 J           X(J)                G(J)         Status
 1           1.5881              8.4415       Free
 2          -0.1435             -0.0004       Free
 3           0.3861              0.0008       Free
 4           1.0000             -1.9952       Lower Bound

Estimated condition number of projected Hessian =       4.62

 Itn     Fn evals              Fn value            Norm of proj gradient
   5          6                  2.7680                    2.5810

 J           X(J)                G(J)         Status
 1           1.3847              2.5810       Free
 2          -0.1233             -0.0002       Free
 3           0.3941              0.0004       Free
 4           1.0000              3.7808       Lower Bound

Estimated condition number of projected Hessian =       9.60

 Itn     Fn evals              Fn value            Norm of proj gradient
   6          7                  2.5381                    0.8503

 J           X(J)                G(J)         Status
 1           1.2396              0.8503       Free
 2          -0.1090             -0.0001       Free
 3           0.3998              0.0002       Free
 4           1.0000              5.4513       Lower Bound

Estimated condition number of projected Hessian =      18.55

 Itn     Fn evals              Fn value            Norm of proj gradient
   7          8                  2.4702                    0.3609

 J           X(J)                G(J)         Status
 1           1.1165              0.3609       Free
 2          -0.0968             -0.0001       Free
 3           0.4047              0.0001       Free
 4           1.0000              5.8896       Lower Bound
```

```
Estimated condition number of projected Hessian =      27.45

 Itn     Fn evals              Fn value            Norm of proj gradient
   8          9                 2.4338                    0.0002

 J           X(J)                G(J)         Status
 1           1.0000              0.2953       Lower Bound
 2          -0.0852             -0.0001       Free
 3           0.4093              0.0002       Free
 4           1.0000              5.9069       Lower Bound

Estimated condition number of projected Hessian =       4.43

 Itn     Fn evals              Fn value            Norm of proj gradient
   9         10                 2.4338                    0.0000

 J           X(J)                G(J)         Status
 1           1.0000              0.2953       Lower Bound
 2          -0.0852             -0.0000       Free
 3           0.4093              0.0000       Free
 4           1.0000              5.9070       Lower Bound

Estimated condition number of projected Hessian =       4.43

 Itn     Fn evals              Fn value            Norm of proj gradient
  10         11                 2.4338                    0.0000

 J           X(J)                G(J)         Status
 1           1.0000              0.2953       Lower Bound
 2          -0.0852             -0.0000       Free
 3           0.4093              0.0000       Free
 4           1.0000              5.9070       Lower Bound

Estimated condition number of projected Hessian =       4.43
Warning: e04lb returned a non-zero warning or error indicator (3)
blOut =
           1
          -2
    -1000000
           1
buOut =
           3
           0
     1000000
           3
xOut =
    1.0000
   -0.0852
    0.4093
    1.0000
hesl =
   -0.0935
         0
   -0.1978
         0
         0
         0
hesd =
  209.8031
   47.3802
   45.5183
         0
istate =
          -2
           1
           2
          -2
f =
    2.4338
g =
```

```
    0.2953
   -0.0000
    0.0000
    5.9070
iwOut =
          -1
           0
wOut =
    1.0000
   -0.0852
    0.4093
    1.0000
    0.2953
    0.0000
   -0.0000
    5.9070
    0.2953
   -0.0000
    0.0000
    5.9070
         0
    0.0000
   -0.0000
         0
   -0.0000
    0.0000
    0.0001
         0
   -0.0000
    0.0000
   -0.0086
         0
    2.0000
  209.8031
   49.2125
   10.0000
   20.0000
         0
  -19.6062
         0
         0
  -10.0000
ifail =
           3
```