NAG Fortran Library Routine Document

E04KYF

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

E04KYF is an easy-to-use quasi-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 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 E04KYF(N, IBOUND, FUNCT2, BL, BU, X, F, G, IW, LIW, W, LW,
1                  IUSER, USER, IFAIL)
 INTEGER           N, IBOUND, IW(LIW), LIW, LW, IUSER(*), IFAIL
 real              BL(N), BU(N), X(N), F, G(N), W(LW), USER(*)
 EXTERNAL          FUNCT2
```

3 Description

This routine is applicable to problems of the form:

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

when first derivatives 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 $l_1 = l_2 = \ldots = l_n$ and $u_1 = u_2 = \ldots = u_n$. The user must supply a subroutine to calculate the values of $F(x)$ and its first derivatives at any point x.

From a starting point supplied by the user 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. An attempt is made to verify that the final point is a minimum.

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

$$LDL^T p_z = -g_z$$

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

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

If a saddle point is suspected, a local search is carried out with a view to moving away from the saddle point. A local search is also performed when a point is found which is thought to be a constrained minimum.

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 the user will be 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 $\leq$ 3.

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

This routine must be supplied by the user to calculate the values of the function $F(x)$ and its first derivative $\frac{\partial F}{\partial x_j}$ at any point x. It should be tested separately before being used in conjunction with E04KYF (see the E04 Chapter Introduction).

Its specification is:

```
SUBROUTINE FUNCT2(N, XC, FC, GC, IUSER, USER)
INTEGER            N, IUSER(*)
real               XC(N), FC, GC(N), USER(*)
```

1: N – INTEGER *Input*

On entry: the number n of variables.

2: XC(N) – ***real*** array *Input*

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

3: FC – ***real*** *Output*

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

4: GC(N) – ***real*** 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: USER($*$) – ***real*** array *User Workspace*

FUNCT2 is called from E04KYF with the parameters IUSER and USER as supplied to E04KYF. The user is free to use the arrays IUSER and USER to supply information to FUNCT2 as an alternative to using COMMON.

FUNCT2 must be declared as EXTERNAL in the (sub)program from which E04KYF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: BL(N) – ***real*** array *Input/Output*

On entry: the lower bounds l_j.

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

If IBOUND is set to 3, the user must set BL(1) to l_1; E04KYF will then set the remaining elements of BL equal to BL(1).

On exit: the lower bounds actually used by E04KYF.

5: BU(N) – ***real*** array *Input/Output*

On entry: the upper bounds u_j.

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

If IBOUND is set to 3, the user must set BU(1) to u_1; E04KYF will then set the remaining elements of BU equal to BU(1).

On exit: the upper bounds actually used by E04KYF.

6: X(N) – ***real*** 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 at the starting point, and is more likely to detect any error in the user's programming if the initial X(j) are non-zero 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.

7: F – ***real*** *Output*

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

8: G(N) – ***real*** array *Output*

On exit: the value of $\frac{\partial F}{\partial x_j}$ 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.

9: IW(LIW) – INTEGER array *Output*

On exit: if IFAIL $= 0$, 3 or 5, the first N elements of IW contain information about which variables are currently on their bounds and which are free. Specifically, if x_i is

(a) fixed on its upper bound, IW(i) is -1;

(b) fixed on its lower bound, IW(i) is -2;

(c) effectively a constant (i.e., $l_j = u_j$), IW(i) is -3;

(d) free, IW(i) gives its position in the sequence of free variables.

In addition, IW$(\mathrm{N}+1)$ contains the number of free variables (i.e., n_z). The rest of the array is used as workspace.

10: LIW – INTEGER *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which E04KYF is called.

Constraint: LIW $\geq \mathrm{N}+2$.

11: W(LW) – ***real*** array *Output*

On exit: if IFAIL $= 0$, 3 or 5, W(i) contains the ith element of the projected gradient vector g_z, for $i = 1, 2, \ldots, \mathrm{N}$. In addition, W$(\mathrm{N}+1)$ contains an estimate of the condition number of the projected Hessian matrix (i.e., k). The rest of the array is used as workspace.

12: LW – INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which E04KYF is called.

Constraint: LW $\geq \max(10 \times \mathrm{N} + \mathrm{N} \times (\mathrm{N}-1)/2, 11)$.

13: IUSER(*) – INTEGER array *User Workspace*

Note: the dimension of the array IUSER must be at least 1.

IUSER is not used by E04KYF, but is passed directly to FUNCT2 and may be used to pass information to that routine.

14: USER(*) – ***real*** array *User Workspace*

Note: the dimension of the array USER must be at least 1.

USER is not used by E04KYF, but is passed directly to FUNCT2 and may be used to pass information to that routine.

15: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL $= 0$ unless the routine detects an error (see Section 6).

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.**

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).

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 $< \max(11, 10 \times \mathrm{N} + \mathrm{N} \times (\mathrm{N} - 1)/2)$.

IFAIL $= 2$

There have been $100 \times n$ 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 $= 4$

An overflow has occurred during the computation. This is an unlikely failure, but if it occurs the user should restart at the latest point given in X.

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

There is some doubt about whether the point x found by E04KYF 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 ($\sim 10^6$). This indicates that there is a mistake in FUNCT2, that the user's problem has no finite solution, or that the problem needs rescaling (see Section 8).

IFAIL $= 10$

It is very likely that the user has made an error in forming the gradient.

If the user is 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. If persistent trouble occurs it may be advisable to try E04KZF.

7 Accuracy

A successful exit (IFAIL $= 0$) is made from E04KYF when (B1, B2 and B3) or B4 hold, and the local search confirms a minimum, where

$$\mathrm{B1} \equiv \alpha^{(k)} \times \|p^{(k)}\| < (x_{tol} + \sqrt{\epsilon}) \times (1.0 + \|x^{(k)}\|)$$

$$B2 \equiv |F^{(k)} - F^{(k-1)}| < ({x_{tol}}^2 + \epsilon) \times (1.0 + |F^{(k)}|)$$

$$B3 \equiv \|g_z^{(k)}\| < (\epsilon^{1/3} + x_{tol}) \times (1.0 + |F^{(k)}|)$$

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

(Quantities with superscript k are the values at the kth iteration of the quantities mentioned in Section 3, $x_{tol} = 100\sqrt{\epsilon}$, ϵ is the ***machine precision*** and $\|.\|$ denotes the Euclidean norm. The vector g_z is returned in the array W.)

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 x_{tol}.

If IFAIL $= 3$ or 5, x_{sol} may still be a good estimate of x_{true}, but the following checks should be made. Let k denote an estimate of the condition number of the projected Hessian matrix at x_{sol}. (The value of k is returned in W(N + 1)). If

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

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

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

then it is almost certain that x_{sol} is a close approximation to the position of a minimum. When (ii) is true, then usually $F(x_{sol})$ is a close approximation to $F(x_{true})$.

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 E04KYF is roughly proportional to n^2. In addition, each iteration makes at least one call of FUNCT2. So, unless $F(x)$ and the gradient vector can be evaluated very quickly, the run time will be dominated by the time spent in FUNCT2.

Ideally the problem should be scaled so that at the solution the value of $F(x)$ and the corresponding values of $x_1, x_2, \ldots, x_n$ 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 the user will be able to follow these recommendations very closely, but it is worth trying (by guesswork), as sensible scaling will reduce the difficulty of the minimization problem, so that E04KYF 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

$$\begin{array}{r} 1 \leq x_1 \leq 3 \\ -2 \leq x_2 \leq 0 \\ 1 \leq x_4 \leq 3, \end{array}$$

starting from the initial guess (3, -1, 0, 1).

9.1 Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     E04KYF Example Program Text.
*     Mark 18 Release. NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          N, LIW, LW
      PARAMETER        (N=4,LIW=N+2,LW=10*N+N*(N-1)/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             F
      INTEGER          IBOUND, IFAIL, J
*     .. Local Arrays ..
      real             BL(N), BU(N), G(N), USER(1), W(LW), X(N)
      INTEGER          IUSER(1), IW(LIW)
*     .. External Subroutines ..
      EXTERNAL         E04KYF, FUNCT2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04KYF Example Program Results'
      X(1) = 3.0e0
      X(2) = -1.0e0
      X(3) = 0.0e0
      X(4) = 1.0e0
      IBOUND = 0
      BL(1) = 1.0e0
      BU(1) = 3.0e0
      BL(2) = -2.0e0
      BU(2) = 0.0e0
*
*     X(3) is unconstrained, so we set BL(3) to a large negative
*     number and BU(3) to a large positive number.
*
      BL(3) = -1.0e6
      BU(3) = 1.0e6
      BL(4) = 1.0e0
      BU(4) = 3.0e0
      IFAIL = 1
*
      CALL E04KYF(N,IBOUND,FUNCT2,BL,BU,X,F,G,IW,LIW,W,LW,IUSER,USER,
     +            IFAIL)
*
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Error exit type', IFAIL,
     +     ' - see routine document'
      END IF
      IF (IFAIL.NE.1) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Function value on exit is ', F
         WRITE (NOUT,99997) 'at the point', (X(J),J=1,N)
         WRITE (NOUT,*)
     +     'The corresponding (machine dependent) gradient is'
         WRITE (NOUT,99996) (G(J),J=1,N)
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,A,F9.4)
99997 FORMAT (1X,A,4F9.4)
99996 FORMAT (13X,4e12.4)
      END
*
      SUBROUTINE FUNCT2(N,XC,FC,GC,IUSER,USER)
*     Routine to evaluate objective function and its 1st derivatives.
*     .. Scalar Arguments ..
      real              FC
      INTEGER           N
```

```
*     .. Array Arguments ..
      real              GC(N), USER(*), XC(N)
      INTEGER           IUSER(*)
*     .. Local Scalars ..
      real              X1, X2, X3, X4
*     .. Executable Statements ..
      X1 = XC(1)
      X2 = XC(2)
      X3 = XC(3)
      X4 = XC(4)
      FC = (X1+10.0e0*X2)**2 + 5.0e0*(X3-X4)**2 + (X2-2.0e0*X3)**4 +
     +     10.0e0*(X1-X4)**4
      GC(1) = 2.0e0*(X1+10.0e0*X2) + 40.0e0*(X1-X4)**3
      GC(2) = 20.0e0*(X1+10.0e0*X2) + 4.0e0*(X2-2.0e0*X3)**3
      GC(3) = 10.0e0*(X3-X4) - 8.0e0*(X2-2.0e0*X3)**3
      GC(4) = -10.0e0*(X3-X4) - 40.0e0*(X1-X4)**3
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 E04KYF Example Program Results

 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  0.3022E-08 -0.1236E-07  0.5907E+01
```