NAG Library Routine Document

E04LYF

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

E04LYF is an easy-to-use modified-Newton algorithm for finding a minimum of a function, $F(x_1, x_2, \ldots, x_n)$ subject to fixed upper and lower bounds on the independent variables, $x_1, x_2, \ldots, x_n$ when first and second derivatives of F are available. It is intended for functions which are continuous and which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

2 Specification

```
SUBROUTINE E04LYF (N, IBOUND, FUNCT2, HESS2, BL, BU, X, F, G, IW, LIW, W,       &
                   LW, IUSER, RUSER, IFAIL)
INTEGER            N, IBOUND, IW(LIW), LIW, LW, IUSER(*), IFAIL
REAL (KIND=nag_wp) BL(N), BU(N), X(N), F, G(N), W(LW), RUSER(*)
EXTERNAL           FUNCT2, HESS2
```

3 Description

E04LYF is applicable to problems of the form:

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

when first and second derivatives of $F(x)$ are available.

Special provision is made for 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 $u_1 = u_2 = \cdots = u_n$. You must supply a subroutine to calculate the values of $F(x)$ and its first derivatives at any point x and a subroutine to calculate the second derivatives.

From a starting point you supplied there is generated, on the basis of estimates of the curvature of $F(x)$, a sequence of feasible points which is intended to converge to a local minimum of the constrained function.

4 References

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: IBOUND – INTEGER *Input*

On entry: indicates 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 you are supplying all the l_j and u_j individually.

IBOUND $= 1$
: If there are no bounds on any x_j.

IBOUND $= 2$
: If all the bounds are of the form $0 \leq x_j$.

IBOUND $= 3$
: If $l_1 = l_2 = \cdots = l_n$ and $u_1 = u_2 = \cdots = u_n$.

Constraint: $0 \leq$ IBOUND ≤ 3.

3: FUNCT2 – SUBROUTINE, supplied by the user. *External Procedure*

You must supply this routine to calculate the values of the function $F(x)$ and its first derivatives $\frac{\partial F}{\partial x_j}$ at any point x. It should be tested separately before being used in conjunction with E04LYF (see the E04 Chapter Introduction).

The specification of FUNCT2 is:

```
SUBROUTINE FUNCT2 (N, XC, FC, GC, IUSER, RUSER)

INTEGER             N, IUSER(*)
REAL (KIND=nag_wp) XC(N), FC, GC(N), RUSER(*)
```

1: N – INTEGER *Input*

On entry: the number n of variables.

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

On entry: the point x at which the function and its derivatives are required.

3: FC – REAL (KIND=nag_wp) *Output*

On exit: the value of the function F at the current point x.

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

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

5: IUSER$(*)$ – INTEGER array *User Workspace*
6: RUSER$(*)$ – REAL (KIND=nag_wp) array *User Workspace*

FUNCT2 is called with the parameters IUSER and RUSER as supplied to E04LYF. You are free to use the arrays IUSER and RUSER to supply information to FUNCT2 as an alternative to using COMMON global variables.

FUNCT2 must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which E04LYF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: HESS2 – SUBROUTINE, supplied by the user. *External Procedure*

You must supply this routine to evaluate the elements $H_{ij} = \frac{\partial^2 F}{\partial x_i \partial x_j}$ of the matrix of second derivatives of $F(x)$ at any point x. It should be tested separately before being used in conjunction with E04LYF (see the E04 Chapter Introduction).

The specification of HESS2 is:

```
SUBROUTINE HESS2 (N, XC, HESLC, LH, HESDC, IUSER, RUSER)

INTEGER            N, LH, IUSER(*)
REAL (KIND=nag_wp) XC(N), HESLC(LH), HESDC(N), RUSER(*)
```

1: N – INTEGER *Input*

On entry: the number n of variables.

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

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

3: HESLC(LH) – REAL (KIND=nag_wp) array *Output*

On exit: HESS2 must place the strict lower triangle of the second derivative matrix H in HESLC, stored by rows, i.e., set HESLC$((i-1)(i-2)/2+j) = \dfrac{\partial^2 F}{\partial x_i \partial x_j}$, for $i = 2, 3, \ldots, n$ and $j = 1, 2, \ldots, i-1$. (The upper triangle is not required because the matrix is symmetric.)

4: LH – INTEGER *Input*

On entry: the length of the array HESLC.

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

On exit: must contain the diagonal elements of the second derivative matrix, i.e., set HESDC$(j) = \dfrac{\partial^2 F}{\partial x_j^2}$, for $j = 1, 2, \ldots, n$.

6: IUSER($*$) – INTEGER array *User Workspace*
7: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

HESS2 is called with the parameters IUSER and RUSER as supplied to E04LYF. You are free to use the arrays IUSER and RUSER to supply information to HESS2 as an alternative to using COMMON global variables.

HESS2 must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which E04LYF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

On entry: the lower bounds l_j.

If IBOUND is set to 0, BL(j) must be set 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 -10^6.)

If IBOUND is set to 3, you must set BL(1) to l_1; E04LYF will then set the remaining elements of BL equal to BL(1).

On exit: the lower bounds actually used by E04LYF.

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

On entry: the upper bounds u_j.

If IBOUND is set to 0, BU(j) must be set to u_j, for $j = 1, 2, \ldots, n$. (If an upper bound is not specified for any x_j the corresponding BU(j) should be set to 10^6.)

If IBOUND is set to 3, you must set BU(1) to u_1; E04LYF will then set the remaining elements of BU equal to BU(1).

On exit: the upper bounds actually used by E04LYF.

7: 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 $j = 1, 2, \ldots, n$. The routine checks the gradient and the Hessian matrix at the starting point, and is more likely to detect any error in your programming if the initial X(j) are nonzero and mutually distinct.

On exit: the lowest point found during the calculations. Thus, if IFAIL $= 0$ on exit, X(j) is the jth component of the position of the minimum.

8: F – REAL (KIND=nag_wp) *Output*

On exit: the value of $F(x)$ corresponding to the final point stored in X.

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

On exit: the value of $\dfrac{\partial F}{\partial x_j}$ corresponding to the final point stored in X, for $j = 1, 2, \ldots, n$; the value of G(j) for variables not on a bound should normally be close to zero.

10: IW(LIW) – INTEGER array *Workspace*
11: LIW – INTEGER *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which E04LYF is called.

Constraint: LIW $\geq$ N $+ 2$.

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

On entry: the dimension of the array W as declared in the (sub)program from which E04LYF is called.

Constraint: LW $\geq$ max(N $\times$ (N $+ 7), 10)$.

14: IUSER($*$) – INTEGER array *User Workspace*
15: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

IUSER and RUSER are not used by E04LYF, but are passed directly to FUNCT2 and HESS2 and may be used to pass information to these routines as an alternative to using COMMON global variables.

16: 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: E04LYF may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, N < 1,
or IBOUND < 0,
or IBOUND > 3,
or IBOUND $= 0$ and BL$(j) >$ BU(j) for some j,
or IBOUND $= 3$ and BL$(1) >$ BU(1),
or LIW $<$ N $+ 2$,
or LW $< \max(10, \mathrm{N} \times (\mathrm{N} + 7))$.

IFAIL $= 2$

There have been $50 \times$ N function evaluations, yet the algorithm does not seem to be converging. The calculations can be restarted from the final point held in X. The error 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 and the algorithm has failed.

IFAIL $= 4$

Not used. (This value of the parameter is included so as to make the significance of IFAIL $= 5$ etc. consistent in the easy-to-use routines.)

IFAIL $= 5$
IFAIL $= 6$
IFAIL $= 7$
IFAIL $= 8$

There is some doubt about whether the point x found by E04LYF is a minimum. The degree of confidence in the result decreases as IFAIL increases. Thus, when IFAIL $= 5$ it is probable that the final x gives a good estimate of the position of a minimum, but when IFAIL $= 8$ it is very unlikely that the routine has found a minimum.

IFAIL $= 9$

In the search for a minimum, the modulus of one of the variables has become very large $\left(\sim 10^6\right)$. This indicates that there is a mistake in user-supplied subroutines FUNCT2 or HESS2, that your problem has no finite solution, or that the problem needs rescaling (see Section 8).

IFAIL $= 10$

It is very likely that you have made an error in forming the gradient.

IFAIL $= 11$

It is very likely that you have made an error in forming the second derivatives.

If you are dissatisfied with the result (e.g., because IFAIL $= 5$, 6, 7 or 8), 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

When a successful exit is made then, for a computer with a mantissa of t decimals, one would expect to get about $t/2 - 1$ decimals accuracy in x, and about $t - 1$ decimals accuracy in F, provided the problem is reasonably well scaled.

8 Further Comments

The number of iterations required depends on the number of variables, the behaviour of $F(x)$ and the distance of the starting point from the solution. The number of operations performed in an iteration of E04LYF is roughly proportional to $n^3 + O(n^2)$. In addition, each iteration makes one call of HESS2 and at least one call of FUNCT2. So, unless $F(x)$, the gradient vector and the matrix of second derivatives can be evaluated very quickly, the run time will be dominated by the time spent in user-supplied subroutines FUNCT2 and HESS2.

Ideally the problem should be scaled so that at the solution the value of $F(x)$ and the corresponding values of $x_1, x_2, \dots x_n$ are each in the range $(-1, +1)$, and so that at points a unit distance away from the solution, F is approximately a unit value greater than at the minimum. 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 E04LYF will take less computer time.

9 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

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

starting from the initial guess $(3, -1, 0, 1)$. (In practice, it is worth trying to make user-supplied subroutines FUNCT2 and HESS2 as efficient as possible. This has not been done in the example program for reasons of clarity.)

9.1 Program Text

```
!   E04LYF Example Program Text
!   Mark 24 Release. NAG Copyright 2012.
    Module e04lyfe_mod

!     E04LYF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                   :: n = 4, nout = 6
      Integer, Parameter                   :: liw = n + 2
      Integer, Parameter                   :: lw = n*(n+7)
    Contains
      Subroutine funct2(n,xc,fc,gc,iuser,ruser)
!       Routine to evaluate objective function and its 1st derivatives.

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: fc
        Integer, Intent (In)                 :: n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: gc(n)
        Real (Kind=nag_wp), Intent (Inout)   :: ruser(*)
        Real (Kind=nag_wp), Intent (In)      :: xc(n)
        Integer, Intent (Inout)             :: iuser(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: x1, x2, x3, x4
!       .. Executable Statements ..
        x1 = xc(1)
        x2 = xc(2)
        x3 = xc(3)
        x4 = xc(4)
        fc = (x1+10.0_nag_wp*x2)**2 + 5.0_nag_wp*(x3-x4)**2 + &
          (x2-2.0_nag_wp*x3)**4 + 10.0_nag_wp*(x1-x4)**4
        gc(1) = 2.0_nag_wp*(x1+10.0_nag_wp*x2) + 40.0_nag_wp*(x1-x4)**3
        gc(2) = 20.0_nag_wp*(x1+10.0_nag_wp*x2) + 4.0_nag_wp*(x2-2.0_nag_wp*x3 &
          )**3
        gc(3) = 10.0_nag_wp*(x3-x4) - 8.0_nag_wp*(x2-2.0_nag_wp*x3)**3
        gc(4) = -10.0_nag_wp*(x3-x4) - 40.0_nag_wp*(x1-x4)**3

        Return

      End Subroutine funct2
      Subroutine hess2(n,xc,heslc,lh,hesdc,iuser,ruser)
!       Routine to evaluate 2nd derivatives.

!       .. Scalar Arguments ..
        Integer, Intent (In)                :: lh, n
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)    :: hesdc(n), heslc(lh)
        Real (Kind=nag_wp), Intent (Inout)  :: ruser(*)
        Real (Kind=nag_wp), Intent (In)     :: xc(n)
        Integer, Intent (Inout)             :: iuser(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: x1, x2, x3, x4
!       .. Executable Statements ..
        x1 = xc(1)
        x2 = xc(2)
        x3 = xc(3)
        x4 = xc(4)
        hesdc(1) = 2.0_nag_wp + 120.0_nag_wp*(x1-x4)**2
        hesdc(2) = 200.0_nag_wp + 12.0_nag_wp*(x2-2.0_nag_wp*x3)**2
        hesdc(3) = 10.0_nag_wp + 48.0_nag_wp*(x2-2.0_nag_wp*x3)**2
        hesdc(4) = 10.0_nag_wp + 120.0_nag_wp*(x1-x4)**2
        heslc(1) = 20.0_nag_wp
        heslc(2) = 0.0_nag_wp
        heslc(3) = -24.0_nag_wp*(x2-2.0_nag_wp*x3)**2
        heslc(4) = -120.0_nag_wp*(x1-x4)**2
        heslc(5) = 0.0_nag_wp
        heslc(6) = -10.0_nag_wp

        Return

      End Subroutine hess2
    End Module e04lyfe_mod
    Program e04lyfe

!     E04LYF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: e04lyf, nag_wp
      Use e04lyfe_mod, Only: funct2, hess2, liw, lw, n, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                  :: f
      Integer                             :: ibound, ifail
!     .. Local Arrays ..
      Real (Kind=nag_wp)                  :: bl(n), bu(n), g(n), ruser(1),    &
                                             w(lw), x(n)
      Integer                             :: iuser(1), iw(liw)
!     .. Executable Statements ..
      Write (nout,*) 'E04LYF Example Program Results'
      Flush (nout)
      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 e04lyf(n,ibound,funct2,hess2,bl,bu,x,f,g,iw,liw,w,lw,iuser,ruser, &
        ifail)

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

99999 Format (1X,A,4F9.4)
99998 Format (13X,4E12.4)
    End Program e04lyfe
```

9.2 Program Data

None.

9.3 Program Results

```
 E04LYF Example Program Results
 ** It is probable that a local minimum has been found,
 ** but it cannot be guaranteed.
 ** ABNORMAL EXIT from NAG Library routine E04LYF: IFAIL =     5
 ** 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
              0.2953E+00 -0.5867E-09  0.1173E-08  0.5907E+01
```